T.C.

AKDENİZ ÜNİVERSİTESİ

DESING AND IMPLEMENT MACHINE LEARNING FOR THE SOFTWARE DEFECT PREDICTION

Akim AYENA SOULE AMIDOU

INSTITUTE OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

JUNE 2022

ANTALYA

T.C.

AKDENİZ ÜNİVERSİTESİ

DESING AND IMPLEMENT MACHINE LEARNING FOR THE SOFTWARE DEFECT PREDICTION

Akim AYENA SOULE AMIDOU

INSTITUTE OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

JUNE 2022

ANTALYA

T.C.

AKDENİZ ÜNİVERSİTESİ

INSTITUTE OF NATURAL AND APPLIED SCIENCES

DESING AND IMPLEMENT MACHINE LEARNING FOR THE SOFTWARE DEFECT PREDICTION

Akim AYENA SOULE AMIDOU

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

This thesis was supported by myself

JUNE 2022

T.C.
AKDENİZ ÜNİVERSİTESİ
INSTITUTE OF NATURAL AND APPLIED SCIENCES

DESING AND IMPLEMENT MACHINE LEARNING FOR THE SOFTWARE DEFECT PREDICTION

Akim AYENA SOULE AMIDOU

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

This thesis was unanimously accepted by the jury on 17/06/2022.

Asst.Prof.Dr. Joseph William LEDET (Supervisor)
Asst.Prof.Dr. Hüseyin Gökhan AKÇAY
Prof.Dr. Cesim ERTEN

ÖZET

YAZILIM HATA TAHMİNİ İÇİN MAKİNE ÖĞRENİMİ TASARLARMAK VE UYGULAMAK

Akim AYENA SOULE AMİDOU

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr.Öğr Üyesi Joseph William LEDET

Haziran 2022; 61 Sayfa

Yazılım hata tahmin tekniklerinin temel amacı, yazılım geliştirme değerini engelleyen hataları bulmaktır. Hata tahmin teknikleri, modeller oluşturmak için yazlım geçmişi verilerini kullanır ve bu veriler daha sonra dosyalar, değişiklikler ve prosedürler gibi kod alanlarının yeni oluşumlarındaki hataları tahmin etmek için kullanılır. Doğru tahmin çerçeveleri oluşturmaya yönelik daha önceki çalışmalar tarafından yapılan girişimler genellikle aşağıdaki iki kategoriye ayrılır: birincil taktik, kusurları daha verimli bir şekilde karakterize etmek için fiziksel olarak orijinal özellikler veya yeni özellik birleşimleri tasarlamaktır ve ayrıca ikinci yaklaşım, makine öğrenimini destekleyen yeni ve daha kaliteli sınıflandırma çerçevelerinin kullanılmasını içerir. Mevcut araştırma çalışmasında, özellik çıkarımı için PSO ile birlikte genetik algoritma uygulanmıştır. Torbalama sınıflandırması, nihai sınıflandırılmış sonuçları elde etmek için kullanılır. Bu araştırma çalışması sırasında, konuyu daha iyi anlamak için üç topluluk sınıflandırıcısı oluşturulmuştur. Bu topluluk sınıflandırıcıları özellik azaltma için PCA algoritmasında kullanıldığında, doğruluk daha da artmaktadır. Son aşamada, kategori (sınıf) dengesizliği sorunu çözülür ve bu nedenle doğruluk yüzden 93'e kadar artırılır.

ANAHTAR KELİMELER: Doğruluk, Makine Öğrenmesi, özelliklerin birleştirilmesi, Tahmin çerçeveleri, Topluluk sınıflandırıcıları, Yazılım geliştirme, Yazılım hatası tahmini**.**

JÜRİ: Dr.Öğr. Üyesi Joseph William LEDET

Dr.Öğr. Üyesi Hüseyin Gökhan AKÇAY

Prof.Dr. Cesim ERTEN

ABSTRACT

DESING AND IMPLEMENT MACHINE LEARNING FOR THE SOFTWARE DEFECT PREDICTION

Akim AYENA SOULE AMIDOU

MSc Thesis in Computer Engineering

Supervisor: Asst.Prof.Dr. Joseph William LEDET

June 2022; 61 pages

The fundamental goal of software defect prediction (SDP) techniques (DPTs) is to find errors that hamper software development value (SDV). DPTs employ software history data to construct models, which are then used to anticipate faults in fresh occurrences of code areas such files, modifications, and procedures. The attempts made by earlier studies towards building accurate prediction frameworks are generally divided into the subsequent two categories: the primary tactic is physically designing original features or new amalgamations of features to characterize defects more efficiently, and also the second approach includes the utilization of nascent and better-quality classification frameworks supported machine learning. In the existing research work, genetic algorithm is applied with PSO for the feature extraction. The bagging classification is used to come up with final classified results. During this research work, three ensemble classifiers are generated to better understand the topic. When these ensemble classifiers are used for the PCA algorithm for the feature reduction, accuracy is increased further. In the last phase, the category(class) unbalancing problem gets resolved, and therefore accuracy is increased up to 93 percent.

KEYWORDS: Accuracy, Amalgamations of features, Ensemble classifiers, Machine Learning, Prediction frameworks, Software Defect Prediction, Software Defect Prediction Techniques

COMMITTEE: Asst.Prof.Dr. Joseph William LEDET

Asst.Prof.Dr. Hüseyin Gökhan AKÇAY

Prof.Dr. Cesim ERTEN

ACKNOWLEDGEMENTS

Thank you, Almighty and Merciful God, for bringing this scholarly effort to a successful conclusion. I couldn't help but pray that Almighty God bless my supervisor and advisor, Dr. Joseph William Ledet, for his excellent guidance during the research. My heartfelt appreciation goes to my nearest and dearest for being unwaveringly supportive and prayerful.

My dissertation is dedicated to Heavenly Father, my parents, mates and professors. I owe a special debt of appreciation to my devote parents, whose encouraging words and tenacity still sing in my ears. Dr. Joseph William Ledet, my mentor, has never left my side and is truly exceptional.

This thesis is also dedicated to my bosom friends, who have been extremely supportive throughout the process. I'll always be grateful for what they've done for me.

LIST OF CONTENTS

TEXT OF OATH

I hereby declare that the work presented during this research report entitled "Design and Implement machine learning for the software defect prediction" was administered by me for the degree of master's in Computer Engineering to be awarded by Akdeniz University, Antalya. I've got not submitted the matter embodied during this report for the award of any degree or diploma of any the other University or Institute.

For all words, concepts, diagrams, drawings, computer program, experiments and results that aren't my unique contribution, I've provided due acknowledgment the initial writers.

To identify verbatim sentences, I utilized quotation marks and credited the original creators or publishers.

I certify that no part of my work has been plagiarized, and that the experiments and results presented in the report have not been tampered with. In the event of a plagiarism accusation or the manipulation of experiments and results, I will be held totally liable and accountable

17/06/2022

Akim AYENA SOULE AMIDOU

SYMBOLS (UNITS) AND ABBREVIATIONS

Symbols

A : Attribute (Dataset)

$|a_{i,D}|$: Number of tuples of value a_i

$|C_{j,D}|$: Number of tuples of class C_j

c : Scalar

$|D|$: Tuples of training dataset

D_n : Training sample

d(p, q) : Euclidean distance between p and q

E_Θ : Expectation of random parameters

$f(a)$: Sigmoid function

g^* : Global best solution

H_m : Accuracy of base classifiers

i : Number of samples

m : Number of classes

N(**μ**, **σ²**) : Probability density function

N(estimated) : Estimated Number of defects

P(X, i) : Probability of X given i

R : Estimated Naïve Bayes 'probability

$\overrightarrow{r_n}$: Regression Tree

$t_{k(i)}$: Vectors' transformation

u : M-dimensional Vector

v_i : Particle's velocity

W_i : Expected variance

x_i : Particle's location

Z_i : M-dimensional real vectors

α : Vector weight/Learning parameter

α^2 : Variance

α_0 : Bias

β : Learning parameter

μ : mean

Θ : Randomizing variable

Abbreviations

ABP : Alternating Bit Protocol

BNB : Bernoulli Naïve Bayes

COQUALMO : Constructive Quality Model

DAG : Directed Acyclic Graph

DeP : Defect Prediction

DI : Defect Introduction

DR : Defect Removal

DT : Decision Tree

GA : Genetic Algorithm

GNB : Gaussian Naïve Bayes

KNN : K-Nearest Neighbor

$L(Z)$: Hyperplane Definition

MLP : Multilayer Perceptron

MRT : Monte Regression Trees

NB : Naïve Bayes

NN : Neuron Network

PC : Personal Computer

PCA : Principal Component Analysis

PDF : Probability Density Function

PSO : Particle Swarm Optimization

RF : Random Forest

ROM : Read-Only-Memory

RT : Regression Trees

SRGM : Software Reliability Growth Model

SVM : Support Vector Machine

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The term 'Software Engineer' consists of two words i.e., software and engineering. Software is referred to as much more than just computer program code. A program is a piece of computer code that be executed. The term "Software" can be interpreted as a collection of executable computer code, any required libraries, and accompanying documentation objects. When a Software is built for a particular requirement, it's thus termed as a product or package. At the same time, Engineering's stated to be clarified as a merchandise development through transparent technological fundamentals and methodologies. Thus, the expression "Software Engineering (SE)" will be defined as an engineering discipline focused on developing software products with the assistance of definite scientific principles, techniques, and processes. The output of software engineering is obtained within the type of a competent and consistent merchandise.

SE, according to some scholars, is defined as the methodical design and development of software products, as well as the management of the software process [1]. One of its key goals is to develop projects that fulfill specification, are demonstrably accurate, are delivered on time, and are under "budget". All terms included during this description signify that software development is crucial not just to unify a processor, a programming language, and a programmer, but also to plan a technique which will satisfy some predefined requirements by considering quality, time, and cost. Generally, software engineers perform their tasks by adopting a methodical and systematized approach. This is often also considered because the best thanks to produce first-class software products.

The importance of software engineering can be understood by the following factors:

- At present, a large number of people and society depend on sophisticated software systems. Therefore, it is required to develop consistent and reliable systems quickly and within budget.
- The use of software engineering methodologies and techniques for software systems is quite economical, over the long term, instead of just writing the programs for a personal programming project. Once the use of software starts, the maximum number of costs represents the costs of changing the software for multiple systems.

In the current scenario, the classification of software may be carried out into several different classes. The nature of software in a particular class differs from the software in the other class. This provision creates challenges for software engineers. The different types of software used currently have been described below:

a) **System software**: Infrastructure software is included in this category. Compilers, OSs, editors, and drivers are only a few examples. System software is a collection of applications that work together to provide services to other programs.
b) **Real-time software**: The use of this type is quite common for monitoring, controlling, and analyzing real-time events once they happen. Software used for weather prediction belongs to this category [2]. This type of software collects and processes the data of environmental parameters such as temperature, humidity, etc. for weather prediction.
c) **Embedded software**: It's a sort of software that is sorted in the product's "Read-Only-Memory (ROM)" and is responsible for a variety of functions. The product might be in the form of an airplane, vehicle, security system, signaling system, a power plant control unit, and so on. Embedded software is used to control hardware and is often referred to as intelligent software.
d) **Business software**: The application domain of software placed in this category is the biggest. This software is designed to serve business purposes. This type of software might be workforce, file monitoring system, employee management, and account management. It is also possible to use this software as a data warehousing instrument for getting support in decision-making based on present data. Some well-known examples of this software include MIS (Management Information System), ERP (Enterprise Resource Planning), etc.
e) **Personal computer software**: This sort of software includes software for PCs [3]. Word processors, computer graphics, multimedia and animation tools, database administration, pc games are just a few examples. Because of the high number of customers, this is an extremely important domain, and numerous large-scale organizations are concentrating their effort on it.
f) **Artificial intelligence software**: non-numerical algorithms are used in this area of software to address complex tasks that cannot be handled through computation or straightforward analysis. Some eminent examples include expert systems, artificial neural networks, signal processing software, and so on.

g) **Web-based software**: Web application software is covered in this category. The list of this category of software comprises CGI, HTML, Java, Perl, C, C++, Python, DHTML, etc. [4].

The Software process is a systematic scheme adopted in software engineering. A software process refers to a sequence of tasks that makes possible the development of a product. It consists of all the essential activities needed for developing and maintaining the software products. **Figure 1.1.** represents the general software steps:

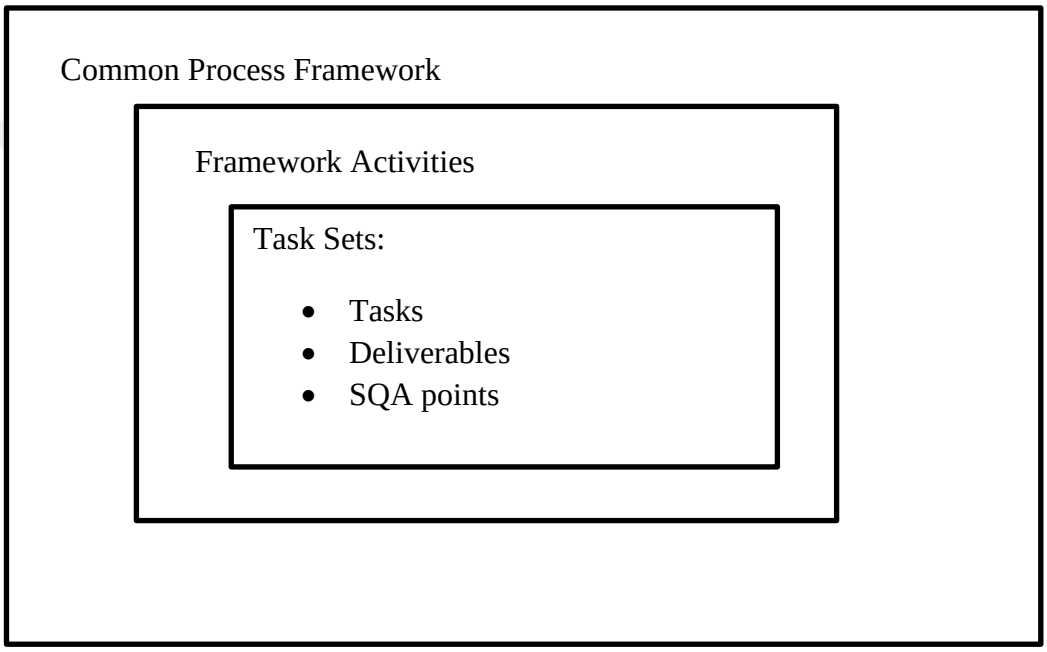

Figure 1. 1. General Software Process

A modest amount of framework activities suited for all software projects, regardless of size or complexity, are established to construct a general process framework. Several task sets, each representing a catalogue of informatics work tasks, key milestones, application work packages and deliverables, and QA comments, permit for changes in the framework activities depending on the features of the project and the needs of the team working on it. In the flow chart, umbrella activities (UAs) that include software QA, software creation management, and measurements are employed. UAs do not depend on a particular framework activity and take place during the process.

In general, all software processes involve four basic steps. These steps are:

i. **Software specification**: In this step, clients and engineers outline how the software is going to be developed along with the constraints on its developing process.
ii. **Software development**: This step is related to the designing and programming of the software.
iii. **Software validation**: In this step, the software is verified to make sure that it meets the customer's requirements.
iv. **Software evolution**: Software modification is carried out in this step for reflecting a change in the requirements of the market as well as the customer.

SD, together with a fundamental component of a program package is additionally a crucial feature of a software quality (SQ). SDs are an inevitable coproduct of the built program [5]. Additionally to the current, the guarantee of SQ assurance isn't straightforward and is time consuming too. There exist alternative means to understand defects, in terms of quality, for example. Nevertheless, the flaws are normally expounded within the discrepancies' style from specs or goals, which elucidate failure in the way they function. Overall, more than 70% of software projects out there lack adequate time and necessary workforce to eliminate all defects before the developed project gets discharged to end users. Because of that reason, the general project quality and possibly the reputation of a corporation delivering the merchandise might be affected severely. In these circumstances, the potentiality of diverse approaches with the power to supply alternatives for standardizing project products gets cavernous. The bugs (defects) also exist altogether project outputs whether it's a tiny low or masterplan computer code. Several defects are often discovered with no tapestry. However, it should be emphasized that some flaws that are profoundly hidden can't be detected effortlessly. Let's admit that some flaws harm less while some cause a large properties' destruction and perhaps threaten lives. Lots of span and labor from programmers, customers, and service personnel, is necessitated from the first designing process to the final word program utilization. Predicting SDs efficiently may be a necessity to ensure the quality of a computer program.

An SD refers to an error in the code or algorithm because of which the software product cannot satisfy the customer's hopes and stated software requirements. The program may generate an unpredicted output or may malfunction because of the error. The fundamental ideas explaining software defects have been expressed below:

- The defect is identified as a bug that is generated within the application as the result of some mistake done by a programmer.
- The deviation in the outcome of a software product from the expected one, defined in the software specification document is termed a defect [6].
- A case, where a software product cannot fulfill or meet the expectations of the end customer may be believed as a defect. This dissatisfaction may be the result of an error in the techniques or the product code in the course of development.

The eye of QA activities towards computer program that's mostly exposed to defects may be diverted by SDP methods. These methods are good candidates in providing assets to find solutions to problem with rigid complexity. We can understand from Defect Prediction (DeP) in "Software" that's the method of detecting portions of "Software System" that will contain flaws. In SE, "Software Defects" (Bugs) prediction may be a research area with a lot of potential. DeP models (DPMs) provide a record of flaw-prone software items that serves QA teams to optimally distribute necessary and ready-to-use means in order to test and inspect software products. The initial application of DPMs in the SE process allows professionals to focus their testing team on areas indicated as "prone to bugs", allowing them to complete testing of those areas with greater certainty than other parts of the product. This feature may cut staff expenses during development and provide some relief from maintenance operations. Two methodologies are used to construct DeP models [7]. For developing a DeP model, the primary methodology is name "Software Metrics", which are measurable characteristics of computer code. On the opposite hand, the second methodology attempts to do so by using fault data from a similar software project. The DeP model will be used in future software projects when it is complete. Experts can spot bug-prone areas of a software package this way.

SDP approaches are divided into categories. The first group of techniques is used to forecast various flaws within the product module/class, whilst the second one is

employed to label the module/class as defected or non-defected. **Figure 1.2.** presents the classification of software defect prediction techniques:

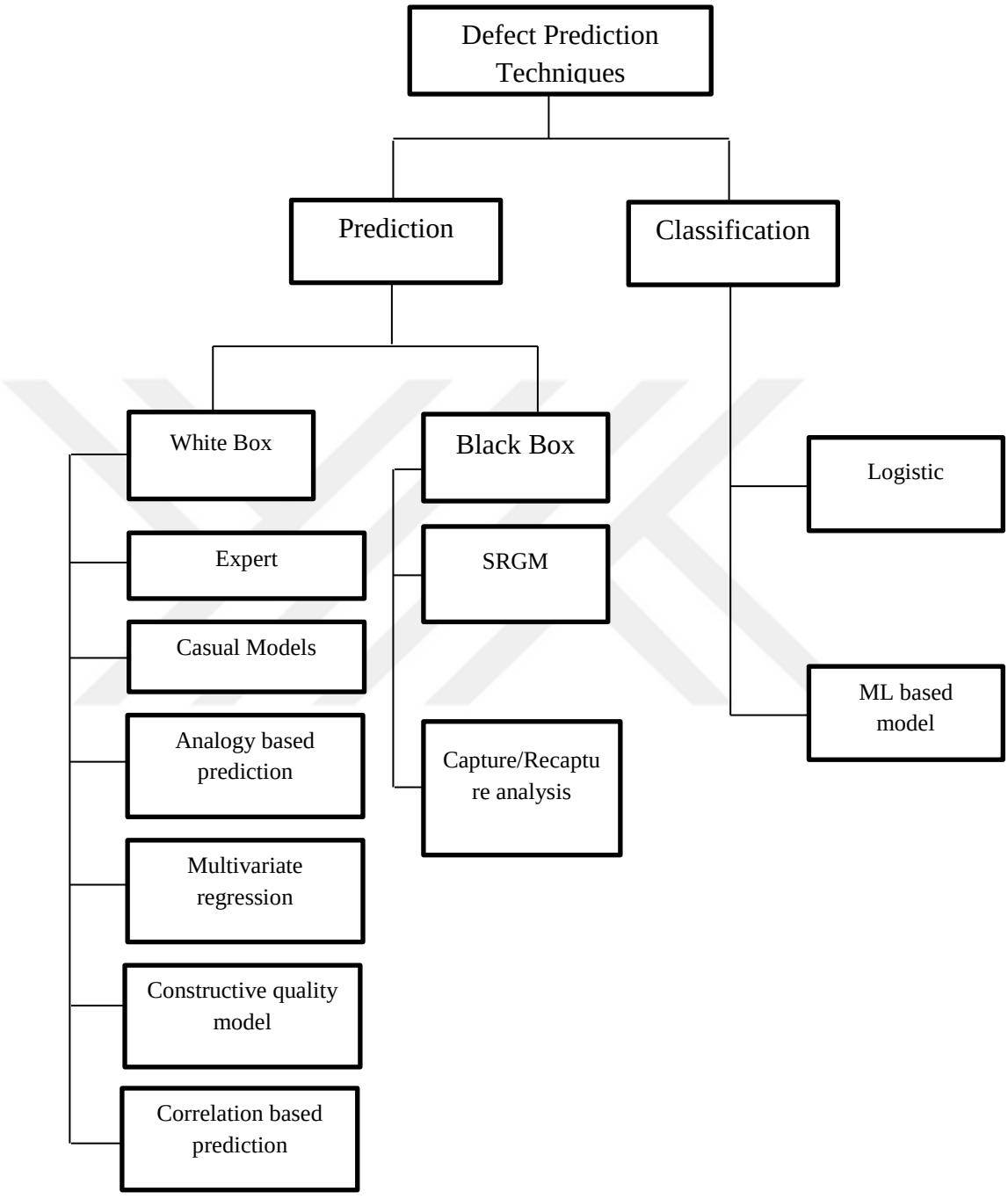

Figure 1. 2. SDP Techniques

There are mainly two categories of methods available that are used for the forecasting of different types of defects. These methods are identified as black-box methods and white box methods. Both methods have been described below:

a. **Black-Box technique**: The defect prediction techniques based on the black-box approach make use of past values of a metric for predicting its future values [8]. For instance, many hidden defects in the product can be predicted utilizing several discovered defects during software development or software testing. These models do not make use of other features of the software for forecasting. The two commonly used black-box methods have been discussed below:

- **Software Reliability Growth Models** (SRGMs): This defect prediction model makes use of some mathematical formulas for estimating the consistency of software systems based on the faults. These models use the defect data obtained from the development and testing stage. First of all, an appropriate model is chosen by considering a testing and development approach. After that, one more sub-model from the chosen model will be adopted for predicting defects. Initially, these models are implemented on the available defect data of the product. Further, partial defect arrival is monitored to fit the mathematical growth model. Lastly, the defects hidden in the software system are predicted using this model.
- **Capture-Recapture Analysis**: This approach makes use of autonomous defect detection activities for analyzing the identified defect patterns. These activities involve inspectors or inspections regarding testing [9]. Afterward, the remaining number of defects is estimated by applying the formula given below:

$$N\ (estimated) in work product = \frac{n(inspector1) \times n(inspector2)}{n(number\ of\ defects\ found\ by\ both\ inspectors)}$$

$$Remaining\ defects(est.) = N(estimated) - N(unique\ discovered)$$

In the case of extremely less no. of inspectors, no model is sufficiently accurate, and underestimation may be substantial.

b. **White Box technique**: The defect prediction techniques based on the white box approach also make use of other product features along with the 'discovered defects'

parameter for predicting various hidden software defects. Some popular defect prediction techniques based on the white box approach have been explained below:

- **Expert Opinions**: The key idea of this technique is based on the expert opinion in case he/she is present for analysis. This defect prediction technique has limited scope due to its subjective nature. This means that the outcomes based on different opinions do not show consistency. The use of this technique is limited to just predicting some defects at the product level. The use of this approach is also not appropriate for predicting defects at the small-scale sub-products or module level.
- **Casual Models**: These models are adopted for establishing a fundamental relationship between software process and product features with several defects that may be discovered in the system.
- **Analogy-Based Predictions**: These defect prediction techniques are based on the gathering and comparison of various parameters amid the previous and the new project to identify the most similar project. The estimation for predicting software defects generally depends on size, type of application, the intricacy of operation, and other metrics. The analysis may be performed at the sub-system/ component level of the project.
- **Multivariate Regression**: These types of models make use of some software metrics predefined as the predictor variable and build the model based on statistical regression so that flaws' prediction can occur in the software. The use of MLRs (Multiple linear regressions) is very common for predicting the software variations through a set of software intricacy parameters in the form of independent variables [10].
- **Constructive Quality Model** (COQUALMO): This model relies on software defect outline and removal models similar to the Tank and Pipe approach. This model is understood as a fused model of DI (Defect Introduction) and DR (Defect Removal).
- **Correlation Based Models**: Like the former models of defect prediction, these models also make use of defect data present throughout the development and testing phases. This model uses several identified defects figured out in

the course of development and testing as an information source for predicting the anticipated number of defects that possibly will occur in the subsequent phase or iteration.

In addition to the approaches that are used for predicting hidden defects, software defect classification approaches focus on classifying the defective and non-defective parts of the software product. These approaches make use of several features of the software product for classifying the software artifacts. In general, defects within the software are classified at the lower level of granularity including file and class levels. The techniques used for classifying defects have been elaborated below:

- **Logistic Regression (LR)**: LR is a classification method that used one or perhaps more independent variables to determine the outcome. The categorization output denotes the value of one of the two probable outcomes. This method differentiates between defective and non-defective components. Like another regression technique, this algorithm uses some metrics information to categorize program pieces. This classification algorithm makes estimations regarding the probability of action of two possible values (binary classification). As a result, if the likely values are binaries (0,1), this approach can be successfully employed for a forecasting. This technique is also recommended to analyze data. It could also be used to interpret the relationship between one dependent binary variable and one or more independent variables [11].
- **Machine Learning (ML) Models**: Data Mining strategies and statistical techniques-bases algorithms are used in ML to classify faults. Using some prediction factors as input information, the models propose categorization results of software pieces as faulty or non-faulty. For the classification of software components, researchers are increasingly turning to ML algorithms. **Figure 1.3.** depicts a general procedure for ML-based SDP.

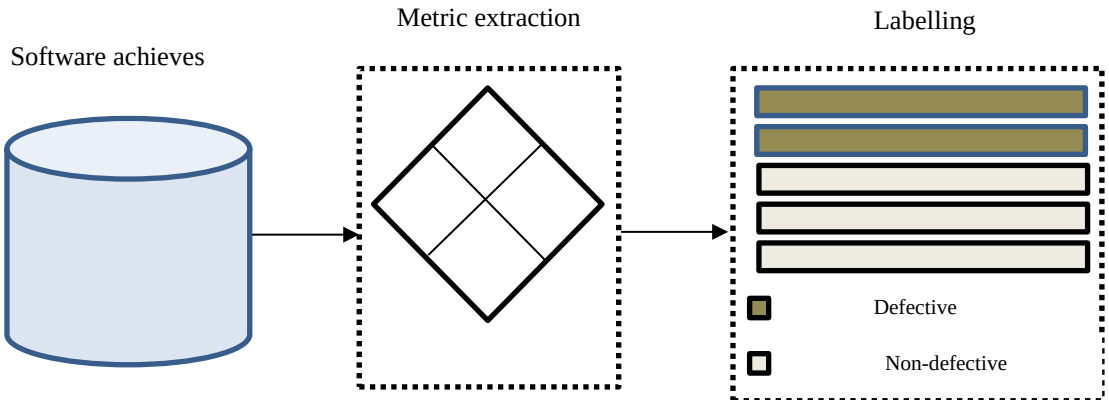

Figure 1. 3. Overview Procedure of SDP

It is necessary to discover data trends in product databases such as issue tracking systems, VCSs, and so on, as shown in the above diagram during the first step in the process of constructing an SDP model. Within the VCSs, the scripts and a few push messages are stored. On the other hand, certain defective information can be found in "Problem Tracking Systems (PTSs)". Each pattern could represent a method, a class, an ASCCII text file, a package, or a code change, according to the prediction granularity. The pattern includes a couple of DeP attributes extracted from software repositories. The worth of the metrics reflects the tapestry of software and its implementation process. A pattern is labeled as defective or non-defective depending on whether or not it contains faults. Following that, a set of teaching patterns is employed to build DeP models that are backed by measurements and labels [12]. Finally, the predictive algorithm could classify a replacement pattern as defective or non-defective.

Following are some well-known MLMs used for SDP:

a) **Naïve Bayes**:

It is a kind of simple Bayesian networks which are composed of DAGs (Directed Acyclic Graph) with only one unobserved node and various observed nodes. This algorithm is based on the hypothesis that all the samples do not rely on other events. Therefore, the NB model is depending on the estimation as:

$$R = \frac{P(i\backslash X)}{P(j\backslash X)} = \frac{P(i)P(X\backslash i)}{P(j)P(XJay)} = \frac{P(i)\Pi P(X_r\backslash i)}{P(j)\Pi P(X_r Jay)}$$

These 2 probabilities are compared, and the class label value is more likely to be the actual label. This algorithm focuses on calculating the probabilities $P(X, i)$using a product operation. Thus, the probabilities of 0 lay impact on it unduly. Laplace estimator is implemented to avoid this issue. For this, all numerators are added, and the number of added ones is added to the denominator. The given hypothesis often faces some errors. Thus, the accuracy acquired from the NB classification algorithm is lower in contrast to other algorithms.

b) Neural Network

A neural network (NN) classifier is predicated on the concept of neurons within the human brain. The working of modules in an exceedingly neural network is that the same as in human neurons. Input, output, and hidden layers make up this model. Neurons classify data. In this technique, math calculations are made. These units communicate through multiple-layer connections. Every link has a weight number. After receiving an input, a neural network processes it using connection weights to generate an output [13]. NN training adjusts connection weight. ABP algorithm trains weights from input to output and vice versa. A gradient-based technique can be used to train an NN by reducing its loss function across a training set.

c) KNN (K-nearest Neighbor)

KNN classifies immediately. This classifier evaluates every outcome. This pattern classifier uses similarity. Most nearby patterns support a pattern's categorization. It considers the classification conclusion for each pattern with the most votes. Euclidean distance is used to measure information patterns. The following formula calculates the distance between neighbors p and q, (p, q). When two points in an extremely Euclidean

k-space are defined as $p = (p1, p2, \ldots, pn)$ and $q = (q1, q2, \ldots, qn)$ in Cartesian coordinates, the distance d between them is determined using the Pythagorean formula:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

d) Support Vector Machine (SVM)

SVM is used for classification and regression. It's generalizable. Despite SVM's large input space, prior knowledge can't be added. It becomes a high-quality classifier. After finding the optimum classification function, the SVM classifier differentiates between two classes in training data. It's also a generalized linear classification. SVM increases geometric margin and reduces classification error. It develops a separation hyperplane after analyzing input data as two sets of vectors in n-dimensional space [14]. Developing two parallel hyperplanes quantifies the margin on each side of the separating hyperplane. Good separation is achieved by keeping both classes' data points far apart. Maximizing margin reduces generalization error. Support vectors and margins help find hyperplanes. Here's the data:

$$\{(Z_1, X_1), (Z_2, X_2), \ldots (Z_i, X_i)\}$$

In which, $Z_i = \frac{1}{-1}$ is a constant and utilized to represent the class to which Z_i belongs where *i* denotes the samples' number. The m-dimensional real vector is illustrated by Z_i. The separating hyperplane is utilized to view the training data, which is described as

$$u.z + c = 0$$

Where u is an m-dimensional vector and c is used as a scalar.

u is perpendicular to the dividing hyperplane and c increases the margin. The hyperplane is passed through the origin forcibly if the c is not present. There is a need for parallel hyperplanes for the maximization of the margin. The equation represents the parallel hyperplanes as:

$$u.z + c = 1$$

$$u.z + c = -1$$

The selection of parallel hyperplanes is done because there are no points among them when training data is separated linearly. The distance among the hyperplanes is quantified $\frac{2}{|u|}$ geometrically. Thus, the reduction of |u| is required. The equation is:

$$u.z_j - c \geq 1 \; or \; u.z_j - c < -1$$

A hyperplane is described properly using the following notation:

$$L(z) = \alpha_o + \alpha^c z$$

In which α weight is a vector and α_o is used to show bias.

The scaling of α and α_o illustrates the optimal hyperplane [15]. One representation is selected from various possible representations of a hyperplane that is described as:

$$|\alpha_o + \alpha^c z| = 1$$

Where the training examples are represented using z which are near the hyperplane and called support vectors. This representation is also considered a canonical hyperplane.

2. LITERATURE REVIEW

The section is going to clarify some notions about Design and Implement Machine Learning for the Software Defect Prediction, the types of defects that may exist in a software, and to review other researchers works on the topic. Additionally, we will be discussing about dataset, implementation in the next section "MATERIAL and METHOD".

2.1. Sample Definitions

a) Software: Collection of executable computer code, libraries, documentation objects

b) Product/Package: a built software for a particular requirement

c) Engineering: product development through transparent technological Fundamentals and methodologies.

d) Software Engineering: an engineering discipline focused on developing software products with assistance of definite scientific principles, techniques, and processes.

e) Defect: is identified as a bug that is generated within the application as the result of some mistake done by a programmer.

f) Software Defect: refers to an error in the code or algorithm because of which the software product cannot satisfy the customer's hopes and stated software requirements.

g) Accuracy: It is about how accurate a software can be. In this current context, an accuracy can ascertained as how near a measured or experimental value is to a standard (true value).

i) Precision: It checks how precise a software can be. In other words, precision is the degree of repeated measurements with the same results under unchanged conditions.

k) Recall: It's concerned about finding **true positives** i.e., it is in charge of recalling found non-defective segment of code.

Defect's Types

Below are some types of defects in Software development[1]:

Performance Defects: When a system or software application fails to deliver the desired or expected results, this is a performance defect. Performance defects occur when a system or software application does not meet the needs of its users. The system's response to changing loads is also included.

Arithmetic Defects: This includes any errors in an arithmetic expression that the developer made, as well as any errors in determining the solution to an arithmetic expression. This type of error is usually caused by the programmer's lack of knowledge or access work. The arithmetic errors may also be caused by code congestion, as the programmer is unable to keep an eye on the written code.

Logical Defects: A logical defect is a flaw in the code that results from an error in the way it was implemented. Defects of this nature occur when the programmer fails to grasp the problem or approaches it incorrectly. Additionally, logical errors can occur during the implementation of the code if the programmer neglects to account for corner cases. It has to do with the software's core.

Syntax Defects: The term "syntax defect" refers to an error in the way the code was written. It also focuses on the developer's mistakes while coding. Syntax errors are frequently introduced by developers in order to catch any small symbols that have been escaped. An escape character like semicolon (;) can be used accidentally when writing Java code.

Researchers Works and Suggestions

Yuchen Guo and colleagues say a strong defect classification capability is needed to predict effort-aware software failures (2018). Six open-source software datasets were utilized to experimentally examine categorization performance and cost-effectiveness curves. Because of the lack of correlation between classification performance and the ability to find acceptable test orderings for finding faults, skewed distributions of change size were closely observed. Trimming enabled all effort-aware approaches that connected strong classification for effort-aware performance. Finally, effort-aware models distributed stress. Trimming was the best solution.

A study by Chun Shan, et al. (2014) evaluated how testers assisted in discovering software faults by accurately predicting the software defect. [17] Data redundancy

[1] https://www.geeksforgeeks.org/types-of-defects-in-software-development/

caused by too many attributes in the defects' dataset affected the accuracy in predicting the defects. In order to deal with the issue, this research proposed an LLE-SVM-based model. As a basic classifier, the Support Vector Machine (SVM) was used in this model. It was decided to deal with the information redundancy by using a locally linear embedding approach that could maintain local geometry. Grid search and ten-fold cross-validation were used to improve the SVM parameters. It was used to test the experimental comparison of locally linear embedding and support vector machines with SVMs. The results showed that the proposed model outperformed the SVM model, and it had the potential to avoid the drop in accuracy that happened because of the data redundancy.

K. Punitha, et. al. (2013) stressed the need of supporting software engineers in identifying software flaws using current software metrics integrated within data processing techniques. As a result, the quality of the program was improved, which led to a decrease in the development and maintenance costs of the software [18]. During this study, particular attention was paid to the found malfunctioning modules. Consequently, prioritizing the flaws needed a thorough analysis of the software's breadth. The developers used the test cases to implement the test cases in the anticipated modules. As a result of this strategy, the program became more reliable since the most critical faults were addressed first, allowing the rest to be addressed in a timely manner. The primary goal of this strategy was to improve the classification accuracy of the information mining algorithm. In order to begin this procedure, the current categorization algorithms were recommended. Another suggestion was to add a degree of fuzziness to the algorithm's hidden layer to improve classification accuracy.

Kwabena EboBennin et al. (2018) studied software fault datasets and proposed MAHAKIL. This method increased chromosome inheritance [19]. These two classes were regarded as parents and a replacement instance was provided, with various features passed down from one parent to the other. The MAHAKIL was compared to 20 defect dataset releases from the PROMISE repository and 5 prediction models. MAHAKIL provided superior and more significant pf values and enhanced prediction performance in experiments. Valid statistical tests were supported.

Fei Wu, et al. (2017) provided a solution for CSDP and WSDP difficulties while anticipating software flaws [20]. SSDL, a good ML algorithm for defect prediction, was recommended for CSDP and WSDP. In semi-supervised structured dictionary learning, relevant information was applied to limited labelled defect data and a large amount of unlabeled data. Experiments used two available datasets. Semi-supervised structured dictionary learning outperformed SSDP techniques in predicting faults in CSDP.

Goran Maua, et al. (2014) demonstrated a technique for collecting scientific data from open-source bug tracking and ASCII text file management repositories [21]. This utility linked information from two sources on the same entity. Interfaces were used to bug and

ASCII text file repositories, and other programs computed software metrics. The user was finally allowed to build datasets for forecasting software defects, despite not understanding all the essential features of this hard work.

Hamdi A. Al-Jamimi et al. (2016) said software defect prediction helped anticipate future module faults [22]. Predictions used software metrics. Prediction metrics and uncertainty were linked. For ambiguity and uncertainty, it was crucial to specify these measurements in language. Prediction models used software measurements and expert judgments as inputs. Various software metrics were used to create a fuzzy logic-based software prediction model. This study offers generic measures for predicting software faults. Important software projects data that used a Takagi-Sugeno fuzzy inference engine in the prediction of software faults were used to authenticate the performance of suggested Fuzzy-based models, leading to encouraging validation results.

Jing Sun et al. (2018) proposed GFKSDP to manage various distributions between source and target domains [23]. An infinite number of subspaces were integrated to characterize changes in geometric and statistical features from source to destination domains using the geodesic flow kernel software defect prediction technique. Adaptive parameter determination reduces computational complexity. The strategy outperformed unsupervised learning studies. Experiments used AEEEM and Relink datasets. The suggested technique improved cross-project prediction. It outperformed unsupervised learning techniques.

Safial Islam Ayon, et al. (2019) suggested GA for attribute choice [24]. Then, PSO generated a cluster of selected attributes. Model training used FNN, RNN, and ANN. Final calculations included accuracy, sensitivity, NPV, F1 score, and MCC. NASA, a software engineering repository, provided 5 datasets for this investigation. DNN was most accurate. Experiments demonstrated the recommended strategy accurately predicted software faults.

Ye Xia, et al. (2014) stressed the usefulness of many process parameters when predicting TT&C software problems [25]. Experiments showed that the mix of CM + MMSLC + HM improved defect forecasting performance, and the history change process metrics improved TT&C software prediction performance. Every process metric's importance was examined for future study.

T P Pushphavathi, et.al (2017) described that many researchers had provided attention to classification/clustering algorithms that had been employed for predicting the software defect [26]. But it had the lack of performance because of utilization of single classifier/clustering algorithms to predict the defects. The integrated approaches were introduced at the place of single classifier/clustering and implemented for predicting the software defects. The results of experiments proved that the soft computing methods had potential to provide superior prediction performance.

Tanvi Sethi, et.al (2016) emphasized on developing high quality software and implemented an ANN approach for predicting the defect [27]. There were nine metrics carried out in the multiple phases of SDLC and executed 20 genuine software projects. A group of organizations was used to gather the software project data and recording of their responses was done in linguistic terms. The MMRE and BMMRE measures were employed to compute the model. The execution of software defect prediction based on NN, and the results obtained from the fuzzy logic basic approach were compared in this research work. The presented approach indicated that the superior and effectual results were obtained on multiple parameters using NN based training models.

Ye Xia, et.al (2014) discussed that an intelligent selection of metrics was the major factor due to which the model performance was affected for predicting the software defects based on metrics [28]. A novel algorithm was intended in which ReliefF feature selection algorithm was integrated with the correlation analysis so that the issue that only the correlation between software metrics was regarded had been resolved and the issue of redundancy was ignored in the existing studies. Three distinct classifiers were employed over classic data sets carried out from the PROMISE repository for conducting the experiment. The comparison was performed with two other popular feature selection algorithms. It was demonstrated in the ANOVA analysis that the performance of defect prediction was enhanced using a novel technique named ReliefF-LC.

Yan NaungSoe, et.al (2018) analyzed that the prediction of the software defect had potential to develop fine quality software [29]. The prediction was performed by applying PROMISE public dataset. The RF algorithm was carried out with the RAPIDMINER machine learning tool. The evaluated performance was compared with the distinct number of trees in Random Forest. Consequently, the accuracy was slightly maximized with the increasing number of trees. The maximum accuracy was evaluated 99.59 and the minimum was evaluated 85.96. The other comparison was performed based on AUC curve in which the most informative indicator related to accuracy of prediction was demonstrated in the field of software defect prediction. The efficiency of the Random Forest algorithm for the prediction was proved in the results and it proved more appropriate when hundred trees were employed in the Random Forest algorithm.

Md FahimuzzmanSohan, et.al (2019) proposed multiple project data which were carried out for developing a balance and an imbalance dataset [30]. Then, the implementation of this dataset was done in various prediction models along with 8 different classifier algorithms. The one balanced and imbalanced test datasets were utilized to cross-check the trained models. The performance of the models was computed by considering the 5-evaluation metrics. It was demonstrated in the experimental outcomes that there was not any prominent change evaluated in balanced as well as imbalanced training models. The maximization in scores of AUC had been seen concerning balanced training models along with imbalanced test datasets. The considerable increase was also observed in the

accuracy and AUC score in the same training model. But the classification model was developed from historical data of multiple projects that assisted to cover up broadly this study.

Ruchika Malhotra, et.al (2014) emphasized on the capability for prediction of the evolutionary computation and hybridized evolutionary computation methods in predicting the defect [31]. The efficiency of the fifteen computation methods was investigated on five datasets which was carried out from the Apache Software Foundation through the defect collection and reporting system. The evaluation of the outcomes was done based on values of accuracy. The Friedman ranking was employed for comparing the evolutionary computation systems. The outcomes represented that the best performance had been acquired from defect prediction models which were developed through the evolutionary computation techniques performed on all the datasets concerning prediction accuracy.

He Can, et.al (2013) described that the consistency of software was enhanced, and development costs were decreased using software defect prediction [32]. The lower prediction accuracy was obtained through conventional prediction models. A novel model named P-SVM was presented for predicting software defects in which PSO and SVM were carried out. SVM's non-linear computational power and PSO's ability to optimize parameters were taken for dealing with this issue. First, the PSO algorithm was implemented in the P-SVM model for the computation of the best SVM's parameters. Afterward, the optimized SVM model was adopted for predicting the software defect. The software defects were predicted on the JM1 dataset as an experiment using the P-SVM model and 3 other prediction models. The results demonstrated that a superior predictive accuracy was obtained from the suggested model as compared to BP-NN model, SVM model, GA-SVM model.

Biwen Li, et.al (2014) advocated a scenario-based technique for the prediction of software problems using the compressed C4.5 model [33]. Using a k-medoids technique to cluster functions after generating functional scenarios and the C4.5 model to anticipate when a defect would occur in those scenarios was the primary goal of this approach. An experiment was performed for the computation of scenario-based approach and its comparison was done with file-based prediction approach. The outcomes of experiments showed that high performance was obtained from the scenario-based approach after the reduction of the size of the DT with 52.65% on average and the slight maximization of accuracy.

Taek Lee, et.al (2016) analyzed that the developer behavior was not regarded in the defect prediction metrics although software quality was impacted due to the developer behavioral interaction patterns [34]. Thus, the MIMs, metrics to leverage developer interaction information had suggested. The Mylyn which was an Eclipse plug-in employed to capture and store the developer interactions in which file editing and

browsing events were included in task sessions. It was proved in the experimental evaluation that overall predictive accuracy was enhanced using micro interaction metrics integrating with existing software measures, better performance for the cost-effective manner was achieved and intuitive feedback was offered due to which developers facilitated for identifying their own inefficient behaviors under software development.

ChakkritTantithamthavorn, et.al (2016) presented a different way of looking at the data from Shepperd et al. [35]. It was indicated that a strong association was shared with other explanatory variables using a research group at first. Secondly, the differentiation of effect of the research group has become hard on model performance due to the strong connection among these explanatory variables. Thirdly, when the effect of this strong connection was reduced, it was discovered that a lesser effect occurred due to the research group as compared to the metric family. It was indicated in the observations that there was more probability in the association of the research group with the defect prediction model's performance as the researchers had a tendency for reusing the experimental elements. This paper proposed that a broader selection of datasets and metrics were employed to conduct experiments by researchers for combating any potential bias in their outcomes.

Yu Qu, et.al (2018) suggested a new network embedding approach called node2vec that was employed to automatically encode the dependency network structure into low-dimensional vector spaces in order to improve SDP [36]. An initial CDN was created. Node2vec was used to automatically learn the structural features. When the learnt features were included into normal software engineering features, they were able to accurately forecast the error. This method was calculated using a total of fifteen open-source apps. It was indicated in the results that the existing approach was enhanced using the node2 defect by 9.15% concerning F-measure.

Chun Shan, et.al (2015) advocated a novel model for predicting software defects which was planned based on ILLE-SVM [37]. The coarse-to-fine grid search algorithm was utilized in the suggested model for searching the optimal parameters. The parameters' great degree of precision was ensured and time for the optimization of parameters was decreased when the search scope was constricted, and the parameters step was enlarged. A performance bias affected the Support Vector Machine in classification due to the unbalancing of datasets. Thus, a grid search algorithm was carried out so as the reasonable weights related to different classes had been set in an automatic way. There were 4 NASA defect data sets implemented to compare the LLE-SVM with the improved Locally Linear Embedding and Support Vector Machines model. The optimal parameters were searched more quickly, and a superior performance was obtained using the suggested model as compared to LLE-SVM in SDP.

Yan Zhou, et.al (2019) recommended a model to predict the software defect based on KPCA-SVM. First, the dimension reduction pretreatment was implemented for the software defect datasets [38]. Afterward, SVM was carried out in the classification. The global features were carried out while choosing the dimension reduction algorithm for the enhancement of accuracy of model. Thus, the dimensionality was reduced by selecting KPCA algorithm. The NASA MDP dataset, widely used in the field of predicting software defects had been implemented for determining the model's performance. The comparison of KPCA -SVM model was performed with CM1, JM1, PC1 and KC1 dataset using single SVM and LLE–SVM. This paper demonstrated that the issue of data redundancy in defect prediction data set was resolved using kernel principal component analysis-SVM model keeping the global benefits and acquiring superior prediction precision.

Fuqun Huang, et.al (2018) analyzed that one of the fundamental reasons for software defects was the understanding as well as the prediction and prevention of software defects [39]. But research was done for the prediction of software defects based on cognitive nature of defects. A design to predict the software defects was suggested in this paper in which the current scientific understanding related to human error mechanisms had been carried out. This novel technique was dependent on the major causal mechanism that dealt with a sequence of human error modes through error-prone scenarios. The fundamental evidence for this idea was also provided in this paper.

Fei Wu, et.al (2018) focused on providing a unified and effective solution to deal with CSDP and WSDP [40]. The semi-supervised dictionary learning method was developed and the CKSDL approach was recommended in this paper. This recommended strategy was used to collect the small amount of labeled defect data and the vast amount of unlabeled data. It was also considered in the proposed technique to dictionary learning that misclassification costs were considered. Extensive experiments were carried out on sixteen projects. The results showed that the cost-sensitive kernelized semi-supervised dictionary learning had a better performance than current WSDP strategies that used unlabeled cross-project defect data to enhance the WSDP performance. In the CSDP situation, the proposed strategy outperformed SSDP techniques.

Ling Xu, et.al (2018) proposed a misclassification CSDP approach [41]. There were two aspects of this new proposed approach. CSDP initially combined unsupervised sampling with a domain-specific misclassification cost model to handle unlabeled software detection datasets. Second, CSDP built a cost-sensitive SVM model to forecast defect-proneness of the remaining modules, using overall classification error rate and domain-specific misclassification cost as quality measures. CSDP used 4 NASA missions. Experiments used real-world software datasets. CSDP outperformed semi-supervised learning algorithms that neglected F-measure rate imbalance classification costs. With less labelled data, the CSDP outperformed previous supervised learning models.

Bharti Bisht, et.al (2019) discussed that Software defect prediction models had acquired significant attention to obtain highly reliable software systems [42]. The faults were discovered at the initial stage using these models that assisted to remove these faults for generating effectual and reliable software. Some defect prediction models based on several data mining approaches were reviewed in this paper. The neural networks concept that was regarded as the rapid emerging approaches in defect prediction models.

Lu Liu, et.al (2015) analyzed that one of the considerable researches was SDP in the area of product reliability [43]. The link between software defect forecasting attributes affected classifier performance. A fuzzy integral technique based on MIFI was suggested for defect prediction to reflect attribute priority and feature interaction. This algorithm determined the fuzzy measure set function based on attribute information and interaction information. Experiments showed that the suggested method outperformed 3 other strategies.

Jiaxi Xu, et.al (2019) After introducing GitHub, an open-source software hosting platform, he emphasized software defect data analysis and classification [44]. A defect data gathering technique using open-source software pull requests was provided. This technology was planned using GitHub and Git. In addition, software defect data preliminary treatment was improved, and a big dataset was created by gathering fix-inducing change and contextual information of defects to address class imbalance. Using this technique, a GitHub-based platform was created to automatically collect software defect data. Experiments determined the application's efficiency, accuracy, and validity. The results showed the technique's efficacy.

Md. FahimuzzmanSohan, et.al (2019) focused on discovering the performance within imbalance and balance data set and for finding the dissimilarity among the performance of single classifier and aggregate classifier [45]. The precision, recall and F-1 score were evaluated in the prediction of software defects, gathering 8 datasets, and applying 7 algorithms and hard voting. Two sets were almost balanced in this gathered data. The conversion of balanced datasets had been done into imbalanced sets and considered as the average non-defective and defective ratio of the other six datasets while investigating. The results of the experiment demonstrated that the lower performance had been obtained from two balanced datasets in comparison with other 6 datasets. However, the performance of 2 datasets was maximized when these sets were converted. It was also observed in the acquired results that the precision was evaluated to 0.92, recall was evaluated to 0.84 and F1-Score for voting was 0.87 that were considered superior as compared to another single classifier.

Ye Xia, et.al (2013) described that an intelligent selection of metrics acted significantly for the advancement of the model performance in prediction of defect based on metrics [46]. Various ways to select feature and dimensionality reduction had been carried out

in this paper for determining the most important software metrics. The SVM, NB and DT were distinct classifiers that had been employed. The NASA dataset which was publicly available had been implemented to perform experiments. The results proved that less than ten metrics had achieved superior performance rather than 22 or more metrics.

Xuan Huo, et.al (2018) introduced CAP-CNN, a deep learning algorithm that uses code comments to identify software defects [47]. The issue of missing comments was generated by a unique training technique in CAP-CNN in which comments information was encoded and monitored by the network to automatically extract semantic features during training, which did not require any testing modules for containing comments. Experiments on widely available software data sets revealed comment features could improve defect prediction.

Kang Yang, et.al (2018) analyzed that the defect data set was utilized in software defect prediction for developing a predictive model in which software defect metrics were compiled in the dataset [48]. Afterward, the potential defect program modules were forecasted in the project using a predictive model. Fourier expression of Boolean function was implemented for creating a model to predict defects in this paper. The Fourier coefficients were computed and the predicted function that assisted in predicting defects, obtained with the help of algorithms. The Fourier learning algorithm was contrasted with the conventional ML algorithm named RF algorithm. At last, the results indicated the better efficiency and stability of the Fourier learning algorithm.

Mohamed Samir, et.al (2019) presented a deep neural network for developing a software defect prediction model and this model was compared with other ML algorithms named Random Forest, DT and NB networks [49]. In the comparison, lower improvement over the other learning models had been demonstrated in the outcomes. The value obtained to predict defects through deep learning had been shown in these results and provided opportunity to more experiments. For the future work, more enhanced deep learning methods would be developed, and more datasets would be discovered from different resources. Furthermore, some feature generation methods would be applied for carrying out the features which assisted to enhance the accuracy of the model.

Kazuya Tanaka, et.al (2019) presented an auto-sklearn, a recently developed software library that was employed in machine learning automatically as it had potential for selecting appropriate prediction models, hyperparameters and data preprocessing methods in automatic manner for a given data set and creating their ensemble using optimized weights [50]. The efficiency of auto-sklearn was computed to predict the number of defects in software modules. The software metrics in which 20 OSS projects contained had been implemented to foresee cross-release defects. The Norm auto-sklearn was employed as a performance measure for comparison with RF, DT, and

linear discriminant analysis. Consequently, the same predictive performance had been achieved from the auto-sklearn as RF, which was the finest prediction models to predict defects in previous studies. This

demonstrated that the good prediction performance had been obtained from auto-sklearn while predicting defects without any knowledge of ML methods and models.

Ebubeogu Amarachukwu Felix, et.al (2017) advised using defect acceleration predictor factors and determining their link with defect count [51]. Based on these predictive variables, an integrated ML strategy was reported. 10 PROMISE datasets were used to run 22838 experiments. The model's R-square was 98.6%. The blueprint was obtained utilizing the recommended way to test the application, improving software development efficiency.

3. MATERIAL AND METHOD

This section will be responsible for formulating the problem firstly. Second, we will elaborate the main goals fort his research work. Third, we will describe the existing methodology. Forth, we will propose our own methodology. Fifth, Dataset used in this study will be explained. Sixth, we talk about implantation that gives the obtained results.

3.1. Problem Formulation

This study predicts software bugs (defects). Pre-processing, feature extraction, and classification are SDP processes. During pre-processing, missing and redundant data are removed from the dataset. Prediction accuracy is good when the dataset is preprocessed well. In the second phase, feature extraction, attribute and target sets are related. In the final phase, classification algorithms classify data. Improved genetic algorithm and PSO are used for feature extraction in prior studies. Classification uses bagging with SVM. This approach can't fix sophistication unbalancing and is inaccurate. The unique technique is proposed to accurately predict software defects.

3.2. Objectives

This research has four goals listed below: -

1. To review and analyze various software defect prediction techniques.

2. To implement bagging with the SVM technique for the SDP techniques.

3. To propose an innovative scheme for the SDP.

4. To implement the proposed scheme and compare it with existing schemes for accuracy, precision, and recall.

3.3. Existing Methodology

The SDP and its techniques are the staple of this research. The SDP involves three stages: pre-processing, feature extraction, and fault classification. In the previous study, the feature extraction and bagging approaches with the use of "SVM" were employed using improved GA and PSO algorithms.

3.3.1. Particle swarm optimization (PSO)

In 1995, Kennedy and Eberhart modeled the PSO on natural swarming behavior, such as that of fish and birds. In essence, the particle's location and velocity represented as x_i and v_i is updated as:

$$v_i^{t+1} = v_i^t + \alpha\epsilon_1[g^* - x_i^t] + \beta\epsilon_2[x_i^* - x_i^t] \quad (1)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (2)$$

In this, two random vectors are represented as $\epsilon_1$and ϵ_2 and attained the value amid 0 and 1 for each entry. The learning parameters are represented with α and β which are obtained as α ≈ β ≈ 2. A pattern-search-type mutation produces the novel location while the current global best solution is implemented to choose in and implicit way g^* found so far and using the individual best x_i^*. Even if the current global best appears to be highly important for selection, the individual best does not act as clearly, as indicated in the accelerated PSO.

Therefore, mutation and selection are involved in particle swarm optimization. It does not contain any crossover that implies high mobility which is included in particles along with a high degree of exploration in PSO. But the g^* is utilized as strongly discerning which is like a double-edged sword. The main benefit of PSO is that it assisted in boosting the convergence after drawing toward the present best g^*, whereas premature convergence occurs due to it simultaneously, although this is not the true optimal solution to deal with the problem of interest.

3.3.2. Genetic Algorithm (GA)

The GA is a stochastic worldwide search technique in which the metaphor of natural biological evolution is impersonated. The GA is an adaptive technique that is carried out for solving the search and optimization of problems. There are four stages involved in this algorithm. In general, the encoding of Genotypes of individuals is done as chromosome-like bit strings.

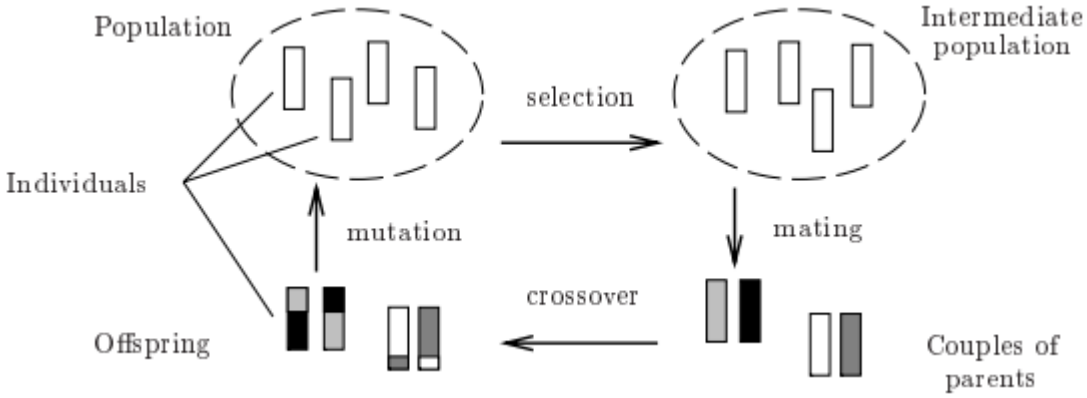

Figure 3. 1. Four phases of genetic algorithm

First, individuals are created. Then, individuals are chosen based on fitness to create an intermediate population. It's an allocation of reproductive opportunities in which a person's higher fitness value is preferred for filling the intermediate population. This population is emptied in pairs. Each pair has a crossover operator with probability P_c.

It's written as two generations exchange information. This operator mates the given ones to produce two new ones. To illustrate, two-bit strings are cut through a random one-point crossover and recombined by swapping their ends. This creates two-bit strings. This new population is intermediate. Without the crossover operator, a few individuals are added to the novel intermediate population. Selection and crossover generate the Genetic algorithm's cooperation stage. Mutation operator adds noise to the population. Randomly modifying some people achieves this. The self-adaptation stage of the Genetic Algorithm prevents early population convergence. It's created using the mutation operator that flips a randomly chosen bit of a bit string. Each individual's operator is implemented using P_m.

The intermediate population replaces all or part of the preliminary population. The intermediate population is the same size as the premature population, but the Genetic Algorithm renews it in one generation. The intermediate population is much smaller than the premature population, so the Genetic Algorithm is not in a steady state. Offspring aren't always swapped with their parents. They can replace anyone, though. A predefined number of generations completes the execution.

3.3.3. Bagging

Bagging is a scheme utilized to enhance the outcomes of machine learning classification algorithms. Leo Breiman developed this technique, and its name was derived from the phrase bootstrap aggregating. In case of classification, there are two possible classes, a classifier $H: D \rightarrow \{-1,1\}$ is generated using a classification algorithm on the base of a training set of illustration descriptions D. A sequence of classifiers $H_m, m = 1, \dots, M$ is generated through this technique in terms of modifications of the training set. The integration of these classifiers is done into a compound classifier. The prediction of the compound classifier is provided as a weighted mixture of individual classifier predictions:

$$H(d_i) = sign\left(\sum_{m=1}^{M} \alpha_m H_m(d_i)\right) \quad (3)$$

The meaning of the above-mentioned formula can be interpreted as a voting process. An example di is categorized into the class for which the majority of classifiers vote. Parameters α_m, $m = 1, \dots, M$ are investigated in this way so that more precise classifiers can have a strong effect on the final prediction as compared to less precise classifiers. The accuracy of base classifiers H_m is only a little bit superior in comparison with the accuracy of a random classification. Thus, these classifiers Hm are categorized as weak classifiers.

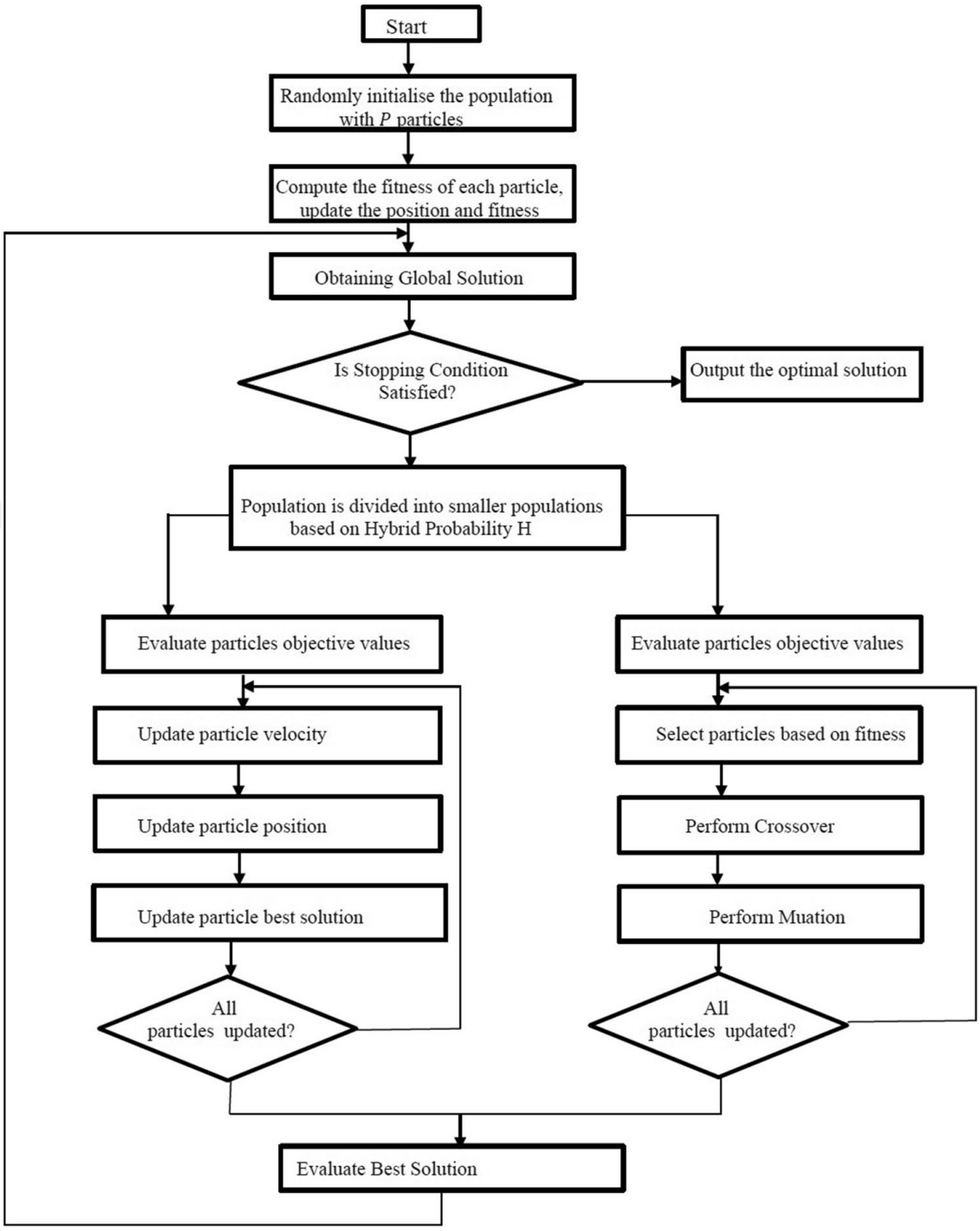

Figure 3.2. Existing Methodology Flow chart

3.4. Proposed Methodology

RF, GBN, BNB, MLP, DT, and SVM are some of the classifiers used in the suggested idea. For SDP, the ensemble {1,2,3} classifiers are created. GNB, BNB, RF and C4.5 make up the core ensemble i.e. ensemble1 classifier. Combining GNB, BNB, RF, and

SVM form the ensemble2 classifier. The merge of GNB, BNB, RF, and MLP is the last ensemble classifier (ensemble3). The feature extraction approach known as “Principal Component Analysis (PCA)” is used with those three ensemble classifiers. The main points of every classifier are explained below: -

3.4.1. Support Vector Machine (SVM)

SVM can do classification and regression using supervised learning. It improves generalization. Even with a large input space, prior knowledge isn't required. So, it's a good classifier. This classifier differentiates two classes in training data. Here, the best classification function is found. It's linear generalization. SVM increases geometric margin and reduces classification error [14]. It constructs a separation hyperplane to maximize margin between datasets. Input file is two n-dimensional vector sets. The creation of two parallel hyperplanes for computing the margin is complete. The closest surrounding data points in both groups give the best separation. Margin increases generalization mistake. Using support vectors and margins, hyperplanes are found. Here, info points are in:

$$(Z_1, X_1), (Z_2, X_2), (Z_3, X_3), \ldots, (Z_i, X_i)\}$$

Here, $Z_i = \frac{1}{-1}$ is a constant representing Z_i’s class. The *i* indicates sample size. Z_i denotes m-dimensional real vector. The dividing hyperplane helps visualize training data according to the formula below:

$$u.z + c = 0 \quad (4)$$

u is an m-dimensional vector and c a scalar

The vector u is perpendicular to the dividing hyperplane and the margin is maximal. If the c is missing, the hyperplane goes through the origin. Parallel hyperplanes maximize margin necessarily. This equation describes parallel hyperplanes.

$$u.z + c = 1 \quad (5)$$

$$u.z + c = -1$$

Parallel hyperplanes are chosen because linearly separating training data leaves no points between them. Geometrically, $\frac{2}{|u|}$ is the hyperplane distance. This requires $|u|$. Hence, it equates to:

$$u.z_j - c \geq 1 \; or \; u.z_j - c < -1$$

The following notation is used to formally define a hyperplane:

$$L(z) = \alpha_o + \alpha^c z$$

In which α weight is vector and $\alpha_o$represents the bias.

The ideal hyperplane is denoted by scaling α and α_o values. From diverse hyperplane representations, one is chosen:

$$|\alpha_o + \alpha^c z| = 1$$

Where z denotes close-to-hyperplane training instances, termed support vectors. It's a canonical hyperplane.

3.4.2. Decision Trees (DTs)

DTs classify cases by ranking feature values. In a classification example, each node in a DT illustrates a feature and each branch indicates the node's price. First, instances are classified at the root node based on their attribute values. Real-world datasets are handled by C5.0, Id3, or CART. The C5.0 could be a DT built with ID3's multi-valued features and Information Gain. C5.0 computes the Information Gain Ratio for each feature to solve this challenge. The training dataset's root node is the attribute with the highest Information Gain Ratio. The attribute with the highest Gain Ratio is split so that less information is needed to predict a specific instance in the ensuing feature split. Gain Ratio evaluation for characteristic A:

$$GainRatio(A) = \frac{Gain(A)}{Split\ Info(A)}$$

$$Gain(A) = info(D) - Info_A(D)$$

Where D is the training dataset

$$Info(D) = -\sum_{i=1}^{n} p(ci) \log_2 P(C_i)$$

$P(C_i) = |C_{i,D}|/|D|$ where $|C_{i,D}|$ is the number of tuples in the training dataset class C_i, |D| is the number of training dataset tuples, and n is the number of class values.

$$Info_A(D) = \sum_{i=1}^{n} (\frac{|a_{i,D}|}{|D|} \times (-\sum_{j=1}^{m} \frac{|C_{JD}|}{|a_{i,D}|} \log_2 \frac{|C_{J,D}|}{a_{i,D}})$$

Where :

$|a_{i,D}|$ => the number of the tuples of the value a_i of the attribute A in the training dataset.

$|D|$ => tuples of the training dataset.

n => number of the values of attribute A.

$|C_{J,D}|$ => number of the tuples of class $|C_J|$ associated with the value ai of the attribute A.

m => the number of the classes of class C.

3.4.3. Bernoulli Naïve Bayes (BNB)

For data whose distribution is finished using multivariate Bernoulli distributions, the NB training and classification method are distributed in the BNB. This implies that there are multiple characteristics. All attributes are binary variables. The category needs binary-valued feature vector samples. When BNB receives quiet data, it binarizes. The BNB choice rule asserts that

$$P(x_i|y) = P(i|y)x_i + \big(1 - P(i|y)\big)\,(1 - x_i) \qquad (6)$$

varies from the multinomial Naïve Bayes' rule, which penalizes the non-occurrence of an attribute i that's an indicator for class y. The multinomial variation simply ignores a non-occurring attribute. This classifier is trained using text word vectors. Bernoulli Naïve Bayes performs better on some datasets, notably shorter documents. If time permits, models should be computed.

3.4.4. Gaussian Naïve Bayes (GNB)

Making use of Gauss's distribution for the interpretation of the likelihood of the characteristics depending on the classes is a common technique of dealing with continue attributes in NB classification. As a result, a Gaussian Probability Density Function (PDF) is used to describe each property as $X_i \sim N(\mu, \sigma^2)$

The Gaussian PDF is shaped like a bell and is characterized by the equation below:

$$N\,(\mu, \sigma^2)(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e - \frac{(x-\mu)^2}{2\sigma^2} \qquad (7)$$

$$\mu = mean$$

$$\sigma^2 = variance$$

NB requires $O(n, k)$ parameters, where n is the number of characteristics and k is the number of classes. $P(X_i \setminus C) \backsim N(\mu, \sigma^2)$ must be defined for every continuous attribute. The below formula gives normal distribution parameters.

$$\mu_{X_i|C=c} = \frac{1}{N_c}\sum_{i=1}^{N_c} x_i \qquad (8)$$

$$\sigma^2{}_{X_i|C=c} = \frac{1}{N_c}\sum_{i=1}^{N_c} {x_i}^2 - \mu^2 \quad (9)$$

$N_c$is the number of training cases where $C = c$ and N is the total number. With relative frequencies, $P(C = c)$may be calculated for all classes

$$P(C = c) = \frac{N_c}{N} \quad (10)$$

3.4.5. Multilayer Perceptron (MLP)

Multilayer perceptrons are the simplest feed-forward network (MLP). An MLP uses the output of one perceptron as the input of the next. Nonlinear functions assess neuron activation (state). **Figure 4.3.** shows the general architecture of the multilayer perceptron:

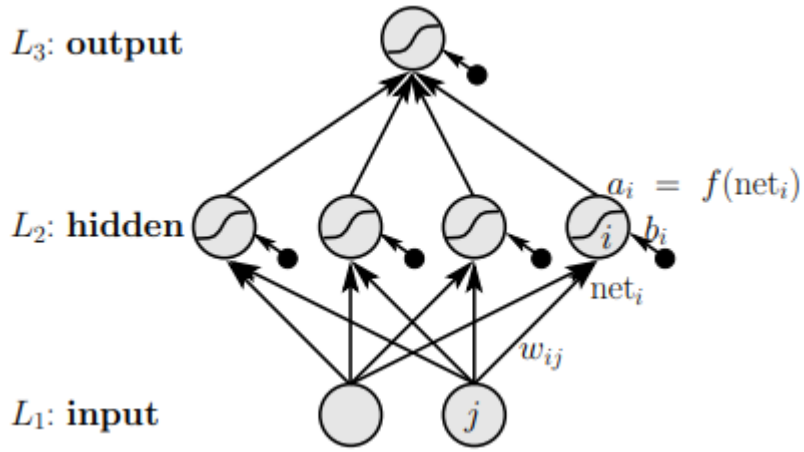

Figure 3.3. Multilayer Perceptron

In this figure:

a_i= activity of the ith unit.

a_o=1: activity of 1 of the bias unit

w_{ij}= weight from unit j to unit i

w_{io}= b_i : bias weight of unit i

W: number of weights

N: number of units

I: number of inputs units $(1 \leq i \leq I)$ situated in the first layer known as the input layer

O: number of output units $(N - O + 1 \leq i \leq N)$ positioned in the last layer known as the output layer

M: number of hidden units $(I < i \leq N - O)$ positioned in the hidden layers.

L: number of layers, where L_v is the index set of the vth layer; $L_1 = \{1, \ldots., I\}$ and $L_1 = \{N - O + 1, \ldots., N\}$

net_i : network input to the ith unit (I <i) measured as:

$$net_i = \sum_{j=0}^{N} w_{ij}\, a_j \tag{11}$$

f: activation function with

$$a_i = f(net_i) \tag{12}$$

Several activation functions f_i for different units can be defined. The activation function is also known as the transfer function.

A feed-forward MLP includes just links from units in lower layers to units in higher layers:

$$i \in L_v \; and \; j \in L_{v'} \; and \; v' \leq v \Rightarrow \; w_{ij} = 0 \tag{13}$$

Merely connections or weights between consecutive layers occur for a traditional multi-layer perceptron. Other weights are fixed to zero. After this, the network input is for node i in hidden or output layer ν with ν > 1

$$\forall_{i \epsilon L_v} : net_i = \sum_{j : j \epsilon L_{V=1}}^{N} w_{ij}\; a_j \tag{14}$$

Non-adjacent connections between units in layers are known as shortcut connections.

Activation Functions

Sigmoid functions are the most frequently used activation functions. The logistic function is:

$$f(a) = \frac{1}{1 + \exp(-a)} \tag{15}$$

and the tanh activation function is:

$$f(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} \tag{16}$$

3.4.6. Random Forest (RF)

Ensemble learning is RF. A low DT with few properties can be generated computationally cheaply. When several small, weak DTs are developed in tandem, the trees are combined to form one, strong learner. RFs are the most accurate training learning algorithms thus far.

Formally, an RF might be a predictor where a group of randomized base regression trees is generated as $\{r_n(x,\ominus_m, D_n), m \geq 1\}, where\ \ominus_1,\ominus_2$ .... are independent and identically distributed outputs of a randomized variable $\ominus$. Integration of RTs for aggregated regression estimate

$$\bar{r}_n(X, D_n) = \mathbb{E}_{\ominus}[r_n(X,\ominus, D_n)] \quad (17)$$

In which $\mathbb{E}_{\ominus}$ denotes the expected value of the random parameter, conditional on X and therefore the dataset D_n. To simplify the notation, the estimates' dependency would be eliminated inside the population and stated as $\bar{r}_n(X)$ instead of $\bar{r}_n(X, D_n)$. In practice, Monte Carlo is used to calculate the aforementioned expectation when the MRTs are created, as well as the average of the individual outcomes. The randomizing variable $\ominus$ is distributed to determine the performance of subsequent cuts while creating the various trees within which selection of the coordinates to separate and divide position are composed. Because of the independence of X and the training sample D_n, the variable $\ominus$ is deduced.

3.4.7. PCA Algorithm

PCA is a statistical technique that collects connected elements and transforms them into linearly unrelated subsets that give uncorrelated variables. The PCA, also called orthogonal linear transformation, helps generate a projection of the initial dataset to a different projection system with the goal that the most important variance includes a projection of the primary coordinate, and the second biggest variance is a vertical projection of the second coordinate. PCA positions a linear transformation, $z = W_k^T x$ where $x \in R^d$, and $r < d$, to enhance the data variance in the projected space. The $X = \{x_1, x_2, \ldots\ldots, x_i\}$, $x_i \in R^d$, $z \in R^r$ and $r < d$ is employed to specify the information matrix, and therefore the transformation is often described with the assistance of a group of p-dimensional vectors of weights $W = \{w_1, w_2, \ldots\ldots, w_p\}$, $w_p \in R^k$, which matches every x_i vector of X to a

$$t_{k(i)} = W_{|(i)} T_{x_i} \quad (18)$$

An initial weight $W1$ must adhere to the following requirement in order to increase variance:

$$W_i = \arg max_{|w|} = \{\sum_i (x_i \cdot W)^2\} \quad (19)$$

The following condition is an extension of the previous one:

$$W_i = \arg max_{\|w\|=1} \{\|X.W\|^2\} = \arg max_{\|w\|=1} \{W^T X^T XW\} \quad (20)$$

It is easier to decipher an $X^T X$ symmetric grid when W is the biggest possible eigenvalue, because W is the associated eigenvector. It will be possible to determine the first principal component after getting W_1 through the projection of initial data matrix X onto the W_1. The further segments are obtained along these lines after the subtraction of the newly obtained components.

3.4.8. Ensemble 1 Classifier

C4.5 is an ensemble classification approach for the Gaussian Naïve Bayes, Bernoulli Naïve Bayes, Random Forest, and C4.5 ensembles of classification methods. The voting classifier uses the four methods as input for further prediction.

3.4.9. Ensemble 2 Classifier

Gaussian Naïve Bayes, Bernoulli Naïve Bayes, Random Forest, and SVM are the other ensemble classification methods. The voting classifier uses the information from the four algorithms to predict the outcome of a vote in the future.

3.4.10. Ensemble 3 Classifier

RF, BNB, and MLP ensembles are classified using this method. Voting classifiers use these four methods as input to make predictions.

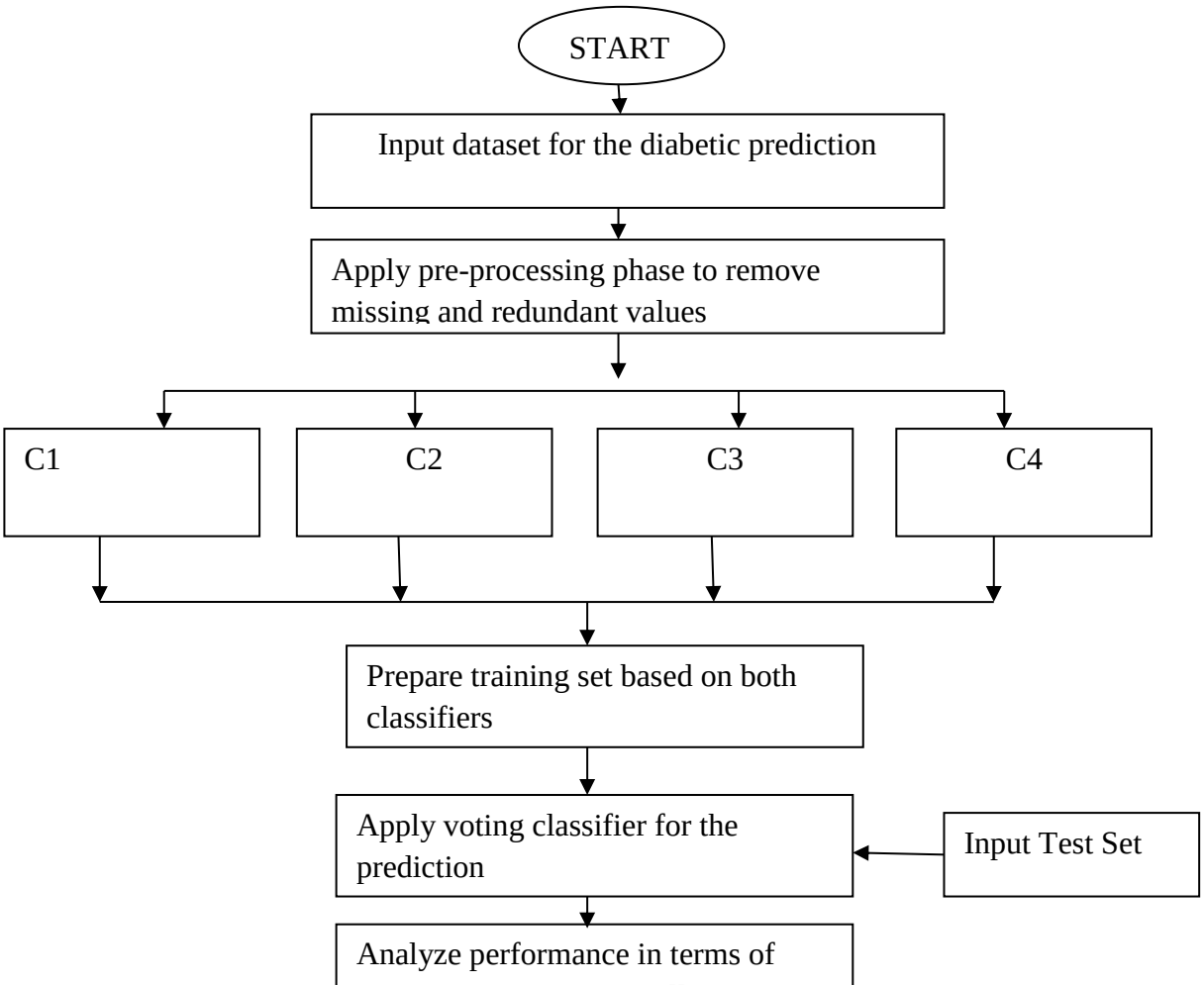

Figure 3.4. Flow Chart for Ensembles Classifiers

3.4.11. PCA + Ensemble 1 Classifier

The initial ensemble classification method of the GNB, BNB, RF, and C4.5 uses the PCA algorithm for feature reduction. The voting classifier uses all five of these algorithms as input to make a prediction.

3.4.12. PCA + Ensemble 2 Classifier

The PCA algorithm is used for the feature reduction with the first ensemble classification method of the Gaussian Naïve Bayes, Bernoulli Naïve Bayes, Random Forest, and MLP. The five algorithms are given as input to the voting classifier for the further prediction.

3.4.13. PCA + Ensemble 2 Classifier

SVM, Random Forest, and Bernoulli Nave Bayes are all ensemble classification methods that use the PCA technique to reduce the number of features. The voting classifier uses the information from the five algorithms to make a prediction.

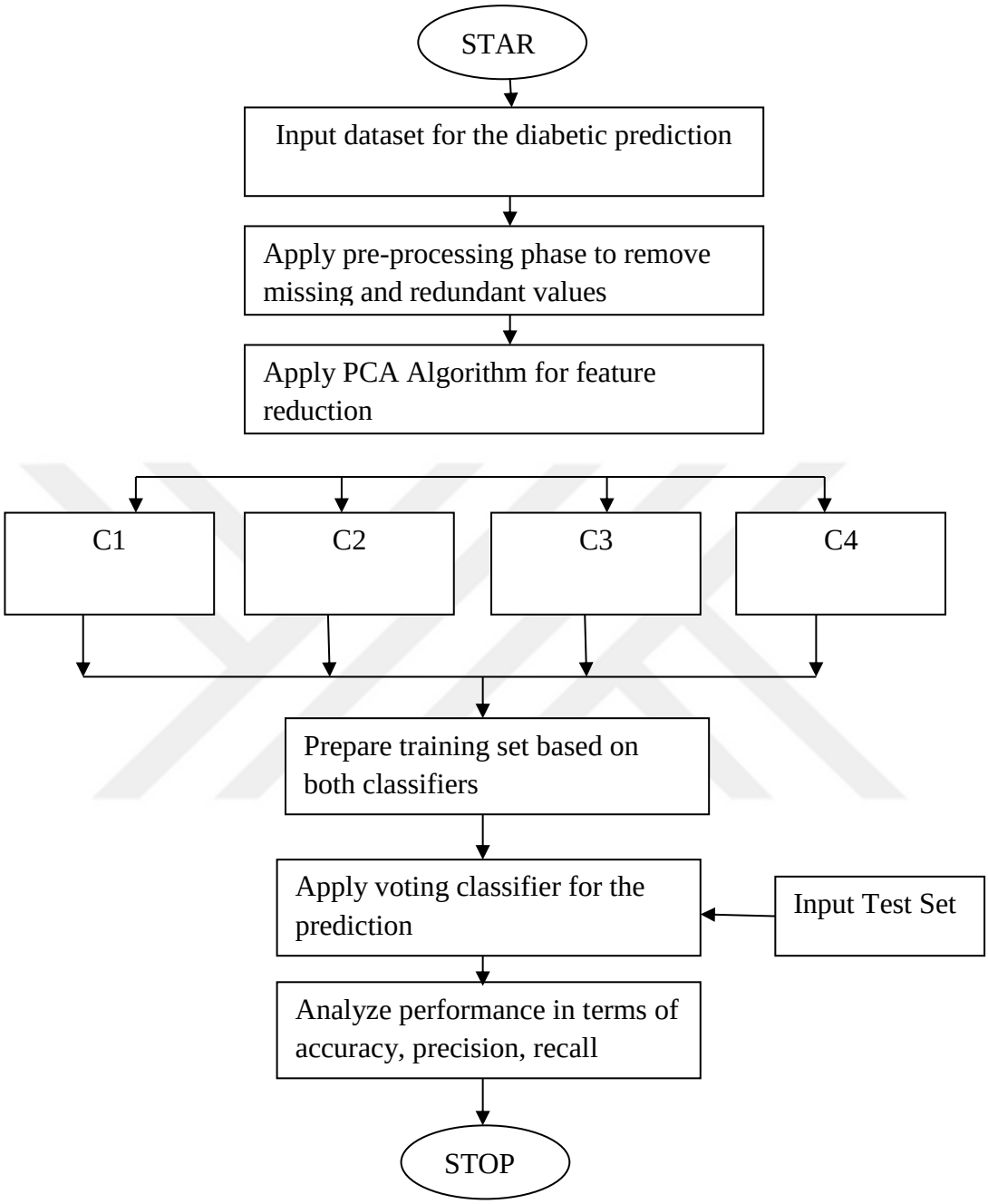

Figure 3. 4. PCA+ Ensembles Classifier Flow Chart

3.5. Dataset Description

McCabe and Halstead's ASCII text file extractors provide the data for this report. During the 1970s, these features were developed to describe software qualities in code in an objective manner. The nature of the organization is being debated. Notes on McCabe and Halstead are next. Modules, the lowest functional unit, are the basis for the McCabe and Halstead measurements. This term "module" would be referred to in C or Smalltalk by the terms function and method.

3.6. Implementation

Python Programming Language is used to implement all classifiers and their combinations.

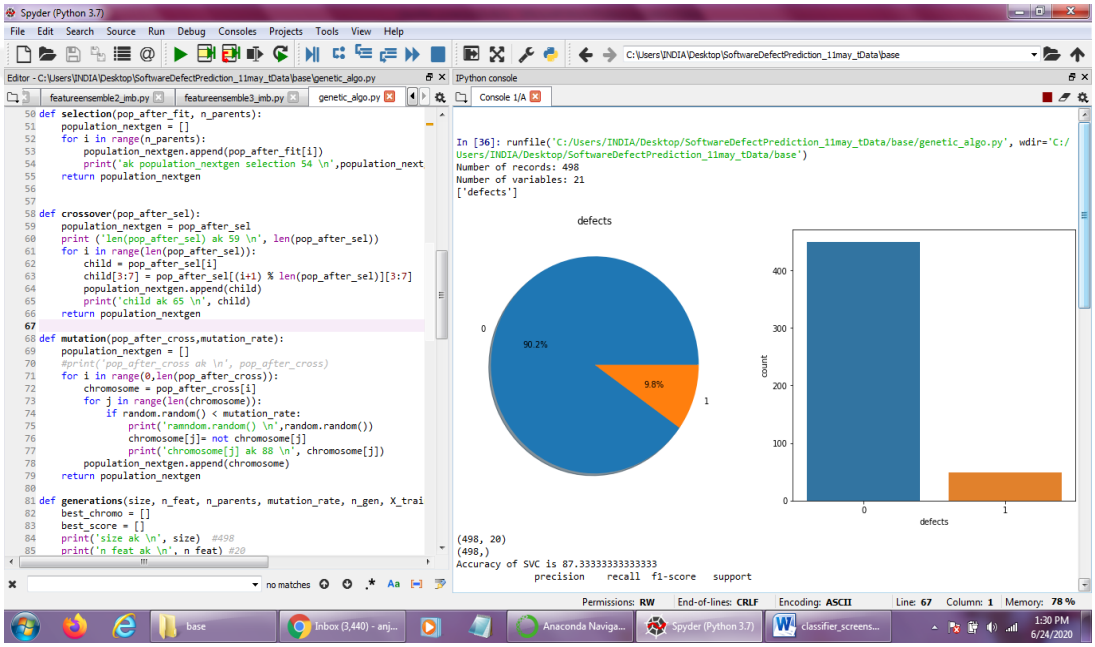

Figure 3. 5. GA+PSO and Bagging Classifier

Figure 3. 6. shows that the GA+PSO and bagging with SVM are used for feature extraction and classification, respectively. There are several optimization and parallelism difficulties that the GA (genetic algorithm) can handle. An approach called PCA (Principal Component Analysis) is used to reduce the number of attributes. The Bagging algorithm improves the outcomes by predicting faults in advance.

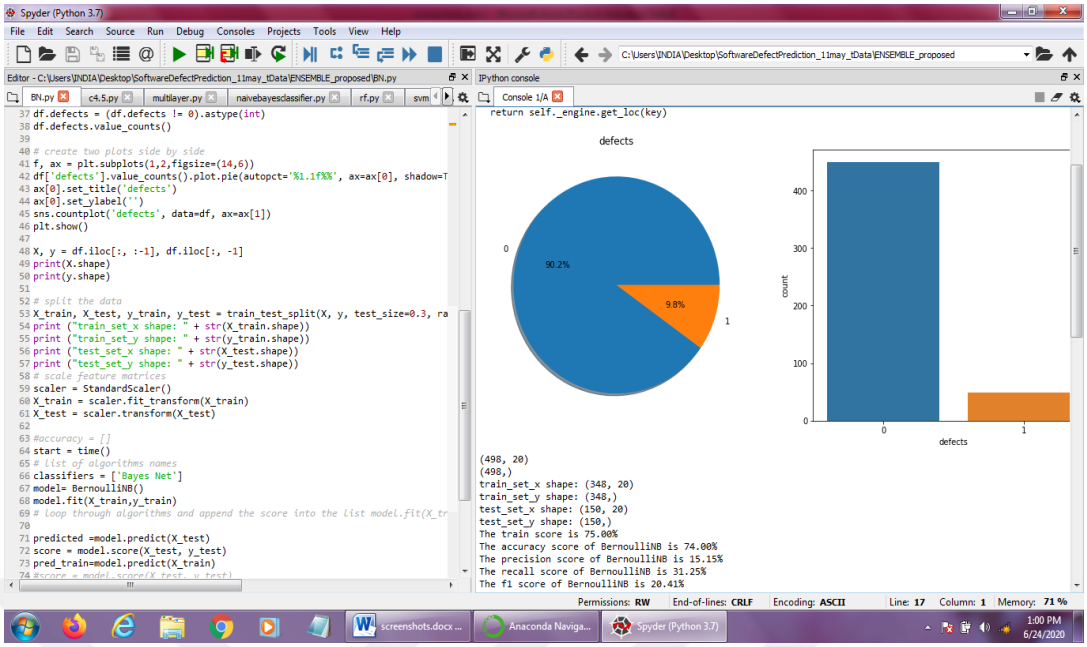

Figure 3. 6. Bernoulli Naïve Bayes Implementation

Figure 3. 7. depicts the implementation of BNB (Bernoulli Naïve Bayes) algorithm to predict the software defect. The BNB has similarity with multinomial NB. However, its predictors are considered as Boolean variables. The parameters considered for predicting the variable take values: *yes* or *no* only.

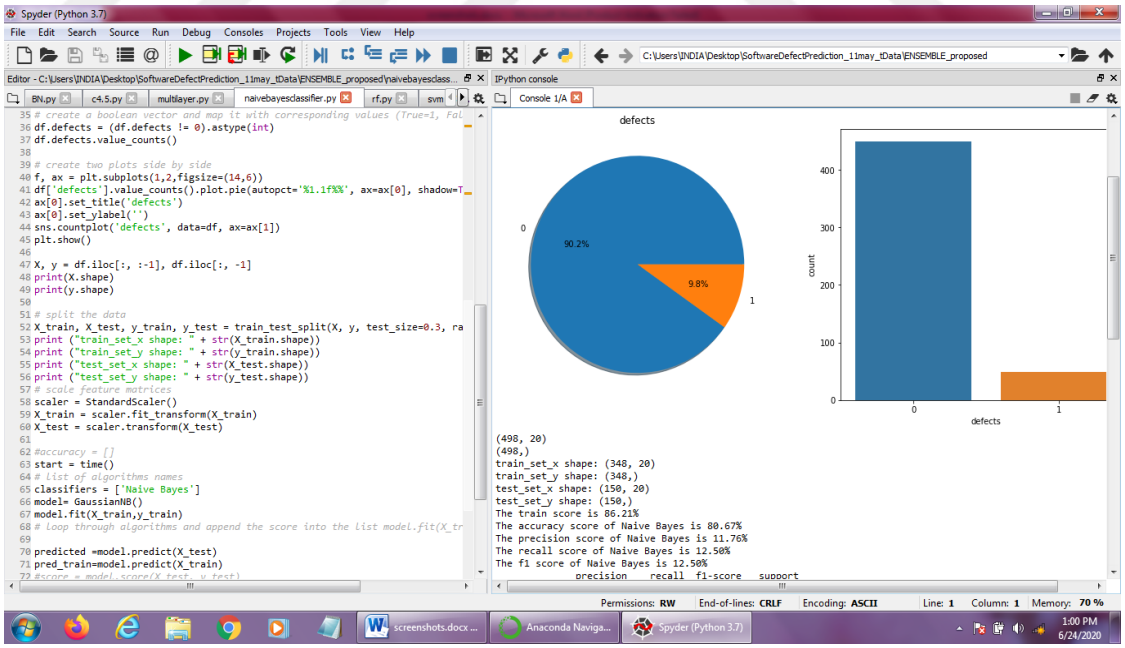

Figure 3. 7. Gaussian Naïve Bayes Implementation

Figure 3. 8. illustrates the deployment of GNB (Gaussian Naïve Bayes) classifier to predict the faults in software. GNB (Gaussian Naive Bayes) employs a probability

density function for making predictions regarding metrics and proves effective for estimating the new input value's probability.

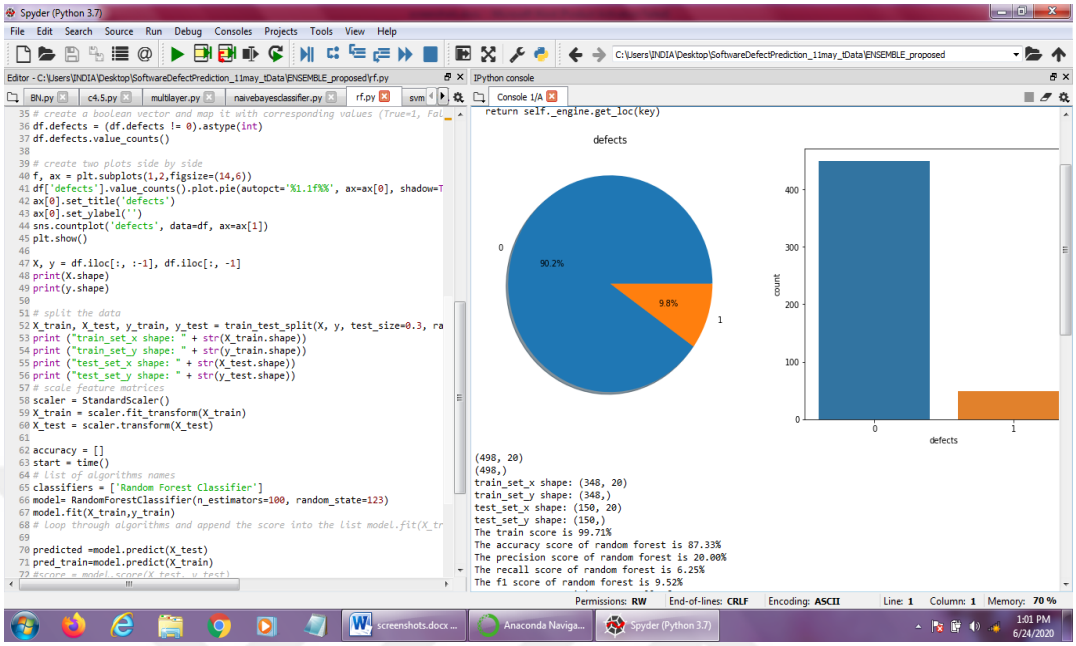

Figure 3. 8. Random Forest Implementation

Figure 3. 9. indicates that the software faults are forecasted using the RF (Random Forest) algorithm. RF is a robust algorithm which renders superior accuracy and maintains the accuracy even in case of presence of misplaced data and predicted the software defects robustly.

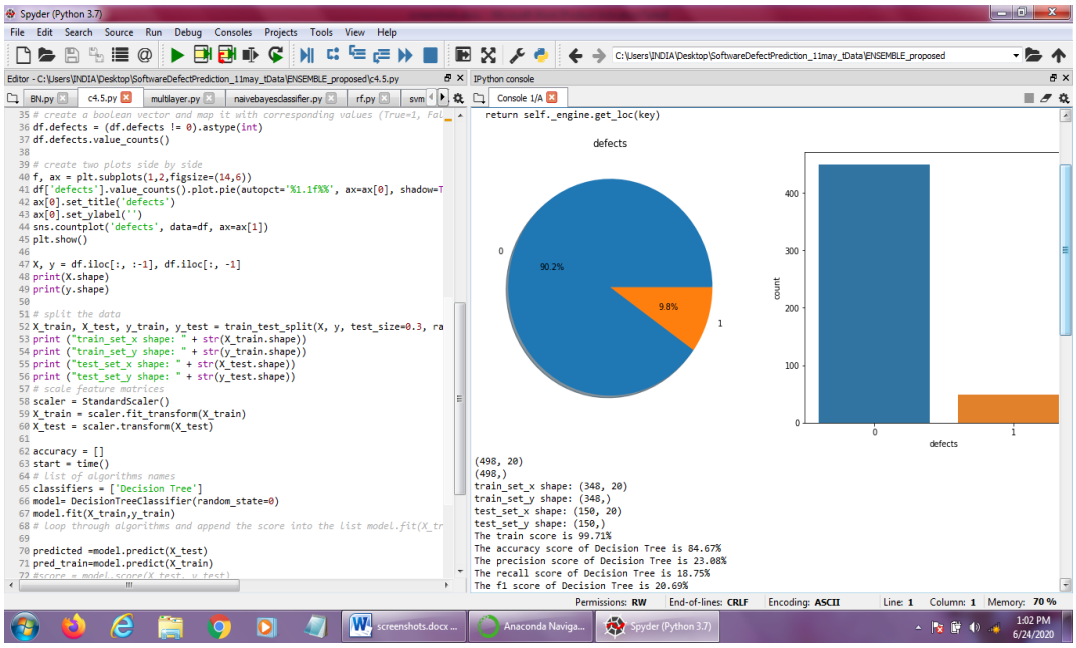

Figure 3. 9. Decision Tree Implementation

Figure 3. 10. exhibits that the DT (Decision Tree) algorithm is adopted to predict the defects occurred in software. DT algorithm emphasizes on producing a training system so that the class or value of the target variable can be predicted. For this, this algorithm learns simple decision rules taken from prior data.

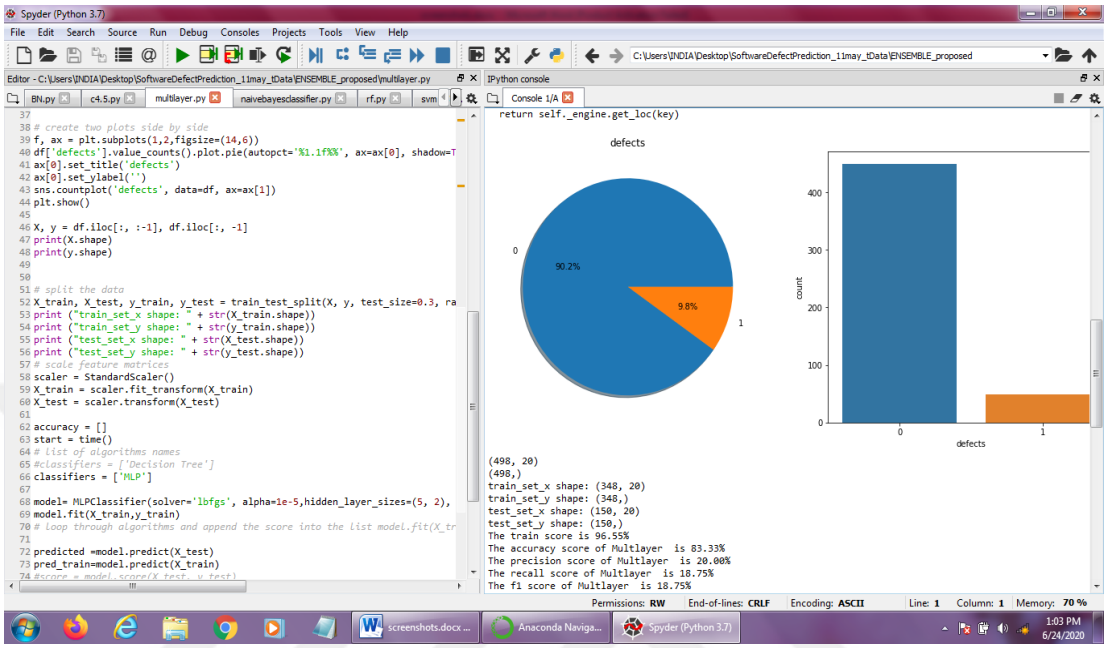

Figure 3. 10. Multilayer Perceptron Implementation

Figure 3. 11. demonstrates that the software defects are forecasted based on MLP (multilayer perceptron) algorithm. MLP algorithm is exploited to engender various outputs from a set of inputs.

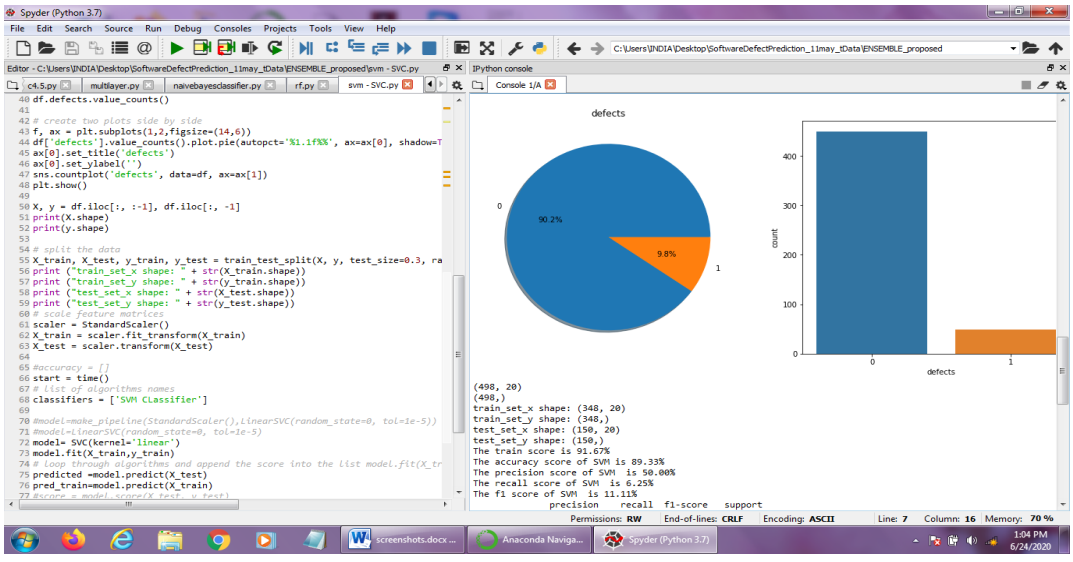

Figure 3. 11. SVM classifier Implementation

Figure 3. 12. exhibits that the SVM algorithm is exploited to predict the software defect. The SVM (Support Vector Machine) is a reliable algorithm for generating the finest decision boundary to segregate n-dimensional space into classes so that the defects can be predicted in accurate manner.

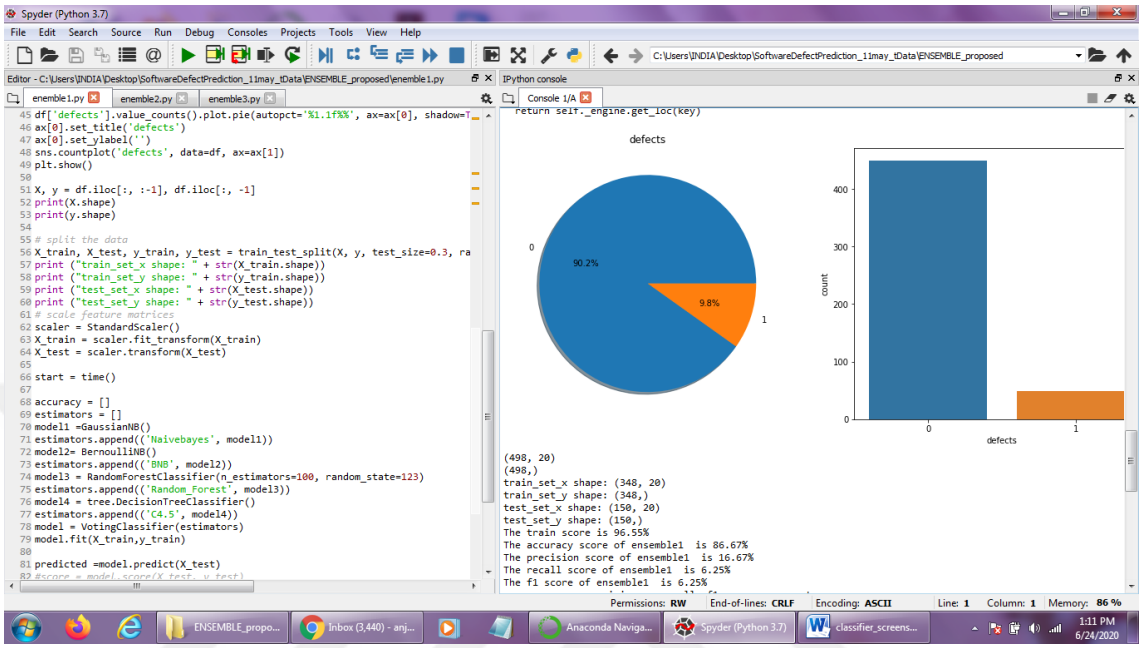

Figure 3. 12. Ensemble 1 classifier Implementation

Figure 3. 13. represents the implementation of the ensemble 1 classifier in which four classification algorithms namely GNB, BNB, RF and C4.5 are integrated. This ensemble algorithm can predict the defects occurred in software. GNB employs a probability density function for making predictions regarding metrics.

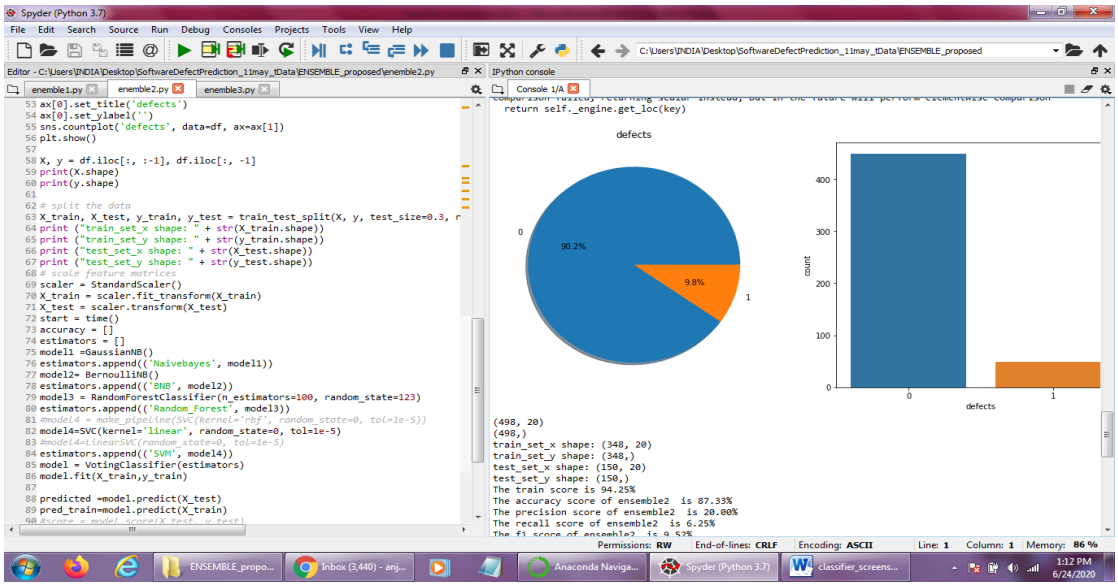

Figure 3. 13. Ensemble 2 classifier Implementation

Figure 3. 14. reveals the deployment of the ensemble 2 classifier in which four classification algorithms are put together such as GNB, BNB, RF, and SVM (Support Vector Machine). The software defects can be predicted efficiently using the ensemble 2 algorithm.

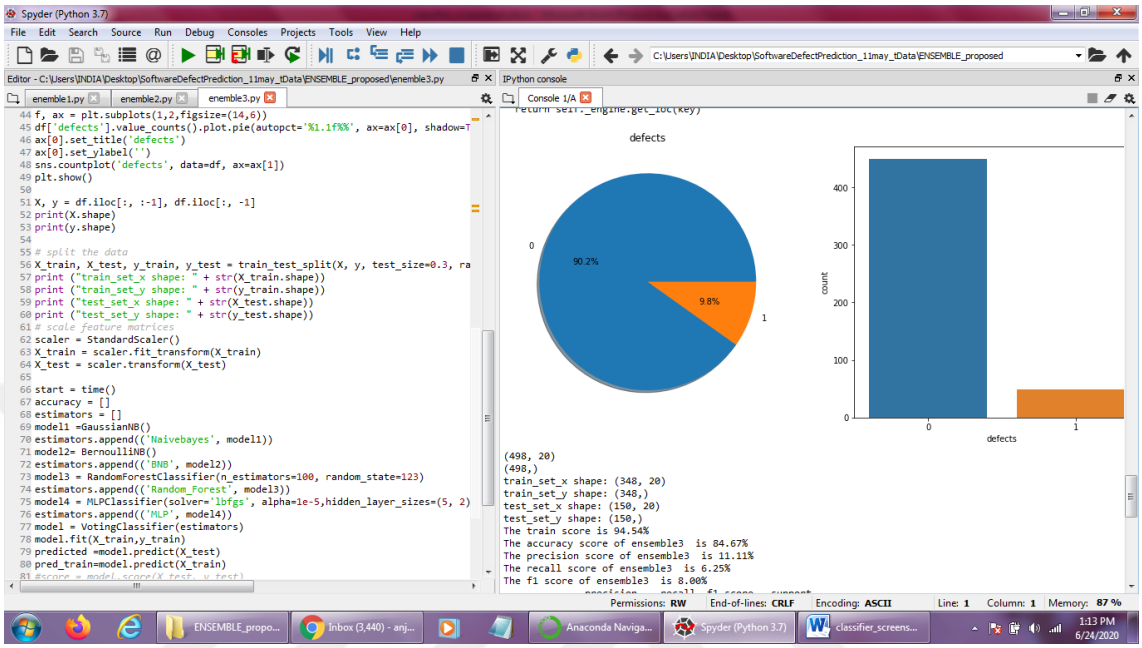

Figure 3. 14. Ensemble 3 classifier Implementation

Figure 3. 15. illustrates the GNB, BNB, RF, and MLP (Multilayer Perceptron) algorithms get integrated in ensemble3 classifier with the objective of predicting the software faults.

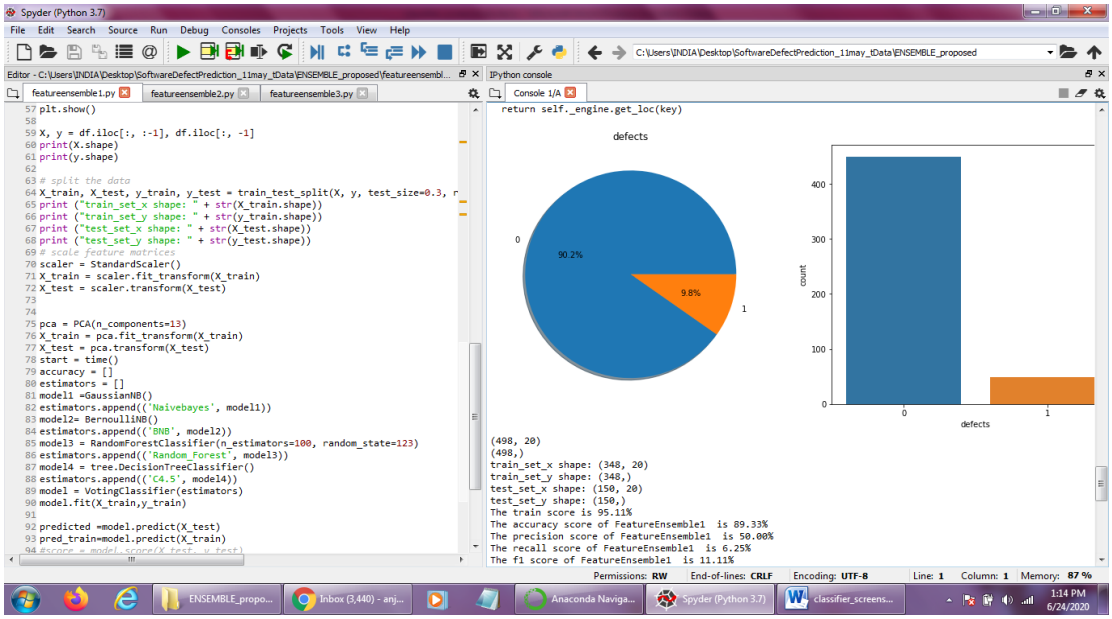

Figure 3. 15. PCA+Ensemble 1 Classifier Implementation

Figure 3. 16. defines that the integration of PCA with ensemble 1 classifier in which GNB, BNB, RF, and C4.5 are combined. Features are alleviated using PCA. The software defects are predicted successfully using PCA+Ensemble 1 Classifier.

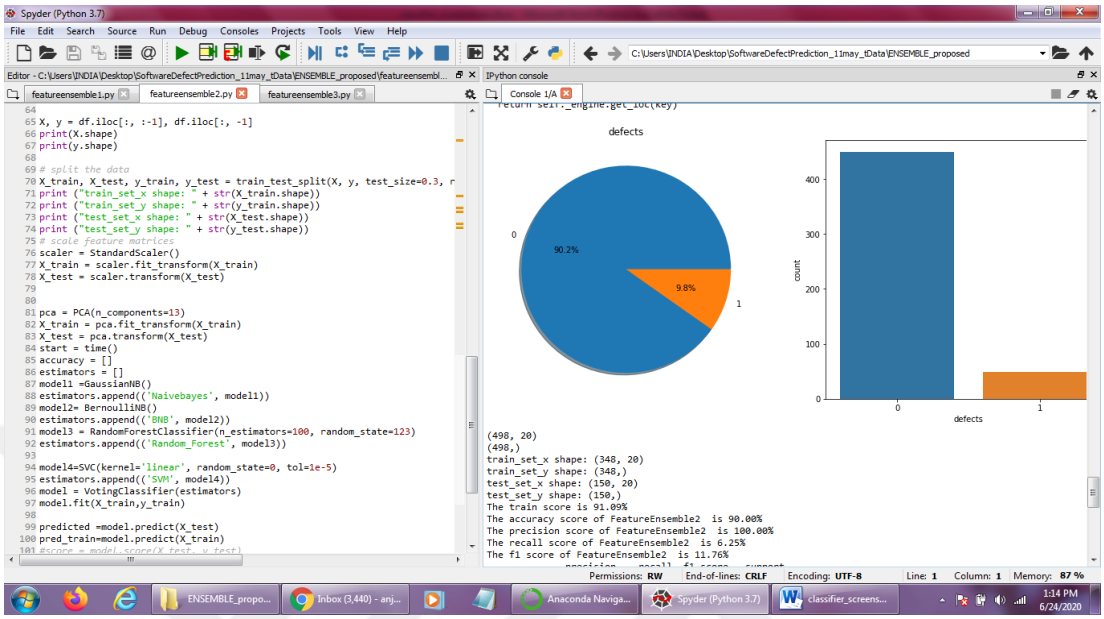

Figure 3. 16. PCA+Ensemble 2 Classifier Implementation

As shown in Figure 3. 17., the PCA in integration of ensemble 2 classifier which combines all four classifiers such as GNB, BNB, RF, and SVM (Support Vector Machine). PCA algorithm is exploited for reducing the features. This integration is effectual to predict the defects of software.

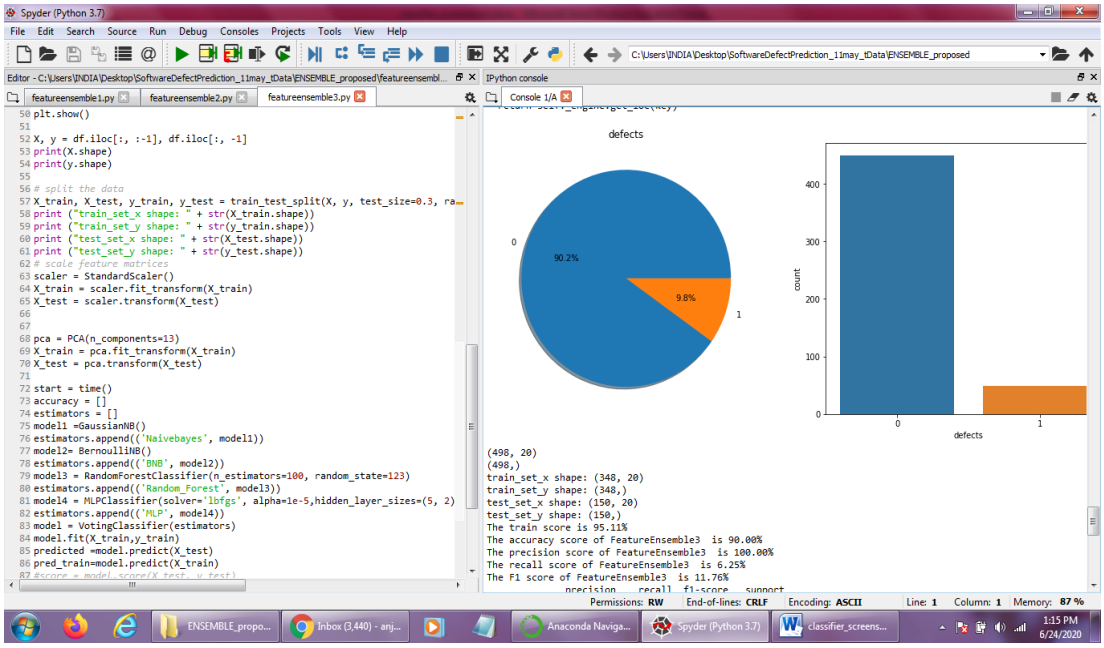

Figure 3. 17. PCA+Ensemble 3 Classifier Implementation

As **Figure 3. 18.** shows, t PCA + ensemble 3 classifier (a combination of 4 classifiers that are GNB, BNB, RF, and MLP (Multilayer Perceptron) are employed jointly to predict the defects of software. The features are mitigated using Principal Component Analysis.

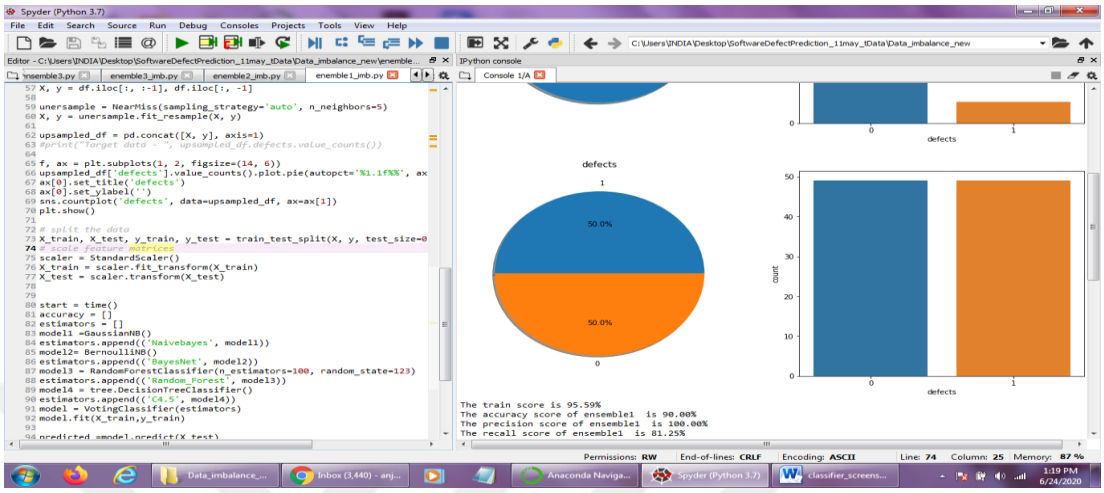

Figure 3. 18. Ensemble1 Classifier with Class Balancing (CB)

As depicted in **Figure 3. 19.**, the CB ensemble1 classifier is the combination of four classifiers that are GNB, BNB, RF, and C4.5. This classification is implemented to predict the defects in software.

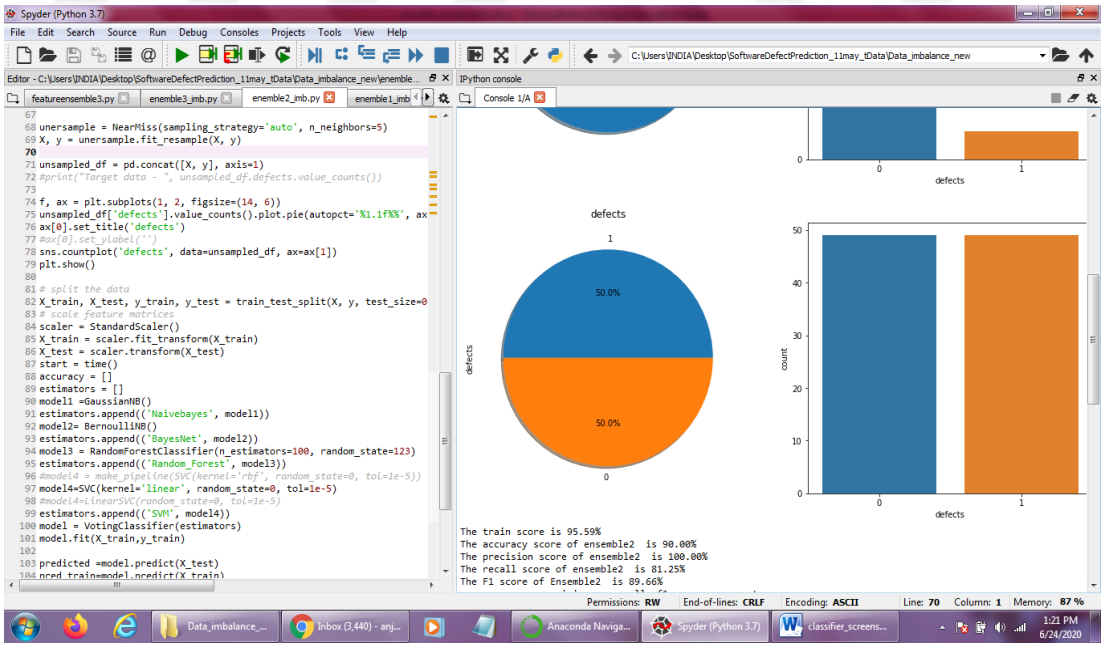

Figure 3. 19. Class Balancing with Ensemble 2 Classifier

Figure 3. 20. indicates the implementation of the class balancing ensemble2 classifier that merges four classifiers such that GNB, BNB, RF, and SVM (Support Vector Machine). This classification algorithm is utilized to predict the software defects.

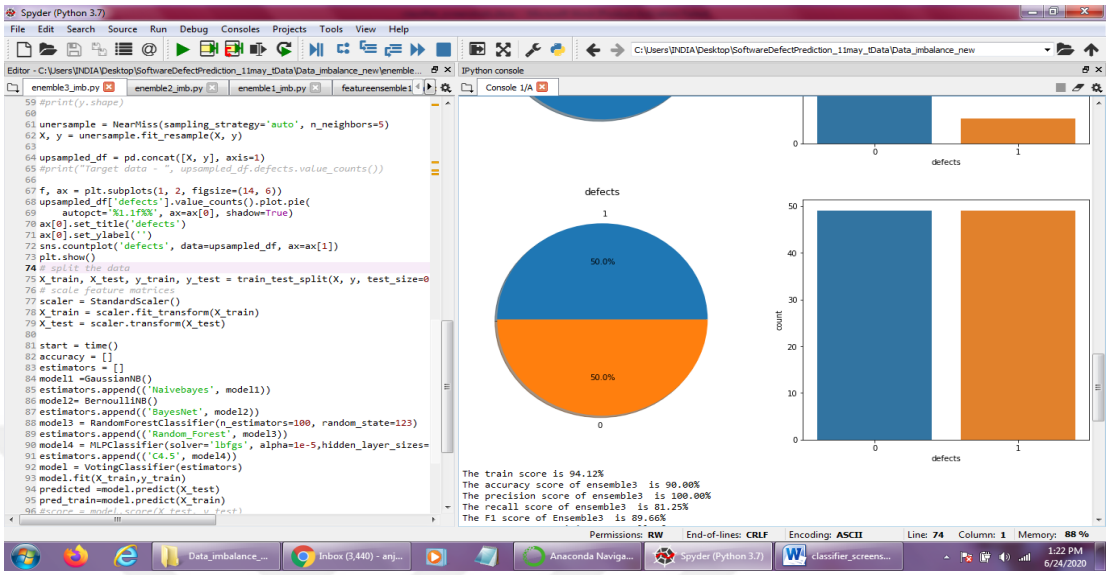

Figure 3. 20. Class Balancing with Ensemble 3 Classifier

Figure 3. 21. depicts that the class balancing ensemble 3 classifier is also combining four classifiers such as GNB, BNB, RF, and MLP (Multilayer Perceptron) which assist in forecasting the software defects.

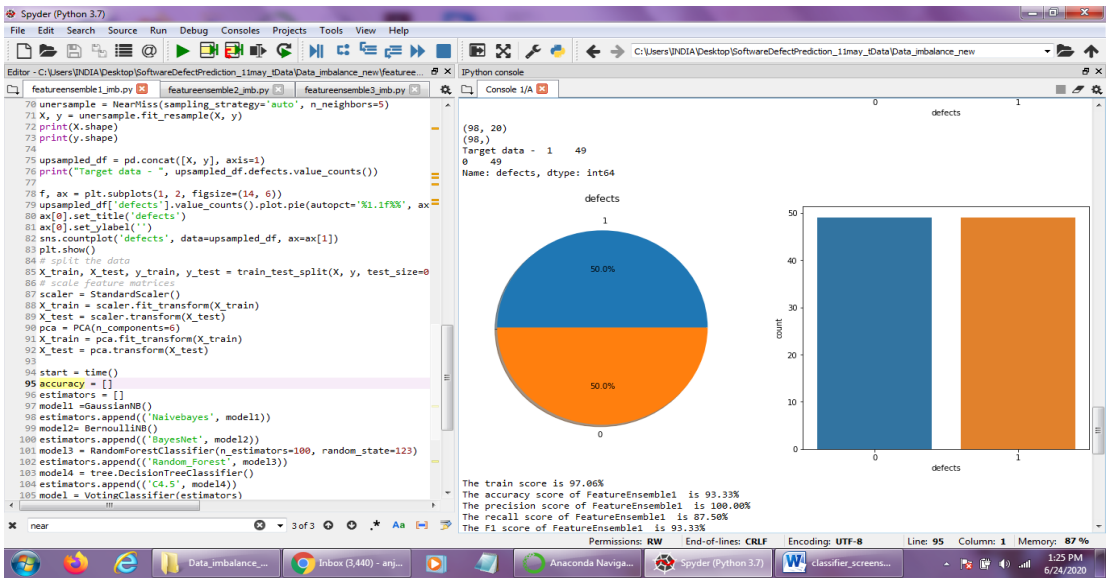

Figure 3. 21. PCA with CB and Ensemble1 Classifier

As shown in **Figure 3. 22.**, the CB + PCA ensemble1 classifier in which four classifiers GNB, BNB, RF, and C4.5 are assimilated. Here, features are diminished using Principal Component Analysis algorithm. This classifier can predict the software defects.

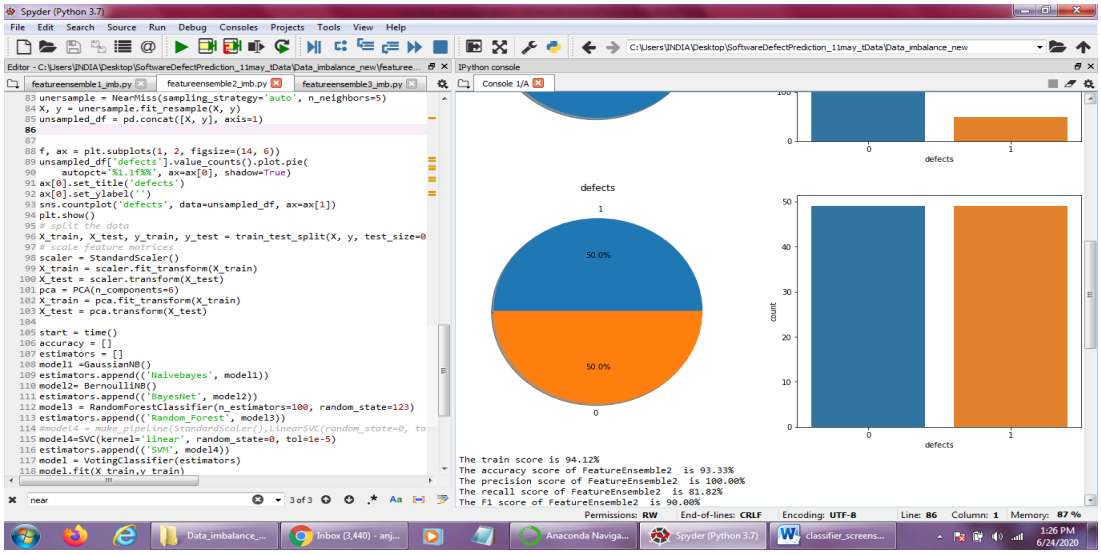

Figure 3. 22. PCA with CB and Ensemble2 Classifier

We can see that, **Figure 3. 23.** demonstrates that the CB + PCA ensemble2 classifier is mix of 4 other classifiers namely GNB, BNB, RF, and SVM (Support Vector Machine). This classifier is implemented to predict the software defects.

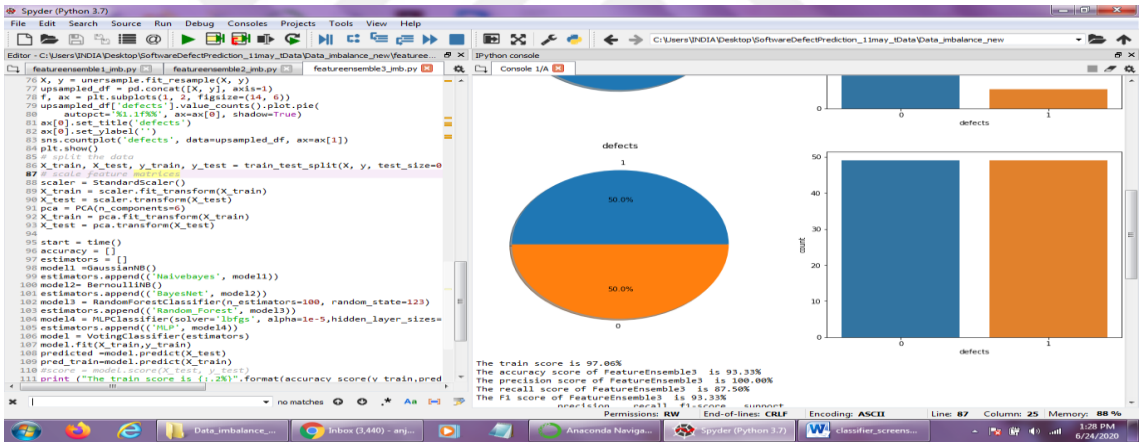

Figure 3. 23. PCA with CB and Ensemble3 Classifier

The above **Figure 3. 24.** exhibits the CB in conjunction of PCA ensemble3 classifier that mixes the following 4 classifiers which are GNB, BNB, RF, NB (Naive Bayes) and MLP (Multilayer Perceptron) for predicting the software defects.

4. RESULTS AND DISCUSSION

The accuracy, precision, and recall of all of the methods that have been implemented are displayed below in tables with their charts.

The results of all the classifiers are entered in **Table 4.1.**

Table 4. 1. Individual Classifier Results

Model	**Accuracy (in %)**	**Precision (in%)**	**Recall (in%)**
BernoulliNB	74.00	15.20	31.30
C4.5	84.70	23.10	18.80
GaussianNB	80.70	11.70	12.50
MLP classifier	83.33	20	18.750
SVM (kernel=-linear)	89.33	50	6.25
Random Forest	87.33	20	6.25

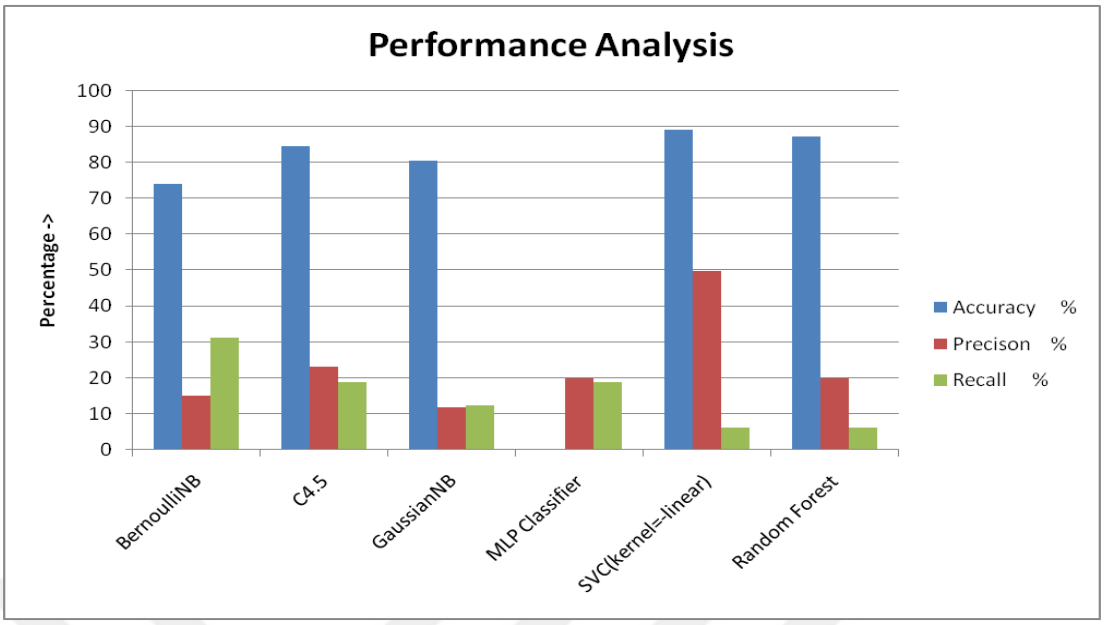

Figure 4. 1. Performance of Individual Classifiers

After showing the individual classifier results, we will be moving forward with Ensemble Classifers in terms of the same parameters (Accuracy, Precision, and Recall).

Table 4.2. shows the results obtained from all ensemble classifiers.

Table 4. 2. Ensemble Classifiers

Model	**Accuracy (%)**	**Precision (%)**	**Recall (%)**
Ensemble 1	86.67	16.6	6.25
Ensemble 2	87.33	20	6.25
Ensemble 3	84.67	11.11	6.5

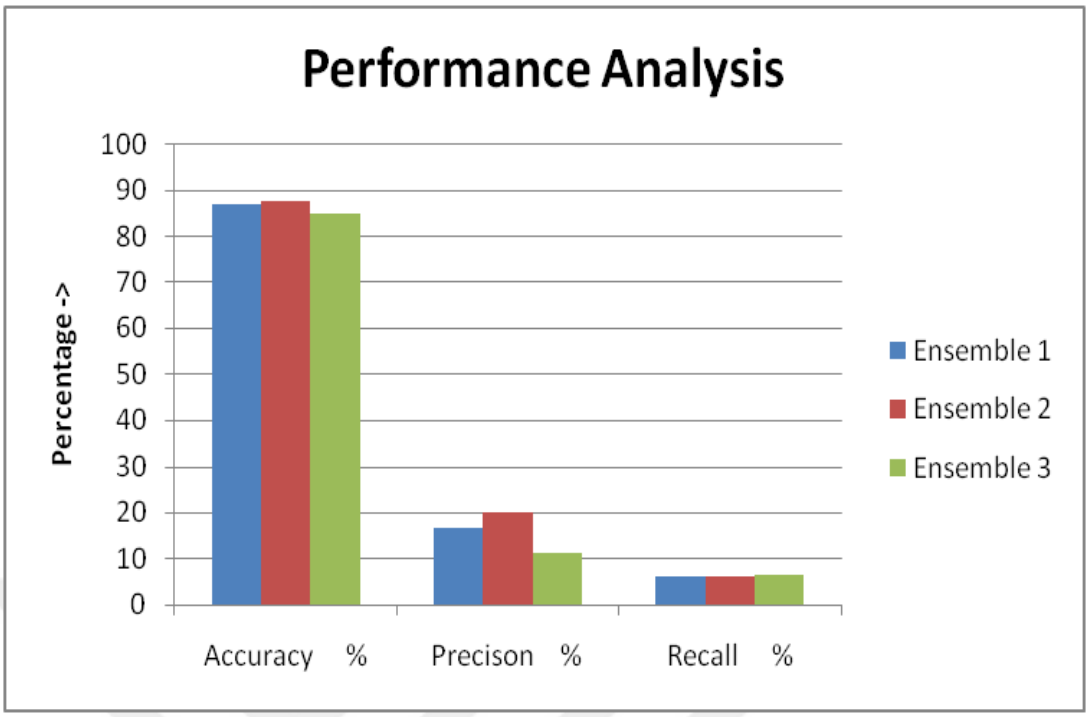

Figure 4. 2. Performance of Ensemble Classifiers

Let's now display the results obtained from PCA with Ensemble Classifier implementation.

The results of all PCA with ensemble classifiers are shown in **Table 4.3.**

Table 4. 3. PCA with Ensemble Classifiers

Model	**Accuracy (%)**	**Precision (%)**	**Recall (%)**
PCA + Ensemble 1	89.33	50	6.25
PCA + Ensemble 2	90	100	6.25
PCA + Ensemble 3	89.33	50	6.5

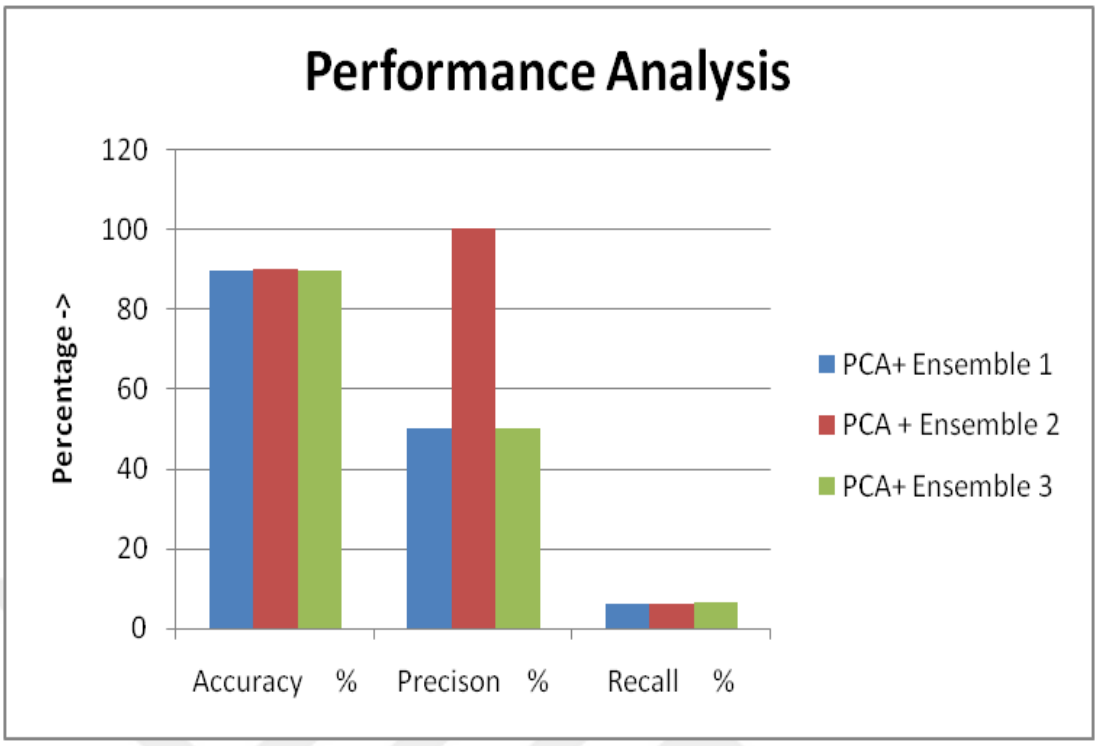

Figure 4. 3. Performance of PCA with Ensemble Classifiers

Class Balance in combination with ensembles was also implemented in this study and the performance has been enhanced. The results of all class balancing with ensemble classifiers are shown in **Table 4.4.**

Table 4. 4. Class Balancing with Ensemble Classifiers

Model	**Accuracy (%)**	**Precision (%)**	**Recall (%)**
Class Balance + Ensemble 1	90	100	81.25
Class Balance + Ensemble 2	90	100	81.25
Class Balance + Ensemble 3	90	100	81.25

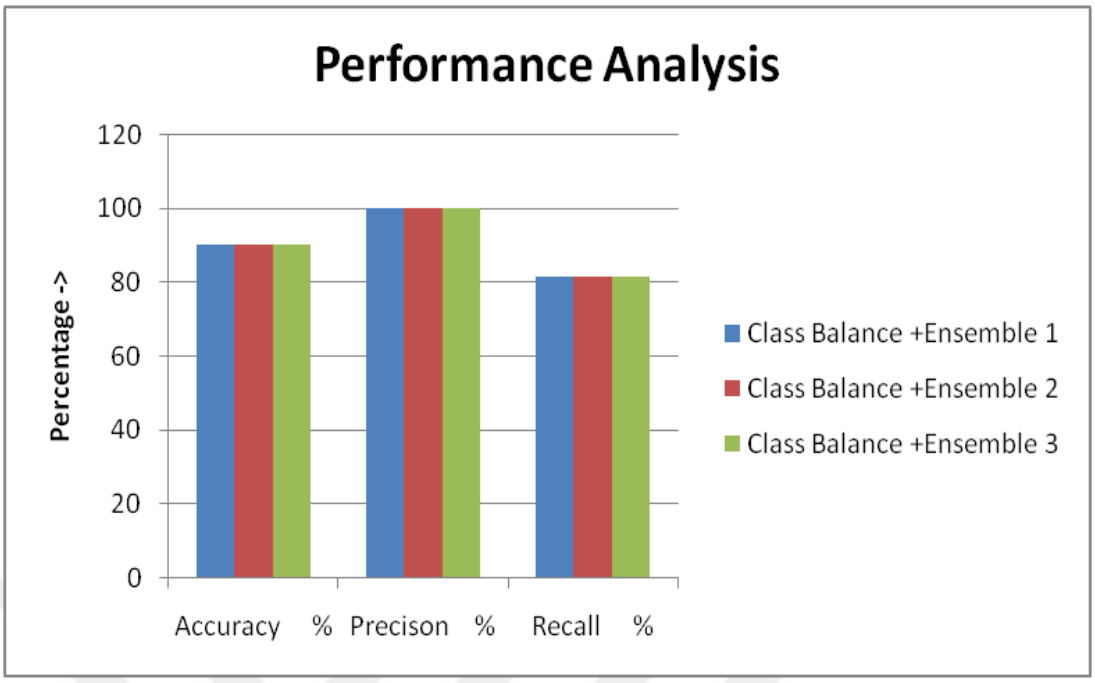

Figure 4. 4. PCA and Ensemble Classifiers can be used to achieve class balance

Lastly, we will explore the results got from the combination of PCA, Class Balance and Ensemble Classifiers. The results of all class balancing with PCA, and ensemble classifiers are shown in **Table 4.5.**

Table 4. 5. Class Balancing with PCA and Ensemble Classifiers

Model	**Accuracy (%)**	**Precision (%)**	**Recall (%)**
Class Balance + PCA + Ensemble 1	93.33	100	87.5
Class Balance + PCA + Ensemble 2	93.33	100	81.82
Class Balance + PCA + Ensemble 3	93.33	100	81.25

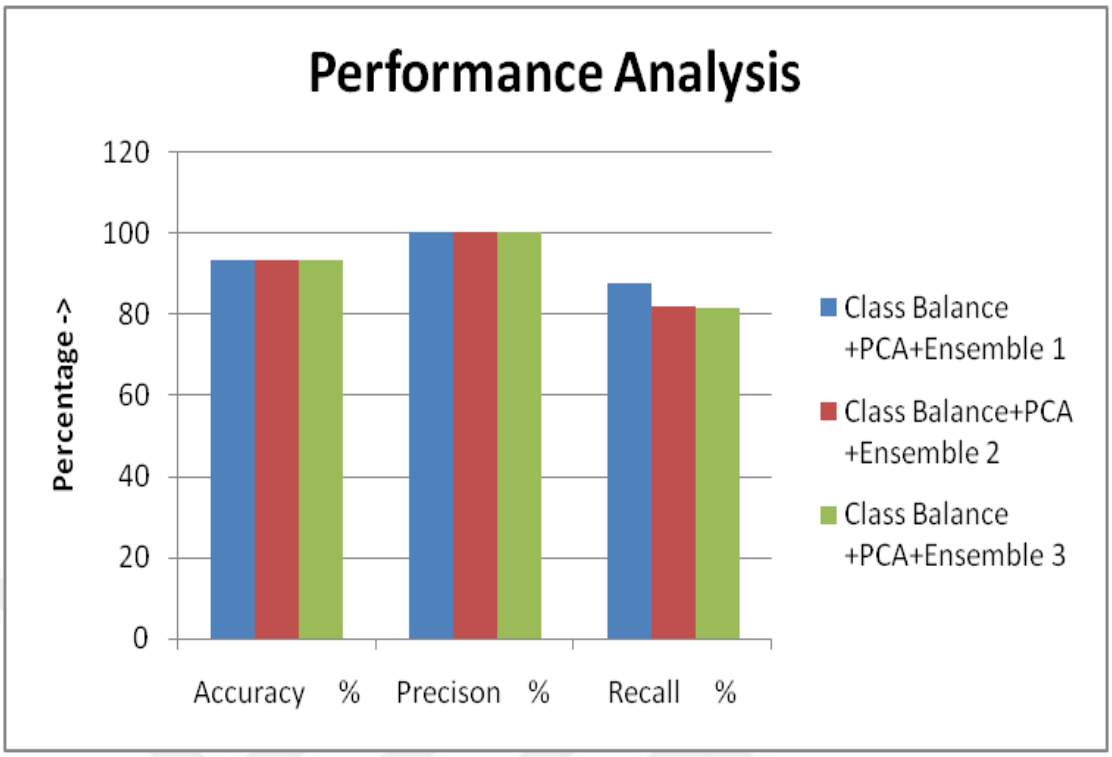

Figure 4.5. With PCA and Ensemble Classifiers, class balance is improved

5. CONCLUSION

Defects in software systems can lead to inappropriate behavior and even cause major financial losses and significant security mishaps. Conventionally, testing and code review methods have been adopted to detect and fix defects in software systems. Nevertheless, in current days testing all components in increasingly big-scale and complex software systems may be highly time-consuming and impracticable. Because of the insufficient resources, defect prediction, which concentrates on automatically identifying the most fault-prone modules, has pulled intense attention in software engineering. Various models such as GNB, BNB, RF, C4.5, SVM, and MLP have been applied throughout this SDP research study. Ensemble classifiers have been designed for SDP, which merge GNB, BNB, RF and C4.5, SVM and MLP in order. The “Feature Extraction (FE)” mechanism known as PCA is paired with the ensembles 1, 2, and 3 classifiers in the third scenario. Class balance is merged with the ensembles 1, 2, and 3 classifiers in the fourth scenario. At last, PCA and ensembles 1, 2, and 3 are combined for the study of SDP. It is analyzed that class balancing with PCA and ensemble 2 gives a higher performance as accuracy is achieved up to 93.33 percent.

Other angles of analyses exist in which “Software Defect Prediction” could be seen and potentially for better results based on the current study. It is a question of the “comparison of the proposed models with other models of SDP to analyze reliability better.” Another aspect could be “the improvement of the proposed model using supervised learning models.”

6. REFERENCES

[1] P Lakshmi, T. LathaMaheswari, “An effective rank approach to software defect prediction using software metrics”, 2016, 10th International Conference on Intelligent Systems and Control (ISCO)

[2] Prianka Mandal, Amit Seal Ami, “Selecting best attributes for software defect prediction”, 2015, IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)

[3] Misha Kakkar, Sarika Jain, Abhay Bansal, P.S. Grover, “Evaluating Missing Values for Software Defect Prediction”, 2019, International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)

[4] A Shanthini, R M Chandrasekaran, “Analyzing the effect of bagged ensemble approach for software fault prediction in class level and package level metrics”, 2014, International Conference on Information Communication and Embedded Systems (ICICES2014)

[5] Ling-Feng Zhang, Zhao-Wei Shang, “Classifying feature description for software defect prediction”, 2011, International Conference on Wavelet Analysis and Pattern Recognition

[6] Shiwang Agarwal, Sajal Gupta, Rishabh Aggarwal, Shashank Maheshwari, Lipika Goel, Sonam Gupta, “Substantiation of Software Defect Prediction using Statistical Learning: An Empirical Study”, 2019, 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)

[7] Jun Wang, Beijun Shen, Yuting Chen, “Compressed C4.5 Models for Software Defect Prediction”, 2012, 12th International Conference on Quality Software

[8] Mahesha Bangalore Ramalinga Pandit, Nitin Varma, “A Deep Introduction to AI Based Software Defect Prediction (SDP) and its Current Challenges”, TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)

[9] Martin Shepperd, David Bowes, Tracy Hall, “Researcher Bias: The Use of Machine Learning in Software Defect Prediction”, 2014, IEEE Transactions on Software Engineering, Volume: 40, Issue: 6

[10] LameesAlhazzaa, Anneliese Amschler Andrews, “Trade-Offs between Early Software Defect Prediction versus Prediction Accuracy”, 2019, International Conference on Computational Science and Computational Intelligence (CSCI)

[11] SukmawatiAnggraeni Putri, Frieyadie, “Combining integrated sampling technique with feature selection for software defect prediction”, 2017, 5th International Conference on Cyber and IT Service Management (CITSM)

[12] Yuan Chen, Xiang-heng Shen, Peng Du, Bing Ge, “Research on software defect prediction based on data mining”, 2010, The 2nd International Conference on Computer and Automation Engineering (ICCAE), Volume: 1

[13] MoloukMishmastNehi, Zahra Fakhrpoor, Mohammad R. Moosavi, “Defects in The Next Release; Software Defect Prediction Based on Source Code Versions”, 2018, Electrical Engineering (ICEE), Iranian Conference on

[14] Ning Li, Martin Shepperd, Yuchen Guo, “A systematic review of unsupervised learning techniques for software defect prediction”, 2020, Information and Software Technology, Volume 122, Article 106287

[15] Hua Wei, Changzhen Hu, Shiyou Chen, Yuan Xue, Quanxin Zhang, “Establishing a software defect prediction model via effective dimension reduction”, 2019, Information Sciences, Volume 477, Pages 399-409

[16] Yuchen Guo, Martin Shepperd, Ning Li, “Poster: Bridging Effort-Aware Prediction and Strong Classification - A Just-in-Time Software Defect Prediction Study”, 2018, IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)

[17] Chun Shan, Boyang Chen, Changzhen Hu, JingfengXue, Ning Li, “Software defect prediction model based on LLE and SVM”, 2014, Communications Security Conference (CSC 2014)

[18] K. Punitha, S. Chitra, “Software defect prediction using software metrics - A survey”, 2013, International Conference on Information Communication and Embedded Systems (ICICES)

[19] Kwabena EboBennin, Jacky Keung, PassakornPhannachitta, AkitoMonden, Solomon Mensah, “[Journal First] MAHAKIL: Diversity Based Oversampling

Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction", 2018, IEEE/ACM 40th International Conference on Software Engineering (ICSE)

[20] Fei Wu, Xiao-Yuan Jing, Xiwei Dong, Jicheng Cao, Mingwei Xu, Hongyu Zhang, Shi Ying, Baowen Xu, "Cross-project and within-project semi-supervised software defect prediction problems study using a unified solution", 2017, IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)

[21] Goran Mauša, TihanaGalinacGrbac, Bojana DalbeloBašić, "Software defect prediction with Bug-Code analyzer - A data collection tool demo", 2014, 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM)

[22] Hamdi A. Al-Jamimi, "Toward comprehensible software defect prediction models using fuzzy logic", 2016, 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)

[23] Jing Sun, Xiaoyuan Jing, Xiwei Dong, "Manifold Learning for Cross-project Software Defect Prediction", 2018, 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)

[24] Safial Islam Ayon, "Neural Network based Software Defect Prediction using Genetic Algorithm and Particle Swarm Optimization", 2019, 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)

[25] Ye Xia, Guoying Yan, Huiying Zhang, "Analyzing the significance of process metrics for TT&C software defect prediction", 2014, IEEE 5th International Conference on Software Engineering and Service Science

[26] T P Pushphavathi, "An approach for software defect prediction by combined soft computing", 2017, International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)

[27] Tanvi Sethi, Gagandeep, "Improved approach for software defect prediction using artificial neural networks", 2016, 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)

[28] Ye Xia, Guoying Yan, Xingwei Jiang, Yanyan Yang, “A new metrics selection method for software defect prediction”, 2014, IEEE International Conference on Progress in Informatics and Computing

[29] Yan NaungSoe, Paulus InsapSantosa, Rudy Hartanto, “Software Defect Prediction Using Random Forest Algorithm”, 2018, 12th South East Asian Technical University Consortium (SEATUC)

[30] Md FahimuzzmanSohan, Md Ismail Jabiullah, Sheikh Shah Mohammad Motiur Rahman, S M Hasan Mahmud, “Assessing the Effect of Imbalanced Learning on Cross-project Software Defect Prediction”, 2019, 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)

[31] Ruchika Malhotra, Nakul Pritam, Yogesh Singh, “On the applicability of evolutionary computation for software defect prediction”, 2014, International Conference on Advances in Computing, Communications and Informatics (ICACCI)

[32] He Can, Xing Jianchun, Zhu Ruide, Li Juelong, Yang Qiliang, XieLiqiang, “A new model for software defect prediction using Particle Swarm Optimization and support vector machine”, 2013, 25th Chinese Control and Decision Conference (CCDC)

[33] Biwen Li, Beijun Shen, Jun Wang, Yuting Chen, Tao Zhang, Jinshuang Wang, “A Scenario-Based Approach to Predicting Software Defects Using Compressed C4.5 Model”, 2014, IEEE 38th Annual Computer Software and Applications Conference

[34] Taek Lee, Jaechang Nam, Donggyun Han, Sunghun Kim, Hoh Peter In, “Developer Micro Interaction Metrics for Software Defect Prediction”, 2016, IEEE Transactions on Software Engineering, Volume: 42, Issue: 11

[35] ChakkritTantithamthavorn, Shane McIntosh, Ahmed E. Hassan, Kenichi Matsumoto, “Comments on “Researcher Bias: The Use of Machine Learning in Software Defect Prediction”, 2016, IEEE Transactions on Software Engineering, Volume: 42, Issue: 11

[36] Yu Qu, Ting Liu, Jianlei Chi, YangxuJin, Di Cui, Ancheng He, Qinghua Zheng,

"node2defect: Using Network Embedding to Improve Software Defect Prediction", 2018, 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)

[37] Chun Shan, Hongjin Zhu, Changzhen Hu, Jing Cui, JingfengXue, "Software defect prediction model based on improved LLE-SVM", 2015, 4th International Conference on Computer Science and Network Technology (ICCSNT), Volume: 01

[38] Yan Zhou, Chun Shan, Shiyou Sun, Shengjun Wei, Sicong Zhang, "Software Defect Prediction Model Based On KPCA-SVM", 2019, IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)

[39] Fuqun Huang, Lorenzo Strigini, "Predicting Software Defects Based on Cognitive Error Theories", 2018, IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)

[40] Fei Wu, Xiao-Yuan Jing, Ying Sun, Jing Sun, Lin Huang, Fangyi Cui, Yanfei Sun, "Cross-Project and Within-Project Semisupervised Software Defect Prediction: A Unified Approach", 2018, IEEE Transactions on Reliability, Volume: 67, Issue: 2

[41] Ling Xu, Bei Wang, Ling Liu, Mo Zhou, Shengping Liao, Meng Yan, "Misclassification Cost-Sensitive Software Defect Prediction", 2018, IEEE International Conference on Information Reuse and Integration (IRI)

[42] Bharti Bisht, Parul Gandhi, "Review Study on Software Defect Prediction Models premised upon Various Data Mining Approaches", 2019, 6th International Conference on Computing for Sustainable Global Development (INDIACom)

[43] Lu Liu, Kewen Li, Mingwen Shao, Wenying Liu, "Fuzzy Integral Based on Mutual Information for Software Defect Prediction", 2015, International Conference on Cloud Computing and Big Data (CCBD)

[44] Jiaxi Xu, Liang Yan, Fei Wang, Jun Ai, "A GitHub-Based Data Collection Method for Software Defect Prediction", 2019, 6th International Conference on

Dependable Systems and Their Applications (DSA)

[45] Md. FahimuzzmanSohan, Md Alamgir Kabir, Md. Ismail Jabiullah, Sheikh Shah Mohammad Motiur Rahman, “Revisiting the Class Imbalance Issue in Software Defect Prediction”, 2019, International Conference on Electrical, Computer and Communication Engineering (ECCE)

[46] Ye Xia, Guoying Yan, Qianran Si, “A Study on the Significance of Software Metrics in Defect Prediction”, 2013, Sixth International Symposium on Computational Intelligence and Design, Volume: 2

[47] Xuan Huo, Yang Yang, Ming Li, De-Chuan Zhan, “Learning Semantic Features for Software Defect Prediction by Code Comments Embedding”, 2018, IEEE International Conference on Data Mining (ICDM)

[48] Kang Yang, Huiqun Yu, Guisheng Fan, Xingguang Yang, Song Zheng, ChunxiaLeng, “Software Defect Prediction Based on Fourier Learning”, 2018, IEEE International Conference on Progress in Informatics and Computing (PIC)

[49] Mohamed Samir, Mohammad El-Ramly, Amr Kamel, “Investigating the Use of Deep Neural Networks for Software Defect Prediction”, 2019, IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)

[50] Kazuya Tanaka, AkitoMonden, Zeynep Yücel, “Prediction of Software Defects Using Automated Machine Learning”, 2019, 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)

[51] EbubeoguAmarachukwu Felix, Sai Peck Lee, “Integrated Approach to Software Defect Prediction”, 2017, IEEE Access, Volume: 5

[52] Zhang Tian, Jing Xiang, Sun Zhenxiao, Zhang Yi, Yan Yunqiang, “Software Defect Prediction based on Machine Learning Algorithms”, 2019, IEEE 5th International Conference on Computer and Communications (ICCC)

[53] Md Alamgir Kabir, Jacky W. Keung, Kwabena E. Benniny, Miao Zhang, “Assessing the Significant Impact of Concept Drift in Software Defect Prediction”, 2019, IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)

[54] Ying Liu, Fengli Sun, Jun Yang, Donghong Zhou, “Software Defect Prediction Model Based on Improved BP Neural Network”, 2019, 6th International Conference on Dependable Systems and Their Applications (DSA)

[55] Guisheng Fan, XuyangDiao, Huiqun Yu, Kang Yang, Liqiong Chen, “Deep Semantic Feature Learning with Embedded Static Metrics for Software Defect Prediction”, 2019, 26th Asia-Pacific Software Engineering Conference (APSEC)

[56] Houleng Gao, Minyan Lu, Cong Pan, Biao Xu, “Empirical Study: Are Complex Network Features Suitable for Cross-Version Software Defect Prediction?”, 2019, IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)

[57] Jian Li, Pinjia He, Jieming Zhu, Michael R. Lyu, “Software Defect Prediction via Convolutional Neural Network”, 2017, IEEE International Conference on Software Quality, Reliability and Security (QRS)

[58] Yuanxun Shao, Bin Liu, Shihai Wang, Guoqi Li, “Software defect prediction based on correlation weighted class association rule mining”, 2020, Knowledge-Based Systems, Volume 19621, Article 105742

RESUME

AKIM AYENA SOULE AMIDOU

EDUCATIONAL INFORMATION

Master 2019-2022	Akdeniz University Engineering Faculty, Computer Engineering, Antalya
Bachelor 2013-2018	Antalya International University Engineering Faculty, Computer Engineering, Antalya

PROFESSIONAL AND ADMINISTRATIVE DUTIES

Frontend & tester 2018-present	Akdeniz University Hotech, Teknokent ArGe-3 Binasi kat:1, Antalya
Full-Stack development 2017-2018	Akdeniz University EticSoft, Teknokent ArGe-1 Binasi 5/C, Antalya