Predict Software Failure-prone by Learning Bayesian Network

Yuyang Liu[1], Wooi Ping Cheah[1], Byung-Ki Kim[1], Hyukro Park [1],

[1] Chonnam National University, South Korea
wxllyy@hotmail.com, cheahwooiping@gmail.com, bgkim@chonnam.ac.kr, hyukro@chonnam.ac.kr

Abstract. We explore the software metrics and build a Bayesian Network Model for defect prediction. Much previous work has concentrated on how to select the software metrics that are most likely to indicate fault-proneness, based on the hypnosis that these metrics are independent. But in reality, software metric values are predicted not only correlated with fault-proneness, but also observed internal complex relationship with each other. In this paper, we build a Bayesian network model to represent the probability distribution of each factor and how they affect defects, considering strong or weak correlations are existed between individual metric attributes. We perform a comparative experimental study of effectiveness of Bayesian Network, logistic regression and Naive Bayes on a public data set from an open source software system. The result shows that our approach produces statistically significant estimations.

Keywords: Bayesian analysis, Bayesian networks, software defects, code metrics, software quality.

1. Introduction

Acquisitions of which files in the large software system are most likely to contain the largest number of defects always are significant valuable to any project developer. As software defects cannot be directly measured, many researches tend to explore software metrics, which are considered to efficiently predict software quality in the early age of software development, for example code complexity, OO metrics, file change histories and etc., to predict product’s failure-proneness.

One of the empirical facts towards software failures is that software modules tend to be classified as failure-prone or non-failure-prone. Basically, former work on software failure-prone prediction has prove that correlation exists between software metrics and fault-proneness, but most of them concentrated on how to select software metrics, and build a model to compute the result. These work are weak in two points, for one thing, the effect is subsequently cut down, when evaluable dimensions decreased [1]. For another, they are based on the hypnosis that these metrics are considered singly. Whereas in reality, in terms of these metrics measure various aspects of the same software product, individual metric attributes tend to be highly correlated with each other (known as multicollinearity) [13]. To avoid these weaknesses, our approach tends to predict failure probabilistic by a Bayesian Network

Classifier that related software metrics and failure history, in order to classify free defect module from non-free ones.

Traditionally, to constructed a BN (structure and conditional probability distributions) usually by following two ways: 1) to provide a structural model which representing relationships between the attributes by domain experts, then learn available data to get their corresponding probabilities, and 2) mining the available data in order to get relationships, typically using learning algorithms [14]. For the reason of exploring unsure composition of software metrics, in our approach, we use the second way to draw the structure by learning from data.

In this paper, we use a BN which relates software product metrics and failure history to fault proneness after removing the sick related factors. Then learning the conditional probability of each factor to draw up how they impact the probability of one file inclines to have defect. The original data set is derived from the Eclipse project (www.eclipse.org), which has been provided and published by Zimmermann et al. [4]. While Zimmermann et al. predict defects using logistic regression and complexity metrics, and our work differ from theirs in using a Bayesian Network to represent correlation between metrics and get a more accuracy result in file level.

The remainder of this paper is organized as follows: In Section 2, we discussed related work in the literature and introduced Bayesian networks. Our research method, including data description, model construction is proposed in Section 3. Experiments and model evaluation are discussed in Section 4. At last, we conclude our work with directions for future work in Section 5.

2. Related work

In this section, we review the relevant literature from software quality management. In particular, we look at the 1) use and analysis of software metrics, 2) introduction of Bayesian Network methodology.

2.1 Analysis of software metrics

In the field of software development, software metrics are collected at various stages in the development cycle, and utilized to evaluate the quality of a software product. They are also considered as the most critical factors to identify potentially error-prone modules in software systems, so that extra development and maintenance effort can be directed at those modules [7], [8], [9]. Several statistical tools used in the analysis of software metrics include logistic regression, linear least-squares regression, Poisson regression and etc.

Although various software metrics suites have developed, there is still no one metric or known combination of metrics can predict software quality as a general model. Furthermore, due to experience that specific subset of predictors from the suite which seem to show a significant relationship to fault proneness differs from system to system, therefore, to predict software quality accuracy is always to be a hard work. To fill up the inadequacy of traditional linear models, many machine learning techniques of non-linear models (Fuzzy [11], ANN [15]) are applied by researchers, which are expected to provide superior performance than their linear counterparts.

Some studies also show that software quality at the module level of future releases can also be predicted base on the past history. Former studies have proved that modules with more defects in development are likely to have more defects after release [12].Zimmermann et al. [4] mapped defects from the bug database of Eclipse to source code locations for three releases of the Eclipse project and additionally annotated such data with a vast amount of size and complexity metrics extracted from source code. They found a significant correlation between complexity metrics and pre- and post-release defects.

Although there is one existing work in the literature that describes BN-based methods for fault content and fault proneness prediction [5], but they draw a general linear relation directly between attributes and target. We present a more reasonable BN model to predict failure-prone and perform a comparative experimental study of the effectiveness of Bayesian Network, logistic regression and Naive Bayes model on a data set from an open source software system - eclipse.

2.2 Bayesian Network Introduction

A Bayesian Network is based on the Bayes thermo by calculating the conditional probabilities between variables, then to derive a more precise result. Assuming a simple set that is $S = \{A_1, A_2, \ldots . A_{n,}, B\}$, where A_i(i ∈[1, n]) is the occurred events in E and B refers to one existing event $p(B) \neq 0$. Bayesian Network of S consists of two components:

1) Joint probability distribution of $\{A_i\}$(i ∈ $[1, \mathrm{n}]$), as a directed acyclic graph *D* that encodes a set of conditional independence assertions about variables in *S*.

2) A set *P* of local probability distribution for A_i, which is called Conditional Probability Table (CPT). Each node corresponds to a variable, and the CPT given every possible combination of states of its parents.

The set of parents of A_i, denoted π_i, is the set of nodes with an arc to A_i in the graph. Then the probability of an arbitrary event A can be calculated as

$$p(A) = \prod_{i=1}^{n} p(A_i|\pi_i) \quad (1)$$

Given this joint probability, the marginal probability of an A_i is computed as

$$p(A_i) = \sum_{\mathrm{x_j, j \neq i, j=1}}^{\mathrm{n}} p(A) \quad (2)$$

BN are practically applied to model causal influences, where the modules of a system are modeled as the nodes of a BN, while the edges represent the cause-effect relationships between the entities. The qualitative part of a BN is encoded in the structure of the digraph, while the conditional probability distributions for the nodes encode the quantitative portion [5].

We are motivated to choose BNs based on three reasons. One, Bayesian networks in presentation of intra-relationship between software metrics. Two, Bayesian networks allow one to learn about causal relationships. Especially facilitate the combination of domain knowledge and data. Three, once a BN has been specified, not only defect probability can be deduced, but also software quality can be controlled by balance evaluable factors. Thus, in our context, using a BN to handle the relationship between evaluable factors of software product permits us to explore the drivers of observed good quality.

3. Research Method

Our research approach is to use a BN to model the relationships between the measurable properties of a software product and its quality. Especially in the following way: to formulate a BN structure that represents the relationships and conditional probability between the software metrics and error-prediction.

3.1 Model Parameters

In this paper, the main variables are a suite of metrics measuring the structural quality of object-oriented code and design; specifically, we consider the suite in [6]. The dataset we used is coming from a well-known open source project- eclipse that is written in Java which encompasses 10593 files. The following 32 OO code metrics are associated with each file, we choose these metrics for the advantage of being easier to implement and understand.

1. Fan OUT (**FOUT**): The number of method calls, including *avg, max, sum*[1].
2. Method Lines of Code (**MLOC**): The number of lines of code inside method bodies, excluding blank lines and comments, including *avg, max, sum*.
3. Total Lines of Code (**TLOC**): Total lines of code in the selected file. Only counts non-blank and non-comment lines inside method bodies computed KLOC.
4. Number Of Methods (**NOM**): The number of methods implemented in a given file, including *avg, max, sum*.
5. Number Of Attributes (**NOF**): The number of attributes in a given file, including *avg, max, sum*.
6. Number Of Static Methods (**NSM**): The number of static methods in a given file
7. Number of Static Attributes (**NSF**): The number of static attributes in a given file, including *avg, max, sum*.
8. McCabe cyclomatic complexity (**VG**): Counts the number of flows in a given file. Each time a branch occurs (if, for, while, do, case, catch and the ?: ternary operator, as well as the && and || conditional logic operators in expressions) this metric is incremented by one. Calculated for methods only, including *avg, max, sum*.
9. Number of Parameters (**PAR**): The number of parameters in a given scope, including *avg, max, sum*.
10. Nested Block Depth (**NBD**): The depth of nested blocks of code in a given file, including *avg, max, sum*.
11. Number of Interfaces (**NOI**): Total number of interfaces in a given file, including *avg, max, sum*.
12. Number of Classes (**NOT**): Total number of classes in a given file, including *avg, max, sum*.
13. Pre-release Defects (**pre**): The number of non-trivial defects that were reported in the last six month before release.

One output is as follows:

1. Post-release Defects (**post**): The number of non-trivial defects that were reported in the first six months after release.

[1] *avg, max* and *sum* is the abbreviation of average, maximum and accumulation respectively, means the specified variable's average, maximum and accumulation number of each method in a given file.

In terms of classifying modules as failure-prone or not, we discrete the number of **post** in two classes, for “Yes” means the 0 of defects in a given class while “No” represent there is at least 1 defects appeared during testing. Then this problem can simply be considered as a two-class classification.

3.2 Construct Bayesian Network

In our approach, Bayesian Network is used as a classifier. Considering former set S in software defect prediction case, we suppose X= $\{"yes", "no"\}$ is a discrete random variable related to B, accords with the following Equation (3):

$$\mathrm{X} = \begin{cases} yes, \ when\ there\ is\ defects\ showed\ in\ module \\ no, \ when\ there\ is\ no\ defects\ detected \end{cases} \qquad (3)$$

Then we can deduct the posterior probability (or conditional probability) under the condition of B occurred: $\mathrm{p}(\mathrm{A}_i|\mathrm{X} = yes) = \frac{\mathrm{p}(yes|\mathrm{A}_i)\mathrm{p}(\mathrm{A}_i)}{\sum_{\mathrm{j}=1}^{\mathrm{n}} \mathrm{p}(yes|\mathrm{A}_j)\mathrm{p}(\mathrm{A}_j)}, i = 1,2, \ldots \mathrm{n}$ (4)

The learning procedure is:

1. Compute $\mathrm{I}_{\widehat{\mathrm{P}_\mathrm{D}}}(\mathrm{A}_i; \mathrm{A}_j|\mathrm{B})$ between each pair of attributes, $\mathrm{i} \neq \mathrm{j}$.
2. Build a complete undirected graph in which the vertices are the attributes $A_1, A_2, \ldots . A_n$. Annotate the weight of an edge connecting A_i to A_j by $(\mathrm{A}_i; \mathrm{A}_j|\mathrm{B})$.
3. Build a maximum weighted spanning tree.
4. Transform the resulting undirected tree to a directed one by choosing a root variable and setting the direction of all edges to be outward from it.
5. Construct a BN model by adding a vertex labeled by B and adding an arc from B to each A.
6. Remove the edges with low weight between B and each A.
7. Learn the parameters and output the structure.

In the step one, $\mathrm{I}_{\widehat{\mathrm{P}_\mathrm{D}}}(\mathrm{A}_i; \mathrm{A}_j|\mathrm{B})$ is given by

$$\mathrm{I}_{\widehat{\mathrm{P}_\mathrm{D}}}(\mathrm{A}_i; \mathrm{A}_j|\mathrm{B}) = \sum_{\mathrm{A}_i, \mathrm{A}_j, B} \mathrm{P}(\mathrm{A}_i, \mathrm{A}_j, B) \log \frac{\mathrm{P}(\mathrm{A}_i, \mathrm{A}_j|B)}{\mathrm{P}(\mathrm{A}_i|B)\mathrm{P}(\mathrm{A}_j|B)}$$

In the last step of construct BNs, marginal distribution of each node in our model is calculated by Bayes Network Editor, a tool in Weka[2], for its underlying Bayesian propagation algorithm. However, learning a BN towards cost sensitive for software failure prediction is left as an aspect of future work.

4. Experiments

In this section, we evaluate our approach empirically using the metrics from the [4] data set. By analyzing the error in software defect prediction, we examine our model in three dimensions, and compare it with exist work and Naïve-Bayes model.

[2] Weka is a collection of machine learning algorithms for solving real-world data mining problems. It contains the tool of edit Bayesian Network and learning dataset.

4.1 Data pre-processing

Before constructing the BNs, we first discretize the data into 5 levels. Discretization reduces the number of values for a given continuous attribute by dividing the range of the attribute into intervals, therefore reduces the size of the network. It also reduce the data by collecting and replacing low level concepts (such as numeric values for the attribute pre in our model) by higher level concepts (showed as number field), then evaluate the correlation between selected key factors. In our model, the numeric values of each attribute are collapsed into 5 levels with equal-frequency binning. Equal-frequency binning is method in discrete the data, which is simply but proved to be efficiency.

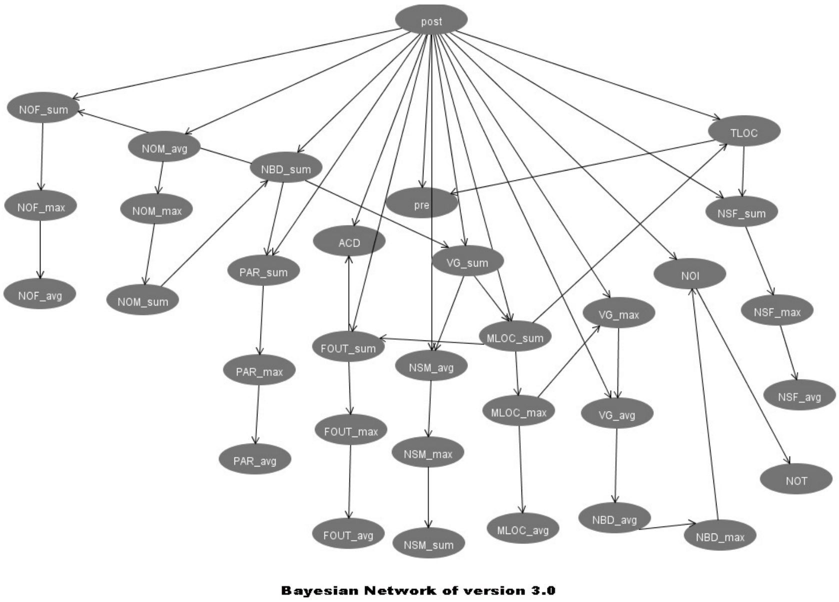

Figure 1 BN model for fault proneness analysis in file level

4.2 Fault Proneness Analysis

By applying the previous steps in chapter 3, Bayesian Network structures are constructed as Figure 1, and Table 1 shows the confusion matrices obtained. As in the BN model for assessing fault content, we performed a 10-fold cross validation to construct the functional form of the CPD for fault proneness.

Correspond to table, the less numerical value of False Negative (FN) and True Negative (TN) represent better software quality. One of the goals of this paper is to experimentally evaluate how Bayesian methods can be used for assessing software fault proneness. Table 2 illustrates the comparison among Zimmermann’s model, Naïve-Bayes model and our approach. The FN equals 0.0522 means very little observed as non-failure modules are predicted as a failure-prone module, while TN equals 0.6103 means 61% of high failure-prone modules are predicted as non-failure modules. All corrected classified cases in our model occupy 86.51%. In Zimmermann's case, although we get the similar value of TN, FN rate in our approach is much less than their values, the Accuracy is higher as well. When compared with the Naïve-Bayes model, FN and Accuracy present précised.

TABLE 1 Confusion Metrics

Observed as have defects	Classified as		
	No	Yes	Total
No	8553	957	9510
Yes	472	611	1083
Total	9025	1568	10593

TABLE 2 Comparsion with other researches

	FN rate	**TN rate**	**Accuracy**
Zimmermann. [4]	0.337	0.621	0.711
Naïve-Bayes	0.2575	0.3327	0.7310
Our approach	0.0522	0.6103	0.8651

5. Conclusion and Future Work

The broad goal of our research is to build a model to analysis the causal relation between evaluable metrics and software quality in software development. Then enhance software development efficiency by exploring the dependence between them.

In this paper, we apply BNs to solve the problem of classifying software modules as defect-free or non-defect-free. The experimental results confirmed the superior performance by comparing our approach with logistic model. The BN model provides a robust mechanism to include diverse sources of data into the analysis.

However, there is always more to do. Our future work will be improved in these fields: First, to exam this method with more data. Besides only use one project's data is not convictive enough, dataset in different software project which focus on different functions tends to present different weight of each matrix [3]. Secondly, to evaluate more efficient way to discrete the dataset, naming data preprocessing, which is a important factor in present results.

Finally, there have been several empirical studies that have examined the relationship of product metrics, failure history and fault proneness, but few that have explored the casual inference between them. The BN model provides a robust mechanism to detect the software defect prone.

6. ACKNOWLEDGMENTS

The authors would like to thank Kamswee, who help us in using Matlab to get the model. We also acknowledge all the reviewers who helped to improve the paper with their detailed and constructive comments.

7. REFERENCES

1. John C Munson, Taghi M Khoshgoftaar, “The Detection of Fault- Prone Programs,” IEEE Transactions on Software Engineering, 1992.
2. Taghi M. Khoshgoftaar, Naeem Seliya, “Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study,” Empirical Software Engineering, Volume 9, Issue 3, Pages: 229 - 257, September 2004.
3. N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures", Proceedings of the International Conference on Software Engineering (ICSE 2006), Shanghai, China, 2006.
4. T.Zimmermann, R.Premraj, A.Zeller, “Predicting Defects for Eclipse”, Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops, 2007.
5. G.J.Pai, J.B.Dugan, “Emprical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods”, IEEE Trans. Software Eng., vol. 33, no. 10, pp. 675- 686, July 2007.
6. Henderson-Sellers, B., “Object Oriented Metrics - Measures of Complexity”, Prentice Hall, Upper Saddle River, NJ, 1996.
7. N. Fenton and S. L. Pfleeger. “Software Metrics: A Rigorous and Practical Approach”, International Thomson Computer Press, London, UK, second edition, 1997.
8. C. Lewerentz and F. Simon. A product metrics tool integrated into a software development environment. In S. Demeyer and J. Bosch, editors, Object-Oriented Technology (ECOOP'98 Workshop Reader), LNCS 1543, pages 256 - 257. Springer-Verlag, 1998.
9. L.C. Briand, W.L. Melo, and J. Wu. st, “Assessing the applicability of Fault-Proneness Models across Object-Oriented Software Projects,” IEEE Trans. Software Eng., vol. 28, no. 7, pp. 706-720, July 2002.
10. R. Subramanyam and M.S. Krishnan, “Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects,” IEEE Trans. Software Eng., vol. 29, no. 4, pp. 297-310, Apr. 2003.
11. R.K. Lind, K. Vairavan, An experimental investigation of software metrics and their relationship to software development eIort, IEEE Trans. Software Eng. 15 (1989) 649–653.
12. S. Biyani, Santhanam, P., “Exploring defect data from development and customer usage on software modules over multiple releases", Proceedings of International Symposium on Software Reliability Engineering, pp. 316-320, 1998.
13. S. Dick, A. Meeks, M. Last, H. Bunke, A. Kandel, “ Data Mining in Software Metrics Databases”, Fuzzy Set and Systems, pp. 81-110
14. Dan Geiger, Moises Goldszmidt, G. Provan, P. Langley, P. Smyth, “Bayesian Network Classifiers”, Machine Learning, 1997
15. D.Heckerman, “A Tutorial on Learning With Bayesian Networks”, Microsoft, 1996
16. I.Gondra, “Applying machine learning to software fault-proneness prediction”, System and Software, pp. 186-195, 2008