Subgroup Discovery in Defect Prediction

Daniel Rodríguez[1], R Ruiz[2], JC Riquelme[3] and R Harrison[4]

[1]University of Alcalá, Madrid, Spain, *daniel.rodriguezg@uah.es*
[2]Pablo de Olavide University, Seville, Spain, *robertoruiz@upo.es*
[3]University of Seville, Seville, Spain, *riquelme@us.es*
[4]Oxford Brookes University, Oxford, UK, *rachel.harrison@brookes.ac.uk*

Introduction

- Subgroup Discovery (SD) algorithms aim to find subgroups of data (represented by rules) that are statistically different given a property of interest [3] and do not describe all instances in the dataset. They usually describe the minority class (the interesting one).
- We deal with the problem of software defect prediction through SD identifying software modules with a high probability of being defective.

SD Algorithms

In this work, we compare two well-known SD algorithms:

- The SD [2] algorithm is a covering rule induction algorithm that using beam search aims to find rules that maximise $q_g = \frac{TP}{FP+g}$, where TP and FP are the no. of true and false positives and g is a generalisation parameter to control the *specificity* of a rule.
- The CN2-SD [4] algorithm is an adaptation of the CN2 algorithm.It uses *WRAcc* as a measure of the quality of the induced rules.

Quality Measures

Table: Confusion Matrix for Two Classes

		Actual		
		Positive	Negative	
Prediction	Positive	True Positive (TP)	False Positive (FP) Type I Error False alarm	$Confidence = Precision = \frac{TP}{TP+FP}$
	Negative	False Negative (FN) Type II error	True Negative (TN)	
		$Recall = Sensitivity = TP_r = \frac{TP}{TP+FN}$	$Specificity = TN_r = \frac{TN}{FP+TN}$	

- *Coverage of a rule*, $Cov(R_i) = \frac{n(Cond)}{N} = p(Cond)$ where R_i is a single rule, $n(Cond)$ is the number of instances covered by condition *Cond* and N is the total number of instances.
- *Support*, $Sup(R_i) = \frac{n(Class \cdot Cond)}{N}$ where the $n(Class \cdot Cond)$ corresponds to the TP and N is the total number of instances.
- *Accuracy (Confidence)*, $Acc(R_i) = \frac{n(Class \cdot Cond)}{n(Cond)}$
- *Weighted Relative Acc*, $WRAcc(R_i) = \frac{n(Cond)}{N}\left(\frac{n(Class \cdot Cond)}{n(Cond)} - \frac{n(Class)}{N}\right)$
- *Significance*, $Sig(R_i) = 2 \cdot \sum_{k=1}^{n_c} n(Class_k \cdot Cond) \cdot log\frac{n(Class_k \cdot Cond)}{n(Class_k)}$ where n_c is the number of values of the target class.

Datasets

- PROMISE repository (CM1, KC1, KC2, KC3, MC2, MW1 and PC1)
- D'Ambros et al [1] repository (Equinox, Lucene and Eclipse PDE-UI)

Table: Description of the Datasets

DS	#	NonDef	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++
KC3	458	415	43	9.39	Java
MC2	161	109	52	32.29	C++
MW1	434	403	31	7.14	C++
PC1	1,109	1,032	77	6.94	C

Table: Description of the Datasets

DS	#	NonDef	Def	% Def	Lang
JDT Core	997	791	206	20.66	Java
PDE-UI	1,497	1,288	209	13.96	Java
Equinox	324	195	129	39.81	Java
Lucene	691	627	64	9.26	Java
Mylyn	1,862	1,617	245	13.15	Java

Table: McCabe and Halstead Metrics

	Metric	Definition
McCabe	*loc*	McCabe's Lines of code
	v(g)	Cyclomatic complexity
	ev(g)	Essential complexity
	iv(g)	Design complexity
Halstead	*uniqOp*	Unique operators, n_1
Base	*uniqOpnd*	Unique operands, n_2
	totalOp	Total operators, N_1
	totalOpnd	Total operands N_2
Branch	*brnchCnt*	Branches–flow graph
	Class	defects?

Table: OO Metrics - summary

Metric		Definition
C & K	*wmc*	Weighted Method Count
	dit	Depth of Inher. Tree
	cbo	Coupling Btwn Objects
	noc	No. of children
	lcom	Lack of Cohesion Methods
	rfc	Response For Class
	Class	defects?

Experimental Results

Table: Rules - KC2 with SD

pd	pf	TP	FP	Rules
.24	0	26	0	$ev(g) > 4 \wedge totalOpnd > 117$
.28	.01	30	5	$iv(G) > 8 \wedge uniqOpnd > 34 \wedge ev(g) > 4$
.27	.01	29	5	$loc > 100 \wedge uniqOpnd > 34 \wedge ev(g) > 4$
.27	.01	29	5	$loc > 100 \wedge iv(G) > 8 \wedge ev(g) > 4$
.27	.01	29	5	$loc > 100 \wedge iv(G) > 8 \wedge totalOpnd > 117$
.24	.01	26	5	$iv(G) > 8 \wedge uniqOp > 11 \wedge totalOp > 80$
.24	.01	26	5	$iv(G) > 8 \wedge uniqOpnd > 34$
.23	.01	25	5	$totalOpnd > 117$
.31	.01	34	5	$loc > 100 \wedge iv(G) > 8$
.29	.01	32	5	$ev(g) > 4 \wedge iv(G) > 8$
.29	.01	32	5	$ev(g) > 4 \wedge uniqOpnd > 34$
.28	.01	30	5	$loc > 100 \wedge ev(g) > 4$
.27	.01	29	5	$iv(G) > 8 \wedge totalOp > 80$
...	...	...	...	...

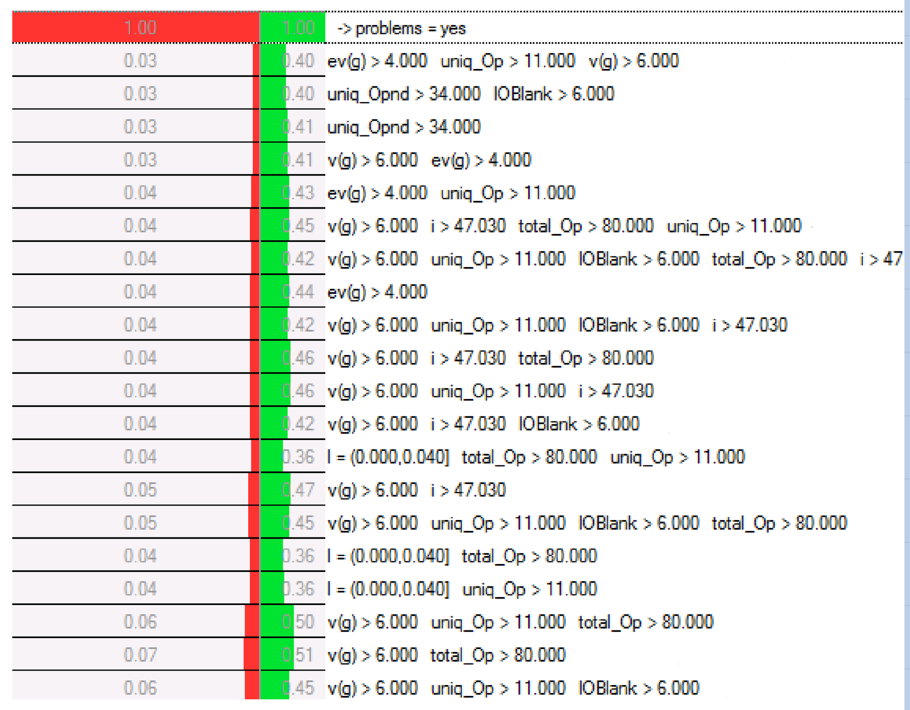

Figure: Bar Chart - KC2 with SD

Table: Rules KC2 - CN2-SD

pd	pf	TP	FP	Rules
.35	.01	38	5	$uniqOpnd > 34 \wedge ev(g) > 4$
.4	.02	43	9	$totalOp > 80 \wedge ev(g) > 4$
.78	.21	84	88	$uniqOp > 11$

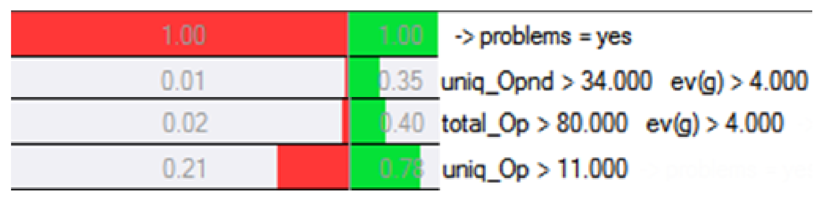

Figure: Bar Chart KC2 with CN2-SD

Table: 10 Cross-Validation Results

		COV	SUP	Size	Complex	SIG	WRAcc	ACC	AUC
SD	CM1	.23	.72	20	3.05	4.548	.029	.60	.75
	KC1	.08	.43	20	2.61	16.266	.023	.61	.66
	KC2	.08	.53	20	2.19	9.581	.049	.70	.74
	KC3	.29	.91	20	2.44	5.651	.037	.60	.83
	MC2	.16	.65	20	2.05	2.204	.042	.64	.69
	MW1	.07	.5	20	2.51	3.767	.02	.73	.68
	PC1	.12	.37	20	3.51	3.697	.01	.66	.62
CN2 SD	CM1	.11	.64	5	1.3	2.97	.023	.628	.62
	KC1	.11	.61	5	1.1	2.91	.03	.634	.71
	KC2	.16	.80	5	1.6	11.78	.065	.733	.82
	KC3	.13	.89	4.9	1.29	3.14	.019	.68	.80
	MC2	.15	.43	5	2.32	2.20	.04	.593	.59
	MW1	.08	.56	5	2.02	3.52	.02	.661	.74
	PC1	.09	.66	5	1.86	2.81	.007	.632	.69
SD	JDT	.08	.54	20	2.48	13.77	.039	.66	.73
	PDE	.11	.41	20	3.94	1.94	.023	.60	.64
	Equ	.27	.90	20	2.08	4.58	.054	.62	.76
	Luc	.11	.58	20	2.29	4.37	.017	.74	.69
	Myl	.10	.43	20	2.9	12.63	.021	.67	.63
CN2-SD	JDT	.12	.54	5	1.58	18.961	.055	.61	.73
	PDE	.14	.59	3.7	2.89	1.106	.023	.57	.68
	Equ	.17	.78	5	1.020	3.772	.043	.63	.71
	Luc	.07	.41	5	2.2	4.378	.016	.58	.65
	Myl	.08	.38	4.5	2.818	11.06	.018	.55	.63

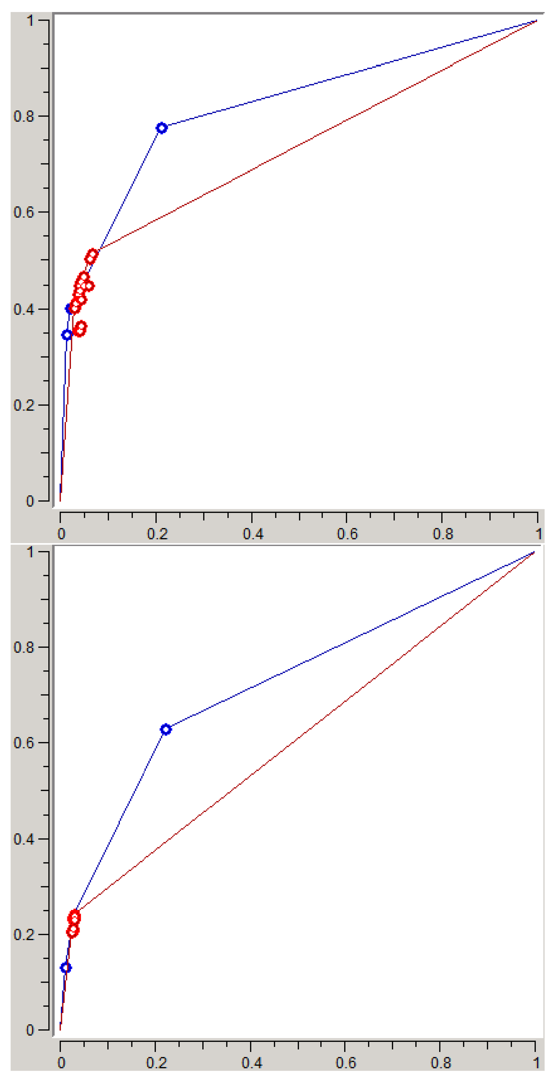

Figure: AUC KC2 & JDT

Conclusions

- SD algorithms focus on finding rules for defective modules ignoring the non-defective ones so that the algorithms are robust to problems faced by classification algorithms such as datasets being unbalanced, noise, inconsistency and redundancy of the data. These problems are present in most defect prediction datasets in the software engineering domain.
  - In unbalanced datasets and considering only the number of TP and FP as evaluation measures, the best classification rules using the CN2 algorithm (classifier) correspond to those rules covering samples of the non-defective modules, failing with defective ones.
  - The metrics used for classifiers cannot be directly applied in SD and need to be adapted.

References

[1] M. D'Ambros, M. Lanza, and R. Robbes.
An extensive comparison of bug prediction approaches.
In *7th IEEE Mining Software Repositories (MSR10)*, pages 31–41, May 2010.

[2] Dragan Gamberger and Nada Lavrac.
Expert-guided subgroup discovery: methodology and application.
Journal of Artificial Intelligence Research, 17(1):501–527, 2002.

[3] F. Herrera, C.J. Carmona del Jesus, P. González, and M.J. del Jesus.
An overview on subgroup discovery: Foundations and applications.
Knowledge and Information Systems, 2010 – In Press.

[4] Nada Lavrač, Branko Kavšek, Peter Flach, and Ljupčo Todorovski.
Subgroup discovery with CN2-SD.
The Journal of Machine Learning Research, 5:153–188, 2004.