# Software Fault Prediction using Artificial Neural Networks

Prajwal D Prakash
Dept. of CS&E
BMSIT&M
Bangalore
Email: prajwalpy6@gmail.com

Gaurav Vattikuti
Dept. of CS&E
BMSIT&M
Bangalore
Email: vattikutigaurav@gmail.com

Yeresime Suresh
Dept. of CS&E
BMSIT&M
Bangalore
Email: ysuresh@bmsit.in

***Abstract*—Software faults are dangerous. Software systems are often essential to a business operation or organization and failures in such systems cause disruption of some goal directed activity (mission critical). Malfunctions in other safety-critical systems may result in death, loss of property or environmental harm. Most damaging software malfunctions are run-time malfunctions that are not always caught during the testing process. In such cases, predicting software faults before they occur, gives designers an insight to the possible malfunctions and the relative probability of their occurrence. This helps in focused testing and saves time during the software development life cycle. The artificial neural network is a powerful tool for the detection of faults due to its ability to differentiate between various patterns. This paper presents the design and development of artificial neural network based model for software fault detection in context of fault-prone modules.**

***Index Terms*—software fault detection; artificial neural networks;**

## I. Introduction

There has been a tremendous growth in the demand for software quality during recent years. As a consequence, issues related to testing are becoming increasingly crucial. The ability to measure software fault-proneness can be extremely important for minimizing cost and improving the overall effectiveness of the testing process. The majority of faults in a software system are found in a few of its com- ponents (Boehm and Papaccio, 1988; Porter and Selby, 1990). The “80:20” rule states that approximately 20% of a software system is responsible for 80% of its errors, costs, and rework (Boehm, 1987). Boehm and Papaccio (1988) conclude:

> “The major implication of this distribution is that software verification and validation activities should focus on identifying and eliminating the specific high-risk problems to be encountered by a software project, rather than spreading their available early-problem-elimination effort uniformly across trivial and severe problems.”

Therefore, the possibility of estimating the potential faultiness of software components at an early stage can be of great help for planning testing activities. For example, testing effort can be focused on a subset of software components (e.g., the potentially “most troublesome” 20%). Unfortunately, software fault-proneness cannot be directly measured. However, it can be estimated based on software metrics, which provide quantitative descriptions of program attributes.

## II. Literature Survey

Table I shows the relevant literature survey.

## III. Methodology

The field of machine learning focuses on the study of algorithms that improve their performance at some task automatically through experience (Mitchell, 1997). An artificial neural network (ANN) is an efficient information processing system which resembles in characteristics with a biological neural network. ANNs possess large number of highly interconnected processing elements called neurons. Each neuron is connected with the other by a connection link. Each connection link is associated with weights which contain information about the input signal. This information is used by the neuron net to solve a particular problem. Each neuron has an internal state of its own. This internal state is called the activation level of neuron, which is the function of the inputs the neuron receives. There are a number of activation functions that can be applied over net input such as Gaussian, Linear, Sigmoid and Tanh. It is the Sigmoid function that is the most frequently used in neural nets. Thus, the models of ANN are specified by the three basic entities namely:

1) The models synaptic interconnections;
2) The training or learning rules adopted for updating and adjusting the connection weights;
3) Their activation functions.

The neural network process starts by developing the structure of the network and establishing the technique used to train the network using an existing data set. Neural network architectures are divided into two groups:

1) Feed forward networks where no loops in the network path occur.
2) Feedback networks that have recursive loops.

The most common architecture of neural networks which is used in software cost estimation is the Back-Propagation trained Feed Forward networks. The training algorithm of back propagation involves four stages:

1) Initialization of weights

TABLE I
LITERATURE SURVEY

| **Paper Title** | **Authors** | **Year of Publication** | **Method Proposed** | **Data-set Used** | **Results Obtained** |
|---|---|---|---|---|---|
| Comparative analysis of neural network and genetic programming for number of software faults prediction [3] | Santosh Singh Rathore, Sandeep Kuamr | 2015 | Generic method better than neural networks. | Camel 1.4 | 143.47% completeness |
| | | | | Camel 1.2 | 69.10% completeness |
| Applying swarm ensemble clustering technique for fault prediction using software metrics [4] | Rodrigo A. Coelho, Fabrcio dos R. N. Guimares, Ahmed A. A. Esmin | 2014 | K-means with the Euclidean similarity function (KM-E) | CM1 (KM-E) | 83.94% accuracy |
| | | | K-means with the Manhattan similarity function (KM-M) | CM1 (EM) | 81.33% accuracy |
| | | | PSO clustering with the Euclidean similarity function (PSO-E) | CM1 (EM) | 70.29% accuracy |
| | | | PSO clustering with Manhattan similarity function (PSO-M) | CM1 (PSO-E) | 89.96% accuracy |
| | | | Expectationmaximization clustering algorithm (EM) | CM1 (PSO-M) | 71.29% accuracy |
| Supplementing Object-Oriented software change impact analysis with fault-proneness prediction [5] | Bassey Isong, Ohaeri Ifeoma, Munienge Mbodila | 2016 | NASA KC1, 3 sets of predictors [m1,m2,m3] | M1 | 97% training accuracy |
| | | | | M2 | 90% training accuracy |
| | | | | M3 | 99%,training accuracy |
| A model for early prediction of faults in software systems [6] | Parvinder S. Sandhu, Amanpreet S. Brar, Raman Goel, Jagdeep Kaur, Sanyam Anand | 2010 | Decision tree based Model in combination of K-means clustering as pre-processing technique | CM1 | 100% on 10-fold cross validation |
| Artificial neural network-based metric selection for software fault-prone prediction model [7] | C. Jin, S.-W. Jin, J.-M. Ye | 2012 | DT | PC1 | 94.14% accuracy |
| | | | SFPM | CM1 | 91.17% accuracy |
| | | | SFPM | KC1 | 87.74% accuracy |
| | | | LR | KC3 | 93.42% accuracy |

2) Feed forward
3) Back Propagation of errors
4) Updating of the weights and biases

## IV. DATA-SET AND METRICS

In this section, we present a brief description of the software fault dataset, software metrics and evaluation metrics used for analysis.

### A. *Data-set*

The camel-1.6 data-set is used for the experimental evaluation of our model, which has been collected from the PROMISE data repository[1]. The data-set is an analysis of 92 versions of 38 proprietary, open-source and academic projects. The data-set contains a record of twenty object-oriented metrics and the number of faults found in the software system during testing and after the release.

The camel-1.6 data-set has 965 total number of modules, with 188 faulty modules, each with a given number of faults. In our setup, we only classify whether a module is faulty. Therefore, the number of faults in the module is not taken into account.

### B. *Used Metrics*

The camel-1.6 data-set uses the Chidamber & Kemerer object-oriented metrics. These Metrics were designed to measure the complexity of an object-oriented modelled design. Though the data-set uses twenty metrics, we use only the metrics the suite originally consisted of. These 6 metrics are calculated for each class: WMC (Weighted Methods Per Class), DIT (maximum inheritance path from the class to the root class), NOC (number of immediate sub-classes of a class), CBO (number of classes to which a class is coupled), RFC (Response for a Class) and LCOM1 (Lack of Cohesion of Methods).

### C. *Metrics for evaluation of classifiers*

The experimental evaluation of such a model is performed according to several performance metrics, such as probability of detection (PD), probability of false alarm (PF), balance, or area under the ROC (Receiver Operating Characteristics) curve.

The following set of evaluation measures are being used to find the results:

1) Probability of Detection (PD), also called recall or specificity, is the probability of correct classification of a module that contains a fault.

$$PD = recall = \frac{TP}{TP + FN} \quad (1)$$

2) Probability of False Alarms (PF) is the ratio of false positives to all non defect modules.

| | NO (Prediction) | YES (Prediction) |
|---|---|---|
| NO (Actual) | TN (True Negative) | FP (False Positive) |
| YES (Actual) | FN (False Negative) | TP (True Positive) |

TABLE II
CONFUSION TABLE

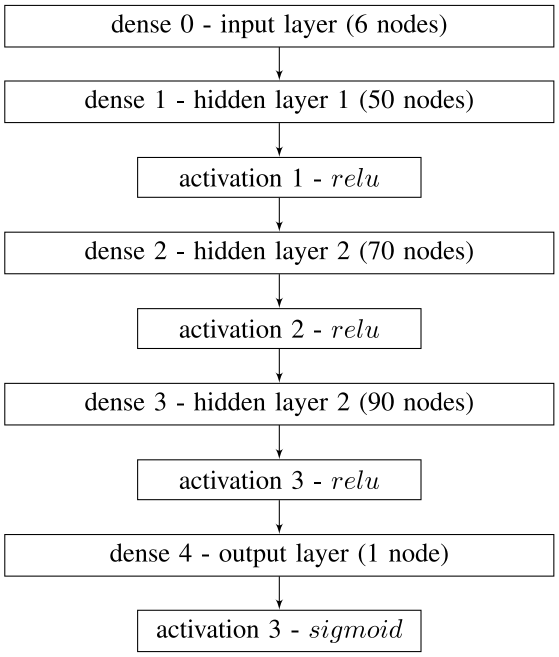
dense 0 - input layer (6 nodes)
dense 1 - hidden layer 1 (50 nodes)
activation 1 - *relu*
dense 2 - hidden layer 2 (70 nodes)
activation 2 - *relu*
dense 3 - hidden layer 2 (90 nodes)
activation 3 - *relu*
dense 4 - output layer (1 node)
activation 3 - *sigmoid*

Fig. 1. The architecture of the ANN

$$PF = \frac{FP}{FP + TN} \tag{2}$$

3) The accuracy of the model is the probablity of the correct classification of a module.

$$Accuracy = \frac{TN + TP}{TN + FP + FN + TP} \tag{3}$$

4) The precision is the ability of the classifier not to label as positive a sample that is negative.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

5) The F-measure can be interpreted as a weighted harmonic mean of the precision and recall.

$$F - measure = \frac{2 * precision * recall}{precision + recall} \tag{5}$$

6) ROC(Receiver Operator Characteristic) is a plot of PD versus PF across all possible experimental combinations.

In Table II, a confusion matrix is calculated after N*M cross-validation.

## V. IMPLEMENTATION AND RESULTS

The neural network is implemented using Keras[2] running on top of Theano. The Keras Sequential model is set up as shown in figure 1.

| | NO (Prediction) | YES (Prediction) |
|---|---|---|
| NO (Actual) | 77 (True Negative) | 3 (False Positive) |
| YES (Actual) | 12 (False Negative) | 4 (True Positive) |

TABLE III
RESULTS

It was concluded that the architecture (in figure 1) performed the best for the camel-1.6 dataset. The neural network has three hidden layers. The input layer has 6 nodes, one for each CK Metric class. The first hidden layer has 50 nodes, the second hidden layer has 70 nodes and the third hidden layer has 90 nodes. The implementation is a binary classification. The activation function used is $ReLU$, i.e. rectified linear unit function for both the hidden layers and $sigmoid$, i.e the exponential function for the output layer. The $ReLU$ activation function's output ranges from -1 to 1 and is better suited for the normalized input from the normalization layer that has a mean close to 0, and a standard deviation close to 1. The $ReLU$ function is also known to be faster than $tanh$, i.e. the hyperbolic tangent function [8]. The $sigmoid$ function is suited for a binary classification like the one in our setup, because it produces output probabilities in the range [0, 1]. The optimizer used was $Adagrad$ and the loss considered was $binarycrossentropy$.

### A. Results

The result obtained on 10-fold cross validation is shown in III.

The accuracy obtained on the above fold was 84.36% with a good AUC of 0.8. The ROC Curve is shown in figure 2.

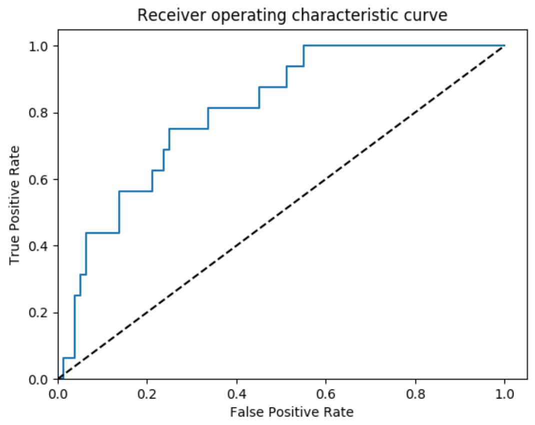

Fig. 2. Receiver Operator Characteristic Curve

## VI. CONCLUSION

The conclusion goes here. this is more of the conclusion

ACKNOWLEDGMENT

## REFERENCES

[1] Menzies, T., Krishna, R., Pryor, D. (2016). The Promise Repository of Empirical Software Engineering Data; *http://openscience.us/repo* . North Carolina State University, Department of Computer Science

[2] Keras, a high-level neural networks library, written in Python; *https://keras.io/*

[3] Rathore, Santosh Singh, and Sandeep Kuamr. ”Comparative analysis of neural network and genetic programming for number of software faults prediction.” Recent Advances in Electronics & Computer Engineering (RAECE), 2015 National Conference on. IEEE, 2015.

[4] Coelho, Rodrigo A., Fabrcio dos RN Guimares, and Ahmed AA Esmin. ”Applying swarm ensemble clustering technique for fault prediction using software metrics.” Machine Learning and Applications (ICMLA), 2014 13th International Conference on. IEEE, 2014.

[5] Isong, Bassey, Ohaeri Ifeoma, and Munienge Mbodila. ”Supplementing Object-Oriented software change impact analysis with fault-proneness prediction.” Computer and Information Science (ICIS), 2016 IEEE/ACIS 15th International Conference on. IEEE, 2016.

[6] Sandhu, Parvinder S., et al. ”A model for early prediction of faults in software systems.” Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on. Vol. 4. IEEE, 2010.

[7] Jin, Cong, S-W. Jin, and J-M. Ye. ”Artificial neural network-based metric selection for software fault-prone prediction model.” IET software 6.6 (2012): 479-487.

[8] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks http://www.cs.toronto.edu/ fritz/absps/imagenet.pdf