T.C.

ALTINBAS UNIVERSITY

Institute of Graduate Studies

Information Technologies

A NEW FRAMEWORK FOR SOFTWARE DEFECT PREDICATION USING ENHANCED AI TECHNIQUE

Isam Shishakhan Taaban AL-HASNAWI

Master Thesis

Supervisor

Asst. Prof. Dr. Abdullahi Abdu IBRAHIM

Istanbul, 2021

A NEW FRAMEWORK FOR SOFTWARE DEFECT PREDICATION USING ENHANCED AI TECHNIQUE

by

Isam Shishakhan Taaban AL-HASNAWI

Information Technologies

Submitted to the Graduate School of Science and Engineering

in partial fulfillment of the requirements for the degree of

Master of Science

ALTINBAŞ University

2021

The thesis titled “A NEW FRAMEWORK FOR SOFTWARE DEFECT PREDICATION USING ENHANCED AI TECHNIQUE” prepared and presented by Isam Shishakhan Taaban AL-HASNAWI was accepted as a Master of Science Thesis in Information Technologies.

_____

Asst. Prof. Dr. Abdullahi Abdu IBRAHIM

Supervisor

Thesis Defense Jury Members:

Asst. Prof. Dr. Abdullahi Abdu IBRAHIM	Faculty of Engineering and Natural Science department of Computer Engineering Altinbas University	_____
Asst. Prof. Dr. Ahad Khaleghi Ardabili	Faculty of Engineering and Natural Science department of Civil Engineering Altinbas University	_____
Asst. Prof. Dr. Sewale Musadaq Taha	Faculty of Communication department of Digital game Design, Beykent University	_____

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Approval Date of Institute of Graduate Studies:

____/____/____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Isam Shishakhan Taaban AL-HASNAWI

DEDICATION

Thank you to my academic adviser who guided me in this process and the committee who kept me on track.

ACKNOWLEDGEMENTS

I would like to thank Dr. Abdullahi Abdu IBRAHIM supervisor for his help and assistance throughout my work All of me have been supported by my close friends to them all thanks and appreciation

ABSTRACT

A NEW FRAMEWORK FOR SOFTWARE DEFECT PREDICATION USING ENHANCED AI TECHNIQUE

AL-HASNAWI ,Isam

MSc, Information Technologies, Altınbaş University,

Supervisor: Asst. Prof. Dr. Abdullahi Abdu IBRAHIM

Date: June, 2021

Pages: 58

In this thesis, new framework planned for software defect estimation using LSTM using TSA. The proposed framework combined AI technique with TSA to optimize the defect predication accuracy. Then, in the first stage the software defect dataset become input to the PCA. Furthermore, the TSA applied to optimize the bias and weight of the LSTM. The proposed framework validated by using number of datasets proposed by NASA datasets to improve the framework results. The experimental results presented by using by calculating several parameters to evaluate the proposed method. Additionally, the obtained results compared with various studies proposed in this field.

Keywords: AI, Tree Seed Algorithm, Software Defect Predication, LSTM.

ÖZET

GELİŞTİRİLMİŞ AI TEKNİĞİNİ KULLANAN YAZILIM HATASI TAHMİNİ İÇİN YENİ ÇERÇEVE

AL-HASNAWI ,Isam

Yüksek Lisans, Bilişim Teknolojileri, Altınbaş Üniversitesi,

Danışman: . ABDULLAHI ABDU IBRAHIM

Tarih: Haziran, 2021

Sayfa Sayısı: 58

Bu tezde, TSA kullanılarak LSTM kullanılarak yazılım hata tahmini için yeni bir çerçeve planlanmıştır. Önerilen çerçeve, kusur tahmin doğruluğunu optimize etmek için AI tekniğini TSA ile birleştirdi. Ardından, ilk aşamada yazılım hatası veri seti PCA'ya girdi haline gelir. Ayrıca, TSA, LSTM'nin önyargısını ve ağırlığını optimize etmek için uygulanmıştır. Önerilen çerçeve, çerçeve sonuçlarını iyileştirmek için NASA veri kümeleri tarafından önerilen veri kümelerinin sayısı kullanılarak doğrulanmıştır. Önerilen yöntemi değerlendirmek için çeşitli parametreler hesaplanarak deneysel sonuçlar sunulmuştur. Ek olarak, elde edilen sonuçlar bu alanda önerilen çeşitli çalışmalarla karşılaştırılmıştır..

Anahtar kelimeler: AI, Ağaç Tohumu Algoritması, Yazılım Kusur Tahmini, LSTM.

CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

SVM : Support vector machine

NN : Neural Network

RBF : Radial Basis Function

TSA : Tree Seed Algorithm

LSTM : Long Short-Term Memory

PCA : Principal Component Analysis

1. INTRODUCTION

With the widespread use of electronic technologies, project management has become necessary for software that gains importance day by day and whose scope is expanding. One of the most important factors in software project management is that the project is predictable. Making the projects as specific as possible in terms of both cost and workforce planning eliminates possible risks. Software effort estimates are an important part of software development efforts and provide the necessary inputs for feasibility analysis, bidding, budgeting and planning. It is vital to accurately estimate the development time to develop a software product. Because there is serious competition for quality software in the software industry. Estimates that differ from the effort required may result in losses in many aspects, especially cost and quality. Today, the most common methods used for effort estimation are methods based on human judgment. However, human judgment may not be reliable enough as it can be affected by many conditions. In addition, the application process of methods based on human judgment may cause a serious workload as the forecast items increase [1]. In recent years, the rapidly changing dynamics of the market have increased the tendency towards agile methods in software project management processes compared to traditional methods. Story scores are the most commonly used metric for predicting effort in software projects where agile project management methodology is applied. In today's operation, these estimates are usually made intuitively manually by the relevant people for each request, and then these estimates are controlled by the manager of the relevant unit. However, in its current form, this process is a process with low consistency and continuity, despite using more human resources [2,3].

Figure 1.1: Software Engineering Procedures [4].

1.1 GOAL OF THE THESIS

This study solves and explain malware attacks problem which analysis the previous studies and determine the weakness of these studies and try to fixed and presented new approaches in investigating this problem, there is number of points that will investigate in this research:

a. Presented new AI based framework for software defect detection.
b. The proposed framework presented effective results in minimum execution time.
c. Compared the presented framework with well-known studies proposed in this field.

1.2 THE PROBLEM STATEMENT

In this research, the several issues are investigated to deal with software defect detection problem.

i. How detect software defect?

ii. How develop accurate framework to detect defects by using AI techniques?

iii. How detect software defect in minimum execution time?

1.3 CONTRIBUTIONS

In this study several contributions are presented in comparision with previous studies:

i. New software defect framework presented by using AI technqiues.

ii. Detect software defect in minimum time and high accurate.

iii. Verified using several datasets and compared with common researches in this field and presented remarkable results.

1.4 THESIS STRUCTURE

The letter includes five chapters prepared as follows:

a. Chapter I: Introduction, Goal of the thesis, problem statement, contributions, thesis structure

b. Chapter II: Literature Review: This chapter provides an overview of the work related to the detection of software defects anomalies.
c. Chapter 3: Methodology: This chapter outlines the research methodology used in this thesis. AI techniques, the proposed method.
d. Chapter 4: Experimental Results
e. Chapter 5: Conclusion and Future Work.

2. OVERVIEW

2.1 SOFTWARE PROJECT MANAGEMENT

Project management is the efforts of the project manager and the project organization to plan, organize, coordinate, control and resource planning in order to achieve a system management approach towards project specific goals. Software project management is the activities that involve people, the software product, the project management process such as planning and monitoring. Basic tasks of software project management include developing the project implementation plan, organizing resources, quality management and cost management.

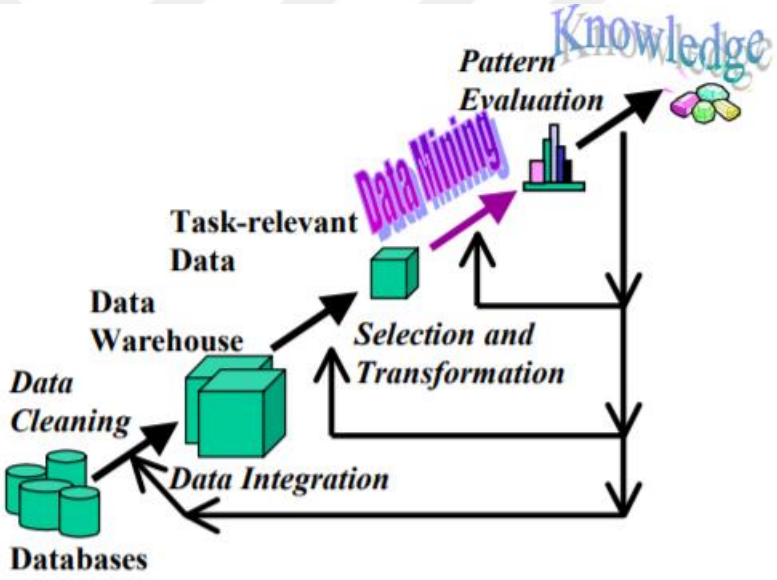

Figure 2.1: Data Mining is Score of Knowledge Discovery Process [5]

Software project management covers the definition, development and maintenance of software. Preventive and supportive software engineering activities that continue throughout the maintenance process are also included in project management. In a successful software engineering organization, software is completed in accordance with the resource, cost, schedule and quality determined within the scope of production, application development and maintenance tasks. In this process, the purpose of software project management is to advance and complete the project in accordance with the planned schedules, costs and other requirements. Therefore, the project management process has a defined beginning and end. When the project is

finished, project teams move on to other projects. Project management includes many variables. Therefore, there is no single way to manage a project. The same project can be managed with different methodologies. Project management methodology, in other words software process models, can be defined as procedures created to manage projects effectively. The project management process or methodology chosen for the development of the project according to its purpose and objectives constitutes the software development model. The forecast is a crucial and integral part of the software development lifecycle. Software cannot be improved without effort and cost estimates. It is important to predict as accurately as possible. Both high pricing and low pricing can hurt the software developer business [6-11]. Software effort estimation plays an important role in improving the software development process. The critical aspect of cost in software development is human effort. Therefore, most cost estimation techniques focus on this issue. Software effort estimation is one of the steps aiming to produce a software development project on time, within budget and in quality to meet the requirements. One of the biggest concerns of most developers is their uncertainty due to the prediction being made at an early stage of the development process [12]. The risk arising from this uncertainty increases as the magnitude of prediction errors increases for a software project that can be very costly in terms of resources to be allocated to the project [13-18]. The sensitivity and reliability of predicting the effort of software projects is very important for software companies' competitiveness. The use of software is constantly increasing today. Software companies need to produce high quality and timely software to ensure their competitive power. Due to the high competition environment, the software industry forces companies to produce software projects on time and on budget. The increasing need for planning and project management requires more and more attention and control of software project managers. Effort estimation is one of the most important steps in software planning and management.

The processes related to the management of a project are mainly carried out by the Project Manager and are placed in parallel with the processes of technical realization of the solution or product of the project (life cycle: design, implementation, testing and release of the solution). It is convenient to group the activities that make up the management processes into three main phases1, based on their nature and timing [19].

The management processes are organized in such a way as to provide a complete set of activities to start the project (first phase), for the execution and management of technical activities (second phase) and to close it (third phase) [20].

The activities for the management are sequential and repetitive by nature, and can be performed, according to specific needs, or on a daily, weekly, monthly basis, or they can be triggered by particular events and be carried out continuously for a certain time. Another very useful function of Project Management is the progressive breakdown of both the activities to be carried out (i.e. the WBS defined in the previous chapter), and the organizational entities (for example, individuals, individual offices, etc.) that carry them out. . By Analytical Structure of the Organization or Structured Breakdown of the Organization, usually indicated by the English acronym OBS (Organization Breakdown Structure) we mean a standard methodology for breaking down the organization to assign the responsibilities of the individual activities to the elementary components in a timely manner of the organization [21]. The units are usually individual people but can also represent offices or other hierarchical or functional subdivisions of the company. Within a project, the OBS is usually created after the WBS in a complementary way, to identify the responsibilities in the individual activities, thus transforming the activities into tasks assigned to specific people or divisions. Therefore the OBS is not necessarily the same organizational subdivision expressed by the organization chart of a company or entity, but it can also be built ad hoc [22]. We often talk about software "project", meaning by this term all the work from the idea to the realization of the software itself, which is also understood as a unit, resulting from a contract between a customer and a supplier and quantifiable in terms of price. But the software "project" can also be defined as the organization of the software development process, ie the application of project management to this development process. Organization is necessary for a team of developers, under penalty of loss of control of the project itself. Before the beginning of the actual coding phase, that is, the drafting of the source code, the requirements to be achieved must be established and, at least, the division of work among the members of the working group [23]. In small projects the analysis is almost always a phase concluded once and for all at the beginning of the software product realization process, in large projects instead the analysis is often carried out iteratively in the process itself. The particular development process used, with its software life cycle model, defines whether or not the analysis is repeated, as well as other subsequent phases. The objective of the analysis phase is to produce a complete

formalized description, with the right level of detail, of everything the system has to do (functional requirements), the environment in which it will have to operate (non-functional requirements) and the constraints that will have to respect [24-29]. This description must therefore explain what the system will have to do without addressing how it will have to do it, in such a way as to be able to serve as a basis for discussion and possibly a contract between the development team and the client, to arrive at a specification that can be uniquely interpreted by both. leave for the project. The description of the software system to be implemented will therefore follow the black box model, focusing only on the interactions between the system and the outside world, primarily on the user interfaces. The analysis documents, sometimes also called specification documents or simply specifications, are the product resulting from the analysis phase and must clearly contain the information defined above. Their form, the languages and the symbols used are obviously dependent on the type of development process followed. The output of the analysis phase must accurately and unambiguously describe the requirements that the system object of the project must have. Starting from the results of the analysis, the design phase must produce the operating instructions for the actual implementation of the IT project (implementation). The instructions must have the right level of detail and be expressed in the form of structured documents. Therefore, the design of an application consists of activities to identify the best implementation solution with respect to functional objectives, non-functional ones and constraints. These activities can be of various kinds, be carried out in different times and ways based on the approach followed, but in general they help designers and development teams to make important decisions, often of a structural nature. The result of the design is the definition of the system architecture, meaning by this term the structural organization of the system itself, which includes its software components, the externally visible properties of each of them (the component interface) and the relationships between the parties. In analysis the requirements and structure of the system are represented in an abstract form and (theoretically) independent of the technology. The design also takes into account all the factors relating to the use of a concrete technology and therefore, unlike the analysis, the design cannot be carried out regardless of the technology used. During the design, all aspects necessary for an unambiguous implementation must also be defined. A widely used tool in design is the flow chart, or its evolution UML activity diagram. Both tools are used to break down the activities to be

performed into smaller and smaller elements that can be easily implemented with appropriate sets of instructions of the chosen programming language, that is the WBS of programming work.

2.2 THE PHASE OF IMPLEMENTING OR WRITING THE CODE

Writing the source code can be done with many tools, the simplest of which is the basic ASCII file editor. However, due to the complexity that modern object-oriented languages require this approach is too little productive. In fact, having to guarantee compliance with tight deadlines, it is necessary to have all the facilitation functions integrated into a single tool for writing the source code. An integrated development environment (IDE), in Italian integrated development environment, (also known as integrated design environment or integrated debugging environment, respectively integrated design environment and integrated debugging environment) is a software that helps programmers in code development. It usually consists of a source code editor, a compiler and / or interpreter, an automatic building tool, and (usually) a debugger. Sometimes it is integrated with a version control system and with one or more tools to simplify the construction of a GUI. Some IDEs, aimed at object-oriented software development, also include a class browser, an object analyzer, and a class hierarchy diagram [30]. 20 Although some multi-language IDEs are in use, such as Eclipse, NetBeans, and Visual Studio, IDEs generally target a specific programming language, such as Visual Basic or Delphi. The writing of the source code, although often little considered, however, remains the fundamental phase of any IT project [31]. The analysis and design may have been done well, but, if the code is poorly written, the resulting application will have problems and malfunction. Making an analogy in the field of civil engineering, it is clear that even the most beautiful project, if the pillars are not made well, if the bricks are not laid with accuracy or if the materials used are of poor value, will give rise to a poor quality building [32]. For this reason, even modern programming methodologies tend to make the code writing phase the main one of an entire IT project (see for example [Web Agile]). During the drafting of the code, the first debugging also takes place, i.e. the removal of syntax errors and the most obvious errors, such as the failure to initialize variables, which can compromise the compilation or operation of the program. Once the program has been completed or in any case can begin to function, it is necessary to verify that its operation complies with all the specifications that were established in the analysis phase. This is the fundamental purpose of the testing phase. Errors that lead to non-specs are usually much

more insidious to uncover than those found during debugging. They are not syntax errors, but logical and conceptual errors, such as, for example, writing the '+' sign instead of the '-' sign within an algorithm that requires subtraction and not sum. The test phase therefore involves verifying the actual behavior of the program with respect to the expected one and reporting the differences in behavior to the programmers who will have to search for and eliminate the causes of these differences [33].

2.3 INTRODUCTION TO UML

Unified Modeling Language (UML) is a unified semi-graphic language for modeling concepts, entities, functionalities, processes and relationships that exist between them for further information). UML was born in 1997, unifying previous modeling syntaxes and today it is an international standard managed by the OMG consortium). After 1997, the standard extensions of UML follow one another and also the use is extended to the various stages of implementation of IT systems, also including the analysis phase of business processes (also referred to as "business analysis", see in this regard and) [34]. In fact, over time, the need for a clear language common to all components of the information system has emerged, from business to technology. In the Use Case Diagrams, the main use cases of the system are identified, which are then described in detail as a succession of interactions between an actor (who can identify a human operator with a specific role or external system) and the system object of the analysis that will have to be realized [35]. The detail of the individual use cases thus becomes a succession of elementary interactions (requests) performed by the operator, the active subject of the situation, and the resulting system responses. Let's now try to define more precisely what a use case and its diagram represent. In a formal way, we can define a use case or use case as: a sequence of transactions, performed by an actor interacting with the system, which provides a measurable value for the actor. The link between the use case and the activity is therefore clear. passive, i.e. participates in a use case, but does not initiate it. An example of an active actor is the clerk who enters data into a management system, while an example of a passive actor is the cashier operator of a supermarket who participates in the use case purchase of items, which however is initiated by the customer. Let's now see the graphical syntax of the Use Case Diagrams. A generic use case is represented, where the actor is the stylized man, the system is explained with a rectangle and the use case with an ellipse. The use case in the subsequent stages of analysis is

expanded in the sequence of transactions, or single actions or operations, which can not be further broken down, which compose it [36]. The use of diagrams serves, once the individual use cases have been identified, to explain both the relationships with the actors, and above all any relationships of dependence or temporal succession that exist between the various use cases. Since these relationships are not clearly explained by the Use Case Diagrams, there is an IBM methodology that provides for passing from the Use Case Diagram to a corresponding Activity Diagram, where the individual activities correspond to the individual use cases and the succession of steps from a activity to the other identifies the chronological navigation between the use cases, or between the user interface masks that originate from them during implementation. In this way, the navigation diagram between the program windows can be constructed, as will be shown in the examples in the next chapter. connects an actor or use case to another more general. The derivative specializes the parent by adding new characteristics to it. The include relationship, expresses a dependence between use cases; the included case is part of the behavior of the one that includes it. The inclusion is not optional and takes place in every instance of the use case. The correct execution of the use case it includes depends on that of the included use case. Cycles of include cannot be formed. In practice, it graphically represents the action of the WBS on the set of interactions that the use case encompasses. Usually the include relationship is used to reuse parts common to multiple use cases. expresses a different dependence between use cases (note the direction of the arrow, which goes from the extended case to the extended one). The use case that extends (client) specifies an increase in behavior to the extended one (supplier) [37]. This is additional and optional behavior that occurs in particular or non-standard cases. The extension is different from a generalization between use cases: in a generalization, both use cases are equally significant, while, in an extend, the client does not necessarily make sense if taken alone. A use case reached by at least one extend can optionally display its own extension points, i.e. the points and / or conditions of execution in which the behavior is extended. an example is shown, in which the customer registration must take place in the condition "customer not (yet) registered", described by the extension point. they identify the entities and the relationships or associations that exist between them. Entities essentially represent the formal model associated with the concept they express (for example, people, products, orders, invoices), obtained through a process of abstraction from reality. The associations identify the relationships that bind the entities, including numerical links, defined by

the multiplicities (for example, a car has 4 wheels, a company has many employees, a car has only one owner at a given time). Sometimes it is necessary to indicate not abstract entities, but concrete instances of such entities (for example not a generic teacher, but Professor Giulio Destri, not a generic manager, but the director of personnel Dr. Guido Ventura, not a generic product, but Mantovani soap). The diagrams related to this particular case are defined as Object Diagrams, they are similar to Class Diagrams, but for each concrete entity within them, they define the unique name of the object (i.e. the modeled real world object) and its class of belonging (i.e. the abstract category to which it belongs, defined with the name of the class it belongs to). To define the Class Diagrams, it is also necessary to define the terminology of the real system being analyzed, precisely identifying the entities (people, roles, places, material objects, events, structures, etc.) involved in the real world system (i.e. of the business domain) which are important for the information system that will have to manage it and for the IT components within it. The terminology is defined through a specific document: the glossary, which will be presented with examples in the next chapter. The Class Diagrams are an evolution of Entity-Relationship diagrams and are also used to design the Databases designed to contain and make persistent (i.e. permanent until they are explicitly deleted) the company data that make up the information within the company. . In class diagrams the elementary components are the classes themselves, represented as rectangles if they identify elements of the real world, even abstract ones.

The class diagrams also express relationships of a logical type, of dependence, of derivation and of inclusion, through appropriate symbologies. The general bond that two classes can have with each other is called association. Next to the simple association we find the association with multiplicity, which indicates a numerical value, that is, a cardinality associated with a bond. For example, a car has four wheels (plus a spare), a bicycle has two wheels and so on; this information is indicated by the numbers next to the bond, as indicated. If the link expressed by the association has characteristics that can be expressed as attributes, or in any case the multiplicity of the association is many-to-many, it may be necessary to introduce the concept of association class. The association class expresses and “gives dignity of entity” to a logical link, such as the ownership of a car, characterized by start dates and end dates. In the same the UML elements note (identified by the graphic symbol of the post-it) and constraint (post-it with the addition of the graphic brackets to enclose the text are also introduced. These elements can be

used in any UML diagram. a link of "inclusion" of one entity (or of a group of entities) in another, the symbols of aggregation and composition, indicated respectively, with the white (empty) and the black rhombus can be used (peino). Aggregation expresses a bond of inclusion in which, however, the included element also exists without the inclusion. Conversely, in the composition, the included element cannot exist without the inclusion, therefore, in case of cancellation of the latter, all the elements included through the composition must also be eliminated. The order includes the order lines through a composition and (so we can read: the order is made up of order lines) and each order line includes a product, with multiplicity one or more (so it aggregates a certain amount of products, for example 10 mobile phones). The cancellation of the order also provides for the cancellation of the order lines that compose it, but not of the products that they aggregate. Another type of logical link is derivation or inheritance, represented by the link with the empty triangle, which expresses the specialization of one class in another, for example the (generic) person specializes in the employee who in turn specializes in employed, in the worker and in the manager. If followed in the opposite direction, this logical link is instead a generalization. The derivation is common to other UML diagrams, as already seen for the Use Case Diagram. A limitation of Class and Object Diagrams is to express only “static” links between entities, without placing emphasis on the dynamic interactions that occur between them. On the other hand, the Use Case Diagrams describe interactions only at a very high level of abstraction, deliberately neglecting many internal features of the systems. They also do not represent entities on an equal footing, placing emphasis on the role of the actor. For this reason, to clearly evaluate a dynamic behavior, interaction diagrams are added, of which very important are the Sequence Diagrams or Communication Diagrams, which define the interactions between entities. The interactions are practically flows of information that flow between classes and, normally, represent commands, or service invocations made from one class to another, or transfer of process control from one class or object to another. The two types of diagrams are semantically equivalent, but the sequence diagram temporally orders the sequence of messages and this makes it a better tool for understanding time constraints, while the communication diagram highlights the dependency links between classes, and therefore between the entities that they represent, for the execution of the activities. The basic syntactic elements of the two diagrams are shown in. Sequence Diagrams consist of a set of objects under which the downward-directed temporal axes are located. The messages that the objects exchange are

expressed through the arrows and in correspondence with the duration of the action performed by the objects in response to the messages is the white rectangle on the axis. Objects can sometimes also be directly replaced by classes, indicating that at that moment the role of the category is represented and not the individual component of the category. Shown is a sequence diagram with two types of interactions, the first is asynchronous and therefore the caller does not wait for an answer from the callee, while the second is synchronous and, at the end of the execution of what is requested, the callee must return a result to the caller (return value). The synchronous interaction expressed both with a sequence diagram and with the equivalent communication diagram is shown. In the latter, it is necessary to specify the sequence of message numbers, as the ordering element of the time axis facing down is missing.

2.4 SOFTWARE DEFECT

Software defect prediction studies; It helps programmers to focus on the code that may be prone to error in the development process, increasing the quality of the software and using the resources more effectively. Software error prediction studies are generally collected under 3 different categories on metrics, data sets and methods. In this study, software error prediction data sets containing method-level software metrics experimental results using machine learning methods is obtained. Metrics: method-level, class-level, component-level, file-level, process-level, quantitative-level It is in different categories such as. Software metrics briefly providing a concrete expression of software quality, are measurable values or sets of values. Software bug prediction in the field of research, method-level software metrics are commonly used metrics. These metrics are machine It is popularly used with learning algorithms. The dataset is where raw data on a subject is kept structure. Public software bug prediction dataset their stores are increasing day by day. One of them is space PROMISE dataset belonging to the NASA organization conducting the research store. These software data sets are creating models and obtaining experimental results makes it possible [38, 39].

In addition, these data sets allow discussions and developments on the method and results among researchers. CM1, JM1, KC1, PC1, which are data sets belonging to some software developed by NASA and belonging to the widely used PROMISE data store, are seen. There are a number of metric measurement values and variables in the data sets used. Each record of the dataset has a

class label indicating the known or reported error or non-error status in the respective software module. The presence of the error indicates that changes or updates are required in the relevant software module or other related modules [40]. Although some of the method-level metrics were proposed by Halstead and McCabe in the 1970s, they are still in popular use today. Method-level metrics can also be used for software developed by object-oriented programming that have structured or indeed unique metrics. With method-level metrics, it means that when the error-prone module is predicted in advance, there is probably an error in the relevant module in system or field tests. Evaluations were made on CM1, JM1, KC1 and PC1 software error prediction datasets belonging to some software developed by NASA [41].

A defect is a fault or a bug, in the application which is generated. A developer while scheming and structure the software can brand mistakes or error. These mistakes or errors nasty that there are faults in the software. These are named defects. This Defect statement or Bug account contains of the following info:

i. Defect ID – Each defect or bug has it's single ID number
ii. Defect Description – This contains the theoretical of the subject.
iii. Product Version – This contains the creation version of the application in which the defect is creating.
iv. Detail Steps – This contains the full steps of the subject with the screenshots devoted so that designers can reconstruct it.
v. Date Raised – This contains the Date when the bug is described
vi. Reported By – This contains the facts of the sample who stated the bug like Name and ID
vii. Status – This arena contains the Position of the flaw like New, Allocated, Open, Retest, Confirmation, Shut, Unsuccessful, Delayed, etc.
viii. Fixed by – This area contains the particulars of the designer who secure it like Name and ID.

In 1987, Frederick Brooks says that the presence of bugs in software is not an accident but is due to the very nature of software, in other words, there are bugs in software because they are software. . He also says that there is no silver bullet - a miracle tool - to ward off bugs, here alluding to a legend from the Middle Ages that only a silver bullet, suggestive of the color of the Moon, can ward off the werewolf.

Software is invisible and intangible products, its modification does not require raw material. The very rapid evolution of the IT market generates a strong demand for change. All of these factors make changes in software much more frequent than in other products such as automobiles or buildings.

Computers are among the most complex products that man has made, and therefore have a very large number of states. Software is more complex than computers, and unlike an automobile, no part is alike. Compliance with many standards, characteristic of fields close to telecommunications, increases their complexity. Software is moreover invisible products, which cannot be represented in a geometric space, the graphic representations of software often comprise two, even three or four diagrams which each correspond to a different reality.

Bugs can cause software to attempt to perform operations that cannot be performed (exceptions): division by zero, search for non-existent information. These operations - which are never used during the correct functioning of the software - trigger a mechanism both hardware and software which then deactivates the faulty software, which causes a computer crash or a denial of service. A watchdog is an autonomous electronic device used to detect malfunctions. This mechanism is often used with critical systems and industrial computing. The blue screen of death is popular parlance for the Microsoft Windows operating systems decommission message, which is displayed when an exception is detected in the core of the operating system. The Kernel panic is the message displayed under similar conditions on UNIX operating systems. A memory leak is a malfunction caused by a bug in memory allocation operations. With this malfunction, the amount of memory used by the failing software will continuously increase. If the faulty software manages to use almost all of the available memory, it then interferes with the progress of the other software and causes them to malfunction. A segmentation error is a malfunction due to a bug in operations for manipulating pointers or memory addresses. The faulty software will attempt to read or write information to a memory location (segment) that does not exist or that is not authorized for it. The exception detection mechanism then causes the faulty software to be taken out of service.

An integer overflow is a malfunction due to a bug in mathematical calculation operations. The faulty software will attempt to perform a calculation whose result is greater than the maximum authorized value. The exception detection mechanism then causes the faulty software to be taken out of service. A buffer overflow is a malfunction due to a bug. A software which must write information in a determined and limited memory location (buffer memory) exceeds the limits of this location and will then write information on a location intended for another use, this unexpected modification leads to an erratic execution of the software, which can end with a segmentation error or a capacity overflow. It is a common server security vulnerability that is often exploited by hackers. see Exploit. A stack overflow is a malfunction in which the size of a software's execution stack exceeds the capacity of the buffer that contains it, causing malfunctions similar to a buffer overflow. The execution stack is a data structure stored in memory which contains the status of the software automatisms (see process (data processing)), this structure is recorded in a buffer memory whose size is oversized. A stack overflow results from an erroneous sequence due to a bug. A race condition is a malfunction due to a bug, which causes that in the same software two automations which work simultaneously give different results depending on the automatism which ends before the other. A deadlock is a malfunction during which when several automations wait for each other, that is to say they each wait for the other to release the resources it uses to continue. The resources remain locked during the waits, which can block other automatisms and by domino effect block the whole system. A prevention mechanism causes the cancellation of the operation when the waiting time exceeds the admissible timeout. Bugs are the result of human errors during the specification, design, programming and testing of software and hardware. The increasing complexity of software, communication problems, lack of training of engineers and pressure of deadlines and costs during engineering work are factors that tend to increase the number of bugs. Software testing is the first step in dealing with bugs. For practical reasons (cost of work and deadlines), it is not possible to test a software under all the conditions that it could meet during its use and therefore not possible to counter all the bugs: software like Microsoft Word has 850 commands and 1,600 functions, for a total of over 500 million conditions to test. The use of high level programming languages, which facilitate the work of the engineer. The implementation of writing conventions are other preventive techniques intended to decrease the number of bugs. Debugging is the activity of diagnosing and fixing bugs. During online debugging, the engineer runs the software

step by step and after each step performs a series of checks. During post-mortem debugging, the engineer examines software following a computer crash. When the bug is detected and corrected after the software has been distributed, the supplier often provides a patch, that is to say a kit which replaces the faulty parts of the software with those which have been corrected.

2.4.1 Software Defect Categories

The categories disadvantages are: commission errors, omissions errors, clarity errors, speed and capacity errors see Figure 3.1.

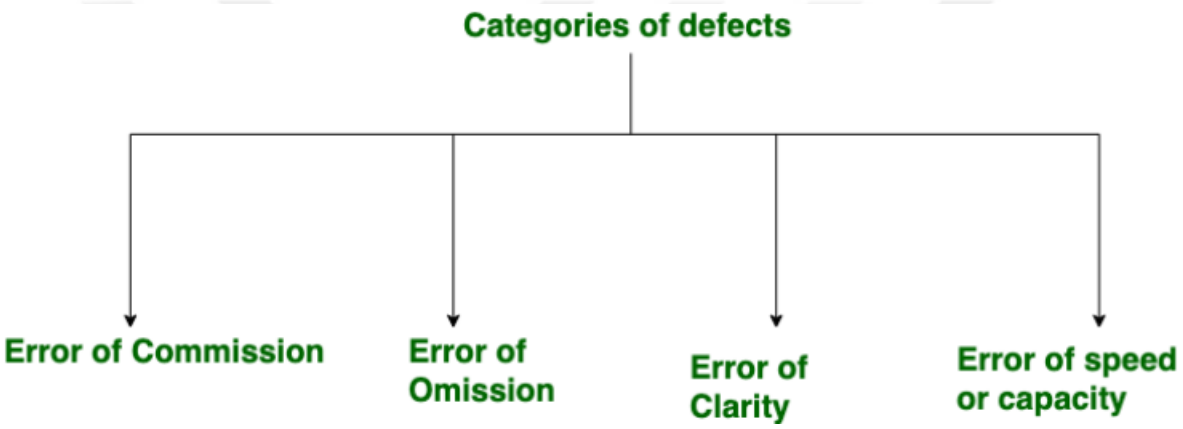

Figure 2.2: Software Defect Categories [42]

- **Error of Commission:**

Commission means instructions or some orders. Now a commission error means that the error occurred in the command or instruction. For example, suppose I wrote an episode I was trying to play 10 times but I ordered it to run more than 10 times by mistake, it's wrong [43].

- **Errors of Omissions:**

As the name already describes the omission error is some thing that happens by mistake. The word omission means something that has been neglected or implemented The most common practical example of this error is that we assume that we make a programming function open its arch but forget to close it at the end [44].

- **Error of Clarity:**

The most common mistake in natural languages. This error is caused by a loss of understanding between the developer and the client. Most of the time goes from the requirements to the program [45].

- **Error of Speed or Capacity:**

The name of the error itself is enough I think to say about this error. The program works fine but does not run at the required time, this is a speed error. When it comes to capacity, it can be memory related. For example, a small integer is advertised where a long integer is required [46].

2.5 SOFTWARE DEFECT DATASET

Software Defect Prediction (SDP) datasets are problematic to discovery and firm to gather. This section aids fellow investigators to discovery the databases that we have composed throughout our investigation. The database is assumed in .csv format with comma as the delimiter and opinion as number spot. First row covers account of metrics: The Columns represented the features of the data and the rows represented the examples [47,48] . The datasets that are used in our thesis listed below:

i. JM1
ii. KC1
iii. CM1

These datasets features are presented in Figure 3.1, Figure 3.2, and Figure 3.3 respectively.

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

Figure 2.3: dataset features

These features are same for all three datasets only the value of each feature different from dataset to another also the number of examples.

3. MATERIAL AND METHODS

3.1 FEATURES EXTRACTION

Extracting features includes decreasing the size of resources needed to define a huge set of features. When analyzing difficult features, a major problem stalks from the amount of variables complicated. Investigation with a huge number of parameters usually needs a huge quantity of memory and computation power, and may reason a cataloguing techniques to fly training examples and poorly circulate them to new examples. Feature selection is a generic term for ways to create sets of variables to overcome these problems while still describing data accurately enough. Many machine learning practitioners trust that correctly enhanced feature selection is the main to making a real model [49].

3.1.1 Principal Component Analysis (PCA)

In statistics, principal component analysis (PCA) is a method of finding the projection of data in a multidimensional space to a lower dimensional space in a way that maximizes the variance. [50] For a set of points in space, the "best fit" line with the least average distance to all points is chosen. Then, the most suitable line is selected among those perpendicular to this line, and these steps are repeated until the variance of a new dimension falls below a certain threshold. The lines obtained at the end of this process form the bases of a linear space. These base vectors are called fundamental components. Basic components of data are independent from each other [51].

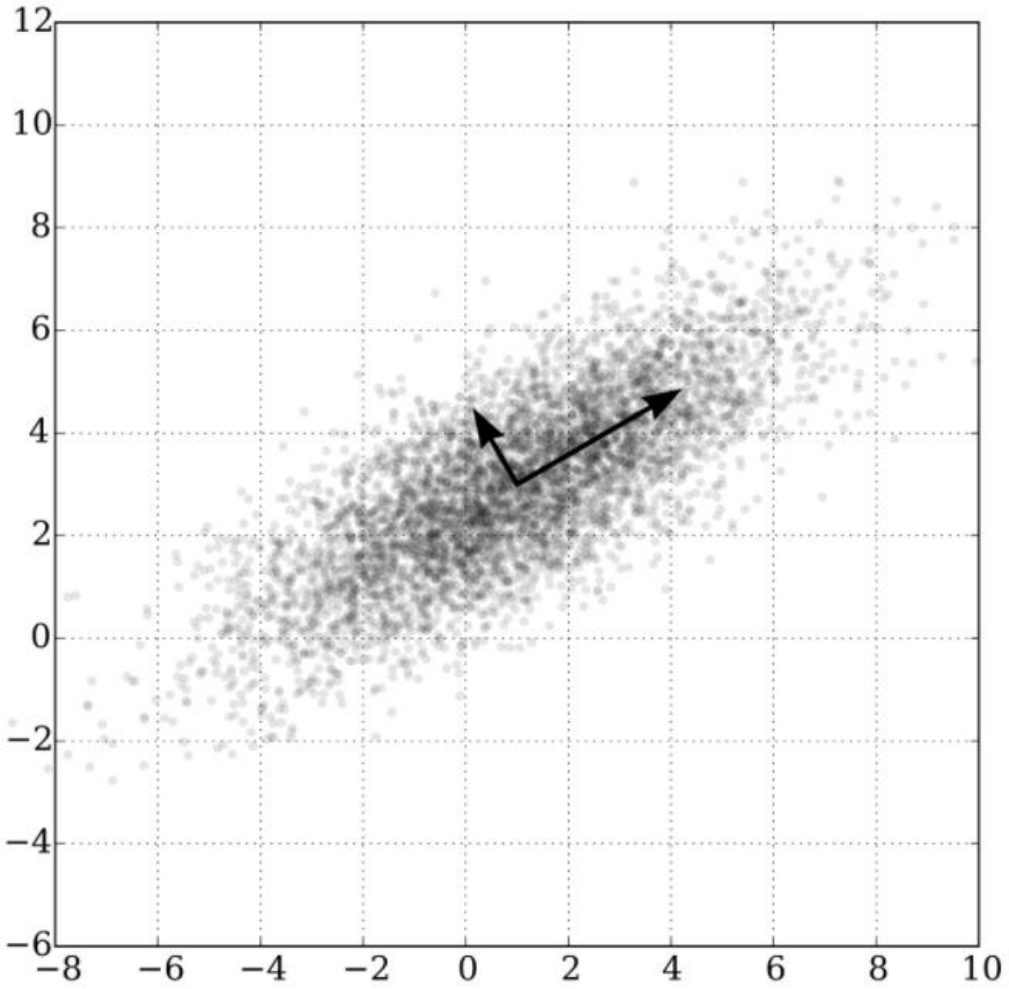

Figure 3.1: Principal Component Analysis

3.1.2 Linear Discriminant Analysis

In statistics, linear discriminant analysis (LDA) or linear discriminant analysis is a method for classifying data by finding a linear combination of features. [52] The model obtained is used as a linear classifier or more commonly in preliminary size reduction analysis. It is sometimes abbreviated LDA (English: Linear discriminant analysis), which is short for the original term. Linear separation analysis is closely related to principal component analysis (PCA) and factor analysis in terms of examining the linear combination of variables in a data that best describes the data. [54] While DAA finds a join that separates the given classes, TBA ignores the classes. Factor analysis differs from DAA in that it examines variance instead of in-class similarity and modeling latent variables.

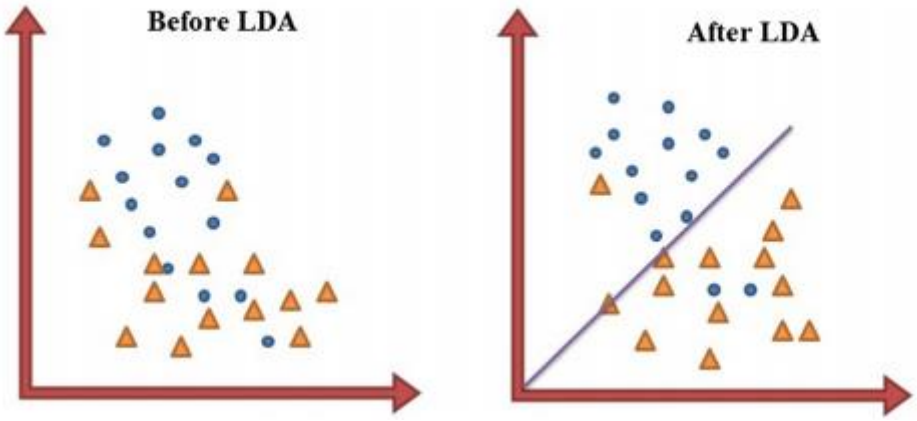

Figure 3.2: Linear Discriminant Analysis

3.1.3 Isomap

Isomap is a non-linear dimensionality reduction method. It is one of many widely used small-size inlay techniques. [55] Isomap is used to compute low-dimensional quasi-isometric embedding of a set of multidimensional data points. The algorithm provides a simple way to estimate the internal geometry of a data distributor based on a rough estimate of the neighbors of each data point in the distributor. Isomap is highly effective and generally applicable to a wide variety of data and measurement sources. Isomap is one of the representatives of isometric display methods and extends the metric multi-dimensional scaling (MDS) by geodetic distances that are given by a weighted graph. More specifically, the classic MDS metric scale performs low-dimensional interleaving based on the paired distance between data points, which is typically measured using the Euclidean direct distance. Isomap is characterized by the use of the geodetic distance, which is induced by the neighborhood graph integrated on the classical scale. This is done to include the collector structure in the resulting nesting. Isomap defines the geodetic distance as the sum of the edge weights on the shortest path between two nodes (e.g. calculated using Dijkstra's algorithm). The n upper eigenvectors of the geodetic distance matrix represent the coordinates in the new n-dimensional Euclidean space [56].

3.2 CLASSIFIERS

Classification is the procedure of forecasting the lesson of assumed features opinions. Lessons are occasionally named as targets/ labels or groups. Cataloguing prognostic demonstrating is the

task of resembling a charting function (f) from input variables (X) to separate production parameters (y).

For instance, spam recognition in email service breadwinners can be recognized as a classification problem. This is s binary classification meanwhile there are only 2 lessons as spam and not spam. A classifier uses approximately training data to comprehend how assumed input parameters tell to the lesson. In this instance, recognized spam and non-spam emails have to be applied as the training features. When the classifier is trained exactly, it can be applied to discover an unknown email [57].

3.2.1 Radial Basic Function Neural Networks

Radial basic function neural networks are feed forward networks that have been trained using the supervised learning algorithm. In these networks, the activation function is usually structured with a single hidden layer selected from a class of functions called elementary functions. Although similar to feedback in many ways, radial basic function networks have different advantages. Usually they run much faster than feedback networks.

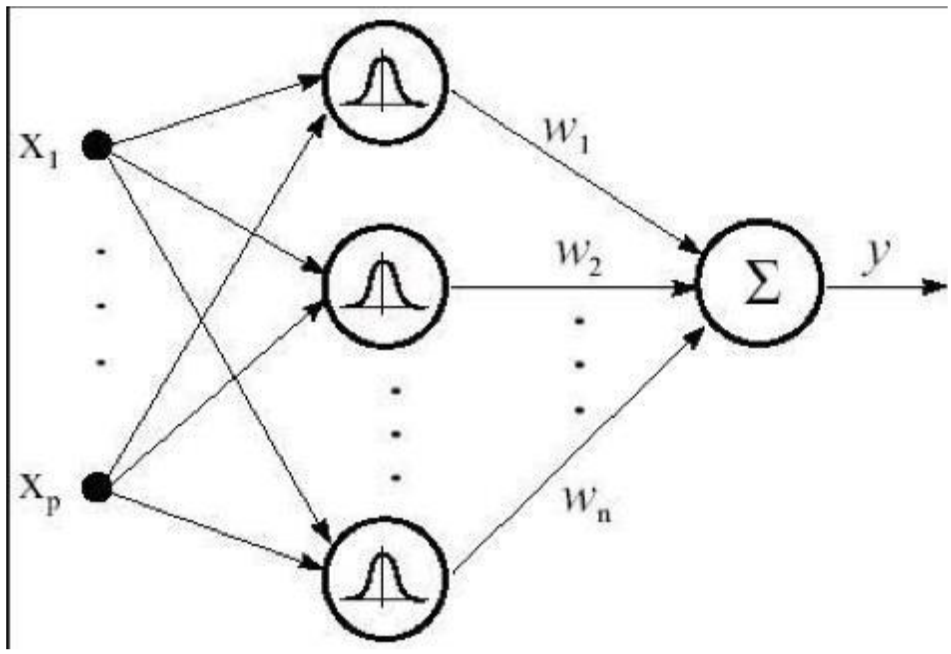

Figure 3.3: Radial basic function neural networks

3.2.2 Decision Trees

Decision Trees is a hierarchical machine learning method in which the data set is iteratively divided into a tree structure according to a specific criterion. In this section, firstly, the basic structure of decision trees will be explained. Decision tree splits occur in a way that creates regression or classification trees based on the data type of the target variable. Since the regression model is applied in this study, regression trees will be emphasized in the following titles. Then, by mentioning the model combination methods used to increase performance in decision trees, the gradient strengthening method used in the study and the way it is used in decision tree regressions will be explained. Decision trees consist of root knots, inner nodes and leaves. There is no branch to the root node; There are zero or more outgoing branches. The root node contains all the data in the data set. Each of the internal nodes has an incoming branch and two or more outgoing branches. Internal nodes and root node contain rules used to separate values in the data set by their properties. Each of the leaves has an incoming branch and no outgoing branch. Leaves contain predictions of the target variable. Decision trees are created by dividing each node consecutively based on explanatory variables. Each explanatory variable is sent to one of two sub-nodes, one on the left and one on the right. There are regression trees when the target variable is continuous and classification trees when it is categorical. A single cleavage point is determined for a continuous explanatory variable. Observations with the explanatory variable value smaller than this split point are divided to the left, and other observations to the right. For a categorical explanatory variable, cleavage splits one subgroup of categories to the left and the others to the right. Smaller subgroups are created with binary splits each time. The cleavage process is repeatedly applied to its output until it reaches a stop criterion. An example of a stop criterion is stopping the split when all nodes contain less than a specified number of observations. After reaching the stop criterion, the tree is pruned by breaking the weakest links detected to prevent overfitting. Predictive values are obtained from observations in the undivided nodes (leaves) remaining in the tree after pruning. Prediction values are obtained by taking the average of the target variable in regression problems and by calculating the ratios of class membership in classification problems [58, 59].

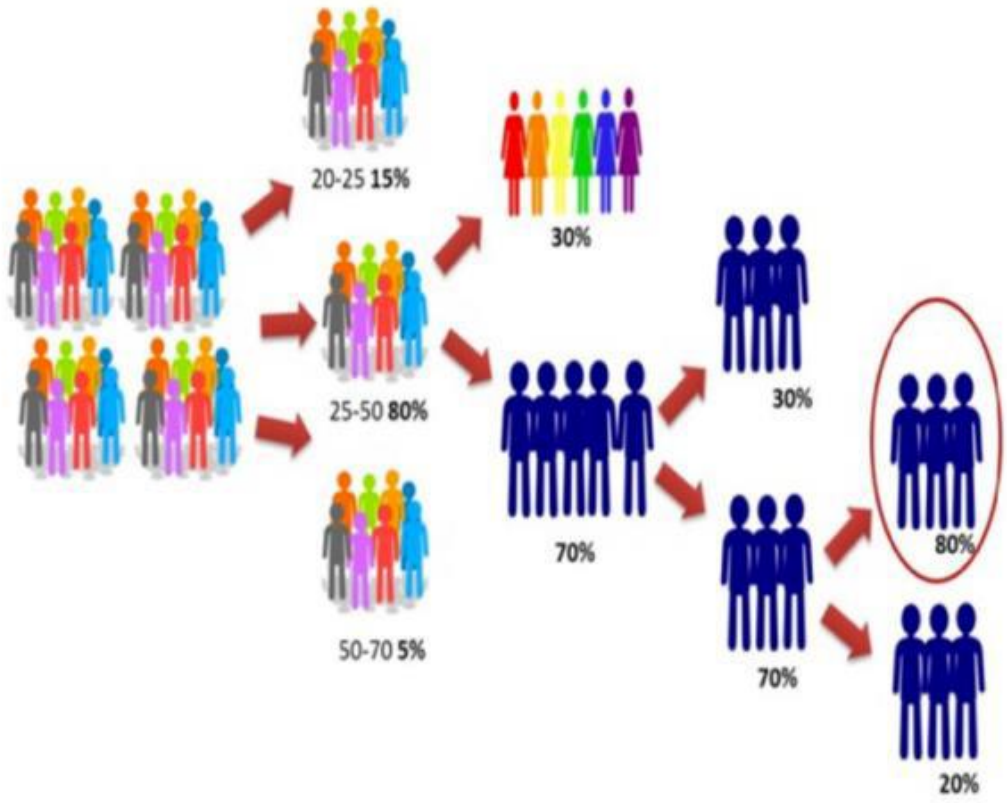

Figure 3.4: Decision Trees

3.2.3 Artificial neural networks (ANN)

Artificial neural networks (ANN) is one of the artificial intelligence techniques; These are computer systems that aim to realize the abilities of the human brain without assistance. Traditional methods are generally not sufficient when programming these systems. In fact, artificial neural networks; Inspired by the biological neural network in humans and animals. For example, in the biological nervous system, neurons alone cannot perform their storage function. Therefore, they transmit signals from one to another and reach the brain. In other words, the memory used by the computer for storage imitates the brain in the biological system. The aims are the same, but the mechanism used differs. Briefly; ANN is considered to be a computer-based science that deals with data processing, developed for events that are difficult to program [60]. Artificial neural networks (ANNs) are systems that imitate the functioning of the human brain. There are many processing units, namely neurons, working in harmony in an ANN. These neurons are connected to each other by connections that have certain weights, namely synapses.

As a result of these connections, they form a network that can learn by extracting the relationships between data . ANN consists of input layer, output layer and many hidden layers. A neuron is connected with all the neurons in the next layer (Figure 2.3). Neurons receive the data they receive, transform the data into a mathematical function, and send the result to the next layer of neurons. ANN needs a prior training to determine connection weights. This education can be supervised or unsupervised. In supervised training, correct results of all inputs are given to ANN after each training cycle. Connection weights are changed in a way to minimize the error rate between the results found by ANN and the results given. In unsupervised training, only inputs are given and ANN's categorizing the inputs by comparing the relationships between them.

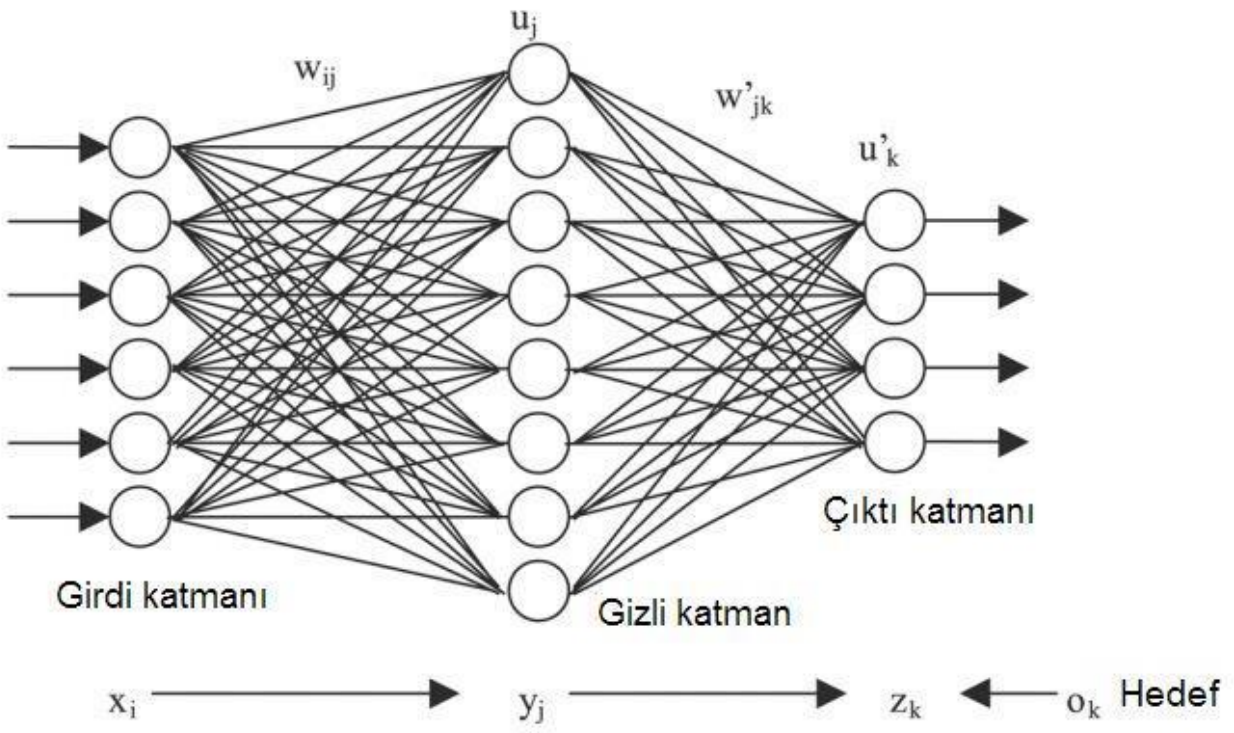

Figure 3.5: Neural Network

ANN is used in data-intensive problems where the solution is not known or difficult to express. Thanks to its structure and nonlinear calculation form, it can process the data in parallel and is less affected by the errors that may occur in the data. He can reach the conclusion by

generalizing in erroneous data sets. Since ANN is a closed system, solution steps cannot be observed. It is not useful for problems where the solution is important. In addition, if ANN does not solve a problem, it is almost impossible to find out why it cannot be solved.

3.3 DEEP LEARNING

Deep learning has been developed in relation to the mathematical redesign of biological neurons. An artificial neuron model of artificial neural networks, which is the basis of deep learning applications, is seen in Figure 1.2. x1..xn refers to the input values and w1..wn refers to the weight values of each input. Each input value is multiplied by its own weight value. The results obtained are collected in the transfer function and passed through the activation function. The activation function limits the result obtained by the transfer function within a certain range. For example, when the sigmoid activation function is used, the Y output value takes a value between 0-1. Deep learning is a field included in the class of machine learning techniques that computationally extracts attributes from data with its hierarchical layer architecture [61, 62].

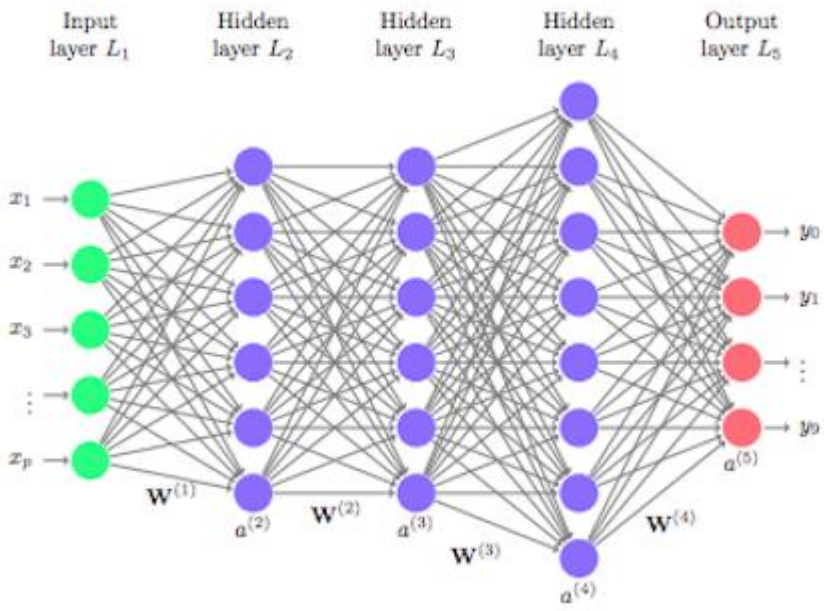

Figure 3.6: Deep learning

The concept of deep here refers to a large artificial neural network and the situation where the number of hidden layers is more than one in multilayer artificial neural networks. According to

Andrew Ng, the essence of deep learning is that we have enough speed of computers and data to train large neural networks. In Figure 3.7, the performances of the deep learning method and classical machine learning algorithms are compared over the amount of data. Here, it is seen that the performance of the deep learning method increases as the amount of data increases. Andrew Ng explains why deep learning methods can be preferred in applications that can be done with big data [63,64].

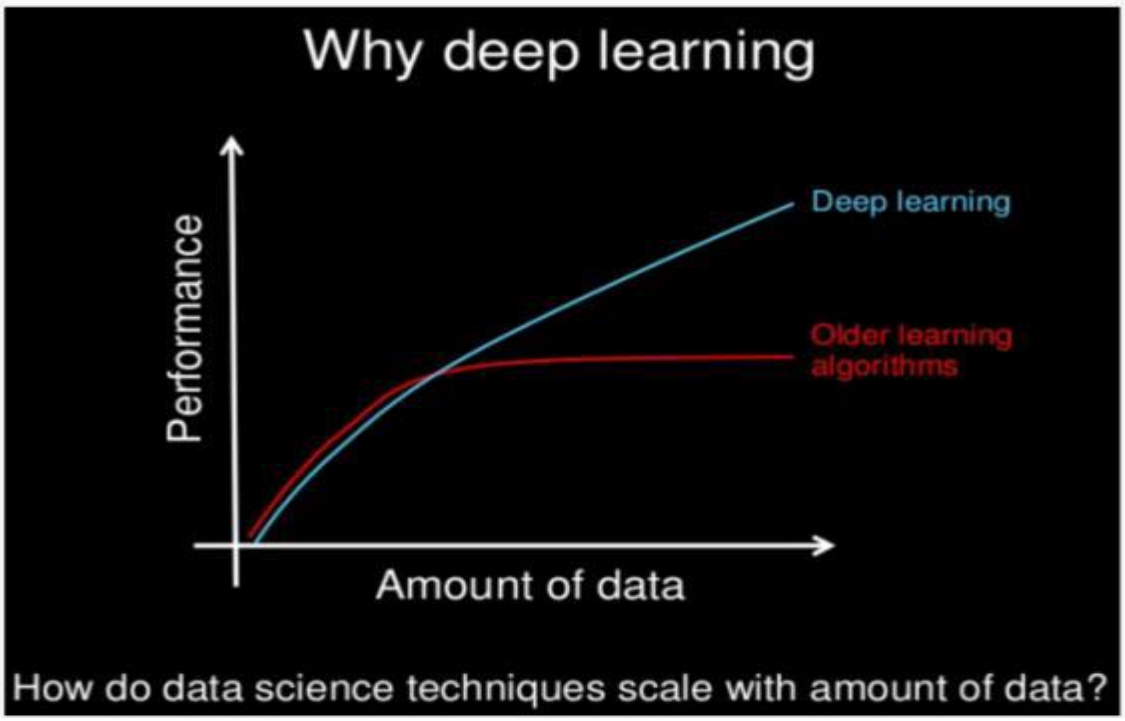

Figure 3.7: Deep learning scale

3.3.1 Convolutional neural networks

Convolutional neural network is a layered neural network model designed to extract feature from a dataset (image). Image attributes in each layer are automatically extracted and transmitted to the next layer. Convolutional neural network is a layered neural network model designed to extract feature from a dataset (image). Image attributes in each layer are automatically extracted and transmitted to the next layer. They have proved in their studies that the attributes of the object in recognition are successfully revealed by CNN. An example of convolutional neural network model architecture can be seen in Figure 3.7 the convolutional neural network architecture in Figure 3.38 represents a model created to classify numbers between 0-9. First of

all the convolution layer and then maximum pooling layer is used. After the pooling layer, the activation function was used in order to increase the non-linear states and continued with the pooling layer and the activation function together with the convolution. Finally, a fully bonded layer was obtained. After this layer, in order to apply the classification process, the result was reached on probability by using the score values obtained by the artificial neural network. If it is to be applied for two classes, the sigmoid function is preferred, and if there are more classes, the softmax function is preferred [65, 66].

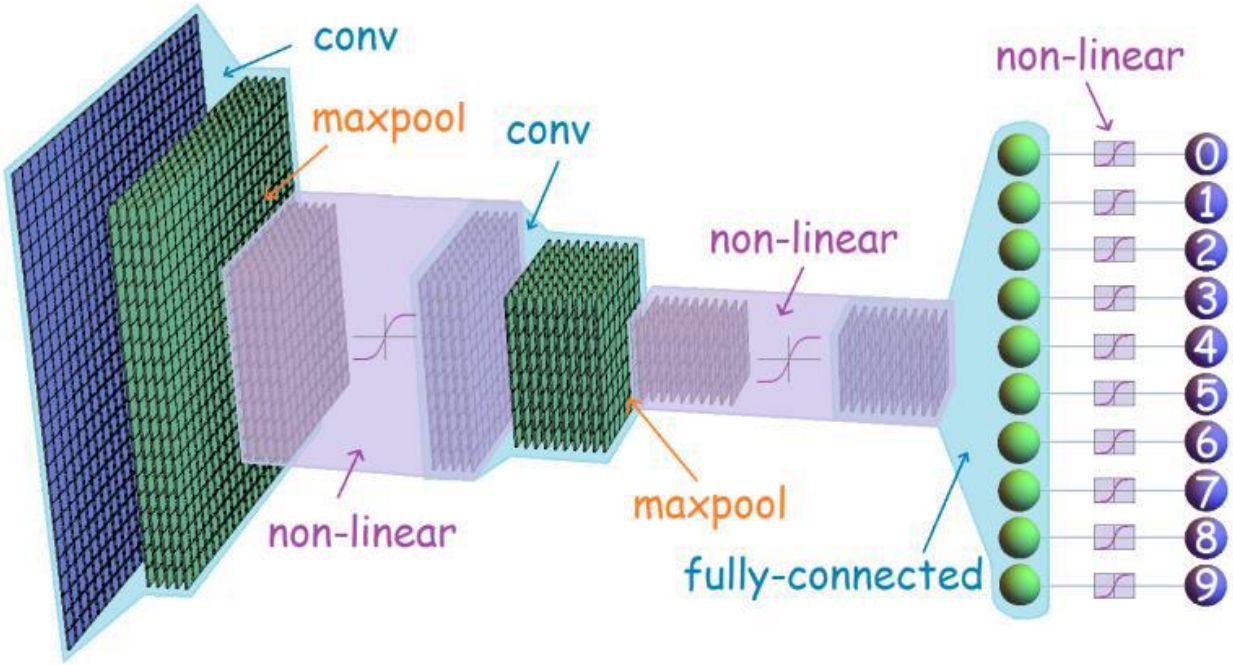

Figure 3.8: Convolutional neural network

Today, convolution networks are mostly used for image recognition. Convolutional neural networks are designed on 3 main themes: local receiver areas, shared weights, and pooling. In short, local receiver areas are regions that are separated on the input image (such as 3x3, 5x5, 7x7) that help us in extracting the attributes of the image. Figure 1.5 shows how the information of a 5x5 region is stored in the hidden neuron as an example local receiving area. Here, the attribute information of the input image is also stored in these areas.

Convolution Layer: In this layer, which we can call perhaps the most important layer of CNN, an output data and subsequently a feature map is created by circulating a filter of determined size over the whole image. Based on this, it is determined which relevant area of the image is

important. The applied filters can be in different sizes such as 5x5, 3x3. The image after the convolution layer is a state in which the input image becomes blurred. In other words, such a result occurs because the pixel values change. At the end of the process, feature map extraction helps to analyze the data better. Pooling Layer (Pooling): As in the convolution layer, there is a filtering here, and the data size is reduced. This reduces the size of the data to be worked on, and prevents the system from memorizing, ie producing the same results. Fully Connected Layer: This part is connected to all areas of the previous layer. Based on this layer, a weight matrix is created. Classification Layer: A prediction is made based on the data in the classification layer, which is the result part after all operations, steps, layers, and an output is generated about what the object.

3.3.1.1 ReLu

The result of the matrix multiplication operation on the image must be transformed into a nonlinear one, this process is performed with activation functions. ReLu function is used to provide nonlinear transformations. ReLu is the most commonly used activation function with CNN. It is often used after the convolution layer in deep learning models [67]. If it takes any negative input value, the function returns 0, for positive values the function returns that value itself. Thus, it can be mathematically expressed as f (x) = max (0, x). The graph of the function is shown in Figure 1.9.

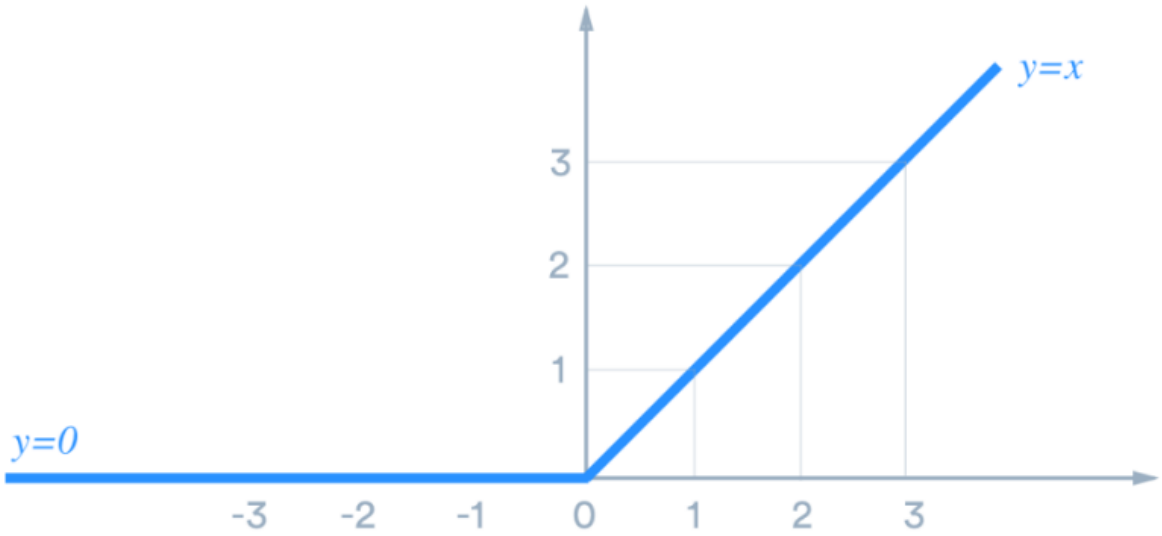

Figure 3.9: Relu transfer function

3.3.1.2 Vanishing Gradient

This gradient can be thought of as a concept similar to the delta of the back propagation algorithm. The disappearance gradient occurs when the exit error is likely to fail to access remote nodes in the learning process with the backpropagation algorithm. The back propagation algorithm trains the neural network by spreading the output error back to hidden layers. However, as errors rarely come back to the first layer, hidden layers close to the input layer cannot be fully trained [68].

3.3.1.3 Overfitting

The success of Machine Learning is highly dependent on how well the generalization process is implemented. We need a sufficient amount of unbiased training data to avoid performance degradation due to differences between training data and actual input data. When we take a glance at the data points, some outlier data goes into the other group's domain and breaks the boundary. Machine learning has no way of distinguishing this problem. Training data are not perfect and may contain varying amounts of noise. If you believe that every element of the training data is correct and fits the model perfectly, you will get a model with less generalizability. This is called extreme fitness. Extreme incompatibility significantly affects the performance level of Machine learning. The reason why the deep neural network is particularly vulnerable to extra overfitting (overfitting) is that the more complex the model, the more hidden layers and therefore more weight. A complex model is less susceptible to overfitting. Deepening layers for higher performance forces the neural network to face machine learning challenges. We divide training data into two groups. One group is used for training, the other group is used for validation. Generally, the ratio of training set to validation set is 8: 2. The model is trained using the training set. The performance of the model is evaluated using the validation set. There are also different verification processes. Cross validation is a small variation of the validation process. It divides training data into groups for training and validation, but continues to change data sets. Instead of initially preserving split sets, cross validation iterates the split of data. Cross

validation can better detect the overfitting of the model while preserving the randomness of the validation dataset [69].

3.3.1.4 Hyper-Parameters

Many of the machine learning algorithms contain a number of variables (hyper-parameters) that must be set before optimizing the parameters of the model. Adjusting the values of hyper parameters can be viewed as a model selection step. Usually the hyper-parameters are determined by the person who designed the model, and sometimes the model selection and optimization is made with the help of search algorithms or some hyper-learner. There are two hyper-parameter groups in artificial neural networks that determine the structure of the network and how the network will be trained. These are model hyper parameters and optimizer hyper parameters. Among the model hyper parameters, the most used activation functions and the dropout layer hyper parameters are. Mini-batch dimension, learning rate (learning rate) and epoch (number of training rounds) are commonly used among the optimizer hyper-parameters. Some ANN systems can use both supervised and unsupervised training at the same time.

3.4 SOFTWARE DEFECT PREDICATION FRAMEWORK BASED PCA, LSTM AND TSA

In this section new method proposed for software defect predication. The presented method consist from feature selection, classifier, and optimization algorithm. The PCA applied as selection which used to reduce the size of the input feature. This step assists to reduce the execution time and remove redundancy in the dataset. Then, the PCA output wired to the LSTM which is used as classifier in this study. The LSTM is time series classifier which is very common used in the last days. The contribution in this study is applied TSA to train LSTM which presented best results than traditional LSTM. The presented model shown in the Figure 3.10.

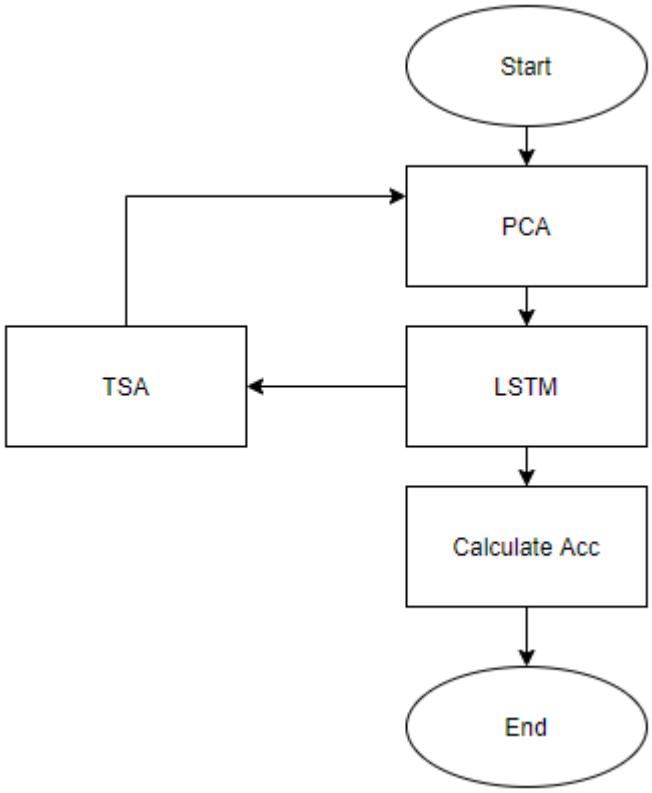

Figure 3.10: Predication framework

4. EXPERIMENTAL RESULTS

4.1 MATLAB

MATLAB is a powerful technical computer language. It integrates processing, visualization and programming in a user-friendly environment where problems and solutions are expressed in familiar mathematical notation.

MATLAB is an interactive matrix system that does not require scaling. This allows you to solve many technical computational problems, particularly those involving the formulation of matrices and vectors, in a fraction of the time it takes to write a program in a non-interactive scalar language such as C or Fortran.

MATLAB stands for Matrix Laboratory. MATLAB was originally developed to provide easy access to matrix software that was developed by the LINPACK and EISPACK projects and which together constitute state-of-the-art matrix computer software.

MATLAB has evolved over the years thanks to the contributions of many users. In an academic context, this is the standard study guide for introductory and advanced courses in math, engineering, and science. In industry, MATLAB is the ideal tool for high-performance research, development and analysis.

MATLAB offers a number of application-specific solutions called toolboxes. Toolkits, which are very important to most MATLAB users, allow you to learn and use certain technologies. Toolkits are comprehensive collections of MATLAB functions (M files) that extend the MATLAB environment to solve specific classes of problems. Areas where tools are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, modeling, etc.

4.2 SOFTWARE DEFECT PREDICATION RESULTS

The term "software development contract" refers to the agreement concerning the custommade software, that is the "tailor-made" computer program. A subject turns to a company (or to a self-employed worker) to design and implement software with functional characteristics such as to meet its particular needs (such as, for example, the computerization of the accounting of a company or the management of warehouse).

The development contract, therefore, is very different from that of the purchase of standard software, sold through large-scale distribution channels, because the user does not purchase a product already existing on the market, but gives the task of "drafting" a specific program suited to its needs and by means of which it intends to computerize its entire activity or a phase of it. The activity that leads to software development can be broken down into two phases. The first phase is the design phase (usually defined as a "feasibility study"), which, starting from the requirements set by the client, that is, from the objectives it intends to achieve and the means it intends to make available for this purpose, leads to a document in which indicate the functional specifications of the software to be created, the conditions that it must meet, the time and cost of implementation and, finally, the acceptance test plan.

The second phase, on the other hand, is the actual executive phase of the designed software. The two phases in question - design and construction - could be the subject of two separate contracts (similarly to what happens, for example, in the field of building constructions, where the design phase, entrusted to the designer, and the execution phase, entrusted to the instead to the contractor of the works).

In practice, however, it is not usual, except for transactions of high economic value, to stipulate two separate contracts. On the contrary, the client usually turns to an IT company, entrusting it with both the design and the implementation of the software under a single contract.

4.3 CONFUSSION MATRIX

In the context of artificial intelligence, the confusion matrix, also called the misclassification table, returns a representation of the accuracy of statistical classification. Each column of the matrix represents the predicted values, while each row represents the real values. The element on row i and column j is the number of cases in which the classifier has classified the "true" class i as class j. Through this matrix it is observable if there is "confusion" in the classification of different classes. Through the use of the confusion matrix it is possible to calculate the kappa coefficient, also known as Cohen's kappa coefficient. In the field of machine learning and statistical classification, a confusion matrix is a table where predictions are represented in columns and actual state is represented by rows (sometimes this is reversed, with real instances in columns and predictions in rows) . The table is an extension of the confusion matrix in

predictive analysis and makes it easy to see if a naming error has occurred and if the predictions are more or less correct. From this table, therefore, it is possible to understand the performance of a predictive classification model in order to determine how accurate and effective this model is. We can have two types of confusion matrix:

- 2-Class: attempt to predict the performance of a two-class predictive model (Yes / No, True / False, and so on);
- Multiclass: attempt to predict the performance of a predictive model consisting of more than two classes.

True positive (TP) if the expected class is YES and is equal to the actual class, this is a case of true positive. The model answered YES correctly. True negative (TN) If the expected class is NO and is equal to the actual class.

i. Sensitivity: given by the ratio between the true positives and the total of the predicted positive units. Also known as the accuracy rate, it gives information on how much data is correctly estimated positive out of the total of positive observations:
ii. Specificity: obtained by comparing the number of true negatives to the total of negative predicted units, it measures how many predicted units are really negative on the total of negative observed values:
iii. False alarm: it is the complementary index of specificity, also known as the false positive rate, which informs about how many incorrect predictions, out of the total negative observed values, have a positive value:
iv. Accuracy: ratio of correct data to total forecasts, measures the model's ability to predict correctly:
v. Total error: Complementary accuracy index, which gives information on the forecast error made by the model:
vi. The model examined, in the 30 days considered, correctly predicts 98% of cases against 98% of errors. When it predicts positively it has a probability of being correct equal to 97%, if instead it predicts negatively it will have a higher percentage of correctness equal to 99.12%, against an error of 38%, where it positively predicts negative observed data.

In light of these results, it can be said that the model, in this last month of forecast, predicts better the falls than the rises, given that the real yields in these 30 days undergo 13 days of decline and of these 8 are correctly predicted by the model. , against the 17 raises correctly predicted 8 times.

Finally, the model can be considered acceptable to make predictions since more than half of the time it predicts correctly, however it must be used as a tool to make decisions, it cannot be totally relied on its trends.

Finally, it has been shown, by evaluating the errors, that inserting the VIX in the model is significant, since it reduces the forecast error compared to a model that does not consider this component.

In light of the results of the analysis, the estimated model can be positively evaluated, considering it a good predictor for future returns: however, it must be used as a tool for making decisions, and not completely rely on the results. Then, the confusion matrix of the presented method shown in Figure 4.1.

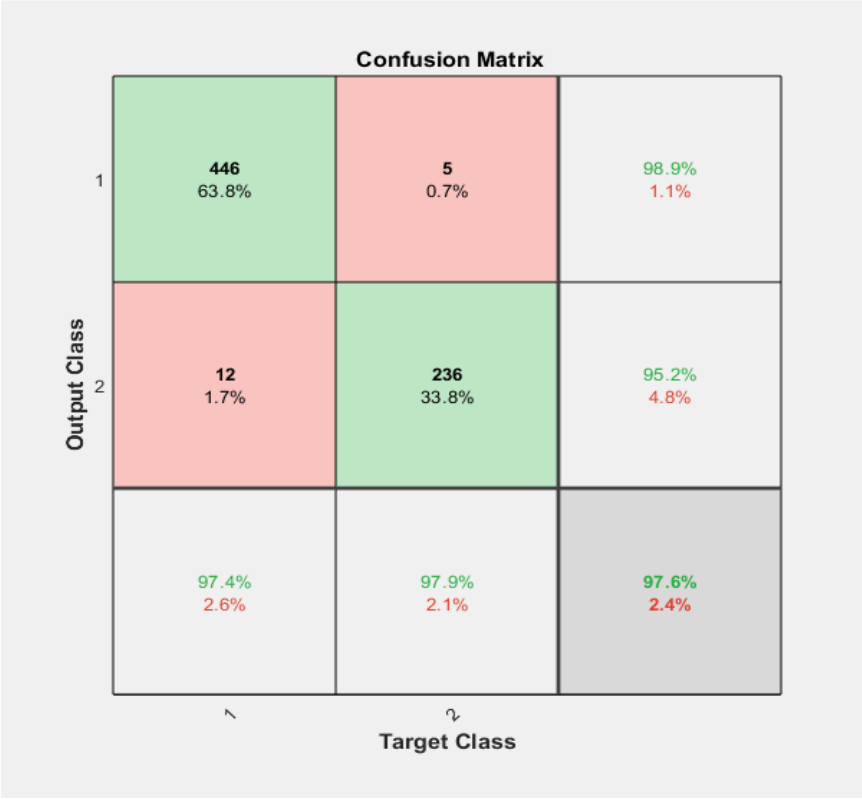

Figure 4.1: Confusion Matrix of Framework

4.4 ROC CURVE

In signal detection theory, the receiver operating characteristic (original name; Receiver Operating Characteristic - ROC) is defined as simply ROC curve. The ROC curve arises from the ratio of sensitivity to precision in cases where the discrimination threshold differs in binary classification systems. ROC can be simply expressed as the fraction of true positives versus false positives. As with any classification process, the methods strive to strike the balance between precision (the ability to eliminate false positives) and sensitivity (the ability to detect correct positives). Since the positive and negative samples in the dataset are not evenly distributed, before the direct precision and sensitivity criteria, the ROC abbreviation is the Receiver Operating Characteristics (Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching is used to evaluate the balance between precision and fastness. The area under the ROC curve can be defined as the ROC score. The ROC curve is formed by plotting the number of correct positives as a function of false positives according to changing classification threshold values. When the ROC score is 1 (one) it means that the positives are perfectly separated from the negatives. When the ROC score is 0 (zero), it means that no positive was found. Where significant cost disparities exist between false negatives and false positives, it may be essential to minimize one of the types of classification error. For example, in the context of spam detection it will probably be preferable to minimize false positives as a priority (even if this results in a significant increase in false negatives). AUC is not a criterion to be used for this type of optimization. The roc curve of this study presented in the figure 4.2.

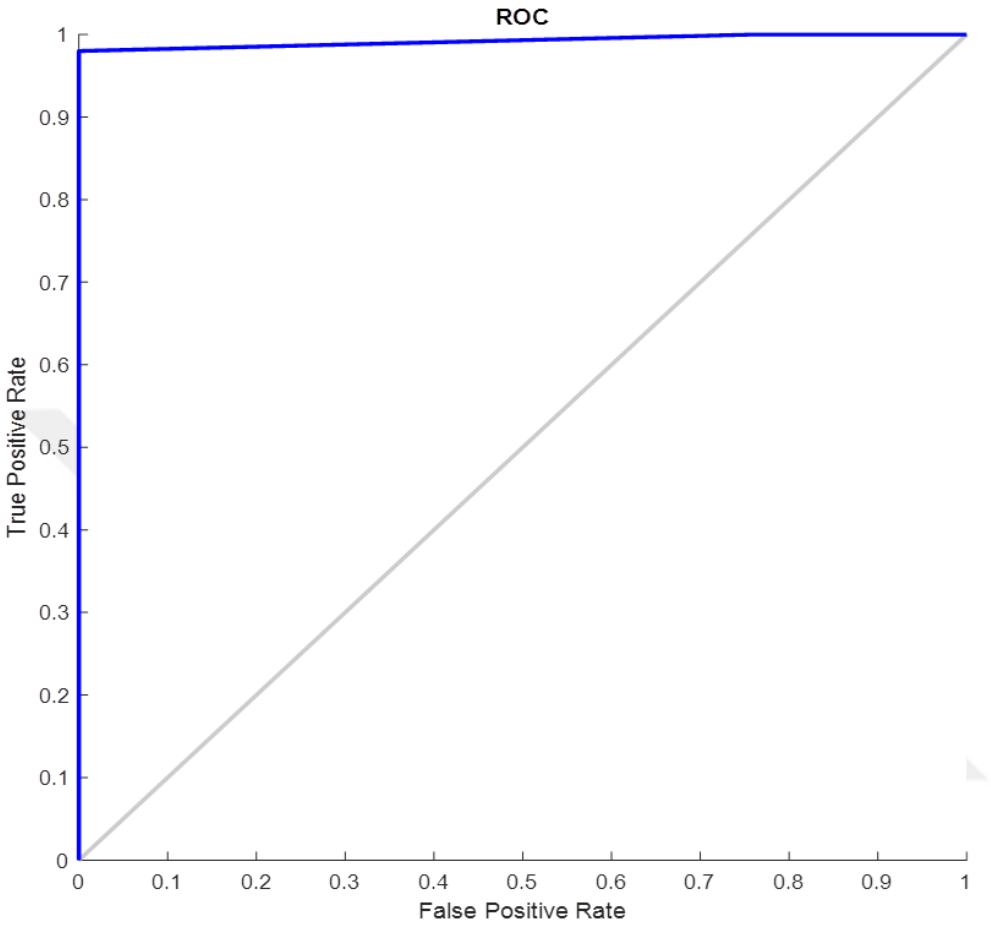

Figure 4.2: Roc Curve of framework

4.5 RESULTS

In this section, the results of the proposed method shown in the Table 4.1 which average of 10 experiments calculated for avoiding the overfitting.

Table 4.1: PCA based LSTM Results for CM1

Results	**Results**
Sensitivity	0.9804
Specificity	0.9709
Precision	0.9709
Negative Predictive Value	0.9804
False Positive Rate	0.0291
False Discovery Rate	0.0291
False Negative Rate	0.0196
Accuracy	**0.9856**
F1 Score	0.9756
Matthews Correlation Coefficient	0.9513

In statistics, principal component analysis (PCA) is the method of finding the projection of data in a multidimensional space to a lower dimensional space that maximizes the variance. For a set of points in space, it shows the "best fit" with the least average distance to all points. Later, those who say perpendicular to this line are selected again, the most appropriate line, and this type is repeated until the variance of a new dimension descends under a certain partner. The lines obtained at the end of this process form the bases of a linear space. These base vectors are called fundamental components. The baseline of data becomes independent. The PCA lead to reduce the size of input features this increase the performance of the system. The PCA increase the accuracy and reduce the execution time. Because its convert the data from nonlinear to linear which lead LSTM to presented remarkable results. Furthermore, our method compared with common studies presented in this field. See Table 4.2.

Table 4.2: Comparisons Table

Ref	Accuracy
[63]	94.21
[64]	93.98
[65]	92.01
Our Method	**98.56**

Our method presented best accuracy results then other previous studies that presented in [63] and [64]. The [63] presented Ordinal Classification method which is presented by the author for defect (bug) detection. Furthermore, [64] also presented several classifiers to predicate the software defect and presented 93.98%. Moreover, in [65] Li et al. presented CNN for software defect predication and presented 92.01 %. The compared with previous studies presented in the figure 4.1.

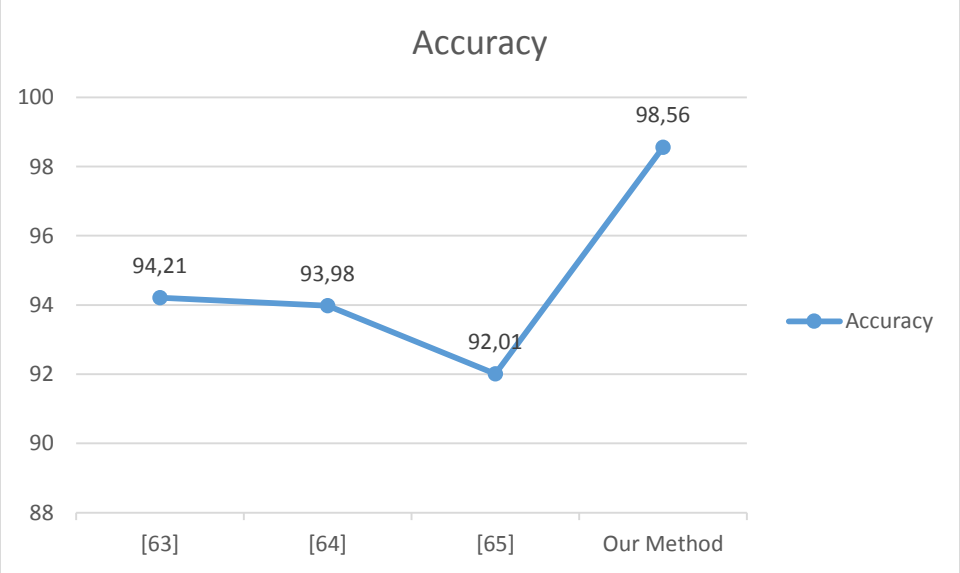

Figure 4.3: Comparison with previous studies

5. CONCLUSION

This thesis presented a new PCA-based deep learning method for predicting software errors. The PCA analyzes data from fault record to key performance data. The extracted features are then classified using the LSTM deep learning technique. The new method has shown remarkable results compared to previous studies.

In statistics, principal component analysis (PCA) is the method of finding the projection of data in a multidimensional space to a lower dimensional space that maximizes the variance. For a set of points in space, it shows the "best fit" with the least average distance to all points. Later, those who say perpendicular to this line are selected again, the most appropriate line, and this type is repeated until the variance of a new dimension descends under a certain partner. The lines obtained at the end of this process form the bases of a linear space. These base vectors are called fundamental components.

The presented method is a new idea that combines unsupervised learning with supervised learning. The aim of this study is to develop a new system for predicting software failures using data mining methods based on PCA.

The problem of predicting software failures is a complex problem that few research studies have solved. Our method, PCA + TSA + LSTM, performed better than any other method that this study could have applied to many other areas.

Looking to the future, the author advises researchers to investigate bug detection using machine learning, data mining, and deep learning techniques. Automatic detection of software errors reduces execution time and improves detection performance.

The author recommends using other unsupervised learning methods such as: B. the problem of predicting software failures (k-mean and clustering) and combining supervised methods with automated methods to improve system performance of Predicting software failures.

REFERENCES

[1] J. Mirkovic and P. Reiher, “A taxonomy of DDoS attack and DDoS defense mechanisms,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, p. 39, 2004.

[2] M. Boy, “DDOS Detection types.”Southeast Asian J. Trop. Med. Public Health, vol. 44, no. 4, pp. 568–573, 2016.

[3] N. Patani and R. Patel, “A Mechanism for Prevention of Flooding based DDoS Attack,” vol. 13, no. 1, pp. 101–111, 2017.

[4] I. Users, “No Title,” 2015. [Online]. Available: internetlivestats.com/internet-users.

[5] A. Meek, “DDoS attacks are getting much more powerful and the Pentagon is scrambling for solutions,” 2015.

Transactions on Software Engineering, vol. 25, no. 5, pp. 675-689, Sept.-Oct. 1999, doi: 10.1109/32.815326..

[6] Qinbao Song, M. Shepperd, M. Cartwright and C. Mair, "Software defect association mining and defect correction effort prediction," in IEEE Transactions on Software Engineering, vol. 32, no. 2, pp. 69-82, Feb. 2006, doi: 10.1109/TSE.2006.1599417.

[7] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden and S. 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), Gothenburg, Sweden, 2018, pp. 699-699, doi: 10.1145/3180155.3182520.

[8] Fei Wu et al., "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), Buenos Aires, Argentina, 2017, pp. 195-197, doi: 10.1109/ICSE-C.2017.72..

[9] S. Pradhan, V. Nanniyur and P. K. Vissapragada, "On the Defect Prediction for Large Scale Software Systems – From Defect Density to Machine Learning," 2020 IEEE 20th

International Conference on Software Quality, Reliability and Security (QRS), Macau, China, 2020, pp. 374-381, doi: 10.1109/QRS51102.2020.00056..

[10] L. Madeyski and M. Kawalerowicz, "Continuous Defect Prediction: The Idea and a Related Dataset," 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, Argentina, 2017, pp. 515-518, doi: 10.1109/MSR.2017.46.

[11] G. Mauša, T. G. Grbac and B. D. Bašić, "Software defect prediction with Bug-Code analyzer - A data collection tool demo," 2014 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 2014, pp. 425-426, doi: 10.1109/SOFTCOM.2014.7039122.

[12] C. L. Prabha and N. Shivakumar, "Software Defect Prediction Using Machine Learning Techniques," 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184), Tirunelveli, India, 2020, pp. 728-733, doi: 10.1109/ICOEI48184.2020.9142909.

[13] J. Sun, X. Jing and X. Dong, "Manifold Learning for Cross-project Software Defect Prediction," 2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), Nanjing, China, 2018, pp. 567-571, doi: 10.1109/CCIS.2018.8691373.

[14] S. Rathaur, N. Kamath and U. Ghanekar, "Software Defect Density Prediction based on Multiple Linear Regression," 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2020, pp. 434-439, doi: 10.1109/ICIRCA48905.2020.9183110..

[15] K. Okumoto, "Software defect prediction based on stability test data," 2011 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering, Xi'an, China, 2011, pp. 385-387, doi: 10.1109/ICQR2MSE.2011.5976636.

[16] Y. Xia, G. Yan and H. Zhang, "Analyzing the significance of process metrics for TT&C software defect prediction," 2014 IEEE 5th International Conference on Software Engineering and Service Science, Beijing, China, 2014, pp. 77-81, doi:

10.1109/ICSESS.2014.6933517.

[17] T. Sethi and 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), Noida, India, 2016, pp. 480-485, doi: 10.1109/ICRITO.2016.7785003.

[18] Y. Xia, G. Yan, X. Jiang and Y. Yang, "A new metrics selection method for software defect prediction," 2014 IEEE International Conference on Progress in Informatics and Computing, Shanghai, China, 2014, pp. 433-436, doi: 10.1109/PIC.2014.6972372.

[19] K. Sun, J. Zhang, C. Zhang, and J. Hu, "Generalized extreme learning machine autoencoder and a new deep neural network," *Neurocomputing*, vol. 230, no. November 2016, pp. 374–381, 2017.

[20] R. Malhotra and L. Bahl, "A defect tracking tool for open source software," 2017 2nd International Conference for Convergence in Technology (I2CT), Mumbai, India, 2017, pp. 901-905, doi: 10.1109/I2CT.2017.8226259.

[21] C. Nalini and T. Murali Krishna, "An Efficient Software Defect Prediction Model Using Neuro Evalution Algorithm based on Genetic Algorithm," 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2020, pp. 135-138, doi: 10.1109/ICIRCA48905.2020.9182869.

[22] K. Wongpheng and P. Visutsak, "Software Defect Prediction using Convolutional Neural Network," 2020 35th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), Nagoya, Japan, 2020, pp. 240-243..

[23] M. Assim, Q. Obeidat and M. Hammad, "Software Defects Prediction using Machine Learning Algorithms," 2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI), Sakheer, Bahrain, 2020, pp. 1-6, doi: 10.1109/ICDABI51230.2020.9325677.

[24] Q. Song, Z. Jia, M. Shepperd, S. Ying and J. Liu, "A General Software Defect-Proneness Prediction Framework," in IEEE Transactions on Software Engineering, vol. 37, no. 3, pp.

356-370, May-June 2011, doi: 10.1109/TSE.2010.90.

[25] D. Wahyudin, A. Schatten, D. Winkler, A. M. Tjoa and S. Biffl, "Defect Prediction using Combined Product and Project Metrics - A Case Study from the Open Source "Apache" MyFaces Project Family," 2008 34th Euromicro Conference Software Engineering and Advanced Applications, Parma, Italy, 2008, pp. 207-215, doi: 10.1109/SEAA.2008.36.

[26] R. Malhotra, N. Pritam and Y. Singh, "On the applicability of evolutionary computation for software defect prediction," 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 2014, pp. 2249-2257, doi: 10.1109/ICACCI.2014.6968592.

[27] P. Lakshmi and T. L. Maheswari, "An effective rank approach to software defect prediction using software metrics," 2016 10th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, India, 2016, pp. 1-5, doi: 10.1109/ISCO.2016.7727030.

[28] B. Li, B. Shen, J. Wang, Y. Chen, T. Zhang and J. Wang, "A Scenario-Based Approach to Predicting Software Defects Using Compressed C4.5 Model," 2014 IEEE 38th Annual Computer Software and Applications Conference, Vasteras, Sweden, 2014, pp. 406-415, doi: 10.1109/COMPSAC.2014.64.

[29] D. Wang, Z. Yufu, and J. Jie, "A multi-core based DDoS detection method," *Proc. - 2010 3rd IEEE Int. Conf. Comput. Sci. Inf. Technol. ICCSIT 2010*, vol. 4, pp. 115–118, 2010.

[30] Öztürk E , Birant K , Birant D . An Ordinal Classification Approach for Software Bug Prediction. Dokuz Eylül Üniversitesi Mühendislik Fakültesi Fen ve Mühendislik Dergisi. 2019; 21(62): 533-544.

[31] F. Zhang, A. E. Hassan, S. McIntosh and Y. Zou, "The Use of Summation to Aggregate Software Metrics Hinders the Performance of Defect Prediction Models," in IEEE Transactions on Software Engineering, vol. 43, no. 5, pp. 476-491, 1 May 2017, doi: 10.1109/TSE.2016.2599161.

[32] H. B. Yadav and D. K. Yadav, "Defects prediction of early phases of Software

Development Life Cycle using fuzzy logic," Confluence 2013: The Next Generation Information Technology Summit (4th International Conference), Noida, 2013, pp. 2-6, doi: 10.1049/cp.2013.2284.

[33] T. Lee, J. Nam, D. Han, S. Kim and H. Peter In, "Developer Micro Interaction Metrics for Software Defect Prediction," in IEEE Transactions on Software Engineering, vol. 42, no. 11, pp. 1015-1035, 1 Nov. 2016, doi: 10.1109/TSE.2016.2550458.

[34] Z. A. Rana, S. Shamail and M. M. Awais, "Ineffectiveness of Use of Software Science Metrics as Predictors of Defects in Object Oriented Software," 2009 WRI World Congress on Software Engineering, Xiamen, China, 2009, pp. 3-7, doi: 10.1109/WCSE.2009.92.

[35] T. Mende, R. Koschke and M. Leszak, "Evaluating Defect Prediction Models for a Large Evolving Software System," 2009 13th European Conference on Software Maintenance and Reengineering, Kaiserslautern, Germany, 2009, pp. 247-250, doi: 10.1109/CSMR.2009.55.

[36] Y. Qu et al., "node2defect: Using Network Embedding to Improve Software Defect Prediction," 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), Montpellier, France, 2018, pp. 844-849, doi: 10.1145/3238147.3240469.

[37] M. Kakkar, S. Jain, A. Bansal and P. S. Grover, "Evaluating Missing Values for Software Defect Prediction," 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Faridabad, India, 2019, pp. 30-34, doi: 10.1109/COMITCon.2019.8862444.

[38] Kaspersky, “Kaspersky DDoS Intelligence Report Q3 2015,” Kaspersky Lab, 2015.

[39] “DDOS ATTACKS,” 2011. [Online]. Available: https://www.incapsula.com/ddos/ddos-attacks.html.

[40] B. Hang, R. Hu, and W. Shi, “An enhanced SYN cookie defence method for TCP DDoS attack,” J. Networks, vol. 6, no. 8, pp. 1206–1213, 2011.

[41] S. Taghavi Zargar, J. Joshi, D. Tipper, and S. Member, “A Survey of Defense

Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," pp. 1–24, 2013.

[42] M. Bogdanoski, T. Shuminoski, and A. Risteski, "Analysis of the SYN Flood DoS Attack," Int. J. Comput. Netw. Inf. Secur., vol. 5, no. 8, pp. 15–11, 2013.

[43] G. Zhang and M. Parashar, "Cooperative Defense against DDoS Attacks," 2002.

[44] M. Sachdeva, G. Singh, K. Kumar, and K. Singh, "DDoS incidents and their impact: A review," Int. Arab J. Inf. Technol., vol. 7, no. 1, pp. 14–20, 2010.

[45] S. Sinha and M. Sharma, "Simulation and Analysis of DDoS Attacks by Specialized Simulator using Virtualization," vol. 3, no. 2, pp. 2–4, 2014.

[46] Cisco, "Internet Protocols."Comput. Stat. Data Anal., vol. 125, pp. 44–56, 2014.

[47] X. Rui, W. L. Ma, and W. L. Zheng, "Defending against UDP Flooding by negative selection algorithm based on eigenvalue sets," 5th Int. Conf. Inf. Assur. Secur. IAS 2009, vol. 2, pp. 342–345, 2009.

[48] M. Rouse, "UDP (User Datagram Protocol) definition." [Online]. Available: https://searchnetworking.techtarget.com/definition/UDP-User-Datagram-Protocol.

[49] cloudflare, "What is a DDoS Attack?" [Online]. Available: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/.

[50] E. Alpaydin, "Introduction to machine learning," Methods Mol. Biol., vol. 1107, pp. 105–28, 2004.

[51] X. Zhu, "Semi-Supervised Learning Tutorial (ICML 2007)," 2007.

[52] K. Sun, J. Zhang, C. Zhang, and J. Hu, "Generalized extreme learning machine autoencoder and a new deep neural network," Neurocomputing, vol. 230, no. November 2016, pp. 374–381, 2017.

[53] Z. Ghahramani, "LNAI 3176 - Unsupervised Learning," Mach. Learn., pp. 72–112, 2004.

[54] T. Hastie and R. Tibshirani, “Week 9: Unsupervised Learning ,” Stat. Learn. Course, no. 9, pp. 1–60, 2013.

[55] C. Sun, C. Hu, Y. Tang, and B. Liu, “More accurate and fast SYN flood detection,” Proc. - Int. Conf. Comput. Commun. Networks, ICCCN, 2009.

[56] H. Liu, Y. Sun, and M. S. Kim, “Fine-grained DDoS detection scheme based on bidirectional count sketch,” Proc. - Int. Conf. Comput. Commun. Networks, ICCCN, 2011.

[57] J. Tajer, M. Adda, and B. Aziz, “Comparison Between Divergence Measures for Anomaly Detection of Mobile Agents in IP Networks,” Int. J. Wirel. Mob. Networks, vol. 9, no. 3, pp. 01-13, 2017.

[58] C. Sun, J. Fan, and B. Liu, “A robust scheme to detect SYN flooding attacks,” Proc. Second Int. Conf. Commun. Netw. China, ChinaCom 2007, no. September 2007, pp. 397–401, 2008.

[59] D. Nashat, X. Jiang, and S. Horiguchi, “Router based detection for low-rate agents of DDoS attack,” 2008 Int. Conf. High Perform. Switch. Routing, HPSR 2008, no. March, pp. 177–182, 2008.

[60] J. Singh, M. Sachdeva, and K. Kumar, “Detection of DDoS Attacks Using Source IP Based Entropy,” vol. 3, no. 1, pp. 201–210, 2013.

[61] A. M. Brues, “Genetic effects of the atom bomb,” J. Hered., vol. 38, no. 5, pp. 137–137, 1947.

[62] D. Wang, Z. Yufu, and J. Jie, “A multi-core based DDoS detection method,” Proc. - 2010 3rd IEEE Int. Conf. Comput. Sci. Inf. Technol. ICCSIT 2010, vol. 4, pp. 115–118, 2010.

[63] Öztürk E , Birant K , Birant D . An Ordinal Classification Approach for Software Bug Prediction. Dokuz Eylül Üniversitesi Mühendislik Fakültesi Fen ve Mühendislik Dergisi. 2019; 21(62): 533-544.

[64] Software Defect Prediction: Do Different Classifiers Find the Same Defects?, Software

Quality Journal, Volume. 26, Issue. 2, p.525-552. DOI: 10.1007/s11219-016-9353-3.

[65] Li, J., He, P., Zhu, J., Lyu, M. R. 2017. Software Defect Prediction via Convolutional Neural Network. IEEE International Conference on Software Quality, Reliability and Security (QRS), 25-29 July, Praque, Czech Republic. DOI: 10.1109/QRS.2017.42.

[66] A. M. Karim, M. S. Güzel, M. R. Tolun, H. Kaya, and F. V Çelebi, "A new framework using deep auto-encoder and energy spectral density for medical waveform data classification and processing," Biocybern. Biomed. Eng., vol. 39, no. 1, pp. 148–159, 2019.

[67] A. M. Karim, Ö. Karal, and F. V Çelebi, "A New Automatic Epilepsy Serious Detection Method by Using Deep Learning Based on Discrete Wavelet Transform," no. 4, pp. 15–18, 2018.

[68] A. M. Karim, M. S. Güzel, M. R. Tolun, H. Kaya, and F. V Çelebi, "A New Generalized Deep Learning Framework Combining Sparse Auto-encoder and Taguchi Method for Novel Data Classification and Processing," pp. 1–22, 2018.

[69] A. M. Karim, F. V. Çelebi, and A. S. Mohammed, "Software Development for Blood Disease Expert System," Lecture Notes on Empirical Software Engineering, vol. 4, no. 3, pp. 179–183, 2016.