Searching for Rules to find Defective Modules in Unbalanced Data Sets

D. Rodríguez
Dept. of Computer Science
University of Alcalá
Alcalá de Henares, Madrid, Spain
e-mail: daniel.rodriguezg@uah.es

J.C. Riquelme
Dept. of Computer Science
University of Seville
Seville, Spain
e-mail: riquelme@us.es

R. Ruiz, J.S. Aguilar-Ruiz
Dept. of Computer Science
Pablo de Olavide University
Seville, Spain
e-mail: {robertoruiz, aguilar}@upo.es

Abstract

The characterisation of defective modules in software engineering remains a challenge. In this work, we use data mining techniques to search for rules that indicate modules with a high probability of being defective. Using data sets from the PROMISE repository[1]*, we first applied feature selection (attribute selection) to work only with those attributes from the data sets capable of predicting defective modules. With the reduced data set, a genetic algorithm is used to search for rules characterising modules with a high probability of being defective. This algorithm overcomes the problem of unbalanced data sets where the number of non-defective samples in the data set highly outnumbers the defective ones.*

1. Introduction

The characterization of defective software modules, which has been addressed from different perspectives, remains a challenge in the software engineering field. Recently there is an increasing interest in applying search-based techniques to relevant aspects, such as effort estimation, defect prediction and maintainability [3]. With the availability of repositories with actual data from software projects, such as the PROMISE repository [2], it is now possible to apply data mining techniques to learn from real data. In this work, we use a genetic algorithm as subgroup discovery technique to characterise defective modules from data sets contained in such repository. However, for this problem, we need to tackle two problems: (i) data sets are highly unbalanced and (ii) there are attributes (metrics) that are not relevant for predicting defective modules. The background is summarised as follows:

Feature Selection. A large number of attributes can also interfere in the data mining learning process. In theory the more attributes, the more information capable of discriminate defective modules. In practice, however, this does not hold as many attributes are redundant or irrelevant degrading the accuracy rate. The problem of feature selection received a thorough treatment in pattern recognition and data mining [7]. As stated previously, feature selection is used to identify the most relevant attributes from a data set and to remove those redundant and/or irrelevant attributes that have negative effect on the data mining learning algorithm. Feature selection is part of the data preparation phase (preprocessing) to generate a reduced data set which can be useful in different aspects:

- A reduced volume of data facilitates the application of different data mining or searching techniques to be applied. Furthermore, data mining algorithms can be executed faster with smaller data sets.
- Irrelevant and redundant attributes can generate less accurate and more complex models which are harder to understand.
- Knowing which data is redundant or irrelevant can be used to avoid data collection of those attributes in the future so that the data collection becomes more efficient and less costly.

Unbalanced Data Sets. Most data sets in defect prediction are highly unbalanced, i.e., samples of non-defective modules vastly outnumber the cases of defective modules. This is a problem that affects most data mining learning algorithms which aim at classifying correctly the maximum number of instances assuming that data are balanced. For example, a model which always selects the majority class when, for example this class represents 90% of the samples, already produces very good results so modifications to the algorithm to improve the accuracy results are generally discarded. With unbalanced data sets, data mining learning algorithms produce degenerated models that do not take into account the minority class as most data mining algorithms assume balanced data sets. When this happens, there are two alternatives, either (i) to apply algorithms that are robust to unbalanced data sets or (ii) balance the data using sampling techniques before applying the data mining algorithm .

Subgroup Discovery. Initially proposed by Klösgen [6] and Wrobel [9], [10], subgroup discovery are a set of algorithms that extract rules or patterns for subsets of the data of a previously specified the concept, for example defective modules in this work. The idea is to search for properties of

1. `http://promisedata.org/`

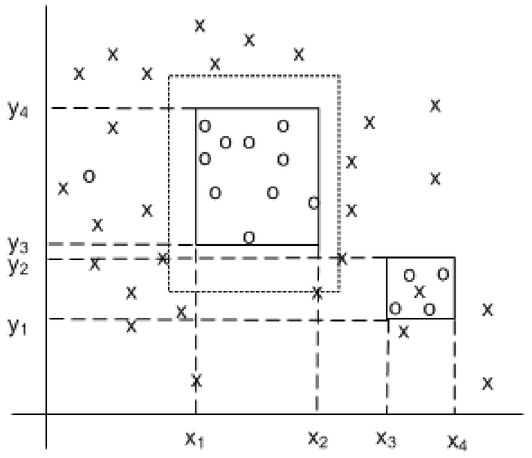

Figure 1. Examples of Rules with 2 Dimensions

subgroups with different behaviour in relation to the rest of the data. Rules for subgroup discovery have also the "*Condition* → *Class*" where the *condition* is the conjunction of a set of selected variables (pairs attribute–value) among all variables. In this work, we have used an evolutionary algorithm for subgroup discovery adapted from an algorithm to discover hierarchical rules [1]. This algorithm generates rules covering samples representing defective modules (the minority class). In this way, the algorithm is able to handle unbalanced data sets. Figure 1 shows an example of rules generated in this work with 2 dimensions. With the condition $x_1 \leq X \leq x_2$ AND $y_3 \leq Y \leq y_4$, we can consider the inner rectangle or the outer one. Also, instead of having too many rules covering perfect areas with no errors, we need a trade-off between the number of rules and number of erroneous instances included in those rules.

2. Experimental Work

In this paper, we have used the CM1, KC1, KC2, and PC1 data sets available in the PROMISE repository [2], to generate models for defect classification. These data sets were created from projects carried out at NASA and collected under their metrics[2]. Table 1 shows the number of instances (modules) for each data set together with the number of defective, non-defective and their percentage showing that all data sets are highly unbalanced varying from 7% to 20%. According to their Web site, the term module is applied to the lowest level functional unit from which metrics can b collected such as (functions, modules, or subroutines). The last attribute is the programming language used to develop those modules.

All data sets contain the same 22 attributes composed of 5 different lines of code measure, 3 McCabe metrics [8], 4

2. `http://mdp.ivv.nasa.gov/`

Table 1. Data Sets used in this Work

	instances	*Non-def*	*Def*	*% def*	*Lang*
CM1	498	449	49	9.83	C
KC1	2,109	1,783	326	15.45	C++
KC2	522	415	107	20.49	C++
PC1	1,109	1,032	77	6.94	C

base Halstead measures [5], 8 derived Halstead measures [5], a branch-count, and the last attribute is 'problems' with 2 classes (false or true, wether the module has reported defects). Table 2 summarizes the metrics collected from the data sets.

These sets of metrics (both McCabe and Halstead) have been used for quality assurance during (i) development to obtain quality measures, code reviews, etc., (ii) testing to focus and prioritize testing effort, improve efficiency, etc. and (iii) and maintenance as indicators such as comprehensibility of the modules or to detect error prone modules.

As stated previously, we are interested in rules for characterizing error prone modules. Generally, the developers or maintainers use rules of thumb or threshold values to keep modules, methods, etc within certain ranges. For example, if the cyclomatic complexity of a module is between 1 and 10, it is considered to have a very low risk; however, any value greater than 50 is considered to have an unmanageable complexity and risk. For the essential complexity (`ev(g)`), the threshold is 4, etc. Although these metrics have been used for long time, there are no clear thresholds for most of them and furthermore, they are open to interpretation. For example, although McCabe suggest a threshold of 10 for `v(g)`, NASA in-house studies of this metric concluded that a threshold of 20 is a better predictor of defective modules. Table 3 shows the number of times an attribute was selected by the feature selection algorithm as well as the range of the attribute and typical thresholds suggested in the literature.

As stated previously, the aim of this work is to search for rules that provide an indication of defective modules. To do so, we first performed a feature selection process to the data sets in order to simplify the input before applying the subgroup discovery algorithm (searching rule algorithm). In this work, we have used the Correlation-based Filter Selection [4] (CFS) as feature selection technique. The CFS is applied to the data before any other data mining algorithm and independently of them. This algorithm is in turn another searching algorithm that selects a set of attributes highly correlated with the class attribute and a low grade of redundancy between them. The CFS filter applied to the original CM1, KC1, KC2 and PC1 data sets are shown in Table 4.

Then, rules were generated using a genetic algorithm capable of handling unbalanced data sets. As a result, there is no need of applying any sampling techniques by either

Table 2. Attribute Definition Summary

	Metric	*Definition*
McCabe	loc	McCabe's Lines of code
	v(g)	Cyclomatic complexity
	ev(g)	Essential complexity
	iv(g)	Design complexity
Halstead base	uniq_Op	Unique operators, n_1
	uniq_Opnd	Unique operands, n_2
	total_Op	Total operators, N_1
	total_Opnd	Total operands N_2
Halstead derived	n	Vocabulary
	l	Program length
	v	Volume
	d	Difficulty
	i	Intelligence
	e	Effort
	b	Error Estimate
	t	Time estimator
	loCode	Count of statements
	loComment	Count of lines of comments
	loBlank	Count of blank lines
	loCodeAndComment	Code and comments
Branch	branchCount	No. branches
Class	false, true	Reported defects

Table 3. No. of Times Selected Attributes were Used by the Rules and Thresholds Suggested in the Literature

Attribute	*# of times*	Range	Threshold
loc	2	0-∞	60
iv(g)	4	1-v(g)	7
i	13	0-∞	120
loBlank	12	0-∞	10
uniq_op	3	0-∞	20
uniq_opnd	12	0-∞	20
v	1	0-∞	1,500
d	8	0-∞	30
loCode	8	0-∞	30
loComment	13	0-∞	10
branchCount	7	0-∞	19
ev(g)	3	1-v(g)	4
b	3	0-∞	0.60

Table 4. Attributes Selected for each Data Set

CM1	*KC1*	*KC2*	*PC1*
loc	v	ev(g)	v(g)
iv(g)	d	b	i
i	i	uniq_Opnd	loComment
loComment	loCode		loCodeAndComment
loBlank	loComment		loBlank
uniq_Op	loBlank		uniq_Opnd
uniq_Opnd	uniq_Opnd		
	branchCount		

increasing artificially the number of instances of the minority class artificially (in our case, number of cases with defective samples) or removing samples form the majority class. Also, the generated rules are simpler than the ones generated by other data mining techniques such as C4.5. This algorithm is a variant of another algorithm called HIDER [1]. The difference is that while HIDER generates rules for all the values of the class and those rules have to be applied hierarchically (rules have to be applied in order), the variant used in this work focuses only on one value of the class, i.e., the minority class[3].

The generated rules combine a set of attributes (metrics) to provide better estimation and explanation of defective modules. As it can be expected, most of the selected attributes were used by the rules and only in the case of the PC1 data set, the `v(g)` attribute was selected by the feature selection algorithm but not included in any of the generated rules. Table 3 shows the number of times that selected attributes were used in the condition of the rules to cover defective data sets as well as their possible range and thresholds suggested in the literature. The number of times an attributes are used reveals some interesting outcomes that need further research. For example, it seems quite natural that the intelligence of a module (i) or the number of unique operands influence the possibility of error prone modules, however it is less obvious that attributes such as $loBlank$ or $loComment$ (lines of blanks and lines of comments respectively) can be used as good predictors of defective modules.

Also, the disparity of the selected attributes in each data set may be the consequence of feature selection algorithm not being able to handle unbalanced data sets appropriately. The analysis of feature selection attributes under this condition is part of our future work. In any case, some attributes seem to be consistently more relevant than others as a predictors of defective modules.

Table 5 shows only the first rule for each data set which is the one covering the largest number of defective modules. The accuracy of each rule can be measured by the number of defective modules and non-defective modules covered by the rule space, taking into account that there is an upper limit to the number of defective modules. There is a trade-off between the number of rules, number of defective and non-defective samples covered by each rule.

In this work, in all four data sets around one third of the defective modules were covered by the rules and there is no defective samples covered by more than a single rule.

One problem observed, however, is that each rule seems to captures a small number of defective modules in proportion to the total number of them. If we consider that attributes of a data set, in a large n-dimensional space, rules cover small islands of points representing the minority class (defective

3. Implementation available at:
`http://www.ieru.org/wiki/index.php5?title=Software`

Table 5. Main Decision Rule for each Data Set

Data Set	*Rule Condition*
CM1	$37.27 \leq$ i AND $27 \leq$ loBlank ≤ 32 AND uniq_Op ≤ 32
KC1	$24{,}33 \leq$ d AND i ≤ 59.7 AND $24 \leq$ uniq_Opnd AND $12 \leq$ branchCount ≤ 31
KC2	$5 \leq$ ev(g) AND $37 \leq$ uniq_Opnd
PC1	$15 \leq$ loComment ≤ 71 AND $85 \leq$ uniq_Opnd

modules) which seem to be located sparsely and surrounded by non-defective modules. We considered a low value (5%) as an acceptable error when searching for rules; such value generated a large number of rules covering few instances. In software engineering, it could make more sense not to focus on the accuracy but to generate rules that are able to provide an indication of defective modules even if a rule covers as many defective modules as non-defective. The large number of rules and attributes for each rule make it hard to analyse and provide results for *human consumption*. The variance and number of selected attributes can also be affected by the fact that datasets are highly unbalanced; although our selected attributes are similar to others found in the literature for the same datasets, this needs further research.

3. Conclusions and Future Work

In this work, we have applied data mining techniques to characterize defective modules to four data sets from the PROMISE repository. To do so, we faced the problem that data sets are highly unbalanced, i.e., there are many more non-defective samples in the data sets than defective ones which conditions the range of techniques that we can apply and moreover, reduces the accuracy of the results. To solve this problem, we applied feature selection as a necessary step to reduce the data sets and then, as a subgroup discovery technique, a genetic algorithm as a subgroup discovery technique was used to generate rules for covering only defective modules.

Results showed that in general data sets are not very homogeneous in both the feature selection (attributes) selected in each data set or rules generated. The results, however, provide some points for further research. In relation to the rules we need to improve the algorithm to obtain a small number of simpler rules than the ones obtained. One approach can be to increase the percentage of error allowed for each rule. This will increase the number of instances covered per rule but also the error rate. In this way, rules will cover many more defective modules but rules will be simpler and could be used as indicators instead of almost certainty of being defective. We also need further research about the quality of the data sets. We found a large number of inconsistencies in the data sets used in this work. For example, datasets have replicated instances and contradictory instances (same values for all attributes but different class value). We believe that in general in software engineering datasets are of poor quality compared to other disciplines and this needs to be further analysed. Finally, we are only considering a binary class, it should be possible to consider more classes such as low, medium or high problematic modules.

Acknowledgements

Authors would like to thank to the anonymous reviewers for their useful comments. This research was supported by the Spanish Research Agency (TIN2007–68084–C02–00) and the Universities of Seville, Pablo de Olavide and Alcalá.

References

[1] Aguilar-Ruiz, J.S., Riquelme, J.C., Toro M., Evolutionary Learning of Hierarchical Decision Rules, *IEEE Transactions on Systems, Man and Cybernetics*, Part B, Vol 33, No. 2, pp. 324-331

[2] Boetticher G., Menzies T., Ostrand T.; PROMISE Repository of empirical software engineering data. (`http://promisedata.org/`), West Virginia University, Department of Computer Science (2007)

[3] Harman M., Jones B.F., Search-based software engineering, *Information & Software Technology*, Vol. 43, No. 14, pp. 833–839, 2001

[4] Hall M.A.; Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. In: 17th Int. Conf. on Machine Learning, 359–366 (2000)

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

[6] Klösgen, W., Explora: A Multipattern and Multistrategy Discovery Assistant. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padraic Smyth, and Ramasamy Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, pages 249-271. AAAI Press, 1996.

[7] Liu H. and Yu L., Toward Integrating Feature Selection Algorithms for Classification and Clustering, IEEE Trans. on Knowledge and Data Eng., 17(3), 1–12, 2005.

[8] McCabe, T. J.; A complexity measure, *IEEE Transactions on Software Engineering* 2 (4), 308–320, 1976.

[9] Wrobel, S. An algorithm for multi-relational discovery of subgroups. In J. Komorowski & J. Zytkow (Eds.), Proc. First European Symposion on Principles of Data Mining and Knowledge Discovery (PKDD-97) (pp. 7887). Springer Verlag, 1997.

[10] Wrobel, S. (2001). Inductive logic programming for knowledge discovery in databases. In S.Deroski & N. Lavra (Eds.), Relational data mining. Springer-Verlag.