International Journal of Marketing and Technology
(ISSN: 2249-1058)

CONTENTS

Editorial Advisory Board

Associate Editors

Title

APPLICATION OF SOFT COMPUTING TECHNIQUES TO PREDICTION OF FAULTY CLASSES IN OBJECT ORIENTED SOFTWARE

Author(s)

Divya Jain

Assistant Professor (CSE/IT),

HMR Institute of Technology and Management, Delhi

Vibhor Sharma

Assistant Professor,

Lovely Professional University, Phagwara, Punjab

Abstract:

Estimating number of defects or predicting fault-proneness in object oriented software modules plays a key role in quality control of software products. Over the last few years, software quality has become one of the most important requirements in the development of systems. Fault-proneness of a software module is the probability that the module contains faults. The objective of this paper is to analyze experimentally the object oriented metrics as predictors of fault-prone classes and, therefore, determine whether they can be used as early quality indicators. This early detection of fault-prone software components enables verification experts to concentrate their time and resources on the problem areas of the software system under development. In this paper we describe how we calculated the object oriented metrics given by Chidamber and Kemerer to illustrate how fault-proneness detection can be carried out. Empirical validation of software metrics to predict quality using machine learning methods is important to ensure their practical relevance in the software organizations. The aim of this research work is to establish a method for identifying software defects using machine learning methods. In this work we used NASA's Metrics Data Program (MDP) as software metrics data. The repository at NASA IV & V Facility MDP contains software metric data and error data at the function/method level. In this paper we introduce Generalized Regression Neural Networks, fuzzy subtractive clustering and Adaptive-Neuro Fuzzy Inference System (ANFIS) for predicting number of defects using Object Oriented metrics.

Keywords- Object Oriented Software metrics, Neural Networks, fuzzy logic, fuzzy inference, Subtractive Clustering, Software quality, Fault count, Adaptive Neuro-Fuzzy Inference System

Introduction:

Developing a large software system is both time consuming and resource consuming activity. Thus there is a need to provide accurate information to software engineers to help them make decisions, schedule and plan different activities, and allocate required resources for the different software activities that take place during entire software life-cycle.

Software metrics thus provide necessary information that can be used in many ways to make assessments of the software products and the development processes, to help engineers in identifying where the resources are needed and project managers in decision making [1].

Metrics data provides quick feedback to software engineers. Various properties of the final software such as complexity, design quality, resource requirement, cost etc. can be predicted by analyzing the collected data. This early feedback enables software designers and developers to correct the inadequacies in their design or implementation without too much effort. Also if properly used, it can lead to a significant reduction in costs of the overall implementation and improvements in quality of the final software product. The improved quality, in turn, reduces future maintenance efforts.

During the software testing phase, software metrics are particularly useful. With the metrics data, various software quality attributes can be calculated to optimize resource allocation for testing. Testing a large system is both a time consuming and resource consuming activity. Applying equal testing and verification efforts to all parts of a software system has become cost-prohibitive. Therefore, one needs to be able to identify fault-prone modules so that testing/verification effort can be concentrated on these classes [2]. In this paper we describe an experimental validation of the Object oriented metric suite defined in Chidamber & Kemerer, 1994, with respect to their ability to identify fault-prone classes. In this paper we make use of the public domain data set KC1 from the NASA Metrics Data Program [5].

The data in KC1 was collected from a storage management system for receiving/processing ground data, which was implemented in the C++ programming language. This system consists of 145 classes that comprise of 2107 methods, with 40K lines of code. KC1 provides both class-level and method-level static metrics.

The software metric data gives us the values for specific variables to measure a specific module/function or the whole software. When combined with the weighted error/defect data, this data set becomes the input for a machine learning system. A learning system is defined as a system that is said to learn from experience with respect to some class of tasks and performance measure, such that its performance at these tasks improves with experience. To design a learning system, the data set in this work is divided into two parts: the training data set and the testing data set. Some predictor functions are defined and trained with respect to Multi-Layer Perceptron and Decision Tree algorithms and the results are evaluated with the testing data set.

The paper is organized as follows: Section 2 describes the metrics studied, and also sources from which data is collected. Section 3 presents the overview of our proposed model for defect prediction method. In Section 4 we have listed the results of the experiments and a detailed evaluation of the machine learning algorithms is also given in the same section. Conclusions of the research are presented in Section 5.

Metrics suite:

Software metrics are often divided into two categories: software product metrics and software process metrics [1]. Software product metrics are used to measure aspects of software products, such as source code, specifications, software design, etc. Software process metrics are used to measure software development processes, including development effort, staffing levels, etc. Software product metrics can be further divided into two categories: those measuring dynamic attributes and those measuring static attributes. Dynamic metrics can only be evaluated at run-time and are thus difficult to measure. These metrics are not widely used, except those measuring code coverage in software testing. Static metrics, in turn, are used to measure static attributes of software and they are widely used in software engineering.

A. Dependent and independent Variables

Our goal of this study was to analyze empirically the OO design metrics proposed in [6] for the purpose of evaluating whether or not these metrics are useful for predicting the probability of detecting faulty classes. Assuming testing was performed properly and thoroughly, the probability of fault detection in a class during acceptance testing should be a good indicator of its probability of containing a fault and, therefore, a relevant measure of fault-proneness. The binary dependent variable in our study is fault proneness. The goal of our study is to explore empirically the relationship between OO metrics and fault proneness. Fault proneness is defined as the probability of fault detection in a class [7]. We use machine learning methods to predict the probability of fault proneness. Dependent variable [3, 8] will be predicted based on the faults found during software development life cycl**e.** The target metrics (dependent variables) are the

"Error Count" and "Defects". "Error Count" refers to the number of module errors associated with the corresponding set of predictor software metrics, while "Defect" metric refers to whether the module is fault prone or fault-free.

B. C K Metrics Suite

The metrics used in our study is metrics defined by Chidamber and Kemerer. The metrics are as follows:

1) Weighted Methods per Class (WMC): WMC measures the complexity of an individual class. Based on [Chidamber & Kemerer, 1994], if we consider all methods of a class to be equally complex, then WMC is simply the number of methods defined in each class. Thus, WMC is defined as being the number of all member functions and operators defined in each class. However, "friend" operators (C++ specific construct) are not counted. Member functions and operators inherited from the ancestors of a class are also not counted. The assumption behind this metric is that a class with significantly more member functions than its peers is more complex, and by consequence tends to be more fault-prone[2,9].

2) Depth of Inheritance Tree of a class (DIT): DIT is defined as the maximum depth of the inheritance graph of each class. C++ allows multiple inheritances. DIT, measures the number of ancestors of a class. The assumption behind this metric is that well-designed OO systems are those structured as forests of classes, rather than as one very large inheritance lattice. In other words, a class located deeper in a class inheritance lattice is supposed to be more fault-prone because the class inherits a large number of definitions from its ancestors [2, 7, 9].

3) Number of Children of a Class (NOC): This is the number of direct descendants for each class. Classes with large number of children are difficult to modify and usually require more testing because the class potentially affects all of its children. Thus, a class with numerous children has to provide services in a larger number of contexts and must be more flexible. We expect this to introduce more complexity into the class design [2, 7, 9].

4) Coupling Between Object classes (CBO): A class is coupled to another one if it uses its member functions and/or instance variables. CBO provides the number of classes to which a given class is coupled. The assumption behind this metric is that highly coupled classes are more

fault-prone than weakly coupled classes. So coupling between classes should be identified in order to concentrate testing and/or inspections on such classes [2].

5) Response for a Class (RFC): This is the number of methods that can potentially be executed in response to a message received by an object of that class. In our study, RFC is the number of functions directly invoked by member functions or operators of a class. The assumption here is that the larger the response set of a class, the higher the complexity of the class, and the more fault-prone and difficult to modify [2].

6) Lack of Cohesion on Methods (LCOM): This is the number of pairs of member functions without shared instance variables, minus the number of pairs of member functions with shared instance variables. However, the metric is set to 0 whenever the above subtraction is negative. A class with low cohesion among its methods suggests an inappropriate design, (i.e., the encapsulation of unrelated program objects and member functions that should not be together), which is likely to be fault-prone[2].

Methodology:

A. Generalized Regression Neural Network

Radial basis networks can require more neurons than standard Feedforward backpropagation networks, but often they can be designed in a fraction of the time it takes to train standard Feedforward networks. They work best when many training vectors are available. A radial basis function network is a neural network approached by viewing the design as a curve-fitting (approximation) problem in a high dimensional space. Learning is equivalent to finding a multidimensional function that provides a best fit to the training data, with the criterion for "best fit" being measured in some statistical sense. In a neural network, the hidden units form a set of "functions" that compose a random "basis" for the input patterns (vectors). These functions are called radial basis functions. Generalized regression neural networks (grnns)[12] are a kind of radial basis network that is often used for function approximation. grnns can be designed very quickly. newgrnn creates a two-layer network. The first layer has radbas neurons, and calculates weighted inputs with dist and net input with netprod. The second layer has purelin neurons, calculates weighted input with normprod, and net inputs with netsum. Only the first layer has biases.

B. Fuzzy Subtractive Clustering

In this paper, a fuzzy technique based on subtractive clustering [11] is applied. This method automatically generates fuzzy inference rules by clustering the training data. Each cluster center represents a fuzzy inference rule. A Gaussian membership function is designed for each variable. When only one cluster center is obtained through the training process, the method becomes equivalent to multiple linear regression. Subtractive clustering, is a fast, one-pass algorithm for estimating the number of clusters and the cluster centers in a set of data. The cluster estimates, which are obtained from the subclust function, can be used to initialize iterative optimization-based clustering methods (fcm) and model identification methods (like anfis).The subclust function finds the clusters by using the subtractive clustering method. The genfis2 function builds upon the subclust function to provide a fast, one-pass method to take input-output training data and generate a Sugeno-type fuzzy inference system that models the data behavior.

C. Adaptive Neuro Fuzzy Inference

The basic idea behind this neuro-adaptive learning technique is to compute the membership function parameters that best allow the associated fuzzy inference system to track the given input output data. This learning works similar to that of neural networks.Using a given input/output data set, function anfis constructs a fuzzy inference system (FIS) whose membership function parameters are tuned (adjusted) using either a backpropagation algorithm alone or in combination with a least squares type of method. This adjustment allows fuzzy systems to learn from the data they are modeling.

The knowledge base is developed from the training data set by training the system to a number of epochs to minimize the error rate. The fuzzy inference process involves the following steps.

1) Fuzzify inputs. Determine the degree to which inputs belong to each of the appropriate fuzzy sets via membership functions.

2) Apply fuzzy operator. If the antecedent of a given rule has more than one part, apply relevant fuzzy operators to obtain one membership value that represents the match to the antecedent for that rule.
3) Apply implication method. Shape the consequent based on the antecedent.
4) Defuzzify. The input for the defuzzification process is a fuzzy set and the output is a single number. In Sugeno inference, the centroid is simply the weighted average of a few data points

Analysis results:

The fuzzy inference system is implemented using various MATLAB Toolbox .

The Experimental setup was as follows:

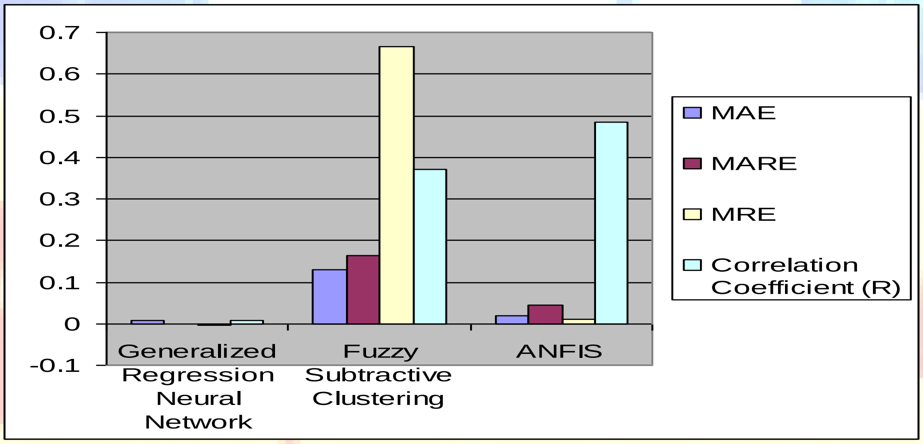
1) The input metrics were normalized using Z score normalization technique using the function prestd preprocesses the network training set by normalizing the inputs and targets so that they have means of zero and standard deviations of 1.
2) Principal Component Analysis of the data was performed using singular value decomposition technique using the function prepca. Prepca preprocesses the network input training set by applying a principal component analysis. This analysis transforms the input data so that the elements of the input vector set will be uncorrelated.
3) The transformed data set was divided into training set and test set which was input to neural network, fuzzy inference system and ANFIS.

The results are obtained using various measures of performance evaluation.

TABLE I

VALIDATION RESULTS OF NUMBER OF DEFECT PREDICTION ACCURACY

	Generalized Regression Neural Network	Fuzzy Subtractive Clustering	ANFIS
MAE	0.0073	0.1311	0.0205
MARE	0.00	0.165	0.046
MRE	-0.0032	0.666	0.0104
Correlation Coefficient (R)	0.0086	0.372	0.4839



Conclusion:

This paper proposed a comparative study of three different soft computing techniques Artificial Neural Networks, Fuzzy Subtractive Clustering and ANFIS to prediction of number of faults in object oriented software. From the results presented above it is found that Generalized Regression Artificial Neural Network has a better generalization capability in comparison with other soft computing techniques used.

References:

- P.Yu, T.Systa, and H.Muller,"Predicting Fault-Proneness using OO Metrics An Industrial Case Study," Proceedings of the Sixth European Conference on Software Maintenance and Reengineering (CSMR.2002), pp. 99-107, 2002 IEEE.
- V.Basili, L.C.Bria nd, and W.L.Melo. "A validation of object-oriented design metrics as quality indicators," IEEE Transactions Software Engineering, 22(10):751–761, 1996.
- Lionel C. Briand , Walcelio L. Melo , Jurgen Wust "Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects" IEEE Transactions On Software Engineering, Vol. 28, No. 7, July 2002.
- R.Subramanyam and M.S. Krishnan," Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," IEEE Transactions On Software Engineering, Vol. 29, No. 4, April 2003.
- www.mdp.ivv.nasa.gov, NASA Metrics data Repository.
- S. R. Chidamber and C. F. Kemerer (1994). "A metrics suite for object-oriented design ". IEEE Trans. on Software Engineering, 20(6):476–493.
- Lionel C. Briand , Jurgen Wust , John W. Daly , D. Victor Porter ," Exploring the relationships between design measures and software quality in object-oriented systems", The Journal of Systems and Software 51 (2000) 245-273.
- Venkata U.B. Challagulla, Raymond A. Paul, "Empirical Assessment of Machine Learning based Software Defect Prediction Techniques" Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'05) .
- Daniela Glasberg, Khaled El Emam, Walcelio Melo, and Nazim Madhavji," Validating Object-Oriented Design Metrics on a Commercial Java Application",National Research Council of Canada, NRC 44146.
- http://www.mathworks.com.
- Chiu S.L. "Extracting Fuzzy Rules from Data for Function approximation and pattern Classification," Chapter 9 in Fuzzy Information Engineering: A Guided Tour of Applications John Wiley & Sons, 1997.
- Miyoung Shin, Amrit Goel, "Empirical Data Modeling in Software Engineering Using Radial Basis Functions", IEEE Transactions on Software Engineering, Vol 26 June 2006.