International Journal of Advanced Research and Multidisciplinary Trends (IJARMT)
An International Open Access, Peer-Reviewed Refereed Journal
Impact Factor: 6.4 Website: https://ijarmt.com ISSN No.: 3048-9458

Software Defect Prediction using Supervised Machine Learning: A Systematic Literature Review

Ashwini Sunil
Department of Computer Science and Engineering
Technocrats Institute of Technology, Bhopal, India
Ram Kumar Sahu
Department of Computer Science and Engineering
Technocrats Institute of Technology, Bhopal, India
Saurabh Karsoliya
Department of Computer Science and Engineering
Technocrats Institute of Technology, Bhopal, India

Abstract
Software testing is the process of finding faults in software while executing it. The results of the testing are used to find and correct faults. Software defect prediction estimates is where faults are likely to occur in source code. The results from the defect prediction can be used to optimize testing and ultimately improve software quality. Machine learning, that concerns computer programs learning from data, is used to build prediction models which then can be used to classify data. Many researchers have already been working in the field of defect prediction in software using some machine learning algorithms. Their results vary from dataset to dataset. These algorithms give inconsistent output for predicting defects in a random software project. Researchers have not decided which machine learning algorithm is best suitable for correctly predicting the defects in software so recent developments in machine learning introduce ensembling methods to predict defects. Ensembling takes the advantages of different techniques to give a better prediction of defects compared to individual base models. In this paper the study of different machine learning algorithm for software defect prediction.
Keywords: - Software Testing, Machine Learning, Ensemble Technique

1. INTRODUCTION

Through the creation of multiple categorization or classification models utilizing various machine learning approaches, software defect prediction (SDP) is a method for enhancing software quality and lowering the cost of software testing. Many organizations that foster different sorts of programming need to anticipate issues to keep up with programming quality for consumer loyalty and save testing costs. SDP is important for the product improvement life cycle in which we foresee the shortcoming utilizing an AI (ML) approach with verifiable information [1]. In order to meet customer expectations, it is a structured methodology that enables the development of low-cost, high-quality software in the shortest amount of time possible.

SDP's main goal is to give excellent programming and reliability while taking advantage of restricted assets. Software developers will be able to prioritize how computer resources are used at each stage of the software development process because of this [2, 3]. In order to maintain software quality, ensure customer satisfaction, and save money on testing, numerous software manufacturing companies wish to anticipate software defects. SDP is used to improve software quality, and a variety of classification models built with various machine learning techniques enable effective testing. To improve software quality and save money on software testing, a wide range of machine learning (ML) methods have been investigated. The SDP Decision tree, Naive Bayes (NB), Radial Basis Function, Support Vector Machine (SVM), K-nearest neighbor, Multi-layer Perceptron, and Random Forest (RF) are all examples of ML techniques [4].

New advances in ML are ensemble procedures and different ML strategies with highlight choice techniques like PCA, and so on [5, 6]. In the surviving writing, there are a few sorts of programming measurements that have been found and used for SDP. It would be more common-sense to manage the main programming measurements and spotlight on them to anticipate imperfection in programming [7]. SDP investigations information from the past gained from programming vaults to figure out the quality and dependability of programming modules [8]. In the existing literature, a variety of software metrics have been identified and utilized for SDP. Software metrics are used to generate SDP models from data gathered from established systems or similar software initiatives [9].

It would be more useful to check out and zero in on the main programming measurements to anticipate bugs in the product. Thusly, the dataset utilized in the paper has been openly accessible on the Commitment Storehouse beginning around 2005, giving data on different applications that NASA (Public Aviation and Space Organization) has researched. In the context of the research, the output categorization is carried out using K-means clustering following the process of feature selection (FS) and dataset pre-processing. Then, ML approaches, for example, SVM, NB and RF with and without molecule swarm improvement (PSO) are utilized. A gathering approach is then used to coordinate the outcomes. In the end, all ML models are looked at and compared to previous research. The models' exhibition is assessed utilizing accuracy, exactness, review, F-measure, execution blunder measurements, and disarray lattice.

2. SOFTWARE DEFECT PREDICTION

The Software is the system of physical gadgets, vehicles, home machines and others things inserted with hardware, software, sensors, actuators and availability which empowers these items to associate and trade information. Software is frequently separated into three classes:

- System software fills in as a base for application software. Framework software incorporates gadget drivers, working frameworks (OSs), compilers, plate formatters, word processors and utilities helping the PC to work all the more proficiently. It is additionally answerable for overseeing equipment parts and giving essential non-task-explicit capacities. The framework software is normally written in C programming language.
- Programming software is a lot of apparatuses to help engineers recorded as a hard copy programs. The different apparatuses accessible are compilers, linkers, debuggers, translators and word processors.
- Application software is planned to play out specific undertakings. Instances of utilization software incorporate office suites, gaming applications, database frameworks and instructive software. Application software can be a solitary program or an assortment of little projects. This sort of software is the thing that customers most regularly consider as "software."

Defect prediction is the study of predicting which software “modules” are defective. Here “modules” means some primitive unit of a running system such as a function or a class. A typical, object-oriented, software project can contain hundreds to thousands of classes [11, 12]. In order to guarantee general and project-related fitness attributes for those classes, it is commonplace to apply some quality assurance (QA) techniques to assess the classes inherent quality. These techniques include inspections, unit tests, static source code analyzers, etc. A record of the results of this QA is a defect log. We can use these logs to learn defect predictors, if the information contained in the data provides not only a precise account of the encountered faults (i.e., the "bugs"), but also a thorough description of static code features such as lines of code (LOC), complexity measures (e.g., McCabe's cyclomatic complexity), and other suitable object-oriented design metrics. There are three very good reasons to study defect predictors learned from static code attributes: they are easy to use, widely used, and useful to use.

Software deformity (or deficiency) prediction is viewed as one of the most practical and furthermore useful device which let us know whether a specific module is having imperfection or not. Software professionals consider it to be an essential stage for guaranteeing the nature of the procedure or the item which is to be created. It made light of an exceptionally pivotal job in achieving the cases the software industry that it can't meet the necessities in the spending plan and on schedule [10]. The colossal venture and cash spent on software designing improvement prompts an increment in software framework support costs. Today, the huge size of the software created is getting progressively mind boggling. Countless program codes, as well. To this end, the likelihood of software insufficiencies has been expanded and strategies for quality affirmation are not adequate to defeat all software lacks in colossal frameworks. One of the viable method to improve the nature of software is to anticipate software abandons, which is additionally a powerful method to alleviate the exertion of assessing or testing software code [13]. Under this situation, just piece of the software antiquities should be reviewed or tried and the remaining ones disregarded. Settling an imperfection, or flaw, prompts exponential increments in the event that it enters to the resulting stages of a software advancement lifecycle. The benefit of distinguishing software deficiencies in the underlying stages not just yields less blames and an upgraded software w.r.t quality, yet in addition helps in creating of a practical model. Likewise, we just spotlight on the more vulnerable modules during testing and upkeep stages which in the long run prompts powerful advancement of model. Subsequently, the constrained assets of an association could be sensibly apportioned with the target of identification and rectification of the most extreme number of software abandons. In this manner, this subject of prediction of the software shortcomings has been dissected widely and a ton of strategies have been recommended to address this issue [14].

3. LITERATURE REVIEW

A. Khalid et al. [1], software without flaws is always desired to maintain software quality for customer satisfaction and save money on testing. Therefore, we inspected different known ML procedures and improved ML methods on an unreservedly accessible informational collection. The motivation behind the examination was to work on the model execution as far as exactness and accuracy of the dataset contrasted with past exploration. The accuracy can be further improved, as previous investigations demonstrate. For this reason, we utilized K-implies bunching for the order of class marks. Further, we applied arrangement models to chosen highlights. ML models are optimized using Particle Swarm Optimization. We assessed the exhibition of models through accuracy, precision, review, f-measure, execution mistake measurements, and a disarray lattice. The outcomes show that all the ML and streamlined ML models accomplish the most extreme outcomes; in any case, the SVM and advanced SVM models outflanked with the most elevated accomplished exactness, close to 100% and 99.80%, separately. The precision of NB, Improved NB, RF, Upgraded RF and troupe approaches are 93.90%, 93.80%, 98.70%, 99.50%, 98.80% and 97.60, individually. Our objective was to achieve maximum accuracy in comparison to previous studies in this manner.

S. Stradowski et al. [2], the accompanying deliberate planning concentrate on means to dissect the present status of-the-craftsmanship as far as AI programming deformity forecast displaying and to distinguish and characterize the arising recent fads. Especially noteworthy is the fact that the analysis is conducted from a business perspective, assessing the possibilities of implementing the most recent methods and techniques in commercial settings to raise software quality and decrease the cost of the development life cycle. We made an expansive inquiry universe to respond to our exploration questions, playing out a computerized inquiry through the Scopus data set to distinguish pertinent essential examinations. Then, we assessed all found examinations utilizing a characterization plan to plan the degree of business reception of AI programming imperfection forecast in light of the watchwords utilized in the distributions. In addition, we validate reporting by utilizing the PRISMA 2020 guideline.

A. Faizin et al. [3], software systems have grown in size and complexity like never before. Software defect prevention is extremely difficult due to these characteristics. As a result, it is necessary to automatically predict the number of defects in software module components, which may assist developers in effectively allocating limited resources. To find and fix such flaws at a low cost, numerous strategies have been proposed. Be that as it may, the exhibition of these methodologies require critical improvement. Consequently, in this paper, we propose a clever methodology that use profound learning strategies to foresee the quantity of imperfections in programming frameworks. First, we perform data normalization and log transformation on a publicly accessible dataset. Second, we perform information displaying to set up the information input for the profound learning model. Third, we pass the displayed information to a uniquely planned profound brain network-based model to foresee the quantity of deformities. We additionally assess the proposed approach on two notable datasets. The assessment results delineate that the proposed approach is exact and can enhance the best in class draws near. The proposed approach significantly decreases the mean square error by more than 14% on average and boosts the squared correlation coefficient by more than 8%.

Emin Borandag et al. [4], close by the cutting edge programming improvement life cycle draws near, programming testing has acquired significance and has turned into an area explored effectively inside the computer programming discipline. In this review, AI and profound learning-related programming issue forecasts were made through an informational index named SFP XP-TDD, which was made utilizing three different created programming projects. This data set was used to train and test a Rotation Forest classifier ensemble version of five different classifiers that have been extensively cited in the literature. Various distributions in the writing talked about programming shortcoming forecasts through ML calculations addressing answers for various issues. The use of feature selection algorithms to improve classification performance was mentioned in some of these articles, while operating ensemble machine learning algorithms for software fault predictions was mentioned in others. In addition, a comprehensive literature review revealed that, as a result of the small sample sizes in the data sets and the low success rates in the tests carried out on these datasets, there were few studies involving software fault prediction with DL algorithms. Using three distinct software fault prediction datasets, this study statistically demonstrated that DL algorithms performed better than ML algorithms in data sets with large sample values. The exploratory results of a model that incorporates a layer of intermittent brain organizations (RNNs) were encased inside this review. Close by the previously mentioned and produced informational indexes, the concentrate additionally used the Overshadowing and Apache Dynamic MQ informational collections in to test the adequacy of the proposed profound learning technique.

Muhammad Azam et al. [5], one of the most dynamic areas of examination in the computer programming local area is imperfection forecast. The hole between information mining and computer programming should be crossed over to build the pace of programming achievement. Before the testing stage, programming deformity expectation predicts where these defects will happen in the source code. Numerous techniques, including clustering, statistical methods, neural networks, and machine learning models, are utilized to investigate the impact area in software. The objective of this examination is to analyze different AI calculations for anticipating programming deserts. There have been a lot of new fault prediction methods developed, but no single method or strategy can be applied to all kinds of datasets. Different machine learning algorithms, including Bayesian Net, Logistic Regression, Multilayer Perceptron, Ruler Zero-R, J48, Lazy IBK, Support Vector Machine, Neural Networks, Random Forest, and Decision stump, were used to find the largest subset of predicted defects in order to achieve maximum accuracy. This examination concern is to find abandons utilizing five NASA informational collections JM1, CM1, KC1, KC2, and PC1. When compared to other methods, logistic regression has been shown to produce the best results (93%).

I. Batool et al. [6], programming issue/imperfection expectation helps programming engineers to recognize defective builds, for example, modules or classes, from the get-go in the product advancement life cycle. Software fault prediction employs data mining, machine learning, and deep learning methods. We perform investigation of recently distributed surveys, reviews, and related examinations to distil a rundown of inquiries. These inquiries were either responded to in the past however required a new look or they were not considered by any means. We divide prior work into data mining, machine learning, and deep learning sections and compare their performance to demonstrate the significance of answers to newly added questions. We investigate which comparison criteria and datasets were most frequently used for software fault prediction. Following our quality assessment criteria, we select 68 primary studies from a large set that were initially selected and present answers to our research questions.

J. Pachouly et al. [7], conveying great programming items is a difficult errand. During the planning, execution, and testing phases, various teams must work together effectively. In a production environment, many software products have a high number of defects that are discovered. When used in critical applications, software failures can be life-threatening and cost a business money, time, and reputation. Recognizing and fixing programming surrenders in the creation framework is expensive, which could be a unimportant errand whenever distinguished prior to delivery the item. Double order is regularly utilized in existing programming imperfection expectation studies. In order to produce high-quality software products, it is possible to provide software development teams with meaningful information thanks to advancements in artificial intelligence techniques. A broad review for Programming Imperfection Expectation is fundamental for investigating datasets, information approval techniques, deformity identification, and expectation approaches and instruments. According to the survey, standard datasets used in early studies lack sufficient data validation techniques and features. The literature review revealed that the standard datasets lack labels, providing insufficient defects-related information. Orderly Writing Surveys (SLR) on Programming Deformity Forecast are restricted. Thus this SLR presents a far reaching examination of deformity datasets, dataset approval, location, expectation approaches, and instruments for Programming Imperfection Forecast. The survey demonstrates the futuristic recommendations that will make it possible for researchers to create a software defect prediction tool. The architecture for creating a software prediction dataset with sufficient features and statistical data validation methods for multi-label software defect classification is discussed in the survey. **L.-Q. Chen et al. [8],** SDP has emerged as an essential component of software testing and ensures the delivery of high-quality software products. Programming deformity expectation is partitioned into conventional programming imperfection forecast and without a moment to spare programming imperfection forecast (JIT-SDP). Nonetheless, the majority of the current programming deformity expectation systems are moderately rearranged, which makes it incredibly challenging to give engineers more point by point reference data. This paper proposes a software defect prediction framework based on heterogeneous feature selection and Nested-Stacking to improve software defect prediction and allocate testing resources efficiently. There are three stages to the framework: evaluation of model classification performance, Nested-Stacking classifier, and data set preprocessing and feature selection. The clever heterogeneous element choice and settled custom classifiers in the structure can really work on the precision of programming deformity expectation. This paper conducts probes two programming deformity informational collections (Kamei, Commitment), and shows the grouping execution of the model through two complete assessment markers, AUC, and F1-score. Within-project defect prediction (WPDP) and cross-project defect prediction (CPDP) were carried out on a large scale in the experiment. The outcomes show that the structure proposed in this paper has a superb order execution on the two sorts of programming deformity informational indexes, and has been enormously worked on contrasted and the gauge models.

M. Pavana et al. [9], programming shortcoming expectation (SFP) has an essential impact in nature of programming. It assists in recognizing the broken builds at an underlying periods of programming advancement existence with cycling (SDLC). When the predicted results do not match the actual output, this software defect is also known as a defect. This can be alluding as a mistake, issue, bug in a PC program. AI (ML) handles with the subject of how to assemble or plan PC programs that generally increment their adequacy at specific assignments through encounters. In field of computer programming, AI assumes a vital part and contains various methodologies like test exertion forecast and cost expectation. In these expectation method, programming deficiencies forecast (SFP) is the most noticeable concentrate in field of computer programming. In this paper, the forecasts were made utilizing AI calculations like Credulous Bayes (NB), support vector machine (SVM), strategic relapse (LR), irregular woods (RF). Highlight determination technique, for example, Spearman's position relationship is utilized for choosing profoundly connected input factors. The dimensions are reduced using dimensionality reduction techniques like LDA and PCA. Execution and assessment of these calculations are done utilizing exactness, accuracy, and review. With two feature selection and dimensionality reduction techniques, four machine learning methods are compared. When compared to other algorithms, the findings demonstrate that SVM provides 94% of the highest accuracy by employing Spearman's rank correlation. Using the PCA method, RF achieves the highest accuracy among the four classifiers with 92.8%, while NB and LR achieve the highest accuracy of 92% when applied to the dataset using the LDA method.

A. Al-Nusirat et al. [10], programming testing is the principal step of identifying the flaws in Programming through executing it. Hence, it is significant to foresee the flaws that might occur while executing the product to keep up with the presence of the product. There are various procedures of computerized reasoning that are used to anticipate future deformities. The AI is one of the main strategy that used to construct anticipating models. In this paper, directed an efficient survey of the managed AI procedures which are utilized for programming deformity expectation and assessed the exhibition. As a result, a number of the data are used to predict software faults during the evaluation by utilizing five cutting-edge supervised machine learning (classifiers). Notwithstanding, contrasted the presentation of these classifiers and different boundaries. After that, a number of experiments are carried out to alter the default classifier parameters in order to boost the accuracy of defect predictions. The findings demonstrated that supervised machine learning algorithms are capable of classifying classes as bugs or not. Consequently, supervised machine learning models are superior to conventional statistical models for predicting software bugs. Moreover, utilizing PCA never perceptible effect on forecast frameworks execution while adjusting the default boundaries decidedly influence classifier values, particularly with Counterfeit Brain Organization (ANN).The principal finding of this paper is acquired through the use of Outfit Learning strategies, though Sacking accomplishes 95.1% exactness with Mozilla dataset and Casting a ballot accomplishes 93.79% precision with kc1 dataset.

4. MACHINE LEARNING

Machine Learning is a subset of Artificial Intelligence concerned with “teaching” computers how to act without being explicitly programmed for every possible scenario. The central concept in Machine Learning is developing algorithms that can self-learn by training on a massive number of inputs. Machine learning algorithms are used in various applications, such as email filtering and computer vision, where it is difficult or infeasible to develop conventional algorithms to perform the needed tasks [4]. Machine learning enables the analysis of vast amounts of information. While it usually delivers faster, more precise results to identify profitable prospects or dangerous risks, it may also require additional time and assets to train it appropriately. Merging machine learning with AI and perceptive technologies can make it even more effective in processing vast volumes of information. Machine learning is closely associated with computational statistics, which focuses on making predictions using computers. Machine learning approaches are conventionally divided into three broad categories, namely Supervised Learning, Unsupervised Learning & Semi-supervised Learning, depending on the nature of the "signal" or "feedback" available to the learning system.

Face anti-spoofing (FAS) has lately attracted increasing attention due to its vital role in securing face recognition systems from presentation attacks (PAs). As more and more realistic PAs with novel types spring up, traditional FAS methods based on handcrafted features become unreliable due to their limited representation capacity. With the emergence of large-scale academic datasets in the recent decade, machine learning based FAS achieve remarkable performance and dominate this area.

Supervised Learning

A model is trained through a process of learning in which predictions must be made and corrected if those predictions are wrong. The training process continues until a desired degree of accuracy is reached on the training data. Input data is called training data and has a known spam / not-spam label or result at one time.

Unsupervised Learning

By deducting the structures present in the input data, a model is prepared. This may be for general rules to be extracted. It may be through a mathematical process that redundancy can be systematically reduced, or similar data can be organized. There is no labeling of input data, and there is no known result.

Semi-Supervised Learning

Semi-supervised learning fell between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). There is a desired problem of prediction, but the model needs to learn the structures and make predictions to organize the data. Input data is a combination of instances that are marked and unlabeled.

4.1 Technique

Nearest Neighbors Algorithm

The Nearest Neighbor (NN) rule differentiates the classification of unknown data point because of closest neighbor whose class is known. The nearest neighbor is calculated based on estimation of k that represents how many nearest neighbors are taken to characterize the data point class. It utilizes more than one closest neighbor to find out the class where the given data point belong termed as KNN. The data samples are required in memory at run time called as memory-based technique. The training points are allocated weights based on their distances from the sample data point. However, the computational complexity and memory requirements remained key issue. For addressing the memory utilization problem, size of data gets minimized. The repeated patterns without additional data are removed from the training data set [15].

Naive Bayes Classifier

Naive Bayes Classifier technique is functioned based on Bayesian theorem. The designed technique is used when dimensionality of input is high. Bayesian Classifier is used for computing the possible output depending on the input. It is feasible to add new raw data at runtime. A Naive Bayes classifier represents presence (or absence) of a feature (attribute) of class that is unrelated to presence (or absence) of any other feature when class variable is known. Naive Bayesian Algorithm is used to predict the heart disease. Raw hospital dataset is employed. After that, the data gets preprocessed and transformed. Finally by using the designed data mining algorithm, heart disease was predicted and accuracy was computed.

Support Vector Machine

SVM are used in many applications like medical, military for classification purpose. SVM are employed for classification, regression or ranking function. SVM depends on statistical learning theory and structural risk minimization principal. SVM determines the location of decision boundaries called hyper plane for optimal separation of classes as described in figure 3. Margin maximization through creating largest distance between separating hyper plane and instances on either side are employed to minimize upper bound on expected generalization error. Classification accuracy of SVM not depends on dimension of classified entities. The data analysis in SVM is based on convex quadratic programming. It is expensive as quadratic programming methods need large matrix operations and time consuming numerical computations.

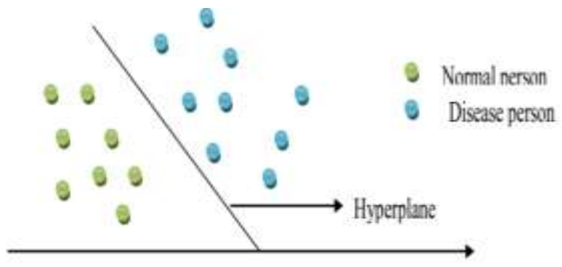

Fig. 1: Support Vector Classification

Neural Network

An artificial neural network is a computational model inspired in the natural neurons of a biological nervous system. These models mimic the real life behavior of neurons and the electrical messages they produce between input processing by the brain and the final output from the brain. In other words, artificial neural networks form an attempt to create machines that work in a similar way to the human brain by building these machines using components that behave like biological neurons. The function of an artificial neural network is to produce an output pattern when presented with an input pattern.

Traditionally the term neural network was used to refer to a network or circuit of biological neurons. But the modern usage of the term often refers to artificial neural networks, which are composed of artificial neurons or nodes. Biological neural networks are made up of real human neurons that are connected or functionally related in a nervous system (see Fig. 2). In a typical ANN, input units store the inputs, hidden units transform the inputs into an internal numeric vector and an output unit transforms the hidden values into the prediction. From a practical point of view, an ANN is just a parallel computational system consisting of many simple processing elements like units, connections and weights with numeric inputs and out- puts connected together in a specific way in order to perform a particular task. A simple artificial neural network model with multi-input and single-output is presented in Fig. 3. Basic components of artificial neurons are described as follows:

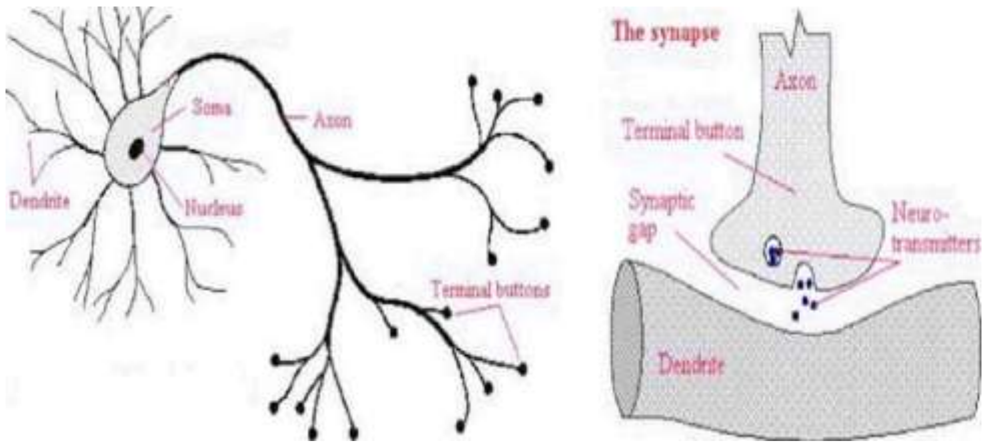

Fig. 2: Schematic diagram of biological neuron

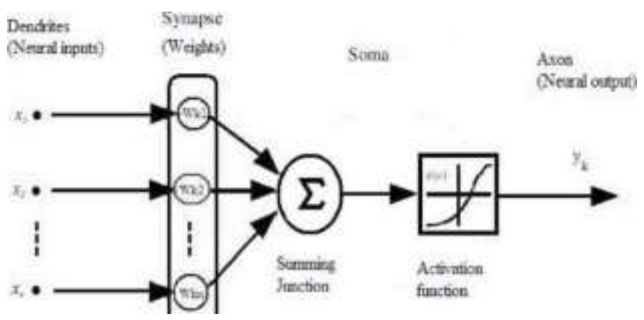

Fig. 3: A simple model of artificial neural network

A set of connections (that is, synapses) brings in activations from other input neurons (dendrites) and provides long-term memory to the past accumulated experience.

A processing unit (soma) sums the inputs and then applies a non-linear activation function (that is, transfer/threshold function); almost all the logical functions of the neuron are carried out in the soma.

An output line (axon) transmits the result to other neurons.

ANNs processes the information using connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. The typical elements of an artificial neural network are generally described as the network architecture, learning algorithm and activation function.

5. CONCLUSION

The goal of the thesis was to investigate whether a test, coupled to a source code file, contained enough information to enhance the SDP performance if metrics from both the source file and test file are combined. To investigate this literature review followed by an experiment was conducted. The literature review helped us identify of what kind of metrics and ML algorithms to use in the experiment, while the experiment ultimately helped us test the hypotheses.

The main motive of this study was to examine the various classification techniques and ensemble techniques in order to find which method gives better fault prediction performance. Thus we employed four classification techniques (naive bayes, KNN, neural network and support vector machine). Out of these 4, we selected the best 3 techniques on the basis of AUC values.

REFERENCES

[1] A. Khalid, G. Badshah, N. Ayub, M. Shiraz and M. Ghouse, "Software defect prediction analysis using machine learning techniques", *Sustainability*, vol. 15, no. 6, 2023.

[2] S. Stradowski and L. Madeyski, "Machine learning in software defect prediction: A business-driven systematic mapping study", *Information and Software Technology*, vol. 155, pp. 107128, 2023.

[3] A. Faizin, A. Iqbal, E. Suharto, A.P. Widodo, H. Faizal, M. N. Pratama, et al., "Software defect prediction using deep learning", *AIP Conference Proceedings*, vol. 2738, no. 1, 2023.

[4] Emin Borandag, "Software Fault Prediction Using an RNN-Based Deep Learning Approach and Ensemble Machine Learning Techniques", *Applied Sciences*, vol. 13, no. 3, pp. 1639, 2023.

[5] Muhammad Azam, Muhammad Nouman, Ahsan Rehman Gill, “Comparative Analysis of Machine Learning techniques to Improve Software Defect Prediction”, Journal of Computing & Information Sciences (KJCIS) Volume 5, Issue 2, pp. 41-66, 2022.

[6] I. Batool and T. A. Khan, "Software fault prediction using data mining machine learning and deep learning techniques: A systematic literature review", *Computers and Electrical Engineering*, vol. 100, pp. 107886, 2022.

[7] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran and A. Abraham, "A systematic literature review on software defect prediction using artificial intelligence: Datasets data validation methods approaches and tools", *Engineering Applications of Artificial Intelligence*, vol. 111, 2022.

[8] L.-Q. Chen, C. Wang, and S.-L. Song, ‘‘Software defect prediction based on nested-stacking and heterogeneous feature selection,’’ *Complex Intell. Syst.*, vol. 8, no. 4, pp. 3333–3348, Aug. 2022

[9] M. Pavana, L. Pushpa, and A. Parkavi, ‘‘Software fault prediction using machine learning algorithms,’’ in *Proc. Int. Conf. Adv. Elect. Comput. Technol.*, 2022, pp. 185–197

[10] A. Al-Nusirat, F. Hanandeh, M. K. Kharabsheh, M. Al-Ayyoub, and N. Al-Dhfairi, ‘‘Dynamic detection of software defects using supervised learning techniques,’’ *Int. J. Commun. Netw. Inf. Secur.*, vol. 11, no. 1, pp. 185–191, Apr. 2022.

[11] M. P. Basgalupp, R. C. Barros, A. G. C. De Sa, G. L. Pappa, R. G. Mantovani, A. C. P. L F. De Carvalho, et al., "An extensive experimental evaluation of automated machine learning methods for recommending classification algorithms", *Evolutionary Intelligence*, vol. 14, no. 4, pp. 1895-1914, 2021.

[12] Akimova EN, Bersenev AY, Deikov AA, Kobylkin KS, Konygin AV, Mezentsev IP, et al., "A Survey on Software Defect Prediction Using Deep Learning", *Mathematics.*, vol. 9, no. 11, pp. 1180, 2021.

[13] R. Bahaweres, F. Agustian, I. Hermadi, A. Suroso, and Y. Arkeman, ''Software defect prediction using neural network basedSMOTE,'' in *Proc. 7th Int. Conf. Electr. Eng., Comput. Sci. Informat. (EECSI)*, Oct. 2020, pp. 71–76

[14] N. Li, M. Shepperd, and Y. Guo, ''A systematic review of unsupervised learning techniques for software defect prediction,'' *Inf. Softw. Technol.*, vol. 122, Jun. 2020, Art. no. 106287.

[15] K. Tanaka, A. Monden and Z. Yucel, "Prediction of software defects using automated machine learning", *20th IEEE/ ACIS International Conference on Software Engineering*, pp. 490-494, 2019.