OBJECT ORIENTED SOFTWARE QUALITY PREDICTION USING GENERAL REGRESSION NEURAL NETWORKS

S.Kanmani[1], V. Rhymend Uthariaraj[2], V. Sankaranarayanan[2] and P. Thambidurai[1]
[1] Department of Computer science and Engineering & Information Technology,
Pondicherry Engineering College Pondicherry – 605 014, India
n_kanmani@hotmail.com
[2] Ramanujam Computing Centre, Anna University, Chennai – 605 025, India

Abstract

This paper discusses the application of General Regression Neural Network (GRNN) for predicting the software quality attribute – fault ratio. This study is carried out using static Object-Oriented (OO) measures (64 in total) as the independent variables and fault ratio as the dependent variable. Software metrics used include those concerning inheritance, size, cohesion and coupling. Prediction models are designed using 15 possible combinations of the four categories of the measures. We also tested the goodness of fit of the neural network model with the standard parameters. Our study is conducted in an academic institution with the software developed by students of Undergraduate/Graduate courses.

Keywords

Object oriented, metrics, neural network, software quality

1. Introduction

Many Object Oriented measures have been. Prediction models using OO designs can be used for obtaining assurance about the software quality. In practice, quality estimation concentrates on reliability. Reliability of the modules can be measured by fault ratio. Khoshgoftaar et al [5] conducted a study for the modules in procedural environment and found that a neural network model had better predictive accuracy. They used Multi-layer back propagation feed forward neural networks. Quah and Thewin [3]&[4] used Ward neural network (a special type of multi-layer back propagation feed forward neural network) with different activation functions and GRNN for OO paradigm. Their results show that GRNN performance is better. We adopt this neural network model for the prediction of fault ratio, which takes a range value between 0 and 1. The distribution of the value is highly skewed towards 0, which is very difficult for any standard statistical technique to predict.

2. Related work

A variety of statistical techniques [11] are used in software quality modeling. Models are often based on statistical relationship between the measure of quality and measures of software attributes (software metrics). However relationships between static software metrics and quality factors are often complex and nonlinear, limiting the accuracy of conventional approaches. Artificial neural networks are adopted at modeling non-linear functional relationships that are difficult to model with other techniques and thus are attractive for software quality modeling. Few applications of artificial neural networks to software quality have appeared in the literature [1-9]. Khoshgoftaar [9] introduced neural networks for predicting software development faults. They used 14 structured (raw) metrics as the input variables to the neural network and total number of faults found in the programs as the output variable. They found that neural network model perform better prediction than the statistical model.

The applications of principal component analysis are explored [1]&[8] to neural network modeling as a way of improving the predictive quality of neural network models. The results show that the predictive quality is improved and also the training time of the neural network is very much reduced. Neural network model for predicting the testability of program modules [6] also found to be successful. Neural network models are used in the prediction of fault prone modules of highly reliable very large telecommunication systems [5]. The misclassification rate (overall 26.2% only) in this model is found to be very less compared to non-parametric discriminant models. Two neural networks (Ward, GRNN) are introduced by [3]&[4] for quality prediction of classes developed in OO environment using 9 measures (mainly from CK metric suite). They applied this model for predicting number of faults and number of lines of change in the classes. The results show that GRNN models work better than Ward neural network. In our proposed work the quality prediction model using GRNN is developed to predict the fault ratio of the classes in the OO software. For this case study, we use 64 measures from the broad categories of size, cohesion, coupling and inheritance.

3. System description

The software system used for this study was developed by the Undergraduate / Graduate students of the courses (B.Tech Information Technology, B.Tech Computer Science and Engineering & M.C.A – Master of Computer Applications) offered by the Department of Computer Science and Engineering at Pondicherry Engineering College. All students had experience with C and C++ programming language. The systems were developed over a period of one month during their laboratory hours (three hours per week). The students were grouped into 200 teams each with one or two students. Each team was asked to develop a departmental library Management information system (problem definition given in Appendix A) that supports the issue and return process of books to staff members and students. They developed the system in Turbo C++ environment with necessary libraries, using ASCII files for data storage.

A team of 3 experienced staff members of the department carried out testing of the systems. This group tested the systems for the 45 test cases given in Appendix B (by relating the classes responsible). The number of test cases responsible for each of the classes is identified. The fault ratio is calculated as the ratio of the number of test cases not satisfied over the total number of test cases responsible to the class.

The Object Oriented Metric Calculator (OOMC) - a measurement tool [12] developed was used to collect the values of the Object Oriented design measures from the C++ source code [13].

The various OO measures considered in the study are given in the Appendix C. Among the 64 measures taken, 7 are size measures, 18 are inheritance measures, 10 are cohesion measures and 29 are coupling measures. The measures SPD, Nmnpub, IFCMIC, ACMIC, FCMEC, DCMEC, IFMMIC and FMMEC had less than five non zero values hence are omitted from the analysis (as given in [11]). So the resulted measures are 56 in number.

Even though there are many measures in each category, each of them measure different dimension of the property. The principal component analysis led to few domain measures due to the stopping rule applied (eigen value should be greater than 1). In order to see the full influence of the measures they are grouped in 15 possible combinations (as given in Table 1) and a detailed study is carried out.

S.No	1	2	3	4	5	6	7	8	9
Measures Group*	Sz	Ch	In	Cp	Sz-Ch	Sz-Cp	Sz-In	Ch-Cp	Ch-In
No. of principal components	2	3	5	8	5	9	6	9	7
% of cum variance	66	80	80	84	80	82	76	81	80

Table 1. Measures Group and their model attributes
(* Sz – Size, Ch – Cohesion, Cp-Coupling, In-Inheritance)

S.No	10	11	12	13	14	15
Measures Group*	Cp-In	Sz-Ch-Cp	Sz-Ch-In	Ch-Cp-In	Sz-Cp-In	Sz-Cp-Ch-In
No. of principal components	11	10	9	13	13	14
% of cum variance	80	81	82	84	83	84

Table 1. Measures Group and their model attributes (contin..)
(* Sz – Size, Ch – Cohesion, Cp-Coupling, In-Inheritance)

4. Method

We used the following methodology on the sample of 1185 classes collected using the 15 groups of measures. The steps are carried out for each of the group separately.

- Perform principal component analysis on the standardized metrics to produce domain metrics.
- Prepare fit (two third of 1185) and validate data sets (one third of 1185).
- Develop GRNN models using fit data set
- Predict the fault ratio of each module in the validate data set using the model.
- Compare the predictions to actual values and perform the goodness of fit test. The following Sub-sections discuss steps 1 and 3 in detail.

4.1 Principal component analysis

Many software metrics have high correlation with each other. Principal components analysis transforms raw data into variables that are not correlated to each other. When the original data are Object-Oriented software metrics, the new principal component variables are called as domain metrics. The principal components are found to improve the results [1] & [8] within the domain. Principal components analysis is also a data reduction technique. The number of domain metrics could be the same as the number of metrics but a stopping rule chooses a few domain metrics that represent most of the variability in the data. Otherwise the model would not contribute to the predictive accuracy.

The principal components are linear combinations of m standardized metrics $Z_1,\ldots,Z_m$. The principal components represent the same data in a new rotated coordinate system where the variability is maximized in each direction and the principal components are uncorrelated. Principal components analysis performs the following calculations, given an Z(n×m) matrix of standardized metric data

- Calculate the covariance matrix $\sum$ of Z
- Calculate eigen values λ_j and eigen vectors e_j of $\sum$, j = 1,…,m.
- Reduce the dimensionality of the data. We choose the stopping rule as components with eigen value > 1.
- Calculate a standardized transformation matrix T where each column is defined as

$$t_j = \frac{e_j}{\sqrt{\lambda_j}} \text{ for } j = 1,\ldots,p. \quad (1)$$

- Calculate the domain metrics for each module, where $D_j = Z\, t_j$ and $D = Z\, T$

Since these metrics are orthogonal they are used as independent variables in neural network models. For each group, the number of domain measures and the cumulative % of variance of each of the group of measures are given in the Table 1.

4.2 General regression neural network model

GRNN is based on a one-pass learning algorithm with 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 [2] 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, 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 one neuron. The primary advantage of 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. We use 790 neurons (number of training data) in the hidden layer. Number of principal components (in the respective group) is the number of neurons in the input layer and 1 neuron in the output layer. GRNN model is developed using MATLAB simulation package [10].

4.3 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.

Coefficient of correlation (r) (Pearson's Linear correlation coefficient) is a statistical measure of the strength of the relationship between the actual versus predicted outputs. The r coefficient can range from –1 to +1. The closer r is to 1, the stronger the positive linear relationship, and closer r is to –1, the stronger the negative linear relationship. When r is near 0, there is no linear relationship.

$$r = \frac{S_{xy}}{\sqrt{S_{xy} - S_{yy}}} \tag{2}$$

$$S_{xx} = \sum x^2 - \frac{1}{n}(\sum x)^2 \tag{3}$$

$$S_{yy} = \sum y^2 - \frac{1}{n}(\sum y)^2 \tag{4}$$

$$S_{xy} = \sum xy - \frac{1}{n}(\sum x)(\sum y) \tag{5}$$

where n equals the number of patterns, x refers to the set of actual outputs and y refers to the predicted output.

Coefficient of multiple determinations is a statistical indicator. It compares the accuracy of the model to the accuracy of a trivial benchmark model wherein the prediction is just the mean of all of the samples. It can range from 0 to 1, and a perfect fit would result in an R-square value of 1, a very good fit near 1, and a very poor fit less than 0, it is calculated as follows:

$$R^2 = 1 - \frac{\sum (y - \hat{y})^2}{\sum (y - \bar{y})^2} \tag{6}$$

where y is the actual value for the dependent variable, $\hat{y}$ is the predicted vale of y and $\bar{y}$ is the mean of the y values.

5. Empirical results

The data set was randomly divided into a fit data set with 790 classes and a validate data set with 395 classes. The goodness of fit of the models is evaluated by the parameters discussed in Section 4.3. Figures 1, 2 and 3 show the comparison result arrived at the parameters r-correlation coefficient, R–coefficient of multiple determination, MSE-Mean square error, MiAE-Minimum absolute error, MxAE-Maximum Absolute error. Among the 15 groups, size-coupling-inheritance group predicts the fault ratio more accurately (with r = 0.9702 & R-sq = 0.9355). The group size-cohesion-coupling is also equivalently good (with r = 0.9702 & Rsq = 0.9355). However their individual (groups 1, 2, 3 & 4) predictions are not good (negative R values).

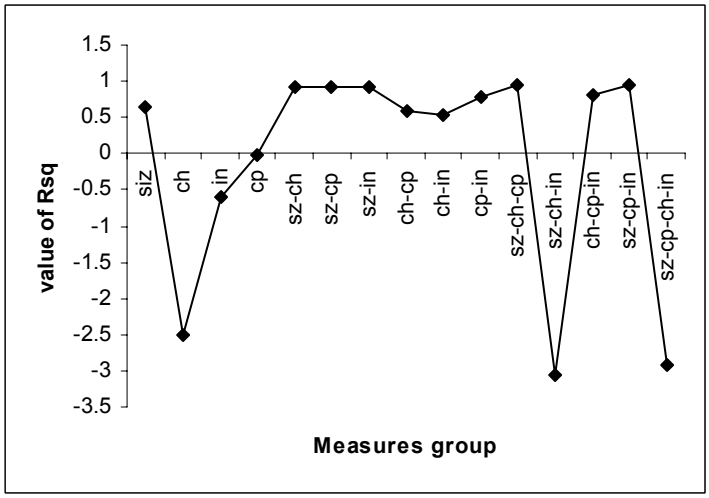

Figure 1. Rsq values for the 15 group measures

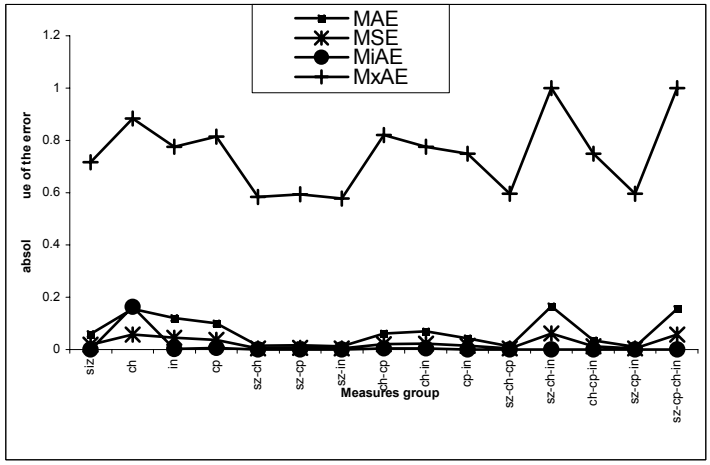

Figure 2. Results of MSE, MiAE, MxAE and MAE for the 15 group measures

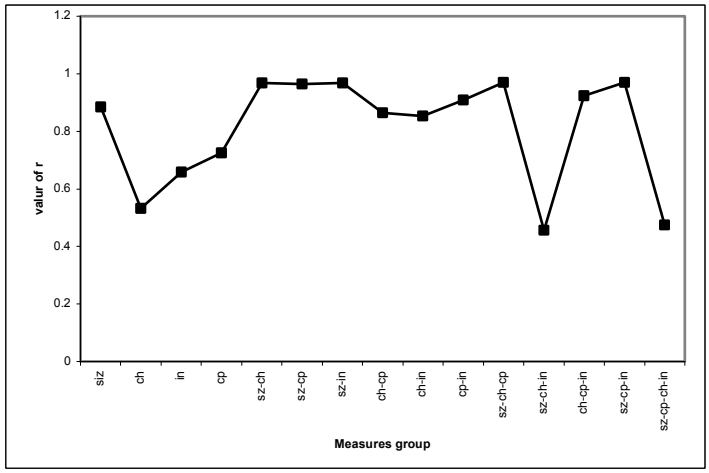

Figure 3. Correlation coefficient values for the 15 group measures

6. Conclusion

The empirical study presents the prediction of fault ratio in the Object-Oriented systems using GRNN models. From the results presented above, Object-Oriented metrics appear to be useful in predicting the fault ratio that takes the range between 0 and 1 (and the distribution is highly skewed towards 0). Among the possible groups of the measures size-cohesion-coupling and size-coupling – inheritance groups perform very well.

References

[1] Khoshgoftaar T. M., R.M.Szabo and P.J.Guasti (1995): Exploring the Behavior of Neural Network Software Quality Models, *Software Engineering Journal* May, pp. 89-96.

[2] Specht D. F (1991):, A General Regression Neural Network, *IEEE Transactions on Neural Networks*, vol. 2, no. 6, pp. 568-576.

[3] Quah T. S, and M.M.T.Thewin (2003): Application of Neural Networks for Software Quality Prediction using Object-Oriented Metrics, *Proceedings of the International Conference on Software Maintenance (ICSM'03)*, Vol 3.

[4] Thewin M. M, and T.S.Quah (2003): Application of Neural Networks for Predicting Software Development Faults using O-O Design Metrics. *Proceedings of 9th International Conference on Neural Information Processing (ICONIP 2002)*, vol5, pp 2312-2316.

[5] Khoshgoftaar T. M., E.B.Allen, J.P.Hidepohl, and S.J.Aud (1997): Application of Neural Networks to Software Quality Modeling of a very Large-Scale Telecommunications Systems, *IEEE Transactions on Neural Networks*, vol 8, no 4, July pp. 902-909.

[6] Khoshgoftaar T. M., E.B.Allen, and Z.Xu (2000): Predicting Testability of Program Modules using Neural Networks, *Proceedings of 3rd IEEE Symposium on Applied Specific Systems and Software Engineering Techniques* – March, pp. 57-62.

[7] Khoshgoftaar T. M, and R.M.Szabo (1995): Detecting Program Modules with Low Testability, *Proceedings of the International Conference on Software Maintenance*, 1995, pp. 242-250.

[8] Khoshgoftaar T. M, and R.M.Szabo (1994): Improving Neural Network Predictions of Software Quality using Principal Components Analysis, *Proceedings of IEEE International World Congress on Computational Intelligence*, pp. 3295-3300.

[9] Khoshgoftaar T. M., A.S.Pandya and H.M More (1992): A Neural Network Approach for Predicting Software Development Faults, *Proceedings of 3rd IEEE ISSRE*, pp. 83-89.

[10] Matlab Neural network Tool Box and Manual. Mathworks Inc.

[11] Briand L., J.Wust, J.W.Daly, D.V. Porter (2000): Exploring the Relationships Between Design Measures and Software Quality, *Journal of Systems and Software*, 51, pp. 245-273.

[12] Kanmani S., V.Prathiba (2001): Object Oriented Metric Calculator, *Technical Report*, Pondicherry Engineering College.

[13] Kanmani S., V.Sankaranarayanan and P.Thambidurai (2003): A Measurement Model for C++ Program Complexity Analysis, *Proceddings of the 9th International Conference EPMESC,* Macao, China, pp. 575-580.

Appendix A

The problem taken for the study is the automation of the department library maintenance of our institution.

Department library is an extension of main library of our institution intended to allow the department members (students, staff members) to access the books available. The department library must contain the text/reference books required for the courses (B.tech, MCA,…) and also the project reports submitted by the department students. Students studying the courses offered by the department are eligible to access the library. The students can only refer the textbooks but can borrow project reports. Staff members can refer/borrow the textbooks and reports (no constraint on the number).

One of the staff members is assigned as the library-in-charge who maintains all library activities. When the books (reference / text required for a course) recommended by a staff member are not available in the department library, can be requested for a transfer from main library. Book transfer is possible only when more than one copy of the book is available in the main library. Staff members can donate books to the department library. Library-in-charge can make use of the binding section available in the main library on request.

The Figure 4, usecase diagram shows the users and their responsibilities.

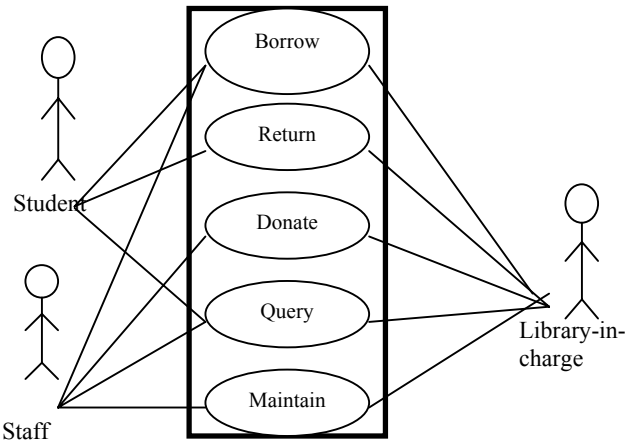

Figure 4. Use-case diagram for the defined problem

Appendix B: Test Cases

1. Borrowing a book not available in the library (ID not valid)-for staff
2. Borrowing a book already issued – for staff
3. Borrowing a book sent for binding – for staff
4. Borrowing a donated book – for staff
5. Borrowing a project book not available in the library (ID not valid)-for student
6. Borrowing project book not in the library – by staff
7. Borrowing project book already issued –by staff
8. Borrowing project book already issued –by student
9. Borrow a book by a non member student
10. Borrow a project book by a non member student
11. Borrow a book by a non member staff
12. Borrow a project book by a non member staff
13. Borrow by library-in-charge
14. Borrow a text book available in library by student
15. Borrow a project book available in library by student
16. Borrow a text book available in library by staff
17. Borrow a project book available in library by staff
18. Return of project book by student matching details
19. Return of text book by staff matching book details
20. Return of project book by staff matching details
21. Return of project book not issued – by staff
22. Return of a book not issued by staff
23. Return of a project book not issued by student (book id not matching)

24. Return of a project book issued but not taken by the given student (member id not matching)

25. Return of a project book issued but not taken by the given staff member (member id not matching)
26. Return of a text book issued but not taken by the given staff (member id not matching)
27. Return of the book in binding status by staff (book id matching) – wrong book id or status
28. Return of the book already returned by staff (book id matching) - wrong book id or status
29. Return of the project book already returned by staff (book id matching)
30. Return of the project book already returned by student (book id matching)
31. Return of a non existing book by staff
32. Return of a non existing project book by student
33. Return of an issued book by staff
34. Return of an issued project book by student
35. Return of an issued project book by staff
36. Donate a book by staff member
37. Donate a book by student member
38. Donate a book by non member staff
39. Donate a project book
40. Books recommended by valid staff member
41. List of binding books – change of status
42. Same-authored books – check for unique ID, title, publishers, etc.
43. Same titled books – check for unique ID, authors, publishers, etc.
44. Student ID duplication possibility
45. BookID duplication possibility

Appendix C

The various measures used in the study are taken from [11]. Their definitions are given below.

1 Cohesion Measures

LCOM1 : Lack of Cohesion in Methods
LCOM2 : Lack of Cohesion in Methods
LCOM3 : Lack of Cohesion in Methods
LCOM4 : Lack of Cohesion in Methods
LCOM5 : Lack of Cohesion in Methods
Co : Connectivity
Coh : Cohesion
TCC : Tight Class Cohesion
LCC : Loose Class Cohesion
ICH : Information flow based Cohesion

2 Coupling Measures

CBO : Coupling Between Object Classes
CBO' : Without Inheritance CBO
RFC : Response set For Class
RFC' : Response set For Class
MPC : Message Passing Coupling
DAC : Data Abstraction Coupling
DAC' : Data Abstraction Coupling
ICP : Information flow Based Coupling
IHICP : With Inheritance
NIHICP : Without Inheritance
IC : Inheritance Coupling

IFCAIC, ACAIC, OCAIC, FCAEC, DCAEC, OCAEC, IFCMIC, ACMIC, OCMIC, FCMEC, DCMEC, OCMEC, IFMMIC, AMMIC, OMMIC, FMMEC, DMMEC, OMMEC: These coupling measures are counts of interactions between classes. The measures distinguish the relationship between classes (friendship, inheritance, none) different types of interactions and the locus of impact of the interaction. The acronym for the measures indicates what interactions are counted: The first or first two letters indicate the relationship (A: coupling to ancestor classes, D: Descendants, F: Friend classes, IF: Inverse friends (classes that declare a given class c as their friend) O: others, i.e none of the other relationships). The next two letters indicate the type of interaction: CA: There is a Class-Attribute interaction between classes c and d if c has an attribute of type d. CM: There is a class-Method interaction between classes c and d, if c invokes a method of d, or if a method of class d is passed as parameter (function pointer) to a method of class c. The last two letters indicate the locus of impact: IC: Import coupling, the measure counts for a class c all interactions where c is using another class. EC: Export coupling count interactions where class d is the used class.

3 Inheritance Related Measures

DIT : Depth of Inheritance
AID : Average Inheritance Depth
CLD : Class to Leaf Depth
NOC : Number of Children
NOP : Number of Parents
NOD : Number of Descendants
NOA : Number of Ancestors
NMO : Number of Methods Overridden
NMI : Number of Methods Inherited
NMA : Number of Methods Added
SIX : Specialization Index
OVO : Over loading in Stand Alone classes
DPA : Dynamic Polymorphism in Ancestors
DPD : Dynamic Polymorphism in Descendants
SPA : Static Polymorphism in Ancestors
SPD : Static Polymorphism in Descendants
SP : Static Polymorphism
DP : Dynamic Polymorphism

4 Size Measures

STMTS : Number of Statements
NM : Number of Methods
NAIMP : Number of Attributes Implemented
NMPUB : Number of Public Methods
NMMPUB: Number of Non-Public Methods
NAINH : Number of Attributes Inherited
NMIMP : Number of Methods Implemented
NUMPAR : Number of Parameters