International Journal of Scientific Research & Engineering Trends
Volume 5, Issue 3, May-Jun-2019, ISSN (Online): 2395-566X

Efficient Artificial Neural Network Based Approach for Software Fault Identification

Karandeep Kaur
Dept. of Computer Science Engg.
Ramgarhia Institute of Engineering & Technology
Punjab, Phagwara, India

Abstract- **Software Fault prediction is designed to predict error prone software modules by using some of the underlying attributes of a software project. It is usually performed training a prediction model using project attributes added to the failure information of a known item and then using the prediction model to predict the failure of the unknown item. In this work, a adaptive-neuro-fuzzy system based algorithm was created to effectively address this issue and assess the execution of neural utilizing software fault dataset at different parameter settings utilizing different execution estimation methods. The examination used data assembled from the software fault database of programming bug data. The results show that the Adaptive Neural Networks strategies regards to anticipating programming weakness tendency can be used to identify bugs effectively.**

Keywords- Software Engineering, Fault prediction, Fuzzy Reasoning, ANFIS (adaptive neuro fuzzy inference system).

I. INTRODUCTION

In the present day it is observed that in many software organizations emphasis is laid on reducing the development cost [1],effort [2], time consumed for development, and produce-reliable software [3] by increasing the software quality. Due to the presence of large line of code constituting to a huge number of modules in a program, has lead to increase in complexity. This lead to the difficulty in producing reliable software without faults. The other obvious reason for failing to produce reliable software is due to the lack of proper testing activities and time [4].

This sort of problem can be better handled by predicting certain quality attributes such as fault proneness, maintenance effort, and the testing effort during the early stages of software design. To achieve these objectives, sufficient testing of the software product needs to be carried out. Also exhaustive testing is not possible because it leads to more testing cost to be incurred, and can be very time consuming due to the large size of the product. Thus, it is very much essential to recognize the classes which are often quite fault prone [5]. There are many approaches to identify such as fault prone classes and software metrics are one such indicator.

The fault prone models predicted using these software metrics can be used in early stages of SDLC. This will benefit the developers to emphasize on reducing the utilization of testing resources on the predicted faulty classes only. Hence, this will significantly benefit in saving time and resources during the development of a software. Choosing proper metrics [6] is basic for execution change of machine learning models. For instance, line of code (LOC) of a module is a code highlight. Static deformity investigation is another way to deal with discovers surrenders in the code to guarantee programming quality. Programming deformity expectation has still been one of the most smoking themes in the product designing zone. On one hand programming imperfection expectation strategies comes in two flavors: static and dynamic, contingent upon whether code execution is required. Static imperfection forecast principally uses code highlights to foresee deserts. Dynamic imperfection forecast predicts abandons in view of the dissemination of deformities in various programming life-cycle stages. Then again, programming imperfection forecast has two research objectives: deformity thickness expectation and imperfection inclined module expectation.

II. RELATED WORKS

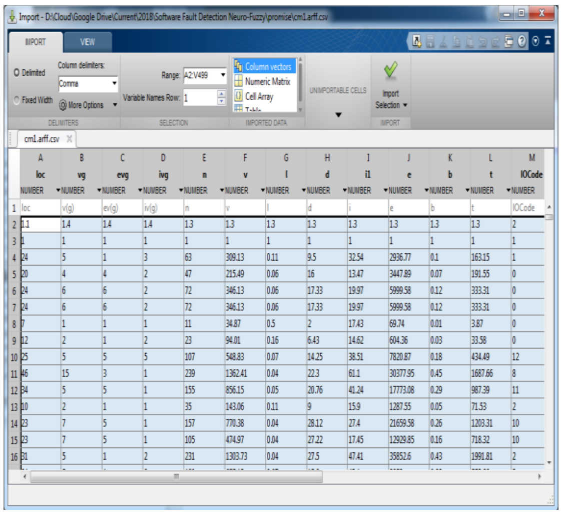
Nowadays, with the overwhelming of Objected Oriented (OO) programming [8], certain essential plot thoughts, for instance, heritage, coupling, and association have been battled to on a very basic level impact versatile quality. Those diagram features have been entangled in reducing the understandability of challenge arranged tasks, thusly raising diverse quality. Machine learning systems are science and architects, keen machines, particularly savvy PC programs. These strategies have the capacity of PC, programming and firmware to do those things that we, as people, perceive as clever conduct. Strategies in view of Machine Learning [9] have ended up being perfect for expectation models as saw in writing.

ML procedures cover extensive variety of points, for example, neural systems, Transformative Algorithm [10], Swarm knowledge, bacterial scavenging Algorithm [11] Fuzzy frameworks [12], and Artificial Immune frameworks (AIF) [13]. The main ideas behind this work into improve the execution of programming blemish need models. Our technique is using Neural Systems with Back propagation neural structure with Levenber-marquardt Algorithm [15] minimizing mean error using static features of software bug metric data set. The Machine-learning procedures are utilized to discover the defect, blame, powerlessness, and horrible stench to achieve quality, sensibility, and reusability in programming. Programming issue gauge strategies are utilized to predict programming issues by utilizing quantifiable systems.

Regardless, Machine-learning frameworks are in like way immense in perceiving programming insufficiency, [16] demonstrated a study of programming imperfection check utilizing machine-learning frameworks to predict the event of deficiencies also demonstrated the standard methods. It goes for delineating the issue of blame propensity.

III. METHODOLOGY

For implementation we have used the PROMISE [17] dataset made publicly for repeatable and verifiable predictive models in software engineering especially software fault identification. There are various metrics are available of which we have used CM1 and JM1 for fault identification. The CM1 is a NASA spacecraft instrumentation project written in "C language". figure below gives the import module used in MATLAB of CM1 project.



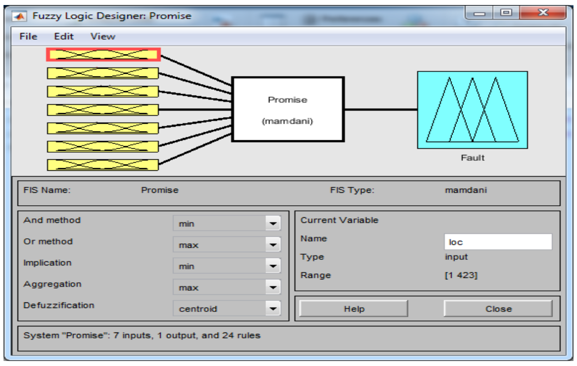
Column delimiters: Comma
Range: A2:V499
Variable Names Row: 1
Column vectors
Numeric Matrix
Cell Array
Table

cm1.arff.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M
	loc	vg	evg	ivg	n	v	l	d	i1	e	b	t	IOCode
	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER
1	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b	t	IOCode
2	1.1	1.4	1.4	1.4	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	2
3	1	1	1	1	1	1	1	1	1	1	1	1	1
4	24	5	1	3	63	309.13	0.11	9.5	32.54	2936.77	0.1	163.15	1
5	20	4	4	2	47	215.49	0.06	16	13.47	3447.89	0.07	191.55	0
6	24	6	6	2	72	346.13	0.06	17.33	19.97	5999.58	0.12	333.31	0
7	24	6	6	2	72	346.13	0.06	17.33	19.97	5999.58	0.12	333.31	0
8	7	1	1	1	11	34.87	0.5	2	17.43	69.74	0.01	3.87	0
9	12	2	1	2	23	94.01	0.16	6.43	14.62	604.36	0.03	33.58	0
10	25	5	5	5	107	548.83	0.07	14.25	38.51	7820.87	0.18	434.49	12
11	46	15	3	1	239	1362.41	0.04	22.3	61.1	30377.95	0.45	1687.66	8
12	34	5	5	1	155	856.15	0.05	20.76	41.24	17773.08	0.29	987.39	11
13	10	2	1	1	35	143.06	0.11	9	15.9	1287.55	0.05	71.53	2
14	23	7	5	1	157	770.38	0.04	28.12	27.4	21659.58	0.26	1203.31	10
15	23	7	5	1	105	474.97	0.04	27.22	17.45	12929.85	0.16	718.32	10
16	31	5	1	2	231	1303.73	0.04	27.5	47.41	35852.6	0.43	1991.81	2

Figure 1 importing CM1 software metrics in MATLAB [18] for analysis.

Also a subset of complete table is shown below which contains various fault metrics such as Line of code (LOC), v(g) being the McCabe cyclomatic complexity, etc with last column as identified defects [19]. The goal of this project is to establish the relationship between Object-Oriented metrics and fault proneness at the class level. In this study, a fault is used as a dependent variable and each of the CK metric is an independent variable. Its intended to develop a function between fault of a class and CK metrics suite (WMC, NOC, DIT, RFC, CBO, LCOM) [20][21]. Fault is a function of WMC, NOC, DIT,RFC, CBO and LCOM and can be represented as shown in the following equation



FIS Name: Promise
FIS Type: mamdani
And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzification: centroid
Current Variable
Name: loc
Type: input
Range: [1 423]
System "Promise": 7 inputs, 1 output, and 24 rules

Figure 2 Fuzzy Logic Designer for Software fault detection

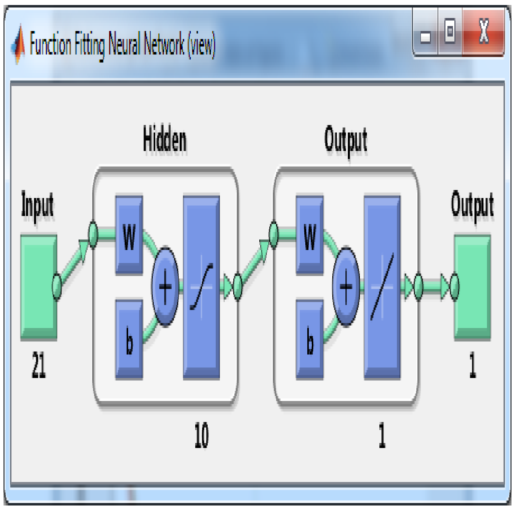


Figure 3 Neural Network Pattern Recognition Using Feed Forward Levenberg Marquardt Algorithms showing 10 hidden layers and 1 output layers.

The Proposed SVM is compared using various comparison metrics such as Mean squared error, Accuracy and execution time is evaluated in this section.

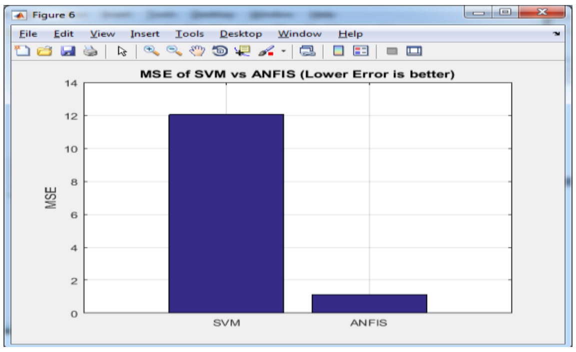


Figure 4 MSE of SVM compared with ANFIS showing that ANFIS has lower error <2% as compared to the SVM which has about 12%.

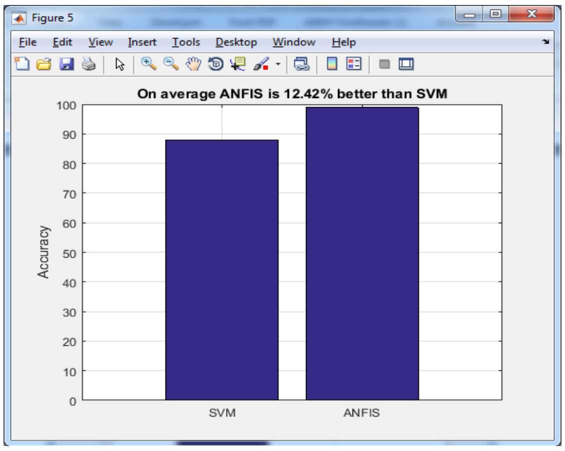


Figure 5 Accuracy of ANFIS compared with SVM Accuracy is defined as Accuracy $= 100 - \text{MSE}$.

As shown in figure on average ANFIS is 12.42% better than SVM in case of MSE and about 21.62% better than SVM in case of execution time on average as shown in figure below.

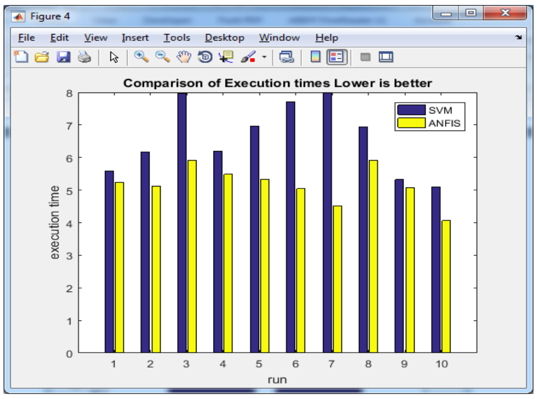


Figure 6 Execution time of SVM compared with ANFIS lower values indicate faster algorithm.

Also when we consider mean execution time SVM on average took 6.60s and Mean Execution time of ANFIS was 5.17s On average ANFIS is 21.62% better than SVM.

IV. CONCLUSION

This work was about using Neural Network strategies for software fault identification. The results show that the Neural Network strategies with regards to can be used to identify bugs effectively. Ensuing to having engaging outcomes, we are sure that a customized by programming issue acknowledgment and course of action structure can be delivered to help the specialists by giving second ends and disturbing them to cases that require advance thought. In future, one can use other training algorithms to increase the accuracy level for predicting the software defects. Increase the use of models which are based on machine learning techniques. Machine learning models have better features than other approaches. Using class level metrics, conduct more studies on fault prediction models. Increase the usage of public datasets for software fault prediction problem. As discussed the ANFIS is provides 12.42% better accuracy than SVM in case of MSE and about 21.62% better than SVM in case of execution time on average. For the future work we would like to extend our work in to following domains Devising an efficient method for predicting exact number of faults also Devising techniques that can help identify faults during testing.

REFERENCES

1. Huang, Jianglin, Yan-Fu Li, and Min Xie. "An empirical analysis of data preprocessing for machine learning-based software cost estimation." Information and software Technology 67 (2015): 108-127.
2. Jorgensen, M. (2014). What we do and don't know about software development effort estimation. IEEE software, 31(2), 37-40.
3. Garcia-Valls, M., Bellavista, P., & Gokhale, A. (2017). Reliable software technologies and communication middleware: A perspective and evolution directions for cyber-physical systems, mobility, and cloud computing.
4. Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis. arXiv preprint arXiv:1709.08439.
5. Weyuker, E., & Ostrand, T. (2016). Identifying fault-prone files in large industrial software systems. In Perspectives on Data Science for Software Engineering (pp. 103-106).
6. Fenton, N., & Bieman, J. (2014). Software metrics: a rigorous and practical approach. CRC press.
7. Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. Information and Software Technology, 58, 388-402.

8. Kochar, B., Gaur, S. S., & Bhardwaj, D. K. (2017). Identification, Analysis & Empirical Validation (IAV) of Object Oriented Design (OO) Metrics as Quality Indicators. International Journal on Recent and Innovation Trends in Computing and Communication, 5(8), 31-40.
9. Robert, C. (2014). Machine learning, a probabilistic perspective.
10. Jeyaraj, A. (2018). Transformative learning in designing algorithms for reporting information systems. Education and Information Technologies, 1-19.
11. Liu, J., Song, B., & Li, Y. (2018, May). An optimum dispatching for photovoltaic-thermal mutual-complementing power plant based on the improved particle swarm knowledge algorithm. In 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA) (pp. 1062-1067). IEEE.
12. Wang, H., Jing, X., & Niu, B. (2017). A discrete bacterial algorithm for feature selection in classification of microarray gene expression cancer data. Knowledge-Based Systems, 126, 8-19.
13. Marín, N., Ruiz, M. D., & Sánchez, D. (2016). Fuzzy frameworks for mining data associations: fuzzy association rules and beyond. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 6(2), 50-69.
14. Saurabh, P., & Verma, B. (2016). An efficient proactive artificial immune system based anomaly detection and prevention system. Expert Systems with Applications, 60, 311-320.
15. Ranganathan, A. (2015). The levenberg-marquardt algorithm (2004).
16. Kalsoom, A., Maqsood, M., Ghazanfar, M. A., Aadil, F., & Rho, S. (2018). A dimensionality reduction-based efficient software fault prediction using Fisher linear discriminant analysis (FLDA). The Journal of Supercomputing, 74(9), 4568-4602.
17. Catal, C. (2011). Software fault prediction: A literature review and current trends. Expert systems with applications, 38(4), 4626-4636.
18. Chapman, S. J. (2015). MATLAB programming for engineers. Nelson Education.
19. Card, D., Tan, C., & Smith, N. A. (2017). A Neural Framework for Generalized Topic Models. arXiv preprint arXiv:1705.09296.
20. Vashisht, V., Lal, M., Suresh chandar, G. S., & Kamya, S. (2015). A framework for software defect prediction using neural networks. Journal of Software Engineering and Applications, 8(8), 384.
21. Gayathri, M., & Sudha, A. (2014). Software defect prediction system using multilayer perceptron neural network with data mining. International Journal of Recent Technology and Engineering, 3(2), 54-59.