REPUBLIC OF TÜRKİYE

ALTINBAŞ UNIVERSITY

Institute of Graduate Studies

Electrical and Computer Engineering

NEW SOFTWARE DEFECT PREDICTION METHOD BASED ON PCA AND OPTIMIZED LSTMOF

Yahya Khalid Ali ALI

Master's Thesis

Supervisor

Asst. Prof. Dr. Zaid HAMODAT

Istanbul, 2022

NEW SOFTWARE DEFECT PREDICTION METHOD BASED ON PCA AND OPTIMIZED LSTM

Yahya Khalid Ali ALI

Electrical and Computer Engineering

Master's Thesis

ALTINBAŞ UNIVERSITY

2022

The thesis titled NEW SOFTWARE DEFECT PREDICTION METHOD BASED ON PCA AND OPTIMIZED LSTM prepared by YAHYA KHALID ALI ALI and submitted on 14/12/2022 has been **accepted unanimously** for the degree of Master of Science in Electrical and Computer Engineering.

Asst. Prof. Dr. Zaid HAMODAT

the Supervisor

Thesis Defense Committee Members:

Asst. Prof. Dr. Zaid HAMODAT	Department of Electrical and Electronics Engineering, Altınbaş University	_____
Asst. Prof. Dr. Mesut ÇEVİK	Department of Electrical and Electronics Engineering, Altınbaş University	_____
Asst. Prof. Dr. Abbas UĞURENVER	Department of Electrical and Electronics Engineering, İstanbul Aydın University	_____

I hereby declare that this thesis meets all format and submission requirements of a Master's thesis.

Submission date of the thesis to Institute of Graduate Studies: ___/___/___

I hereby declare that all information/data presented in this graduation project has been obtained in full accordance with academic rules and ethical conduct. I also declare all unoriginal materials and conclusions have been cited in the text and all references mentioned in the Reference List have been cited in the text, and vice versa as required by the abovementioned rules and conduct.

Yahya Khalid Ali ALI

Signature

DEDICATION

Thanks, praise, and gratitude to God before everything and after everything, I thank God Almighty for providing me with mental strength, health, and endurance for hardships and giving me the necessary ability to finish this course of study. After God, thanks to my parents and family who are the real inspiration behind my success. Thank Asst. Prof. Dr. Zaid HAMODAT for helping me so much. He alleviated all the difficulties I faced in studying. Finally, I would like to thank all the teaching doctors from whom I received so much information and thank everyone who supported and encouraged me during this research process.

ABSTRACT

NEW SOFTWARE DEFECT PREDICTION METHOD BASED ON PCA AND OPTIMIZED LSTM

Ali, Yahya Khalid Ali

M.Sc., Electrical and Computer Engineering, Altınbaş University,

Supervisor: Asst. Prof. Dr. Zaid HAMODAT

Date: 12/2022

Pages: 57

It is natural for the developed software to contain some defects and errors. The important issue is to detect these errors. Tests play an important role in detecting errors, but this is not always sufficient. Therefore, effective methods are needed to detect software errors and defects. The detecting error before publishing and serving the software the is main aim of the companies and the software engineers but this issue remains difficult and challenging issue.

In this study, new method based KNN, and parameter optimization techniques presented in the field of software defect prediction. The proposed method applied ensemble learning with KNN based Grid search to present new model for software defect detection. Furthermore, several classifiers are applied to obtain best detection rate. The obtained results are compared and discussed to select best model. The presented method obtained suitable results when compared with traditional techniques. The three datasets that are used in this study are CM1, JM1, and PC1 which proposed method presented various results with these datasets vary between 75% to 94%.

Keywords: Software Defects, KNN, Software Engineering, Grid Search.

TABLE OF CONTENT

Pages

LIST OF FIGURES

Pages

ABBREVIATIONS

DT	:	Decision Tree
KNNA	:	K-Nearest Neighbor Algorithm
SVM	:	Support Vector Machine
NBC	:	Naive Bayes Classifier
SDP	:	Software Defect Prediction
LSTM	:	Long Short Term Memory
PC-OPA	:	Principle Com-Pent Analyses

1. INTRODUCTION

Today, technology is advancing rapidly and the software world is also developing with these advances. Despite these developments, the software revealed by the developers may contain some errors. Errors in software can generally be grouped under three categories; design errors; are typically errors in the design of the software architecture, implementation and delivery errors; These are errors that occur after the software developer's delivery to the customer. Detection of these errors at an early stage plays a major role. Software bugs; It causes developers to review the faulty modules in the projects repeatedly and as a result spend more time on the developed modules. Early detection of bugs in software; It ensures that the software is error-free and of higher quality before it is delivered to the customer [1].

When a software defect is encountered at any stage of the software development lifecycle, if the software development lifecycle is Waterfall, the analysis returns to the first step. A worse situation is that this situation can be encountered after delivery to the customer. In this case, which is encountered after delivery to the customer, the same step is repeated. Therefore, software maintenance costs increase significantly. In addition, trust problems may arise for customers who encounter software errors. It is important to constantly test and verify the source code in ensuring the quality of the developed software. According to Boehm, the cost calculated after the delivery of a software that is in a condition to be delivered to the end user is higher than the cost calculated during the development of the software. Software testing tools are used to detect errors in developed software. With test tools, the accuracy of the software is ensured before it is delivered to the customer. Test tools may not always be sufficient to catch the errors that occur [2].

As the dependency on software increases, software quality becomes more important today. Software; It has been used almost everywhere and at every stage of life. Results that affect the software, such as errors, can lead to customer dissatisfaction and reduce the quality of the software. A software error causes both loss of time and financial losses [3]. The experience of a software developer when he/she first develops is not the same as the ability to predict errors in the software he/she develops after he/she develops himself/herself and gains experience. In

order to achieve this, it is necessary to know which programs are more error-prone than others, and at which stages the probability of error is higher. Previous studies have shown that 27% man-hours were consumed by testing during the overall development process [4, 5].

In software, larger modules tend to have more bugs than smaller modules. This is because larger modules offer more opportunities for bugs to settle, larger modules are more difficult to check and test for bugs, and are more complex. An example of possible linear and nonlinear relationships between magnitude and error is given in Figure 2.2. A linear relationship means that the number of errors will increase steadily as the module size increases. In a non-linear relationship, it is stated that as the size of the module increases, the error density also increases Software maintenance activities are an important part of the life cycle of software projects that are critical to software quality. Bug reports play an important role in software maintenance activities. Software bugs are inevitable, bug reports can be used to guide developers in processes such as fixing bugs in software, estimating bug fix time, deciding which bug should be fixed first, deciding who needs to fix a particular bug, etc. In addition, bug reports can provide valuable information about the status of the software project. Usually, errors are reported to a bug tracking system used to manage software development and maintenance activities in software projects. Bug tracking systems are designed not only for developers, testers and customers to identify and report problems found in systems, but also to store errors and information about errors and monitor the status of related tasks. Therefore, a bug tracking system can increase the efficiency of detecting bugs while further improving software quality. Error reports that describe a previous error in the system differently are defined as duplicate error reports. As recurring error reports are entered into the system, the number of errors in the system increases. As shown in studies, the rate of recurring error reports among the total error reports can be up to 25-30%. Duplicate bug reports lead to multiple developers being assigned the same bugs for the same bug, who reproduce and fix the bug for a similar reason. As a result, the effort spent for the development of the software increases, and time and cost loss occur [6].

In order to produce a solution to a problem, first of all, it is necessary to have a good understanding of what the problem is. All developed software is prone to possible error. In an other saying; All software is probably faulty unless it goes through a good analysis and testing

phase and integrates with the user. Software error estimation is seen as one of the most useful and least costly operations. The fact that the software creates errors after meeting with the user can lead to many bad consequences for the publisher of the software, especially the image of the software itself. Software error estimation can be seen as a vital stage on which the quality of the product being developed depends on the software developers. In addition, the customer's response to the product quality will be satisfactory and flattering [7].

1.1 RESEARCH QUESTIONS

This study try to answer several questions that presented by the author and during the thesis research these questions are answered:

i. What is software defect and how we can presented new methods that can deal with this problem ?

ii. What is the machine learning techniques that can present remarkable results with this problem?

iii. How can we presented new method with accurate results within minimum execution time ?

1.2 MOTIVATIONS

The motivation of this study is that the software defect prediction is very difficult and critical issue in software projects. The defect prediction by humans needs high expertise and long processing time with large costs Experienced software testers demand high salaries, it's hard to find good testers all the time. Therefore, academic researchers and large companies are trying to develop methods based on data mining and machine learning to work in this industry. Moreover, various deep learning and machine learning methods that have been introduced in the field over the past few years have automatically detected a software error. These studies remain marginal when it comes to training and examining the best parameters of the model. A new study was then presented in which several classifiers were combined to select the best classifier. Additionally, we introduce Grid Search to refine the classification parameters.

1.3 AIM OF THIS STUDY

In recent years, there has been an increasing interest in the effective application of machine learning models to software defect estimation problems. However, natural language processing, computer vision, image processing, etc. Compared to other areas, there is little literature on market forecasting using deep learning. With the development of web technology and the expansion of access to large amounts of data on a stock, deep learning is suitable for forecasting and analyzing a market. The main reason for choosing deep learning as an alternative to existing models and approaches is that it allows the representation of abstract features from data, not based on economic assumptions or human experience, and is non-linear. Because relationships can be built. Another reason why research favors machine learning models is that KNN are superior statistical models in modeling nonlinear relationships between dependent and independent variables. This directly improves network performance in general.

In this study, we present a sophisticated and unique method that can be used to solve software defect detection problems that can be easily adapted to any database with different instances and features. This research was a step towards natural communication between humans and machines and made a giant leap in fields as diverse as industry, medicine and society.

2. LITERATURE REVIEW

In this chapter, we review general concepts that are related to software engineering and software management enhancement and discuss their functionality in motivating the research in the subsequent chapters of this thesis. Discrimination between software defect and software management is crucial for both software projects. This chapter presented the previous studies that are used in this field.

2.1 SOFTWARE ENGINNERING PROJECT MANAGEMENT

Project start and finish times are a clearly defined process and it tries to achieve adequate coverage under cost and time constraints, another constraint. Each project is a temporary process that has been put into writing for a specific purpose. It needs resources such as people, money, technical equipment and financiers such as sponsors or customers to provide these resources; it is a process involving uncertainty and risk. It is the act of achieving well-defined goals and objectives under budget and time constraints, with clearly defined activities to begin and end [8].

The general objectives and duration of the project are clear. Projects are temporary jobs, so every project has a beginning and an end. Every project is unique. No project is alike. No project repeats itself in the same form and conditions. Every job is done for a purpose. In projects, the purpose is a focal point, the reason for the existence of the project. If there is no purpose, there is no project. Every job is performed under certain constraints and using certain resources. The project is a system development approach. The systems approach is a method in which multiple interrelated parts are worked together to identify problems and solve them. The system takes input, processes it, and generates output. In addition, the system includes the principles of integrity, efficiency, communication, interaction and continuity. Their systematic arrangement, planning and development require projects. Because projects allow the work to be structured in an easily visible way and to open the way to think and work within the systems [9].

2.1.1 History of Project Management

Project management applications that include the techniques we use today actually date back to the 1790s at most. The industrial revolution that started in America in the 1790s created diversity in the features of products, changes in customers' expectations and ways of doing business. The acceleration of transportation meant the reshaping of the market. With all these changes, unavoidable changes in people's work, home and communication, laws, customer behavior, shopping methods have brought new opportunities as well as threats for institutions. The only way for an organization to be least affected by these changes is to complete a fast and effective adaptation process. In the adaptation process, first of all, the most important processes to be considered are the correct definition of the objectives, the good use of existing resources, the ability to coordinate the communication between responsibilities well, and to keep costs and risks under control [10].

2.2 SOFTWARE PROCESS MODELS

Software models are abstract representation and application forms that deal with the software development process from a special perspective.

2.2.1 Waterfall Model

In the waterfall model (classical model), the software development stages consisted of sequential steps. The waterfall model is the most common process model used in software projects. In this structure, the development of software follows a linear path. That is, after one phase is completed, the next phase begins. Therefore, when a problem is encountered in the later stages of software development, it is difficult to return to the previous stages. Such turns often require additional costs and additional time [11]. The representation of the waterfall model is shown in Figure 2.1.

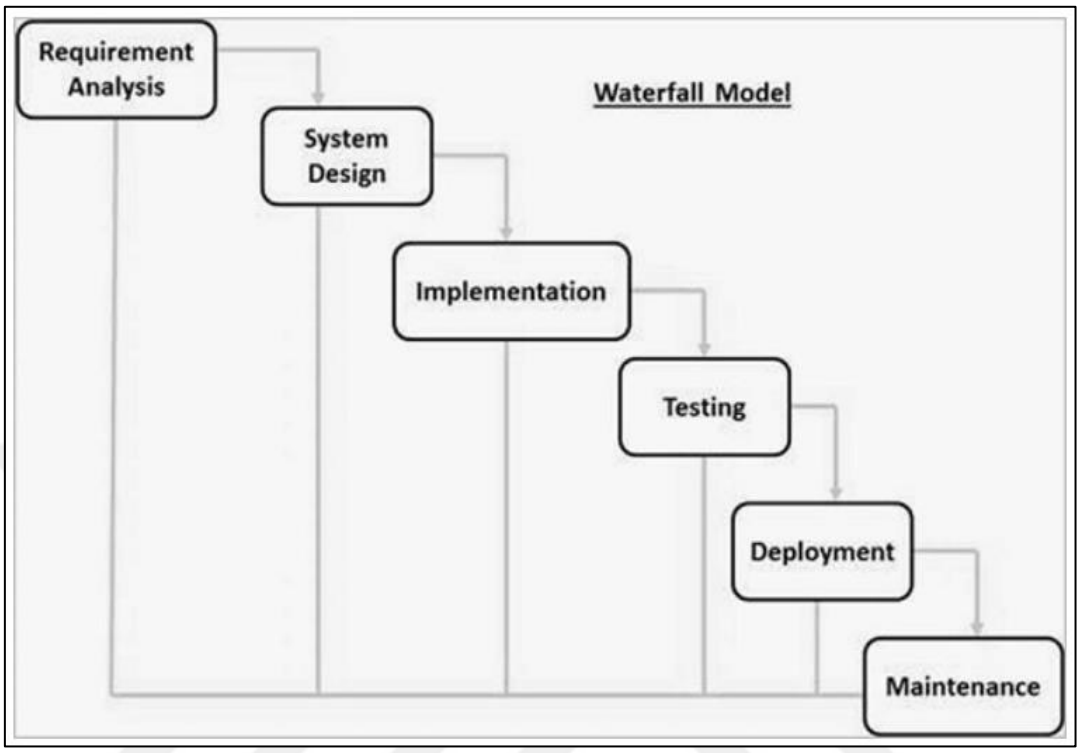

Figure 2.1: Waterfall Model.

Although it is the most widely used process model in software projects, there are the following problems in the application of this model:

i. In this model, even if a return to the previous stages is foreseen during the implementation, the burdens of the returns on the project cost and schedule are high.

ii. Feedback can cause problems among the project team.

iii. It is difficult to fully determine all the requirements of the customer at the very beginning of the work. This is because, as with any project, there can always be some uncertainty at the beginning. In some cases, the user cannot fully express their wishes at an early stage of software development.

iv. A program that works according to this model emerges long after the projects. This can sometimes be too late for the customer to give feedback about the developed software, and customer dissatisfaction can occur.

v. When the customer starts to respond to the software as it emerges, it may come up with a new requirement. This often causes serious disagreements. The waterfall model, as the oldest software process model, has formed a basis for other models defined today on how to do software development[12].

2.2.2 V-Model

The V model is an extensive approach to the waterfall perfect. This methodology is a test activity model that matches every stage of the software development process to develop more reliable and accurate software. Model V also initiates authentication activities, which can save a lot of time on the project. In this model, left branch V covers requirements analysis, system design, software development and implementation (coding) phases. The right side of the V contains the model validation step. This includes unit tests, integration tests, system tests, and acceptance tests [13].The representation of the V-pattern is shown in Figure 2.2.

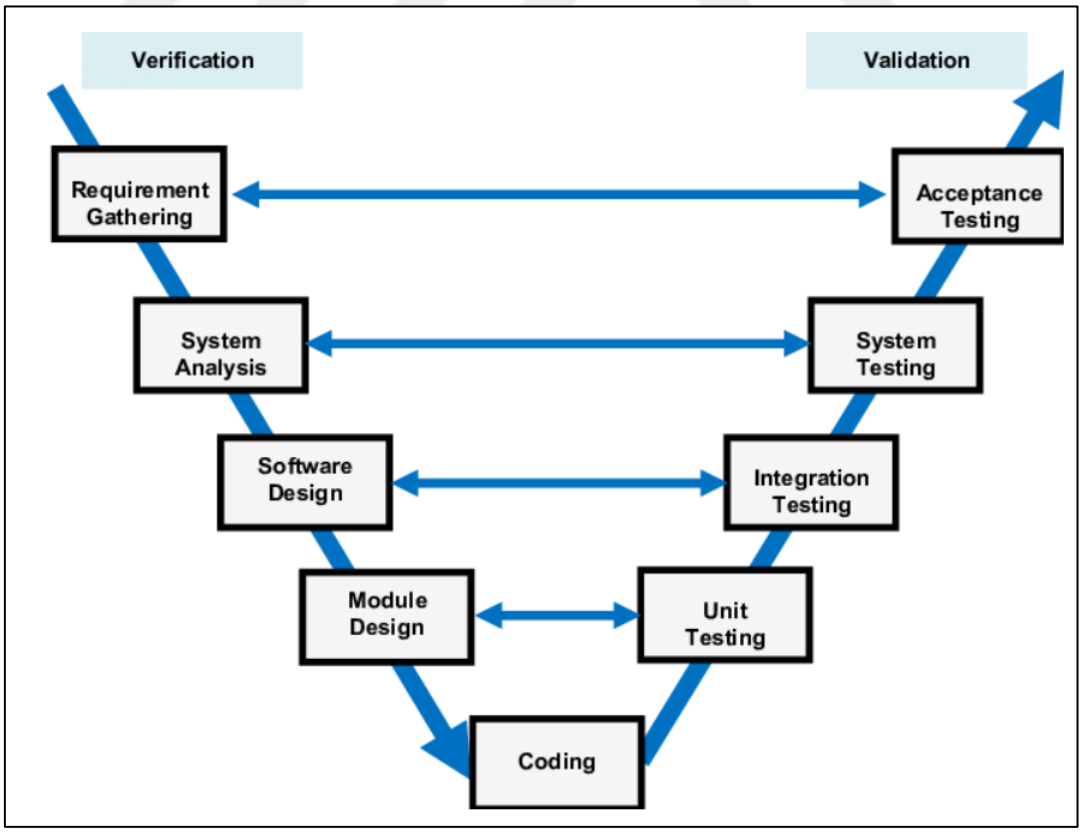

Figure 2.2: V-model.

2.2.3 Prototype Development Model

When the best practices of software projects are examined, it is observed that at the beginning of the project, users can only give very general definitions of their needs. They cannot describe in detail the necessary inputs, processes and outputs for the software to be developed. In this case, the prototype development model can be used to develop the software requested by the users and to eliminate the ambiguity of the requirements [14].

In cases where it is decided to use this model, the requirements that can be determined are quickly collected and the work is started. For this purpose, a quick consensus is reached on the functions expected from the software to be developed and the inputs and outputs of this software at the meetings attended by the developers as well as the users. Then, in the light of the information determined, a prototype is created that describes the direction of the software to be reflected to the user with a design. This software is made available to the user for use and evaluation. By looking at these evaluations, new requirements are tried to be determined. Thus, the user

This work is continued until the desired software is understood. After the requirements are detailed enough, the desired system is developed. In this approach, users can perceive prototype software as real software. However, the sample software developed to determine the requirements is generally slow, inefficient, has low reliability and has high errors. The prototype development model can be successfully applied when the requirements are not clearly known [15].

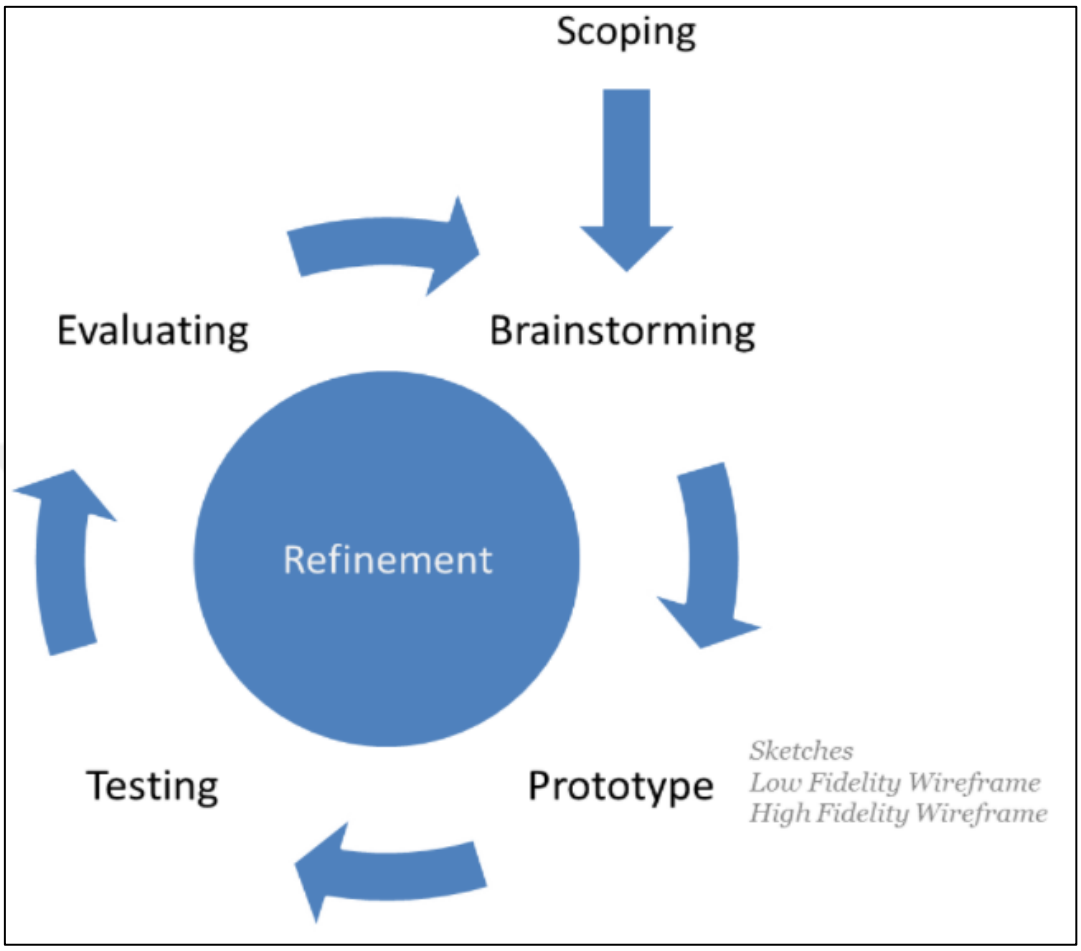

Figure 2.3: Prototype development model.

2.2.4 Spiral Model

In the spiral model, the software development process is expressed in a spirally expanding structure instead of sequential activities with no reversal. When the spiral model is used, a new version of the software emerges at the end of each helix. The software revealed at the auction at the end of the first ring is a true prototype of what is used in the application environment. The whole process was implemented during its development. In subsequent iterations, new functionality is added on top of it, again using all the steps of the process. The development process is continued by ensuring that the software developed in terms of quality, efficiency and capacity has the appropriate features in each spiral cycle (in each new version) [16]. The representation of the spiral model is in Figure 2.5.

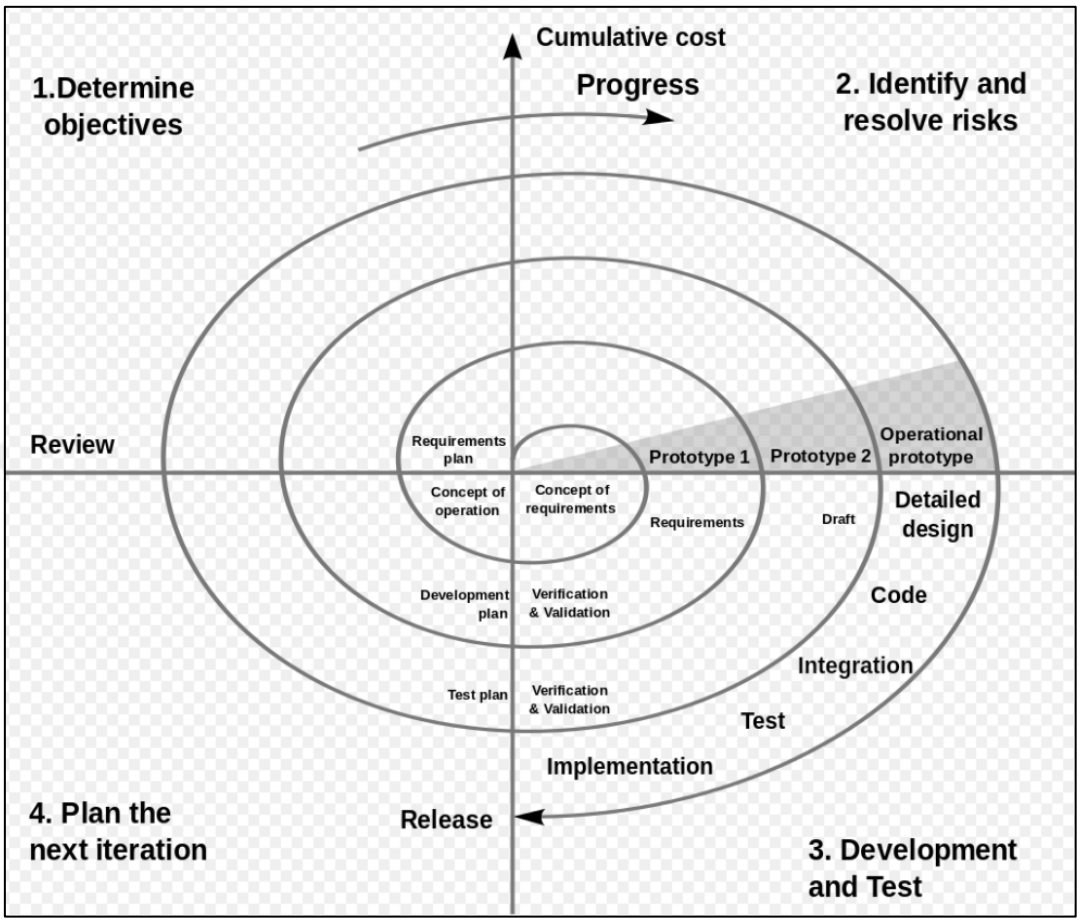

Figure 2.4: Prototype development model.

2.2.5 Incremental Model

An incremental model was created by adding an iterative feature to the waterfall model [17].This model is based on developing and delivering software in releases on a specific schedule. Each new version foresees the addition of some additional functionality on top of the previous one. This model can be used effectively when all the requirements are clear. This model differs from evolutionary development in that each release includes all the functionality of the final product. The incremental development model is shown in Figure 2.6.

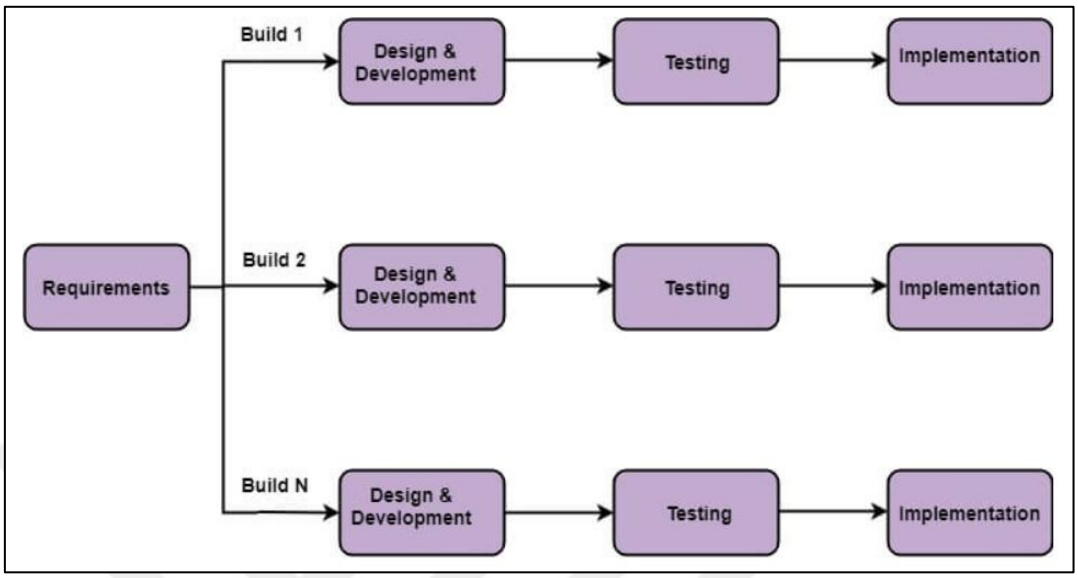

Figure 2.5: Incremental model.

2.2.6 Software Testing

Programmers may think they write their code to meet requirements, but software systems contain many different states, formulas, and complex algorithms. However, the size of the project and the multitude of employees can increase complexity. Therefore, the existence of errors creates a problem not only for the software, but also for the expectations of the user and the customer [18].

Generally speaking, if the software does not meet the described requirements, it is faulty. The error may have occurred as a result of many different reasons.

Requirements may be incorrect or incomplete. It may be impossible to implement some requirements in hardware and software. The overall design of the system may contain errors or, most commonly, the code written may be faulty. As a result, one or more the error may be the cause of the failure.

Testing is important in terms of finding errors, gaining confidence for software quality, obtaining information for decision mechanisms and preventing deficiencies. Testing increases the quality and reliability of the program [19].

Software testing is a technical activity, but it takes into account human psychology and several important criteria. In general, you want to try every possible change to your software, but most of the time you don't. Even a seemingly simple program can have hundreds or thousands of possible inputs and outputs. It is impossible to write test cases for all these possibilities. Thorough testing of complex applications takes time and effort. Software testers should have a good understanding of software application testing. Sometimes the evaluator's point of view is more important than the process you'll face. The main reason software applications aren't tested is because developers start the process with the wrong definition. When we test our program, we want to add value to it. The added value of software enhances the reliability and quality of the software. The reliability of the program depends on finding and fixing bugs. It doesn't check if the program is running, it assumes there is a bug and tries to find as many bugs as possible. In this sense, testing is the process of running a program to find bugs. Testing is a destructive process that most people struggle with, but we all prefer to rely on destructive things in our lives. This definition includes how test data is actually developed and how it is determined who can or cannot perform the tests. In addition to these definitions, you can specify pass and fail words that project managers use to rank test results based on certain characteristics. The project manager defines no-error test cases as passing test cases and bad test cases as errors. In fact, bankruptcy is a strange definition. As bugs are discovered and fixed, run well-designed and executed test cases. I have run the same test multiple times and it runs without errors. A test that does not actually use a program and has no errors in its test cases can be defined as a failed test. Due to the nature of programming, an error-free program is optional. Conditions that can be controlled include normal and abnormal conditions. In other words, testing is about predicting what will happen if you intentionally make a mistake. The purpose of testing should be to find what is needed and what is missing, and vice versa.

2.2.7 Software Defect Prediction

Software bug studies; The program examines examples of development error-prone code, helps to scale up the software and get more use [20].Software defect investigations can generally be grouped into 3 different categories as metrics, datasets and application. In this study, software error prediction datasets containing method-level software metrics have been learned for a long

time. The metrics are in different categories such as method-level, class-level, radio-level, file-level, operation-level, quantitative-level. Software is the summary expression of software that can be measured, values or sets of values that can be measured. In the field of software defect research, method-level software metrics are the most widely used metrics. These metrics are popularly used with machine learning [21]. Datasets are raw data about a topic. Public software bug datasets are getting better by the day. Someone related to space exploration is NASA's PROMISE repository [22]. These software datasets provide maturation and complete availability of error prediction models.

In addition, these datasets allow discussions and developments on methods and results among researchers. There are a number of metric measurement values and variables in the datasets used [23, 24].

2.2.8 Litreature Review

[25]The Smote algorithm was applied simultaneously to different percentages of the dataset used to rank software bugs, increase minority class, and decrease majority class. They measured the success of the classification of the G-means of the C4.5 decision tree algorithm. This showed that the probability of success of the classification was increased. In this study, multiclass data was extracted from the dataset. Instead of shrinking the existing majority, only a few were enlarged. The software metrics of the new dataset obtained in this way were examined with regard to the information collection and its influence on algorithms such as C4.5 and PART [26].

[27] used the quartile chart for software error estimation. Using data mining methods OneR, C4.5 and NaiveBayes, they developed an error estimation method with non-dynamic code features.

[28] compared classification success for software error prediction using 22 classifiers and 10 publicly available software error prediction datasets created by NASA. In the results obtained, they stated that the metric-based classifiers gave successful results, but there were no substantial

changes in difference in conditions of the success story of the classifiers. In this study, data preprocessing was used to the datasets with the Thumped procedure.

[29] deal with randomness, data increase, data reduction and cost control suggested above to solve the class imbalance problem. They studied the diversity of classification success with the size of the training sample and the level of class imbalance. The general results obtained showed that the distributions of unbalanced classes influence the results of decision trees, other neural networks and supporting vector machines.

[30] performed experiments measurement learning working on 13 datasets by improving artificial data and reducing artificial data applying the techniques they named Smote + Tomek and Smote + ENN. In the general findings found, it has been shown that artificial data augmentation techniques are advanced to artificial data saving techniques.

[31] developed ADASYN: adaptive synthetic data sampling approach for the unbalanced learning problem. different minority a weighted distribution was preferred for the difficulties experienced in the learning process according to the class percentage levels. Thus, the tendency resulting from class imbalance has been reduced and it has been shown that the limits of the classifier have been increased to include difficult examples.

[32] Nima Shiri Harzevili et al. We propose a new framework "MLMNB-SDP" to identify software metrics with large conditional dependencies using statistical hypothesis testing. MLMNB-SDP is designed to handle conditional dependencies with a single hidden variable in a predefined structure that preserves relationships between software metric pairs in class variable instances. We measure the effectiveness of our approach based on its ability to measure conditional dependencies in program values and predict downtime. The first uses conditional mutual information (CMI) and the second uses three parameters to predict failures. (1) Design error estimation (WPDP), (2) Design error estimation (CPDP), (3) k-fold cross validation. The results of the metric dependency analysis show that traditional file-level software measurements show significant modal correlations and the use of pure Bayesian classifiers in this area is theoretically unacceptable. Our 3D results show that MLMNB-SDP outperforms known test

classifiers in accuracy and performance, increasing the pure Bayesian classifier from 5.45% to 75.86%. H. Random logistic regression and forest. indicator F1.

In [33] Chao Ni et al. Propose a new method MOFES (Multipurpose Feature Selection) which takes into account two goals for improvement. The goal of optimization is to reduce the number of specific features, and this goal is closely related to the cost analysis of this problem. Another goal is to improve the performance of the installed SDP models. This objective is related to the utility analysis of this question. To solve this problem, MOFES uses the Pareto-based Multi-Objective Optimization (PMA) algorithm. In our test cases, we developed and tested RELINK and PROMISE databases in a real open source project. They first analyzed the different effects of PMA on MOFES and found that NSGA-II was the best fit for both data sets. We then compared the MOFES method with 22 filter-based selection and processing methods and found that MOFES can select fewer but closely related features to generate high-quality models. MOFES also frequently analyze selected features and can use these results to make recommendations for compiling high-quality SDP databases. Finally, we analyzed the computational cost of MOFES and found that MOFES took only 107 seconds on average.

In [34] Ruchika Malhotra et al. Evaluate the performance of a machine learning classifier in predicting minor errors in 12 volatile NASA datasets by applying rigorous sampling and classification methods. They review the five existing minority sample cloning resampling methods and propose a new method called SPIDER3, which replaces the SPIDER2 resampling method. This article measures MetaCost student performance in cost-effective learning using a disproportionately large dataset. The results show that the use of the resampling method improves the prediction capacity of the machine learning classifier. Moreover, the proposed SPIDER3 method shows promising results.

In [35] Diana-Lucia Miholca et al. Basically, there is a new map classification method called HyGRAR to predict bugs in software. HyGRAR is a nonlinear hybrid model that combines association rules and cascaded neural network analysis to distinguish between good and bad software objects. In experiments based on 10 open source datasets, we found that the HYGRAR

classifier works well. Using the same data set to assess performance, HyGRAR has outperformed many previously proposed approaches to assessing software defects.

In [36] Yuanxun Shao et al. proposes a new control method to predict programming errors based on ATOmic Class Association Rule (ACAR) logic. There are specific types of result class tags, unique ancestral properties, and custom mapping rules that examine the relationship between properties and categories. There are specific types of result class tags, unique ancestral properties, and custom mapping rules that examine the relationship between properties and categories. These attribute models can provide valuable information that software developers can easily understand. A new free-associative rule-based error prediction model is used for data preprocessing, rule modeling, and performance evaluation to improve the prediction of failed devices. ACAR, NASA CDM and PROMESA (Association-Based Classification Extension (CBA2), Support Vector Machine, Naive Bayes, Decision Tree, OneR, K-Nearest Neighbors, and RIPPER) are data from 7 additional study participants. was evaluated using result. .Evaluation results. A detailed analysis of the comparative experience of ACAR and CBA2 evaluated the relevance of programming errors. ACAR was superior to CBA2 in AUC, mean G, balance, and visual acuity. At the same time, the average ACAR ACAR can reach 81.1%, which is 2.9% higher than CBA2.

In [37] Ning Li et al. conducted a systematic literature review and identified 49 studies that included results from 2,456 individual experiments published between January 2000 and March 2018. Based on meta-analysis, automated forecasting models are similar to those tested within and across projects. Of the missing 14 model families, the most influential are Fuzzy CMeans (FCM) and Fuzzy SOM (FSOM). Checking this, almost 11% (262/2456) of published results (out of 16 articles) are inconsistent and 33% (823/2456) are insufficient to confirm this. Details are included.

In [38] Zhongbin Sun et al. Provides a recommendation filter-based sampling (CFSR) algorithm that automatically recommends efficient sampling methods for new error data. CFSR first evaluates current sampling techniques against historical defect data, then identifies meta-attributes and data similarities between old and new defect data. Finally, we combine all the

information and data similarities of sequential sampling methods to create a proposed network that uses a common user-based filtering algorithm to propose a suitable sampling technique for new data. It's buggy. We performed a wide range of experiments on 20 soft error datasets using 5 classification algorithms, 2 predictive runs, 5 speculative runs, and 12 ensemble sampling methods. The experimental results demonstrate on the one hand the importance and necessity of this study and on the other hand demonstrate the validity and efficiency of the proposed CFSR method.

In [39] Xingjuan Cai et al. The SVM-based Large-Scale Software Error Prediction (HD-SDP) model is used to troubleshoot registry class instability and determines the time zone of error registry class predictors that support vector machine of parameters (SVM). It has four goals: false positives, probability of detection, F-score, and balance. In addition, a complex integration of the multi-parameter temporal optimization algorithm (UIMaOTO) was developed in this model, which supports the simultaneous selection of SVM parameters and global modules. UIMaOTO uses specific strategies to create formula subelements and develops a unified integration strategy to reduce the burden of algorithm selection. UIMaOTO can be compared to other advanced algorithms for experimental results using the common DTLZ test suite.

In [40] Shuo Feng et al. Experiments were conducted to empirically investigate the effect of sampling interval on the performance of three common SMOTE-based methods. Based on the experimental results, they developed a newly proposed resampling method. That is, our experimental results of 5 group classifiers from 30 unbalanced datasets in the SMOTUNED repository: (1) the choice of distance criterion has limited effect on SMOTE-based performance; method, (2) number of synthesized noise samples, noise classification; . The overall performance of methods based on AUC and SMOTE measurements is largely unaffected by sample spacing and (3) detection probability (pd) and false positives. According to the SMOTE method, the probability (pf) does not depend much on the distance between samples. The sample is taken when the resistance is independent of capacitance.

In [41] Cong Jin et al. KTSVM with DA capability (called DA-KTSVM) is also used as CPDP model. Since DA-KTSVM parameters affect prediction performance, these parameters were

optimized using the Advanced Quantum Particle Swarm Optimization (IQPSO) algorithm, and the improved DA-KTSVM was named DAKTSVMO. To verify the efficacy of DA-KCMMO, we conducted experiments on 17 open source projects. Moreover, DA-KTSVMO can improve prediction performance by utilizing good knowledge of available data and implementing erroneous data reuse.

In [42] Lei Qiao et al. propose a new approach using deep learning methods to estimate the number of errors in software systems. First, we preprocess the entire dataset, including data transformation and normalization. It then models the data to prepare input for the deep learning model. Third, the number of defects is estimated by inputting the simulation data into a specially designed deep neural network model.

In [43] Meetesh Nevendra et al. studies the effect of hyperparameter optimization on error rate estimation. Based on the experimental analysis of 15 corrected error data sets, a hyperparametric optimization of the training procedure was performed. This is required for hyperparameter tuning. Random search and grid search worked well, but grid search always gave better results than the default settings. However, this is not possible with a random search. This underscores the importance of examining the parameter space when using parameter-dependent regression methods.

In [44] Hua Wei et al. propose a new model using Local Tangent Vector Spatial Alignment Assistant (LTSA-SVM) algorithm to solve the smooth error estimation problem. This model uses the SVM algorithm as the main classifier for the program error distribution prediction model. Optimize model parameters using a built-in grid lookup method with more than 10 validations. Conventional dimensionality reduction algorithms lose unacceptable non-linear data features, which decreases the precision of SVM. To solve this difficulty, this article uses the LTSA techniques to extract a small internal data structure and effectively reduce the size. The SVM algorithm is based on small data. Finally, we test the feasibility of the predictive model..

In [45] Zhiguo Dinga et al. Provides an insulation property value chart to determine which property values are appropriate for a building's insulation tree. In community building,

community reduction strategies motivated by the principle of "more is better" are used to improve the resulting isolated forest. convergence with histogram-based class value selection and cluster reduction, which can improve cluster reduction prediction performance. Extensive experiments using 10 real data sets are underway to demonstrate the performance of the proposed SDP method.

Compare the performance of the custom error estimator with that of the current model. It was piloted in 6 open source projects involving 222 developers. Current and standard error estimators are generated using two algorithms and tested against historical validation data as well as seven performance metrics and statistical tests. Our results show that custom templates (PMs) increase the recovery rate by 24% for 83% of developers, with a higher false positive rate for 77% of developers. Private messages are better for modders and many older developers. Conformity factors influence the effectiveness of most leaders and vary based on experience, scope, and past performance.

3. MATERIAL AND METHODS

3.1 MACHINE LEARNING

Learning method is a statistical method that uses system inputs and outputs to analyze relationships between data. If the relationship between input and output is reasonably assumed, the output can be calculated from known input values. From optimizing data flow to location-based marketing, companies can create analytics applications to take advantage of data in a variety of ways. Converting data into information and knowledge is critical. Machine learning approach is defined as a learning process that can be automatically improved with experience. The knowledge gained from machine learning can be used to develop applications such as robot design, patient-centered care, and search engines relevant to user interests. Machine learning is a technique that allows computers to "learn" from specific data without the need for a detailed, detailed program for each problem. The goal is to model deep relationships in the input data and to determine the properties of the data. Learning outcomes can be used for assessment, prediction, and classification. Machine learning, a subset of artificial intelligence, generates mathematical models. The overall goal of machine learning is to identify patterns in data that indicate possible solutions to seemingly invisible problems. For example, in complex systems such as self-driving vehicles, computers “trained” to recognize “dangerous” situations must convert large amounts of sensor data into driving decisions [46]. One of the main advantages of machine learning is that machine learning algorithms can analyze large amounts of data and find patterns that may go unnoticed. Methods for finding patterns and trends in data are rapidly evolving with massive amounts of new data being collected and stored and more valuable computing power. This led to the machine learning phase. Machine learning originates from a branch of computer science, and it performs complex algorithms capable of processing big data to simplify model assumptions and discover useful decision rules based on information, statistics, and computer theory. Most real world phenomena are too complex to be directly modeled as I/O loops. Machine learning allows the automatic creation of mathematical models of these complex relationships by manipulating existing data and maximizing the key impact metrics associated with a problem. 4 The process of creating an automated model is called "learning", and the data used for training purposes is called "learning data". The learned model

provides new information about how well the input variables correspond to the output variables and can be used to predict new input values as well as training data. Machine learning is a research field devoted to the formal study of learning systems [47]. In addition, fraud detection, online intrusion detection, educational games such as chess, friendship or credit cards, medical diagnostics, system design recommendations and research using tools such as insurance and telephone companies, robots or unmanned cars. Engine or system information extraction These problems are solved using machine learning techniques. Machine learning systematically uses algorithms to synthesize key relationships between data and information. Machine learning algorithms can be divided into four groups according to the type of training. B. Unsupervised learning, supervised learning, partial supervision, and reinforcement learning. Classification is one of the most common decision-making problems in human behavior. Classification problems arise after classes are based on groups or sets of observable properties, where an object must be assigned to a predefined group or class associated with that object. That is, it is necessary to determine whether a particular sample belongs to a set of categories. One of the most common machine learning classification problems is to classify an unknown sample into one of the above classes. An important observation in classification is that the objective functions are different. In general, it is not possible to intentionally assign a number or other value to a class tag. In other words, properties of an evaluated class are categorical properties.

Regression is the problem of estimating the true value of each element. Estimates of changes in education costs or economic variables are examples of regression. In contrast to classification problems where there is no concept of convergence between different classes, in regression, the penalty for wrong estimation usually depends on the magnitude of the difference between the actual and expected values.

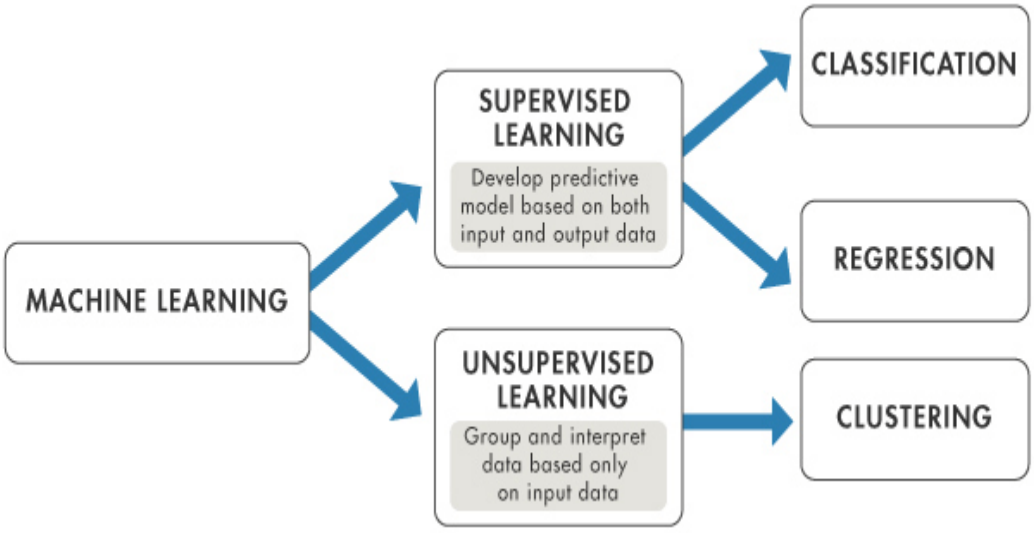

Figure 3.1: Machine learning techniques.

3.1.1 Unsupervised Learning

In unsupervised learning, machine learning algorithms try to find data patterns in previously unnamed data sets. In this case, we focus on user behavior in relation to advertisements, such as b. Previous behaviour, ad demographics, 'characteristics', and backlink and click suggestions. Unstructured learning on the same platform can also be used to identify past consumer behavior patterns, rank consumers, and adjust ad serving accordingly. This learning is not continuous, so customer classes are recognized in terms of attributes rather than labels. Discrete learning does not contain response variables or labels, and the goal of discrete learning is to understand the main features of an observation. So unassociated learning attempts to learn certain data attributes and data distribution relationships. Therefore, an important application of uncorrelated learning is to analyze group and cluster sample data to extract insights [48].

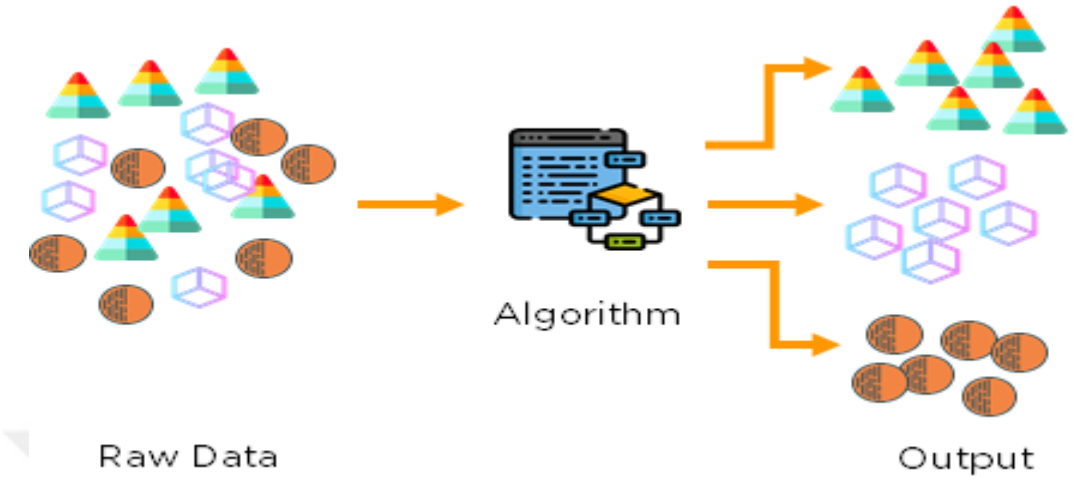

Figure 3.2: Unsupervised learning.

A reduction in size means a reduction in the number of items available. Reduction techniques simplify the use of data, often removing noise and making other machine learning tasks more accurate. This is usually a pre-processing step that can be performed to clean up the data before applying it to other algorithms. Dimension reduction requires mapping higher-dimensional inputs to lower dimensions in order to map the same point in input space to a neighboring point in the manifold. Dimensionality reduction, i.e. reducing the number of dimensions/features, is one of the most important tasks when analyzing multidimensional data. The main motivations for performing size reduction:

i. Calculation is faster with fewer features.

ii. Time and labor savings can be achieved and forecast accuracy can be increased if non-distinctive features are detected and removed.

cluster; Group posts, reviews, or samples for classes with similar features. A set is a set of records that are similar to records in other sets and to records in other records. Grouping differs from sorting in that there is no target variable for grouping operations. The clustering activity does not attempt to classify, predict, or predict the value of the target variable. In contrast, clustering algorithms attempt to divide the database into homogeneous subgroups or clusters. Increase similarity to grouped records and decrease similarity to ungrouped records.

Clustering is more complex than associative classification because no labels are added to the clustering model. For the cartographic taxonomy, this tag is the index that links the data objects. When grouping, it's hard to tell which group a model belongs to because there are no labels. Cluster analysis finds "natural" groups by grouping "similar" objects based on certain similarity criteria. Sorting the data points by blood group results in more similarity between groups and less similarity between groups, as shown in Figure 3.3.

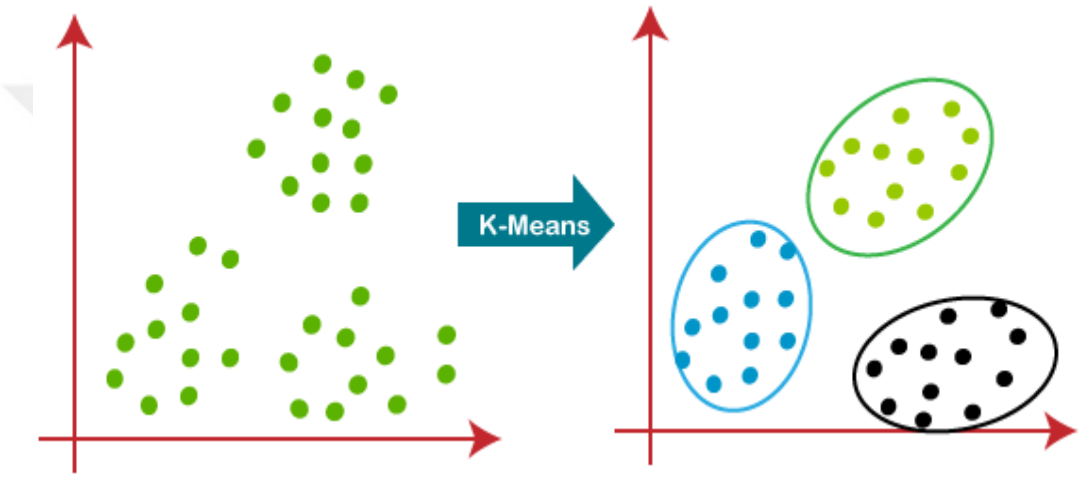

Figure 3.3: Clustering.

3.1.2 Supervised Learning

Supervised learning requires learning a map among a set of key variable quantity and an production changing and using this map to calculate yields for hidden (outputs unknown) data. In traditional teacher-led learning, home is taught through multiple educational examples. Each training example has a label indicating the desired outcome of the event described in the example [47]. In classification problems, labels indicate the category to which the relevant sample belongs, while in regression they are. Temperature, altitude, price etc. Represents the actual output like.

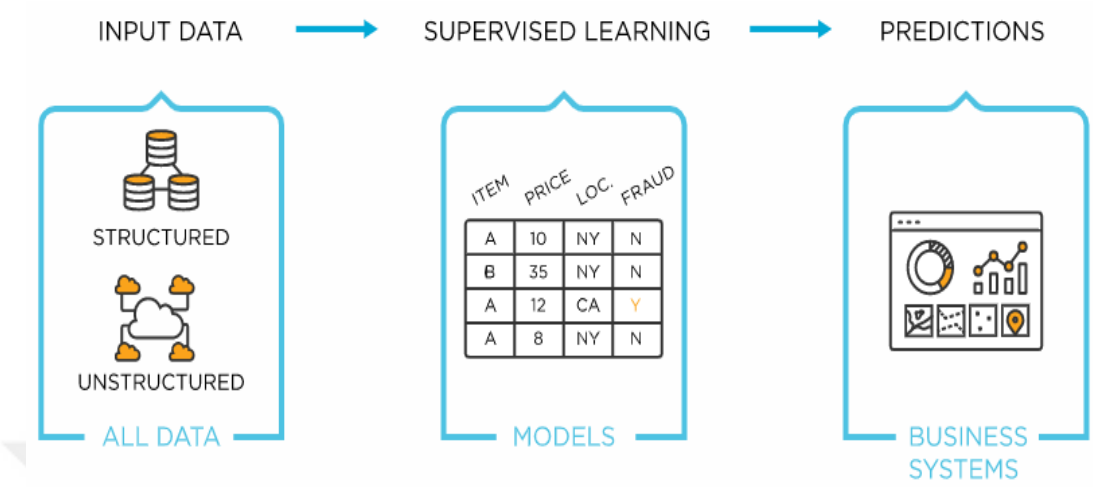

Figure 3.4: Supervised learning.

3.1.3 Semi-Supervised Learning

Semi-Supervised Learning Semi-supervised learning is in the middle of supervised and unsupervised learning. In addition to the unlabeled data, some control information is provided to the algorithm, but this is not mandatory for all examples. Often, this information is targets/tags associated with some examples. In this case, the dataset $X = (xi)i\in[n]$, the points $Xl \coloneqq (x1, \dots , xl)$ for which the labels $Yl \coloneqq (y1, \dots , yl\ )$ are provided, and the label unknown $Xu \coloneqq (xl{+}1, \dots , xl) {+}u)$ points can be divided into two parts.

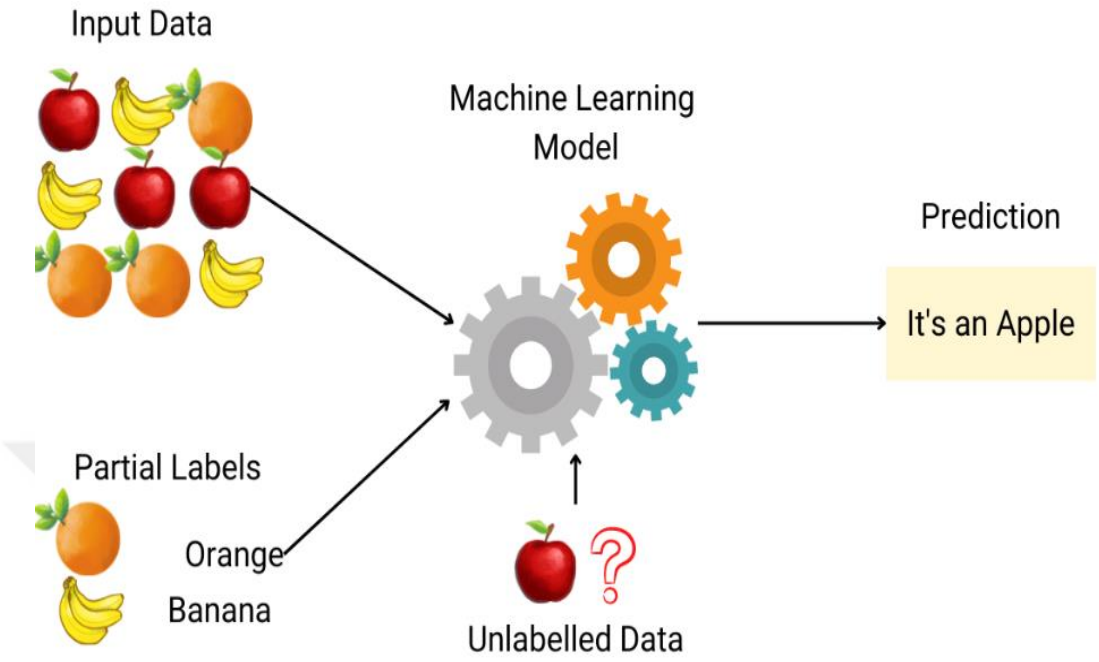

Figure 3.5: Semi-supervised learning.

3.1.4 Artificial Neural Networks (ANN)

The first study on the artificial neuron, which forms the basis of Artificial Neural Networks, was the work by McCulloch and Pitts (1943). They revealed that the model proposed in the study can be applied in a computable function using a limited number of neurons and adjustable synaptic weights. Artificial Neural Networks are the information processing process inspired by the processing processes of the structures of nerve cells such as the brain. The network expressed here refers to the correlation between neurons in various layers of the system. With ANN, which uses the optimization theory, meanings and trends that cannot be revealed by human eyes or other computer analyzes can be revealed by using complex or imprecise information. A trained ANN is seen as an expert on the subject to be analyzed. ANN gives successful results when applied in learning, optimization, classification, feature identification, association and generalization, which are functional features of the human brain. The main function of ANN is to provide machine learning. It has a different information processing process than the traditional transaction 29 processes, and the information on the connections of the network is

stored. It can produce effective results in patterns with missing information. They can learn and generalize over an event. It can only work with numerical information, symbolic information needs to be converted to numerical information [37].

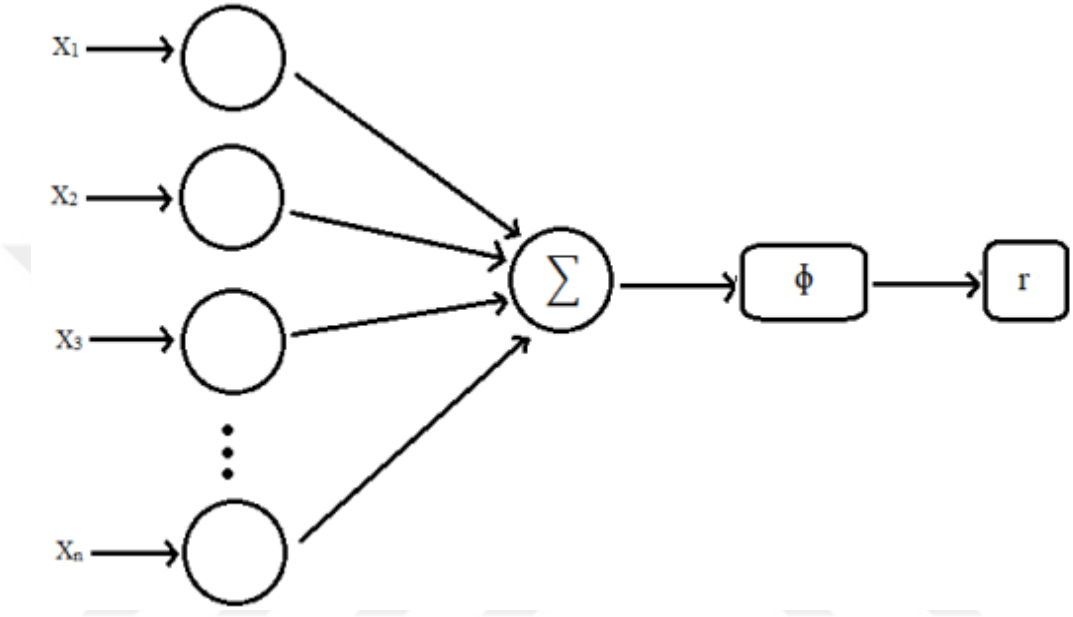

Figure 3.6: Shadow neural network.

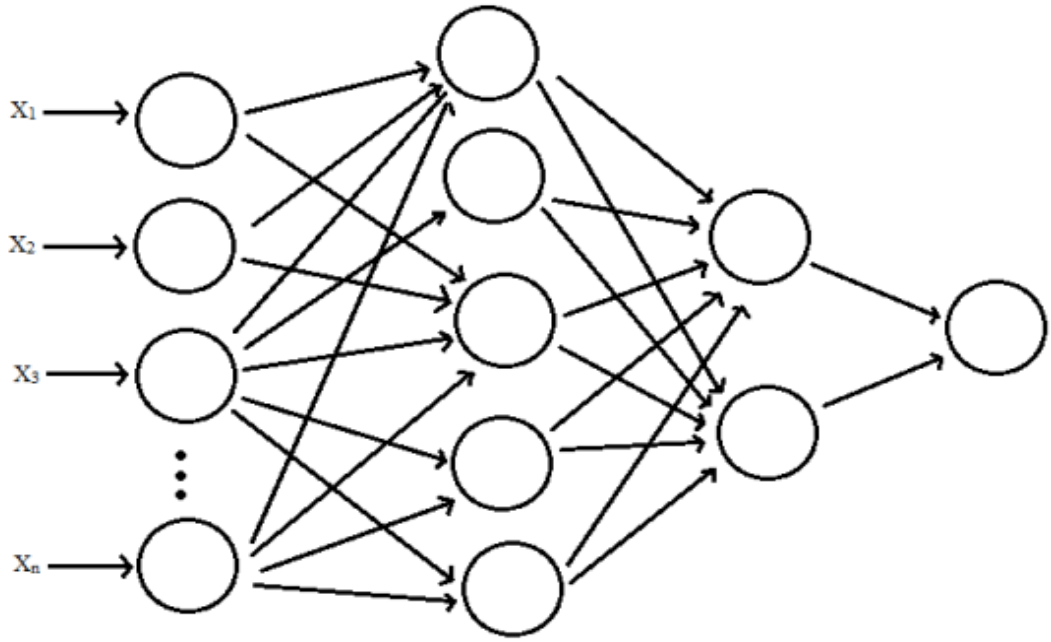

Figure 3.7: Multilayer neural network.

ANNs are divided into supervised and unsupervised. It implements applications using unsupervised ANNs. With these methods, which are rarely used and learned from the data, the narrative is mostly explained in applications such as statistical, clustering, reference separation and compression. In supervised ANNs, however, they need use. The data is divided into test data and training data. The patterns learned from the training data are predicted and made in the test data.

3.1.5 K-Nearest Neighborhood (KNN)

KNN, which performs classification by calculating distance and proximity, is based on the assumption that objects that are close to each other have a high probability of belonging to the same class. It is known to which class all the data in the training set belong. Each test data that does not know which class it will be in is compared with the training data. Similar parts of the test data with the training data are called close neighbors, and labels of similar parts are checked. Here k units with the greatest similarity are determined. The number K is an integer and is usually set to less than 20. The class of the test data is determined by determining the most similar class among the k element classes with the highest similarity .

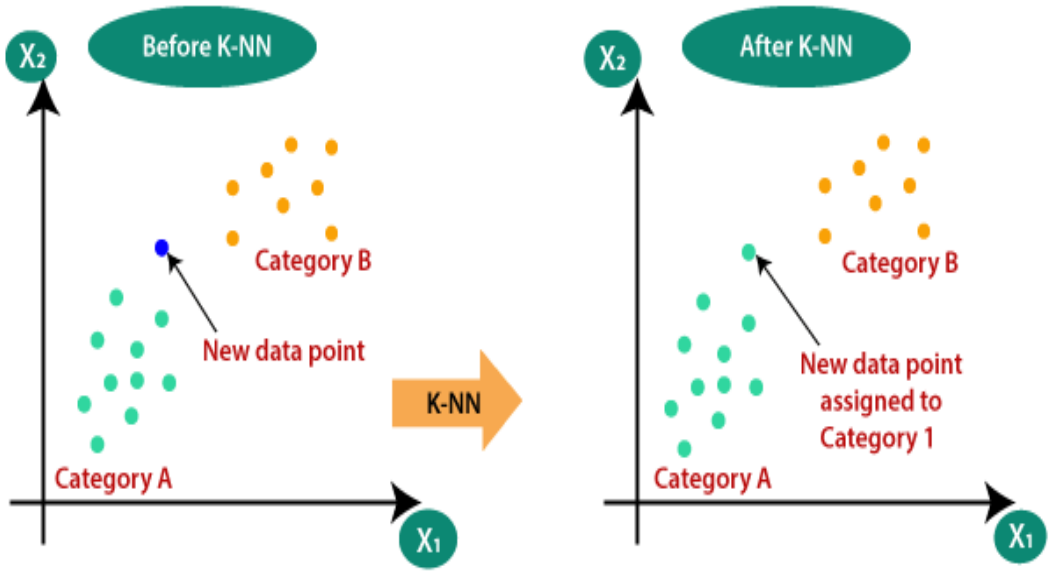

Figure 3.8: KNN.

3.1.6 Random Forest (RF)

It is a supervised machine learning method and is used in classification problems. It has a target variable defined. The decision trees that Morgan and Sonquist put forward in their work in 1963 form the basis of their work. It is formed by the combination of more than one decision tree. It started to be used in classification processes with the regression and classification trees (CART) algorithm in Breiman's study in 1984. It can be used with both continuous and categorical variables. Inputs can be divided into two or more classes. Inputs are classified until each data cannot be separated into different classes. At each stage of this classification process, branches called nodes are formed. Regression and Gini Index are used to create nodes [48].

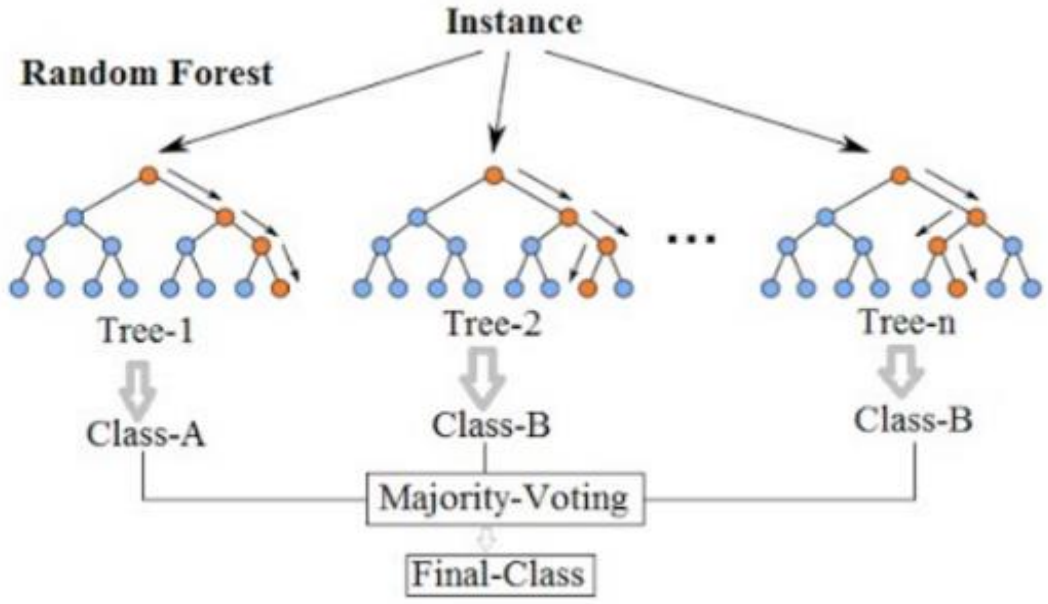

Figure 3.9: RF.

3.1.7 Reinforced Learning

Training data is only provided in return in the form of rewards and penalties for AI agents in a dynamic environment. improved learning; Robotics is a powerful tool for training artificial intelligence models that can help optimize operational efficiency or increase the automation of complex systems such as autonomous driving, manufacturing and supply chain logistics, but

can also be used to directly solve or exploit fundamental problems. . Reinforcement learning works when the desired behavior pattern is not available, but behavior patterns can be evaluated against certain performance criteria. For example, cell phone users find it difficult to get a good signal in areas with poor network coverage. We can think of this easy support learning difficulty as an optimization of an undiscovered prize event R. For a spot x in the realm, R (x) is the incentive that can be achieved for that place (e.g. phone signal strength). The goal of reinforcement learning is to determine the position $x*$ that yields the maximum reward $R(x*)$. Neither the R reinforcement learning system nor the training examples are provided. Instead, we can choose x and see the result $R(x)$.

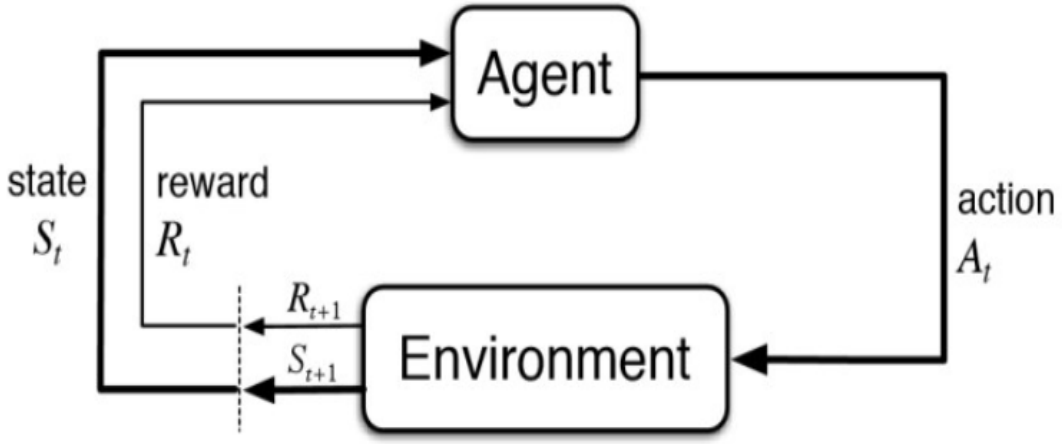

Figure 3.10: Reinforcement learning.

3.2 PROPOSED METHOD

In this study, new method based KNN, and parameter optimization techniques presented in the field of software defect prediction. The proposed method applied ensemble learning with KNN based Gridsearch to present new model for software defect detection. The proposed method consists from several steps to analyses the datasets by training and testing stages.

i. Start the simulation that written using Jupyter notebook. The proposed method executed using Jupyter notebook with python language. The aim of using these tools because of these tools are easy to use and installation.

ii. Read datasets that are used in this study, for example in this research three datasets used CM1, JM1, and PC1. These datasets are contain the features that can train the classifier to detect bugs (defects) in the software projects.

iii. The dataset divided into train and test which is main steps to develop high quality performance models.

iv. Train the KNN by using the training section of the dataset. And the GridSearch optimize the structure of the KNN to present best performance.

v. After the best structure parameters selected by the GridSearch than the test stage started to evaluate and test the performance of the model. This section mathematically represented as shown in equation 3.1.

$$\text{Euclidean } \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2} \tag{3.1}$$

$$\text{Manhattan } \sum_{i=1}^{k}|x_i - y_i| \tag{3.2}$$

$$\text{Minkowski } \left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q} \tag{3.3}$$

These, equation explained the calculating the KNN classifier by calculating the distance between the test point and the example. The y_i represented each feature point in the example dataset. the x_i represented the feature point in the test data. After the finding nearest points to the tested case the K number selected and the class type of the example estimated according to the nearest examples.

vi. In training and testing sections four classifiers in addition to our method is also trained and tested to find the best model.

vii. The accuracy of the models calculated.

viii. Unsupervised LearningThe execution complete after the results presented.

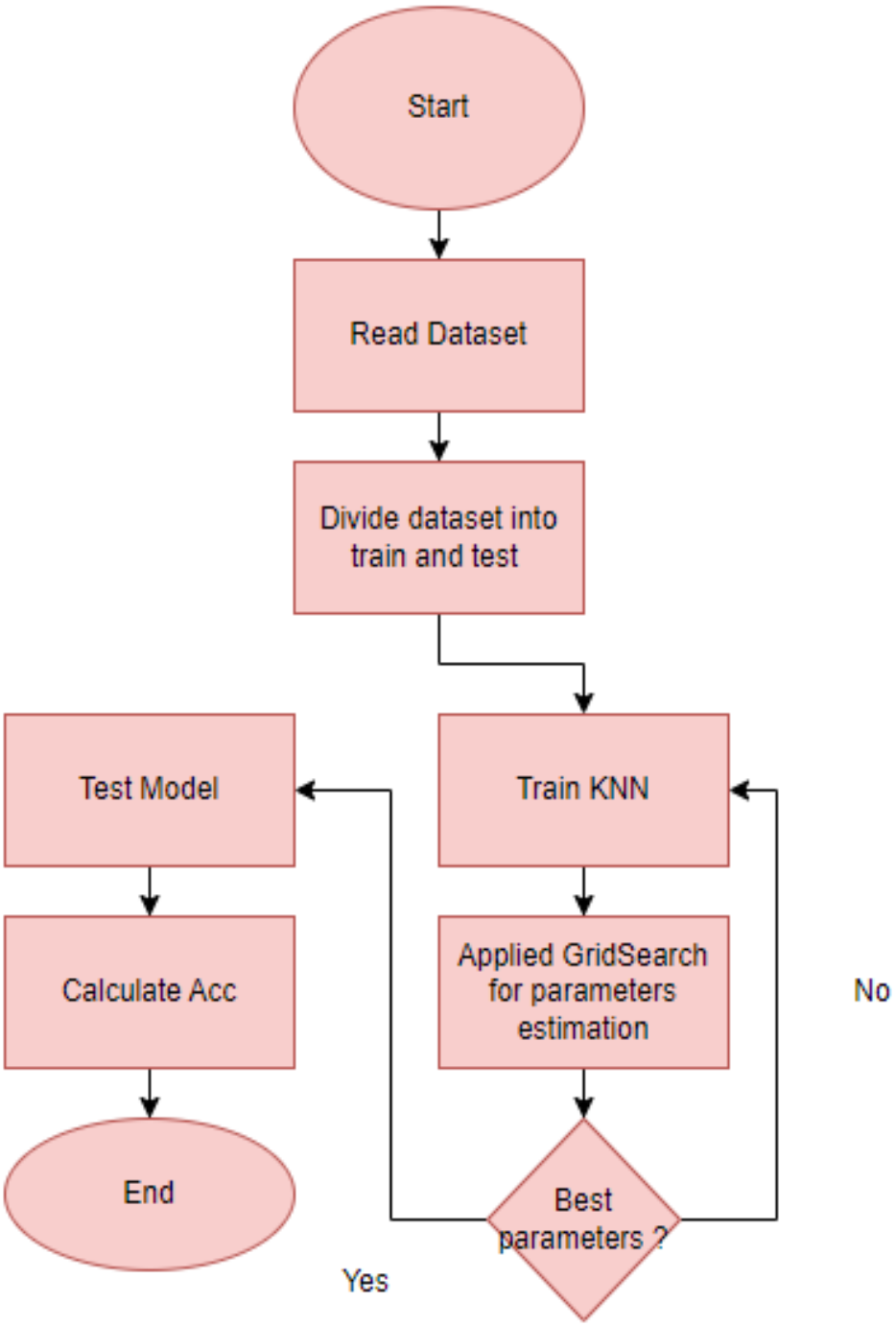

Figure 3.11: Proposed method.

4. EXPERIMENTS AND DISSCUSION

4.1 DATASETS

Datasets are the name given to the structures in which raw data about a subject is stored. The data sets that everyone can easily access are increasing day by day. This includes datasets created for software error estimation. One of them is the publicly available PROMISE dataset owned by space explorer NASA. In this study, CM1, JM1, and PC1 datasets from the PROMISE dataset were used [40].

4.2 CODE EXECUTION OF CM1

In this section several techniques applied with various dataset. The aim of this section is to execute the code with various technique to obtain results as shown below scripts:

In the first stage, the CM1 dataset read, and the features of the dataset presented as shown in the figure 4.1.

```
In [26]: import pandas as pd
         from sklearn import metrics
         db=pd.read_csv('CM1.csv')
         db.head()
```

Out[26]:

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CONDITION_COUNT	CYCLOMA
0	6.0	9.0	2.0	1.0	0.0	16.0	
1	15.0	7.0	3.0	1.0	19.0	12.0	
2	27.0	9.0	1.0	4.0	22.0	16.0	
3	7.0	3.0	2.0	0.0	0.0	4.0	
4	51.0	25.0	13.0	0.0	14.0	48.0	

5 rows × 38 columns

Figure 4.1: CM1 dataset.

In the second stage, the dataset separates to the target and features as shown in the figure 4.2.

```
In [27]: from sklearn.model_selection import train_test_split

         X=db.drop(columns=['Defective'])
         y=db['Defective']
         X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

Figure 4.2: CM1 features and target.

The first classifier applied to classify is the SVM and presented 91% accuracy which is suitable when compared with other studies.

```
In [31]: #SVM
         import matplotlib.pyplot as plt
         from sklearn import metrics
         from sklearn.svm import SVC
         svclassifier = SVC(kernel='poly', degree=8)
         svclassifier.fit(X_train, y_train)
         y_pred = svclassifier.predict(X_test)
```

Figure 4.3: SVM with CM1.

The decision tree is the second classifier applied to detect the defects of the software in the CM1 dataset. the script of this technique presented in the 4.4.

```
In [9]: #DT
        from sklearn.tree import DecisionTreeClassifier
        clf = DecisionTreeClassifier()
        clf = clf.fit(X_train,y_train)
        y_pred = clf.predict(X_test)
```

Figure 4.4: DT with CM1.

Furthermore, the RF applied to classify the same datasets CM1 as shown in the figure 4.5.

```
In [46]: #RF
         from sklearn.ensemble import RandomForestClassifier
         rf=RandomForestClassifier(n_estimators=100)
         rf=rf.fit(X_train,y_train)
         y_pred=rf.predict(X_test)
```

Figure 4.5: RF with CM1.

Moreever, the naïve bayes classifier also applied to software defect detection as shown in the 4.6.

```
In [13]: from sklearn.naive_bayes import GaussianNB
         nb = GaussianNB()
         nb=nb.fit(X_train,y_train)
         y_pred=nb.predict(X_test)
```

Figure 4.6: Naïve bayes classifier with CM1.

```
from sklearn.ensemble import AdaBoostClassifier
ab = AdaBoostClassifier(n_estimators=30,random_state=0)
ab=ab.fit(X_train,y_train)
y_pred=ab.predict(X_test)
```

Figure 4.7: Adaboost classifier with CM1.

Then, the KNN based Gridsearch applied to detect software defects as shown in the scripts 4.8. and figure 4.9.

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

Figure 4.8: KNN classifier with CM1.

```
from sklearn.model_selection import GridSearchCV

leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2]
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
knn_2 = KNeighborsClassifier()
clf = GridSearchCV(knn_2, hyperparameters, cv=10)
best_model = clf.fit(X_train,y_train)
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best leaf_size: 1
Best p: 1
Best n_neighbors: 1
```

Figure 4.9: KNN based gridsearch classifier with CM1.

The results of all classifiers based the CM1 dataset presented in the figure 4.10.

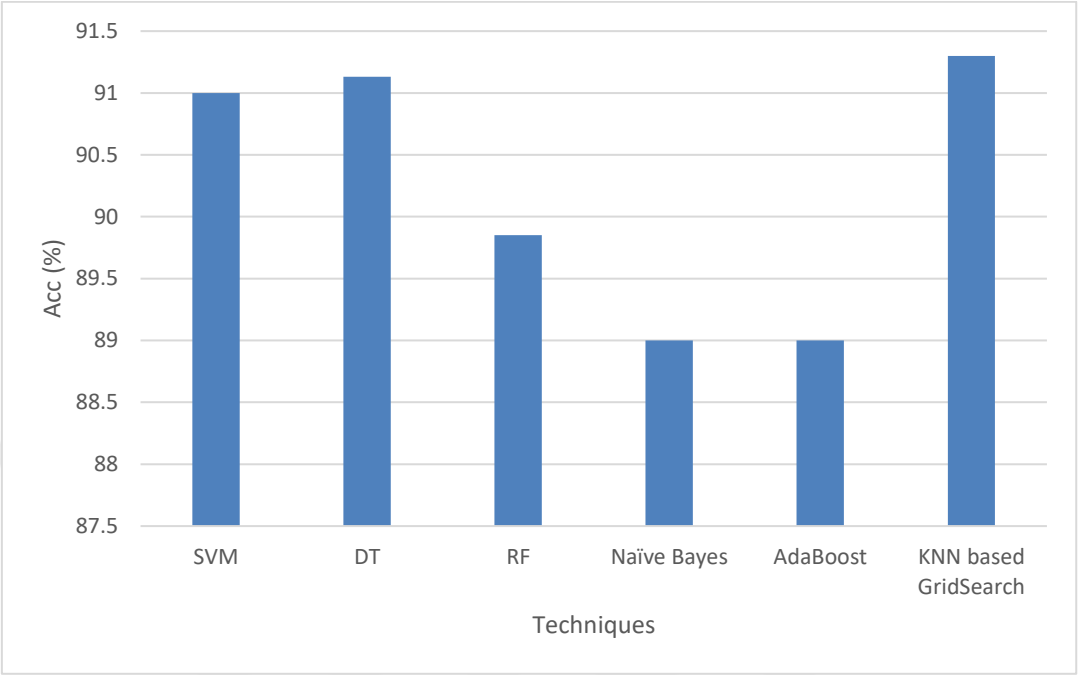

Figure 4.10: Classifier results with CM1.

4.3 CODE EXECUTION OF JM1

In this section several techniques applied with various dataset. The aim of this section is to execute the code with various technique to obtain results as shown below scripts:

In the first stage, the CM1 dataset read, and the features of the dataset presented as shown in the figure 4.11.

```
import pandas as pd
from sklearn import metrics
db=pd.read_csv('JM1.csv')
db.head()
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComme
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	

Figure 4.11: JM1 dataset.

In the second stage, the dataset separates to the target and features as shown in the figure 4.12.

```
In [27]: from sklearn.model_selection import train_test_split

         X=db.drop(columns=['Defective'])
         y=db['Defective']
         X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

Figure 4.12: JM1 features and target

The first classifier applied to classify is the SVM and presented 91% accuracy which is suitable when compared with other studies.

```
In [31]: #SVM
         import matplotlib.pyplot as plt
         from sklearn import metrics
         from sklearn.svm import SVC
         svclassifier = SVC(kernel='poly', degree=8)
         svclassifier.fit(X_train, y_train)
         y_pred = svclassifier.predict(X_test)
```

Figure 4.13: SVM with JM1.

The decision tree is the second classifier applied to detect the defects of the software in the CM1 dataset. the script of this technique presented in the 4.14.

```
In [9]: #DT
        from sklearn.tree import DecisionTreeClassifier
        clf = DecisionTreeClassifier()
        clf = clf.fit(X_train,y_train)
        y_pred = clf.predict(X_test)
```

Figure 4.14: DT with JM1.

Furthermore, the RF applied to classify the same datasets CM1 as shown in the figure 4.15.

```
In [46]: #RF
         from sklearn.ensemble import RandomForestClassifier
         rf=RandomForestClassifier(n_estimators=100)
         rf=rf.fit(X_train,y_train)
         y_pred=rf.predict(X_test)
```

Figure 4.15: RF with JM1.

Moreever, the naïve bayes classifier also applied to software defect detection as shown in the 4.16.

```
In [13]: from sklearn.naive_bayes import GaussianNB
         nb = GaussianNB()
         nb=nb.fit(X_train,y_train)
         y_pred=nb.predict(X_test)
```

Figure 4.16: Naïve bayes classifier with JM1.

```
from sklearn.ensemble import AdaBoostClassifier
ab = AdaBoostClassifier(n_estimators=30,random_state=0)
ab=ab.fit(X_train,y_train)
y_pred=ab.predict(X_test)
```

Figure 4.17: Adaboost classifier with JM1.

Then, the KNN based Gridsearch applied to detect software defects as shown in the scripts 4.18. and figure 4.19.

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

Figure 4.18: KNN classifier with JM1.

```
from sklearn.model_selection import GridSearchCV

leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2]
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
knn_2 = KNeighborsClassifier()
clf = GridSearchCV(knn_2, hyperparameters, cv=10)
best_model = clf.fit(X_train,y_train)
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best leaf_size: 1
Best p: 1
Best n_neighbors: 1
```

Figure 4.19: KNN based gridsearch classifier with JM1.

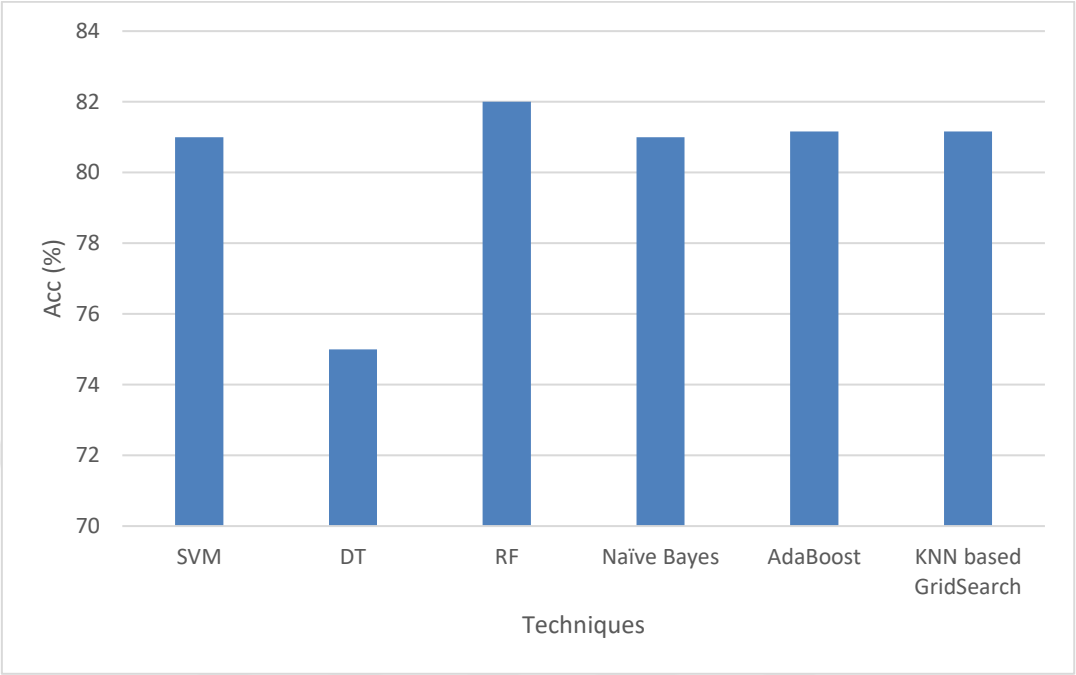

Figure 4.20: Classifier results with JM1.

4.4 CODE EXECUTION OF PC1

In this section several techniques applied with various dataset. The aim of this section is to execute the code with various technique to obtain results as shown below scripts:

In the first stage, the PC1 dataset read, and the features of the dataset presented as shown in the figure 4.21.

```
import pandas as pd
from sklearn import metrics
db=pd.read_csv('JM1.csv')
db.head()
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComme
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	

Figure 4.21: PC1 dataset.

In the second stage, the dataset separates to the target and features as shown in the figure 4.22.

```
In [27]: from sklearn.model_selection import train_test_split

         X=db.drop(columns=['Defective'])
         y=db['Defective']
         X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

Figure 4.22: PC1 features and target.

The first classifier applied to classify is the SVM and presented 91% accuracy which is suitable when compared with other studies.

```
In [31]: #SVM
         import matplotlib.pyplot as plt
         from sklearn import metrics
         from sklearn.svm import SVC
         svclassifier = SVC(kernel='poly', degree=8)
         svclassifier.fit(X_train, y_train)
         y_pred = svclassifier.predict(X_test)
```

Figure 4.23: SVM with JM1.

The decision tree is the second classifier applied to detect the defects of the software in the PC1 dataset. the script of this technique presented in the 4.24.

```
In [9]: #DT
        from sklearn.tree import DecisionTreeClassifier
        clf = DecisionTreeClassifier()
        clf = clf.fit(X_train,y_train)
        y_pred = clf.predict(X_test)
```

Figure 4.24: DT with PC1.

Furthermore, the RF applied to classify the same datasets CM1 as shown in the figure 4.24.

```
In [46]: #RF
         from sklearn.ensemble import RandomForestClassifier
         rf=RandomForestClassifier(n_estimators=100)
         rf=rf.fit(X_train,y_train)
         y_pred=rf.predict(X_test)
```

Figure 4.25: RF with PC1.

Moreever, the naïve bayes classifier also applied to software defect detection as shown in the 4.26.

```
In [13]: from sklearn.naive_bayes import GaussianNB
         nb = GaussianNB()
         nb=nb.fit(X_train,y_train)
         y_pred=nb.predict(X_test)
```

Figure 4.26: Naïve bayes classifier with PC1.

```
from sklearn.ensemble import AdaBoostClassifier
ab = AdaBoostClassifier(n_estimators=30,random_state=0)
ab=ab.fit(X_train,y_train)
y_pred=ab.predict(X_test)
```

Figure 4.27: Adaboost classifier with PC1.

Then, the KNN based Gridsearch applied to detect software defects as shown in the scripts 4.28. and figure 4.28.

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

Figure 4.28: KNN classifier with PC1.

```
from sklearn.model_selection import GridSearchCV

leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2]
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
knn_2 = KNeighborsClassifier()
clf = GridSearchCV(knn_2, hyperparameters, cv=10)
best_model = clf.fit(X_train,y_train)
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best leaf_size: 1
Best p: 1
Best n_neighbors: 1
```

Figure 4.29: KNN based gridsearch classifier with PC1.

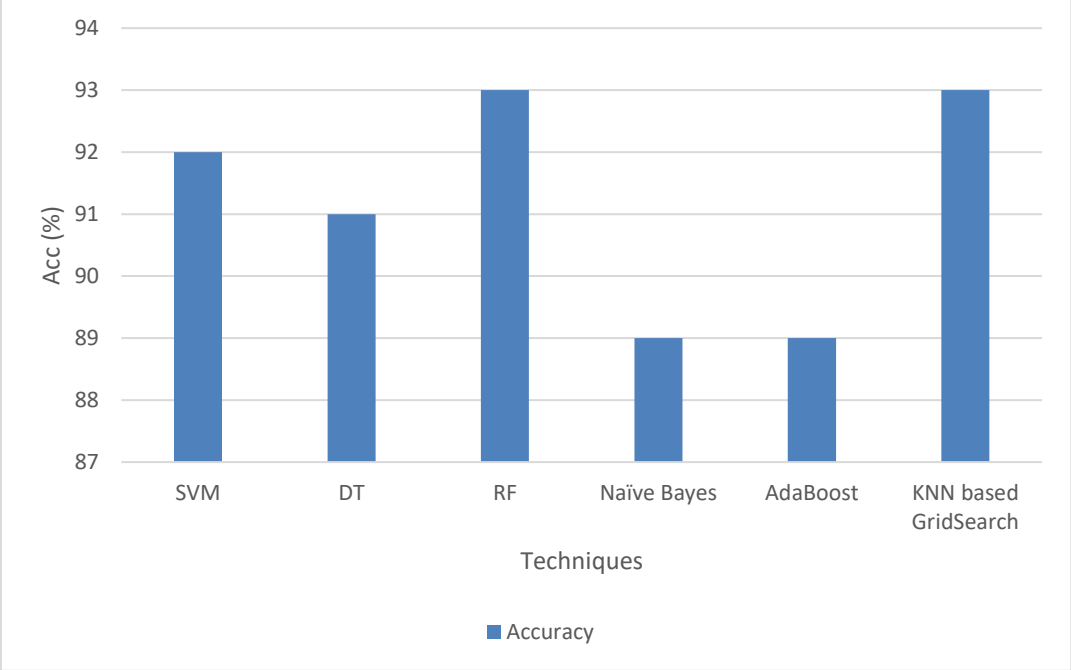

Figure 4.30: Classifier results with PC1.

When analyzing the three datasets show that the proposed method presented suitable results when compared with previous studies. In CM1 dataset the proposed method presented best results than other with more than 90%. In JM1 dataset the all datasets presented low classification rate according to the first dataset. The accuracy rate of the dataset with all classifiers and our method vary between 70%-81% accuracies. The proposed method presented little bit low results from random forest. On the other hand, in the last dataset the proposed method also presented highest results with the random forest. The KNN is effective with low number of dataset and can learn easily and fast. The GridSearch technique is assist the classifier to enhance its classification rate compared with classic KNN.

5. CONCLUSION

Software defect prediction is one of the most important processes in software engineering. Accurate evaluation of bugs in software (especially its modules) is critical to creating bug-free software or software with as few bugs as possible. By using appropriate predictive models, software managers can improve the productivity of their employees. It can spend more time not only on testing, but also on maintaining software systems. Thus, the costs will be at a higher level.

In this study, several classifiers applied to detect software defects and the results compared to select best classifier. In the second part of the applications in the thesis study, the performance of 6 machine learning algorithms in Python language in Jupyter environment on NASA datasets and experimental datasets was evaluated. In this study the K nearest Neighbor algorıthm (KNN) method is a non-parametric classification and regression algorithm. Their ease of use and simple mathematical rationale generalize the use of predictive models in many other areas. It is based on the idea that the outcome of a case is the same as the outcome of the next case. Using a training set based on past observations, determine the dependent variable resulting from each item (observation) in the data set. The estimate of the future prediction is the average of the results of the current case compared to the results of subsequent elements in the training sample. Generally, the nearest observation is defined as the smallest Euclidean distance from the data point in question. The parameters of the KNN estimated using GridSearch to obtain best results. The obtained results vary between 81%-94% which is best of other classifiers.

In the future works, the researcher can used real datasets to evaluate the proposed method or the researcher can present new datasets as new academic studies. Furthermore, ensemble deep learning techniques also can apply as new study. The using several deep learning techniques such as deep belief network, deep sparse autoencoders, deep convolutional neural network, and deep recurrent network, long short-term memory, all these techniques can applied in one study to present new robust model that present best classification rate. Furthermore, feature selection techniques such as k mean and PCA applied to enhance the performance of the classifier to obtain best model that obtain maximum accuracy with low processing time.

REFERENCES

[1] V. Mitra, H. Franco, M. Graciarena, and D. Vergyri, "Medium-duration modulation cepstral feature for robust speech recognition," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 1749-1753: IEEE.

[2] M. A. Ali, "Computer network trafic classification based AI techniques," Altınbaş Üniversitesi/Lisansüstü Eğitim Enstitüsü, 2022.

[3] R. Lotfidereshgi and P. Gournay, "Biologically inspired speech emotion recognition," in *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 2017, pp. 5135-5139: IEEE.

[4] Y. Suh *et al.*, "Development of distant multi-channel speech and noise databases for speech recognition by in-door conversational robots," in *2017 20th Conference of the Oriental Chapter of the International Coordinating Committee on Speech Databases and Speech I/O Systems and Assessment (O-COCOSDA)*, 2017, pp. 1-4: IEEE.

[5] P. S. Nidadavolu, V. Iglesias, J. Villalba, and N. Dehak, "Investigation on neural bandwidth extension of telephone speech for improved speaker recognition," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6111-6115: IEEE.

[6] L. Li, W. Xu, J. Wu, S. He, and X. Li, "The Hokkien isolated word recognition system based on FPGA," in *2014 International Conference on Anti-Counterfeiting, Security and Identification (ASID)*, 2014, pp. 1-5: IEEE.

[7] W. Shu-Guang, Z. Xiang-Yang, and W. Qiang, "Isolated word recognition in reverberant environments," in *2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, 2011, pp. 1-4: IEEE.

[8] S. Sawant and M. Deshpande, "Isolated spoken Marathi words recognition using HMM," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 2018, pp. 1-4: IEEE.

[9] Y. Chen, P.-I. Mak, J. Yang, R. Yue, and Y. Wang, "Comparator with built-in reference voltage generation and split-ROM encoder for a high-speed flash ADC," in *2015 International Symposium on Signals, Circuits and Systems (ISSCS)*, 2015, pp. 1-4: IEEE.

[10] S. C. Sajjan and C. Vijaya, "Comparison of DTW and HMM for isolated word recognition," in *International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012)*, 2012, pp. 466-470: IEEE.

[11] S. Masood, S. Gupta, and S. Khan, "Novel approach for musical instrument identification using neural network," in *2015 Annual IEEE India Conference (INDICON)*, 2015, pp. 1-5: IEEE.

[12] M. Sher, N. Ahmad, and M. Sher, "TESPAR feature based isolated word speaker recognition system," in *18th International Conference on Automation and Computing (ICAC)*, 2012, pp. 1-4: IEEE.

[13] C. Wei and Y. Yang, "Mandarin isolated words recognition method based on pitch contour," in *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, 2012, vol. 1, pp. 143-147: IEEE.

[14] G.-Y. Wang, Y.-M. Zhang, M.-L. Sun, X. Wang, and Y. Zhang, "Speech signal feature parameters extraction algorithm based on PCNN for isolated word recognition," in *2016 International Conference on Audio, Language and Image Processing (ICALIP)*, 2016, pp. 679-682: IEEE.

[15] M. Raczynski, "Speech processing algorithm for isolated words recognition," in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, 2018, pp. 27-31: IEEE.

[16] U. Patil, S. Shirbahadurkar, and A. Paithane, "Automatic Speech Recognition of isolated words in Hindi language using MFCC," in *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, 2016, pp. 433-438: IEEE.

[17] S. Masood, M. Mehta, and D. R. Rizvi, "Isolated word recognition using neural network," in *2015 Annual IEEE India Conference (INDICON)*, 2015, pp. 1-5: IEEE.

[18] M. L. Murthy and G. Murthy, "Isolated word recognition using LPC & vector quantization," *Int. J. Res. Eng. Technol,* vol. 1, no. 3, pp. 479-482, 2012.

[19] L. Xu and M. Ke, "Research on isolated word recognition with DTW-based," in *2012 7th International Conference on Computer Science & Education (ICCSE)*, 2012, pp. 139-141: IEEE.

[20] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering,* vol. 38, no. 6, pp. 1276-1304, 2011.

[21] C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert systems with applications,* vol. 36, no. 4, pp. 7346-7354, 2009.

[22] B. Pal, A. Hasan, M. Aktar, and N. Shahdat, "Cluster ensemble and probabilistic neural network modeling of class ımbalance learning in software defect prediction," in *Artificial Intelligence and Applications*: Citeseer.

[23] İ. B. Aydilek, "Yazılım hata tahmininde kullanılan metriklerin karar ağaçlarındaki bilgi kazançlarının incelenmesi ve iyileştirilmesi," *Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi,* vol. 24, no. 5, pp. 906-914, 2018.

[24] B. Turhan, A. Tosun, and A. Bener, "Empirical evaluation of mixed-project defect prediction models," in *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications,* 2011, pp. 396-403: IEEE.

[25] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," in *NAFIPS 2007-2007 Annual meeting of the North American fuzzy information processing society*, 2007, pp. 69-72: IEEE.

[26] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," 1998.

[27] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE transactions on software engineering,* vol. 33, no. 1, pp. 2-13, 2006.

[28] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering,* vol. 34, no. 4, pp. 485-496, 2008.

[29] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent data analysis,* vol. 6, no. 5, pp. 429-449, 2002.

[30] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter,* vol. 6, no. 1, pp. 20-29, 2004.

[31] S. S. M. Mohammed, "A new computer vision application based on deep learning technique for brain MRI recognition," Altınbaş Üniversitesi/Lisansüstü Eğitim Enstitüsü, 2020.

[32] D. A. Nitz, "Tracking Route Progression in the Posterior Parietal Cortex," *Neuron,* vol. 49, no. 5, pp. 747-756, 2006/03/02/ 2006.

[33] C. Ni, X. Chen, F. Wu, Y. Shen, and Q. Gu, "An empirical study on pareto based multi-objective feature selection for software defect prediction," *Journal of Systems and Software,* vol. 152, pp. 215-238, 2019.

[34] R. Malhotra and S. Kamal, "An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data," *Neurocomputing,* vol. 343, pp. 120-140, 2019/05/28/ 2019.

[35] D.-L. Miholca, G. Czibula, and I. G. Czibula, "A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks," *Information Sciences,* vol. 441, pp. 152-170, 2018/05/01/ 2018.

[36] Y. Shao, B. Liu, S. Wang, and G. Li, "A novel software defect prediction based on atomic class-association rule mining," *Expert Systems with Applications,* vol. 114, pp. 237-254, 2018/12/30/ 2018.

[37] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *Information and Software Technology*, vol. 122, p. 106287, 2020/06/01/ 2020.

[38] Z. Sun, J. Zhang, H. Sun, and X. Zhu, "Collaborative filtering based recommendation of sampling methods for software defect prediction," *Applied Soft Computing*, vol. 90, p. 106163, 2020/05/01/ 2020.

[39] X. Cai, S. Geng, D. Wu, and J. Chen, "Unified integration of many-objective optimization algorithm based on temporary offspring for software defects prediction," *Swarm and Evolutionary Computation*, vol. 63, p. 100871, 2021/06/01/ 2021.

[40] S. Feng, J. Keung, P. Zhang, Y. Xiao, and M. Zhang, "The impact of the distance metric and measure on SMOTE-based techniques in software defect prediction," *Information and Software Technology*, vol. 142, p. 106742, 2022/02/01/ 2022.

[41] C. Jin, "Cross-project software defect prediction based on domain adaptation learning and optimization," *Expert Systems with Applications*, vol. 171, p. 114637, 2021/06/01/ 2021.

[42] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," *Neurocomputing*, vol. 385, pp. 100-110, 2020/04/14/ 2020.

[43] M. Nevendra and P. Singh, "Empirical investigation of hyperparameter optimization for software defect count prediction," *Expert Systems with Applications*, vol. 191, p. 116217, 2022/04/01/ 2022.

[44] H. Wei, C. Hu, S. Chen, Y. Xue, and Q. Zhang, "Establishing a software defect prediction model via effective dimension reduction," *Information Sciences*, vol. 477, pp. 399-409, 2019/03/01/ 2019.

[45] Z. Ding and L. Xing, "Improved software defect prediction using Pruned Histogram-based isolation forest," *Reliability Engineering & System Safety*, vol. 204, p. 107170, 2020/12/01/ 2020.

[46] K. Moshkbar-Bakhshayesh, "Performance study of bayesian regularization based multilayer feed-forward neural network for estimation of the uranium price in comparison with the different supervised learning algorithms," *Progress in Nuclear Energy*, vol. 127, p. 103439, 2020.

[47] L. Han, W. Li, and Z. Su, "An assertive reasoning method for emergency response management based on knowledge elements C4. 5 decision tree," *Expert Systems with Applications*, vol. 122, pp. 65-74, 2019.

[48] X. Wang, C. Zhou, and X. Xu, "Application of C4. 5 decision tree for scholarship evaluations," *Procedia Computer Science*, vol. 151, pp. 179-184, 2019.