Application of Neural Networks for Estimating Software Maintainability Using Object-Oriented Metrics

Mie Mie Thet Thwin, Tong-Seng Quah
School of Electronic & Electrical Engineering
Nanyang Technological University
miethet@pmail.ntu.edu.sg

Abstract

This paper presents the application of neural networks in software maintainability estimation using object-oriented metrics. Maintenance effort can be measured as the number of lines changed per class. In this paper, the number of lines changed per class (modification volume) is predicted using Ward neural network and General Regression neural network (GRNN). Object-oriented design metrics concerning with inheritance related measures, complexity measures, cohesion measures, coupling measures and size measures are applied in this study. Principal components, which are derived from these object-oriented metrics, are used as independent variables.

1. Introduction

Many object-oriented metrics have been proposed over the last decade. Prediction models using object-oriented design metrics can be used for obtaining assurances about software quality. In practice, quality estimation means either estimating reliability or maintainability. Reliability is typically measured as the number of defects. These can be pre-release or post-release. The estimated number of defects can also be normalized by a size measure to obtain a defect density estimate. Maintainability is typically measured as change effort. Change effort can mean either the average effort to make a change to a class, or the total effort spent on changing a class.

Khoshgoftarr et al. introduced the use of the neural networks as a tool for predicting software quality. In [19], they presented a discriminant model and a neural network model of the large telecommunications system, classifying modules as not fault-prone or fault-prone. They compared the neural-network model with a nonparametric disciminant model, and found the neural network model had better predictive accuracy.

We conduct our study in the object-oriented paradigm. However since the object-oriented paradigm exhibits different characteristics from the procedural paradigm, different software metrics have to be defined and used.

Our neural network models aim to predict object oriented software maintainability by estimating the number of lines changed per class. We used software metrics concerning with inheritance related measures, cohesion measures, coupling measures, complexity measures and size measure.

We also introduce using Ward neural network and General Regression neural network to improve prediction result for estimating software maintainability. Ward neural network is a backpropagation network with different activation functions. They are applied to hidden layer slabs to detect different features in a pattern processed through a network to lead to better prediction. We use a Gaussian function in one hidden slab to detect feature in the mid-range of the data and a Gaussian complement in another hidden slab to detect features for the upper and lower extremes of the data. Thus, the output layer will get different "views of the data". Combining the two feature sets in the output layer leads to a better prediction.

Another architecture that we have chosen is the General Regression Neural Network (GRNN). Specht [18] has stated that it is a memory-based network that provide estimates of continuous variables and converges to the underlying (linear or nonlinear) regression surface. This is a one-pass learning algorithm with a highly parallel structure. Even with sparse data is a multidimensional measurement space; the algorithm provides smooth transitions from one observed value to another.

2. Related work

There is great interest in the use of object-oriented approach in software engineering. With the increasing use of object-oriented methods in new software development there is a growing need to both document and improve current practices in object-oriented design and development.

Many measures have been proposed in the literature to capture the quality of object-oriented (OO) code and design and used for detecting fault-proneness of classes

[3, 4, 6, 8, 15]. Many investigations using statistical methods had been made to predict software quality.

Emanm and Melo [4] have constructed a model to predict which classes in a future release of a commercial Java application will be faulty. The model was then validated on a subsequent release of the same application. Their results indicated that the prediction model had a high accuracy.

Fioravanti and Nesi have extracted over 200 different object-oriented metrics to identify a suitable model for detecting fault-proneness of classes [8]. They came to the conclusion that only few of them were relevant for identifying fault-prone classes.

A set of object-oriented metrics in terms of their usefulness in predicting fault-proneness, an important software quality indicator is empirically validated in [17]. Their validation is carried out using two data analysis techniques: regression analysis and discriminant analysis.

L. Briand et al., the relationships between existing object-oriented coupling, cohesion, and inheritance measures and the probability of fault detection in system classes during testing explored empirically. Their univariate analysis have shown that many coupling and inheritance measures are strongly related to the probability of fault detection in a class. Their multivariate analysis results showed that by using some of the coupling and inheritance measures, very accurate models could be derived to predict in which classes most of the faults actually lie [10].

Wei Li and Sallie Henry [16] have done the statistical analyses of a prediction model incorporating ten object-oriented metrics. Their result has shown that there is a strong relationship between metrics and maintenance effort in object-oriented systems.

Marcela Genero et al.[9] have presented a set of metrics for measuring the structural complexity of UML class diagrams and to use them for predicting their maintainability that will heavily be correlated with object-oriented information system maintainability.

Most of these prediction models are built using statistical models. Neural networks have seen an explosion of interest over the years, and are being successfully applied across a range of problem domains, in areas as diverse as finance, medicine, engineering, geology and physics. Indeed, anywhere that there are problems of prediction, classification or control, neural networks are being introduced. Neural network can be used as a predictive model because it is very sophisticated modeling techniques capable of modeling complex functions.

In [2], Khoshgoftaar et al presented a case study of real-time avionics software to predict the testability of each module from static measurements of source code. They found that neural network is a promising technique for building predictive models, because they are able to model nonlinear relationships.

Our neural network models aim to predict object oriented software maintainability by estimating the number of lines changed per class. We also introduce using Ward neural network and General Regression neural network to improve prediction result for estimating software maintainability.

3. Design of the study

3.1. Object-oriented metrics

As discussed in section I, we are introducing the research on maintenance efforts predictions into object oriented paradigm using neural networks. As such object oriented metrics have to be selected and used in our study. To predict the maintenance effort, the following software metrics are used:

Depth of Inheritance Tree (DIT) of a class is the length of the longest path from the class to the root in the inheritance hierarchy. This determines the complexity of a class based on its ancestors, since a class with many ancestors is likely to inherit much of the complexity of its ancestors. The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit making it more complex to predict its behavior. This has direct relationship to maintainability.

Number of Children (NOC) measures the number of immediate descendants of a particular class. This measures an amount of potential reuse of the class. The more reuse a class might have, the more complex it may be, and the more classes are directly affected by changes in it implementation. This increases the magnitude of ripple effects.

Message Passing Coupling (MPC) gives an indication of how many message are passed among objects of the classes. The number of messages sent out from a class indicates how dependent the implementation of the local methods is on the methods in other classes.

Response For a Class (RFC) is the number of methods that can potentially be executed in response to a message received by an object of that class. The response set of a class consists of the set of M methods of the class, and the set of methods directly or indirectly invoked by methods in M.

Lack of Cohesion in Methods (LCOM) is the number of pairs of methods in the class using no attributes in common, minus the number of pairs of methods that do. If this difference is negative, LCOM is set to zero.

Data Abstraction Coupling (DAC) is the number of attributes in a class that have as their type another class.

Weighted Methods per Class (WMC) is the summation of McCabe's cyclomatic complexity of each local method. The more control flows a class's methods have,

the harder it is to understand them, thus, the harder it is to maintain them. A method with a low cyclomatic complexity is generally better.

The number of local methods (NOM) defined in a class indicates the operation property of a class. The more methods a class has, the more complex will be the class's interface.

SIZE1 is calculated by counting the number of executable statements (measured by number of semicolons) in a class.

SIZE2 is the total number of attributes and methods of a class.

3.2. Neural network modeling

The first neural network architecture that we have chosen is the Ward Network[20]. It is a Backpropagation network that has three slabs in the hidden layer. Hidden layers in neural network are known as feature detectors. A slab is a group of neurons. When each slab in the hidden layer has a different activation function, it offers three ways of viewing the data. We use linear function to the output slab. We use hyperbolic tangent (tanh) function is used in one slab of hidden layer because it is better for continuous valued outputs especially if the linear function is used on the output layer. Gaussian function is used in another slab of the hidden layer. This function is unique, because unlike the others, it is not an increasing function. It is the classic bell shaped curve, which maps high values into low ones, and maps mid-range values into high ones. Gaussian Complement is used in the third slab of the hidden layer to bring out meaningful characteristics in the extremes of the data. The learning rate and momentum are set to 0.1 and initial weight is set to 0.3 in this study.

Another neural network architecture that we have chosen is the General Regression Neural Network (GRNN). GRNN is based on a one-pass learning algorithm with a highly parallel structure. GRNN is a powerful memory based network that could estimates continuous variables and converges to the underlying regression surface. The strength of GRNN is that it is able to deal with sparse data effectively. Specht [18] claims that the algorithm in GRNN is able to provide a smooth transition from one observed value to another, even with sparse data in a multidimensional measurement space. GRNN applications are able to produce continuous valued outputs. For GRNN networks, the number of neurons in the hidden layer is usually the number of patterns in the training set because each pattern in the training set is represented by on neurons. The primary advantage to the GRNN is the speed at which the network can be trained. Training a GRNN is performed in one pass. The smoothing factor allows the GRNN to interpolate between the patterns or spectra in the training set.

3.3. Principal component analysis

If a group of variables in a data set are strongly correlated, these variables are likely to measure the same underlying dimension (i.e., class property) of the object to be measured. Many object-oriented metrics have high correlation with each other. For example, the number of local method (NOM) is strongly correlated with class size. The confounding effect of class size is studied in [7]. Principal component analysis (PCA) is a standard technique to identify the underlying, orthogonal dimensions that explain relations between the variables in a data set. Principal components (PCs) are linear combinations of the standardized independent variables. It is also a data reduction technique. The varimax rotation method was adopted in this study. It is an orthogonal rotation method that minimizes the number of variable that have high loadings on each factor. It simplifies the interpretation of the factors. We have selected the PCs only PCs whose eigenvalue is larger than 1.0.

4. Prediction of maintenance effort

To predict the maintenance effort two commercial software products QUES(Quality Evaluation System) data and UIMS(User Interface System) data are used in this investigation. These data are presented in [16]. The maintenance effort is measured by using the number of lines changed per class. A line change could be an addition or a deletion. A change of the content of a line is counted as a deletion followed by an addition. This measurement is used in this study to estimate the maintainability of the object-oriented systems. In this study, DIT, MPC, RFC, LCOM, DAC, WMC, NOM, SIZE1 and SIZE2 are used in QUES system and DIT, NOC, MPC, RFC, LCOM, DAC, WMC, NOM, SIZE1 and SIZE2 are used in UIMS to produce principal components. Both system were designed and developed with Class-Ada. First, each data pattern was examined for erroneous entries, outliers, blank entries and redundancy. After standardizing the metric data, we performed the principal component analysis. Table 1 and Table 2 present the relationship between the original object-oriented metrics and the domain metrics for these systems.

In QUES system, PCA identified three PCs, which capture 89% of the data set variance. Table 1 shows for each rotated component the coefficients of the measure, with coefficients larger than 0.6 set in boldface. The eigen value, the percentage of the data set variance each PC describes, and the cumulative variance percentage are also provided. Based on the analysis of the coefficients associated with each metrics within each of the three rotated components, the PCs are interpreted as follows:

The first component is highly correlated with NOM, SIZE2, RFC, LCOM, WMC, SIZE1 and DAC. NOM is a better representative, however, because it is less correlated with the other two components. The second component is most highly correlated with MPC. The third component is most highly correlated with DIT. This suggests that NOM, MPC and DIT metrics should be focused on further analysis for QUES system.

In UIMS system, PCA also identified three PCs, which capture 84% of the data set variance as shown in Table 2. The first component is highly correlated with SIZE2, WMC, RFC, LCOM, NOM, SIZE1 and MPC. SIZE2 is a better representative. The second component is highly correlated with DIT and NOC. The third component is most highly correlated with DAC.

We sorted the data according to the number of changes values and divided data into training, testing, and production sets using 3:1:1 ratio. Test set is used to prevent over training network so they will generalize well. We used the production data set to evaluate model performance. It can be tested the network's results with the data the network has never seen before.

We used Ward network and GRNN network for predicting number of changes. Table 3 and Table 4 show the summary of Ward network design. In our General Regression neural network design, there were three neurons in input layer and 1 neuron in output layer for both systems. We used 71 hidden layered neurons in QUES system and 39 hidden layered neurons in UIMS system.

Table 1. Rotated principle components for QUES system

Metrics	PC1	PC2	PC3
DIT	0.060	0.027	**0.966**
MPC	-0.023	**0.966**	0.037
RFC	**0.877**	0.333	0.043
LCOM	**0.869**	-0.156	0.059
DAC	**0.796**	0.027	0.427
WMC	**0.832**	0.258	-0.27
NOM	**0.971**	-0.132	0.097
SIZE1	**0.812**	0.475	-0.089
SIZE2	**0.963**	-0.093	0.190
Eigenvalues	5.384	1.388	1.248
% Variance	59.826	15.424	13.863
Cummulative % Variance	59.826	75.250	89.113

4.1. Goodness of fit test

To measure the goodness of fit of the model, we use the coefficient of multiple determination (R-square), the coefficient of correlation(r), r-square, mean square error, mean absolute error, minimum absolute error and maximum absolute error. These statistical measures are shown in Table 5 and Table 6. The correlation of the predicted change and the observed change is represented by the coefficient of correlation (r). An r value of 0.747 in Ward neural network and 0.8590 in GRNN network represents high correlations for cross-validation for QUES system.

Table 2. Rotated principle components for UIMS system

metrics	PC1	PC2	PC3
DIT	-0.14615	**-0.86673**	-0.01882
NOC	0.094873	**0.788682**	0.077122
MPC	**0.606777**	-0.28106	0.541778
RFC	**0.941184**	0.106288	0.23338
LCOM	**0.881172**	0.06539	-0.00822
DAC	0.13152	0.154133	**0.94183**
WMC	**0.945789**	0.11411	0.073862
NOM	**0.876694**	0.267267	0.274151
SIZE1	**0.803835**	0.323392	0.368195
SIZE2	**0.960316**	0.064614	0.122886
Eigenvalues	5.309595	1.684767	1.472685
% Variance	53.09595	16.84767	14.72685
Cummulative % Variance	53.09595	69.94361	84.67047

Table 3. Ward neural network summary for QUES system

	Slab1	Slab2	Slab3	Slab4	Slab5
No. of neurons	3	3	3	3	1

Table 4. Ward neural network summary for UIMS system

	Slab1	Slab2	Slab3	Slab4	Slab5
No. of neurons	3	2	2	2	1

For UIMS system, r value is 0.7798 in Ward neural network and 0.6984 in GRNN network. The significance level of a cross-validation is indicated by an p value. A commonly accepted p value is 0.05. An two tailed probability p values of 0.000 in both cross-validation shows a high degree of confidence for the successful validations. We conclude that the impact of model prediction is valid in the population.

5. Conclusion

This empirical study presents the prediction maintenance effort using two neural network models. From the results presented above, object-oriented metrics chosen in this study appear to be useful in predicting

software maintainability. Network models are found to be useful to predict modification volume.

Table 5. Experimental result for QUES system

	Ward	GRNN
R-square	0.5545	0.7220
r (correlation coefficient)	0.747	0.8590
r- square	0.558	0.7379
Mean square error	817.004	509.790
Mean absolute error	20.782	12.182
Min absolute error	0.094	0
Max absolute error	114.161	109.385
t values	9.329047	13.98484
p values	0.000	0.000

Table 6. Experimental result for UIMS system

	Ward	GRNN
R-square	0.5444	0.3547
r (correlation coefficient)	0.7798	0.6984
r- square	0.6081	0.4877
Mean square error	2294.174	3249.564
Mean absolute error	31.908	29.259
Min absolute error	1.567	0.300
Max absolute error	204.179	266.249
t values	7.576655	5.934987
p values	0.000	0.000

Our future research direction aims to estimate the software readiness using neural network models. To estimate the readiness, three factors will be considered in our future study: (1) how many faults are remaining in the programs (2) how many changes are required to correct the errors and (3) how much time is required in changing the programs.

6. References

[1] El Emam, ”A primer on object-oriented measurement”, Proc. Seventh International Software Metrics Symposium, 2001, pp. 185 –187.

[2] T. M. Khoshgoftaar, E.B. Allen, Z. Xu, “Predicting testability of program modules using a neural network”, Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, 2000, pp. 57-62.

[3] L.C. Briand, J.W. Daly, J.K. Wust, “A unified framework for coupling measurement in object-oriented systems”, IEEE Transactions on Software Engineering, 1999, pp. 91-121.

[4] El Emam, W. Melo, C.M. Javam, “The Prediction of Faulty Classes Using Object-Oriented Design Metrics”, Journal of Systems and Software, Elsevier Science, 2001, pp. 63-75.

[5] Mei-Huei Tang, Ming-Hung Kao, Mei-Hwa Chen,”An empirical study on object-oriented metrics”, Proceedings of the Sixth IEEE International Symposium on Software Metrics, 1999, pp. 242-249.

[6] A. Mounir Boukadoum, Houari A. Sahraoui and Hakim Lounis, “Machine Learning Approach to Predict Software Evolvability using fuzzy binary trees”, International Conference on Artificial Intelligence, 2001.

[7] El Emam, K., Benlarbi, S., Goel, N. and Rai, S.N., “The confounding effect of class size on the validity of object-oriented metrics”, IEEE Transactions on Software Engineering, vol. 27, pp. 630-650, 2001.

[8] F. Fioravanti, P. Nesi, “A study on fault-proneness detection of object-oriented systems”, Fifth European Conference on Software Maintenance and Reengineering, 2001, pp. 121 –130, 2001.

[9] M. Genero, J. Olivas, M. Piattini, F. Romero, “"Using metrics to predict OO information systems maintainability", Proceedings. of the 13th International Conference Advanced Information Systems Engineering, Interlaken, Switzerland, 2001.

[10] L. Briand, J. Wüst, John W. Daly and V. Porter, "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems", Journal of Systems and Software, 51(2000) p 245-273.

[11] M. Cartwright, and M. Shepperd, "An Empirical Investigation of Object Oriented Software System", IEEE Transactions on Software Engineering, vol. 26, pp. 786-796, 2000.

[12] N.E. Fenton and N. Ohlsson, ”Quantitative analysis of faults and failures in a complex software system”, IEEE Transactions on Software Engineering, vol. 26, pp. 797-814, 2000.

[13] D. Glasberg, K. El Emam, W. Melo, and N. Madhavji, "Validating Object-oriented Design Metrics on a Commercial Java Application". Technical Report, NRC/ERB-1080, NRC 44146, 2000.

[14] Todd L. Graves, Alan F. Karr, J.S. Marron, and Harvey Siy, “Predicting Fault Incidence Using Software Change History”, IEEE Transactions on Software Engineering, vol. 26, 2000.

[15] L.C. Briand, W.L Melo, J. Wust, “Assessing the applicability of fault-proneness models across object-oriented software projects”, IEEE Transactions on Software Engineering, vol. 28 pp. 706 –720, 2002.

[16] Wei Li and S. Henry, “Object-Oriented Metrics that Predict Maintainability”, Journal of Systems and Software, 1993, pp. 111-122, 1993.

[17] Ping Yu, T. Systa, and H. Muller, “Predicting fault-proneness using OO metrics. An industrial case study”, Proceedings. of 6th European Conference on Software Maintenance and Reengineering, 2002, pp. 99 –107, 2002.

[18] D.F, Specht, “A general regression neural network”.IEEE Transactions on Neural Networks, vol. 2, Issue: 6, pp. 568-576, 1991.

[19] T.M. Khoshgoftaar, E.B. Allen, J.P. Hudepohl, and S.J. Aud, “Application of neural networks to software quality modeling of a very large telecommunications system”, IEEE Transactions on Neural Network, vol. 8, pp. 902-909, 1997.

[20] NeuroShell 2 Help, Ward Systms Group, Inc. http://www.wardsystems.com.