Predicting Fault Prone Modules by the Dempster-Shafer Belief Networks

Lan Guo and Bojan Cukic
Lane Department of CSEE
West Virginia University
Morgantown, West Virginia 26506-6109
{lan, cukic}@csee.wvu.edu

Harshinder Singh
Department of Statistics
West Virginia University
Morgantown, West Virginia 26506-6330
hsingh@stat.wvu.edu

Abstract

This paper describes a novel methodology for predicting fault prone modules. The methodology is based on Dempster-Shafer (D-S) belief networks. Our approach consists of three steps: First, building the Dempster-Shafer network by the induction algorithm; Second, selecting the predictors (attributes) by the logistic procedure; Third, feeding the predictors describing the modules of the current project into the inducted Dempster-Shafer network and identifying fault prone modules. We applied this methodology to a NASA dataset. The prediction accuracy of our methodology is higher than that achieved by logistic regression or discriminant analysis on the same dataset.

1. Introduction

Software developers have a keen interest in software quality models, which automatically predict fault prone modules and subject them to more rigorous verification activities. Accurate predictions enable verification experts to concentrate their attention and resources at problem areas in the system under development.

Many modeling techniques have been developed and applied to software quality prediction, such as: logistic regression [1], discriminant analysis [11], the discriminative power techniques [13], optimized set reduction [2], neural networks [7], fuzzy classification [3], and classification trees [14]. The prediction accuracy of those models does not vary significantly. Generally, there exists a trade off between the defect detection rate and the overall prediction accuracy.

In this paper, we introduce a novel software quality prediction methodology, based on the Dempster-Shafer (D-S) belief networks [4]. The methodology is general and not restricted to particular metrics or research objectives. Furthermore, it is fully objective, highly automatic and computationally efficient. The prediction accuracy of our methodology is higher than that achieved by logistic regression or discriminant analysis on the same dataset. In addition, the methodology is more effort economic for determining which modules to inspect than another defect module detector, ROCKY [16].

This paper is organized as follows. Section 2 describes Dempster-Shafer networks. Section 3 introduces the dataset and measurement parameters. Section 4 outlines major steps of the methodology. Section 5 describes the experiments. Section 6 evaluates our results and Section 7 concludes the paper.

2. Dempster-Shafer Belief Networks

The Dempster-Shafer Belief Network is a complete formalism of evidential reasoning for computing and propagating evidential support through the network. Dempster-Shafer (D-S) belief Networks were first built by Liu et al. [9]. We developed an alternative algorithm in [4]. This induction algorithm is based on *prediction logic* [6] and is applicable for implication rules in general.

The induced D-S network is a directed graph. Nodes in D-S networks are connected by implication rules. When evidence from distinct sources is observed for certain node, it is combined by the Dempster-Shafer scheme [15]. Beliefs for the corresponding nodes are updated and propagated through the network by the algorithm from [9].

Dempster-Shafer networks may not be singly connected. In order to prevent circular traversal of the graph, each node in the network is updated only once when an observation is made. Therefore, different order of observations may result in different results, since different paths might be traversed.

3. Datasets and Measurements

The dataset used in the case studies is a NASA project, referred to as KC2. KC2 contains over 3,000 modules (a module is equivalent to a C function). NASA developers built 520 modules. The remaining modules are COTS. Out of the 520 modules, 106 were found to have between 1 to 13 faults. KC2 modules have the average size of 37 lines of code (LOC), while the largest module has 1,275 LOC.

The dataset contains twenty-one metrics, including McCabe [10], Halstead [5], line counts and branch counts. KC2 dataset contains additional three fields: *Error Rate* (number of defects in the module), *Defect* (whether or not the module has any defects), and *Defect Density* (number of defects per LOC).

Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE'03)
1527-1366/03 $ 17.00 © 2003 IEEE

		Defect present?	
		No	Yes
Defect Predicted?	No	TN=true negative	FN=false negative
	Yes	FP=false positive	TP=true positive
	No prediction	NP$_1$	NP$_2$

Figure 1. A defect prediction sheet

In this study, we are interested in predicting whether or not the module contains any defects, instead of how many defects it contains. Software metrics serve as predictors. The predicted variable is *Defect*. Figure 1 presents a defect prediction sheet.

Specificity is used to define the rate of the defect module detection. In the literature, it is also referred to as *Probability of Detection (PD)* [16]:

$$Specificity\ (PD) = \frac{TP}{FN + TP + NP_2}$$

Similarly, *Sensitivity* is defined as the portion of the correct classification of non-fault prone modules:

$$Sensitivity = \frac{TN}{TN + FP + NP_1}$$

The overall prediction accuracy is measured by *Acc*:

$$Acc = \frac{TN + TP}{TN + FN + FP + TP + NP_1 + NP_2}$$

Another parameter is *Probability of False alarm (PF)*. It represents the ratio of non-fault prone modules predicted as fault prone modules:

$$PF = \frac{FP}{TN + FP + NP_1}$$

Effort is defined to represent the resources required for the inspection of faulty modules [16]:

$$Effort = \frac{LOC_{FP} + LOC_{TP} + LOC_{NP}}{LOC_{TN} + LOC_{FN} + LOC_{FP} + LOC_{TP} + LOC_{NP}},$$

where $LOC_{NP} = LOC_{NP_1} + LOC_{NP_2}$.

4. The Methodology

4.1 Primary Data Treatment

KC2 is numerical continuous dataset. Since Dempster-Shafer networks deal with discrete datasets, we discretized the original dataset into binary one by AWK programs. We partition the dataset using the mean value or the median in each field. If the data value is greater than the mean (median) of the corresponding field, it is assigned *1*; otherwise, it is *0*. The predicted variable, *Defect*, is *1* if the module contains fault(s), or *0* if it is fault free.

4.2 Selecting the Predictors

There are 21 predictors in the datasets. Some of them are highly correlated. In order to down-select the best predictors, we applied a logistic regression procedure in SAS [12] to the discretized datasets.

The logistic regression procedure in SAS generates 20 score tables of the candidate predictors within a second. It ranks the Chi-Square scores for each combination of the predictors. The number of the predictors in the score tables increases from 1 to 20. For example, score table 1 contains best single predictors; score table 2 contains best combinations of 2 predictors, etc. The top 5 combinations from each score table were picked as the candidate predictor sets to the Dempster-Shafer networks.

4.3 Empirical Validation

We used 10-fold cross-validation to evaluate the prediction of fault prone modules. The dataset was randomly partitioned into 10 bins of equal size. The D-S network is trained and tested 10 times. Each time 9 bins were picked to build the Dempster-Shafer network by the induction algorithm. Belief revision algorithm was applied to the remaining bin. The experiment was complete when all the 10 bins were validated. Cross validation was run at least 60 times in each experiment. The result with the least variance was finally selected.

The predictors picked by the logistic procedure were used by the inducted Dempster-Shafer networks. Since the order of the observations matters, different sequences of the predictors were tried, and the best sequences were recorded. In addition to the sequence, there are five other tuning parameters in the D-S networks. For example, to achieve maximal *Specificity (PD)* and *Acc*, or to use minimal *Effort* while achieving maximal *PD*, the system can be tuned to output the optimal results. The optimal results for each set of criteria are selected by dominant rule, which first sorts the results in order and then discards the results dominated by others. For example, if the measurement parameters of interest are *Acc* and *PD*, we use $r = < Acc; PD >$ to represent each result. Suppose we have $r_1 = < 0.6; 0.7 >$, $r_2 = < 0.5; 0.8 >$, $r_3 = < 0.6; 0.6 >$. Since r_3 is dominated by r_1, it is discarded. Therefore, the optimal results are r_1 and r_2.

5. Experimental Results

The original KC2 dataset was discretized into two binary datasets, KC2a and KC2b generated by partitioning with the mean and median values, respectively. Each of the datasets was input to the LOGISTIC procedure in SAS to generate the candidate predictors. We tuned the system to meet two sets of real-world requirements: maximize *Acc* and *Specificity (PD)*; and minimize *Effort* while maximizing *PD*.

The prediction results tuned for maximal *Acc* and *Specificity* are depicted in Figure 2. Experiments 1–10 are the results from KC2a and 11–13 are the results from KC2b. Fig. 2 indicates that different data treatments give different range of prediction accuracy. Data partitioned by the mean values have higher overall accuracy *Acc*, while data partitioned by the median tend to give higher defect detection rate, up to 91.5%.

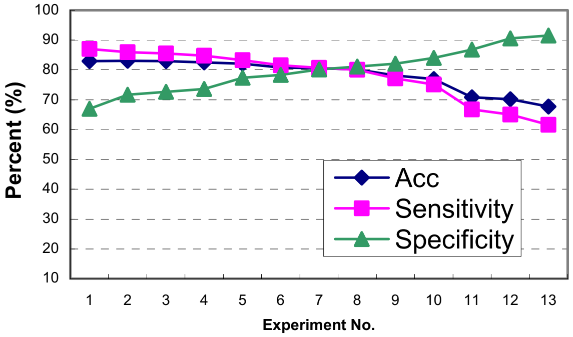

Figure 2. Prediction of fault prone modules by the DS networks

The prediction tuned for minimal *Effort* and maximal *PD* for KC2b is depicted in Figure 3. We could detect 91.5% of defects by reading 71.8% of the lines in source code. On the average, *PD* is higher than *Effort* by 18.2% on KC2b.

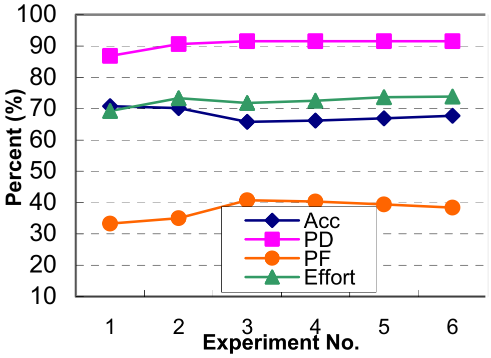

Figure 3. Prediction of fault prone modules by the DS networks on KC2b

From the record of the best sequences of the predictors, we found that, generally, using 2 to 4 predictors results in optimal prediction for KC2 project. The best combinations generally come from the top three candidates from the score tables.

6. Evaluation

6.1 DS Networks vs. Logistic Regression

For comparison, the LOGISTIC procedure in SAS was used as the classifier to predict fault prone modules for KC2. In order to compare our prediction accuracy with that of logistic regression, we picked the data points within the same range from the results of both methods. Figure 4 shows the comparison of software quality predictions obtained by these two methodologies.

The prediction by the D-S networks has overall higher accuracy. The defect detection rate (specificity) of the D-S networks is 1.9% to 5.7% higher than that of logistic regression. On the average, the defect detection rate of the D-S networks is 4.0% higher than that of logistic regression, while the overall accuracy *Acc* is 2.3% higher.

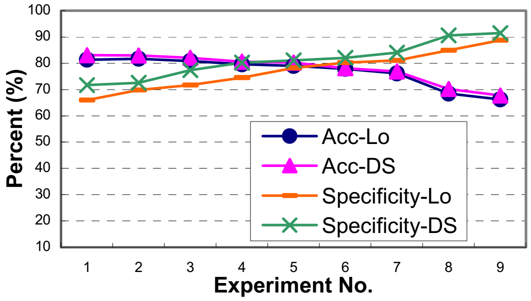

Figure 4. DS networks vs. logistic regression

6.2 DS Networks vs. Discriminant Analysis

The DISCRIM procedure in SAS (linear discriminant function) was used as the classifier on the KC2 dataset. The best predictors were selected by STEPDISC procedure in SAS (stepwise discriminant analysis).

The optimal prediction of discriminant analysis is compared with that of the D-S networks in Figure 5. These two methods have the same accuracy of 83.1%. However, the defect detection rate (*Specificity*) of the D-S networks is 4.7% higher than that of discriminant analysis. Considering that the cost to release a defect into the later phase of the software life cycle caused by imprecise prediction of fault-prone modules is higher than the cost of software inspection, the D-S networks do have an advantage over discriminant analysis.

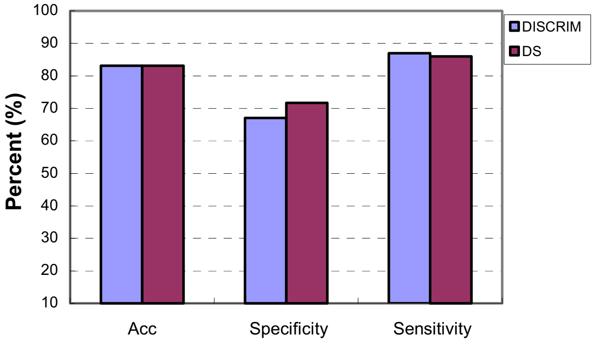

Figure 5. DS networks vs. discriminant analysis

6.3 DS Networks vs. ROCKY

ROCKY is a defect detector toolset experimentally used for the selection of modules for software inspection at NASA IV&V facility [16]. Its main purpose is to facilitate minimal inspection effort (recommend inspecting the minimal number of code lines) while achieving as good as possible defect detection rate *PD*.

Compared with the optimal performance of ROCKY, we notice several advantages of D-S networks. For ROCKY, *Effort* is generally higher than *PD*, except for one or two data points. For D-S networks, *Effort* is generally lower than *PD* for the entire data range. Especially significant advantage of D-S networks can be seen in Figure 3 depicting the effort on KC2b. As mentioned earlier, inspecting approximately 72% of the code (LOC) would expose over 91% of the fault prone modules. In contrast, it required ROCKY to read 94% of code to discover 91% of fault prone modules. The comparison of the *Effort* resulting from D-S predictions and ROCKY toolset predictions are shown in Figure 6. Based on the available data [16], ROCKY predictions lead to higher levels of effort than D-S network predictions.

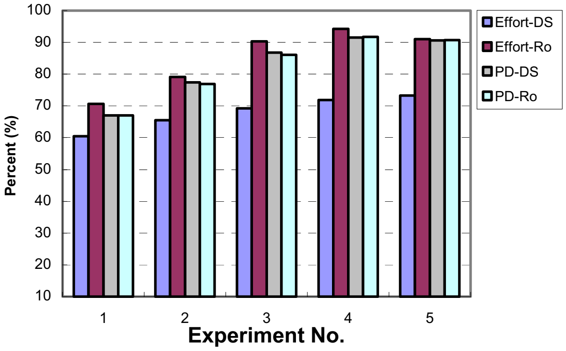

Figure 6. DS networks vs. ROCKY

7. Conclusions and Future Work

This paper contributes a novel methodology to the theoretical framework of software quality prediction. This methodology is based on the Dempster-Shafer belief networks. It is a general framework for prediction under uncertainty and can be applied to other research areas. In our case studies, its prediction accuracy is higher than logistic regression and discriminant analysis. It is also more effort economic than another software defect detector, ROCKY.

The methodology presented in this paper is meaningful for real-world applications in software quality prediction. It has a unique aspect that it can be tuned to meet different optimization criteria, such as to achieve maximal *Acc* and *PD*, or to use minimal *Effort* while achieving maximal *PD*. D-S networks can be tuned to meet new optimization requirements.

In this study, we observed that the sequence of the predictors taken into the D-S networks has effect on optimal network inference, which is consistent with the observation presented in [8]. Currently, we do not have an algorithm to identify the "magic" sequence. One possible research direction is to explore the relationship between the sequence of the predictors and the entropy (the measure of uncertainty) of the D-S network. If each time the predictor taken into the D-S network is the one that is most likely to reduce the entropy of the entire network, the algorithm to decide the sequence of the predictors is then an optimization algorithm. Another direction for improvement would be the discretization of the dataset into a multivariate, rather than binary variables.

Currently, we continue to compare our methodology with other software quality models and machine learning algorithms, and perform statistical significance tests on the comparisons as well.

References

[1] V. R. Basili, L. C. Briand, and W. Melo, "A validation of object-oriented design metrics as quality indicators", *IEEE Trans. Software Eng.* 22(10):751-761, Oct. 1996.

[2] L. C. Briand, V. R. Basili, and C. J. Hetmanski, "Developing Interpretable Models with Optimaized Set Reduction for Identifying High-Risk Software Components", *IEEE Trans. Software Eng.* 19(11):1028-1044, Nov. 1993.

[3] C. Ebert, "Classification techniques for metric-based software development", *Software Quality Journal*, 5(4):255-272, Dec. 1996.

[4] L. Guo, "Estimating Component Availability by Dempster-Shafer Networks", *Proc. Thirteenth International Symposium on Software Reliability Engineering*, Nov. 2002.

[5] M. Halstead, *Elements of Software Science*, Elsevier, 1977.

[6] D. K. Hildebrand, J. D. Laing and H. Rosenthal, *Prediction Analysis of Cross Classifications*, New York: John Wiley & Sons, 1977.

[7] T. M. Khoshgoftaar, and D. L. Lanning, "A neural network approach for early detection of program modules having high risk in the maintenance phase", *Journal of Systems and Software*, 29(1):85-91, Apr. 1995.

[8] J. Liu, D. Maluf and M. Desmarais, "A New Uncertainty Measure for Belief Networks with Applications to Optimal Evidential Inferencing", *IEEE Trans.Knowledge and Data Eng.* Vol. 13, No. 3, May/June 2001.

[9] J. Liu and M. C. Desmarais, "A Method of Learning Implication Networks from Empirical Data: Algorithm and Monte-Carlo Simulation-Based Validation", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 6, Nov./Dec. 1997.

[10] McCabe and Associates, "Software metrics: McCabe metrics", http://www.mccabe.com/metrics.php, 2003.

[11] J. C. Munson and T. M. Khoshgoftaar, "The detection of fault-prone programs", *IEEE Trans. Software Eng.* 18(5):423-433, May 1992.

[12] http://www.sas.com, 2003.

[13] N. F. Schneidewind, "Methodology For Validating Software Metrics", *IEEE Trans. Software Eng.* 18(5):410-422, May 1992.

[14] R. W. Selby and A. A. Porter, "Learning from examples: Generation and evaluation of decision trees for software resource analysis", *IEEE Trans. Software Eng.* 14(12):1743-1756, Dec. 1988.

[15] G. Shafer, *A Mathematical Theory of Evidence*, Princeton University Press, 1976.

[16] T. Menzies, J. D. Stefano, K. Ammar, R. M. Chapman, K. McGill, P. Callis, J. Davis, "When Can We Test Less?", http://menzies.us/pdf/03metrics.pdf, 2003.