NN and ACO Based Software Fault Proneness Prediction Model

Laxmi Dewangan
Shri Shankaracharya Group of Institution
Dept. of Information Technology
Junwani Bhilai, Chhattisgarh, India
laxmilp081092@gmail.com

Prof. Anish Lazrus
Shri Shankaracharya Group of Institution
Dept. of Information Technology
Junwani Bhilai, Chhattisgarh, India
anishlazrus@gmail.com

Abstract

Software technologies discipline contains a few prediction methodologies, for example, test exertion prediction, revision cost prediction, fault prediction, reusability prediction, security prediction, exertion prediction, and quality prediction. Be that as it may, a large portion of these prediction approaches are still in preparatory stage and more research ought to be led to achieve powerful models. Fault Proneness Prediction Models are the prepared models to foresee important programming quality characteristic, for example, fault proneness utilizing object situated programming measurements. Machine learning strategies with optimization can be utilized for prediction of the product quality properties. In this paper we propose a novel mechanism in combination of Neural Network and Ant Colony Optimization technique to predict the fault in software's. The proposed mechanism is far too better than the alone back propagation method.

Keywords: *Software Fault Prediction, Prone Detection, SVM, Prediction Techniques, ACO, Neural Network.*

I. INTRODUCTION

Software fault prediction approaches utilize past programming measurements and fault information to foresee fault-inclined modules for the following arrival of programming. In the event that a mistake is accounted for amid framework tests or from field tests, that module's fault information is set apart as 1, generally 0. For prediction demonstrating, programming measurements are utilized as free factors and fault information is utilized as the reliant variable. In this way, we require a rendition control framework, for example, Subversion to store source code, a change administration framework, for example, Bugzilla to record faults, and an apparatus to gather item measurements from adaptation control framework. Parameters of the prediction demonstrate are registered by utilizing past programming measurements and fault information. Software fault prediction models have been examined since 1990s as of not long ago and fault-inclined modules can be distinguished before framework tests by utilizing these models. As per late investigations, the likelihood of detection (PD) (71%) of fault prediction models might be higher than PD of programming surveys (60%) if a strong model is fabricated.

Advantages of programming fault prediction are recorded as takes after:

- Achieving an exceedingly reliable framework,
- Enhancing test process by concentrating on fault-inclined modules,
- Choice of best outline from plan choices utilizing object-oriented measurements,
- Recognizing refactoring hopefuls that are anticipated as fault prone,

Enhancing quality by enhancing test process.

II. PARAMETERS FOR PERFORMANCE COMPARISONS OF SOFTWARE PRONENESS EVAULUATION

The accompanying sub-areas give the essential meanings of the execution parameters utilized for prone prediction [7].

A. *Precision*

It is utilized to quantify how much the repeated estimations under unaltered conditions demonstrate similar outcomes.

$$Precision = \frac{TP}{FP+TP}$$

B. Recall

It shows the what number of the applicable item that are to be recognized. it is represented as:

$$Recall = \frac{TP}{FN+TP}$$

C. F-Measure

F-Measure join the exactness and recall numeric value to give a solitary score, which is characterized as the harmonic mean of the recall and accuracy. F-Measure is communicated as:

$$F - Measure = \frac{2 * Recall * Precision}{Recall + Precision}$$

D. Specificity

Specificity concentrate on how adequately a classifier recognizes the negative labels. It is characterized as:

$$Specificity = \frac{TN}{FP+TN}$$

E. Accuracy

Accuracy measure is the extent of anticipated fault-inclined modules that are investigated out of all modules. It is characterized as:

$$Accuracy = \frac{TN+TP}{TP+TN+FP+FN}$$

Fault prediction is useful in choosing the measure of exertion required for programming improvement. A great number of methodologies have been considered and assessed on programming items to decide best reasonable approach for fault prediction in light of certain execution criteria (Accuracy, recall, precision and so forth.) [8, 9]. However less critical work has been done on achievability of fault prediction approach. In this investigation, a cost assessment structure has been proposed which performs cost based examination for misclassification of faults [10].

III. ANN TECHNIQUE

We utilized the ANN to lessen dimensionality of the measurements space.

ANN-based classifiers can join both factual and basic data and accomplish preferable execution over straightforward least separation classifiers. An ANN has the accompanying qualities:

1. It has a quick preparing process, basic structure and great extendibility.
2. It gains as a matter of fact, subsequently, has no requirement for any from the earlier displaying suppositions.
3. It is equipped for deducing complex non-straight info – yield changes.

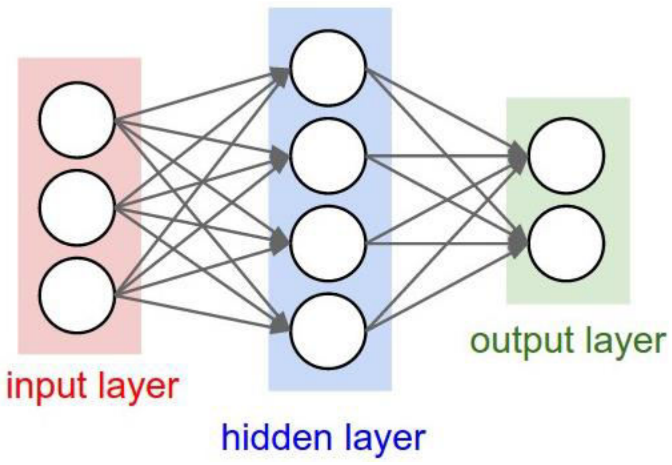

Fig. 1. ANN with 3 Layers

Accordingly in light of the advantages of ANN, multi-layered ANN, normally utilizing the back engendering (BP) calculation, is likewise generally utilized as a part of programming designing [8]. In this paper, a powerful decreased dimensionality approach utilizing ANN as per the attributes of programming measurements is proposed.

IV. LITERATURE SURVEY

Lee et al. [11], Author directs an examination of basic points around there, including techniques for assessing the viability of fault-inclination prediction models, issues of concerns when fabricating the prediction models, and also discoveries shared by the scholastic group. The proposed technique can't configuration to take skewed dataset. Thus require assist improvement.

Arisholm et al. [12] proposed cost viability (CE) by requesting the modules by their anticipated imperfection thickness and after that plotting the quantity of real faults found against the aggregate sum of code investigated beginning with the modules with the most astounding anticipated deformity thickness.

Russo [13] presented the TTV (Train, Test, and Validate) calculation went for formalizing the model preparing and prediction assessment process. TTV contains three stages: preparing a prediction model to locate the best parameters of the model; utilizing the prepared model on new information to figure the prediction execution; and approving the model on encourage new information for prediction.

The proposed TTV intends to formalize the Lessmann et al. [14] proposal that the appraisal and determination of a fault-inclination prediction model ought not be founded on classification exactness alone but instead ought to be contained a few extra criteria like computational proficiency, usability, and fathomability.

D. Rodrguez et al. [15], Author propose EDER-SD (Evolutionary Decision Rules for Subgroup Discovery), a this calculation in view of developmental calculation that instigates rules depicting just fault-inclined modules. EDER-SD has the benefit of working with nonstop factors as the states of the guidelines are characterized utilizing interims.

Martin Shepperd et al. [16], Author contemplated on the openly accessible NASA datasets have been broadly utilized as a component of this exploration to classify programming modules into deformity inclined and not desert inclined classifications. In such manner, the Promise Data Repository 2 has served an essential part in making programming building informational indexes freely accessible. For instance, there are 96 programming deformity datasets accessible.

Ezgi Erturk et al. [17], Author proposed another technique Adaptive Neuron Fuzzy Inference System (ANFIS) for the product fault prediction. Information are gathered from the PROMISE Software Engineering Repository, and McCabe measurements are chosen since they completely address the programming exertion. The outcomes accomplished were 0.7795, 0.8685, and 0.8573 for the SVM, ANN and ANFIS strategies, separately.

Martin Shepperd et al. [18], Author displayed a novel benchmark structure for programming imperfection prediction. In the system includes both assessment and prediction. In the assessment arrange, distinctive learning plans are assessed by that plan chose. At that point, in the prediction arrange, the best learning plan is utilized to manufacture an indicator with every single authentic datum and the indicator is at long last used to anticipate imperfection on the new information

David Gray [19], in this paper the principle concentrate is on classification examination as opposed to classification execution, it was chosen to classify the preparation information instead of having some type of analyzer set. It includes a manual investigation of the predictions made by Support vector machine classifiers utilizing information from the NASA Metrics Data Program storehouse.

Surndha Naidu et al. [20], in this paper concentrated on finding the aggregate number of imperfections with a specific end goal to decrease the time and cost. Here for imperfection classification utilized ID3 (Iterative Dichotomiser calculation. ID3 is a calculation imagined by Ross Quinlan used to create a choice tree from a dataset. The deformities were classified in light of the five characteristic esteems, for example, Volume, Program length, Difficulty, Effort and Time Estimation.

V. PROPOSED METHODOLOGY

In this section we present the proposed method in detail. Fig. 2. Shows the proposed system architecture in detail.

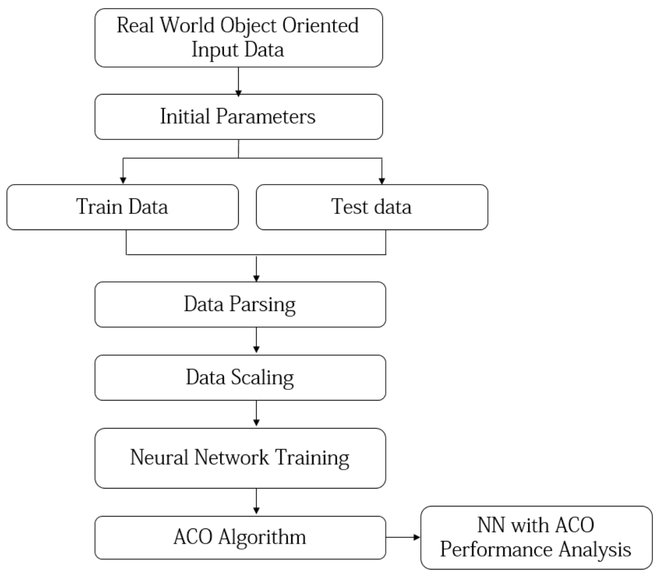

Fig. 2. Proposed System Architecture

A. *Real World Object Oriented Input Dataset*

The ANT packages with different classes are considered for analysis. We prepared two types of dataset which is for testing and training. The testing dataset is relatively small than training ones.

B. *Initial Parameters*

There are various parameters which need to set before running NN and ACO algorithm. The parameters such as number of inputs, number of training datasets repo size etc.

C. *Train Data and Test Data*

Real world dataset are divided into two different kinds of dataset. Train dataset consists of 80% of the datasets. While the test data consist of 20% of data. The classifiers learns from the 80% of the dataset and predicts the output from the 20% of data.

D. *Data Parsing*

The program loops through each of the cases in the dataset. It looks for each classes and its metrics. Its major work is to parse the whole dataset into more informative form to the data scaling phase.

E. *Data Scaling*

In this phase each dataset metrics are normalized using the below formula.

$$ScaleDown_{i,j} = (0.9 * \frac{(data[i][j] - extrema[j][0])}{extrema[j][1] - extrema[j][0]} + 0.5$$

F. *Neural Network Training*

NN training is applied over the normalized dataset. The algorithm step by step is presented below:

```
1. Initialize the Neural network weights.
2. Loop
     a. For each training metrics.
     b. Training based on classes.
     c. Find fault for each test cases of metrics
        and square mean error for each metrics.
     d. Update the weights for each layer.
3. End Loop
4. End Algorithm
```

G. *ACO Algorithm*

An ACO algorithm can be utilized for ideal way identification and is a probabilistic strategy for investigating charts as indicated by some very much characterized objective. Such an approach has a place with basic testing (white box) which tends to the accompanying sorts of testing: information stream testing, control stream/scope testing, essential way testing, and circle testing. Information stream testing centers around the focuses at which factors get values (task administrator or info information) and the focuses at which these qualities are utilized (or referenced). By information stream testing can be recognized despicable utilization of information esteems (information stream peculiarities) because of coding mistakes.

VI. RESULT AND DISCUSSION

In this section we present the experimental results performed using ACO framework.

Table I. Shows the parameter used.

Parameter Name	Value
Input Layer	7
Output Layer	1
Training Cases	210
Test Cases	20
Repo Size	20

TABLE II. Experiment Parameters for ACO

Parameter Name	Value
q	0.80
Epsilon (pheromone evaporation rate)	0.70
Max Iterations	100000
Error Criteria	0.4

ANT dataset is consider for analysis. The ANT consists of 210 classes with various metrics. The experimental results outperforms the existing vanilla based framework by reducing the number of iterations. The test error rate is 9% which is much lower than the existing approach.

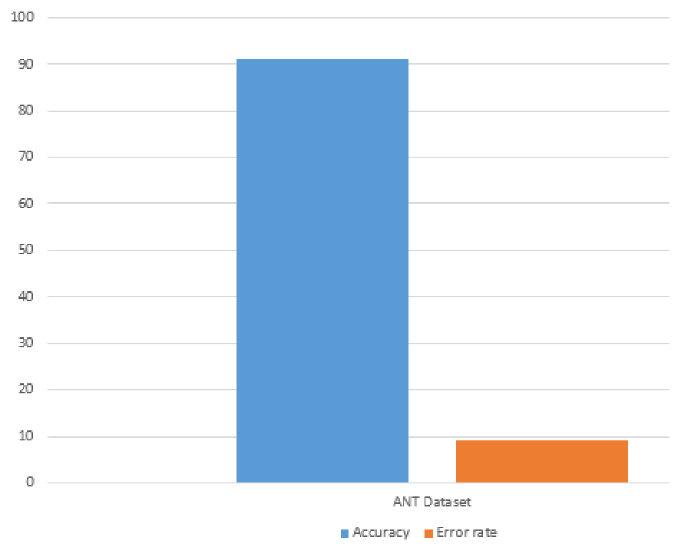

Fig. 3. Shows the output of ANT dataset experiment

VII. CONCLUSION

We propose a novel mechanism in combination of Neural Network and ACO Algorithm. The Neural Network are used for training the input matrices of the classes of the dataset while ACO are used for finding the optimal solution from the results shown by the NN.

We gain 91% of accuracy with just 9% of error rate while experimenting with ANT database. Proposed mechanism simply outperforms the existing Neural Network based approach which just uses NN training.

REFERENCES

[1] F. B. E. Abreu and R. Carapuca, “Object-Oriented software engineering: Measuring and controlling the development process,” in Proceedings of the 4th International Conference on Software Quality, vol. 186, 1994.

[2] J. Bansiya and C. G. Davis, “A hierarchical model for Object-Oriented design quality assessment,” ACM Transactions on Programming Languages and Systems. vol. 128, pp. 4–17, August 2002.

[3] B. K. Kang and J. M. Bieman, “Cohesion and reuse in an Object-Oriented system,” in Proceedings of the ACM SIGSOFT Symposium on software reuseability, pp. 259–262, Seattle, March 1995.

[4] L. C. Briand, J. W¨ ust, J. W. Daly, and D. V. Porter, “Exploring the relationships between design measures and software quality in Object-Oriented systems,” The Journal of Systems and Software, vol. 51, pp. 245–273, May 2000.

[5] L. Etzkorn, J. Bansiya, and C. Davis, “Design and code complexity metrics for Object-Oriented classes,” Object-Oriented Programming, vol. 12, no. 10, pp. 35–40, 1999.

[6] M. Halstead, Elements of Software Sciencel. New York, USA: Elsevier Science, 1977.

[7] B. Henderson-Sellers, Software Metrics. UK: Prentice-Hall, 1996.

[8] W. Li and S. Henry, “Maintenance metrics for the Object-Oriented paradigm,” in Proceedings of First International Software Metrics Symposium, pp. 52–60, 1993.

[9] T. J. McCabe, “A complexity measure,” IEEE Transactions on Software Engineering, vol. 2, pp. 308–320, December 1976.

[10] D. P. Tegarden, S. D. Sheetz, and D. E. Monarchi, “A software complexity model of Object-Oriented systems,” Decision Support Systems, vol. 13, no. 3, pp. 241–262, 1995.

[11] S. Y. Lee, D. Li and Y. Li, "An Investigation of Essential Topics on Software Fault-Proneness Prediction," 2016 International Symposium on System and Software Reliability (ISSSR), Shanghai, 2016, pp. 37-46.

[12] E. Arisholm, L. C. Briand, and M. Fuglerud, “Data Mining Techniques for Building Fault-Proneness Models in Telecom Java Software,” in Proceedings of the 18th IEEE International Symposium on Software Reliability Engineering, Trollhättan, Sweden, pp. 215-224, November 2007

[13] B. Russo, “A Proposed Method to Evaluate and Compare Fault Predictions Across Studies,” in Proceedings of the 10th International Conference on Predictive Models in Software Engineering, Turin, Italy, pp. 2-11, September 2014

[14] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings,” IEEE Transactions on Software Engineering, vol. 34, no. 4, pp. 485-496, August 2008

[15] D. Radjenovic, M. Heri_cko, R. Torkar, and A. Zivkovi_c, Software fault prediction metrics: A systematic literature review," Information and Software Technology, vol. 55, no. 8, pp. 1397-1418, 2013.

[16] M. Shepperd, Q. Song, Z. Sun, and C. Mair, “Data quality: some comments on the nasa software defect datasets,” Software Engineering, IEEE Transactions on, vol. 39, no. 9, pp. 1208 -1215, 2013.

[17] E. Erturk and E. A. Sezer, “A comparison of some soft computing methods for software fault prediction,” Expert Systems with Applications, 2014.

[18] M. Shepperd, D. Bowes, and T. Hall, “Researcher bias: The use of machine learning in software defect prediction,” Software Engineering, IEEE Transactions on, vol. 40, no. 6, pp. 603-616, 2014.

[19] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, “Software defect prediction using static code metrics underestimates defect-proneness,” in Neural Networks (IJCNN), The 2010 International Joint Conference on, pp. 1-7, IEEE, 2010.

[20] Naidu, M. Surendra, and N. GEETHANJALI. “Classification of Defects in Software using Decision Tree Algorithm.” International Journal of Engineering Science and Technology (IJEST) 5.06 (2013).”, Printice Hall india,2014.