Prediction of software development faults in PL/SQL files using neural network models

Tong-Seng Quah, Mie Mie Thet Thwin

Abstract

Database application constitutes one of the largest and most important software domains in the world. Some classes or modules in such applications are responsible for database operations. Structured Query Language (SQL) is used to communicate with database middleware in these classes or modules. It can be issued interactively or embedded in a host language. This paper aims to predict the software development faults in PL/SQL files using SQL metrics. Based on actual project defect data, the SQL metrics are empirically validated by analyzing their relationship with the probability of fault detection across PL/SQL files. SQL metrics were extracted from Oracle PL/SQL code of a warehouse management database application system. The faults were collected from the journal files that contain the documentation of all changes in source files. The result demonstrates that these measures may be useful in predicting the fault concerning with database accesses. In our study, General Regression Neural Network and Ward Neural Network are used to evaluate the capability of this set of SQL metrics in predicting the number of faults in database applications.

Keywords: Structured Query Language metrics; Software prediction; Neural network; Software metrics

1. Introduction

Software metrics have been used as a quantitative means of assessing software development process as well as the quality of software products. Many researchers have studied the correlation between software design metrics and the likelihood of occurrence of software faults. They classified the software modules (or classes) as not fault-prone or fault-prone modules (or classes) and predicted the number of faults in modules (or classes) using various software metrics [1–12,14–16,18,20].

Although database applications are essential to every organization, studies on product measures for static database operation statements are rarely found in literature. Structured Query Language (SQL) is the standard language for relational database management systems. The relationship between the fault occurrence for database applications and SQL metrics is studied in this paper. We defined SQL metrics that have strong relationship with faults and then performed empirical validation for these metrics.

In our study, we analyzed the fault reports of development projects involving database applications using PL/SQL code and found that faults are related to the number of SQL statements and the complexity of SQL statements. SQL commands are mainly composed in the PL/SQL files to perform database operations. SQL complexity can be measured using product metrics such as the number of table names in from-clauses: the number of join-queries criteria in where-clauses.

A variety of statistical techniques are used in software quality modeling. Models are often based on statistical relationships between measures of quality and measures of 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 adopt at modeling nonlinear functional relationships that are difficult to model with other techniques, and thus, are attractive for software quality modeling. In our previous studies, we predicted software development faults using Object-Oriented Design Metrics and neural network [16,20]. In these studies, we found that neural network models had better predict accuracy than regression models. In this study, the General

Regression Neural Network (GRNN) and Ward Neural Network are used to predict faults in PL/SQL codes using SQL metrics.

2. SQL metrics

Database application constitutes one of the largest and most important software domains in the world [17]. Some classes or modules in those applications are responsible for handling database accesses. We analyzed the fault reports of these classes or modules and found that faults are related to the number of SQL statements, which are invoked from a class or module; and the complexity of SQL statements. For example, retrieving wrong database records. In such cases, developers need to check and modify the corresponding SQL statements to correct the error. To be able to predict the number of such faults for these classes or modules are very important in developing database applications.

The following SQL metrics are defined and used in this study. Metrics having weak relationships with fault occurrences, such as the number of Data Definition Language (DDL) commands and Data Control Language (DCL) commands are omitted in this study.

3. Neural network modeling

The first neural network architecture that we have chosen is the Ward Network [21]. It is a backpropagation network that has three slabs (slab2, slab3 and slab4) in the hidden layer (Fig. 1). Hidden layers in neural network are known as feature detectors. A slab is a group of neurons. Each slab in the hidden layer has a different activation function, this offers three ways of viewing the data. We use linear function for the output slab (slab5). Hyperbolic tangent (tanh) function is used in one slab of hidden layer (slab3) 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 (slab2). 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 (slab4) 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 GRNN. GRNN is based on a one-pass learning algorithm with a highly parallel structure. GRNN is a powerful memory based network that could estimate continuous variables and converge to the underlying regression surface. The strength of GRNN is that it is able to deal with sparse data effectively. Specht [19] shown that the algorithm in GRNN is able to provide a smooth

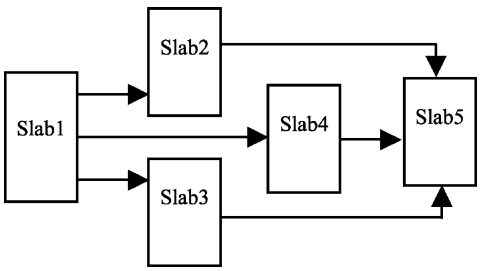

Fig. 1. Ward Neural Network.

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 one neuron. The primary advantage of GRNN is the speed at which the network can be trained. There are no training parameters such as learning rate and momentum in back-propagation network, but there is a smoothing factor that is applied after the network is trained. The smoothing factor allows the GRNN to interpolate between patterns or spectra in the training set. The smoothing factor determines how tightly the network matches its predictions to the data in the training patterns. For GRNN networks, the smoothing factor must be greater than 0 and usually range from 0.01 to 1 with good results.

4. Data collection

The experiment data are collected from a set of warehouse management applications that is developed using C, JAM and PL/SQL languages. This set of applications has more than a thousand source files of C, JAM and PL/SQ codes and uses Oracle database. The warehouse system has been customized and used by many companies. Data access faults were collected from the journal files that contain the documentation of all changes in source files such as status of module, start date, end date, developer, nature of changes, etc. Data on six SQL metrics (Table 1) were extracted from 108 PL/SQL files of the above-mentioned warehouse application using metric

Table 1
Proposed SQL metrics

SQL metrics	Description
NSC	Number of select commands
NIUO	Number of insert/update operations
NDO	Number of delete operations
NT	Number of table in from-clause
NSCC	Number of search condition criteria in where-clause
NJQ	Number of join-queries criteria

extraction tool that we developed using VC++ incorporating MKS LEX and YACC utilities as embedded languages.

5. Experiment

Firstly, each data pattern was examined for erroneous entries, outliers, blank entries and redundancy. We standardized the SQL metrics to a mean of zero and a variance of one for each metric. Many raw software metrics have incompatible units of measures. This step converts all of them to a unit of one standard variation. After standardizing SQL metric data, we performed principal component analysis (PCA). It is a standard technique to identify the underlying, orthogonal dimensions that explain relations between the variables in the data set. The varimax rotation method was adopted in this study. Table 2 presents the relationship between the original SQL metrics and the domain metrics, based on experiment data extracted from the warehouse management applications.

PCA identified two sets of principle components (PCs), which capture 58.34 and 25.39%, respectively, of the data set variance, which gives a representation of about 84% of the population. Table 2 shows the coefficient measure for each rotated component, with coefficients larger than 0.8 set in boldface. The Eigen value, the percentage of data set each PC describes, and the cumulative variance percentage are also shown. Based on the analysis of the coefficients associated with each metric within each of the two sets of rotated components, the PCs are interpreted as follows:

The first set of principle components shows high correlation between metrics NT, NSCC, NSC and NJQ. However, NT is a better representative because it is less correlated with the other components. The second set of principle components shows high correlation between metrics NIUO and NDO. We therefore chose NT and NIUO metrics for further analysis.

We divided our data into training, testing, and production sets using 3:1:1 ratio, which is the commonly accepted proportion used by most neural network researchers. We extracted 21 patterns for the test set and another 21 patterns for the production set. The remaining 66 patterns are used as training set. We used the production data set to evaluate model performance.

Table 2
Principle components

SQL metrics	PC1	PC2
NSC	**0.962**	0.116
NIUO	0.029	**0.872**
NDO	0.052	**0.859**
NT	**0.979**	0.101
NSCC	**0.967**	−0.038
NJQ	**0.823**	0.008
Eigenvalues	3.500	1.524
% Variance	58.336	25.394
Cumulative % variance	58.336	83.730

Table 3
Ward Neural Network architecture used

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

Table 4
GRNN architecture used

	Input layer	Hidden layer	Output layer
No. of neurons	2	108	1

The dependent variable was the number of software faults and the independent variables were the two principal components identified above (out of the six SQL metrics). We used both Ward Network and GRNN Network for predicting faults. Table 3 shows the summary of Ward Network design whereas Table 4 shows the structure of GRNN network used.

For GRNN network, there were 108 neurons in hidden layer, two neurons in input layer and one neuron in output layer. In our experiment smoothing factor 0.075 was used.

6. Experimental results

To measure the goodness of fit of the model, we use the coefficient of multiple determination (R^2), the coefficient of correlation (r), r^2, mean square error, mean absolute error, minimum absolute error and maximum absolute error.

Tables 5 and 6 show the values that are obtained from the two neural network models used for our experiments.

Table 7 shows the experimental result of the production set. The correlation of the predicted change and the observed change is represented by the coefficient of correlation (r). The r value of 0.8586 in Ward Neural Network and 0.7096 in GRNN network represents high correlations for cross-validation. The number of

Table 5
Experimental result of training set

	Ward Net	GRNN
R^2	0.8159	0.7717
R	0.9093	0.8889
r^2	0.8269	0.7901
Mean squared error	1.731	2.145
Mean absolute error	0.832	0.858
Min. absolute error	0.004	0
Max. absolute error	4.187	5.673

Table 6
Experimental result of test set

	Ward net	GRNN
R^2	0.3754	0.1247
R	0.6881	0.5530
r^2	0.4735	0.3058
Mean square error	1.779	2.493
Mean absolute error	0.653	0.936
Min. absolute error	0.019	0.288
Max. absolute error	5.605	6.317

Table 7
Experimental result of production set

	Ward Net	GRNN
R^2	0.6889	0.3993
r	0.8586	0.7096
r^2	0.7372	0.5035
Mean square error	0.518	1.000
Mean absolute error	0.552	0.718
Min. absolute error	0.004	0.106
Max. absolute error	1.845	2.830
t-Value	5.389928	4.389928
P-value	0.0001	0.0003

observations is 21. The significance level of a cross-validation is indicated by p value. A commonly accepted p value is 0.05 [13]. In our experiment, a two tailed probability p values is less than 0.0003 in both cross-validations. This shows a high degree of confidence for the successful validations. The results clearly indicate close relationship between metrics NSC, NIUO, NDO, NT, NSCC and NJQ (independent variables) and predictable faults in software applications (dependent variable).

7. Conclusions

We studied the relationship between fault occurrence for database applications and SQL metrics. First we proposed SQL metrics that have strong relationship with faults and then performed empirical validation for these metrics. We analyzed the fault reports kept by project teams of developing database applications using PL/SQL code and found that faults are related to the number of SQL statements and the complexity of SQL statements. The relationship between the fault occurrence for database applications and SQL metrics has been empirically validated in this study. From the results presented above, our proposed SQL metrics in this study proved to be useful in predicting faults in PL/SQL files.

These findings paved the way for future research into using neural network for predicting software maintainability. In addition, our research results also provide a new avenue for software project manger to determine the readiness of software under development.

8. Future plan

We intend to extend this investigation with wide range of applications and various types of data access techniques. Our future research direction aims to estimate software readiness by using metrics for defect tracking. To estimate 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. Software metrics concerning with polymorphism measures, inheritance related measures, complexity measures, cohesion measures, coupling measure, dynamic memory allocation measure, SQL measures and size measures will be used.

Acknowledgements

We gratefully acknowledged CAIB GmbH, Murrhardt, Germany for providing us with the software development data used in our research.

References

[1] A. Mounir Boukadoum, H.A. Sahraoui, H. Lounis, Machine learning approach to predict software evolvability using fuzzy binary trees, International Conference on Artificial Intelligence, 2001.
[2] L.C. Briand, J.W. Daly, J.K. Wust, A unified framework for coupling measurement in object-oriented systems, IEEE Transactions on Software Engineering 25 (1) (1999) 91–121.
[3] 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 28 (2002) 706–720.
[4] L. Briand, J. Wüst, J.W. Daly, V. Porter, Exploring the relationships between design measures and software quality in object-oriented systems, Journal of Systems and Software 51 (2000) 245–273.
[5] M. Cartwright, M. Shepperd, An empirical investigation of object oriented software system, IEEE Transactions on Software Engineering 26 (2000) 786–796.
[6] K. El Emam, A primer on object-oriented measurement, Proceedings of the Seventh International Software Metrics Symposium, 2001, pp. 185–187.
[7] K. El Emam, S. Benlarbi, N. Goel, S.N. Rai, The confounding effect of class size on the validity of object-oriented metrics, IEEE Transactions on Software Engineering 27 (2001) 630–650.
[8] El Emam, W. Melo, C.M. Javam, The prediction of faulty classes using object-oriented design metrics, Journal of Systems and Software, Elsevier Science 56 (1) (2001) 63–75.
[9] L. Etzkorn, H. Delugach, Towards a semantic metrics suite for object-oriented design, Proceedings of 34th International Conference on Technology of Object-Oriented Languages and Systems, 2000, pp. 71–80.
[10] N.E. Fenton, N. Ohlsson, Quantitative analysis of faults and failures in a complex software system, IEEE Transactions on Software Engineering 26 (2000) 797–814.

[11] F. Fioravant, A metric framework for the assessment of object-oriented systems, Proceedings of IEEE International Conference on Software Maintenance, 2001, pp. 557–560.
[12] 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.
[13] J.E. Frenund, F.J. Williams, B.M. Perles, The Elementary Business Statistics—The Modern Approach, Prince-Hill, 1993.
[14] T.L. Graves, A.F. Karr, J.S. Marron, H. Siy, Predicting fault incidence using software change history, IEEE Transactions on Software Engineering 26 (7) (2000) 653–661.
[15] T.M. Khoshgoftaar, E.B. Allen, Z. Xu, Predicting testability of program modules using a neural network, Proceedings of the Third IEEE Symposium on Application-Specific Systems and Software Engineering Technology, 2000, pp. 57–62.
[16] J.T.S. Quah, M.M. Thet Thwin, Prediction of software readiness using neural network, Proceedings of First International Conference on Information Technology and Applications (ICITA 2002), Australia, 25–28 Nov (2002).
[17] G. Ramarkrishnan, Database Management Systems, Third ed., McGraw-Hill, New York, 2003.
[18] R. ReiBing, Towards a model for object-oriented design measurement, Proceedings of the Fifth International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, 2001, pp. 71–84.
[19] D.F. Specht, A general regression neural network, IEEE Transactions on Neural Networks 2 (6) (1991) 568–576.
[20] M.M. Thet Thwin, T.-S. Quah, Application of neural network for predicting software development faults using object-oriented design, Proceedings of Ninth International Conference on Neural Information Processing, Singapore, 18–22 Nov 2002, vol. 5, 2002, pp. 2312–2316.
[21] NeuroShell 2 Help, Ward Systems Group Inc., http://www.wardsystems.com