International Journal of Advanced Computational Engineering and Networking, ISSN(p): 2320-2106, ISSN(e): 2321-2063
Volume-12, Issue-1, Jan.-2024, http://iraj.in

SOFTWARE TESTING AUTOMATION USING MACHINE LEARNING TECHNIQUES

[1]MUSTAFA ABDUL SALAM, [2]MOHAMED ABDUL-FATTAH, [3]ABDULLAH MOHAMED

[1,2,3]Faculty of Computers and Artificial Intelligence, Benha University, Egypt
[1]Faculty of Computer Studies, Arab Open University, Cairo, Egypt
E-mail: abdalla.mohamed@fci.bu.edu.eg

Abstract - Finding, locating, and resolving software defects takes a lot of time and effort. This paper proposes a hybrid machine learning model to automate the software testing process. The proposed model combines particle swarm optimization (PSO) to optimize artificial neural network (ANN) to overcome the local minima and overfitting problems. The proposed model is compared with different classification algorithms such as: Logistic Regression, K nearest neighbours (KNN), Decision Tree, Random Forest, Gradient Boosting, AdaBoost, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Gaussian NB, Support Vector Machine and deep learning neural networks. The effectiveness of the proposed model is evaluated using four different datasets (CM1, KC1, KC2, and PC1). Datasets have been divided into training part (70%) and testing part (30%). The proposed model achieved higher accuracy than compared algorithms, while also reducing time and space complexities.

Keywords - Software testing automation, classification algorithms, deep learning, particle swarm optimization

I. INTRODUCTION

Testing conducted on a software product is called software testing. The main objective of testing or software testing is to find bugs in the software. Bug is an error or fault caused in the behaviour of the software program or software application. Software testing can be done to check if the software Test automation can be of 2 types:

1. Testing with code uses pre-existing interfaces, libraries, classes, and modules to test with a large number of inputs and check and verify whether the results are right.
2. Testing with a Graphical User Interface (GUI): Interface events such as keystrokes and mouse clicks can be generated and used by the framework to identify changes and test whether the program's performance is right or not [1]

Test cases are a set of conditions used by testers to determine whether or not the system under test operates properly. The creation of test cases aids in the discovery of application flaws or needs [2].

Any software application testing that is automated will go through a series of activities, processes, and tools to complete the test. The outcomes of these runs can be saved and recorded [3].

Software Testing can be majorly classified into two categories:

1. Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester.
2. White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester [4].

The steps of automated software testing as shown in Fig. 1:

1. Software test automation framework configured to collect automated test suite and test execution results.
2. A report parser to parse the test execution results generated by the software test automation framework and configured to identify the failures or exceptions with their respective stack trace.
3. A NoSQL database configured to hold historical defect, bug tickets with past failures or exceptions.
4. A ML engine to evaluate matching results of the NoSQL database and configured to predict type of the failure or exception.
5. A defect - tracking tool configured to create relevant bug tickets based on the type of failure or exception.
6. An automated notification system configured to notify the status of a bug ticket.
7. A dashboard to facilitate the access results, logs, failures, key performance indicators etc. in the form of histograms, pie graphs etc.
8. A manual feedback mechanism for adjusting the machine learning algorithm and NoSQL database table entries.[11]

The process of automating software testing has already seen a number of fascinating approaches. The software development life cycle [6] includes many stages, and machine learning (ML), a subfield of artificial intelligence (AI), is frequently utilised in these stages, particularly for automating software testing processes [7].

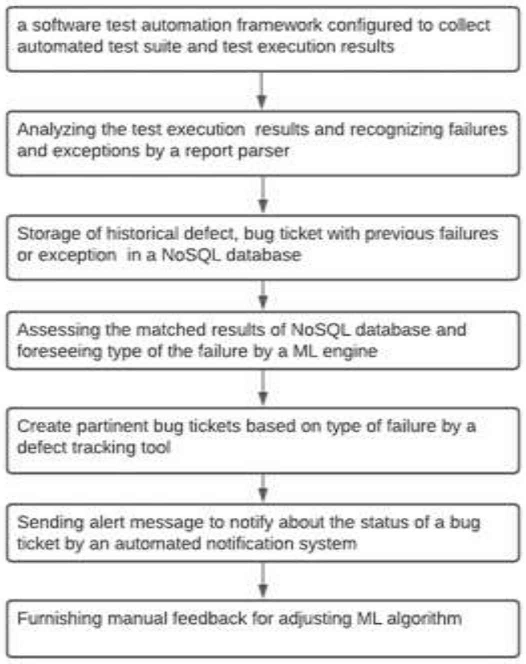

a software test automation framework configured to collect automated test suite and test execution results
Analyzing the test execution results and recognizing failures and exceptions by a report parser
Storage of historical defect, bug ticket with previous failures or exception in a NoSQL database
Assessing the matched results of NoSQL database and foreseeing type of the failure by a ML engine
Create partinent bug tickets based on type of failure by a defect tracking tool
Sending alert message to notify about the status of a bug ticket by an automated notification system
Furnishing manual feedback for adjusting ML algorithm

Figure. 1 Illustrates a method for automated software testing based on ML, in accordance to one or more embodiment of the present invention [5]

There are one or more AI algorithms that put the learning strategy into practise for each ML technique. As we previously said, classification issues (for target variables with discrete values) and regression problems (for target variables with continuous values) are solved using supervised learning [8]. Naive Bayes, decision trees, support vector machines, artificial neural networks (ANN), recurrent neural networks (RNN), and convolutional neural networks are a few examples of AI algorithms that can be employed with this strategy (CNN). RNNs are frequently used with sequential data [9], while CNNs are frequently used as a solution for large-scale inputs [10], such as pictures or sequential data. According to Paramshetti and Phalke [12], the ANN technique when using NASA datasets has numerous advantages over the present methods used to predict software problems. SVM can work with both linear and non-linear dataset values, unlike the Association rule method, which only works with linear dataset values. SVM with a modified kernel function offers improved prediction outcomes. While SVM is unaffected by outliers, Thamilselvana and Sathiaseelan [13] claim that the AdaBoost algorithm is influenced by noisy data and outliers, is not robust at identifying software flaws, and has poor modelling with linear structure.

1.1 Contributions

The essential objective of this paper is to display the best algorithms that give better accuracy and precision and reduce the time complexity by using Artificial Neural Network (ANN) including the particle swarm algorithm (PSO) and we compare the result of ANN-PSO with some of classification algorithms such as DT, KNN, ADC, GBC, NB and SVM and some algorithms of deep learning such as ANN and CNN. Finally, we produce the Proposed Model (ANN-PSO) is better with PROMISE dataset compared to other models while also reducing time and space complexities.

1.2 Research gap

As automation becomes increasingly critical in the software development life cycle, it is imperative to evaluate whether these optimization algorithms can handle the diverse range of test cases and scenarios effectively. Software systems often comprise intricateinterdependencies, dynamic data structures, and unique user inputs, making it essential to assess whether these algorithms can successfully navigate through such complexities and deliver accurate testing results.

In conclusion, the research gap lies in understanding the true potential of improved optimization algorithms in addressing the intricacies and complexities of software testing automation. By identifying the extent of their effectiveness and adaptability in this domain, we can bridge the gap between theoretical advancements and practical implementations, ultimately leading to more robust and efficient automated software testing processes.

1.3 Paper organization

The remainder of the paper is laid out as follows. The related work is illustrated in section 2. The description of the preliminaries of neural networks and Deep learning in section 3. Section 4 describes the suggested method used for solving the problem (proposed model) with flowchart and Pseudo code. The experimental results are shown in section 5. Section 6 describes the result and discussion, finally, section 7 brings the work to a close and identifies areas for potential research.

II. RELATED WORK

Mohd. Mustaqeem and Mohd. Saqib talked about how prior studies that used data without feature reduction were doomed by dimensionality. We employed a hybrid machine learning approach to address the issue, combining Principal component analysis (PCA) and Support vector machines (SVM) [16]. To perform our research, we used PROMISE [19] (CM1: 344 observations, KC1: 2109 observations) data from NASA's directory. The dataset was divided into two parts: training (CM1: 240 observations, KC1: 1476 observations) and testing (CM1: 104 observations, KC1: 633 observations) [14, 17]. The result of the paper is F-measures, Recall,

Accuracy, and Precisions are used to calculate test outcomes. On the KC1 dataset, we discovered that PC-SVM provided accuracy of 85.0 percent. Similarly, the accuracy of the CM1 dataset-specific model was 90.0 percent.
Anna Trudova, Michal Dolezel, they purpose the goal of this Systematic Literature Review (SLR) is to categorise AI approaches and related software testing tasks to which they may be used to emphasise the significance of AI in software test automation. Specifically, the effect of AI on certain activities was investigated. The SLR was focused on research studies that discussed the usage of AI techniques in software test automation for that reason [15]. To describe the role of artificial intelligence and its techniques in software test automation, the following research questions were stated:

What AI techniques can be applied for improving testing activities identified during answering the RQ1? .

The data gathered indicates that machine learning approaches, particularly various kinds of neural networks, tend to be the most often employed AI techniques: Bayesian network, artificial neural network, recurrent neural network, Q-learning, L*, etc. The Bayesian Network and computer vision techniques are among those that were applied to testing activities more frequently than others.

Dr. Subarna Shakya, Dr. S. Smys, they suggested research to improve software testing accuracy and dependability and doing research to generate test cases utilising a hybrid model of differential evolution and ant colony optimization. Traditional models like artificial neural networks and particle swarm optimization are contrasted with the suggested approach to verify its dependability [18, 19-20]. They employed three distinct data sets with various features to make the experimental procedure more challenging [21, 22].

ID	Name	Year	Method	Result	Limitations
1	Automation of software test data generation using genetic algorithm and reinforcement learning[24]	2021	MAAT Algorithm	MAAT algorithm has a success rate of 100%, while none of the other algorithms can reach more than 80% in this criterion	It works into test data generation only
2	Reliable Automated Software Testing Through Hybrid Optimization Algorithm[19]	2020	Hybrid ACO Algorithm	The proposed model hybrid ACO model attains accuracy range of rate of 96.2%, but particle swarm optimization attains an average of 95.5% and artificial neural network obtains an accuracy of 92% which is 4% lesser than the proposed model	It works into test data generation only
3	Automated Software Test Optimization using Language Processing[25]	2019	TLP based framework	Based on our experiments it is concluded that (1) Test execution time using TLP based framework is significantly low and (2) a test suite optimization of 83.78% is achieved through the proposed TLP framework	It works into test data generation only
4	Multiple-Implementation Testing of Supervised Learning Software[26]	2018	k-Nearest Neighbor (kNN) and Naive Bayes (NB)	In particular, 19 kNN implementations detect 13 real errors and 1 potential fault, while 7 NB implementations detect 16 real faults. Among the three widely used open-source ML projects, our technique can detect 7 true problems and 1 potential defect	It works into test data generation only
5	Artificial Intelligence in Software Test Automation: A Systematic Literature Review[18]	2020	Systematic Literature Review (SLR)	Most commonly used AI techniques appears to be from the field of machine learning, specifically different types of neural networks: Artificial Neural Network, Recurrent Neural Network, Bayesian Network; Q-learning; L* etc. Bayesian Network and techniques from the	It works into survey the AI techniques without experimental result

				Computer Vision field belong among the techniques that were used across more testing activities more frequently than others	
6	Software Testing Using G Genetic Algorithm [27]	2016	Genetic Algorithm (GA)	As a result, Genetic Algorithms are being utilized to improve the efficiency and processing time of Software testing by providing us with an automatic test case generator. The evolutionary creation of test cases can be used, and it has been shown to be more efficient and cost effective than Random Testing.	It works into test data generation only
7	Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization [28]	2019	hill-climbing algorithm (HCA), particle swarm optimization (PSO), firefly algorithm (FA), cuckoo search algorithm (CS), bat algorithm (BA) and artificial bee colony algorithm (ABC)	ABC, BA and PSO were the better optimal test suite generators, while CA, HCA and FA produced non-optimal test suites. On the other hand, BA, HCA and ABC were the faster algorithms with similar processing times for the process metrics. FA, PSO and CA were among the slower performing algorithms. Hence, ABC and BA present as suitable algorithms for TSG, while PSO can be improved in the future for the special case of TSG	It works into test data generation and search for the bugs and errors
8	Principal component based support vector machine (PC-SVM) [16]	2021	hybrid strategy that combined Principal component Analysis (PCA) and Support vector machines (SVM)	PC-SVM provided accuracy of 86.6 percent with 86.8% precision, 99.6% recall, and 92.8 percent F-measure. Similarly, the accuracy of the CM1 dataset-specific model was 95.2 percent, with 96.1 percent precision, 99 percent recall, and 97.5 percent F-measure	It works in search for the bugs and errors

Table 1. Related works for this study

The accuracy comparison between ANN, PSO and H-ACO. It is observed that proposed model attains better accuracy compared to other models. The proposed model hybrid ACO model attains accuracy range of rate of 96.2%, but particle swarm optimization attains an average of 95.5% and artificial neural network obtains an accuracy of 92% which is 4% lesser than the proposed model. It is observed from the figure, that the proposed model attains better sensitivity and specificity values than artificial neural network and particle swarm optimization model.

In the past, numerous studies have used a variety of techniques to detect software faults, including artificial neural networks (ANN), deep learning (DL), linear regression, K-nearest neighbours (KNN), decision trees, SVM, etc. [23]. According to Gondra, [24] a machine learning technique using ANN was developed to measure performance for excessive and indecisive datasets. To train ANN, historical software metric values with multiple errors were provided. To measure performance, sensitivity analysis was carried out, and ANN was trained using the training data and various evaluation criteria.

Prediction of software problems using one method SVM is utilized in many fields to forecast flaws, but its accuracy view limits their rate of prediction. Techniques such as feature selection and optimization are employed to improve accuracy. Optimization is carried out using the P-SVM algorithm, a mix of Particle Swarm Optimisation (PSO) and the SVM algorithm [25]. By clustering text documents using the PSO method, a more informative feature selection technique with lower dimensions can be obtained [25]. Text clustering is a technique that divides text into numerous groups, which reduces performance and lengthens calculation time. Consequently, a hybrid approach using particle swarm optimization and genetic operators is applied for the selection of more informative features [27]. But there is a limitation of using PSO it easily falls into local optimization in high-dimensional space it also has a low convergence rate in the iterative processing it works for large datasets, but our hybrid model PSO-ANN will work more effective rather than the P-SVM hybrid model.

In our paper, we are presenting the novel approach by combining the two most promising algorithm for optimization and feature selection to obtain better results with more accuracy. We are using a hybrid of PSO-ANN algorithm whose results and accuracy are more than the below-mentioned algorithm in our knowledge.

III. PRELIMINARIES

Machine learning is a subfield of artificial intelligence that empowers computers to learn and improve from experience without explicit programming. It involves the development of algorithms and statistical models that enable systems to recognize patterns and make data-driven decisions. One of the core concepts in machine learning is the use of neural networks, which are inspired by the functioning of the human brain. These networks consist of interconnected nodes, or artificial neurons, organized into layers. Through the process of training, where the model learns from labelled data, machine learning algorithms can make predictions, classify data, and even solve complex problems across various domains, such as image recognition, natural language processing, and recommendation systems [46].

Neural networks as in Fig 2 are a fundamental component of modern machine learning and artificial intelligence. They are computational models inspired by the structure and functioning of the human brain's neural connections. Each artificial neuron receives input data, processes it through an activation function, and produces an output that is passed to the next layer of neurons. This interconnectedness allows neural networks to detect patterns and relationships in data that might be challenging for traditional algorithms to identify. Neural networks can have different architectures, including feedforward, recurrent, and convolutional neural networks, each suited for specific tasks. The deepening of neural networks, known as deep learning, has led to significant advancements in various fields, including computer vision, speech recognition, and natural language understanding [47].

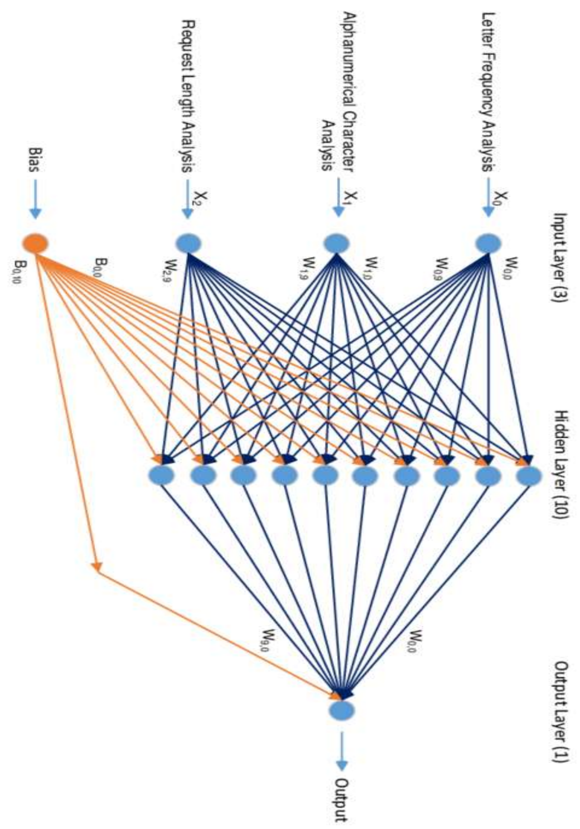


Figure. 2. Neural network model

One of the key advantages of using neural networks in software testing automation is their ability to handle large and diverse datasets with relative ease. By training on vast amounts of historical testing data, neural networks can learn patterns and relationships that might not be apparent to traditional testing approaches. This empowers them to identify and predict potential defects or vulnerabilities more accurately, enhancing the overall test coverage and reducing the risk of critical issues slipping through undetected. Furthermore, neural networks can significantly speed up the testing process. Their inherent parallel processing capabilities enable them to execute multiple test cases simultaneously, thus accelerating testing cycles and allowing for rapid feedback on the software's performance. As a result, software development teams can expedite the release cycle without compromising on the quality of the product. Neural networks also excel in the realm of anomaly detection. By continuously analyzing application behavior, they can recognize unusual patterns and deviations from expected outcomes. This makes them invaluable in scenarios where traditional rule-based testing falls short or becomes too cumbersome to maintain. The ability to detect outliers and potential bugs early on saves considerable time and effort for testers, enabling them to focus on more critical aspects of the software. However, while neural networks bring immense benefits to software testing automation, their implementation requires careful consideration and expertise. Designing and training a neural network tailored to the specific testing requirements demands skilled data scientists and testing professionals working together. Moreover, neural networks are not a silver bullet; they should be seen as complementary tools to traditional testing methodologies, enhancing their effectiveness rather than replacing them entirely. In conclusion, neural networks have revolutionized software testing automation by providing an intelligent, data-driven approach to identify defects, speed up testing cycles, and enhance overall test coverage. With continued research and advancements, the integration of neural networks into testing processes will undoubtedly continue to evolve, enabling software development teams to deliver high-quality products with greater efficiency and confidence ss[47].

Deep learning is a subfield of machine learning that focuses on training artificial neural networks with multiple layers, enabling the models to learn hierarchical representations of data. This layered architecture allows deep learning models to automatically extract and learn intricate features from raw input data. Deep learning has revolutionized various industries due to its ability to handle large amounts of data efficiently and extract meaningful insights. Some of the most remarkable achievements in deep learning include image and object recognition, machine translation, autonomous vehicles, and playing complex games like Go. However, training deep learning models often requires substantial computational resources, which has led to the development of specialized hardware and distributed computing techniques to accelerate the process. Despite the challenges, deep learning continues to be at the forefront of cutting-edge AI research, and its ongoing advancements promise to shape the future of artificial intelligence and the technologies we interact with daily [48].

3.1 Artificial Neural Network (ANN)

ANN is a computer method with biological roots that mimics the behaviour and learning processes of the human brain [28]. This method relies purely on the historical input-output dataset (example set) to learn the relationship between the data through training and does not explicitly require knowledge of the physical phenomena under inquiry [29]. An amazing generalisation capacity, which enables it to properly anticipate outputs for a fresh input data set, and the ability to deal with noisy data and uncertainties are only two of the many benefits that ANN-based models offer [30].

The ANN technique has been widely applied in multiple applications in engineering, medicine, meteorology, economics, psychology, and many other domains because to their many appealing properties [31, 32]. As a result, a number of research have investigated the use of ANNs to the evaluation of sandwich composite panels [33, 34, 35, 36]. However, studies that attempt to develop a generalised prediction model that can account for the influence of each of the honeycomb core's geometrical parameters—cell wall thickness (t), cell wall angle (), cell wall length (a), and core thicknesses (D)—tend to be scarce when it comes to honeycomb core sandwich composites. The few experiments reported in the literature [33, 34, 35, 36] were constrained by the cell geometry modifications and the geometrical parameters taken into account.

3.2 Particle Swarm Optimization (PSO)

PSO is a particularly promising and effective optimization technique for solving highly constrained nonlinear and non-convex optimization problems [38]. The PSO algorithm was first proposed by Kennedy and Eberhart [37] and is based on cooperative behaviour exhibited by species like fish schools and bird flocks. The locations of points (or particles) in the design space represent possible answers to an optimisation issue. In accordance with both its individual optimal position and the optimal position of the entire swarm during each generation, each particle updates its location [39].PSO offers numerous advantages over other optimisation approaches, including fewer parameters to tweak than many of its rivals, quick calculation times, and the ability to mix different techniques to create hybrid tools. Additionally, the PSO algorithm's iteration phase does not depend on the original solution to begin [40, 41].

The PSO method has been widely employed in many engineering applications due to its simplicity of use and quick searching speed [39, 41], including multiple studies that used PSO in honeycomb sandwich composite investigations. However, given the effect that each of these parameters has on the performance of honeycomb cores, it is surprising that these studies did not take into account all of the physical parameters of the honeycomb core [core thickness (D), cell wall angle (), cell wall thickness (t), and the cell wall length (a)] in their optimisation problems.

The PSO in its original form is defined by:

$$V_{id}^{t+1} = w.v_{id}^{t} + c_1.r_{1d}^{t}\left(P_{best,i}^{t} - x_{id}^{t}\right) + c_2.r_{2d}^{t}(G_{best}^{t} - x_{id}^{t})(1)$$

$$X_{id}^{t+1} = X_{id}^{t} + V_{id}^{t+1}, d = 1,2, \ldots\ldots n(2)$$

Where:

- v_{id}^{t}: is the velocity vector of the particle i in dimension d at time t.
- X_{id}^{t+1}: is the position vector of the particle i in dimension d at time t+1.
- w: is representative of the inertia weight
- $P_{best,i}^{t}$: is the personal best position of the particle i in dimension d found from initialization through time t.
- G_{best}^{t}: is the global best position of the particle i in dimension d found from initialization through time t.
- c_1, c_2 : are positive acceleration constants which are used to level the contribution of the cognitive and social components respectively;
- r_{1d}^{t}, r_{2d}^{t}: are random numbers from uniform distribution U (0, 1) at time t.

In PSO, the interaction of a group in nature is imitated in the sense that the group members are inclined to move toward the best member in the group. Hence, the behavior of each member is formed by the personal and social information as shown in Fig 3.

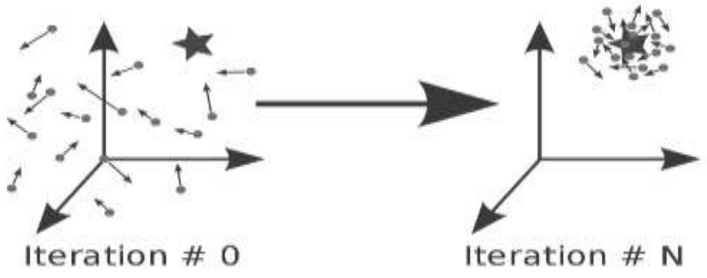


Figure. 3. PSO procedure

IV. PROPOSED MODEL (ANN-PSO)

The proposed model in Fig 4 is provide performs a binary classification task using a Particle Swarm Optimization (PSO) approach to select a subset of features from the dataset. Here's an explanation of the code step by step:

1. Importing Libraries: The necessary libraries and modules are imported, including warnings, pandas, tensorflow, scikit-learn (sklearn), seaborn, numpy, matplotlib, and keras.

2. Loading and Preprocessing Data:-
   The dataset is loaded from a CSV file using pandas, and the first five rows of the dataset are displayed. A LabelEncoder is used to encode the target variable ('defects') into numeric values and create a new column. The 'defects' column is dropped from the dataset, and the resulting DataFrame is stored in 'df'. The feature matrix (data_X) and the target variable (data_y) are extracted from 'df'. The data is split into training and testing sets using the train_test_split function from sklearn.

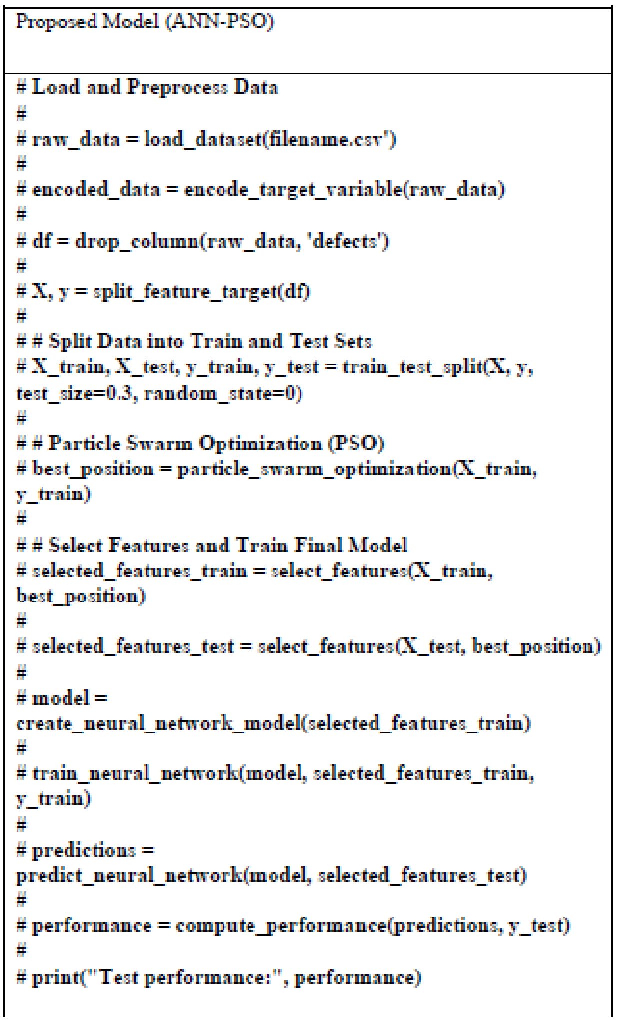
3. Setting Random Seeds: Random seeds are set to ensure reproducibility of the results.

4. Objective Function Definition: The objective function 'f_per_particle' is defined, which takes a binary mask 'm'. The function selects a subset of features based on the binary mask and performs classification using a neural network model. The performance of the model is computed using accuracy.The objective function value 'j' is calculated as a weighted combination of classification performance and the number of selected features.
5. Higher-Level Objective Function: The higher-level objective function 'f' is defined, which takes the entire swarm 'x' and an alpha value as inputs. The function evaluates the objective function 'f_per_particle' for each particle in the swarm. The negative objective function values are returned.

6. Particle Swarm Optimization (PSO): PSO parameters (options) are defined, including cognitive and social coefficients (c1 and c2), inertia weight (w), maximum velocity (k), and p-value. The PSO optimizer is initialized with the defined parameters and the number of dimensions (number of features). PSO algorithm is executed for a specified number of iterations, optimizing the objective function 'f'. The best cost and the

best position (binary mask) obtained from the optimization are returned.

7. Selecting Features and Training the Final Model: The selected features for training and testing are obtained by applying the best position (binary mask) to the training and testing sets. A neural network model is defined and compiled using the selected features as input. The model is trained on the training set for a specified number of epochs. Predictions are made on the selected features of the testing set. The performance of the model on the testing set is computed and printed.



Proposed Model (ANN-PSO)

```
# Load and Preprocess Data
#
# raw_data = load_dataset(filename.csv')
#
# encoded_data = encode_target_variable(raw_data)
#
# df = drop_column(raw_data, 'defects')
#
# X, y = split_feature_target(df)
#
# # Split Data into Train and Test Sets
# X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
#
# # Particle Swarm Optimization (PSO)
# best_position = particle_swarm_optimization(X_train,
y_train)
#
# # Select Features and Train Final Model
# selected_features_train = select_features(X_train,
best_position)
#
# selected_features_test = select_features(X_test, best_position)
#
# model =
create_neural_network_model(selected_features_train)
#
# train_neural_network(model, selected_features_train,
y_train)
#
# predictions =
predict_neural_network(model, selected_features_test)
#
# performance = compute_performance(predictions, y_test)
#
# print("Test performance:", performance)
```

Figure. 4. Proposed Methodology flowchart

V. EXPERIMENTAL SETUP

5.1 Dataset Description

The PROMISE datasets repository was used to collect the data [26]. The proposed approach was put into practise using the CM1 [42], PC1 [43], KC2 [45] and KC1 [44] datasets. We have listed descriptions of the various attributes used for analysis in Table 2. Additionally, we divided the two datasets into two separate portions, one for training the model and the other for assessing the results. 240 observations from the CM1 data were used to train the model, and 104 observations were used to test the model. The model was trained using 1476 observations, just like in KC1, and tested with 633 records and 776 observations from the PC1 data were used to train the model, and 333 observations were used to test the model.

5.2 Parameter Settings

Five hundred repetitions qualified the examined and tested models. The input sheet for ANN is based on No. Since the beneficial procedure was used, there were thousands of concealed nodes. It required more hidden nodes than conventional algorithms did. The 2-class was introduced by a single output layer node. The Particle Swarm Optimisation technique was developed to optimise hyperparameters like (number of estimators, max depth, etc.) (the number of iterations......... 10 iterations were used) as shown in table 3.

5.3 Evaluation criteria

We used a confusion matrix to assess the outcomes. A simple classification of predicted and actual values using the confusion matrix serves as a convincing example of the same. After receiving the values, we perform additional calculations, such as model correctness, precision, F-score. For each dataset, three confusion matrices were produced. A representation of actual value and predictions based on testing datasets is called a confusion matrix. The confusion matrix controls the model's accuracy, and it is also possible to calculate some other performance parameters (such as recall and precision).

Attribute name	Description of attribute
LOC	Counts the total number of line in the module
Iv(g)	Design complexity analysis (McCabe)
Ev(g)	McCabe essential complexity
N	Number of operators present in the software module
v(g)	Cyclomatic complexity measurement (McCabe)
D	Measurement difficulty
B	Estimation of effort
L	Program length
V	Volume
I	Intelligence in measurement
E	Measurement effort
Locomment	Line of comments in software module
Loblank	Total number of blank lines in the module
uniq_op	Total number of unique operators
uniq_opnd	Total number of unique operand
T	Time estimator
Branchcount	Total number of branch in the software module
total_op	Total number of operators
Total_opnd	Total number of operators
Locodeandcomment	Total number of line of code and comments
Defects/Problems	Information regarding defect whether the defect is present or not

Table 2. Attributes description of PROMISE dataset

Model	Parameter	Values
ANN	Input nodes	based on No. Features
	Activation fun for output nodes	sigmoid
	Output nodes	1
	No. of Iterations	100
PSO	Number of particles	50
	Number of iterations	100
	Dimension	Number of features (21)
	α in fitness function	0.88
	β in fitness function	0.01
	Options	{‘c1’:0.5, ‘c2’: 0.5, ‘w’: 0.9, ‘K’: 30, ‘p’: 2}

Table 3. Parameters Settings

The accuracy of the model can be calculated from the following formula.

$$accuracy = \frac{truepositives + truenegatives}{truepositives + falsepositives + falsenegatives + truenegatives} \quad (3)$$

The precision of the model can be calculated using the following formula

$$Precision = \frac{truepositives}{truepositives + falsenegatives} \quad (4)$$

Recall of the model calculated using correctly predicted positive observations and total numbers of positive models in testing datasets.

$$Recall = \frac{truepositives}{truepositives \ + falsepositives} \quad (5)$$

Finally, F-measure is computed from the following formula.

$$F1score = \frac{2 * (precision * recall)}{precision + recall} \quad (6)$$

Confusion matrix as shown in Fig 5:

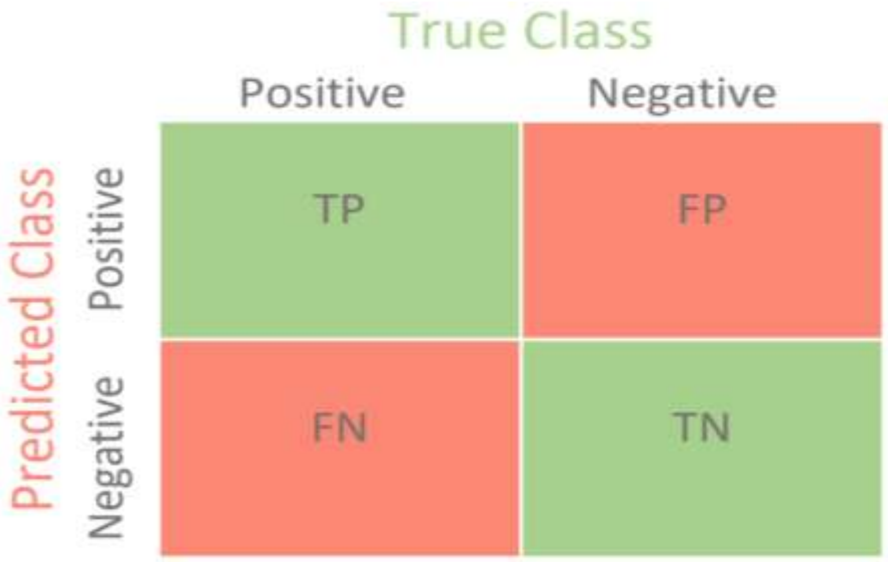


Figure. 5. Confusion matrix of classification rules

VI. RESULT AND DISCUSSION

In this section, we will present the comparison between the classification algorithms and artificial neural network and Convolutional Neural Network based on the paper in related work with the PROMOSE dataset and apply each algorithm on dataset (PC1, CM1, KC1) .

After reading the dataset, presumably containing both features and the target variable and Create a feature matrix (input features) and a target variable array (output labels) required for machine learning then Split Data into Train and Test Sets and Apply the PSO and return the best position then select features and train final model and Select features based on the best position obtained from PSO, define and compile a neural network model, train the model using selected features, make predictions on the test set, compute model performance on the test set and Print test performance.

In table 4, 5, 6. After the implementation of the Classification models and proposed model, we compared our model with previous studies based on precision, recall, F-measure, and accuracy of classification. We have found that the PSO-ANN is more accurate than the other. In KC1 dataset analysis, in Fig [6, 7, 8, 9] we have found precision 0.94, recall 0.90, F-measure 0.95, and accuracy 86.00 that is a significant improvement in the analysis in the table 6. Same as in CM1 dataset analysis, we have found precision 90.0, recall 1.0, F-measure 95.0, and accuracy 90.88. In the table 4. Same as in PC1 dataset analysis, we have found precision 95.0, recall 1.0, F-measure 95.0, and accuracy 94.00. In the table 5. The accuracy of training data in Fig 10 and test data in Fig 11.

		CM1							
		Train Dataset				Test Dataset			
		Acc	F1s	Pre	Rec	Acc	F1s	Pre	Rec
1	LogisticRegression	89.37	0.94	0.91	0.98	89.00	0.95	0.91	0.97
2	KNeighborsClassifier	89.95	0.95	0.91	0.99	89.67	0.95	0.91	0.97
3	DecisionTreeClassifier	84.49	0.91	0.92	0.91	82.00	0.90	0.90	0.90
4	RandomForestClassifier	89.66	0.95	0.90	0.99	89.33	0.94	0.90	0.97
5	GradientBoostingClassifier	88.22	0.94	0.91	0.96	87.33	0.93	0.90	0.96
6	AdaBoostClassifier	85.93	0.93	0.90	0.95	86.00	0.92	0.90	0.95
7	LinearDiscriminantAnalysis	88.23	0.94	0.92	0.95	87.33	0.93	0.90	0.97
8	QuadraticDiscriminantAnalysis	53.47	0.68	0.90	0.54	89.67	0.95	0.91	0.97
9	GaussianNB	52.64	0.66	0.94	0.51	86.67	0.93	0.91	0.94
10	SVC(Linear)	90.23	0.95	0.90	1.00	89.00	0.95	0.90	0.97
11	SVC(probability = true)	89.95	0.95	0.90	1.00	89.00	0.95	0.90	0.97
12	SVC(poly)	88.80	0.94	0.91	0.99	89.00	0.95	0.91	0.97
13	SVC(sigmoid)	89.95	0.95	0.92	0.97	89.00	0.95	0.90	0.97
14	PC-SVM	89.95	0.95	0.92	0.97	90.00	0.95	0.90	0.97
15	Proposed Model (PSO-ANN)	98.95	0.98	0.96	0.97	91.00	0.98	0.95	0.99

Table 4. Classification algorithms and proposed model with CM1 dataset

		PC1							
		Train Dataset				Test Dataset			
		Acc	F1s	Pre	Rec	Acc	F1s	Pre	Rec
1	LogisticRegression	92.66	0.96	0.94	0.99	92.79	0.96	0.94	0.95
2	KNeighborsClassifier	92.66	0.96	0.94	0.98	92.39	0.97	0.94	0.97
3	DecisionTreeClassifier	90.08	0.95	0.95	0.94	90.69	0.95	0.95	0.96
4	RandomForestClassifier	92.91	0.96	0.93	0.99	92.39	0.97	0.94	0.97
5	GradientBoostingClassifier	92.40	0.96	0.94	0.98	91.59	0.96	0.95	0.96
6	AdaBoostClassifier	92.53	0.96	0.94	0.98	92.29	0.97	0.96	0.98
7	LinearDiscriminantAnalysis	92.14	0.96	0.94	0.98	89.79	0.95	0.94	0.95
8	QuadraticDiscriminantAnalysis	37.65	0.51	0.94	0.35	92.99	0.97	0.94	0.97
9	GaussianNB	89.05	0.94	0.95	0.93	83.78	0.91	0.95	0.87
10	SVC(Linear)	92.78	0.96	0.93	1.00	92.99	0.97	0.94	0.97
11	SVC (probability = true)	93.30	0.97	0.93	1.00	92.99	0.97	0.94	0.97
12	SVC (poly)	92.91	0.96	0.94	0.94	92.79	0.96	0.94	0.98
13	SVC (sigmoid)	90.47	0.95	0.93	0.97	92.29	0.97	0.94	0.97
14	PC-SVM	89.95	0.95	0.92	0.97	93.00	0.95	0.90	0.95
15	Proposed Model (PSO-ANN)	97.90	0.98	0.96	0.97	94.00	0.99	0.98	1.00

Table 5. Classification algorithms with PC1 dataset

		KC1							
		Train Dataset				Test Dataset			
		Acc	F1s	Pre	Rec	Acc	F1s	Pre	Rec
1	LogisticRegression	86.86	0.93	0.88	0.98	83.25	0.91	0.85	0.97
2	KNeighborsClassifier	85.64	0.92	0.89	0.96	81.52	0.90	0.84	0.96
3	DecisionTreeClassifier	84.01	0.91	0.90	0.91	75.99	0.86	0.85	0.86
4	RandomForestClassifier	86.58	0.93	0.88	0.98	82.46	0.90	0.84	0.97
5	GradientBoostingClassifier	87.19	0.93	0.89	0.97	81.67	0.90	0.85	0.94
6	AdaBoostClassifier	85.84	0.92	0.88	0.97	82.15	0.90	0.85	0.95
7	LinearDiscriminantAnalysis	86.38	0.92	0.89	0.96	81.36	0.89	0.85	0.94
8	QuadraticDiscriminantAnalysis	33.14	0.38	0.91	0.24	82.62	0.85	0.85	0.90
9	GaussianNB	83.20	0.90	0.90	0.91	81.20	0.89	0.87	0.91
10	SVC(Linear)	86.04	0.92	0.86	0.99	82.78	0.91	0.83	0.99
11	SVC (probability = true)	86.92	0.93	0.87	0.99	82.62	0.90	0.84	0.97
12	SVC (poly)	86.59	0.93	0.87	0.99	83.41	0.91	0.85	0.97
13	SVC (sigmoid)	79.74	0.88	0.88	0.89	78.20	0.87	0.85	0.90
14	PC-SVM	89.95	0.95	0.92	0.97	84.45	0.86	0.86	0.95
15	Proposed Model (PSO-ANN)	97.90	0.98	0.96	0.97	86.00	0.95	0.90	0.99

Table 6. Classification algorithms with KC1 dataset

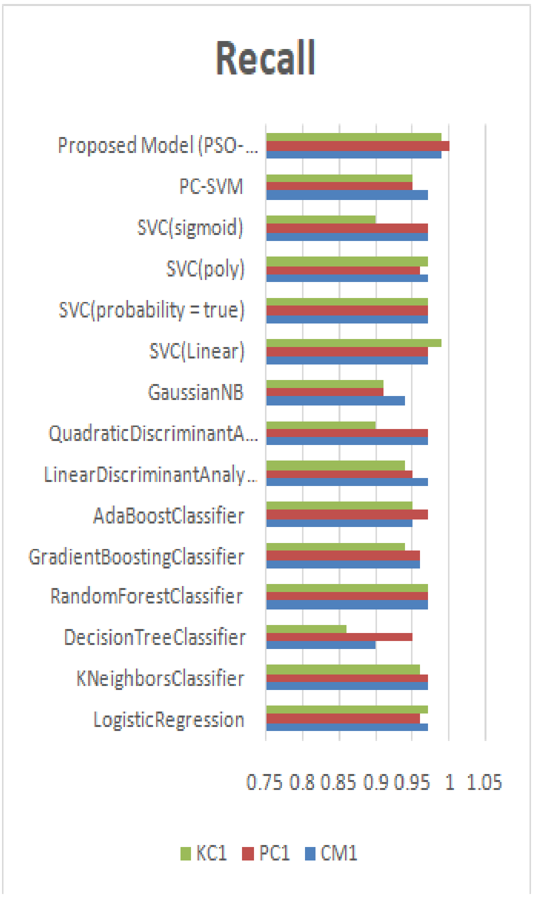


Figure. 6. Recall for CM1, PC1 and KC1 dataset

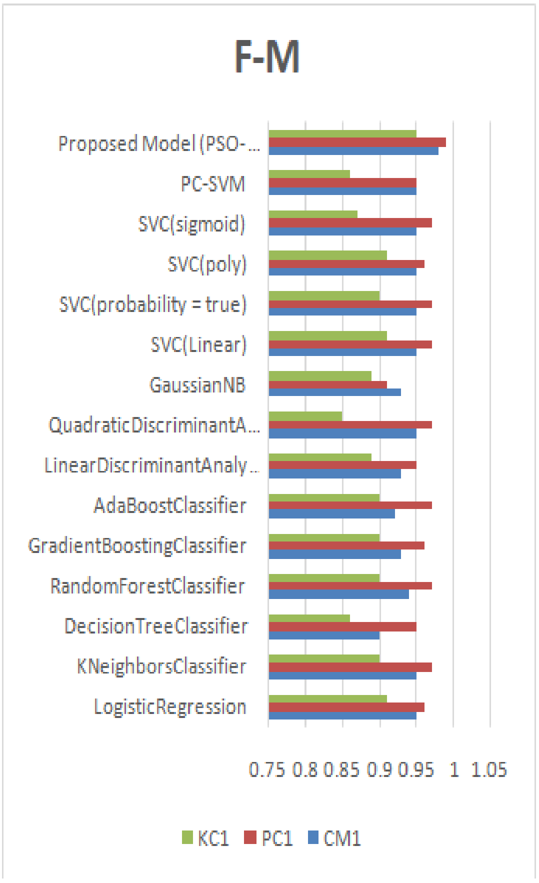


Figure. 7F-M for CM1, PC1 and KC1 dataset

Accuracy

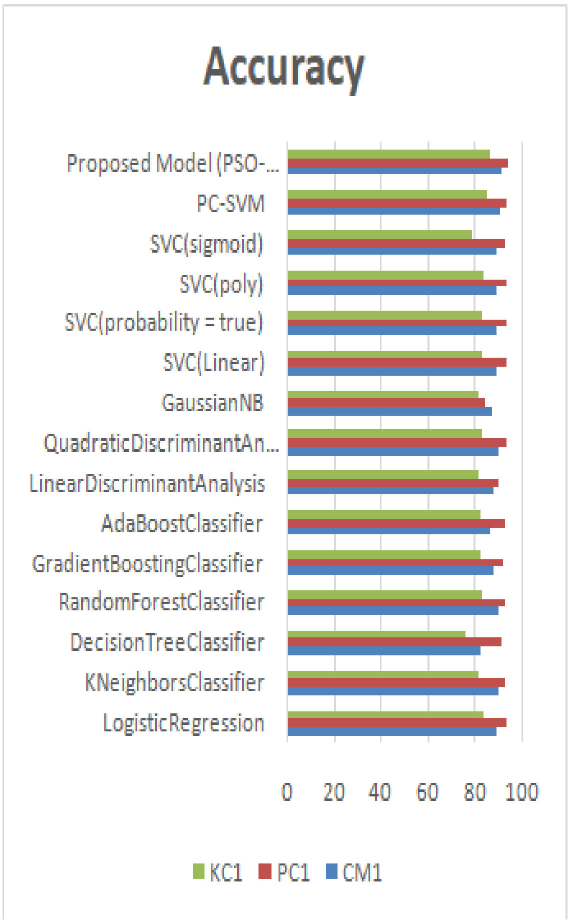


Figure. 8. Accuracy for CM1, PC1 and KC1 dataset

Precision

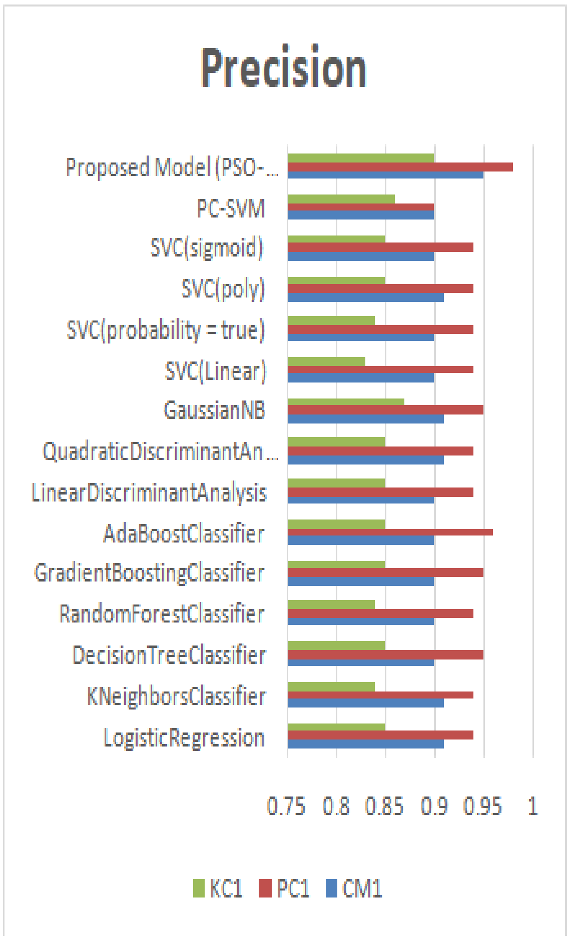


Figure. 9. Precision for CM1, PC1 and KC1 dataset

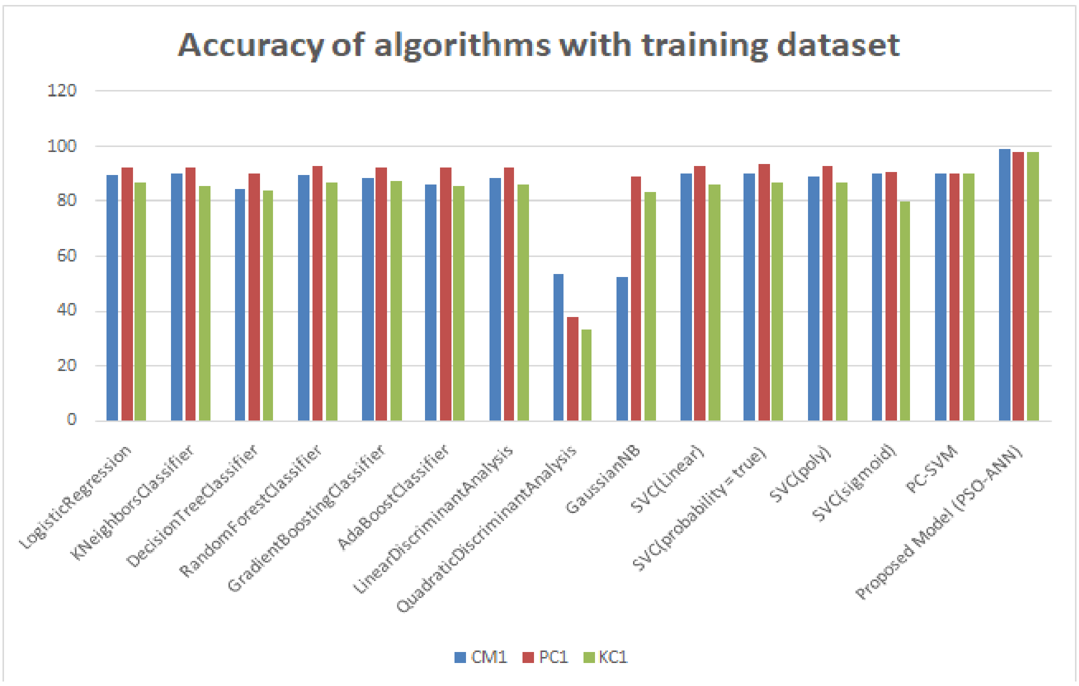


Figure. 10 accuracies of algorithms with training dataset

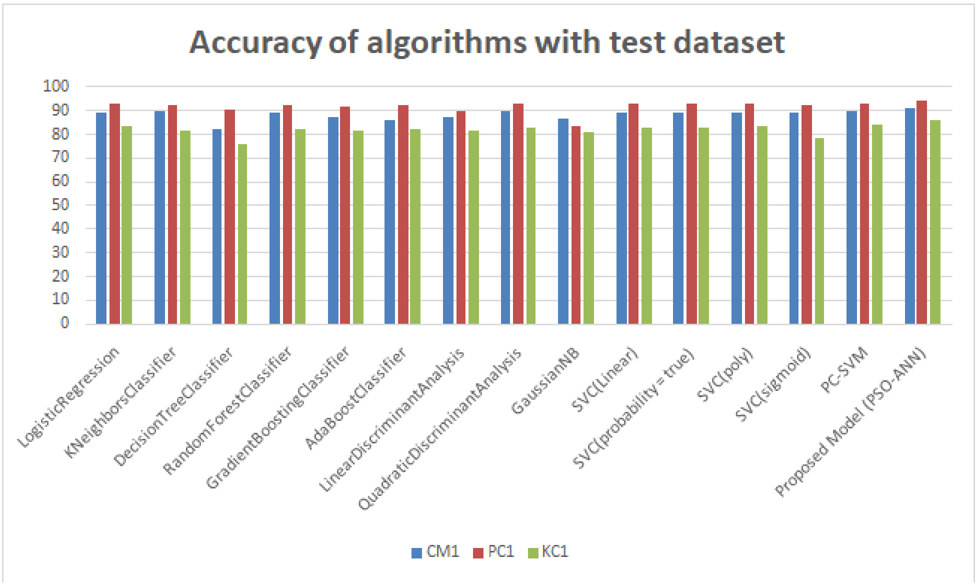


Figure. 11 accuracies of algorithms with test dataset

Algorithm 1: Proposed Model (ANN-PSO)

```
# Start
# |--- Load and Preprocess Data
# |      |
# |      |-- Read dataset
# |      |-- Encode target variable
# |      |-- Create feature matrix and target variable array
# |--- Split Data into Train and Test Sets
# |      |
# |      |-- Split feature matrix and target variable into train and test sets
# |--- Particle Swarm Optimization (PSO)
# |      |
# |      |-- Initialize PSO parameters
# |      |-- Initialize PSO optimizer
# |      |-- Perform optimization using PSO
# |      |
# |      |-- For each iteration:
# |      |      |-- Update particle positions and velocities
# |      |      |-- Evaluate objective function for each particle
# |      |      |-- Update global best position and cost
# |      |
# |      |-- Return best position (binary mask)
# |--- Select Features and Train Final Model
# |      |-- Select features based on the best position obtained from PSO
# |      |-- Define and compile a neural network model
# |      |-- Train the model using selected features
# |      |-- Make predictions on the test set
# |      |-- Compute model performance on the test set
# |      |-- Print test performance
  # End
```

VII. CONCLUSION

The ability to detect faults accurately and early on is a key factor in determining a software's quality. Software defects can be found early on using a variety of methods that researchers and scientists have previously devised. The most effective approach is machine learning-based due to the classifiers' learning mechanisms. We have examined the performance of the ANN-PSO model implemented on the PROMISE dataset repository by using previously developed conventional and traditional methods.
We select ANN-PSO to overcome the problem of the curse of dimensionality and reduce the computational requirements of the proposed task. It was the major problem with the previous research methodology mentioned then after we used PSO-ANN with multi-layer, which is a very powerful methodology of classification in ML. We have found PSO-ANN more sustainable in terms of many cases like it can give better result with small datasets and doesn't affect by outliers.

Some improvements on used models will be added by using metaheuristic optimization and classification algorithms to find the best solution.

REFERENCE

[1] Hussam Hourani, Ahmad Hammad, Mohammad Lafi, "The Impact of Artificial Intelligence on Software Testing", 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT).
[2] Mark Last, Menahem Friedman, Abraham Kandel, "The Data Mining Approach to Automated Software Testing", August 24-27, 2003, Washington, DC, USA.
[3] Ms.Karuturi Sneha, Mr. Malle Gowda M, "Research on Software Testing Techniques and Software Automation Testing Tools". International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS-2017).
[4] Jihyun Lee, Sungwon Kang, and Danhyung Lee, "A Survey on Software Testing Practices", All content following this page was uploaded by Sungwon Kang. 15 January 2015.
[5] Sumit Mahapatra and Subhankar Mishra, "Usage of Machine Learning in Software Testing". July 11, 2020.
[6] Lionel C. Briand. Novel applications of machine learning in software testing. Quality Software, International Conference on, 0:3–10, 2008.
[7] Du Zhang and Jeffrey Tsai. Machine learning and software engineering. Software Quality Journal, 11:87–119, 2003. 10.1023/A:1023760326768.
[8] Huang, T. M., Kecman, V., & Kopriva, I. (2006). Kernel based algorithms for mining huge data sets (Vol. 1). Heidelberg: Springer.
[9] Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In 11th annual conf. of the international speech communication association.

[10] Gulli, A., & Pal, S. (2017). Deep learning with Keras. Packt Publ. Ltd.

[11] Mustafa, Mohamed, Abdullah, "A Survey on Software Testing Automation using Machine Learning Techniques", International Journal of Computer Applications, February 2022.

[12] Cohen-Almagor, R.: Internet history. Int. J. Technoethics 2, 45–64 (2011).

[13] Naughton, J.: The evolution of the Internet: from military experiment to general purpose technology. J. Cyber Policy 1(1), 5–28 (2016).

[14] Mohd. Mustaqeem, Mohd. Saqib, "Principal component based support vector machine (PC-SVM): a hybrid technique for software defect detection", part of Springer Nature 2021, Published online: 16 April 2021.

[15] Anna Trudova, Michal Dolezel, "Artificial Intelligence in Software Test Automation: A Systematic Literature Review", 2020.

[16] Gondra, I.: Applying machine learning to software fault-proneness prediction. J. Syst. Softw. 81(2), 186–195 (2008).

[17] Yang, B., Li, X: A study on software reliability prediction based on support vector machines (2008).

[18] Dr. Subarna Shakya,Dr. S. Smys, "Reliable Automated Software Testing Through Hybrid Optimization Algorithm".Journal of Ubiquitous Computing and Communication Technologies (UCCT) (2020).

[19] Ajmer Singh, Rajesh Bhatia, Anita Singhrova (2018). Taxonomy of machine learning algorithms in software fault prediction using object-oriented metrics. Procedia Computer Science. 132. 993-1001.

[20] Alireza Haghighatkhah, Ahmad Banijamali, Olli-Pekka Pakanen, Markku Oivo, Pasi Kuvaja (2017). Automotive software engineering: A systematic mapping study. Journal of Systems and Software. 128, 25-55.

[21] Amir Elmishali, Roni Stern, Meir Kalech (2018). An Artificial Intelligence paradigm for troubleshooting software bugs. Engineering Applications of Artificial Intelligence. 69, 147-156.

[22] Fuqun Huang, Bin Liu (2017). Software defect prevention based on human error theories. Chinese Journal of Aeronautics. 30(3):1054-1070.

[23] Gondra, I.: Applying machine learning to software fault-proneness prediction. J. Syst. Softw. 81(2), 186–195 (2008).

[24] Can, H., Jianchun, X., Ruide, Z., Juelong, L., Qiliang, Y., Liqiang, X.: A new model for software defect prediction using Particle Swarm Optimization and support vector machine, In: Proceedings of the 2013 25th Chinese Control and Decision Conference (CCDC), pp. 4106–4110 (2013).

[25] Espejo, P.G., Ventura, S., Herrera, F.: A survey on the application of genetic programming to classification. IEEE Trans. Syst. Man Cybern Part C 40(2), 121–144 (2010).

[26] ayyad Shirabad, J., Menzies, T.J.: The {PROMISE} Repository of Software Engineering Databases (2005).

[27] Williams, L.J.: Principal component analysis, pp. 433–459 (2010).

[28] Pandey, D.S.; Das, S.; Pan, I.; Leahy, J.J.; Kwapinski, W.: Artificial neural network based modelling approach for municipal solid waste gasification in a fluidized bed reactor. Waste Manag. 58, 202–213 (2016).

[29] Azizi, S.; Awad, M.M.; Ahmadloo, E.: Prediction of water holdup in vertical and inclined oil–water two-phase flow using artificial neural network. Int. J. Multiph. Flow 80, 181–187 (2016.

[30] Baughman, D.R.; Liu, Y.A.: Neural Networks in Bioprocessing and Chemical Engineering. Academic press, New York, USA (1995).

[31] Kumar, R.; Aggarwal, R.K.; Sharma, J.D.: Comparison of regression and artificial neural network models for estimation of global solar radiations. Renew. Sustain. Energy Rev. 52, 1294–1299 (2015).

[32] Kan, C.W.; Song, L.J.: An artificial neural network model for prediction of colour properties of knitted fabrics induced by laser engraving. Neural Process. Lett. 44, 639–650 (2016).

[33] Lanzi, L.; Bisagni, C.; Ricci, S.: Neural network systems to reproduce crash behavior of structural components. Comput. Struct. 82(1), 93–108 (2004).

[34] Li, J.; Chen, X.; Wang, H.: Comparison of artificial neural networks with response surface models in characterizing the impact damage resistance of sandwich airframe structures. 2009 Second Int. Symp. Comput. Intell. Des. 2, 210–215 (2009).

[35] Sun, G.; Li, G.; Stone, M.; Li, Q.: A two-stage multi-fidelity optimization procedure for honeycomb-type cellular materials. Comput. Mater. Sci. 49(3), 500–511 (2010).

[36] Esfahlani, S.S.; Shirvani, H.; Shirvani, A.; Nwaubani, S.; Mebrahtu, H.; Chirwa, C.: Hexagonal honeycomb cell optimisation by way of meta-model techniques. Int. J. Crashworthiness 18(3), 264–275 (2013)

[37] Kennedy, J.; Eberhart, R.: Particle swarm optimization. IEEE Int. Conf. Neural Netw. (ICNN) 4, 1942–1948 (1995)

[38] Assarzadeh, Z.; Naghsh-Nilchi, A.R.: Chaotic particle swarm optimization with mutation for classification. J. Med. Signals Sens 5(1), 12–20 (2015)

[39] Tao, W.; Liu, Z.; Zhu, P.; Zhu, C.; Chen, W.: Multi-scale design of three dimensional woven composite automobile fender using modified particle swarm optimization algorithm. Compos. Struct. 181, 73–83 (2017)

[40] AlRashidi, M.R.; El-Hawary, M.E.: A survey of particle swarm optimization applications in electric power systems. IEEE Trans. Evol. Comput. 13(4), 913–918 (2009)

[41] Hasanien, H.M.: Particle swarm design optimization of transverse flux linear motor for weight reduction and improvement of thrust force. IEEE Trans. Industr. Electron. 58(9), 4048–4056 (2010).

[42] CM1 Dataset, 2023, Available: http://promise.site.uottawa.ca/SERepository/datasets/cm1.arff

[43] PC1 Dataset, 2023, Available: http://promise.site.uottawa.ca/SERepository/datasets/pc1.arff

[44] KC1 Dataset, 2023, Available: http://promise.site.uottawa.ca/SERepository/datasets/kc1.arff

[45] KC2 Dataset, 2023, Available: http://promise.site.uottawa.ca/SERepository/datasets/kc2.arff

[46] Pedro Domingos, A Few Useful Things to Know About Machine Learning, CommuniCAtionS oF thE ACm (2021)

[47] Geoffrey E. Hinton, S.O, Yee-Whye The, A Fast Learning Algorithm for Deep Belief Nets, Neural Computation, Vol. 18, No. 7, Pages 1527-1554 (2006)

[48] Yann LeCun, Y.B, G.H, Deep Learning, Nature, Vol. 521, Pages 436-444(2015)

★★★