An Investigation on the Use of Machine Learned Models for Estimating Correction Costs

Mauricio A. de Almeida[1] and Hakim Lounis

Centre de Recherche Informatique de Montréal
1801, McGill College Ave., #800
Montréal, H3A 2N4 Qc, Canada
{mdealmei, hlounis}@crim.ca
phone: (1) (514) 840-1234

Walcelio L. Melo

Oracle do Brasil
SCN Qd. 02 - Bl. D - Torre A - Salas 501/506
Brasilia, DF Brazil 70710-500
wmelo@br.oracle.com
phone/fax: (55) (61) 327-3027

1. Guest researcher at CRIM and assistant professor at Faculdade de Tecnologia de Sao Paulo, Sao Paulo , Brazil.

ABSTRACT

In this paper we present the results of an empirical study in which we have investigated Machine Learning (ML) algorithms with regard to their capabilities to accurately assess the correctability of faulty software components. Three different families algorithms have been analyzed: Top Down Induction Decision Tree, covering, and Inductive Logic Programming (ILP). We have used (1) fault data collected on corrective maintenance activities for the Generalized Support Software reuse asset library located at the Flight Dynamics Division of NASA's GSFC and (2) product measures extracted directly from the faulty components of this library.

1.0 INTRODUCTION

Software maintenance consumes most of the resources in many software organizations. We must be able to better characterize, assess, and improve the maintainability of software products in order to decrease maintenance costs. Maintenance involves activities such as correcting errors, migrating software to new technologies, and adapting software to deal with new environment requirements.

Corrective maintenance is the part of software maintenance devoted to correcting errors. Mostly, when software maintainers have to correct a faulty software component, they rely almost exclusively on their previous experience in order to estimate the effort they will spend to do it. Even though highly experienced software maintainers may make accurate predictions, the estimation process remain informal, error-prone, and poorly documented, making it difficult to replicate and spread throughout the organization.

Published in the Proc. of the 21th Int’l Conf. on S/W Eng. Kyoto, Japan, 1997.

In general, software maintenance organizations tend to assign corrective maintenance activities to young software engineers who do not know a great deal about software systems they have to maintain.

In order to improve corrective maintenance, we must be able to provide models which help software maintainers better assess the maintainability of software products and estimate corrective maintenance effort. The benefits of having such models for software maintenance are numerous. For instance, estimation models can help us optimize the allocation of resources to corrective maintenance activities. Evaluation models can help us made decisions about when to re-structure or re-engineer a software component in order to make it more maintainable. Understanding models can help us know better the underlying reasons about the difficulty of correcting specific kinds of errors.

Many different approaches have been proposed to build corrective maintenance estimation/evaluation models. In this paper, we show the results of an empirical study in which we have investigated different ML algorithms with regard to their capabilities to generate accurate and easily interpretable correctability models. We have compared these algorithms with regard to their capabilities to assess the difficulty of correct Ada faulty components. The results show that ML algorithms are able to generate adequate prediction models. The rules produced by the ML algorithms can also be used as coding guidelines. In addition, the rules generated by these algorithms showed to be intuitive to software maintainers.

2.0 MACHINE LEARNING ALGORITHMS

Most of the work done in machine learning has focused on supervised machine learning algorithms. Starting from the description of classified examples, these algorithms produce definitions for each class. In general, they use an attribute-value representation language that allows the use of statistical properties on the learning set. Nevertheless, others use the first order logic language. It has better expressive capabilities than the attribute-value language. It permits the expression of relations between objects. An important consequence is the diminution of the learning data-set size. Both are helpful for constructing efficient software quality models. The following table summarizes the four ML algorithms we have used.

TABLE 1. ML algorithms used in the study

ML algorithms	Algorithm family	Description language	Induced knowledge
NewID [4]	Divide & conquer family: Top Down Induction Decision Tree -TDIDT-	Attribute-value	Decision tree
CN2 [7]	Covering family	Attribute-value	Rules
C4.5 [13]	Divide & conquer family: -TDIDT-	Attribute-value	Decision tree & rules
FOIL [14]	Inductive Logic Programming -ILP-	First order logic	Clauses

3.0 STUDY OVERVIEW

3.1 The studied environment

In this study, we have used data from the maintenance of a library of reusable components. This library, known as the Generalized Support Software (GSS) reuse asset library, is located at the Flight Dynamics Division (FDD) of NASA's Goddard Space Flight Center (GSFC). Component development began in 1993. Subsequent efforts focused on generating new components to populate the library and on implementing specification changes to satisfy mission requirements. The first application using this library was developed in early 1995. The asset library currently consists of 1K Ada83 components totalling approximately 515 KSLOC.

3.2 Data Collection

In this study, we collected error and fault data about this library. An *error* is represented by a single software Change Request Form (CRF) [10] filled by developers and configurers to institute and document a change to one or more components. A *fault* pertains to a single component and is evidenced by the physical change of that component in response to a particular error CRF. In this study, we have only used those components representing Ada 83 files. A *faulty* component version becomes a *fixed* component version after it is corrected. We are only interested in the Ada faulty component versions.

For each CRF, we have collected data on: (1) error identification and error correction, including the names and version numbers of the Ada source code components that had faults in them, (2) the effort expended to isolate all faults associated with the error, (3) the effort required to correct all of these faults, and (4) source code metrics characterizing these particular components. The ASAP tool [1] was used to extract source code metrics from the Ada faulty component versions.

3.3 Dependent and independent variables

In our study, the dependent variable is the total effort spent to isolate and correct a faulty component. Isolation and correction effort at NASA SEL is measured on a 4-point ordinal scale: 1 hour, from 1 hour to 1 day, from 1 to 3 days, and more than 3 days. To build the classification model, we have dichotomized the corrective maintenance cost into two categories: *low* and *high* correct maintenance cost. To do so, we converted the four effort categories into average values following [3]. We assumed an 8 hour day, and took the average value for each of the categories of corrective maintenance effort. Therefore, the category of "1 Hour" was changed to 0.5 hours, the category of "1 hour to 1 Day" was changed to 4.5 hours, the category of "from 1 to 3 Days" was changed to 16 hours, and the category of "more than 3 Days" was changed to 32 hours. We then summed up these values for isolation and correction costs. This gives us an average overall corrective maintenance cost. We used the median of total corrective maintenance cost as the cutoff point for dichotomization.

In this study, the independent variables are the ASAP product measures extracted from the faulty components.

3.4 Evaluating Prediction Accuracy

In order to evaluate the model, we need formal measures for evaluating the classification performance of the estimation models produced by the different ML algorithms. In this paper, we have used five criteria (see Table 2): sensitivity, specificity, predictive value (+), predictive value (-), and accuracy. These are defined below with reference to Table 3. In addition, we have used a measure of prediction validity as it was presented in [11] and used in [2]. It means that if the statistical significant coefficient *p-value* of the computed value of the X^2 test is less than 0.05 then we can say that the generated model has predictive validity.

TABLE 2. Formal measures of classification performance [12]

Sensitivity	$n_{11} / (n_{11}+n_{21})$
Specificity	$n_{22} / (n_{12} + n_{22})$
Predictive value (+)	$n_{11} / (n_{11} + n_{12})$
Predictive value (-)	$n_{22} / (n_{21} + n_{22})$
Accuracy	$(n_{11} + n_{22}) / ((n_{11}+n_{21})+ (n_{12} + n_{22}))$

All the criteria above are expected to be as high as possible, because when they are low, it will lead to a wrong allocation of resources to maintain the components.

TABLE 3. Two-class classification performance matrix

		Predicted Cost	
		High Cost	Low Cost
Real Cost	High Cost	n_{11}	n_{12}
	Low Cost	n_{21}	n_{22}

In order to calculate the values of the formal measures of classification performance as described in Table 2, we used a V-fold cross-validation procedure [5]. For each observation X in the sample, a model is developed based on the remaining observations (sample - X). This model is then used to predict whether observation X will be classified as either costly or not costly. This validation procedure is commonly used when data sets are small, e.g. [2] and [6].

4.0 RESULTS

4.1 Data preparation

The data we have used in this study is a set of 164 Ada faulty components classified as having either 'high' or 'low' corrective maintenance cost. The data was structured as a sequence of attribute-value pairs containing 19 attributes, corresponding to the independent variables and 1 attribute associated to the dependent variable.

4.2 Quantitative comparison

Table 4 presents the quantitative results of the study:

TABLE 4. Results of the study

	NewID	CN2	C4.5	C4.5_rules	FOIL
Sensitivity	53%	56%	70%	74%	80%
Specificity	50%	53%	62%	64%	68%
Predictive-value(+)	55%	58%	59%	59%	65%
Predictive-value(-)	48%	51%	73%	77%	82%
Accuracy	52%	54%	66%	68%	73%
X^2	0.1898	1.1289	17.34	21.91	37.08
p-value	=0.66	=0.2854	<=0.000	<=0.0000	<=0.000

From the point of view of prediction validity measures, the models generated by NewID and CN2 are not statistically significant (X^2 test p-value>0.05). All the other generated models are, from this point of view, statistically significant, since the p-values are less than 0.001 (far away from the threshold of 0.05).

As we can see in Table 4, FOIL presents the best results in our experiment. The model generated by FOIL is composed by 6 rules. Table 5 presents the metrics used in the experiment and their number of occurrences in the rules of the two best learned models.

The number of operands (N_2) appeared in 5 of the 6 generated rules and the number of operators (N_1) in 4. This results demonstrate that some of the Halstead metrics [9] in our data are useful for predicting the cost of corrective maintenance of faulty components. The two new metrics we have introduced, i.e., comments divided by size (*comments div size*) and blank lines divided by size (*blank lines div size*), have also been selected by FOIL: comments divided by size (*comments div size*) appeared twice and blank lines divided by size (*blank lines div size*) once.

TABLE 5. Metrics used in the experiment

Metrics	FOIL	C4.5_rules
Number of operands (N2)	5	0
Number of operators (N_1)	4	0
declarative	3	1
inline comments	3	0
Number of distinct operators (n_1)	3	0
blank lines	2	1
comments div size	2	0
cyclomatic complexity	2	2
lines of comments	2	0
total statement nesting depth	2	0
blank lines div size	1	3
executable	1	0
lines of code	1	0
maximum statement nesting depth	1	0
statements	1	0
total source lines	0	0
Ada language statements	0	0
average statement nesting depth	0	0
Number of distinct operands (n_2)	0	0

In fact, models built without using these two metrics were less accurate than the models we have shown in this section (Due to a lack of space, the models built without these two normalized metrics are not showed).

Another important result is that FOIL, an ILP ML algorithm based on a subset of first order logic description language, provides the best results for all the measures we have computed. The sensitivity and the predictive value (-) is pretty high (around 80%) and the overall accuracy is 5% higher than the second best algorithm (C4.5 rules). This means that in some cases one can allocate resources to corrective maintenance of faulty components with a 82% of confidence.

In a recent study where another set of metrics have been used, Basili and his colleagues [2] obtained similar results using C4.5 rules (sensitivity 76% and overall accurary 73%). Although the results are difficult to compare, since [2] have used a different data set and independent variables (i.e., software metrics), the results of our study demonstrated again that C4.5 rules have worked better than C4.5 decision trees. In addition, the results obtained with C4.5 rules on both studies are quite close (around 75%).

4.3 FOIL rules

FOIL is able to built predictive software models via rules expressed in first order logic. The greatest advantage of FOIL rules is that we are able to compare measures instead of simply listing attribute-value rules. Here, we show one of the rules generated by FOIL taken arbitrarily to exemplify the rule’s interpretation.

> ***high(A)****:-executable(A,B), maximum_statement_nesting_depth(A,C), lines_of_comments(A,D),commentsdivsize(A,E), N1(A,F),N_2(A,G),less_or_equal(E,F), ~less_or_equal(B,G),C<>4,C<>43, less_or_equal(C,D)*

This rule can be read as:

> *“a faulty component has a high corrective maintenance cost* ***if*** *the comments density (#commentsLines / # source lines of code) is less or equal to number of Operators,* ***and*** *executable statements is greater than number of operands,* ***and*** *maximum_statement_nesting_depth is less or equal to the number of lines of comments,* ***and*** *the maximum statement nesting depth is different from 4 and 43”.*

The data used in this study as well as all decision trees and rules are available under request.

5.0 LESSONS LEARNED

In this paper, we have empirically investigated different machine learning techniques with regard to their capabilities to generate accurate correctability models. The results show that the inductive logic programming algorithms are superior to the top-down induction decision tree, top-down induction attribute value rules, and covering algorithms, i.e. the overall accuracy of the model build using FOIL was higher than the other algorithms. The rules provided by the inductive logic programming algorithm we have used, i.e., FOIL, showed to be meaningful.

As far as we know, we are one of the first to investigate the use of such ILP ML algorithms in the field of software engineering [8]. Most of the works done have exploited algorithms based on propositional logic. These latter are limited, in the sense that they can not induce models that compare a descriptor (i.e., metric or independent variable) to another.

The work we have presented addresses two different but complementary domains: software engineering and machine learning. It confirms the usefulness of collaboration between the two domains. With regard to software engineering, we intend to do the following work:

- The generation of other quality models, such as reliability, error-proneness, etc.
- The use of other set of measures which enriches the measures provided by ASAP.
- Replicate the study using other data sets.
- Provide guidelines which help software managers to take preventive action early in the process life-cycle.

ACKNOWLEDGEMENTS

The authors wish to thank Vic Basili from University of Maryland -SEL- and Steven Condon from CSC for providing the data used in the paper. We are also grateful to R. Tesoriero and P. Mackenzie for their feedback on the early versions of this paper. During this work, W Melo was in part, supported by the Software Quality Group of Bell Canada, and by NSERC operation grant #OGP 0197275.

6.0 REFERENCES

[1] Amadeus Software Research Inc. “*Getting Started with Amadeus*”. Amadeus Measurement System. 1994.

[2] V. Basili, Condon, K. El Emam, R. B. Hendrick, W. L. Melo. “*Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components*”. In Proc. of the IEEE 19th Int’l. Conf. on S/W Eng., Boston, MA, May 1997.

[3] V. Basili and B. Perricone. “*Software Errors and Complexity: An Empirical Investigation*”. In CACM, 27(1):42-52, January 1984.

[4] R. Boswell. “*Manual for NewID*”. The Turing Institute, January 1990.

[5] L. Breiman, J. Friedman, R. Olshen and C. Stone. “*Classification and Regression Trees*”. Published by Wadsworth, 1984.

[6] L. Briand, V. Basili, C. Hetmanski. “*A Pattern Recognition Approach for Software Engineering Data Analysis*”. In IEEE TSE, 18(1), Nov. 1992.

[7] P. Clark & T. Niblet. “*The CN2 induction algorithm*”. In Machine Learning Journal, 3, p 261-283.

[8] W. W. Cohen & P. Devanbu. “*A Comparative Study of Inductive Logic Programming Methods for Software Fault Prediction*”. Technical Report AT&T Labs-Research, 1996.

[9] M. Halstead. “*Elements of Software Science*”. North-Holland, Amsterdam, 1977.

[10] G. Heller, J. Valett and M. Wild. “*Data Collection Procedure for the Software Engineering Laboratory (SEL) Database*”. Technical Report SEL-92-002, Software Engineering Laboratory, 1992.

[11] F. Lanubile and G. Visaggio. “*Evaluating Predictive Quality Models Derived from Software Measures: Lessons Learned*”. Technical Report ISERN-96-03, International Software Engineering Research Network, 1996.

[12] S. M. Weiss, C. A. Kulikowski. “*Computer Systems That Learn*”. Morgan Kaufmann Publishers, Inc. Sao Francisco, CA. 1991.

[13] J. R. Quinlan. “*C4.5: Programs for Machine Learning*”. Morgan Kaufmann Publishers, Sao Mateo, CA, 1993.

[14] J.R. Quinlan. “*Learning Logical Definitions from Relations*”. In machine learning journal, vol 5, n°3, p 239-266, August 1990.