# Chapter One
# Introduction

## 1.1 Background to the Study

Software defect prediction (SDP) plays a key role in the timely delivery of good quality software product[1]. Software defect prediction studies aim to predict defect-prone components before the testing stage of the software development process[2]. In this modern-day, technology is rapidly evolving. As software users proliferate, so does the amount of software being produced increase. Software technology is used by different organizations in their day-to-day operations to solve minor or major difficulties. According to "Evans Data Corporation," 26.4 million software engineers were employed in 2019, with 27.7 million predicted for next year (2023)[3]. Software defect prediction (SDP) is a procedure to develop a model that can be utilized by software practitioners and researchers in the initial phases of the software development life cycle (SDLC) for distinguishing defective modules or classes[4]. Most defect prediction methods consider a series of traditional manually designed static code metrics[5]. Identifying and fixing software defects in the production system is costly, which could be a trivial task if detected before shipping the product[6]. Software faults are an unavoidable component of the software development process. Using artificial intelligence (AI) based software defect prediction (SDP) techniques in the software development process helps isolate defective software modules, count the number of software defects, and identify risky code changes[7]. The majority of software organizations are having trouble delivering high-quality software. Some major software programs are shipped without the necessary features. SDP is the crux research topic in software engineering, owing to the increased demand for high-quality software, and the effects of defective software on life and

businesses among others. Even if the program has been tested, the possibility of a defect in the software cannot be ruled out. It can be encountered while using the software product or during the under-testing process. Detected software flaws can have an influence on program reliability, quality, and cost of maintenance. Software failure can cost the software company producer a huge amount of money and possibly our lives; it can be caused by insufficient or incorrect procedures, unreliable surroundings, personnel who aren't properly trained, or inadequate requirements[8].

Software defects in software engineering are a major concern to developers, researchers, and end-users. Defects emanate from the software developer's misconception of the end-users' requirements. The divergence that transpired between the projected characteristics of the software and actual software has caused colossal losses to businesses and society at large. Software defects are the mistakes of the programmer that twists the expected performance of the software to generate different or unexpected results. Software defect and bug are used interchangeably. However, Bugs are faults in a source code, causing the software to produce inappropriate or undesired outputs. When bugs are being found and fixed is termed debugging. Software defects can be classified as the error(s) discovered after the application goes into production. It is an aberration of the customer requirement. With the rapid increase of software evolution in our day, software security has been a major nightmare for developers, end-users as well as stakeholders. The vulnerability of the software solutions calls for imperative attention to the source of the problem (source code), the effect of bugs on the software (Software Defect), and the troubleshooting mechanism (accurate defect prediction). SDP has been the crux area of concern for researchers in the discipline of software development[9].

Defect prediction reduces the time and cost of development while customer satisfaction is increasing[10]. Therefore, defect classification practices are imperative to achieving software quality. SDP is identifying defective components in the source code. Classification of defective software datasets using the classifiers; SVM, ANN, KNN, C4.5, and NB was executed in this study. The predicted results will assist developers to locate and fix potential defects, thereby improving software stability and reliability.

With the continuous expansion of modern software, software reliability has become a key concern. The effectiveness of classifiers depends majorly on the features used for the classification phase. If features are not adequately selected, it can result in misclassification. Feature reduction is a significant phase in the MLA, this technique consists of FE and FS. Feature Selection (FS) is imperative, and it is the picking of the discriminant feature for model construction, thus reducing training times, classification time, and misclassification[11]. This study developed a software defect predictive model using Classifiers and HSA as feature selection and evaluation metrics.

After the software is discovered to be defective, the main or next line of action should be to give an accurate or precise solution as quickly as possible. Faults detection is a major activity in software defect management that involves discovering and categorizing defects according to their features. Defects are repaired according to their severity and the influence they have on the software product. The conventional classifications of flaws depending on their severity degree are as follows[12].

**Critical:** The flaw has an impact on critical functioning or data. There isn't a way around it. For instance, unsuccessful installation, and total failure of an attribute among others.

**Major:** The fault has a salient impact on major functionality or data. There is a workaround, but it is not apparent and complicated to implement.

**Minor:** The flaw only affects non-critical data or minor functionality. There is a simple workaround for this. For example, one module may lack a tiny functionality, but another module can simply perform the same purpose.

**Trivial**: The flaw is insignificant because it has no bearing on functioning or data. There isn't even a need for a workaround. It has no effect on efficiency or production.

Researchers have contributed saliently to the progress of solving software defect challenges, such as forecasting whether or not a software module contains a fault. However, after faults have been identified, it is critical to determine their severity level; severity levels aid in prioritizing the software module with the most defects[13].

## 1.2 Statement of the Problem

In the past, researchers have identified that using a large dataset for SDP may be affected by high dimensionality which in turn may lead to time-wasting and misclassification[14].

Software defect classification is a major challenge in the research world. In the past, researchers have introduced different methods for high dimensionality reduction such as FE and FS. Feature extraction was designed to obtain features using PCA, LDA, ICA, variance, etc. but all these algorithms are FE techniques that don't consider how relevant those features are. This necessitated more findings of a sophisticated and efficient technique that is, FS, that takes into consideration the relevancy of the feature during dimensionality reduction in order to reduce prolonged processing, classification time, and misclassification. Hence,

this study applied a natured inspired meta-heuristic algorithm known as a harmony search algorithm for FS to obtain the most pertinent features for software defect classification.

It is, therefore, necessary to introduce a robust, nature-inspired meta-heuristic optimization algorithm. This study applied HSA for Feature Selection to obtain the most significant features before the classification.

### 1.3 Aim and Objectives of the Study

This study aims to evaluate the performance of some selected learning algorithms for defect classification. The objectives are to:

i. design a model using a meta-heuristic optimization algorithm, 5 LAs, and 5 metrics

ii. pre-process the raw software datasets;

iii. obtain the most relevant features by using an optimization algorithm known as the harmony search algorithm;

iv. classify the relevant features using some selected learning algorithms such as C4.5, SVM, NB, ANN and KNN

v. evaluate the model using five evaluation metrics such as accuracy, recall, precision, classification time and F1-score

## 1.4 Research Questions (RQ)

RQ1: **What is the evaluation performance and classification of some selected MLAs on the software defect dataset?**

To answer this question, this study applied 5 LAs to classify software defect datasets using 5 evaluation metrics to determine the level of defect, and in chapter four, the results are discussed.

RQ2: **Which process will reduce SDP time and misclassification?**

Selecting the most relevant feature for DP will reduce prediction time and misclassification. To answer this question comprehensively, this study designed a model using a meta-heuristic algorithm to select the most relevant feature for prediction. Other tools applied in order to obtain the results include 5 MLA, 4 metrics and 1 optimization algorithm. This study combined these three tools into a single model and also present their performance evaluation in the results section of chapter four

RQ3: **What is the most effective MLA for classification?**

This study identified ANN as the most effective classifier. The work obtained some software defect datasets, selected the relevant feature from each of the datasets, classified the selected relevant feature using MLAs, and evaluated the defect level using 5 evaluation metrics. The design, feature selection process, as well as performance evaluation, are presented in chapter four

## 1.5 Significance of the Study

The software defect predictive model using harmony search and some selected MLA for the selection of the discriminant feature for defect classification is very important because virtually all the fields of study whether small or large scale, are dependent on reliable software. During the categorization process of software defect, there are several steps involved dimensionality reduction which involves feature selection. The selection of the discriminant features remains a significant factor. Hence, the introduction of the HSA is very paramount in this study to remove some level of redundant attributes and also obtain the most appropriate features for prediction. In this case, the reduced features succor real-time software that will be reliable and reduce the high level of misclassification which occurs in SDP.

## 1.6 Scope of the Study

The scope of this study focused on identifying the effective classifier(s) for software defect classification using the Feature Selection technique by applying an optimization algorithm known as HSA together with five MLA; C4.5 Decision Tree, Artificial Neural Network (ANN), Naïve Bayes (NB), K-Nearest Neighbour (KNN) and Support Vector Machine (SVM) to classify Eclipse dataset.

## 1.7 Limitation of the Study

In this study, one meta-heuristics algorithm was applied. In the future, at least two or more of these algorithms should be applied for the sake of performance evaluation comparison.

## 1.8 Operational Definition of Terms

**Classification:** This is a data analysis technique that extracts models describing important data classes.

**Feature Extraction:** this involves the picking of discriminant features without considering how relevant those features are.

**Feature Selection:** This is a pre-processing method that was used to extract relevant features.

**Machine Learning:** It is an artificial intelligence (AI) system that was designed to mimic the human way of solving problems. ML makes software applications more accurate at classifying the results without being deliberately designed to do so.

**Model:** A methodical explanation of an object or phenomenon that shares imperative characteristics with the object or phenomenon.

**Harmony Search Algorithm:** it is a meta-heuristic algorithm designed to solve the optimization problem.

**Software Defect:** it is a flaw in coding which makes the expected unrealistic. The inappropriate and unexpected output from software that falls short of the end-user requirement is considered a defect.

**Prediction:** A prediction can also be a forecast, classification, or declaration about a class, event or dataset. They are frequently based on the experience or knowledge, but not necessarily.

**Optimization Algorithm:** This is the technique of identifying the set of inputs to an objective function that results in a maximum or minimum function evaluation.

## Endnotes

[1] G. Somya. *"Effective software defect prediction using support vector machines (SVMs)."* **International Journal of System Assurance Engineering and Management** 13, no. 2, 2022, 681-696.

[2] J. Manzura, A. Akbulut, C. Catal, & A. Mishra. *"Machine learning-based software defect prediction for mobile applications: A systematic literature review."* Sensors, 22, no. 7 2022.

[3] Anonymous "*How many software developers are in the US and the world? [Updated]"* Available at https://www.daxx.com/blog/development-trends/number-software-developers-world/. 2021.

[4] N. Meetesh & P. Singh *"A survey of software defect prediction based on deep learning."* **Archives of Computational Methods in Engineering,** 2022, 1-26.

[5] Z. Chunying, P. He, C. Zeng & J. Ma. *"Software defect prediction with semantic and structural information of codes based on Graph Neural Networks."* **Information and Software Technology,** 2022, 152.

[6] P. Jalaj, S. Ahirrao, K. Kotecha, G. Selvachandran & A. Abraham. *"A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools."* **Engineering Applications of Artificial Intelligence**, 2022, 111.

[7] P. Mahesha, D. Gupta, D. Anand, N. Goyal, H. M. Aljahdali, A. O. Mansilla, S. Kadry & A. Kumar. *"Towards design and feasibility analysis of DePaaS: AI based global unified software defect prediction framework."* **Applied Sciences** 12, no. 1, 2022, 493.

8 K. C. Youm, J. Ahn, & E. Lee "*Improved bug localization based on code change histories and bug report***s.**" **Information and Software Technology**, 82, Available at https://doi.org/10.1016/j.infsof.2016.11.002/. 2017, 177–192.

[9] A. Abdullah & M. Z. Khan. *"Software defect prediction using supervised machine learning and ensemble techniques: a comparative study."* **Journal of Software Engineering and Applications** 12, no. 5, 2019, 85-100.

[10] C. Guoqing, Y. Luo & W. Ding. "*Recent advances in supervised dimension reduction: A survey."* **Machine learning and knowledge extraction** 1, no. 1, 2019, 341-358.

[11] A. H. Telgaonkar & S. Deshmukh, "*Dimensionality reduction and classification through PCA and LDA*." 122, 2015, 4–8.

[12] Z. Imran "*Predicting bug severity in open-source software systems using scalable machine learning techniques*." ISTQB. available at https://www.istqb.org/. 2019

[13] K. Goyal, "*Data preprocessing in machine learning: 7 easy steps to follow."* ***Up Grad blog*** *22 2020* available at https://www.upgrad.com/blog/data-preprocessing-in-machine-learning/

[14] B. A. Sanusi, S. O. Olabiyisi, A. O. Olowoye, & B. L. Olatunji, "*Software defect prediction system using machine learning-based algorithms*." **Journal of Advances in Computational Intelligence Theory**, 1(3), 2019

# Chapter Two
# Literature Review

## 2.1 Conceptual Review

This chapter focuses on a critical examination of some selected studies as related to this work, as any research work is dependent on previous research in his/her discipline. SDP has been critically examined and focused upon by a huge number of researchers. A large body of work exists on software fault prediction, which uses a variety of MLAs and software metrics to estimate fault count and proneness in software modules. Researchers used a variety of software metrics and MLA to examine and improve the SFP method[1].

A software defect is considered a divergence from the intended behavior of the software. In other words, if a website or app's behaviour differs from what customers would expect, such difference has been deemed a defect. The terms defect and bugs are frequently interchanged in the software testing community. There is, however, a technical distinction. A bug is a flaw in software that occurs as a result of a programming error or flaw. This isn't the case for all defects. The only similarity output is that they both require testing teams to identify and correct them. Software defects are flaws in the software development procedure that causes the software to fail, the failure leading to the inability to deliver the desired outcome. Software defects are the mistakes of the programmer that twists the expected performance of the software to generate different or unexpected results. Bugs are errors in the source code of the software developed. When locating such errors, it is termed "debugging." In this current age, as more problematic software emerges, the need to find solutions is becoming imperative. Now the software is being used almost universally and in every step of the life of human beings. The software cost such as defects and breakdown may reduce the software

relevance which may also create disappointment for the customer. For this reason, there is a need for more work to be done on software defect classification[2].

Software defect classification aims to evaluate the software quality levels and to also reveal problems such software may develop sooner or later. Commonly it is also called a fault (bug) among software experts. Because of the increasing problems and numerous constraints that software is built, it is, therefore, not easy to manage excellent software. Conversely, software development organizations are not ready to take many risks with delivering inferior quality software. It leads to disappointment among customers. The bugs are the main reason for loss of time & cost in software goods.

Gaining from the previous research, bugs should be projected before new software is developed. To achieve this purpose, one must first determine which programs are prone to errors and require standard optimization. There are consequences that product defects cause such as unnecessary time consumption, elevated cost, etc., all at the cost of users or developers. This section presents the reviewed works that are relevant to this study. These publications can be classified into two categories: code metrics and process metrics. Papers that used feature selection strategies to improve the software failure prediction process were also separated[3]. The following are the categories:

i. Defects prediction by code metrics.
ii. Some Selected Metaheuristic Algorithms
iii. Feature selection for SDF
iv. Software Defect Management (SDM)

## 2.2 Other Methods of Software Defect Classification

In spite of painstaking planning, adequate documentation and appropriate processing control mechanism during application software development, incidences of certain faults are unavoidably present itself. These software faults sometimes lead to degradation of the quality which might be the underlying cause of failure. In today's cutting edge competition it's necessary to make conscious efforts towards the identification of the appropriate classification of prediction of defect ever before it gets to the end-user. Some of the software defect classification are described below:

### 2.2.1 Defect prediction by code metrics

Because code metrics-based datasets are readily available, they were mostly employed for fault prediction studies. The most commonly used datasets are open source which are freely available to all the researchers across the globe.

The publications listed below experimented with code metrics and are connected to our study.

The application of multiple ML models to compare the values of Chidamber Kamerer metrics to the figure of defects in the Mozilla and Bugzilla datasets to establish the effectiveness of CK metrics for fault detection was implemented. The conclusion was that the CBO metric offered the best prediction results, while the LOC metric was equally good, the DIT metric produced inconsequential results, and the NOC metric should not be utilized to anticipate fault. The study investigated the association between complexity measures and the certain figure of faults. They experimented with datasets from Eclipse 2.0, 2.1, and 3.0 releases. They employed spearman linear regression as ML model to estimate the number of

defects, and spearman correlation to compare the relationship between the complexity measures and the number of defects. They concluded that complexity metrics may be used to anticipate errors and that when the code is more complex, the number of faults will be higher[4].

Comparison of five-count models for estimating the number of defects was carried out. The study employed alternative complexity and object-oriented metrics to create defect count models for two industrial software systems. Their findings show that the zero-inflated and hurdle negative binomial regression models performed better for fault estimation than other count models.

The observation on how LOC and software faults interact was also observed. The work analysed that SD assessed from a small number of large software components can be used to predict defects in general with reasonable accuracy. In three types of eclipse and NASA datasets, Spearman correlation ranking was utilized to calculate the link between LOC and faults. LOC (range) was classified as defective or non-defective using five MLAs. They concluded that lines of code are an important characteristic for predicting software defects and can be used to build reliable defect prediction models.

Comparing the effectiveness of code metrics to the number of software errors. The study applied the Metric Calculation Tool (CKJM) and used it to extract code lines and complexity metrics from eleven datasets in order to find a link between metrics and the number of defects. They investigated the connection between software metrics and faults' numbers using Pearson's correlation. They discovered that evaluating classes with a higher number of flaws saves money on assessment and testing[5].

A Fuzzy Inference System prediction model has been implemented using PROMISE datasets. When data isn't available, their model starts learning with expert knowledge and details from earlier rounds, then moves on to a data-dependent strategy when enough data is available for defect categorization. They built and tested their models, which were made up of code metrics, using PROMISE datasets. Their performance was measured by the AUC. They concluded that their proposed prediction method might be used to accurately forecast software flaws.

To forecast defects in software, require regression techniques. The study developed a two-step model that classifies findings as defected or non-defected, then predicts the amount of faults if the results are determined to be faulty. They also employed NASA datasets with code lines, complexity, and cohesion measures. They claimed that their method outperformed the other methods they utilized in their study.

The application of an ensemble technique based on three regression approaches to forecasting the frequency of software problems was painstakingly observed. To train and test their models, they used fifteen PROMISE datasets made up of code metrics. Ensemble approaches outperformed single learners at classifying the specific number of defects, according to their findings[6].

Likewise, ROC framework was used to develop a threshold for predicting defected and non-defected modules. To create and assess their model, they employed a dataset that includes code metrics. ROC analysis was also used to investigate imbalance and feature selection.

Ensemble approaches for determining the number of errors in software were analyzed by S. S. Rathore and S. Kumar, claiming that the use of ensemble methods in such a scenario had not been tested. To create and evaluate the ensemble, five different learners were used. To

report and compare findings, the study employed AAE and ARE as performance measures. For evaluation, they examined eleven PROMISE datasets based on code metrics. They concluded that, when compared to individual models, the ensemble method outperformed individual models in terms of defect number prediction. Also, investigations were carried out on the significance of inheritance-based metrics for software defect classification. The study used sixty-five publicly available datasets with CK and inheritance metrics to accomplish this. The testing models were split into two groups: one that included both inheritance and CK measures, and another that only included inheritance metrics. The study used ANN for evaluation and showed the outcome using five evaluation measures. The findings present that inheritance measures have a significant role in SDP.

The importance of coupling metrics for software defect categorization was evaluated applied using seven different coupling metrics over 87 different datasets. The work employed SVM to classify the datasets. The study's findings show that three metrics are incredibly important for defect categorization, and they ranked the coupling metrics in order of importance[7].

In the study of new classification model that was introduced for SDP. To classify software flaws, the study employed ANN and a RABCA. The study performed the tests using five NASA datasets containing several code metrics and other classification-related performance indicators such as accuracy and area under the curve. The study rounded off that the classification worked well and may be utilized to predict faults. In fault prediction analysis, it can be summarized that the CK metrics suite has been used as input in the building of prediction models by various seasoned researchers. In addition, a significant number of authors have incorporated statistical techniques in the development of their respective prediction models. It's worth noting that public databases are frequently employed in fault

prediction studies. The behaviour of seven metaheuristic search approaches, such as GA, PSO, CSA, BPSO, GSAA, ABC, and BFA, was investigated for automated test data generation in the study. The performance of each of the seven algorithms was explained in this project. The collected results show that CSA aids in the more efficient generation of appropriate test data. When CSA was compared to the other six approaches, even the highest amount of unique test data had greater fitness values. More also, the study compared the precision, recall, and accuracy of ten different machine learning algorithms at the system and component levels, including NB, KNN, SVM, maximum entropy (ME), RF, DT, bagging, boosting, Glmnet, and SLDA. The evaluation was carried out on thirteen Apache projects that were retrieved automatically using BRCS tools. The results show that the Boosting algorithm performed best in twelve projects, with accuracy ranging from 81 percent to 98 percent, followed by RF with accuracy ranging from 75 percent to 97 percent, and Glmnet and SLDA with the most accurate results among other LAs. Furthermore, predicting severe defects according to module yields better results than predicting severity at the system level since the component's frequent words are more particular than the system's level prediction yields a better outcome[8].

Furthermore, the presentation of a new classification approach in which the study used five attributes for each reported bug: count, component, operating system, number of comments, and priority, and derived two attributes, summary weight and entropy, from those attributes. To improve the classification process, the work used six different classifications: NB, KNN, RF, RNG, CNN, and MLR. The data was gathered from PITS, Mozilla, and Eclipse. They initialized the classifier, and the results revealed an improvement in F-measure performance when compared to previous research.

Further research work proposed a new technique to define the sternness of bug reports that combines similarity using KL-divergence and topic modeling using LDA. To validate their suggested technique, they used 20,000 bug reports collected from four open-source programs (Xamarin, Eclipse, Wireshark, and Mozilla). The results of implementing their technique revealed that their model outperforms other cutting-edge studies cited in their literature in terms of accuracy.

A DNN based automatic solution was used to predict the level of bug reports. For the classification of severe bugs report, this solution uses a deep learning model, natural language approaches, and emotion analysis on the given dataset. Furthermore, by removing the severity assignment phase from bug reporting, the approach automates the severity evaluation process and benefits users. This approach was tested using historical data from Eclipse and Mozilla open-source products, and the cross-product results demonstrate that it beats state-of-the-art approaches by improving the f-measure by 7.90%.

The analysis conducted on a framework for predicting fine-grained severity levels that use a Minority Over-sampling Technique "SMOTE" to balance the severity classes, as well as a feature selection scheme to reduce the data scale and select the most informative features for training a KNN classifier that uses a distance-weighted voting scheme to classify fresh severe bugs reported. With two bug repositories, Eclipse and Mozilla, the efficiency of the proposed method has been validated. Their method outperformed cutting-edge studies in classifying minority severe classes, according to the findings[9].

The study proposed a new automated classifier that extracts report features using bigram and TF-IDF, and then uses SVM and neural networks to classify the data. The study found that the accuracy level of the classes is above 80, making the approach effective and efficient.

The study developed a model to predict the severe bugs reported in a cross-project context based on multiple attributes such as priority, bug fix time, number of comments, number of bugs on which it is dependent, number of duplicates for it, number of members in the cc list, summary weight, and bug complexity. The researchers looked at 5,859 bug complaints from various open-source platforms. The results reveal that the proposed model can assist in the prediction of bug reports for which historical data is not available, with accuracy ranging from 37.34 to 91.63 percent, 94.99 to 100 percent, 44.88 to 97.86 percent, and 61.18 to 95.99 percent for various classifiers. In the suggested prediction ANN model has been used as a backpropagation learning approach that starts with an initial weight. The gradient descent approach is used to update it throughout the learning phase. After that, a resilient backpropagation technique is applied to build the data. The NNs are implemented using the JM1/software dataset, Python programming language, Numpy, and the Neuro Lab framework. In comparison to other analytical models, it was discovered that the ANN model performs better in terms of error prediction.

A more research work by Imran Z. developed a method for predicting the severity of each issue that combines FE and ML. This method relies on a keyword extraction text-mining algorithm to extract keywords, after which it extracts the important keyword. The dataset used in these classes included four different labels in every binary and multi-class, 90 percent refined data was used with MLA, and the model was then tested on 10% refined data, resulting in better performance and higher classification precision - up to 90 percent -, data collection from Eclipse, Mozilla, GNOME, and other systems[10].

A different evaluation was performed on four datasets of NASA's PITS by Jindal *et al.* utilizing three primary methods: decision tree, Multi-Nominal Multivariate Logistic

Regression (MMLR), and Multi-Layer Perception (MLP) Different top-k terms were fed into prediction models, and these terms were collected from training and testing sets using an Information Gain (IG) feature selection method. The decision tree performed the best of all previous techniques in evaluating the severity of an issue, according to the findings.

### 2.2.2 Some Selected Metaheuristic Algorithms

Metaheuristic algorithms are computational intelligence paradigms that are particularly useful for complex issue solving. Examples of Metaheuristic algorithms are Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Harmony Search (HS), etc.

A metaheuristic algorithm is a search process for finding a decent solution to a complex and difficult to solve the optimization problem. In this real warlord with limited resources, finding a near-optimal solution based on faulty knowledge is critical for instance, computational power and time. One of the most significant breakthroughs of the last two decades in operations research has been the advent of metaheuristics for handling such optimization problems.

There are issues that demand that better solutions be developed above existing standard ones. There are several researchers who have explained different metaheuristic algorithms that are applicable to a wide range of applications to handle non-linear non-convex optimization problems.

Specific NP-hard issues are impossible to solve through combinatorial optimization (i.e., in reasonable run time). Metaheuristics, in contrast to optimization algorithms, iterative approaches, and basic greedy heuristics, can often find good answers with minimal computational work. There are a variety of problems that are impractical to solve using a

global optimality optimization algorithm. When the objective contains stochastic random variables or restrictions, for example, an optimization problem becomes more difficult. As a result, solving large-scale stochastic systems with stochastic programming or resilient optimization techniques is difficult[11].

#### 2.2.2.1 **Deep Learning Algorithm**

Deep learning is an area of ML that deals with ANNs, which are algorithms inspired by the structure and function of the brain. DL is MLAs that allow computers to learn by example in the same way that humans do. It is a critical component of self-driving automobiles, allowing them to detect a stop sign or discriminate between a pedestrian and a lamppost. Voice control is possible in consumer devices including phones, tablets, televisions, and hands-free speakers. It has received a lot of attention recently, and rightfully so because it is all about achieving previously unreachable goals.

In DL, it is required to acquire the knowledge of a computer-related model to perform categorization tasks directly from graphics, text files, or sound. DL models can achieve state-of-the-art correctly, and sometimes even do better than humans. To train models with several layers, a huge number of labeled datasets and NNTs are used.

DLMs are sometimes referred to as deep neural networks because most deep learning approaches use neural network designs. The ambiguous number of layers in a NN is commonly referred to as "deep." Deep neural networks can have up to 150 hidden layers, whereas traditional neural networks only have 2-3. Deep learning models are trained with large quantities of supervised datasets and ANN topologies that learn features directly from the data without the need for manual FE[12].

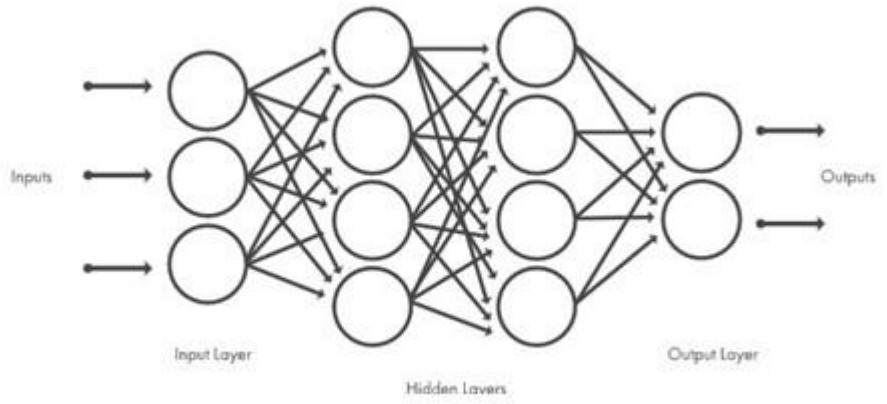



Figure 2.1. Filter Method of Software Defect Management[12]

A hybrid model for SDP that incorporates ANN and Simplified Swarm Optimization (SSO) was introduced. The ANN was used to classify software modules into defective and defected features. SSO is used to reduce the dataset's dimensionality. A total of four datasets from NASA's repository were used. When compared to other prediction approaches, our hybrid model outperformed them all and has been proven to be more operational when it comes to discovering correlations between defects and software metrics.

Moreover, another significant method for high dimensionality reduction was introduced, in which the study employed an ANN model with Sensitivity Analysis (SA-ANN) to determine the optimum variables for obtaining output. PCA-ANN (Principal Component Analysis) was also used. The logarithmic function is used to scale the data, and the ANN model is used to train and predict. Four publicly available datasets were used to test these methods (CM1, PC1, KC1 and KC2). The PCA-ANN approach has higher accuracy than the SA-ANN approach, according to the results. Deep learning can be used as a pre-training method for SDF.

The study used a pre-training technique for a shallow ANN and applied Logistic Regression as the final classifier. For this, a Denoising Autoencoder (DAE) is used. The weights and biases of a trained DAE are used to start the training procedure. The experiment is run on seven NASA datasets that are open to the public. They compare and contrast the proposed model (Pre-ANN) with SVM, ANN, PCA-SVM, Kernel PCA-SVM, and AE-SVM. MATLAB was used to create the model. To ensure the robustness of the experiments, 5-fold cross-validation was utilized. Pre-training boosts accuracy, according to the findings. In four of the seven datasets, it has higher accuracy. The results of earlier research show that deep learning techniques can be used in the field of program analysis. The use of deep learning in other software engineering studies seems promising[12].

#### 2.2.2.2 Particle Swarm Optimization (PSO)

PSO algorithm is based on the behavior of the flock of birds. Each particle calculates the objective function's value in a solution space position. Then, for each particle, it decides the direction to move by integrating the information from its present location and the best location it previously had, as well as the information from one or more of the best particles in the group. One step is completed once all particles have been moved. These stages are repeated multiple times until the desired result is achieved, which is when the stop requirement is satisfied. There are two operators in this algorithm: speed updating and position updating.

Iteratively enhancing a candidate solution around a given measure of quality is a computational optimization strategy that is used for the optimization issue. Animal herds, birds flocking, and fish schooling are all examples of social behavior that inspired this

algorithm. Normally, the congregation of birds without a leader will find food by moving around at random. The study only trail one of the group members who have the closest proximity to a food source (possible solution). Figure 2.2 depicts the entire PSO algorithm.

The study of focused on how the attributes are used for predicting. The study proved that this type of prediction is useful. They got 71% of mean probability detection and 25% mean false alarms rates. The influence of PSO on software failure prediction was examined. The feature was selected using PSO and paired them with bagging to accomplish their evaluations. In their bagging ensemble, they used eleven classifiers and built models utilizing nine NASA datasets with code metrics. The research work concluded that their categorization models surpassed the competition by a wide margin[13].

The cluster were developed using k-means and Neural-Gas techniques. The neural-Gas method works better in the terms of mean squared error and the k-means method works much faster compared to other methods. Comparing various algorithms for clustering the ISBSG and PROMISE datasets using WEKA. There are also some traditional statically methods to predict the defect of software.

Algorithm 2.1 Particles Swamp Optimization[13]

Step 1: Start
Step 2: Initilisation of algorithmic parameter and swarm position
Step 3: Calculate fitness of each particle
Step 4: Update pbest and gbest
Step 5: Update particle velocity and position
Step 6: Termination criteria is met, if not Goto Step 3
Step 7: End

A summary of the significant activities that have fueled and directed particle swarm research, as well as some key new applications and directions. The studies of GSC and publications from 1995 to 2006 on IEEE Xplore was presented in this work, enlightening the sense meant by Kennedy and Eberhart. The potency of this research work was to present complete and open challenges in the PSO algorithm. However, the compatibility of PSO submissions with each applied technique was not considered in this study.

The research work separately presented two parts in which a general and urgent overview and review of the discipline alongside the privileges and challenges emerging from the all-around application and usefulness of PSO. The study focused on the history and context of PSO, as well as its significance within the wider perspective of spontaneous computation. The review then moved on to other advances in PSO's native formulation, both discrete and continuous. The review then continued to discuss diverse developments to the native formulation of PSO both in discrete and continuous problems, swarm behaviour analysis, and measures considered to address stagnation. In addition, the review covered modifications or adaptations of the corresponding operation, algorithm configuration, and dynamic scenarios. The achievement of this study was identifying two significant areas of challenge for future further development: swarm stagnation and dynamic environments. The deficiency of this part is the ambiguous or inadequate elucidation of the related work. Part II 26 has, on the other hand, examined recent research on a number of fascinating topics, including limited and multi-objective optimization, combinatorial issues, and hybridization. Several topics were briefly discussed in the study, including NFNO, ANNO, CB, GIP, PGO, NR and FF. However, the study's fundamental flaw is that it does not analyse the selected

research in terms of evaluation metrics like convergence rate, diversity, accuracy, and processing time as quality elements in this domain[14].

Furthermore, a study detailed analysis on native PSO changes and their practical use in real-world applications. PSO has been rapidly modified in a variety of ways, including the two-step PSO and the PSO-SVM. The PSO's integration and practical application with the industry-standard algorithm has likewise yielded impressive results. This survey had the advantage of presenting recent diverse changes in PSO and analyzing the accuracy of PSO in many domains. The lack of statistical information on the discussed standard PSO and its use in various specified contexts is the study's principal flaw. More also, the study presented PSO variations in terms of swarm initialization, mutation operators, and inertia weight. The main benefit of this overview was to emphasize the need of introducing the various methods and inertia weight factors in order to improve PSO performance. Other promising PSO variations, on the other hand, were not examined. The study looked at the history of clustering algorithms based on PSO and provided the findings of fast-expanding trends in the literature on SI, the PSO paradigm, and PSO-based data clustering methodologies, demonstrating that such approaches are becoming increasingly popular. This study confirmed that the methods are unique and easy to use, and that they promote communication and collaboration. This research detailed the several PSO application fields that are significant to classification. However, there are no applicability to more complex issues. A comprehensive description of PSO shows its ability to proffer a panacea to diverse optimization issues in chemometrics. This paper emphasizes the crux of selecting appropriate PSO meta-parameters by presenting metaheuristic examples in the domains of

variable selection, estimating robust PCA solutions, and signal warping. This work greatly aided in the presentation of chemometrics-related works. However, it lacked the splash of other cutting-edge fields[15].

A comprehensive analysis was carried out on PSO. The study made advances in PSO theory, hybridization (with GA, DE, ABC, ACO, biogeography-based optimization, HSA, TS, AIS, and SA), and simulation (with GA, DE, ABC, ACO, modifications (including fuzzy PSO, chaotic PSO, bare-bones PSO, quantum-behaved PSO), population topologies (including star, ring, random, von Neumann, fully connected, etc.), extensions (to binary, discrete, constrained, and multi-objective optimization), and parallel implementation (in cloud computing, multiprocessor, multicore, and GPU forms). Moreover, the study did a survey on PSO's applications in the following disciplines: biology, chemistry, medicine, electrical and engineering, petroleum and power, mechanical engineering, research performance, messaging theory, and technological regulating systems. Regardless, the research work includes the annual exponential volatility of publications for each variant and application space[16].

The study gave an overview of the PSO's origins and history, as well as a theoretical study of the method. The researchers next examined the existing state of its use and conducted research in algorithm structure, topological structure, parameter selection, multi-objective optimization, discrete and parallel PSO, and engineering applications. This summary is distinguished by its recommendations for distinct future research directions. The study, however, does not include any analytic discussion. For software defect classification, the work integrated attribute ranking with ensemble learning. Using their ensemble, they

compared the performance of six filter-based ranking approaches. They evaluated three NASA datasets and utilized the area under the curve as their performance metric. The study rounded off that ensemble system outperformed individual methods for fault prediction and that different ranking algorithms have varying effects on the prediction process[17].

#### 2.2.2.3 Ant Colony Optimization (ACO)

ACO's genetic is inspired by nature that investigates the behavioural attribute of real ants. One of the population-based metaheuristic algorithms is ACO. Researchers have discovered that ants are social creatures who live in colonies and that their behaviour is more concerned with the colony's existence than with the survival of individual components. Ants' techniques to seeking food, and in particular how to locate the quickest route between sources of food and nests, is among the most captivating, mysterious and significant behaviours. The use of a substance termed pheromone is used by ants to communicate with one another and with the environment. While walking, the ants create a pheromone trail behind them. Although this substance evaporates quickly, it stays on the earth's surface in the form of an ant's track in the near term. When ants have to choose between two roads, they usually go for the one with the most pheromones.

The ACO algorithm is a probabilistic approach to addressing problems that can be simplified to finding good paths via graphs. ACO is a system based on the behavior of natural ants in their colonies, such as cooperation and adaption mechanisms. This method has stood the test of time for a variety of difficult combinatorial optimization problems in a variety of disciplines to find solutions in a reasonable amount of time.

ACO technique were applied to predict software flaws using many datasets. The findings suggest that the recommended strategy produces promising results. Jureczko datasets from the PROMISE repository were used in the study. The study chose nine independent qualities that are present in all of the datasets, as well as one class label attribute. The class label attribute has distinct values in different datasets, such as Y/N, 1/0, and T/F. For all datasets, these values are transformed to 1/0. Each dataset is divided into two parts: a training set with 70% of the records and a test set with 30% of the records. The model is built using the training set, and the test set to predict the class label. SDP's performance is compared to those of existing MLA such as LR (Logistic Regression), K-NN, and SVM. Moreover, suggestion was made on a Hybrid Model Reconstruction Approach (HYDRA) for cross-project DP that consists of two phases: GA and ensemble learning (EL). These two processes combine to form a huge classifier composition[18].

To study the benefits of HYDRA, they examined 29 datasets from the promise repository, totaling 11,196 cases (i.e., java modules) categorized as faulty or flawless. The study compared the technique to the most recent cross-project DP approaches. According to the study's findings, HYDRA achieves an average F1-score of 0.544. In general, these results compare to a change in F1-scores of 26.22 percent, 34.99 percent, 47.43 percent, 28.61 percent, and 30.14 percent over TCA+, Peters Filter, GP, MO, AND CODEP over the 29 datasets.

The ACO algorithm was instigated by the social behavior of natural ants, who have no vision but can locate food sources by depositing a chemical substance called pheromone, which is utilized as a kind of indirect communication to mark the way to food sources. The

distance, quantity, and quality of the food source all influence the amount of pheromone deposited. The likelihood of an ant choosing a route rises as they proliferate who have previously chosen the same path rises. The algorithm of ACO is shown in 2.2[19].

Algorithm 2.2 Ant Colony Optimization[19]

Step 1: Start
Step 2: Initialise parameters
Step 3: Generate global random
Step 4: Calculate fitness
Step 5: Update pheromone
Step 6: Apply transition
Step 7: New path
Step 8: Iteration = N if not Goto Step 3
Step 7: End



#### 2.2.2.4 Artificial Bee Colony Optimization (ABCO)

Many agents work together in the Bee Colony Optimization (BCO) approach to solve the problem in the optimization process. BCO differs from a real bee colony in a few ways. Before BCO, there were two algorithms: an ecological algorithm, which included a bee system algorithm based on the collective intelligence of bees' behavior, and a genetic. Because it finds the best answer, BCO is also known as the population algorithm. It's a metaheuristic that's inspired by bees' foraging behavior. BCO isn't extensively employed to tackle real-world challenges.

The algorithm of bees is built on the feeding habits of bees. the bees as shown in 2.3.

Algorithm 2.3 Artificial Bee Colony Optimization[20]

Step 1: Start

Step 2: Generate food source position

Step 3: Calculate the fitness value for each position

Step 4: Modify neighbor positions

Step 5: Calculation fitness of updates positions

Step 6: Compare food positions and retain best

Step 7: Calculate probability for positions solutions

Step 8: Define the lowest probability for position

Step 9: Update position solutions

Step 10: Stopping criteria met if not Goto Step 4

Step 11: Stop and retain best solution

Step 12: End

Ant Colony is one of the bioinspired computing approaches. The principal conception that foster this method is that the self-organizing rules that allow actual ants to behave in highly coordinated ways can be used to govern populations of artificial agents working together to solve computing problems. Different types of ant algorithms have been inspired by various characteristics of ant colony behaviour. Foraging, labour allocation, issue sorting, and cooperative transportation are among examples. Ants use staggery, a type of implicit contact mediated by changes in the environment, to coordinate their operations. For example, a foraging ant may leave a chemical on the ground, increasing the likelihood that other ants will follow in its footsteps. Many colony-level behaviour observed in social insects can be characterized by relatively simple models that just include stigmergic communication, according to biologists. In other words, biologists have demonstrated that stigmergic,

indirect communication is often adequate to explain how social insects achieve self-organization. Ant algorithms are fundamentally founded on the concept of using a type of artificial stigmergy to coordinate societies of artificial agents[20].

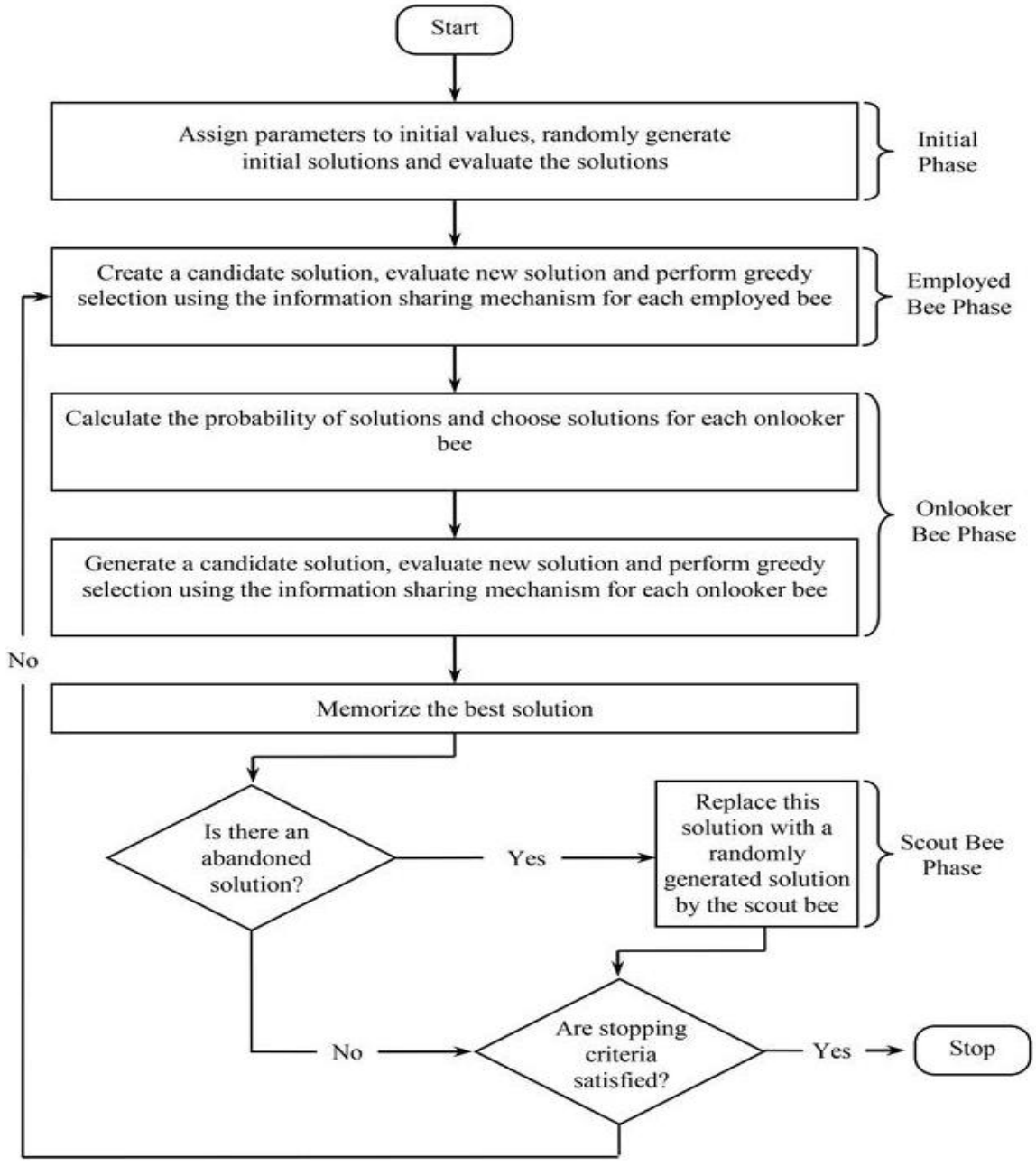



Figure 2.2: Ant Bee Colony Optimization[20]

In the ABCP model, a food source's position is specified as a possible solution, and the nectar of the food source is well-defined by the solution's quality. In the ABCP algorithm, there are three different sorts of bees: employed bee, observer bee, and scout bee. Employed bees are in charge of transporting nectar to the hive from previously found sources, and they exchange information about the quality of the source with observer bees. One employed bee visits each food source, bringing nectar back to the hive. Onlooker bees keep an eye on the employed bees in hives and use the information supplied by the employed bees to find a new source. After the employed and observer bees have completed their searches, the source is checked to see if the nectar reserves have been depleted. When a source is abandoned, the employed bee who was using it becomes the scout bee, searching for new sources at random. The flow chart of the ABCP algorithm in Fig. 3 shows the essential steps of the algorithm.

The most often used ML techniques are feature selection and classification. The crux and goal of FS is to uncover valuable properties carrying class information in datasets by removing noisy and redundant features and making classifiers easier to use. Dataset is disseminated across the many classes based on the resulting feature set using classification. An artificial bee colony (ABCO) is devised and used to pick features for classification tasks. The best models are created using a sensitivity fitness function that is defined based on the quantity of classes in the datasets, and the study applied are compared to models created via genetic programming (GP). When compared to GP in terms of critical FS and classification accuracy on well-known benchmark issues, the results of the trials reveal that the suggested technique is more accurate and efficient[21].

The WebKB dataset was analyzed by using minimum variance measure, information gain, and methodologies were used to choose features. Ward's minimum variance measure is initially used to a web page to discover clusters of redundant features. The finest representative features are kept in each cluster, while the others are removed. By removing such unnecessary features, the amount of time that should be expended on classification can be reduced. Following the clustering procedure, features are chosen from these clusters, and classification is performed using KNN, SVM, naive Bayes, and C4.5 classifiers with 10-fold cross validation. Student web pages are utilized as positive examples, whereas course web pages are used as negative examples. The proposed FS method is compared to other widely used feature selection methods. Experiments demonstrate that the suggested method outperforms most other feature selection strategies in terms of lowering the quantity of features and training time for the classifier. The accuracy of the KNN and SVM classifiers is 95.00 percent and 95.65 percent, respectively[22].

The problem can be solved by studying the behaviour of ants. The probability theory is used to determine which paths to take. It is a heuristic strategy in which ants produce pheromone and place it on pathways for future recollection. The level of pheromone strength and heuristic information are used to determine path visibility. Based on the highest pheromone level and heuristic information, the most viable path is chosen. For identifying appropriate state graph transitions, the method requires four parameters: probability, heuristic information, pheromone strength, and path visibility. ACO is a probabilistic method for finding the best path through a graph. Ants search for the shortest path between their colony and the food source. When the ants walk at random, pheromone deposits reveal the trail. A trail with more pheromone deposits has a higher chance of being followed. The path is

chosen based on the largest pheromone discharge from the initial node, and it is then optimized.

For software defect prediction, Farshidpour and Keynia propose utilizing ABC to train a multilayer perceptron (MLP) neural network. MLP with back propagation is compared to the method that was applied. The authors conclude that the NN can be properly trained when the right parameters are set to ABC. MLP trained using ABC (MLP-ABC) and MLP trained using back propagation are compared (MLP-BP). The NASA data sets CM1, KC1, and KC2 are used in the experiments. MLP-ABC outperforms MLP BP on average by 1.4 percent and 1.8 percent in terms of testing accuracy and precision, respectively.

The study proposed that ANNs prediction accuracy be improved (ANN). They propose using swarm intelligence techniques such as PSO, ACO, ABC, and firefly to train the ANN. The goal of optimal ANN is to find parameters, such as the number of input neurons, hidden layers and hidden neurons, output neurons' numbers, weights, and so on. On the NASA data sets Arc, Camel (versions 1.0, 1.2, 1.4, and 1.6), Interface, and Tomcat, the authors compare ANN-PSO, ANN-ACO, ANN-ABC, and ANN-Firefly. The best strategy, according to the authors, is ANN-PSO, which achieves the greatest outcomes in 7 of the 8 data sets. ANN-ABC comes in second place, with the best performance in three of the eight data sets. Using Chidamber and Kemerer metrics, Di Martino, Ferrucci, Gravino, and Sarro presented a hybrid of GA and SVM for inter-release fault estimation (1994). The GA's goal is to determine an appropriate SVM parameter configuration. Logistic Regression, C4.5, NB, Multi-Layer Perceptrons, K-Nearest Neighbor (K-NN), and Random Forest are six well-known machine learning approaches that are contrasted. Cross validation is done ten times.

Accuracy, precision, recall, and F-measure are the comparison metrics. The hybrid is shown to be quite effective for inter-release defect estimation when tested on the jEdit PROMISE dataset from (Promise).

For their suggested GP, Khoshgoftaar and Liu also use multi-objective fitness functions. However, in their case, the first goal is to condense the quantity of fault-prone modules using their newly developed "Modified Expected Cost of Misclassification" (MECM) measure, while the second goal is to reduce the number of fault-prone modules. The issue at hand is estimating fault-proneness utilizing four product metrics related to line count and one process measure related to "number of times the source file was inspected prior to system test." On testing data sets, the results demonstrate that the newly proposed approach can provide adequate performance without much decrease in accuracy (between 1% and 18%), Type-I misclassification rate (between 1% and 3%), and Type-II misclassification rate (between 1% and 4%)[22].

#### 2.2.2.5 Firefly Optimization Algorithm (FOA)

The discontinuous behaviour of fireflies and the notion of bioluminescent communication inspired this metaheuristic algorithm. The FA is a dependable and an effective practicable algorithm that can solve a wide range of real-world issues, including scheduling, optimization in dynamic contexts, and economic load dispatch52. The irregular conduct of fireflies to attract one another influences this algorithm. The approach for tackling optimization problems was used to model the firefly attraction process[23].

The process of the firefly algorithm is shown in 2.4.

Algorithm 2.4 Firefly Optimization[23]

Step 1: Start

Step 2: Generate initial population of firefly

Step 3: Evaluate fitness of all fireflies from the objective function

Step 4: Update the light intensity fitness value) of fireflies

Step 5: Rank the fireflies and update the position

Step 6: Maximum iteration reached if not Goto Step 3

Step 7: Display optimal result

Step 8: End

#### 2.2.2.6 Harmony Search Algorithm (HSA)

HSA is a metaheuristic algorithm based on the musical performance process. It has three operators: random search, harmony memory, considering rule, and pitch adjusting rule. The three operators' customs of dealing with exploitation or exploration make the HAS a unique practicable optimization algorithm. The HAS is a new metaheuristic optimization algorithm that has been used to solve a variety of difficult problems during the last decade.

Algorithm 2.5 Harmony Search[25]

Step 1: Start

Step 2: Variables initialisation

Step 3: Harmony memory rate consideration

Step 4: Pitch adjustment operation

Step 5: Harmony memory upgrading

Step 6: Is stoppage criterion satisfied if no Goto Step 3

Step 7: End

HSA is a nature-inspired metaheuristic algorithm for solving optimization problems. Its function in this study includes the approach of handling or removing redundancy before classification for accuracy, real-time interval, and the purpose of reducing misclassification. With 14 change-level measurements, implemented with JIT-SDP to fourteen open-source projects in the mobile setting. The algorithms are grouped into two: 4 base classifiers and 4 ensemble classifiers. The best performance result in each category was 49 percent f-measures with NB and 54 percent f-measures with bagging, indicating that no significant differences existed between the two. The study did not take parameter optimization into account. For the first time, the study applied and conducted researches to discover the usage of JIT-SDP in the maritime domain. The study established that SDP can be achieved in the marine domain hence the study presented its discovery and recommendations for practitioners based on the experimental findings. Parameter optimization for the several prediction algorithms was not considered when the study developed the prediction model, save for the usage of default values. The prediction of just-in-time software defects assists in identifying defective modules at an early stage and in a continual manner after software code is changed. The main advantages of JIT-SDP are that it is simple to predict buggy modifications that are mapped to small areas of the code, saving a lot of time and effort. It also made it easier to identify the personnel who made the adjustment, saving time on figuring out who caused the problem. Software systems, subsystems, components, packages, files, classes, methods, and changes in code are all categorized at different granularity levels. It contributes to cost-effectiveness since the smaller granularity allows practitioners to reduce their efforts. The cost-effectiveness of defect prediction at finer-grained levels has been investigated[26].

Thorough analysis on defect-to-non-defect class ratio which is often uneven in many software defect datasets was presented. As a result of the uneven datasets, the prediction model is poor. The issue is referred to as "class imbalance." By addressing class imbalance at the instance or algorithm level, class imbalance learning can increase performance. Only a few industrial implementations of SDP were revealed, however, due to researchers' challenges acquiring industrial defect information due to data protection and privacy concerns. Furthermore, to the best of my knowledge, this is the first case of using HSA-based parameter optimization to choose significant characteristics prior to classifying faulty and non-faulty software datasets. On defect prediction models the used automated parameter optimization system. Several investigations found that because the classifiers employed default parameters, they underperformed. With 18 datasets, this research investigated the impact of parameter optimization on SDP. As a result of the automated parameter adjustment, AUC performance improved by up to 40%. In addition, while tweaking parameter-sensitive classification approaches like the decision tree, the work underlined the significance of exploring the search space.

For cross-project defect prediction, Ryu and Baik used HSA in multi-objective Naïve Bayes learning. Based on three objectives, the HSA looked for the optimal weighted parameters, such as class probability and feature weights. PD, PF, and total performance are the three objectives. As a result, the proposed techniques performed similarly to the within-project DP model in terms of prediction performance, yielding a promising result. The work used HS optimization to anticipate file-level defects. Before classification, HSA is used to pick significant features in this study[16]. The study also proposed a fine-granularity just-in-time DP technique based on-+ a large-scale empirical investigation. A regression model was

utilized to create the prediction model, which was based on data from eleven different projects i.e. 6 datasets and 5 commercial software. Three (3) types of code size, 4 types of dissemination, 3 types of history, 3 types of experience, and 1 kind of purpose were among the 14 change-level metrics employed. Any parameter optimization, on the other hand, is not taken into account. Balance isn't taken into account as much as it should be. 5 MLA, 5 metrics, and 1 optimization technique will be used in this study.

As part of a software measurement program, a research work addressed the implementation of SDP to a Turkish telecommunications corporation. The goal of the program was to improve code quality while lowering costs and defect rates. The Naïve Bayes model was created using file-level code metrics. The work made advice and best practices based on the project expertise. The program, on the other hand, did not give as much thought to parameter optimization[27].

### 2.2.3. Example of Feature Extraction Algorithm

Feature extraction is a part of the dimensionality reduction process, in which, an initial set of the raw data is divided and reduced to more manageable groups. However, these efforts cost money, time and resources. The method of extracting features is useful when there is a large dataset that needs to reduce the number of resources without losing any important or relevant information. Feature extraction helps to reduce the amount of redundant data from the dataset. At the end, the reduction of the data helps to build the model with less machine effort and also increases the speed of learning and generalization steps in the machine learning process. Below are discussions on some of the feature extraction method of defect classification.

#### 2.2.3.1 **Principal Component Analysis**

Pearson proposed Principal Component Analysis in 1901, and Hotelling refined it independently in 1933. Pearson claims that his techniques are simple to apply to numerical difficulties. Although he admits that carrying out computations with four or more variables becomes difficult, he believes they are still doable. PCA can only be done by hand if there are four or less variables. PCA, on the other hand, is most useful when there are more than four variables. To use PCA to analyze data, we need to be well-versed in statistics and matrix algebra. The primary principle behind PCA is to find relationships and patterns in a dataset with several dimensions and reduce it to a single dimension with minimal information loss. The PCA technique is required because high dimensionality data is more difficult due to feature inconsistencies that increase calculation time. An orthogonal linear transformation of a set of variables that optimizes a specified algebraic criterion yields Principal Components. The variance is the criterion to be maximized, and PCA is an unsupervised method in that it does not use the output information.

Due to its twofold nature, PCA is a key method in machine learning. PCA decreases the dataset's dimensionality by removing the dimensions that encode the least salient data and keeping the ones that encode the most important. Because the number of dimensions is reduced, the data takes up less space, allowing for faster classification of larger datasets. Furthermore, by focusing on the most significant dimensions, PCA can project the dataset onto the most meaningful dimensions, revealing patterns.

PCA is a valuable statistical tool for discovering patterns in high-dimensional data that has found applications in disciplines such as face recognition and image reduction. However,

one of the most vital challenges in removal scientific data sets is that the data is frequently multidimensional. The computing time for pattern recognition algorithms might become prohibitive as the number of dimensions reaches hundreds or even thousands. There are numerous dimensionality reduction methods available in different circumstances. This strategy is often referred to as feature or variable selection. qualities that describe the object One issue is that pattern recognition takes a long time to compute. As previously stated, PCA is a numerical process that employs an orthogonal transformation to convert a set of possibly correlated observations into a set of values for linearly uncorrelated variables known as principal components. The number of unique features is less than or equal to the sum of major components. The Matrix approach and the Data method are the two most common PCA methods. To determine the discrepancy covariance structure and express it in the form of a matrix, the Matrix technique uses all of the data in the datasets. A diagonalization approach is used to further decompose the matrix. On the other hand, data methods deal directly with the data[28].

A statistical tool for analysing datasets is principal component analysis. The primary principle behind PCA is to minimize the dimensionality of a dataset with a large quantity of connected variables while preserving as much variance as feasible. Principle component analysis is based on statistics and is based on standard deviation, eigenvalues, and eigenvectors. The entire subject of statistics is founded on the premise that you have a large set of data that you want to evaluate in terms of the correlations between the individual points.

Principal Component Analysis (PCA, commonly known as the Karhunen-Loeve transform) is a technique for data compression and dimensionality reduction. It works by reducing a large dataset to a small number of uncorrelated variables by identifying a few orthogonal linear combinations of the original variables with the highest variance. The direct amalgamation of the original variables with the greatest variance is the first principal component of the transformation; the second principal component is the linear combination of the original variables with the second greatest variance and is orthogonal to the first principal component, and so on. In many data sets, the top few principal components account for the majority of the variation in the original data set, allowing the rest to be ignored with minimum variance loss for data dimension reduction.

The widely used PCA algorithm to find anomalies in the actual world was implemented. The study discovered that using the PCA approach directly leads in poor ROC curve performance; the study addressed the issue and discovered that the main source of the problem is the bias caused by correlation in prediction error terms. Following a thorough theoretical examination, it appears that the correct framework is the KarhunenLoeve expansion, rather than the standard PCA. They proposed the KL expansion as well as a Galerkin method for building a predictive model. This method was then applied to data traces from the Switch network, and we discovered that when temporal correlation is taken into consideration, a significant improvement is achieved.

#### 2.2.3.2 **Principal Component Analysis for Feature Extraction**

For identifying effective features from high dimensional vectors of input data, PCA is among the most prerequisite fundamental approaches of dimensionality reduction.[66] Dimensionality reduction can be categorized into feature selection, which selects a subset of all features, and feature extraction, which combines existing features to generate a new subset of combinations. One of the most prominent approaches used in Feature Extraction is Principal Components Analysis (PCA). PCA employs a signal-based representation criterion, with the main objective of feature extraction being to appropriately represent the samples in a lower-dimensional space, whereas LDA employs a classification-based method. PCA reduces dimensionality while preserving as much arbitrariness as possible in a high-dimensional space.

Because high-dimensional datasets can be reduced to an intrinsic dimension (2D OR 3D) and then plotted using graphs or shown using charts, it can be considered a data visualization method. Annie George presented a study in which the work utilized PCA to reduce dimensionality in an anomaly detection model. The main disadvantage of PCA is that it ignores class reparability since it ignores the feature vector's class label. PCA is nothing more than a coordinate rotation that aligns the transformed coordinate axes along the directions of maximum variance. There is no guarantee that the directions of maximum variance will contain features that are discriminable[28].

### 2.2.4. **Dimensionality Reduction**

Dimensionality reduction is a technique for selecting relevant information from a large number of high-dimensional feature space so as to retain the intrinsic dimensions. For better categorization, regression, presentation, and visualization of data, feature selection is required so as to reduce high-dimensional datasets before classification. It's also important for gaining a deeper understanding of the data's correlations. This allows us to determine the dataset's intrinsic dimensionality and improve generalization. Dimensionality reduction can be done in a variety of ways, both linear and non-linear[29].

The Principal Components Analysis is one of the frequently used linear approaches (PCA). The transformation is also known as the Karhunen and Loeve (KL) transform. The PCA is a linear transformation of a data set from a high-dimensional space to its principal components, which reflect the dataset in lower dimensions. It usually takes use of the data variables' linear relationship. When the relationship between the variables isn't linear, PCA isn't particularly useful. In three dimensions, a helix created by a sine wave over a cosine wave in R 3 would be represented by PCA. One would be the true dimensionality of such a helix. We use non-linear dimensionality reduction approaches to avoid such scenarios when the data has non-linear correlations. One method of completing the non-linear.

To create data-driven decisions, classification and clustering activities are completed. In a variety of industries, these judgments are utilized to improve the quality of service or the production process. In general, clustering and classification algorithms use data to create a predictive model. Then, based on the known dataset, these predictive models are utilized to predict the unknown data. The classification and clustering algorithms take longer to

develop the predictive model when the dataset is huge. Furthermore, the vast magnitude of the dataset affects the predictive model's accuracy. As a result, the dimensionality reduction method can be used to lower the size of the dataset. This strategy is often referred to as feature or variable selection[30].

Wrapper, filter, and embedding techniques are the three types of feature selection methods. The wrapper-based feature selection approach produces all potential variable subsets from a dataset, then evaluates each subset using any of the classification algorithms.

An important variable subset for classification or clustering is chosen based on the evolution's worth. For classification tasks, wrapper-based variable selection produces improved accuracy. It doesn't have a lot of generality because it just improves the accuracy of the classification algorithm that's used to evaluate the variable subsets. Because the variable subsets are evaluated using a classification technique, the computational complexity is higher.

Using any of the statistical measures, the filter-based variable subset selection identifies the significant variable subsets from the dataset. Because no classification technique is used to evaluate the variable subsets, the computational cost is reduced. It has a great degree of generality because the variable selection procedure does not involve a classification algorithm. A part of the classification algorithm is used to evaluate the significance of the variables in the embedded-based variable selection technique. It lacks generality because it employs a classification method, and it also produces higher accuracy just for the classification algorithm that was employed in the variable selection procedure. The embedded-based technique has a lower computation complexity than the wrapper-based

method, but it is higher than the filter-based method. In general, the quality of data-driven decision making is determined by prediction accuracy. To carry out accurate prediction in data-driven decision making, researchers must first improve the accuracy of the classification and clustering method. As a result, this study applied a wrapper-based feature selection strategy for improving classification algorithm accuracy[31].

### 2.2.5 Overview of Machine Learning Algorithms Applied

Machine learning is the study of computer-related methods for learning to make correct predictions or beneficial decisions based on previous observation and experience. ML has grown in popularity, with applications in a variety of fields including natural language processing, speech recognition, computer vision, and gene discovery[22]. ML technique can build models based on labeled historical websites and then these models can be integrated into the browser to detect phishing activities. When a user surfs a new web page, ML models guess the type of the website in real-time and then communicate the outcome to the end-user. The overview of MLAs applied in this study are as follows:

#### 2.2.5.1 Artificial Neural Network (ANN)

ANN is based on the biological neural networks, which is an emulation of the biological neural system. It contains a connected set of artificial neurons and processes information using a method of connectionist for computation. ANN are models which draw their inspiration from biological nervous systems which comprise neural networks. Artificial Neural networks are mathematical models that can be applied to model complex associations between I/O or to find patterns in the data. This technique learns by examples, they are trained with known examples of the problem that knowledge is to be acquired from

when trained well, it can be used effectively to provide a solution to similar problems of unknown instances[32].

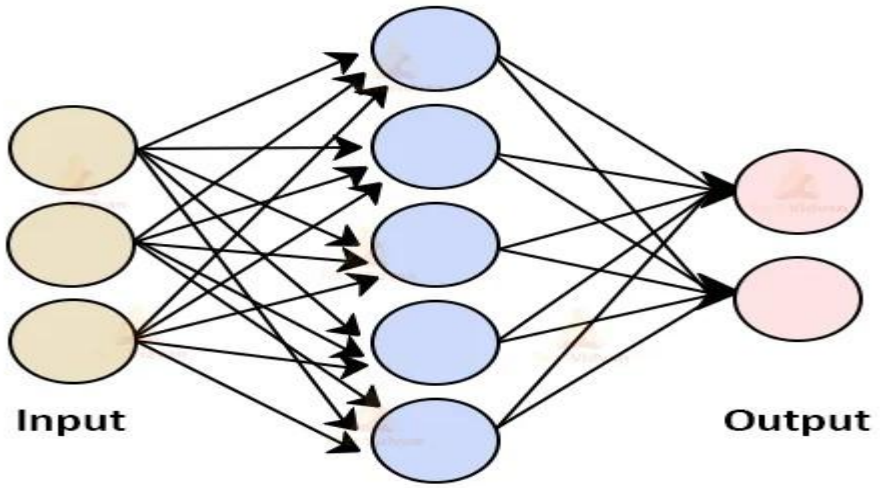



Figure 2.3: Artificial Neural Network[32]

#### 2.2.5.2 Support Vector Machine (SVM)

The support vector machines belong to a generalized family of linear classifiers, which is mostly used in classification, regression, and prediction tools. SVM algorithms apply models linearly to introduce class boundaries non-linearly by converting the instance space applying a nonlinear matching into a new space, a linear model constructed in the new space can then represent a nonlinear decision boundary in the original space[33].

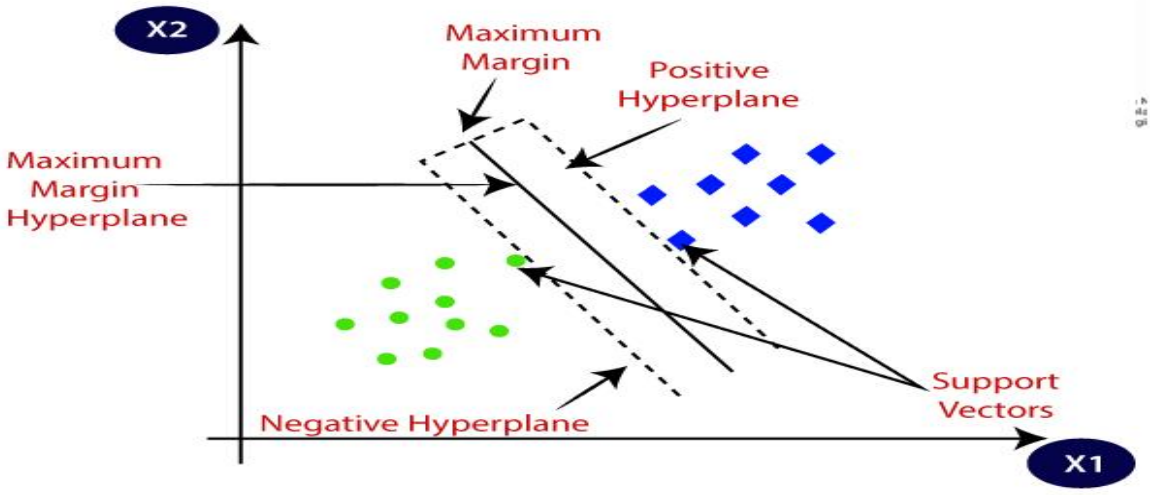



Figure 2.4: Support Vector Machine[33]

### 2.2.5.3 Naïve Bayes (NB)

The naive theorem is combined with an attribute condition where it is assumed to be independent. The method scrutinizes the connection between each feature and module, for each instance in order to calculate a likelihood function for the relationship between feature and module[34]. This technique has been considered to work effectively with actual datasets and when combined with feature selectors eliminate redundant and unimportant features. The Bayes theorem is given as follows:

$$P(H/X) = \frac{P(H)P(\frac{X}{H})}{P(X)} \qquad (2.1)$$

Where P(H/X) signifies the likelihood that event H happened given that test X was positive) P(X/H) symbolizes the prospect that event X happened given that test H was positive. Also,

P(H) represents the possibility that event H occurred and P(X) represents the probability that event x happened[35].

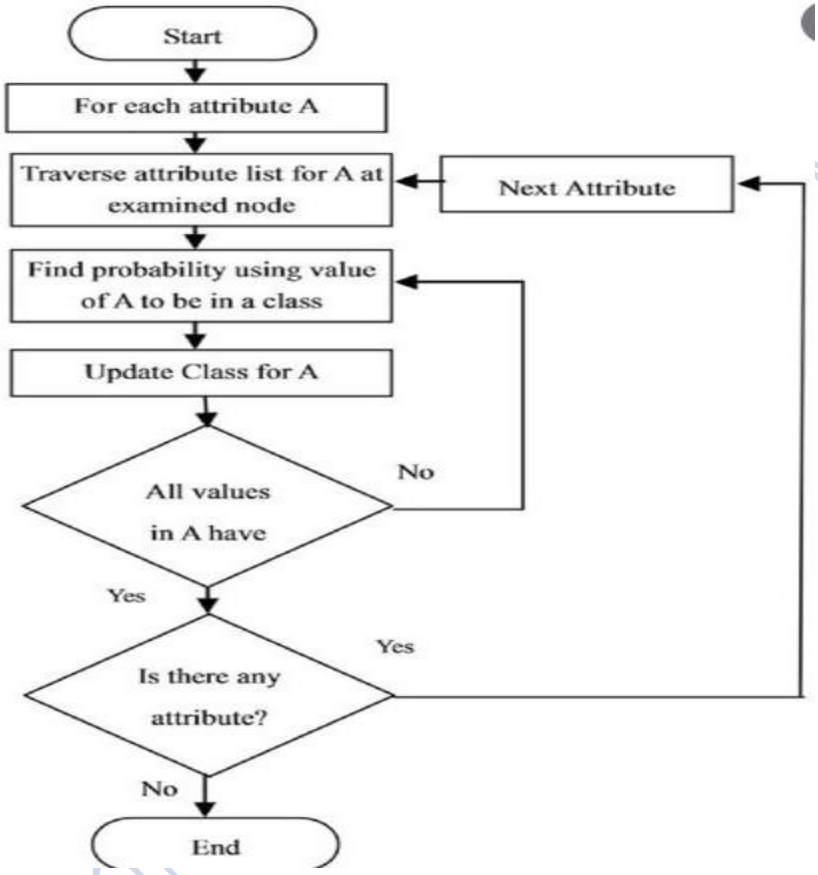



Figure 2.5: Naïve Bayes[34]

### 2.2.5.4 K-Nearest Neighbour (KNN)

The algorithm is used to classify objects and data based on the closest training samples in the feature space. It represents one of the simplest techniques in machine learning. It is simply based on the idea that objects that are near each other will also have similar

characteristics. This technique is also referred to as classification based on memory as the samples of training must be memory during execution. This classification technique is the basic classification technique with little or no prior knowledge about the data distribution

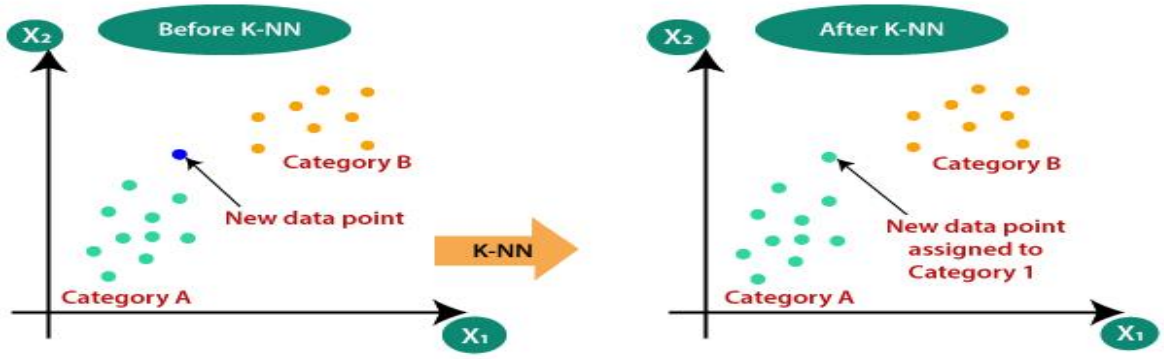



Figure 2.6: K-Nearest Neighbour[35]

### 2.2.5.5 C4.5

Ross Quinlan created the C4.5 algorithm, which is used to produce a decision tree. Also, is expanded in C4.5. C4.5 generates decision trees that is so useful for defects categorization, which is why it's commonly called a statistical classifier (SC). The C4.5 algorithm was defined as "a milestone DT program that is perhaps the ML workhorse most commonly used in practice to date" by the authors of the Weka ML software[36].

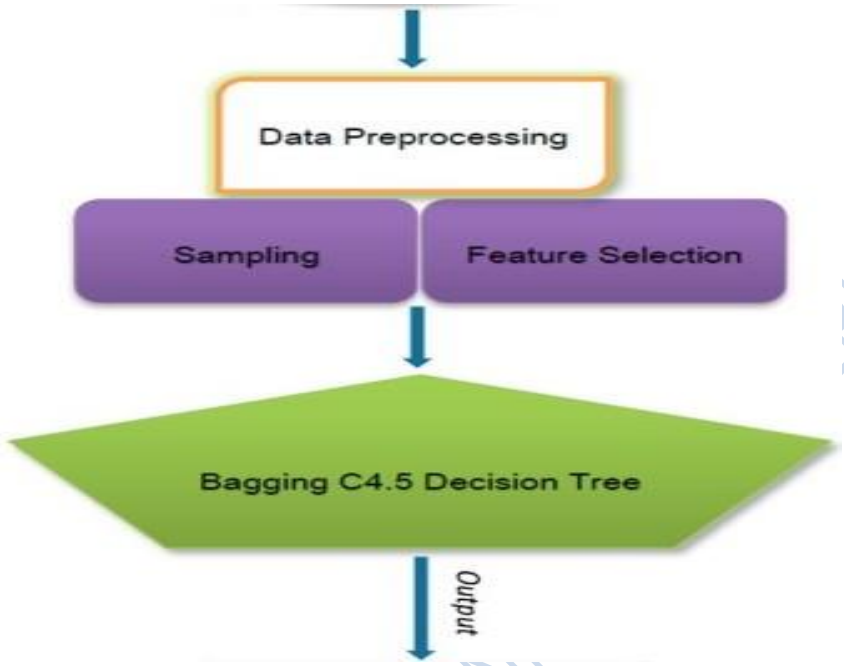



Figure 2.7: C4.5 Decision Tree[36]

### 2.2.6 Features

This section gives a quick overview of the features that this study employed. Two forms of software metrics, namely code and process metrics, were used in this work. Code metrics are software metrics calculated using the program coding structure. They contain information about the code's attributes, size, and structure. Process metrics, which are based on software edits, revisions, changes, etc. The process and code metrics used in our analysis are described in detail below[37].

#### 2.2.6.1 Process Metrics

As previously stated, process metrics are calculated using software change information. This study experimented with 15 distinct process metrics. These metrics include data on new or deleted lines of code, code churn, the number of writers, and the number of versions, among other things. The names and descriptions of these 15-process metrics are listed below:

1. NumberOfVersions: Number of software versions that have been produced since the first release.
2. NumberOfFixes: Number of times a particular module has been scheduled for debugging.
3. NumberOfRefactorings: Number of times a module has been refactored. The "refactoring's" terminology refers to a change in code design that does not affect the software's output.
4. NumberOfAuthors: Number of authors who made changes to the software module
5. Age: How long (Age) has a module existed since its first release.
6. WeightedAge: Age of a module normalized by added lines of code.
7. LinesAdded: Total number of modifications or lines added since the last version
8. MaxLinesAdded: Max lines added in all commits.
9. AvgLinesAdded: Average lines added in all commits.
10. LinesRemoved: Total lines of code deleted or completely modified since the last version
11. MaxLinesRemoved: Max lines deleted in all commits.
12. AvgLinesRemoved: Average lines deleted in all commits.

13. CodeChurn: Code churn is the modifications (change in code, addition or deletion etc.) over a specific period.
14. AvgCodeChurn: Average code churn per commit.
15. MaxCodeChurn: Max code churn per commit[38].

### 2.2.6.2 Code Metrics

Code metrics are the second sort of metric used in this study. For analysis, the study applied 17 different code metrics. Lines of code (size), object coupling, module cohesiveness, class hierarchy, and other object-oriented metrics are included in these code metrics. The following is a list of all 17-code metrics, along with their names and brief descriptions:

1. DIT (Depth of Inheritance Tree): From the root of inheritance, the depth of a class within the class hierarchy.
2. FanIn: Number of functions that call a given function.
3. FanOut: Numbers of functions that are called by a given function.
4. LCOM (Lack of Cohesion Methods): Count the methods that do not share variables and fields with other methods.
5. NOC (Number of Children): Number of child classes of a class.
6. NumberOfAttributes: Total number of attributes of a class.
7. NumberOfAttributesInherited: Total number of inherited attributes of a class.
8. NumberOfLinesOfCode: Total lines of codes in a class/module.
9. NumberOfMethodsInherited: Total number of methods inherited by a class.
10. NumberOfPrivateMethods: Total number of the private methods in a class.
11. NumberOfPublicAttributes: Total number of public attributes in a class.

12. CBO (Coupling between objects): Measure of modules that can access a given module and other modules that this module can access.
13. NumberOfMethods: Number of methods or functions in a class.
14. NumberOfPrivateAttributes: Measure of private attributes in a class.
15. NumberOfPublicMethods: Measure of public methods in a class.
16. RFC (Response for Class): Measure of methods that can be accessed by objects of a given class.
17. WMC (Weighted Methods per Class): Measure of complexities of all methods in a class[39].

### 2.2.6 Feature Selection

Feature selection is a technique for limiting the input variable to your model by only using user data and eliminating noise[40]. It's the process of selecting appropriate characteristics for the ML model based on the sort of problem you're attempting to solve. We accomplish this by retaining or removing critical features without altering them. It aids in the reduction of redundancy in our data as well as the size of our input data.

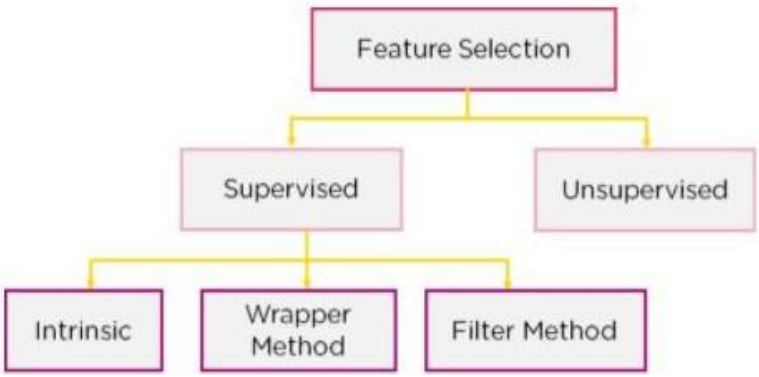



Figure 2.8. Software Defect Management[30]

**Supervised Models:** Supervised FS refers to a process that selects features based on the output label class. They use the goal variables to identify variables that can improve the model's efficiency.

**Unsupervised Models:** Unsupervised FS refers to a method of FS that does not require the output label class. They're what we utilize for unlabeled data[40].

The supervised models can be further divided into three categories:

1. **Filter Method:** In this method, features are filtered out depending on their relationship to the output, or how they correlate with the output. We use correlation to see if the features are favorably or negatively linked with the output labels, and if they are, we drop them. For example, information gain, the Chi-Square Test, Fisher's Score, and so on. These approaches choose features regardless of the classifier type employed. They utilize preprocessing processes before the induction process to remove redundant or unwanted, and noisy characteristics. Without using a mining algorithm, the filter technique for FS scores features based on general criteria such as interclass distance or statistical independence. Wrapper strategic computations are reasonably affordable, but they provide a feature collection that is unrelated to any particular form of the predictive model. The features are ranked and either maintained or removed from the database based on their score.

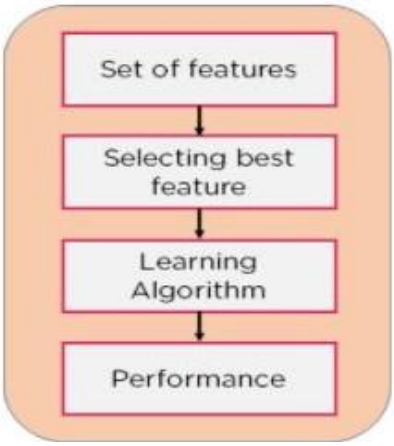



Figure 2.9. Filter Method of Software Defect Management[40]

2. **Wrapper Method**: We divide our data into subgroups and use this to train a model. We add and eliminate features based on the model's output and retrain the model. It uses a greedy strategy to create subsets and analyzes accurately, possibly all the potential feature combinations. For example, Forward Selection, Backwards Elimination, and so on. These methods make use of MLA to evaluate feature subsets using prediction accuracy. Wrapper approaches assess a subset of features in order to uncover possible feature interactions. In bioinformatics, these strategies have been discovered to overcome the challenge of high dimensionality. Because of the method's recurring process, wrapper techniques take longer to compute and are slower. Wrapper methods are divided into two types: greedy and random.

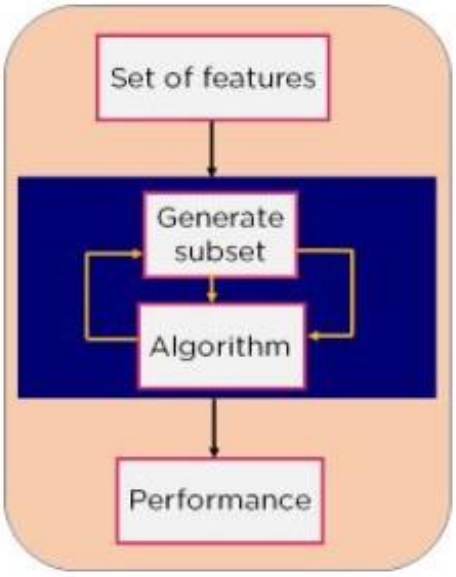



Figure 2.10. Wrapper Method of Software Defect Management[40]

3. **Intrinsic Technique:** To construct the best subset, this method combines the attributes of both the Filter and Wrapper methods. These methods incorporate the benefits of both filter and wrapper methods, resulting in increased efficiency and faster FS. Filter methods are used to choose a feature pool in embedded approaches, and wrapper methods are used to identify the optimal subsets of features from the specified feature pool. These techniques were created to simplify the prediction of ML. The embedded technology is used to handle large databases, avoid pre-specifying criteria, and provide a natural stopping measure from a classifier algorithm's final results. Regularization approaches are frequently sort of embedded FS approach. Regularization methods, also known as penalization methods, impose extra limitations on algorithm prediction optimization[41].

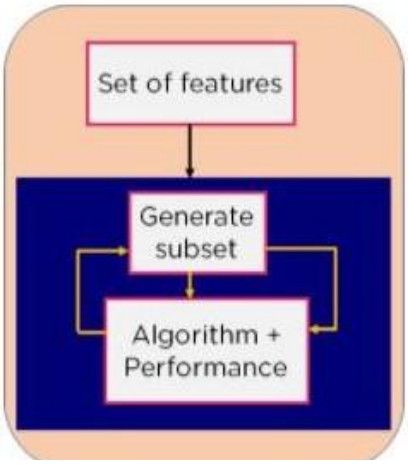



Figure 2.11. Intrinsic Method of Software Defect Management[41]

Many researchers have applied different FS techniques to improve the correct prediction of fault. Some researchers have also performed comparisons of different FS methods used in SDP. These are the summarized lists of the research papers which involved FS.

Feature selection is a preprocessing method applied to pick relevant features. It is a crucial component of methodical style recognition and ML since it can lower computation costs while improving classification performance. Dimensionality reduction occurs when only the significant or relevant features are selected resulting in the precise and accurate classification. When compared to the original datasets, a smaller feature set improves classification accuracy. A universal framework of the FS process is shown below:

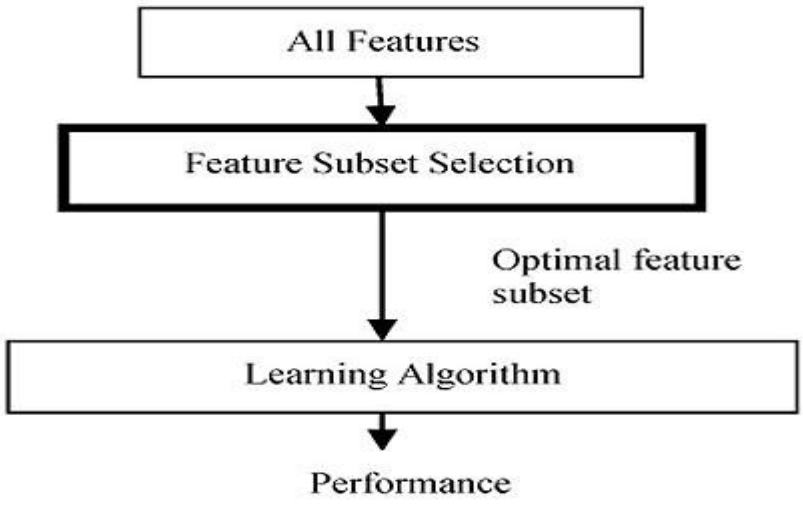



Figure 2.12. Filter Method of Software Defect Management[42]

The two-type classification approaches for software problems were introduced. For an effective categorization of defect, the study used a combination of FS and ensemble learning. The first approach combined FS with ensemble learning for classification, while the second technique focuses solely on ensemble learning (without FS). They compared Greedy Forward Selection to Pearson's Correlation and discovered that greedy forward selection outperformed Pearson's Correlation. They conducted their research utilizing a variety of NASA datasets with code metrics. They determined that their proposed algorithm, which included FS and an ensemble approach, performed admirably for SD classification. For the classification of software flaws, the study compared different feature ranking approaches. They applied ensemble learning to 16 public datasets from eclipse, NASA, and a communications software system with 13 different code metrics. To create and evaluate seventeen ensembles, they used several filter basis ranking approaches in tandem with a

naive Bayes classifier. They concluded that while individual ensembles do not outperform others, only a few ensembles outperform the remaining models as well[42].

For software defect classification, investigations were carried out on four combinations of FS and modeling methodologies. To boost the effectiveness of the classifiers, they employed FS and data sampling. They ran their trials on nine PROMISE datasets using KNN and SVM, with Area Under the Curve as their performance evaluator. They concluded that sampling has a better effect on FS than DP. The study also looked at the effects of the data sample and the selection of the correct attribute method on software defect classification. The research work analyzed five data variations based on sampling vs. non-sampling and attribute selection vs. no attribute selection. They conducted their evaluations using eight NASA public datasets. They concluded that class balance has a very significant influence on classification prediction performance, but attribute selection has a minor impact when compared to balancing[33]. The effects of dataset magnitude, feature set, and attribute selection approaches were considered. They used five public NASA datasets with varied code characteristics to create nine categorization models. As a performance indicator, they employed the area under the curve and accuracy. They found that Naive Bayes performed better for short datasets, while random forest performed best for large datasets. The work presented a new FS method referred to as feature selection employing hybrid data clusters (FeSCH). For FS, they used the density of clusters and feature ranking. They evaluated cross-project defect prediction using their selection technique. They employed F-measure as their performance measure and used AEEEM and Relink CPDP datasets with various types of code metrics. FeSCH outperformed other FS approaches, and its effectiveness was independent of the classifier utilized, according to their findings[43].

Bayesian network were used to pick important Software metrics for classifying defect-prone program modules using object-oriented techniques. The evaluations were carried out using nine PROMISE registry datasets. Their research also discovered two additional metrics: the number of developers and the lack of good code. RFC (Response for Class), LOC (Lines of Code), and LCQ (Lack of Coding Quality) are all critical metrics for defect prediction, according to them. The study selected a smaller collection of code metrics for defect proneness classification using a filter-based feature selection method. Based on top-k and redundancy criteria, they further decreased metrics. In their research, they used 34 PROMISE databases with code metrics. They used six ML classifiers to evaluate their chosen feature set and presented the results using recall, F-measure, and precision.

The importance of nine ML categorization models was also discussed. To balance the datasets, the study used SMOTE and re-sampling, and Fisher linear discrimination analysis was used to identify essential measures. They assessed fifteen PROMISE datasets made up of code metrics and identified the top four classifiers. Recall, precision, and F-measure performance indicators were used to present the findings. The attribute selection method was suggested that used layered iteration of NN to reduce the number of code metrics for software defect categorization. For the evaluation, 19 PROMISE datasets were used, and the findings were presented using AUC. When compared to five other MLAs, they found that their proposed method performed significantly better[44].

### 2.2.8 Software Defect Management

The significance of software defect management is that it improves the quality of software by classifying and fixing the defects in the prompt stage of SDLC. The diverse segments of SDLC are the analysis phase, designing phase, coding phase, requirements gathering phase, testing phase, implementation, and maintenance phase. SDP's role is palpable in developing high-quality software. The describes the schematic representation of software defect management[45].


Figure 2.13. Software Defect Management[45]

The major phases in defect troubleshooting are:

1. Defects Identification
2. Defects Categorization
3. Defects Prioritization
4. Defects Assignment
5. Defects Resolution
6. Defects Verification
7. Defects Closure
8. Defects Management Reporting

**Defect Identification:** The detection of a flaw is the first stage in this process. Ideally, the person who discovers the flaw is a member of the testing team. In the actual world, it may be anyone, including other project team members or, on rare occasions, the end-users.

**Categorization:** When a flaw is discovered, it is reported, it is usually assigned to a specific team member to validate that it is indeed a defect and not an enhancement or another category as determined by the company. The problem progresses to the next stage of the process, which is prioritizing, once it has been classified.

**Prioritization:** Prioritization is usually determined by a mix of the effect or impact on the user, the relative difficulty of fixing the fault, and a comparison to other open problems. Prioritization is often taken care of by an official modification control board, dependent on the magnitude and structural stand of the business. The Management of the organization, the end-users, and the members of the project team should all be involved in determining the priority. Depending on the type of flaw or the degree of the flaw. These flaws constitute a list of priorities for correcting them. It can be handled through a formal channel, or a single team member working on defects can define his or her own priorities for defect resolution.

**Assignment:** A formal channel flaw is assigned to a programmer to address once the priorities have been defined.

**Resolution:** The developer corrects the defect and follows the organization's procedure for deploying the solution to the environment where the problem was first discovered.

**Verification:** The software testing team or the client typically confirms that the fix truly resolved the fault, depending on the environment where the defect was discovered and the patch was implemented.

**Closure:** The reported issue is marked as closed when the problem has been resolved and verified by the authorized persons.

**Management Reporting:** As established reporting requirements, management reports are sent to suitable individuals at regular intervals. This is delivered at regular intervals,

allowing management to see what work has been completed, what work is in progress, and when it will be completed[45].

**Goals of Defect Management Process**

The software building process, not just certain testing or development activities, should adhere to the defect management approach. If an error, for instance, is identified during the testing phase, the question of what will happen to the other flaws still present in the system and could eventually lead to system failure can be posed. Therefore, all processes, including the review process, static testing, inspection, etc., need to be strengthened, and everyone working on the project needs to take the process seriously and help out where it's needed. The organization's management team would be aware and supportive of the defect management process. Depending on the project's goal or other factors, test methods, review procedures, etc., should be chosen. A system when produces an output that differs from the actual business requirement or a divergence from the original or genuine business prerequisite. The testing team encounters a circumstance when the result deviates from the anticipated outcome while executing the test cases. A software defect is essentially an occasion in which the software does not function as intended. A flaw or malfunction in the system that results in unexpected or improper behaviour is known as a defect. There must be an appropriate understanding on how to manage development and release in order to manage projects effectively, but there is a need to also understand how to manage faults. Imagine what would happen if the testing team vocally reported any flaws and the development team updated the defect's status verbally as well. As these faults contain all defects, including

those that have been rectified and are functioning as expected, fixed but are still not functioning, rejected, postponed, and more, the procedure will get more difficult[46].

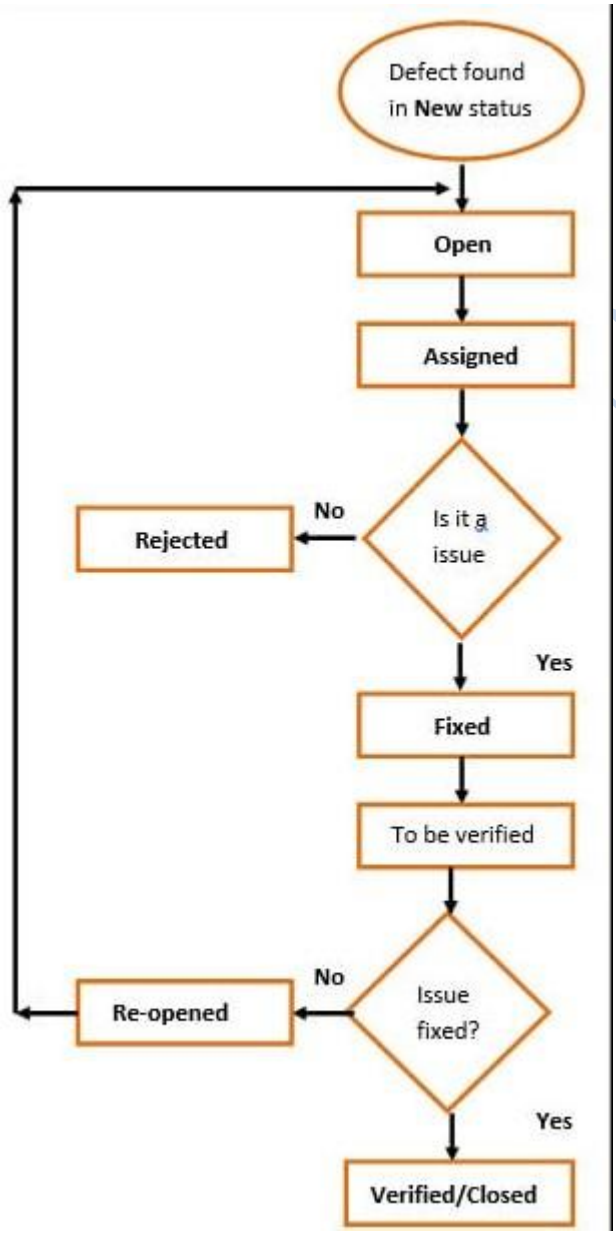
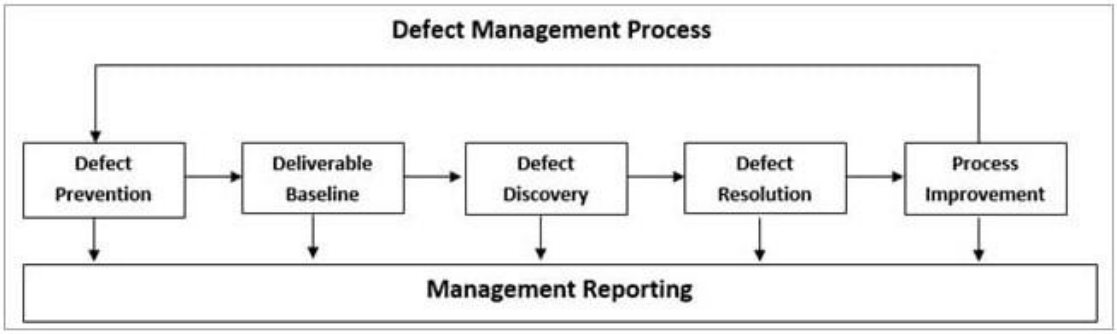



Figure 2.14: defect life cycle[46]

The defect management process is explained below in detail.




Figre 2.15: Defect Management reports[46]

**Defect Prevention:**

The greatest way to eliminate flaws during testing is through defect prevention as opposed to discovering flaws afterward and resolving them. This approach also saves money because addressing flaws discovered during early testing doesn't need much money. Although it is impossible to eliminate every flaw outrightly, you may at least lessen its effects and the cost of fixing it. The steps in defect prevention are as follows:

**Determine Critical Risk:** Find the system's important risks that, if they materialize during testing or at a later point, will have the most impact.

**Estimated Effects Expected:** Determine the financial impact that would result from each key risk, should it actually occur.

**Reduce Predicted Impact:** After identifying all significant risks, focus on the top threats that, if present, could affect the system and attempt to reduce or remove the threat. It lessens

the possible occurrence and the financial impact of risks that cannot be completely eliminated.

**Deliverable Baseline**

A delivery system becomes a baseline when it reaches the predetermined milestone. The product or deliverable progresses through this process from one step to the next, and when it does so, the system's current flaws are likewise carried over to the subsequent milestone or stage. Take coding as an example, followed by unit testing and system testing. System testing is done by the testing team if a developer codes and runs unit tests. Here, the programmer and testing unit are two major accomplishments, and system testing is a third. As a result, if the developer discovers any problems during unit testing, they are not considered defects because they were discovered before the milestone deadline. The developer sends the code to system testing when the programmer and testing team finish, it can be identified that the code has been "baselined" and is prepared for the next milestone, in this instance "system testing."

**Defect Discovery**

As a result, if the developer discovers any problems during unit testing, they are not considered defects because they were discovered before the milestone deadline. The developer turns over the code for system testing when the development and testing team are finished, and makes a system defect-free. However, you can spot the flaws early on before they increase the project's cost. When a flaw is officially reported to the development team

and acknowledged as a defect by that team following examination, we can claim that the defect has been detected.

The steps in defect discovery are as follows:

**Locate a Defect:** Recognize flaws before causing serious issues for the system.

**Report an Error:** The testing team must notify the development team of any defects as soon as they are discovered so that they can investigate and resolve the errors and prepare for the next milestone, in this instance "system testing."

**Acknowledge Defect:** If a genuine problem is assigned to the developers, it is their job to admit and fix it.

**Defect Resolution**

The testing team found the flaw throughout the aforementioned process and informed the development team. The development team must now go forward in order to fix the flaw.

The following steps are involved in fixing defects:

**Prioritize the Risk:** The developers evaluate the flaw and make it the top priority to resolve it. The correction of a problem is given high priority if it has a significant negative impact on the system.

**Fix the Flaw:** The developers correct the flaw according to priority, with higher priority flaws being fixed first and lower priority flaws being fixed last.

**Report the Resolution:** It is the development team's duty to make sure that the testing team is informed when a defect is being addressed and how it was fixed, such as by altering a configuration file or modifying some code. The testing team will find it useful in understanding the reason for the defect.

**Process Improvement**

Though faults are prioritized and addressed during the defect resolution process, this does not imply that lower priority issues are unimportant or do not have a significant influence on the system from a process standpoint. All flaws found are considered critical problems from the perspective of process improvement. Even these tiny flaws present a chance to learn how to enhance the procedure and avoid any flaws that could lead to system failure in the future. Although it is tasking finding a fault that has a modest influence on the system, but it is a big deal when the system experiences such a defect. Everyone stakeholder must look back and determine the source of the issue in order to improve the process. Based on that, you can adjust the base-lining document, review process, and validation process in order to catch defects earlier in the process, when they are less expensive[46].

## 2.3 Review of Empirical Studies

According to the literature survey, some researchers have examined the performance and comparison of some selected MLA for the prediction of software defects. The study considered 3 diverse versions of the eclipse version control system. Data were split into training and tested sets. For the training purpose of the model SVM and ELM were used. The operational system of the classification model was achieved using some selected

evaluation metrics. The result recorded the SVM to be the best fit for the cross-version defect prediction.

An optimization algorithm called a Genetic Algorithm (GA) was applied to pick the most important features in order to reduce the classification time and the possibility of misclassification. One Classifier, ANN was employed for defect classification. The system was implemented in MATLAB (R2018a) simulation environment with the execution of a numerical toolkit and the operating model was measured using four evaluation metrics. The system used Eclipse, Lucene, Equinox, and ECLIPSE JDT CORE to evaluate the performance. The result of the research work is fully represented in a tabular form below:

Table 2.1 Research Result[47]

| **MLA** | **Datasets** | **Evaluation Metrics** | | | |
|---|---|---|---|---|---|
| Classifier | **Datasets** | Accuracy | Precision | Recall | F-Score |
| ANN | ECLIPSE JDTCORE, | 86.93 | 53.49 | 79.31 | 63.89% |
| ANN | ECLIPSE PDE UI, | 83.28 | 31.91 | 45.45 | 37.50% |
| ANN | EQUINOX FRAMEWORK | 83.43 | 57.69 | 45.45 | 50.84% |
| ANN | LUCENE | 91.30 | 33.33 | 50.00 | 40.00% |



Furthermore, a software defect predictive system in the year 2020 using a genetic algorithm with a shuffled frog leap algorithm was developed. The study assessed the accuracy, precision, classification, recall, and F-measure for various classifiers used. The ANN optimizations assumed that more than two algorithms for one optimization have been implemented. The optimization used a meta-heuristic to select the best algorithm for classification. The hybrid optimization technique for creating the linkages method was

applied for the dimensional synthesis of the mechanism. The results, 5.94% of the NN-hybrid classifier showed that it outperformed the fuzzy algorithm by 3.59% and the 1.42% value of the NN-Lm training respectively[47].

More also, the research work conducted a defect classification model using machine learning-based algorithms. The Genetic Algorithm which is among the most used meta-heuristic algorithms was applied to select the discriminant features. The extracted features were classified as defective or non-defective using Random Forest, Decision Tree, and Artificial Neural Network classification technique. Furthermore, the techniques were evaluated using accuracy, f-score, precision, and recall. Among all the learning algorithms used, the random forest outperformed other algorithms in terms of accuracy, precision, and f-score with an average score of 83.40%, 53.18%, and 52.04% respectively. In a nutshell, the Neural Network recorded the result in terms of recall with an average score of 60% among the algorithms[48].

Some researchers assessed the previous research works concerning software defects that employed data mining techniques, datasets, performance measures, and tools used, which are classified into three such as classification, clustering, and regression methods. Finally, there is all possibility of expanding the tentacle of this research as there is a need to review books, dissertations, tutorials, and Thesis[49].

A study discussed how data mining techniques were used for defect classification. The study was able to increase the effectiveness and value of software development using data mining to analyze and predict volumes of defective datasets gathered in the software development. The study reviewed the previous papers and showed that these techniques have performed better results when performed on different data sets.

Furthermore, the forecasting error of software using an Artificial Neural networks classifier was analyzed. Results showed that the NN model outperformed better in terms of correctness as compared to other classifiers and hence the study recommended software fault prediction tests using Neural Networks[50].

In the literature review conducted by Prasad, Florence, and Arya in the year 2015 metrics-based software defect classification using data mining and classifier-based techniques were analyzed. The major purpose of the research is to help programmers predict errors in software using data mining techniques to improve the quality of the software. Several classification techniques were applied as well as classified evaluating metrics.

Last but not the least, an analysis was conducted on how kernel techniques can be applied to classify defects in software development since the class discrepancy can influence negatively, the correct operation of defect classification. Two classifiers were applied which are, asymmetric kernel partial least squares (AKPLS) and asymmetric kernel principal component analysis (AKPCAC) classifier. The study aimed at solving the class inconsistency or imbalance issue. Finally, the study achieved its aim by using a kernel function for the asymmetric partial least square algorithm and asymmetric PCA algorithm, respectively. The kernel function used for the two classifiers is the Gaussian function. The experiment was conducted in NASA and SOFT LAB datasets using the *F*-measure, Friedman's test, and Tukey's test to confirm the validity of our methods[51].

## 2.4 Conceptual Framework

The prolonged processing of prediction can lead to misclassification, most especially when a large dataset is used for classification. In other words, misclassification and prolonged processing is inevitable when a large dataset is used for prediction. That is why this study has applied a meta-heuristic optimization algorithm for feature selection (FS) to reduce high feature dimensionality which may lead to prolonged classification time and misclassification. The HSA was employed for feature selection, and five learning algorithms- SVM, ANN, KNN, NB, and C4.5 were applied for classification. Also, Eclipse datasets were used to predict software defect. The operational output of the model was achieved using some evaluation metrics such as precision, accuracy, recall, classification time, and F1 score.

## 2.5 Summary of Literature Reviewed

Several seasoned researchers have carried-out studies in the area of SDP. After a painstaking comparative analysis of all the literature reviewed, it was shown that applying a huge dataset for evaluation of software defect classification or prediction may suffer high feature dimensionality. Researchers in the past, have applied numerous algorithms such as PCA, ICA, and LDA, to reduce dimensionality problems. But these aforementioned algorithms have not performed optimally because they linearly reduce without taking into consideration how relevant the features are. It is, therefore, necessary to consider the most relevant feature for software prediction. Hence, this study applied five learning algorithms for classification, five evaluation metrics using Eclipse dataset with a meta-heuristic optimization algorithm for FS. Hence, this study applied five learning algorithms for classification, five evaluation metrics, and Eclipse dataset with meta-heuristic optimization algorithms for FS.



**Endnotes**

[1] O. Alqasem & M. Akour *"Software fault prediction using deep learning algorithms"* **International Journal of Open Source Software and Processes.** 2019

[2] Shreya Bose, "*Defect management in software testing*." Available at https://www.browserstack.com/guide/defect-management-in-software-testing/. 2020

[3] R. Shatnawi, "*The application of roc analysis in threshold identification, data imbalance and metrics selection for software fault prediction*," **Innovations in Systems and Software Engineering** 13, no. 2, 2017, 201-217.

[4] A. Chauhan & R. Kumar "*Bug severity classification using semantic feature with convolution neural network computing in engineering and technology*" **In Computing in Engineering and Technology**, Springer, Singapore, 2020, 327-335

[5] N. Hussain, & L.Bixin, "*Performance comparison of ML classifiers in software defects prediction*" Abstract: IOSR **Journal of Computer Engineering** (IOSR-JCE), Available at https://doi.org/10.9790/0661-2205034858/ 22(5). 2020

[6] B. Wenbin & F. *Yu "A feature selection framework for software defect prediction using ISFLA."* **In IOP Conference Series: Materials Science and Engineering,** vol. 677, no. 5, 2019, 052121.

[7] Akimova, E. N., A. Y. Bersenev, A. A. Deikov, K. S. Kobylkin, A. V. Konygin, P. M, Ilya & V. E. Misilov. *"A survey on software defect prediction using deep learning."* Mathematics 9, no. 11 2021, 1180.

[8] Hamdy & El-Laithy. "*Smote and feature selection for more effective bug severity prediction*. **International Journal of Software Engineering and Knowledge Engineering**, 29(06), 2019, 897-919.

[9] E. Ehsan, S. Kanwal & A. N. Asif. *"A new ensemble approach for software fault prediction."* In *2020 17th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, **IEEE**, 2020, 407-412.

[10] S. Balan, "*Metaheuristics in optimization: algorithmic perspective*" available at https://www.informs.org/Publications/OR-MS-Tomorrow/Metaheuristics-in-Optimization-Algorithmic-Perspective/. 2022

[11] S. V. Rao, "*An artificial neural network genetic algorithm with shuffled frog leap algorithm for software defect prediction*.", 2020, 831–836.

[12] B. Patra & S. Dash, "*A FRGSNN hybrid feature selection combining FRGS filter and GSNN wrapper.*" **International Journal of Latest Trends in Engineering and Technology**, 7(2), 2016, 8–15.

[13] D. Wang, D. Tan, & L. Liu "*Particle swarm optimization algorithm: an overview*." **Soft computing 22**, no. 2, 2018, 387-408.

[14] D. Nilanjan, J. Chaki, L. Moraru, S. Fong & X. Yang. "*Firefly algorithm and its variants in digital image processing: A comprehensive review." Applications of firefly algorithm and its variants*, 2020, 1-28.

[15] D. Gao, X. Li & H. Chen, "*Application of improved particle swarm optimization in vehicle crashworthiness*." **Mathematical Problems in Engineering**/ 2019

[16] M. Ruchika, N. Nishant, S. Gurha & V. Rathi. *"Application of particle swarm optimization for software defect prediction using object oriented metrics*." **In 2021 11th International Conference on Cloud Computing, Data Science & Engineering** *(Confluence)*, **IEEE**, 2021, 88-93.

[17] F. Matloob, S. M. Aftab, M. A. Khan, A. Fatima, M. Iqbal & N. S Elmitwally, *"Software defect prediction using supervised machine learning techniques: A systematic literature review."* ***Intelligent Automation and Soft Computing***, *29*(2), 403–421, 2021

[18] K. B. Kiran, J. Gyani & G. Narsimha, "*Software defect prediction using ant colony optimization*" **Int. J. Appl. Eng.** Res. 13, no. 19, 2018, 14291-14297.

[19] S. Arslan & C. Ozturk, "*Feature selection for classification with artificial bee colony programming*" **In Swarm Intelligence-Recent Advances, New Perspectives and Applications. Vienna, Austria: IntechOpen**, 2019.

[20] N. Kalaivani, & R. Beena "*Overview of software defect prediction using machine learning algorithms*." ***International Journal of Pure and Applied Mathematics***, *118*(20), 2018, 3863–3873.

[21] Y. Yang "*Software defect prediction model research for network and cloud software development*." **5th International Conference on Mechatronics, Materials, Chemistry and Computer Engineering** *ICMMCCE 2017,* 717–723.

[22] S. Punitha, A. Amuthan & K. S. Joseph "*Enhanced monarchy butterfly optimization technique for effective breast cancer diagnosis*." **Journal of Medical Systems**, 43(206), 2019, 1–14.

[23] A. Ritthipakdee, A. Thammano, N. Premasathian, & D. Jitkongchuen, "*Firefly mating algorithm for continuous optimization problems*.", **Computational Intelligence and Neuroscience**, 2017.

[24] M. Nasir, A. Sadollah, J. H. Yoon & Z. W. Geem, "*Comparative study of harmony search algorithm and its applications in China, Japan and Korea*." **Applied Sciences (Switzerland)**, 10(11), 2020¸ 1–26.

[25] J. H. Kim, *"Harmony search algorithm: a unique music-inspired algorithm."* **Procedia Engineering**, 154 2016, 1401-1405.

[26] G. Catolino, N. DI & D. Ferrucci "*Cross-project just-in-time bug prediction for mobile apps: an empirical assessment*." **In 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)**, IEEE 2019, 99-110.

[27] P. Wei, "*A study of principal component analysis on classifiers using histogram of gradients features, final report*" available at: http://www.contrib.andrew.cmu.edu/~pwei/papers/FinalReport. Pdf/. 2022

[28] N. Varghese, & V. Verghese, "*A survey of dimensionality reduction and classification*." 3(3), 2012, 45–54.

[29] M. Y. Ramamurthy, H. Robinson, S. Vimal & A. Suresh. *"Auto encoder based dimensionality reduction and classification using convolutional neural networks for hyperspectral images."* **Microprocessors and Microsystems,** 2020, 79.

[30] S. A. Ebiaredoh-Mienye, T. G. Swart, E. Esenogho & I. D. Mienye. "*A machine learning method with filter-based feature selection for improved prediction of chronic kidney disease."* **Bioengineering** 9, no. 8, 2022, 350.

[31] S. V. Rao, "*An artificial neural network genetic algorithm with shuffled frog leap algorithm for software defect prediction*.", 2020, 831–836.

[32] A. Malak, A. Fahd, R. A. Mustafa, S. M. Mohammad, D. H. Alhamed, H. S. Altamimi & S. M. Chrouf. "*An assessment of lexical, network, and content-based features for detecting malicious URLs using machine learning and deep learning models."* **Computational Intelligence and Neuroscience**, 2022.

[33] J. Kang, R. Duksan & B. Jongmoon *"Predicting just-in-time software defects to reduce post-release quality costs in the maritime industry*." ***Software: Practice and Experience*** 51, no. 4 2021, 748-771.

[34] Rizwan, Muhammad, A. Nadeem, M. A. Sindhu. P. S. Kumar, R. B. Mishra & A. K. Triphathi. "*Software bug prediction prototype using bayesian network classifier: A comprehensive model*." **Procedia computer science** 132, 2018, 1412-1421.

[35] From Wikipedia, the free encyclopedia, "*C4.5 algorithm*", https://en.wikipedia.org/wiki/C4.5_algorithm/. 2022

[36] M. Rizwan, A. Nadeem & M. A. Sindhu, "*Empirical evaluation of coupling metrics in software fault prediction*," **In 2020 17th International Bhurban Conference on Applied Sciences and Technology** (IBCAST), IEEE, 2020, 434-440.

[37] C. Ni, W. S. Liu, X. Chen, Q. Gu, D.-X. Chen, & Q.-G. Huang. "*A cluster-based feature selection method for cross-project software defect prediction*," **Journal of Computer Science and Technology**, vol. 32, no. 6, 2017, 1090–1107.

[38] P. R. Anusha, G. Joshi & S. Rane. "*Quality and reliability studies in software defect management: a literature review*." **International Journal of Quality & Reliability Management**. 2021.

[39] S. Bose, "*Defect management in software testing*." Available at https://www.browserstack.com/guide/defect-management-in-software-testing/. 2020

[40] S. Shaik, & M. Tech "*A novel approach to iris recognition based on feature level fusion using*." **International Journal of Scientific Development and Research** (IJSDR), 2(7), 2017, 287–292.

[41] Hamdy & El-Laithy. "*Smote and feature selection for more effective bug severity prediction*." **International Journal of Software Engineering and Knowledge Engineering**, 29(06), 2019, 897-919.

[42] B. Jason, "*How to choose a feature selection method for machine learning*" **Machine Learning Mastery** 2019, 10

[43] R. Jayanthi, & L. Florence. *"Software defect prediction techniques using metrics based on neural network classifier."* **Cluster Computing** 22, no. 1 2019, 77-88.

[44] T. Hamilton, "*Defect management process in software testing* (Bug Report Template)/ available at "https://www.guru99.com/defect-management-process.html/. 2022

[45] Anonymous, "*Defect management process: how to manage a defect effectively*." Available at https://www.softwaretestinghelp.com/defect-management-process/ 2022

[46] B. L. Olatunji, S. O. Olabiyisi, C. A., Oyeleye, B. A. Sanusi, A. O., Olowoye & O. Ofem "*Development of software defect prediction system using artificial neural network*." available at https://pdfs.semanticscholar.org/6047/c62f7d169b133e6389d6840eaca0201e140c.pdf/. 2022

[47] M. Kumari, M. Sharma & V. Singh, "*Severity assessment of a reported bug by considering its uncertainty and irregular state*." **International Journal of Open Source Software and Processes (IJOSSP)** 9, no. 4, 2018, 20-46.

[48] L. Francesco. "*Machine learning for software fault detection: issues and possible solutions*." 2022.

[49] A. Mishbahulhuda "*Data mining techniques in software defect prediction*". **International Journal of Advanced Research in Computer Science and Software Engineering Research***,* 7(3), 2017, 301–303.

[50] S. S. Rathore & S. Kumar "*Towards an ensemble-based system for predicting the number of software faults*," **Expert Systems with Applications** 82, 2017, 357-382.

# Chapter Three
# Methodology

## 3.1 Research Approach

To reduce prolonged processing of datasets which results in misclassification, this study has come up with a model developed using a meta-heuristic optimization algorithm, 5 MLAs, and 5 metrics. The significance of HAS in this study is to pick a relevant feature for defect prediction. The 5 MLAs were applied for classification and 5 metrics were used to determine the level of defect.

In the first step, the study used HSA subset selection method to select pertinent feature subsets. The output of the feature selection method retains the meaningful representation or intrinsic dimension of the original dataset

In the second step, the subset was used to predict defects using 5 MLAs. The results of these experiments are described in detail in the results and discussions section

Lastly, 5 metrics were applied to determine the genuineness and the level of defect

The methodology and procedures employed in this study are discussed in this chapter. Here are the study's approaches utilized to actualize the result which are:

1. System Design
2. Dataset Preprocessing
3. Dataset overview
4. Datasets used

## 3.2. System Design

Software Defect Predictive Model

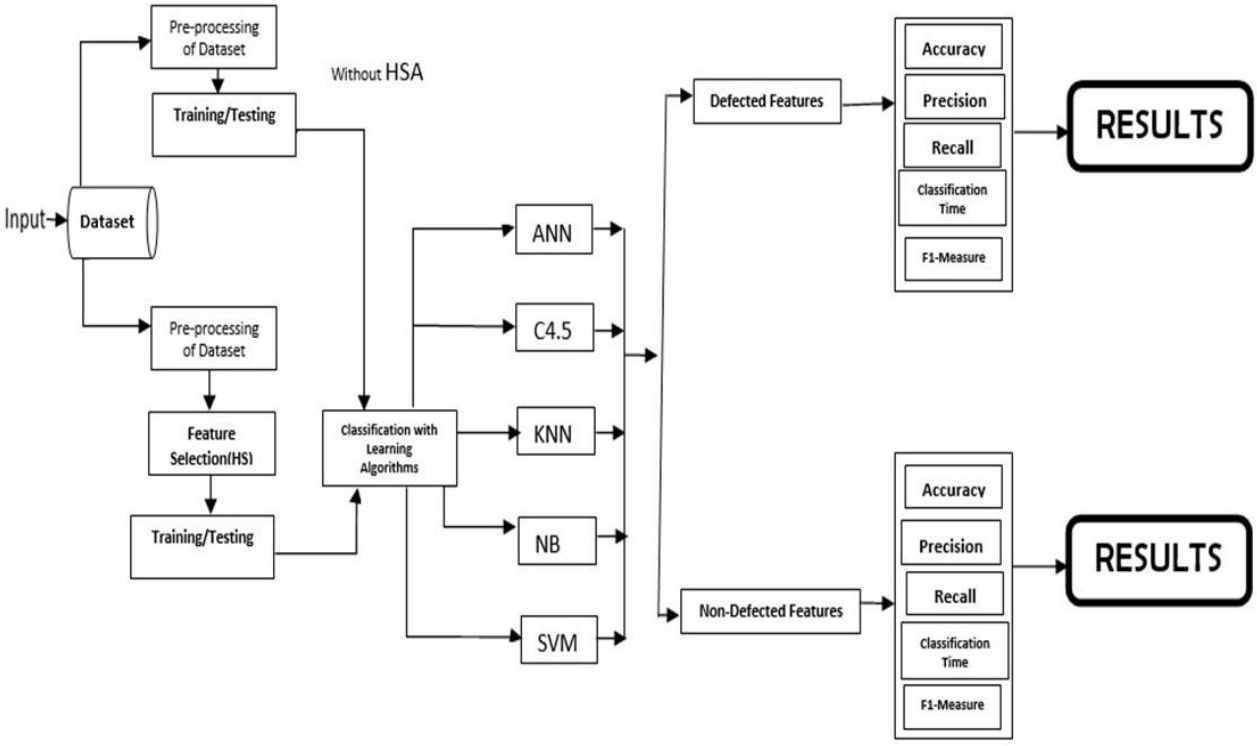



**Figure 3.1: Software Defect Predictive Model**

This study used a feature selection (FS) meta-heuristic optimization approach to reduce high feature dimensionality, which can contribute to long classification times and misclassification. For feature selection, the HSA was used, and for classification, five learning algorithms were used: SVM, ANN, KNN, NB, and C4.5. In addition, Eclipse dataset were used for prediction. Some assessment criteria, such as precision, accuracy, recall, classification time, and F1 score, were used to obtain the model's operational output.

### 3.2.1 Dataset Preprocessing

Data preprocessing is an iterative process for transforming raw data into usable and intelligible formats. By suitably modifying and scaling the entire dataset, data preprocessing tries to make the training/testing process easier. Prior to the training of datasets, the ML models, and preprocessing are required. Outliers are removed during preprocessing, and the features are scaled to an equivalent range[1].

Dataset pre-processing in ML is a sensitive and significant area that helps enhance the selection of the discriminant feature of the dataset for classification. The technique of cleaning and organizing the dataset make it appropriate for training, constructing, and testing the Machine Learning algorithms. Dataset pre-processing is also referred to data mining method of transforming a dataset into a readable and more understandable format.

The supervised learning algorithms in this study required the availability of previous faulty data of software releases. The data for this study was acquired from http://bug.inf.usi.ch/download.php which is a databank that is made available for public use,

Dataset pre-processing marks the initiation of the process and it is necessary to format and familiarize the dataset with the environment so as to make it useable for classifiers.

The initiation of dataset pre-processing is required to clean, format, and organize the raw dataset, thereby making it useable for Machine Learning models. Dataset Acquisition is primarily, the fundamental prerequisite in data pre-processing as it precedes and is required for the development of any machine learning model. To build and develop Machine Learning models is to first acquire the relevant dataset. Thereafter, the dataset was split into

two separate sets – training and testing. Follow by importing the dataset that has been tested and trained into the ML model for classification.

### 3.2.2 Datasets Overview

Eclipse dataset were used in this study which contained a count of defects. The dataset is free but are defective. The dataset used for prediction is Eclipse dataset. The dataset has 17-fileds metrics and five type of defects. The features section has detailed information on all of these measures.

The table below describes the attributes of the datasets used in the cause of this study.

**Table 3.1: Information on Datasets**

| Dataset | Description | Instances |
|---|---|---|
| Eclipse | Tools of a user interface for developing and creating Eclipse plug-ins and features. | 997 |

### 3.3 Requirements Specification

The device with which this study generated outputs was a laptop computer with specifications: 3.4 GHz Intel Core i7 8GB RAM, running Windows OS (64-bit). The simulation took place in python 3.10 to evaluate the efficiency of the classifiers which consist of naïve Bayes, C4.5, Support Vector Machine, K-nearest neighbour, and Artificial Neural Network. The study applied an optimization algorithm to obtain relevant features from the Eclipse dataset and classified. Finally, the operational effectiveness of the classifiers was achieved using precision, accuracy, recall, classification time, and F1-score evaluation metrics.

### 3.4 Data Acquisition

The supervised classifiers in this study required the accessibility of preceding defective software dataset release. The dataset for this study was acquired from http://bug.inf.usi.ch/download.php, the dataset bank repository, available for public use. For this study, the SDP dataset that was used is Eclipse. This software defect dataset contained a different piece of information but the one with the relevant parameter that suits the research work was employed during the cause of implementation.

### 3.5 Evaluation Metrics

The performance evaluation metrics employed are Accuracy, Recall, classification time, Precision, and F-score

i. Sensitivity = $\frac{TP}{TP+FN} X\ 100\%$ ----------3.1

ii. Accuracy = $\frac{TN+TP}{TP+FP+TN+FN} X\ 100\%$ ----------3.2

iii. Precision = $\frac{TP}{TP+FP}$ ----------3.3

iv. F-measure = $\frac{Precision \times \frac{TP}{TP+FN}}{Precision + \frac{TP}{TP+FN}}$ ---------- 3.4

where TP, FP, TN, and FN are defined as follows:

**True Positive (TP):** The outcome is the correctly predicted positive, indicating that the actual and projected results are both "yes."

**True Negative (TN):** The outcome is the correctly predicted negative, indicating that both the actual and expected result values are "No."

**False Positive (FP):** This indicates that the actual result is no, but the projected outcome is yes.

**False Negative (FN):** This indicates that the actual result is yes, but the projected outcome is no.

### 3.5.1 Accuracy

When using asymmetric datasets, where the false positive and false negatives have the same value, accuracy is defined as the percentage of properly predicted values to the total. It's the proportion of subjects who have been correctly identified to the total number of subjects. Accuracy is the most intuitive one. The following question is answered by accuracy:

How many nations of Africa did we correctly label corrupt out of all the nations?

Accuracy = (TP+TN)/(TP+FP+FN+TN)

**numerator**: refers to all correctly labeled subjects (All trues)

**denominator**: refers to all subjects

### 3.5.2 Precision

Precision is the function of relevant instances among the recovered instances. This is the ratio of the correctly and positively labeled by the program to all positive labeled.

Thus, the precision answers the following: How many presidents of Africa nations that we labeled as competent are actually competent?

Precision = TP/(TP+FP)

**numerator**: +ve labeled competent president.

**denominator**: all +ve labeled by the program (whether they're competent or not in reality).

### 3.5.3 Recall

Recall is also known as Sensitivity

The percentage of accurately forecasting positive for everyone in the actual result is referred to as recall. It is also the ratio of the correctly positive labeled by the program to all who are competent in reality.

Recall answers the following question: Of all the presidents who are corrupt, how many of those do we correctly predict?

Recall = TP/(TP+FN)

**numerator**: positively labeled corrupt Presidents.

**denominator**: all presidents who are corrupt (whether detected by the program or not)

### 3.5.4 F1-Score

**F1-score** is also known as F-Score or F-Measure. The F1-Score is the sum of Precision and Recall, adjusted for false positives and negatives. When the data distribution is imbalanced, F1-Score is more effective than accuracy. Precision and recall are both taken into consideration while calculating the F1 Score. It's the harmonic mean of memory and precision.

F1-measure is excellent if a balance can be accomplished between precision (p) and recall (r) in the system. Oppositely F1-Score is not so high if one measure is improved at the expense of the other.

For example, if P is 1 and R is 0, F1 score is 0.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)[1].

### 3.5.5 Classification Time

Classification time is the time to taken to complete software defect prediction using MLA.

## Endnotes


1. S. Ghoneim "*Accuracy, recall, precision, f-score & specificity, which to optimize on*?" 2022 available at https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124/.

2. U. Zahid, S. R. Naqvi, W. Farooq, H. Yang, S. Wang & N. V. Dai-Viet *"A comparative study of machine learning methods for bio-oil yield prediction–A genetic algorithm-based features selection."* **Bioresource Technology**, 2021, 335

# Chapter Four

# Results and Discussion of Findings

## 4.1 Results

This section presents the simulation environment where the results of defect classification were generated. It mentions the result to state clearly how the objectives listed for the study as well as the workability of the methodology used to achieve these objectives. The experiment was carried out by selecting relevant features from the datasets that were used in this study using HAS. Figure 4.1 below shows the simulation interface for the experimental setup.

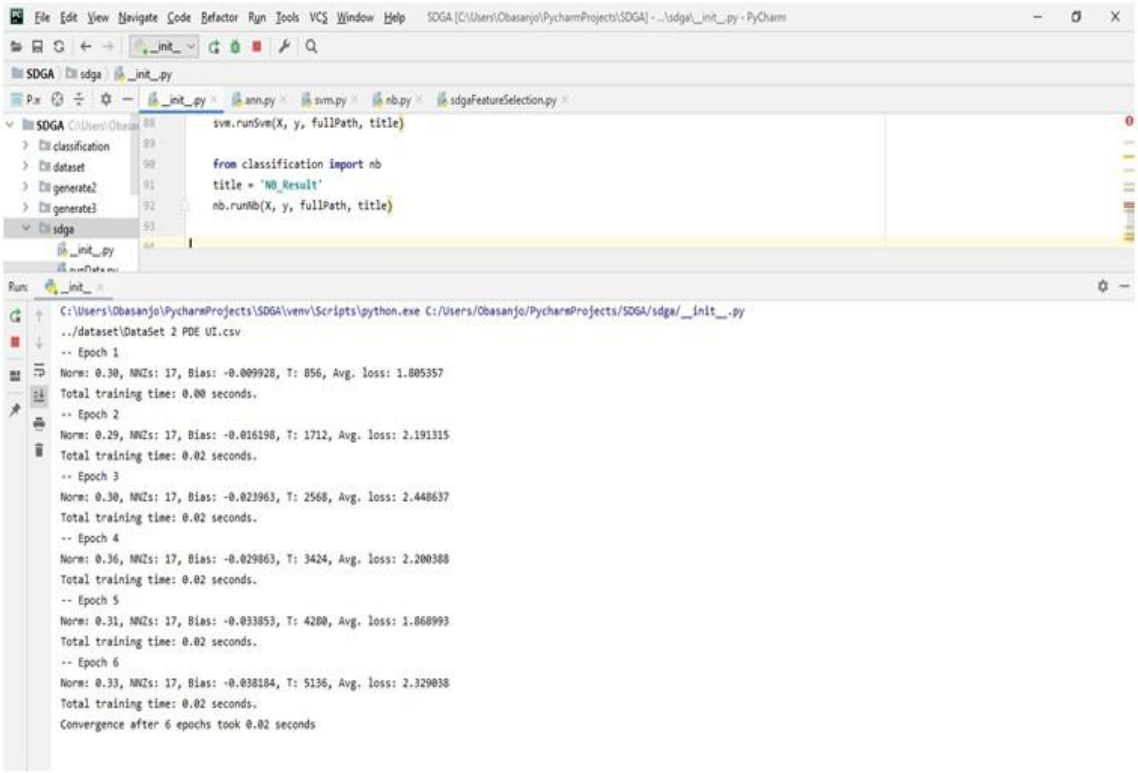



Figure 4.1: **Interface for Experimental Setup**

The FS, defects classification as well as performance evaluation of the results (defect or non-defect) were conducted for different datasets used. The results were provided using the following tables.

**Table 4.1: The Result of Performance Metrics for Eclipse Dataset using ANN for Prediction (without HSA)**

| **Classifier** | **Evaluation Metrics** | | | | |
|---|---|---|---|---|---|
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| ANN | 89.16 | 0.8900 | 1.000 | 0.9400 | 24.14 |

The description of Table 4.1, using ANN classifier with the five evaluation metrics without selecting discriminant feature before prediction are as follows:

The accuracy is 89.16%,

The precision 0.8900,

Recall is 1.000,

F1-Score is 0.9400 and;

Classification Time (s) is 24.14

| **Table 4.2: The Result of Performance Metrics for Eclipse Dataset using ANN for Prediction (with HSA)** | | | | | |
|---|---|---|---|---|---|
| **Classifier** | **Evaluation Metrics** | | | | |
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| ANN | 85.97 | 0.8600 | 1.000 | 0.9200 | 24.09 |

The description of Table 4.2, using ANN classifier, five evaluation metrics with the selection of discriminant features before prediction are as follows:

The accuracy is 85.96%,

The precision 0.8600,

Recall is 1.000,

F1-Score is 0.9200 and;

Classification Time (s) is 24.09

| **Table 4.3: The Result of Performance Metrics for Eclipse Dataset using C4.5 for Prediction (without HSA)** | | | | | |
|---|---|---|---|---|---|
| **Classifier** | **Evaluation Metrics** | | | | |
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| C4.5 | 78.09 | 0.8600 | 0.8800 | 0.8700 | 24.44 |

The description of Table 4.3, using C4.5 classifier with the five evaluation metrics without selecting discriminant feature before prediction are as follows:

The accuracy is 77.78%,

The precision 0.8800,

Recall is 0.8700,

F1-Score is 0.9000 and;

Classification Time (s) is 25.04

| **Table 4.4: The Result of Performance Metrics for Eclipse Dataset using C4.5 for Prediction (with HSA)** | | | | | |
|---|---|---|---|---|---|
| **Classifier** | **Evaluation Metrics** | | | | |
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| C4.5 | 78.09 | 0.8600 | 0.8800 | 0.8700 | 24.44 |

The description of Table 4.4, using C4.5 classifier with the five evaluation metrics without selecting discriminant feature before prediction are as follows:

The accuracy is 78.09%,

The precision 0.8600,

Recall is 0.8800,

F1-Score is 0.8700 and;

Classification Time (s) is 24.44

| **Table 4.5: The Result of Performance Metrics for Eclipse Dataset using KNN for Prediction (without HSA)** | | | | | |
|---|---|---|---|---|---|
| **Classifier** | **Evaluation Metrics** | | | | |
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| KNN | 84.85 | 0.8600 | 0.9800 | 0.9200 | 24.56 |

The description of Table 4.5, using KNN classifier with the five evaluation metrics without selecting discriminant feature before prediction are as follows:

The accuracy is 84.85%,

The precision 0.8600,

Recall is 0.9800,

F1-Score is 0.9200 and;

Classification Time (s) is 24.56

| **Table 4.6: The Result of Performance Metrics for Eclipse Dataset using KNN for Prediction (with HSA)** | | | | | |
|---|---|---|---|---|---|
| **Classifier** | **Evaluation Metrics** | | | | |
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| KNN | 83.11 | 0.8400 | 0.9900 | 0.9100 | 24.18 |

The description of Table 4.6, using KNN classifier with the five evaluation metrics without selecting discriminant feature before prediction are as follows:

The accuracy is 83.11%,

The precision 0.8400,

Recall is 0.9900,

F1-Score is 0.9100 and;

Classification Time (s) is 24.18

| **Table 4.7: The Result of Performance Metrics for Eclipse Dataset using NB for Prediction (without HSA)** | | | | | |
|---|---|---|---|---|---|
| **Classifier** | **Evaluation Metrics** | | | | |
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| NB | 81.35 | 0.8600 | 0.9300 | 0.8900 | 24.68 |

The description of Table 4.7, using NB classifier with the five evaluation metrics without selecting discriminant feature before prediction are as follows:

The accuracy is 81.35%,

The precision 0.8600,

Recall is 0.9300,

F1-Score is 0.8900 and;

Classification Time (s) is 24.68

| **Table 4.8: The Result of Performance Metrics for Eclipse Dataset using NB for Prediction (with HSA)** | | | | | |
|---|---|---|---|---|---|
| **Classifier** | **Evaluation Metrics** | | | | |
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| NB | 81.56 | 0.8800 | 0.9100 | 0.8900 | 25.48 |

The description of Table 4.8, using NB classifier with the five evaluation metrics without selecting discriminant feature before prediction are as follows:

The accuracy is 81.56%,

The precision 0.8800,

Recall is 0.9100,

F1-Score is 0.8900 and;

Classification Time (s) is 25.48

| **Table 4.9: The Result of Performance Metrics for Eclipse Dataset using SVM for Prediction (without HSA)** | | | | | |
|---|---|---|---|---|---|
| **Classifier** | **Evaluation Metrics** | | | | |
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| SVM | 83.22 | 0.8500 | 1.000 | 0.9200 | 24.70 |

The description of Table 4.9, using ANN classifier with the five evaluation metrics without selecting discriminant feature before prediction are as follows:

The accuracy is 82.22%,

The precision 0.8500,

Recall is 1.000,

F1-Score is 0.9200 and;

Classification Time (s) is 24.70

| **Table 4.10: The Result of Performance Metrics for Eclipse Dataset using SVM for Prediction (with HSA)** | | | | | |
|---|---|---|---|---|---|
| **Classifier** | **Evaluation Metrics** | | | | |
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| SVM | 86.44 | 0.8600 | 1.000 | 0.9300 | 25.52 |

The description of Table 4.10, using ANN classifier with the five evaluation metrics without selecting discriminant feature before prediction are as follows:

The accuracy is 86.44%,

The precision 0.8600,

Recall is 1.000,

F1-Score is 0.9300 and;

Classification Time (s) is 25.52

| **Table 4.11: The Summary of the Result when Relevant Features were selected for Prediction** | | | | | |
|---|---|---|---|---|---|
| **Classifier** | **Evaluation Metrics** | | | | |
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| ANN | 85.97 | 0.8600 | 1.000 | 0.9200 | 24.09 |
| C4.5 | 77.78 | 0.8800 | 0.8700 | 0.9000 | 25.04 |
| KNN | 83.11 | 0.8400 | 0.9900 | 0.9100 | 25.18 |
| NB | 81.56 | 0.8800 | 0.9100 | 0.8900 | 25.48 |
| SVM | 86.44 | 0.8600 | 1.000 | 0.9300 | 25.52 |

From Table 4.11,

ANN has the lowest classification of 24.09s,

SVM has the best accuracy of 86.44%,

C4.5 and NB have the highest precision of 0.8800,

ANN and SVM obtained the best recall of 1.000 and;

SVM has the best F1-score value of 0.9300

| **Table 4.12: The Summary of the Results without selecting Relevant Features for Prediction** | | | | | |
|---|---|---|---|---|---|
| **Classifier** | **Evaluation Metrics** | | | | |
| | Accuracy % | Precision | Recall | F1-Score | Classification Time (s) |
| ANN | 89.16 | 0.8900 | 1.000 | 0.9400 | 24.14 |
| C4.5 | 78.09 | 0.8600 | 0.8800 | 0.8700 | 24.44 |
| KNN | 84.85 | 0.8600 | 0.9800 | 0.9200 | 24.56 |
| NB | 81.35 | 0.8600 | 0.9300 | 0.8900 | 24.68 |
| SVM | 83.22 | 0.8500 | 1.000 | 0.9200 | 24.70 |

From Table 4.12,

ANN has the lowest classification of 24.14s,

SVM has the best accuracy of 89.16%,

KNN has the highest precision of 0.8900,

ANN and SVM obtained the best recall of 1.000 and;

SVM has the best F1-score value of 0.9400

## 4.2 Discussion of Findings

In the field of software defect prediction, defect prediction by feature selection plays a significant role in assisting with accurate and real-time prediction. Applying the intrinsic dimension for classification can produce quick results, however using a high dimensionality component or a huge dataset for classification can be a time-consuming process that leads to misclassification. According to the findings of this study, employing a subset of the dataset for defect prediction saves prediction time while greatly improving prediction accuracy. The feature selection constrains the amount of resources allocated to SDP, which is especially important for the SFP process. The results of this study indicate that FS is important in defect categorization. Since the study used a metaheuristic optimization technique for feature selection, they are equally informative and perform admirably[1].

The fault prediction process can be made more efficient and cost-effective by employing feature selection[1]. When compared to using a large dimensionality dataset, the pertinent feature generated in this study by using HSA to select the subset dataset delivers better, timelier, and more accurate results for defect classification. The SFP process becomes most effective and accurate when defects are classified using a dataset subset.

This study also looked into the performance of various types of MLAs for classification and found that ANN had the quickest classification time of 24.09s. Furthermore, the performance of each of the LAs is evaluated. The focus of the research is on feature selection for defect prediction. The study categorized and compared the outcomes of selected features, classification, and the results of 5 metrics for evaluation, although the work has limitations that can be examined further to improve and extend the findings. The

experiments, for example, were carried out using Eclipse datasets. Other text types, such as graphics, can be accommodated by developing using a single model. Furthermore, more datasets can be obtained for subsequent experimentation, and the findings may improve to some degree. Examining the dataset with more commonly used ML ensembles can help improve the research's output.

## 4.3 Comparative Analysis Results of Software Defect Prediction

The comparative analysis of SDP was conducted using different machine learning algorithms, one optimization algorithm known as HAS and Eclipse dataset. The results were provided using the following Figures.

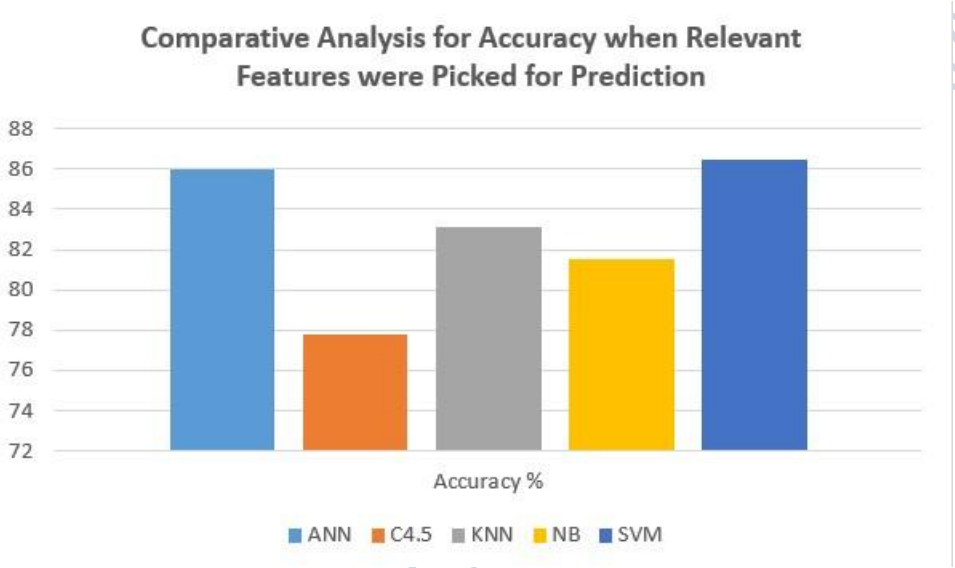



Figure 4.2: **Comparative Analysis for Accuracy when Relevant Features were Selected for Prediction**

The Figure 4.2 shows that the highest accuracy of 85.97% was obtained in ANN when relevant features were not selected for prediction.

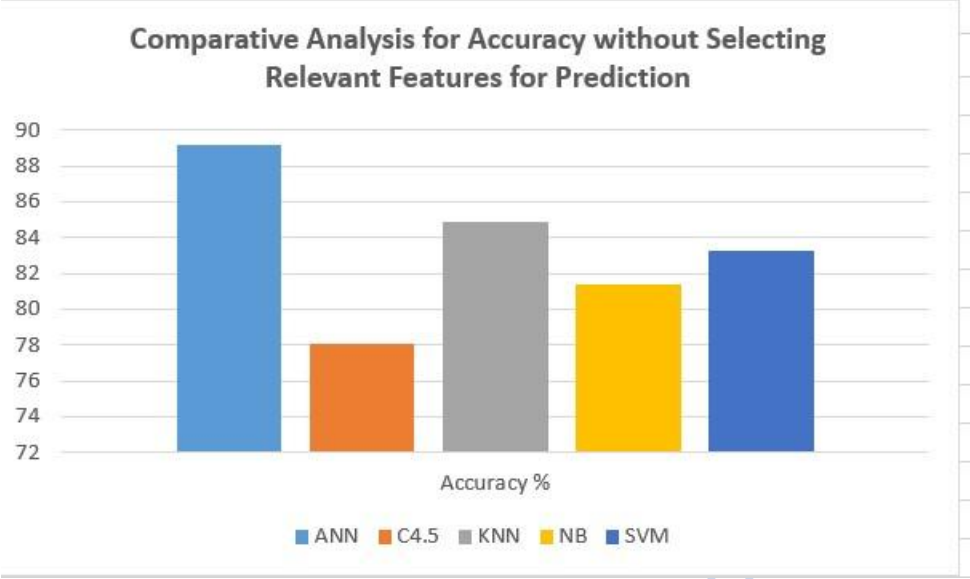



Figure 4.3: **Comparative Analysis for Accuracy when Relevant Features were not Selected for Prediction**

The Figure 4.3: shows that the highest accuracy of 89.16% was obtained in ANN when relevant features were not selected for prediction.

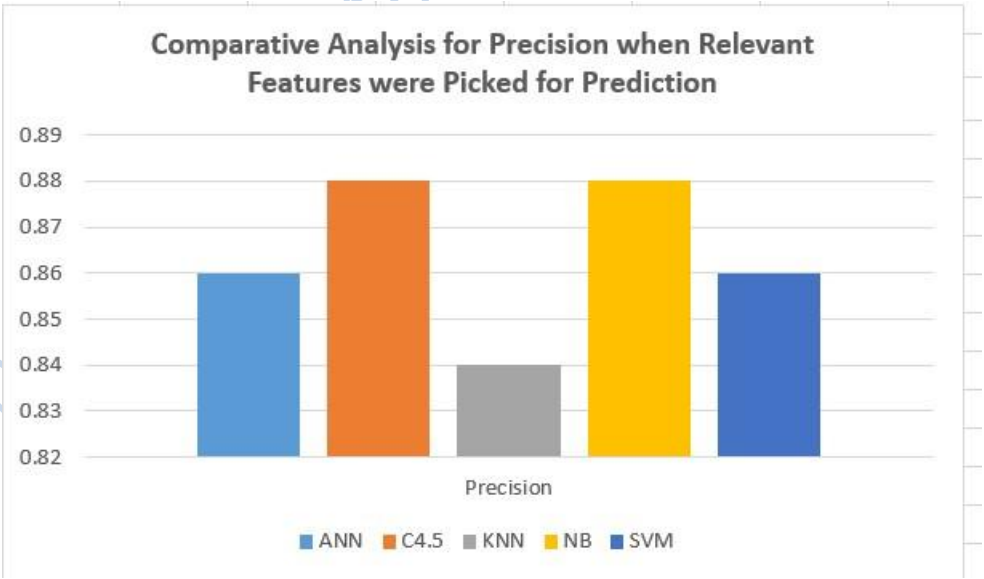



Figure 4.4: **Comparative Analysis for Precision when Relevant Features were Selected for Prediction**

The Figure 4.4 shows that the highest precision of 0.8800 was obtained in C4.5 and NB when relevant features were selected for prediction.

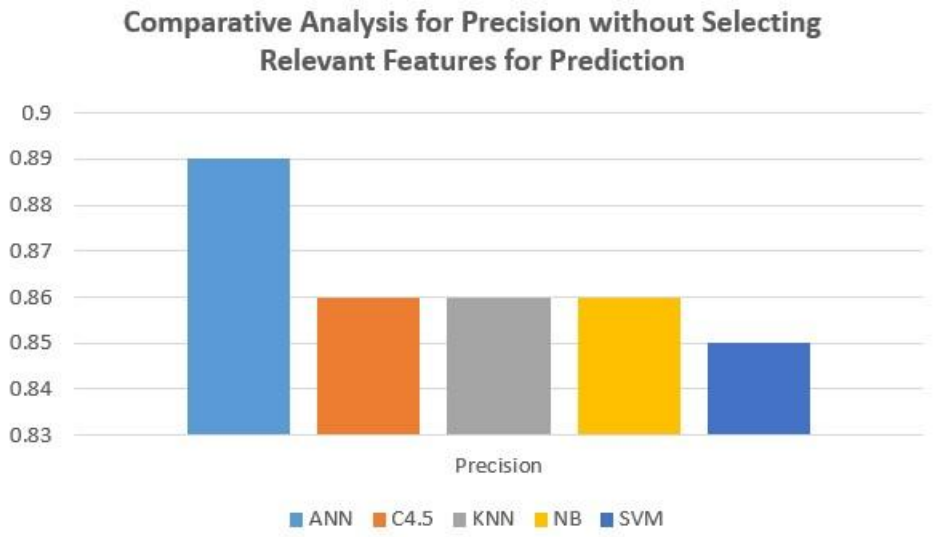



Figure 4.5: **Comparative Analysis for Precision when Relevant Features were not Selected for Prediction**

The Figure 4.5 shows that the highest precision of 0.8900 was obtained in ANN when relevant features were selected for prediction.

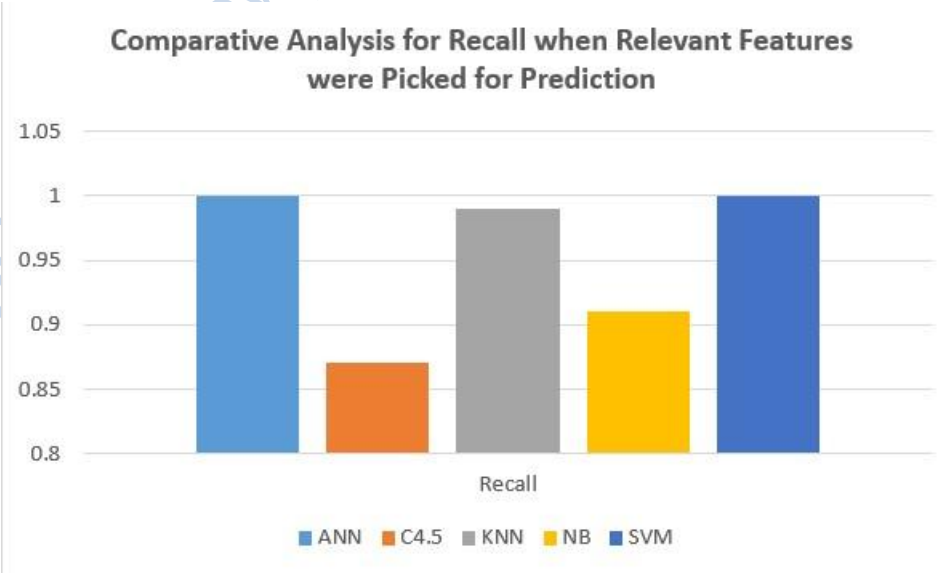


Figure 4.6: **Comparative Analysis for Recall when Relevant Features were Selected for Prediction**

The Figure 4.6 shows that the best Recall of 1.0000 was obtained in ANN when relevant features were not selected for prediction.

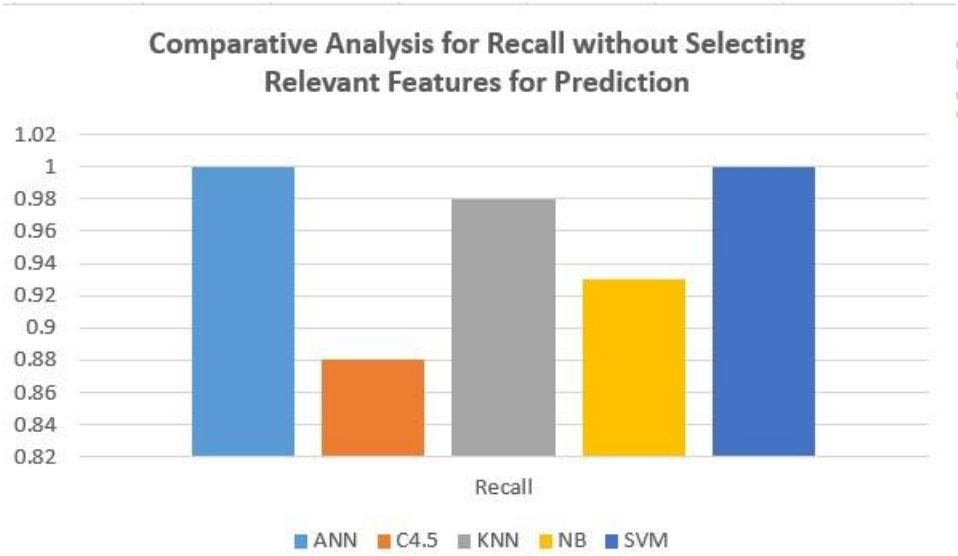



Figure 4.7: **Comparative Analysis for Recall when Relevant Features were not Picked for Prediction**

The Figure 4.7 shows that the best Recall of 1.0000 was obtained in ANN and SVM when relevant features were selected for prediction.

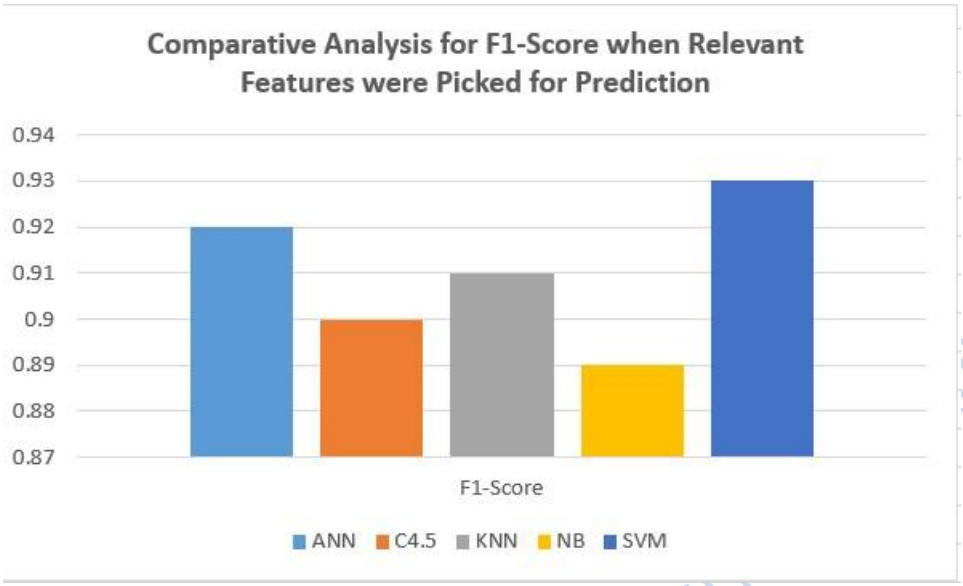



Figure 4.8: **Comparative Analysis for F1-Score when Relevant Features were Selected for Prediction**

From Figure 4.8, the best F1-Score of 0.9300 was obtained in SVM without selecting relevant feature for prediction.

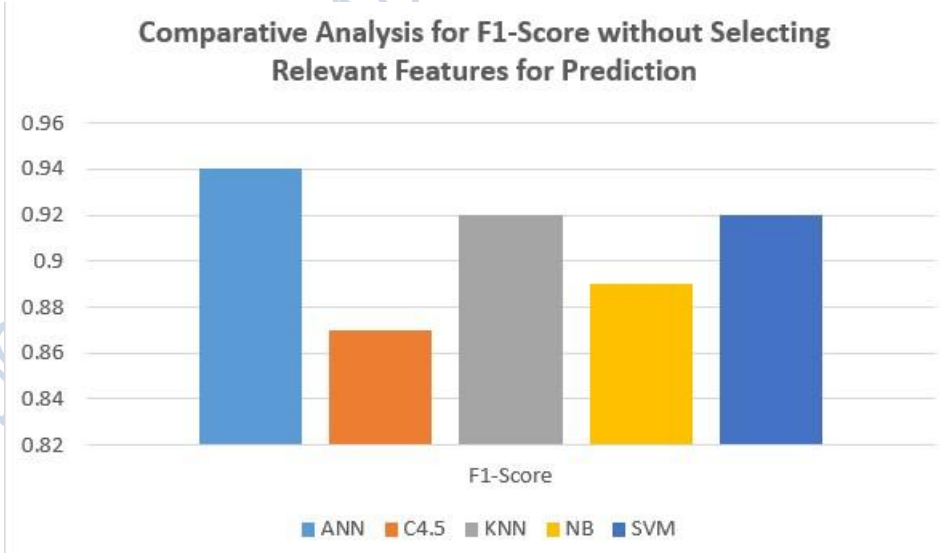



Figure 4.9: **Comparative Analysis for F1-Score when Relevant Features were not Selected for Prediction**

The Figure 4.9 revealed that the best F1-Score of 0.9400 was obtained in ANN when the relevant feature were selected for prediction.

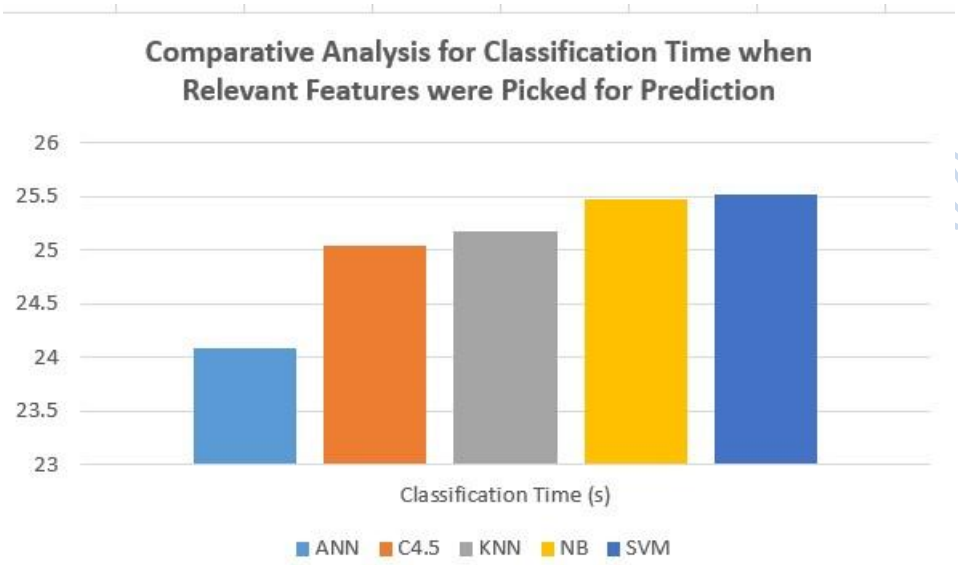



Figure 4.10: **Comparative Analysis for Classification Time when Relevant Features were Picked for Prediction**

From Figure 4.10, when relevant features were selected for prediction, the lowest classification time of 24.09s was obtained in ANN which shows that ANN outperformed other classifiers in term of real time interval for prediction.

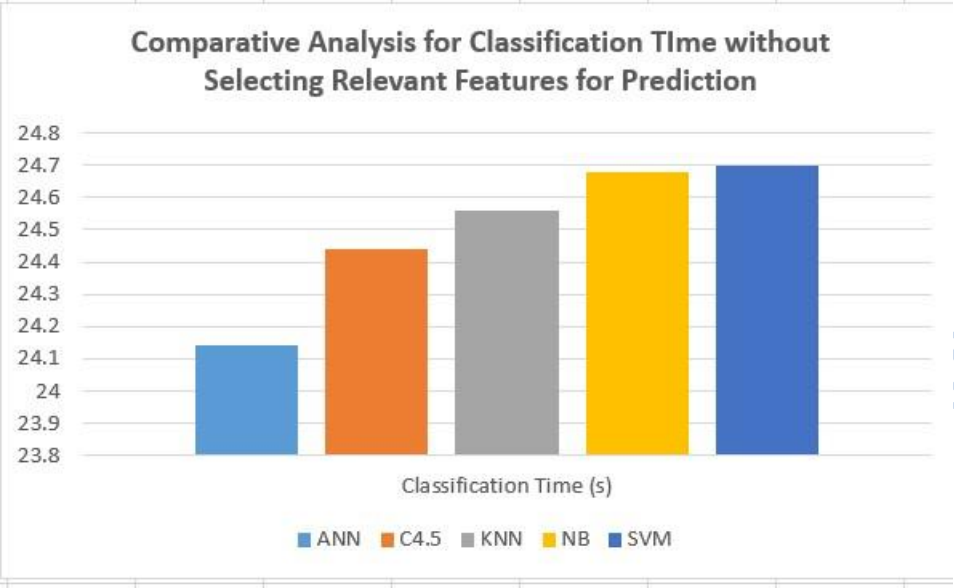



Figure 4.11: **Comparative Analysis for Classification Time when Relevant Features were not Selected for Prediction**

From Figure 4.11, the lowest classification time of 24.14s was obtained in ANN without selecting relevant features for prediction.

## Endnote

1. S. Velliangiria, S. Alagumuthukrishnanb & I. S. Thankumar "*A review of dimensionality reduction techniques for efficient computation"* Volume 165, 2019, 104-111.

# Chapter Five
# Conclusion

## 5.1 Summary of Findings

Feature selection has been considered by researchers to be one of the predominant stages in the SDP system. Feature selection involves a process of removing redundancy from the dataset for the purpose of presenting the relevant features for classification. Representation of feature produces an approximation to the original feature in fewer dimensions, while still maintaining the same structure of original features. This study applied an optimization algorithm and some selected machine learning algorithms. An optimization algorithm known as HSA was introduced to select important features before the classification phase. The data were pre-processed to remove redundant features using the python function from the different datasets employed in this study. After the preprocessing phase, the preprocessed data was passed into stochastic gradient descent to obtain discriminant features using five classifiers: SVM, NB, C4.5, ANN, and KNN. Publicly available online dataset known as Eclipse were used to validate the SDP system.

The experimental results were shown for the dataset with different evaluation parameters for software defect classification systems.

The experimental result when the relevant features were picked for prediction are as follows:

ANN has the lowest classification of 24.09s,

SVM has the best accuracy of 86.44%,

C4.5 and NB have the highest precision of 0.8800,

ANN and SVM obtained the best recall of 1.000 and;

SVM has the best F1-score value of 0.9300

The study concluded that the application of the HSA resulted in a slight improvement in some aspects of the software defect classification.

### 5.2. Conclusion

In the past few decades, academics and industry experts have focused their attention on software faults, a key area of software development. The necessity to develop testing methods through which the dependability, dependability, and effectiveness of the program may be confirmed and established grows as the software develops. It is admirable to try to foresee software flaws or, even better, to create software that is error-free, but there are other considerations that must be given top priority, including time and an accurate prediction rate. When a large dataset is utilized for classification, lengthy prediction processing is especially likely to result in misclassification.

In order to lower the large feature dimensionality that could result in a longer classification time and misclassification, a meta-heuristic optimization approach for feature selection (FS) was used in this study. Five learning algorithms—SVM, ANN, KNN, NB, and C4.5—were used for classification, with the HSA being used for feature selection. In addition, predictions were made using Eclipse software defect dataset. Also, evaluation measures like precision, accuracy, recall, classification time, and F1 score, the model's operational output was attained.

The ANN method beat other classifiers used for defect classification, as evidenced by the recorded results with HSA, which showed that it attained the lowest classification time of 24.09s in the Eclipse Dataset. KNN achieved the greatest accuracy of 86.44 percent, demonstrating that it performed better than other learning algorithms used for prediction in terms of correctness.

### 5.3. Recommendation

There are tendencies for prolonged processing of the dataset and misclassification using the whole dataset for defect prediction. Hence, applying a feature selection technique reduces the risk of prolonged data processing and misclassification. The defect classification model enhances the achievement of quality and reliable software as well as improves customer satisfaction. For an efficient defect prediction, it is strongly recommended to apply feature selection to pick the subset of the discriminant feature.

### 5.4 Contribution to Knowledge

The impact or contribution of this study includes:

1. To begin, this study identified HSA to be an effective Feature Selection.
2. After that, this study also presented an active model for SDP using an optimization algorithm (HSA), classifiers (SVM, ANN, KNN, C4.5, & NB), and metrics (precision, recall, classification time, F1-score, and accuracy)
3. Artificial Neural Network has the lowest classification time of 24.09s

4. HSA was found to be very efficient in dimensionality reduction through feature selection which reduced prolonged data processing, classification time, and misclassification.
5. Finally, the results of our experiments show that selecting relevant features for classification reduces prediction time.

## 5.5 Suggestions for Further Studies

The following recommendations are listed to draw the attention of future studies in SDP systems:

1. Developing a single model that can be used to select features and predict defects of text, images, and others
2. In the future, to further enrich and broaden the scope of the research, we may include more optimization algorithms, MLAs, and diversified software defect datasets in the testing and experimentation process.
3. Furthermore, we may compare the performance of the new ensemble techniques results with this study's.

## Bibliography

### Conference Proceedings


Catolino, Gemma; DI Nucci, Dario; Ferrucci, Filomena. *"Cross-project just-in-time bug prediction for mobile apps: an empirical assessment."* 2019

Fagundes R. A., Souza R. M. & Cysneiros F. J. *"Zero-in ated prediction model in software-fault data,"* IET Software, vol. 10, no. 1, 2016, 1-9.

Ruchika M., Nishant N., Gurha S. & Rathi V. *"Application of particle swarm optimization for software defect prediction using object oriented metrics."* **In 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)**, **IEEE**, 2021, 88-93.

Rizwan M., A. Nadeem, & Sindhu M. A. "*Empirical evaluation of coupling metrics in software fault prediction,"* 2020, 434-440.

Salma Ghoneim *"Accuracy, Recall, Precision, F-Score & Specificity, which to optimize on?"* a Conference Paper is available at https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124/. 2022

Wenbin B. & *Yu* F. *"A feature selection framework for software defect prediction using ISFLA."* **In IOP Conference Series: Materials Science and Engineering,** vol. 677, no. 5, 2019, 052121.


### E-Book


Olatunji B. L., Olabiyisi S. O., Oyeleye C. A., Sanusi B. A., Olowoye A. O., & Ofem O. A., *"Development of software defect prediction system using artificial neural network"* available at https://pdfs.semanticscholar.org/6047/c62f7d169b133e6389d6840eaca0201e140c.pdf/.


### Journals


Abdullah A. & Khan M. Z. *"Software defect prediction using supervised machine learning and ensemble techniques: a comparative study."* **Journal of Software Engineering and Applications** 12, no. 5, 2019, 85-100.

Abdelhamid, N., & Abdel-jaber, H. *"Learning comparison based on models content and features."* **IEEE**, 2017, 72–77

Anusha P. R., Joshi G. & Rane S. "*Quality and reliability studies in software defect management: a literature review*." **International Journal of Quality & Reliability Management**. 2021

Anonymous *"How many software developers are in the us and the world?* [Updated]" Available at https://www.daxx.com/blog/development-trends/number-software-developers-world/. 2021

Chauhan, A., & Kumar, R. *"Bug severity classification using semantic feature with convolution neural network computing in engineering and technology"*, 2020, 327-335.

Ebiaredoh-Mienye S. A., Swart T. G., Esenogho E. & Mienye I. D. "*A machine learning method with filter-based feature selection for improved prediction of chronic kidney disease."* **Bioengineering** 9, no. 8, 2022, 350.

Elsabagh M. A., Farhan M. S. & Gafar M. G. *"Cross-projects SDP using spotted hyena optimizer algorithm."* 2022 Available at https://link.springer.com/article/10.1007/s42452-020-2320-4/

Goyal S. *"Effective software defect prediction using support vector machines (SVMs)."* **International Journal of System Assurance Engineering and Management** 13, no. 2, 2022, 681-696.

Hussain, N., & Bixin, L. "*Performance comparison of ML classifiers in SDP Abstract:" IOSR Journal of Computer Engineering (IOSR-JCE),* 22(5), 2020. Available at https://doi.org/10.9790/0661-2205034858/

Imran, Z. *"Predicting bug severity in open-source software systems using scalable machine learning techniques."* **ISTQB**. 2019: available at https://www.istqb.org/.

Jayanthi R. & Florence L. *"Software defect prediction techniques using metrics based on neural network classifier."* **Cluster Computing** 22, no. 1 2019, 77-88.

Jindal, R., Malhotra, R., & Jain, A. "Prediction of defect severity by mining software project reports.", 8(2), 2017, 334-351.

Kang, J, Duksan R & Jongmoon B. *"Predicting just-in-time software defects to reduce post-release quality costs in the maritime industry*." ***Software: Practice and Experience*** 51, no. 4 2021, 748-771.

Kechit Goyal, *"Data preprocessing in machine learning: 7 easy steps to follow"*, available at https://www.upgrad.com/blog/data-preprocessing-in-machine-learning/. 2022

Kumari, M., Sharma, M., & Singh, V. *"Severity assessment of a reported bug by considering its uncertainty and irregular state."* 2018, 20-46.

Kiran Kumar B., Dr. Jayadev Gyani, Dr. Narsimha G. *"Software defect prediction using ant colony optimization"* 2018

Manzura J., Akbulut A., Catal C. & Mishra A. *"Machine learning-based software defect prediction for mobile applications: A systematic literature review."* **Sensors** 22, no. 7, 2022.

Mishbahulhuda, A. *"Data mining techniques in software defect prediction".* 7(3), 2017, 301–303.

Nasir, M., Sadollah, A., Yoon, J. H., & Geem, Z. W. *"Comparative study of harmony search algorithm and its applications in China, Japan and Korea."* **Applied Sciences** (Switzerland), 10(11), 2020, 1–26.

Nevendra M. & Pradeep S. *"A survey of software defect prediction based on deep learning."* **Archives of Computational Methods in Engineering** 2022, 1-26.

Nilanjan D., Chaki J., Moraru L., Fong S. & Yang X. "*Firefly algorithm and its variants in digital image processing: A comprehensive review." Applications of firefly algorithm and its variants*, 2020, 1-28.

Ni C., W.-S. Liu, X. Chen, Q. Gu, D.-X. Chen, & Q.-G. Huang. *"A cluster-based feature selection method for cross-project software defect prediction,"* **Journal of Computer Science and Technology**, vol. 32, no. 6, 2017, 1090–1107.

Alqasem O. & Akour M., *"Software fault prediction using deep learning algorithms"* **International Journal of Open Source Software and Processes.** 2019

Pachouly, J., Swati A., Ketan K., Ganeshsree S. & Ajith A. *"A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools."* **Engineering Applications of Artificial Intelligence** 2022, 111.

Pandit, M., Deepali G., Divya A., Nitin G., Hani M. A., Arturo O. M., Seifedine K. & Arun K. *"Towards design and feasibility analysis of DePaaS: AI based global unified software defect prediction framework."* **Applied Sciences** 12, no. 1, 2022, 493.

Punitha, S., Amuthan, A., & Joseph, K. S. *"Enhanced monarchy butterfly optimization technique for effective breast cancer diagnosis."* **Journal of Medical Systems**, 43(206), 2019, 1–14.

Rao S. V. *"An artificial neural network genetic algorithm with shuffled frog leap algorithm for software defect prediction."*, 2020, 831–836.

Rathore S. S. & Kumar S. "*Towards an ensemble-based system for predicting the number of software faults,*" **Expert Systems with Applications**, vol. 82, 2017, 357-382.

Rathore S. S. & Kumar S. *"Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems,*" **Knowledge-Based Systems,** vol. 119, 2017, 232-256.

Ritthipakdee, A., Thammano, A., Premasathian, N., & Jitkongchuen, D. "*Firefly mating algorithm for continuous optimization problems."* 2017, 1–11.

Sanusi, B. A., Olabiyisi, S. O., Olowoye, A. O., & Olatunji, B. L. *"Software defect prediction system using machine learning-based algorithms."* **Journal of Advances in Computational Intelligence Theory**, 1(3), 2019, 1–9.

Shaik, S., & Tech, M. *"A noval approach to iris recognition based on feature level fusion using."* **International Journal of Scientific Development and Research (IJSDR)**, 2(7), 2017, 287–292.

Shreya Bose, *"Defect management in software testing."* 2020: Available at https://www.browserstack.com/guide/defect-management-in-software-testing/

Sushant K. P., Ravi B. M., & Anil K. T. *"Software bug prediction prototype using bayesian network classifier: A Comprehensive Model*, Vol. 132, 2018, 1412-1421. https://doi.org/10.1016/j.procs.2018.05.071/

Srinivasan Balan, "*Metaheuristics in optimization: algorithmic perspective",* last modified May 2022, https://www.informs.org/Publications/OR-MS-Tomorrow/Metaheuristics-in-Optimization-Algorithmic-Perspective/

Tantithamthavorn G. & Chakkrit H., "*The impact of automated parameter optimization on defect prediction models.* **IEEE Transactions on Software Engineering**," 2018

Zahid U., Naqvi S. R., Farooq W., Yang H., Wang S. & Dai-Viet N. V. *"A comparative study of machine learning methods for bio-oil yield prediction–A genetic algorithm-based features selection."* **Bioresource Technology**, 2021, 335

Wang D, Tan D, Liu L. *"Particle swarm optimization algorithm: an overview."* 2018, 387–408.

Yang, G., Min, K., Lee, J.-W., & Lee, B. *"Applying topic modeling and similarity for predicting bug severity in cross projects."* TIIS, 13(3), 2019, 1583-1598.

Youm, K. C., Ahn, J., & Lee, E. *"Improved bug localization based on code change histories and bug reports."* **Information and Software Technology**, 82, 2017, 177–192. https://doi.org/10.1016/j.infsof.2016.11.002/

Zhou, C., Peng H., Cheng Z. & Ju M. *"Software defect prediction with semantic and structural information of codes based on Graph Neural Networks."* **Information and Software Technology**, 2022, 152**.**

**Magazines**

Tang J., Alelyani S., & Liu H. "Feature selection for classification: a review," **Data Classification: Algorithms and Applications, CRC Press**, 2014, 37-64.

# Appendix

```
import glob
import os
import time
from tkinter import *
import tkinter as tk
from tkinter import filedialog
from tkinter.ttk import Progressbar
from tkinter import messagebox
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from joblib import dump
from tkinter import messagebox
from harmony_search.harmony_run import harmonyRun
root = tk.Tk()
root.geometry("500x320")
root.title("Improved Feature Selection for Software Defect Prdiction Using Harmony Search
Algorithm")
root.eval('tk::PlaceWindow . center')
root.resizable(False, False)
frame_uploader = tk.Frame(root, width=900, height=120, background="teal")
frame_image = tk.Frame(root, width=900, height=200, background="blue")
frame_show_result = tk.Frame(root, width=900, height=200, background="teal")


from classification import ann
from classification import c45
from classification import knn
from classification import nb
from classification import SVM
```

```
import math
tk.Label(frame_uploader, text="Improved Feature Selection for Software Defect Prdiction Using
Harmony Search Algorithm", font=('helvetica', 9, 'bold')).pack()
entrySelectFolder = tk.Entry(frame_uploader, width = 25,)
entrySelectFolder.pack(anchor=NW, side=LEFT, pady=20, padx=20)
listOfDir = []
def displayBrowser():

   global listOfDir
   filename = filedialog.askdirectory(initialdir="/",
                              title='Select Folder directory')  #
askdirectory(parent=main2,initialdir="/",title='Please select a directory')
   ext = ['csv']  # Add image formats here

   files = []
   [files.extend(glob.glob(filename + '/*.' + e)) for e in ext]
   listOfDir = [file for file in files]
   global nameOfFile
   entrySelectFolder.insert(INSERT, filename)
   entrySelectFolder.pack(anchor=NW, side=LEFT, pady=20, padx=20)
   if listOfDir != []:
      i = 0
   for conver in listOfDir:
      i += 1
      base11 = os.path.basename(conver)
      root.after(100, fileList.insert(i, base11))
def startProcess():

   global listOfDir
   global path_
```

```
i = 0
for filename in listOfDir:
    fileNam = ''
    base11 = os.path.basename(filename)
    datasetName = base11.split('.')[0]

    dataSeteName = datasetName  #
    fullPath = '../generate/' + dataSeteName
    try:
        if not os.path.exists(fullPath):
            os.makedirs(fullPath)

    except OSError:
        print('Error: Creating Terms')

    columns_to_retain = ["cbo", "dit", "fanIn", "fanOut", "lcom", "noc", "numberOfAttributes",
                         "numberOfAttributesInherited", "numberOfLinesOfCode", "numberOfMethods",
                         "numberOfMethodsInherited", "numberOfPrivateAttributes",
"numberOfPrivateMethods",
                         "numberOfPublicAttributes", "numberOfPublicMethods", "rfc", "wmc", "bugs"]

    df = pd.read_csv(filename)
    df.rename(columns=str.lower).head()
    df = df.drop([col for col in df.columns if not col in columns_to_retain], axis=1)

    df.rename(columns={"bugs": "classification"}, inplace=True)

    df.drop_duplicates(keep=False, inplace=True)
    a = np.array(df['classification'].values.tolist())
    df['classification'] = np.where(a > 1, 1, a).tolist()
    X = df.drop(["classification"], axis=1)
```

```
y = df["classification"]

appendSelected = 0

title = 'SVM_Normal'
svm.runSvm(X, y, appendSelected, fullPath, title)
# root.after(100, fileListProcessed.insert(i, dataSeteName+' <==> SVM Normal'))

title = 'ANN_Normal'
ann.ann(X, y, appendSelected, fullPath, title)

title = 'C45_Normal'
c45.c45(X, y, appendSelected, fullPath, title)

title = 'KNN_Normal'
knn.runKnn(X, y, appendSelected, fullPath, title)



title = 'NB_Normal'
nb.runNb(X, y, appendSelected, fullPath, title)
df1 = X.rename(columns=df.iloc[0]).drop(df.index[0])
df2 = y.rename(columns=df.iloc[0]).drop(df.index[0])

appendSelected = harmonyRun(df1, df2, X)
df1 = pd.read_csv(filename)

df1.rename(columns=str.lower).head()

df1 = df1.drop([col for col in df1.columns if not col in columns_to_retain], axis=1)
df1.rename(columns={"bugs": "classification"}, inplace=True)
```

```
    a = np.array(df1['classification'].values.tolist())
    df1['classification'] = np.where(a > 1, 1, a).tolist()

    appendSelectedData = []
    iij = 0
    for clu in columns_to_retain:
       for cl2 in appendSelected:
          if iij == cl2:
             appendSelectedData.append(clu)
       iij += 1
    appendSelectedData.append('classification')
    # print(appendSelectedData)

    df1 = df1.drop([col for col in df1.columns if not col in appendSelectedData], axis=1)

    X = df1.drop(["classification"], axis=1)
    y = df1["classification"]

    f2 = open(fullPath + '/' + 'Harmony Search.txt', 'a+')
    f2.write("Selected Attribute by Harmony Search -> " + str(appendSelectedData) + "\n")

    f2.close()


##############################################################################
##########
    #This part is for Ordinary part

    title = 'SVM_With_HS'
    svm.runSvm(X, y, appendSelected, fullPath, title)
```

```
        title = 'ANN_With_HS'
        ann.ann(X, y, appendSelected, fullPath, title)

        title = 'C45_With_HS'
        c45.c45(X, y, appendSelected, fullPath, title)

        title = 'KNN_With_HS'
        knn.runKnn(X, y, appendSelected, fullPath, title)

        title = 'NB_With_HS'
        nb.runNb(X, y, appendSelected, fullPath, title)



    messagebox.showwarning('Info. Message', 'Successful Message...')


button1 = tk.Button(frame_uploader, text='Browse', command=displayBrowser, bg='brown',
fg='white', font=('helvetica', 9, 'bold')).place(x = 175,y = 38)

button1 = tk.Button(frame_uploader, text='Start Process', command=startProcess, bg='green',
fg='white', font=('helvetica', 9, 'bold')).place(x = 245,y = 38)
frame_uploader.pack()
frame_uploader.pack_propagate(0)

fileList = Listbox(frame_show_result, height=10, width=20)

fileList.insert(0, "Loaded File(s)")
fileList.pack(side=LEFT, pady=10, padx=20)
frame_show_result.pack()
frame_show_result.propagate(0)
```

```
root.mainloop()
from sklearn.model_selection import train_test_split
def runDatas(x, y):
   X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_state=6)

   return  X_train, X_test, y_train, y_test
```

**Harmony Search Algorithm**

```
import numpy as np
from harmony_search.HarmonyCore import HarmonyCore
class objective_function:

   def __init__(self,
            input_X,
            input_Y,
            iteration = 2,
            weight_decimal = 0,
            sample_size = -1,
            hmcr_proba = 0.7,
            par_proba = 0.3,
            adju_proba = 0.5,
            harmony_menmory_size = 50,
            up_down_limit = None):

      self.input_X = input_X
      self.input_Y = input_Y
      self.iteration = iteration
      self.weight_decimal = weight_decimal
      if sample_size == -1:
```

```
            self.sample_size = len(input_X[:0].columns)
        else:
            self.sample_size = sample_size
        self.hmcr_proba = hmcr_proba
        self.par_proba = par_proba
        self.adju_proba = adju_proba

        self.vector_size =  len(self.input_X[:0].columns)#17        self.harmony_menmory_size =
harmony_menmory_size
        self.up_down_limit = up_down_limit

    def fitness(self,weight,input_X,input_Y):
        e = 0.0
        weight = [float(i)/sum(weight) for i in weight]
        for x,y in zip(input_X,input_Y):
            #print('Compare',sum(np.multiply(x,weight)),y)
            e += sum(np.multiply(x,weight)).round(0) != y
        e /= np.array(input_X).shape[0]
        return e

def harmonyRun(input_X, input_Y, input_XX):
    up_down_limit = [[0, 1], [0, 1], [0, 1], [0, 1],[0, 1], [0, 1], [0, 1], [0, 1],[0, 1], [0, 1], [0, 1], [0, 1],[0,
1], [0, 1], [0, 1], [0, 1], [0,1]]

    objective_function_ = objective_function(input_X, input_Y,
sample_size=len(input_X[:0].columns), weight_decimal=2,
                                             up_down_limit=up_down_limit)

    hs = HarmonyCore(objective_function_)

    features = hs.run()
```

```
    return features
```

**import random**

```
import numpy as np
class HarmonyCore(object):

    def __init__(self,harmony_obj):
        self.obj_func = harmony_obj
        self.hmm_matrix = list()

        matrix = []
        for limit in self.obj_func.up_down_limit:
            row = np.random.uniform(low=limit[0], high=limit[1],
size=(1,self.obj_func.harmony_menmory_size))[0]
            matrix.append(row)
        matrix = np.asarray(matrix).transpose().round(self.obj_func.weight_decimal)
        self.hmm_matrix = matrix
        self.fp = [1] * (len(self.obj_func.input_X[:0].columns) - 1)
    def run(self):
        hmm_err_list = [0] * len(self.hmm_matrix)
        for m_i in range(len(self.hmm_matrix)):
            vetor_list = self.hmm_matrix[m_i]

            error = self.obj_func.fitness(vetor_list,self.obj_func.input_X,self.obj_func.input_Y)
            hmm_err_list[m_i] = error

        for itera in range(self.obj_func.iteration):
            vetor_list = [0] * self.obj_func.vector_size
            for i in range(self.obj_func.vector_size):
```

```
            if np.random.rand(1,)[0] < self.obj_func.hmcr_proba:
print(np.random.randint(self.obj_func.harmony_menmory_size, size=1)[0])
                new_vactor =
self.hmm_matrix[np.random.randint(self.obj_func.harmony_menmory_size, size=1)[0]][i]
                if np.random.rand(1,)[0] < self.obj_func.par_proba:
                    if np.random.rand(1,)[0] < self.obj_func.adju_proba:
                        new_vactor -= (new_vactor - self.obj_func.up_down_limit[i][0]) *
np.random.rand(1,)[0]
                    else:
                        new_vactor += (self.obj_func.up_down_limit[i][0] - new_vactor) *
np.random.rand(1,)[0]
                vetor_list[i] = round(new_vactor, self.obj_func.weight_decimal)
            else:
                new_vactor = np.random.uniform(low=self.obj_func.up_down_limit[i][0],
high=self.obj_func.up_down_limit[i][1], size=(1,))[0]
                vetor_list[i] = round(new_vactor, self.obj_func.weight_decimal)
        featureSetIndex = []
        for j in range(self.obj_func.sample_size - 1):
            decision = random.random()
            if decision < self.fp[j] / 2.0:
                featureSetIndex.append(1)
            else:
                featureSetIndex.append(0)
        features = [0]
        for ij, obj in enumerate(featureSetIndex):
            if obj:
                features.append(ij + 1)
        print(features)
np.random.permutation(len(self.obj_func.input_X))[:self.obj_func.sample_size]
```

```
print('HMCR:',self.obj_func.hmcr_proba,'PAR:',self.obj_func.par_proba,'HMM_UPDATE_NEW_ERR
OR:',error)
 hmm_err_list.index(min(hmm_err_list)),
      return features
```

**ANN Classifier**

```
import os
import time

import pandas as pd
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, mean_squared_error,
cohen_kappa_score
  def ann(X, y,  appendSelected, fullPath, title):
   start_time = time.time()

   from sklearn.model_selection import train_test_split
   X_Train, X_Test, y_Train, y_Test = train_test_split(X, y, test_size=0.40)

   from sklearn.neural_network import MLPClassifier
   clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=1)

   clf.fit(X_Train, y_Train)
   MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1,
            solver='lbfgs')

   predictions_NB = clf.predict(X_Test)

   cohen_score = cohen_kappa_score(y_Test, predictions_NB)
   f2 = open(fullPath+'/'+title+'.txt', 'a+')
```

```
f2.write("Confusion Matrix -> " + str( confusion_matrix(y_Test, predictions_NB) ) + "\n")
f2.write("Confusion Matrix -> " + str(classification_report(y_Test, predictions_NB)) + "\n")
mse = mean_squared_error(y_Test, predictions_NB)
f2.write("MSE -> " + str(mse) + "\n")
f2.write("Kappa Statistic -> " + str(cohen_score) + "\n")
accuracy = metrics.accuracy_score(y_Test, predictions_NB)
f2.write("Accuracy -> " + str(accuracy) + "\n")
end_time = time.time()
f2.write("Time -> " + str(end_time-start_time) + "\n")
f2.close()

sub = []
allSub = []
stringCall = ''
df = pd.DataFrame()

for ind, obj in enumerate(X_Test):
   # i = i + 1
   if stringCall == '':
      stringCall = obj
   elif stringCall != obj:
      stringCall = obj

   if stringCall == obj:
      for d in X_Test[obj]:
         sub.append(d)
   df[obj] = sub
   allSub.append(sub)
   sub = []

for i in range(len(y_Test)):
```

```
    if y_Test[y_Test.index[i]] == 0:
        sub.append('Non-Defected')
        ''
    else:
        sub.append('Defected')

        ''
df['classification'] = sub

df = pd.DataFrame(df)
df.to_csv(fullPath+'/'+ title+'.csv')
```

**C45 Classifier**

```
import time
import pandas as pd
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, mean_squared_error,
cohen_kappa_score
def c45(X, y,  appendSelected, fullPath, title):
    start_time = time.time()
    from sklearn.model_selection import train_test_split
    X_Train, X_Test, y_Train, y_Test = train_test_split(X, y, test_size=0.30)
    from sklearn.tree import DecisionTreeClassifier
    classifier = DecisionTreeClassifier()
    classifier.fit(X_Train, y_Train)
    predictions_NB = classifier.predict(X_Test)

    cohen_score = cohen_kappa_score(y_Test, predictions_NB)
    f2 = open(fullPath + '/' + title + '.txt', 'a+')
    f2.write("Confusion Matrix -> " + str(confusion_matrix(y_Test, predictions_NB)) + "\n")
    f2.write("Confusion Matrix -> " + str(classification_report(y_Test, predictions_NB)) + "\n")
```

```
mse = mean_squared_error(y_Test, predictions_NB)
f2.write("MSE -> " + str(mse) + "\n")
f2.write("Kappa Statistic -> " + str(cohen_score) + "\n")
accuracy = metrics.accuracy_score(y_Test, predictions_NB)
f2.write("Accuracy -> " + str(accuracy) + "\n")
end_time = time.time()
f2.write("Time -> " + str(end_time - start_time) + "\n")
f2.close()

sub = []
allSub = []
stringCall = ''
df = pd.DataFrame()

for ind, obj in enumerate(X_Test):
    if stringCall == '':
        stringCall = obj
    elif stringCall != obj:
        stringCall = obj

    if stringCall == obj:
        for d in X_Test[obj]:
            sub.append(d)
    df[obj] = sub
    allSub.append(sub)
    sub = []

for i in range(len(y_Test)):
    if y_Test[y_Test.index[i]] == 0:
        sub.append('Non-Defected')
        ''
```

```
        else:
            sub.append('Defected')
            ''
    df['classification'] = sub
    df = pd.DataFrame(df)
    df.to_csv(fullPath + '/' + title + '.csv')
```

**KNN Classifier**

```
import time
import pandas as pd
from sklearn import metrics
from sklearn.metrics import classification_report, mean_squared_error
def runKnn(X, y,  appendSelected, fullPath, title):
    start_time = time.time()

    from sklearn.model_selection import train_test_split
    X_Train, X_Test, y_Train, y_Test = train_test_split(X, y, test_size=0.30)

    from sklearn.neighbors import KNeighborsClassifier
    classifier = KNeighborsClassifier(n_neighbors = 4, metric = 'minkowski', p = 2)
    classifier.fit(X_Train, y_Train)

    y_pred = classifier.predict(X_Test)

    # Making the Confusion Matrix
    from sklearn.metrics import confusion_matrix, accuracy_score, cohen_kappa_score
    cohen_score = cohen_kappa_score(y_Test, y_pred)
    f2 = open(fullPath + '/' + title + '.txt', 'a+')
    f2.write("Confusion Matrix -> " + str(confusion_matrix(y_Test, y_pred)) + "\n")
    f2.write("Confusion Matrix -> " + str(classification_report(y_Test, y_pred)) + "\n")
    mse = mean_squared_error(y_Test, y_pred)
```

```
f2.write("MSE -> " + str(mse) + "\n")
f2.write("Kappa Statistic -> " + str(cohen_score) + "\n")
accuracy = metrics.accuracy_score(y_Test, y_pred)
f2.write("Accuracy -> " + str(accuracy) + "\n")
end_time = time.time()
f2.write("Time -> " + str(end_time - start_time) + "\n")
f2.close()

sub = []
allSub = []
stringCall = ''
df = pd.DataFrame()

for ind, obj in enumerate(X_Test):
    if stringCall == '':
        stringCall = obj
    elif stringCall != obj:
        stringCall = obj
        i = 0

    if stringCall == obj:
        for d in X_Test[obj]:
            sub.append(d)
    df[obj] = sub
    allSub.append(sub)
    sub = []

for i in range(len(y_Test)):
    if y_Test[y_Test.index[i]] == 0:
        sub.append('Non-Defected')
        ''
```

```
        else:
            sub.append('Defected')
            ''
    df['classification'] = sub

    df = pd.DataFrame(df)
    df.to_csv(fullPath + '/' + title + '.csv')
```

**NB Classifier**

```
import time
import pandas as pd
from sklearn import naive_bayes, metrics
from sklearn.metrics import cohen_kappa_score, confusion_matrix, classification_report,
mean_squared_error

def runNb(X, y,  appendSelected, fullPath, title):
    start_time = time.time()

    from sklearn.model_selection import train_test_split
    X_Train, X_Test, y_Train, y_Test = train_test_split(X, y, test_size=0.30)

    Naive = naive_bayes.GaussianNB()
    # MultinomialNB()
    Naive.fit(X_Train, y_Train)

    # predict the labels on validation dataset
    predictions_NB = Naive.predict(X_Test)
    cohen_score = cohen_kappa_score(y_Test, predictions_NB)

    f2 = open(fullPath + '/' + title + '.txt', 'a+')
    f2.write("Confusion Matrix -> " + str(confusion_matrix(y_Test, predictions_NB)) + "\n")
```

```
f2.write("Confusion Matrix -> " + str(classification_report(y_Test, predictions_NB)) + "\n")
mse = mean_squared_error(y_Test, predictions_NB)
f2.write("MSE -> " + str(mse) + "\n")
f2.write("Kappa Statistic -> " + str(cohen_score) + "\n")
accuracy = metrics.accuracy_score(y_Test, predictions_NB)
f2.write("Accuracy -> " + str(accuracy) + "\n")
end_time = time.time()
f2.write("Time -> " + str(end_time - start_time) + "\n")
f2.close()

sub = []
allSub = []
i = 0
stringCall = ''
df = pd.DataFrame()

for ind, obj in enumerate(X_Test):
    # i = i + 1
    if stringCall == '':
        stringCall = obj
    elif stringCall != obj:
        stringCall = obj
        i = 0

    if stringCall == obj:
        for d in X_Test[obj]:
            # print(d)
            sub.append(d)
    df[obj] = sub
    allSub.append(sub)
    sub = []
```

```
    for i in range(len(y_Test)):
        if y_Test[y_Test.index[i]] == 0:
            sub.append('Non-Defected')
            ''
        else:
            sub.append('Defected')
            ''
    df['classification'] = sub
    df = pd.DataFrame(df)
    df.to_csv(fullPath + '/' + title + '.csv')
```

**SVM Classifier**

```
import time
import pandas as pd
from sklearn import metrics
from sklearn.metrics import cohen_kappa_score, confusion_matrix, classification_report,
mean_squared_error
def runSvm(X, y,  appendSelected, fullPath, title):
    start_time = time.time()
    from sklearn.model_selection import train_test_split
    X_Train, X_Test, y_Train, y_Test = train_test_split(X, y, test_size=0.30)
    from sklearn.svm import SVC
    print('Here 1')
    clf = SVC(kernel='linear')
    print('Here 2')
    # fitting x samples and y classes
    clf.fit(X_Train, y_Train)
    print('Here 3')
    predictions_NB = clf.predict(X_Test)
    print('Here 4')
```

```
# cohen_score = cohen_kappa_score(y_Test, predictions_NB)

# from sklearn.svm import SVC
# svclassifier = SVC(kernel='rbf', C=1E10)
# svclassifier.fit(X_Train, y_Train)
# predictions_NB = svclassifier.predict(X_Test)
cohen_score = cohen_kappa_score(y_Test, predictions_NB)

f2 = open(fullPath + '/' + title + '.txt', 'a+')
f2.write("Confusion Matrix -> " + str(confusion_matrix(y_Test, predictions_NB)) + "\n")
f2.write("Confusion Matrix -> " + str(classification_report(y_Test, predictions_NB)) + "\n")
mse = mean_squared_error(y_Test, predictions_NB)
f2.write("MSE -> " + str(mse) + "\n")
f2.write("Kappa Statistic -> " + str(cohen_score) + "\n")
accuracy = metrics.accuracy_score(y_Test, predictions_NB)
f2.write("Accuracy -> " + str(accuracy) + "\n")
end_time = time.time()
f2.write("Time -> " + str(end_time - start_time) + "\n")
f2.close()
sub = []
allSub = []
stringCall = ''
df = pd.DataFrame()

for ind, obj in enumerate(X_Test):
   if stringCall == '':
      stringCall = obj
   elif stringCall != obj:
      stringCall = obj

   if stringCall == obj:
```

```
        for d in X_Test[obj]:
            sub.append(d)
    df[obj] = sub
    allSub.append(sub)
    sub = []

for i in range(len(y_Test)):
    if y_Test[y_Test.index[i]] == 0:
        sub.append('Non-Defected')
        ''
    else:
        sub.append('Defected')

        ''
df['classification'] = sub

df = pd.DataFrame(df)
df.to_csv(fullPath + '/' + title + '.csv')
```

## Bio-data

**Personal Information**

| | |
|---|---|
| **Surname**: | BAMISAYE |
| **Other Name:** | Folorunsho |
| **Date of Birth**: | 15th November 1983 |
| **Gender**: | Male |
| **Local Government Area**: | Ilejemeje |
| **State of Origin**: | Ekiti |
| **Nationality**: | Nigerian |
| **Marital Status**: | Married |
| **Religion**: | Christian |
| **Next of Kin**: | Mrs. Bamisaye Funmilayo Dorcas |

**Contacts**:

| | |
|---|---|
| **Residential Address**: | 39, Eluku Street, Itupate, Nearest Bus-Stop, Ikorodu Garage, Ikorodu, Lagos, Nigeria |
| **Postal Address**: | 39, Eluku Street, Itupate, Nearest Bus-Stop, Ikorodu Garage, Ikorodu, Lagos, Nigeria |
| **Email**: | bamisayefj@gmail.com |
| **Phone Number**: | 07038010622 |

**Educational Background with Dates**

| | |
|---|---|
| Lead City University, Ibadan, Oyo State | In view |
| **Federal Polytechnic, Offa Kwara State** | |
| Higher National Diploma (HND) | 2011 |
| **Federal Polytechnic, Offa Kwara State** | |
| Ordinary National Diploma (OND) | 2006 |
| **Secondary School** | |
| Iludun Community High School | 2002 |
| Osi Central Secondary School | 2003 |
| **Anglican Primary School** | |
| Saint James Anglican Primary School, Eda-Oniyo Ekiti | 1995 |

**Academic Qualifications and Certificate Obtained with Dates**

| | |
|---|---|
| M.Sc. in Computer Science | 2020-till date |
| HND in Computer Engineering | 2009-2011 |
| OND in Computer Engineering | 2003-2006 |
| National Examination Council | June, 2002 |
| First School Leaving Certificate | 1991-1996 |

**Work Experience**

**Thomas Adewumi University, Oko, Kwara State** **2020-Till Date**
**Position**: Director, ICT
**Responsibilities includes among others**

- Coordinating all ICT related activities
- Leading, mentoring and growing members of ICT team
- Ensuring compliance of Team with all rules and regulations of the institution
- TAU CBT Centre Administrator
- Design strategies to ensure efficient and ICT driven institution
- Portal Design and Management
- Steady implementation of the automation system for TAU running
- Webmaster
- Coordinating all technical admission related matters including, JAMB CAPS and University Portals manager among others
- Administering school, national and international exams
- Supporting all Admission related issues among others

**KolaDaisi University, Ibadan Oyo State** **2018-2020**
**Position**: Director, ICT
**Responsibilities includes among others**

- ✓ Coordinating all ICT related activities
- ✓ Web Manager
- ✓ Ensuring proper functioning of the University Portal
- ✓ Developing ICT solutions and facilitating the use of open software solutions that enhance the delivery of the core mandate of the University
- ✓ Supporting the cataloging and documentation of the University Library
- ✓ Managing all e-library related services
- ✓ Supporting teaching, research, administration, strategic planning, and development in Learning object with respect to ICT

**Joint Komputer Kompany (JKK)** **2014-2018**
**Position**: CBT Application Manager
**Responsibilities includes among others**

- Consultant to the University of Ilorin on Computer Based Test (CBT)
- Maintenance and troubleshooting of system hardware and software
- Setting up of CBT center for an individual, organization, and government parastatal
- Installation and configuration of Servers for CBT administration
- Ensured adequate internet connectivity for all the staff
- Managed over 25,000 pieces of Android Tablets deployed to the University of Ilorin
- Periodic workshop training for the University of Ilorin's lecturers and professional guidance on how to use CBT application software in preparing their questions
- Periodic training of the Unilorin Undergraduate students on how to take CBT exam
- Mentoring of the University of Ilorin's lecturers on the usage of the CBT Application software
- Staff training and mentoring
- Recruitment and training of ad hoc staff for Biometrics data capture
- Recruitment and training of ad hoc staff for CBT administration etc

**Electronic Test Company (TOEFL)** 2013-2014
Technical

- ETS Proctor
- Administration of TOEFL, GRE, etc exam
- Maintenance and troubleshooting of system hardware and software

**Electronic Test Company (eTC)** 2008-2010

**Position**: Head Technical Test Administration

- Setup and managed over 500 network computer systems for CBT administration
- Installation and configuration of Servers for CBT administration etc
- Installation and configuration of testing software and any other 3rd party applications used for computer-based testing such as Biometric Data capture software etc.
- Recruitment of ad-hoc staff for biometrics data capture
- Managed Biometrics data of universities students captured

**Joint Komputer Kompany (JKK)** 2007-2008
Internship

**NYSC** **2012 - 2013**
**Uhi Secondary School- Uhuhende Local Government; Edo State**

- Setup School Computer Lab
- Conducting Computer training for students

**Skills**

- Network administration
- IT Project Management
- Data processing with SPSS
- WordPress Development
- Cloud Computing
- Artificial Intelligence
- Machine Learning
- Software Engineering
- Graphics Designs
- Computer Hardware Maintenance / Repair
- GSM Mobile Phone Engineer

- Installation of server operating system
- Basic Knowledge in Web development (asp.net)
- Diploma in General Computer Studies
- Computer Based Test Engineer and Administrator
- Computer Literate
- Facilitator/trainer
- Digital Marketing Expert

**Professional Certifications**

| | | |
|---|---|---|
| i. | Digital Currency Trading Expert | 2021 |
| ii. | MikroTik Certified Network Associate (MTCNA) | 2020 |
| iii. | IT Essentials A+ CISCO | 2014 |
| iv. | Certified Project Manager IPMP | 2012 |

**Conferences/Seminars/Workshops attended with Dates**

| | | |
|---|---|---|
| **i.** | **The Importance of ICT in Enhancing Performance and Wholesomeness**<br>Thomas Adewumi University, Oko, Kwara State<br>**The Resource Person** | 2022 |
| **ii.** | **Understanding and conforming to the University Culture**<br>TAU, Oko, Kwara State | 2022 |
| iii. | **Drug Abuse and and Addiction**<br>TAU, Oko, Kwara State | 2022 |
| | Computer Appreciation for Faculty Officers & Lecturers, **TAU** | 2022 |
| iv. | Digital Literacy for Enhancing Productivity, **TAU** | 2021 |
| **v.** | **The Importance of ICT in the University**, TAU<br>Thomas Adewumi University, Oko, Kwara State<br>**The Resource Person** | 2021 |
| **vi.** | CBT Workshop Training for Unilorin Lecturers, **Unilorin**<br>**The Resource Person** | 2021 |
| vii. | Be Security Conscious, **TAU, Oko, Kwara State** | 2021 |

viii. Employee Welfarism in Tertiary Institutions, **KDU Oyo State** 2020

ix. Management and Administration of University, **KDU** 2019

x. Conference on Quality Control System in the Institution, NUC **Abuja** 2019

xi. The use of Electronic Board for teaching, **KDU** 2019

xii. Quality Assurance in University, **NUC Abuja** 2018

xiii. Advance Digital Appreciation Programme, **TAU** 2017

**xiv.** Question Authoring Workshop Training for Unilorin Lecturers, **Unilorin The Resource Person** 2017

**xv.** CBT Workshop Training for Unilorin students, **Unilorin The Resource Person** 2017

**xvi.** CBT Administration Workshop Training for Unilorin lecturers, **Unilorin The Resource Person** 2017

xvii. Capacity Building Training, **TAU, Oko-Irese, Kwara State** 2010

xviii. Ethical Values in Administration, **1, Oba-Akran, Ikeja Lagos** 2009

xix. Biometric Data Capture and Authentication **1, Oba-Akran, Ikeja Lagos** 2009

xx. Team Building Management, **Lekki, Lagos** 2008

xxi. The Need for Computer-Based Testing in Nigeria, **Oba-Akran Ikeja** 2008

xxii. Enhancing Staff Welfarism, **1, Oba-Akran, Ikeja Lagos** 2008

xxiii. Building Technological Capacity, **1, Oba-Akran, Ikeja Lagos** 2008

**Administrative Experience and Appointments**

| | |
|---|---|
| **Director, ICT** Thomas Adewumi University, Oko | 2020-Till Date |
| **Member**, Management Staff Thomas Adewumi University | 2020-Till Date |
| **Senate Member**, Thomas Adewumi University, Oko | 2020-Till Date |
| **Technical Admission Staff** Thomas Adewumi University | 2020-Till Date |
| **Chairman,** ICT Unit Committee at TAU, Oko | 2020-Till Date |
| **Member**, Vice-Chancellor in-house- Committee TAU | 2020-Till Date |
| **Member**, ICT Resource & WEB contents Management Committee TAU | 2020-Till Date |
| **Member,** Internally Generated Revenue (IGR) TAU | 2020-Till Date |
| **Member**, Admission Drive Committee TAU | 2020-Till Date |
| **Member,** Library and Publications Committee TAU | 2020-Till Date |
| **Coordinator, E-Library** TAU | 2020-Till Date |
| **Member,** E-Resources Committee TAU | 2020-Till Date |
| **Director, ICT** KolaDaisi University, Ibadan | 2018-2020 |
| **Senate Member**, KolaDaisi University, Ibadan | 2018 -2020 |
| **Chairman,** ICT Unit Committee KolaDaisi University, Ibadan | 2018 -2020 |
| **Member**, Vice-Chancellor in-house- Committee KolaDaisi University | 2018 -2020 |
| **Member**, ICT Resource & WEB contents Management Committee KDU | 2018 -2020 |

**Member**, Management Staff KolaDaisi University, Ibadan 2018 -2020

**Member,** Internally Generated Revenue (IGR) KolaDaisi University, 2018 -2020

**Procurement officer,** KolaDaisi University, Ibadan 2018 -2020

**Member**, Admission Drive Committee KolaDaisi University, Ibadan 2018 -2020

**Member,** Library and Publications Committee KolaDaisi University 2018 -2020

**Coordinator, E-Library** KolaDaisi University, Ibadan 2019-2020

**Member,** E-Resources Committee University KolaDaisi University 2018 -2020

**Referees**

**Professor Kolade Luke Ayorinde**

Former, Sub-Dean Agric. Dept., University of Ilorin

Former, Dean Agric. Dept., University of Ilorin

Former, DVC-Academic, University of Ilorin

Former, Vice-Chancellor KolaDaisi University, Ibadan, Oyo State

Former, Vice-Chancellor Thomas Adewumi University, Oko-Irese Kwara

**PHONE N0:** +2348033945317

**Email**: tomjay917@yahoo.com, tomjay917@gmail.com,

**MR. AJAYI WILLIAMS ADEKUNLE, CNA**

Finance and Accounts Dept, National Judicial
Council, Supreme Court Complex, Three Arms Zone,

**E-mail:** willyadekunle@yahoo.com

**Phone N0:** +2347068164742, +2348098601055

**MR. AMODA OLUFEMI**

229 JKK HOUSE, Ikorodu Road, Lagos
Unit Head Enterprise
Phone: +2348023280483



---------- ----------

Signature Date

**University Compliance Form**

This is to certify that this thesis by Bamisaye Folorunsho with Matriculation Number LCU/PG/002134 in the Department of Computer Science, Faculty of Basic Medical Sciences, Lead City University, Ibadan is in full compliance with the approved University's Format and Style.

---------- ----------

Signature Date