Evaluating Empirical Models for the Detection of High-Risk Components: Some Lessons Learned

Filippo Lanubile

Department of Computer Science
University of Maryland
email: lanubile@cs.umd.edu

Giuseppe Visaggio

Dipartimento di Informatica,
Università di Bari, Italy
email: visaggio@seldi.uniba.it

1. Introduction

Software complexity metrics are often used as indirect metrics of reliability since they can be obtained relatively early in the software development life cycle. Using complexity metrics to identify high-risk components, i.e. components which likely contain faults, allows software engineers to focus the verification effort on them, thus achieving a reliable product at a lower cost. Since in this study the direct metric of reliability is the class to which the software component belongs (high-risk or low-risk), the prediction problem is reduced to a classification model.

Classification problems have traditionally been solved by various methods, which originate from different problem-solving paradigms such as statistical analysis, machine learning, and neural networks. This study compares different modeling techniques which cover all the three classification paradigms: principal component analysis, discriminant analysis, logistic regression, logical classification models, layered neural networks, and holographic networks. A detailed description of our implementation choices in building the classification models can be found in [LLV95].

2. Data Description

Raw data were obtained from 27 projects performed in a software engineering course at the University of Bari, by different three student-teams over a period of 4-10 months. The systems, business applications developed from a same specification, range in size from 1100 to 9400 lines of Pascal source code. There were between 120 and 340 software components in each system, ranging in size from 60 to 530 lines of code. Here, the term software component refers to functional abstractions of code such as procedures, functions and main programs.

From each system, we randomly selected a group of 4-5 components for a total of 118 components. Each group of component was tested by independent student teams of an advanced software engineering course with the aim to find faults. In order to build unbiased classification models, we decided to have an approximately equal number of components in the classes of reliability. Thus, we defined as high-risk any software component where faults were detected during testing, and low-risk any component with no faults discovered.

In addition to the fault data, 11 software complexity metrics were used to construct the classification models: McCabe's cyclomatic complexity (*v(G)*), Halstead's number of unique operands (η_2), Halstead's total number of operands (N_2), total number of lines of code (*LOC*), number of non-comment lines of code (*NCLOC*), Halstead's program length (*N*), Halstead's volume (*V*), Henry&Kafura's fan-in (*fanin*), Henry&Kafura's fan-out (*fanout*), Henry&Kafura's information flow (*IF*), and density of comments (*DC*).

The metrics have been selected so as to measure both design and implementation attributes of the components, such as control flow structure (metric 1), data structure (metrics 2-3), size (metrics 4-7), coupling (metrics 8-10), and documentation (metric 11). Most of these metrics have been already used in other empirical studies to test predictive models with respect to faults [MK92, LK94], and program changes [KLM93, KS94, LK94].

The set of 118 observations was subsequently divided into two groups. Two thirds of the components, made up of 79 observations, were randomly selected to create and tune the predictive models. The remaining 39 observations provided the data to test the models and compare their performances. From now on, the first group of observations will be called training set, while the second one testing set.

3. Evaluation Criteria

We selected statistical criteria which are based on the analysis of categorical data. In our study we have two variables (*real risk* and *predicted risk*) that can assume only two discrete values (*low* and *high*) in a nominal scale. Then the data can be represented by a two-dimensional contingency table with one row for each level of the variable *real risk* and one column for each level of the variable *predicted risk*. The evaluation criteria are predictive validity, misclassification rate, achieved quality and verification cost.

The predictive validity is the capability of the model to predict the future component behavior from present and past behavior. The present and past behavior is represented by data in the training set while the future behavior of components is described by data in the testing set. In our context, where data are represented by a contingency table, we apply the predictive validity by testing the null hypothesis of no association between the row variable (*real risk*) and the column variable (*predicted risk*). In this case, the predictive model is not able to discriminate low-risk components from high-risk components. The alternative hypothesis is one of general association. A chi-square (χ^2) statistic with a distribution of one degree of freedom is applied to test the null hypothesis.

We use the criterion of predictive validity for assessment, since we determine the absolute worth of a predictive model by looking at its statistical significance. A model which does not meet the criterion of predictive validity should be rejected. The remaining criteria are used for comparison, taking into account that the choice between the accepted models depends from the perspective of the software engineering manager. In practice he could be more interested in achieving a better quality at a high verification cost or be satisfied of a lower quality, sparing verification effort.

For our predictive models, which classify components as either low-risk or high-risk, two misclassification errors are possible. A Type 1 error is made when a high-risk component is classified as low-risk, while a Type 2 when a low-risk component is classified as high-risk. It is desirable to have both types of error small. However, since the two types of errors are not independent, software engineering managers should consider their different implications. As a result of Type 1 error, a component actually being high-risk could pass the quality control. This would cause the release of a lower quality product and more fix effort when a failure will happen. As a result of Type 2 error, a component actually being low-risk will receive more testing and inspection effort than needed. This would cause a waste of effort. We adopt from [Sch94], as measures of misclassification, the proportions of Type 1, Type 2, and Type 1 + Type 2 errors.

We are interested in measuring how effective are the predictive models in terms of the quality achieved after that the components classified as high-risk have been undergone to a verification activity. We suppose that the verification will be so exhaustive to find the faults of all the components which are actually high-risk. We measure this criterion using the completeness measure [BTH93], which is the percentage of faulty components that have been actually classified as such by the model.

Quality is achieved by increasing the cost of verification due to an extra effort in inspection and testing for the components which have been flagged as high-risk. We measure the verification cost by using two indicators. The former, inspection [Sch94], measures the overall cost by considering the percentage of components which should be verified. The latter, wasted inspection, is the percentage of verified components which do not contain faults because they have been incorrectly classified.

4. Analysis of the Results

We applied the evaluation criteria on the testing set and analyzed the resulting data, shown in Table 1. The first two columns show the chi-square values and the significance levels across the classification models built using different modeling techniques. From the low values of chi-square and high significance levels in the testing set, we accept the null hypothesis of no association between predicted risk and real risk. In fact, all the values of significance are too high with respect to the most common values which are used to reject the null hypothesis.

As regards the misclassification rate, we recall that a casual prediction should have 50 percent of proportion of Type 1 + Type 2, and 25 percent for both proportion of Type 1 and Type 2. From data showing the misclassification rates, we see that the proportion of Type 1 + Type 2 error ranges between 46 percent and 59 percent. Discriminant analysis and logistic regression, when applied in conjunction with principal component analysis, have a high proportion of Type 2 error (respectively 41 and 46 percent) with respect to the proportion of Type 1 error (respectively 15 and 13 percent). On the contrary, the other models have balanced values of Type 1 and Type 2 error, ranging between 20 and 28 percent.

	Predictive Validity		Misclassification Rate			Achvd Quality	Verification Cost	
Modeling Techniques	χ^2	α	P_1	P_2	P_{12}	C	I	WI
Discriminant anal.	0.244	0.621	28.21	25.64	53.85	42.11	46.15	55.56
Discriminant anal. + Principal cmpnt.	0.685	0.408	15.38	41.03	56.41	68.42	74.36	55.17
Logistic regression	0.648	0.421	28.21	28.21	56.41	42.11	48.72	57.89
Logistic regression + Principal cmpnt	1.761	0.184	12.82	46.15	58.97	73.68	82.05	56.25
Logical classif. model	0.215	0.643	25.64	20.51	46.15	47.37	43.59	47.06
Layered neural ntwrk	0.648	0.421	28.21	28.21	56.41	42.11	48.72	57.89
Holographic ntwrk	0.227	0.634	25.64	28.21	53.85	47.37	51.28	55.00

Table 1. Results of evaluation criteria

Looking at the achieved quality and the verification cost, we can interpret better the misclassification results. In fact, the highest values of quality correspond to the models built with principal component analysis followed from either discriminant analysis or logistic regression (completeness is, respectively, 68 and 74 percent). But these high values of quality are obtained by inspecting the great majority of components (inspection is, respectively, 74 and 82 percent), thus wasting more than one half of the verification effort (wasted inspection is, respectively, 55 and 56 percent). The lowest level of quality (completeness is 42 percent) is achieved by both discriminant analysis and logistic regression, when used without principal component analysis, and by the layered neural network. The logistic regression without principal components and the layered neural network have also the poorest results in correctly identifying the high-risk components (wasted inspection is 58 percent). On the contrary the logical classification model is the only model which dissipates less than one half of the verification effort (wasted inspection is 47 percent).

5. Lessons learned

This empirical investigation of the modeling techniques for identifying high-risk modules has taught us three lessons:

- Predicting the future behavior of software products does not always lead to successful results. Despite of the variegated selection of modeling techniques, no model satisfies the criterion of predictive validity, that is no model is able to discriminate between components with faults and components without faults. This result is in contrast with various papers which report successful results in recognizing fault-prone components from analogous sets of complexity measures.

Briand *et al.* [BBH93] presented an experiment for predicting high-risk components using two logical classification models (Optimized Set Reduction and classification tree) and two logistic regression models (with and without principal components). Design and code metrics were collected from 146 components of a 260 KLOC system. OSR classifications were found to be the most complete (96 percent) and correct (92 percent), where correctness is the complement of our wasted inspection. The classification tree was more complete (82 percent) and correct (83 percent) than logistic regression models. The use of principal components improved the accuracy of logistic regression, from 67 to 71 percent of completeness and from 77 to 80 percent of correctness.

Porter [Por93] presented an application of classification trees to data collected from 1400 components of six FORTRAN projects in NASA environment. For each component, 19 attributes were measured, capturing information spanning from design specifications to implementation. He measured the mean accuracy across all tree applications according to completeness (82 percent) and to the percentage of components whose target class membership is correctly identified (72 percent), that is the complement of the Proportion of Type 1 and Type 2 error.

Munson and Koshgoftaar [MK92] detected faulty components by applying principal component analysis and discriminant analysis to discriminate between programs with less than five faults and programs having 5 or more faults. The data set included 327 program modules from two distinct Ada projects of a command and control communication system. They collected 14 metrics, including Halstead’s metrics together with other code metrics. Applying discriminant analysis with principal components, at a probability level of 80 percent, resulted in recognizing 79 percent of the modules with a total misclassification rate of 5 percent.

Our result is closer with the investigation performed by Basili and Perricone [BP84], where the unexpected result was that module size and cyclomatic complexity had no relationship with the number of faults, although there was a negative relationship with the fault density.

- Predictive modeling techniques are only as good as the data they are based on. The relationship between software complexity measures and software faults cannot be considered an assumption which holds for any data set and project. A predictive model, from the simplest to the most complex, is worthwhile only if there is a local process to select metrics which are valid as predictors.

- Principal component analysis does not always produce a better input for predictive models. The domain metrics have often been used in the software engineering field [BBH93, BTH93, MK92, KLM93] to reduce the dimensions of a metric space when the metrics have a strong relationship between them, and obtain a smaller number of orthogonal domain metrics to be used as input to regression and discriminant analysis models. In our study, we built two classification models for both discriminant analysis and logistic regression. The first couple of models was based on the eleven original complexity measures, while the second one used the three domain metrics which had been generated from the principal component analysis. An unexpected result of the models using orthogonal domain metrics is that the good performance in achieved quality is exclusively the result of classifying very often components to be high-risk.

References

[BTH93] L. C. Briand, W. M. Thomas, and C. J. Hetmanski, “Modeling and managing risk early in software development”, in Proceedings of the 15th International Conference on Software Engineering, Baltimore, Maryland, May 1993, pp.55-65.

[BBH93] L. C. Briand, V. R. Basili, and C. J. Hetmanski, “Developing interpretable models with optimized set reduction for identifying high-risk software components”, *IEEE Transactions on Software Engineering,* vol.19, no.11, November 1993, pp.1028-1044.

[KLM93] T. M. Khoshgoftaar, D. L. Lanning, and J. C. Munson, “A comparative study of predictive models for program changes during system testing and maintenance”, in *Proceedings of the Conference on Software Maintenance,* Montreal, Canada, September 1993, pp.72-79.

[KS94] T. M. Khoshgoftaar, and R. M. Szabo, “Improving code churn prediction during the system test and maintenance phases”, in *Proceedings of the International Conference on Software Maintenance,* Victoria, British Columbia, Canada, September 1994, pp.58-67.

[LK94] D. L. Lanning, and T. M. Khoshgoftaar “Canonical modeling of software complexity and fault correction activity”, in *Proceedings of the International Conference on Software Maintenance,* Victoria, British Columbia, Canada, September 1994, pp.374-381.

[LLV95] F. Lanubile, A. Lonigro, and G. Visaggio, “Comparing models for identifying fault-prone software components”, in *Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering, Rockville, Maryland, USA, June 1995, pp.312-319.*

[MK92] J. C. Munson, and T. M. Khoshgoftaar, “The detection of fault-prone programs”, *IEEE Transactions on Software Engineering,* vol.18, no.5, May 1992, pp.423-433.

[Por93] A. A. Porter, “Developing and analyzing classification rules for predicting faulty software components”, in *Proceedings of the 5th International Conference on Software Engineering and Knowledge Engineering,* San Francisco, California, June 1993, pp.453-461.

[Qui93] J. R. Quinlan, *C4.5: Programs for Machine Learning,* Morgan Kauffman Publishers, San Mateo, CA, 1993.

[Sch94] N. F. Schneidewind, “Validating metrics for ensuring Space Shuttle Flight software quality”, *Computer,* August 1994, pp.50-57.