Available online at www.sciencedirect.com
ScienceDirect
Expert Systems with Applications 34 (2008) 2091–2101
Expert Systems with Applications
www.elsevier.com/locate/eswa

Regression via Classification applied on software defect estimation

S. Bibi *, G. Tsoumakas, I. Stamelos, I. Vlahavas

Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

Abstract

In this paper we apply Regression via Classification (RvC) to the problem of estimating the number of software defects. This approach apart from a certain number of faults, it also outputs an associated interval of values, within which this estimate lies with a certain confidence. RvC also allows the production of comprehensible models of software defects exploiting symbolic learning algorithms. To evaluate this approach we perform an extensive comparative experimental study of the effectiveness of several machine learning algorithms in two software data sets. RvC manages to get better regression error than the standard regression approaches on both datasets.

Keywords: Software quality; Software metrics; Software fault estimation; Regression via Classification; ISBSG data set; Machine learning

1. Introduction

As the size and complexity of software systems increases, software industry is challenged to deliver high quality, reliable software on time and within budget. Although there is diversity in the definition of software quality, it is widely accepted that a project with many defects lacks quality. Defects are commonly defined as deviations from specifications or expectations that might lead to failures in operation (Fenton & Neil, 1999). Knowing the causes of possible defects as well as identifying general software management decisions that may need attention since the beginning of a project could save money, time and work. The estimation of the potential fault-proness of software is an indicator of quality and can help planning, controlling and executing software development and maintenance activities.

An efficient method for defect analysis is learning from past mistakes to prevent future ones. Today, there exist several data sets that could be mined in order to discover useful knowledge regarding defects (International Software Benchmarking Standards Group; Maxwell, 2002). Using this knowledge one should ideally be able to

(a) Identify potential fault-prone software.
(b) Estimate the specific number of faults.
(c) Discover the possible causes of faults.

Several data mining methods have been proposed for defect analysis in the past (Fenton & Neil, 1999, 2001; Khoshgoftaar & Seliya, 2002; Lanubile & Visaggio, 1997; Neumann & Bibi, 2004; Wooff, Goldstein, & Cohen, 2002) dealing successfully with some of the above issues. Most of them create either regression models that provide equations estimating the number of faults, or classification models that identify fault-proneness. Estimating the exact number of faults is too risky, especially in the beginning of a project when too little information is available. In addition, regression models are not easily understood by domain experts, and thus provide little help for the discovery of fault causes. The mathematical nature of these models tends to obscure rather than clarify the potential causes of faults. Classification models that predict fault-proneness on the other hand can be comprehensible, but not very useful, because they give no clue about the actual number of faults.

* Corresponding author. Tel.: +30 2310991910; fax: +30 2310991911.
E-mail addresses: sbibi@csd.auth.gr (S. Bibi), greg@csd.auth.gr (G. Tsoumakas), stamelos@csd.auth.gr (I. Stamelos), vlahavas@csd.auth.gr (I. Vlahavas).

0957-4174/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2007.02.012

These issues led us to propose a different data mining approach, called Regression via Classification (RvC) (Torgo & Gama, 1997), that benefits from the advantages and caters for the disadvantages of both regression and classification approaches. RvC involves the discretization of the target variable into a finite number of intervals, the induction of a classification model for predicting such intervals and the transformation of the model predictions back into specific numerical estimates.

In Bibi, Tsoumakas, Stamelos, and Vlahavas (2006) the authors have introduced the theoretical framework of RvC and applied and evaluated the method in one data set. The results were encouraging and indicated several advantages of RvC when applied for software fault prediction:

- It can provide a better understanding of software defects by automatically dividing their numerical values into significant intervals.
- Apart from a numerical estimate of faults, it also outputs an associated interval of values, within which this estimate lies with a certain confidence. This way it reduces the level of uncertainty associated with just a point estimate, and provides more knowledge concerning the defects to the end user.
- It allows the production of comprehensible models of software defects that are easily interpretable by project managers and other non-experts in data mining technology. This can be achieved through the use of symbolic classification algorithms (rule learning, decision tree learning).

In order to evaluate RvC in terms of its prediction accuracy, a comprehensive experimental study is performed. We use several classification algorithms for the implementation of the RvC framework and we assess their efficiency for the task of software defect prediction. We also make a comparative evaluation of RvC with classical regression algorithms used in past approaches and other state-of-the-art regression algorithms from the field of machine learning such as support vector machines and model trees. Apart from the prediction accuracy we also present and discuss the knowledge that the RvC framework produces, i.e. the result of the automatic discretization of the defects and the comprehensible models that are produced (rules and trees) by symbolic algorithms.

Two data sets were used for the evaluation of all these approaches. The first one involves maintenance data from bank applications (Maxwell, 2002). It contains data about the size and defects of each application. The second data set is the widely known ISBSG multi-organizational repository release 7 that has 1227 projects (International Software Benchmarking Standards Group). Using these two data sets this study serves another purpose as well: to compare company specific models with models based on multi-company data. The results coming from the application of RvC methods to these two data sets are promising. Both regression and classification accuracy of the models is competitive to those of regression models and in most cases RvC outperforms them.

This paper is organized as follows. The next section presents an overview of the related work. In Section 3, we present the RvC framework along with details concerning the implementation of this method for the problem of software defect prediction. The description of the two datasets and the learning algorithms applied to the data sets are given in Section 4. Section 5 presents the evaluation results along with the extracted software fault prediction models. Finally, in Section 6, we conclude the paper and present ideas for future work.

2. Related work

The earliest studies in software defect prediction focused on establishing relationships between software complexity, usually measured in lines of code, and defects. Well known metrics introduced during 1970s is Halstead's theory (Halstead, 1975) which predicts the number of defects based on the language volume and McCabe's cyclomatic complexity (McCabe, 1976) which measures and controls the number of paths through a program. From this point forward several models were suggested based on the above metrics (Gaffney, 1984; Lipow, 1982) and also other code complexity metrics (Cohen & Devanbu, 1999; Neumann & Bibi, 2004).

The usual drawback of complexity metrics is that they indicate software size as the only predictor of faults. Therefore in 1980s and afterwards research has tried to relate software complexity to sets of different metrics, deriving regression models. A simple regression method is linear regression that when applied in large data sets outperforms expert judgement (Tomaszewski, Hakansson, Lundberg, & Grahn, 2006). Recent studies exploring regression models suggest discriminant analysis (Bellini, Bruno, Nesi, & Rogai, 2005) and neural networks for predicting the number of faults in modules (Lanubile, Lonigro, & Visaggio, 1995; Lanubile & Visaggio, 1997). Neural nets are also used in Quah and Thwin (2004) in order to predict the number of faults in PL/SQL projects. In Khoshgoftaar and Seliya (2002), a case study is presented where various tree based regression models using design metrics are suggested for predicting the number of faults in modules. MARS regression method has been introduced in Briand, Melo, and Wust (2002) for fault prediction in object-oriented code. Also MARS is suggested for the prediction of the number of faults from abstract cognitive complexity metrics are presented and analyzed (Neumann & Bibi, 2004). Ostrand, Weyuker, and Bell (2005) suggests the use of a binomial regression model that exploits the knowledge of past releases of a software project regarding faults to predict the number of faults in the next release of the project. Regression models on the other hand present the disadvantage of providing results in the form of a "black box". The results are difficult to interpret and sometimes the models ignore important causal effects. In 1990s, classification mod-

els were adopted to solve this problem. Clustering in combination with expert opinion is proposed for creating fault prediction models and detecting potential noisy modules in Zhong, Khoshgoftaar, and Seliya (2004b) Zhong, Khoshgoftaar, and Seliya (2004a). Logistic regression is applied in studies (Bellini et al., 2005; Emam, Melo, & Machado, 2001 Kamiya, Kusumoto, & Inoue, 1999) for the estimation of fault-proneness. Bayesian nets in combination with multicriteria decision aid in Fenton and Neil (2001) are suggested for providing a support for decision making based on causality. Also in Wooff et al. (2002) a Bayesian model is suggested for tracking modules for software testing. Most of the above studies estimate potential fault-proneness of software components without providing any fault number.

In the same decade due to the large number of research in this field, several studies compared different methods such as regression techniques and classification techniques. However the most accurate method varied according to the context of the study. Principal component analysis, discriminant analysis, logistic regression, logical classification models, layered neural networks, and holographic networks are applied in Lanubile et al. (1995) and Lanubile and Visaggio (1997) while MARS regression method and classification methods such as rules, CART and Bayesian networks are compared in Neumann and Bibi (2004). Fenton and Neil (1999) provided a critical review of literature and suggested a theoretical framework based on Bayesian networks that could solve the problems identified. They argued that complexity metrics should not be the only predictor of defects, they pointed out that statistical methodologies should pay attention on the data quality and the evaluation method and finally they stressed that it is important to identify the relationship between faults and failures.

As mentioned in Fenton and Neil (1999) clearly all of the problems described cannot be solved easily, however modeling the complexities of software development using new probabilistic techniques presents a positive way forward. In this study, we propose the use of Regression via Classification for modeling uncertainty in software defect prediction. Using this method we will attempt to solve several of the problems mentioned in literature such as, interpretability of the results, use of size as the only predictor, combination of results with expert opinion. We will apply RvC in two data sets containing data relative to the environment variables, the application domain and the computer attributes. The main idea behind the prediction models is to provide a clue of which tools, languages and application domains are connected with the existence of faults and prepare the ground of the appropriate managerial decisions.

3. Regression via Classification

In this section we first present the general framework of the Regression via Classification approach and the important stages in this process. We then provide specific details concerning our implementation of this framework for the problem of software defect prediction.

3.1. The Regression via Classification framework

Supervised machine learning considers the problem of approximating a function that gives the value of a dependent or target variable y, based on the values of a number of independent or input variables $x_1, x_2, \ldots, x_n$. If y takes continuous values, then the learning task is called *regression*, while if y takes discrete values then it is called *classification*.

Traditionally, machine learning research has focused on the classification task. It would therefore be very interesting to be able to solve regression problems taking advantage of the many machine learning algorithms and methodologies that exist for classification. This requires a mapping of regression problems into classification problems and back, which has already been studied by some researchers (Torgo & Gama, 1997; Weiss & Indurkhya, 1995).

The whole process of Regression via Classification (RvC) comprises two important stages: (a) The discretization of the numeric target variable in order to learn a classification model, (b) the reverse process of transforming the class output of the model into a numeric prediction. The first stage is necessary for training the classification model and the second for applying it to new data.

3.1.1. Discretizing the target variable

The task of discretizing an input attribute for classification problems is usually divided into supervised discretization, when knowledge about the class attribute is used for the discretization process and unsupervised discretization, when the class values of the instances are unknown or not used. An interesting study and experimental results of supervised and unsupervised discretization methods can be found in Dougherty, Kohavi, and Sahami (1995). The discretization of a target variable in a regression problem is an unsupervised discretization task.

Two simple methods for unsupervised discretization are equal-interval binning and equal-frequency binning. The former divides the range of values of a numerical attribute into a pre-determined number of equal intervals. The latter divides the range of values into a pre-determined number of intervals that contain equal number of instances.

The k-means clustering algorithm (Witten & Frank, 1999) has also been used for unsupervised discretization. The algorithm starts by randomly selecting k values as centers of the ranges. It then assigns all values to the closest of these centers and calculates the new centers as the mean of the values of these ranges. This process is repeated until the same values are assigned to each of the k ranges in two consecutive iterations.

3.1.2. Transforming classifier outputs to numeric predictions

Once the discretization process has been completed, any supervised classification algorithm can be used for modeling the data. The next step is to make numeric predictions from the classification model that is produced. This model

predicts a number of classes which correspond to numerical intervals of the original target variable. There remains the problem of transforming this class to a specific number, in order to assess the regression error of the RvC framework. A choice for this number should be a statistic of centrality that summarises the values of the training instances within each interval. Such a statistic could be for example the mean value of the target variable for each interval.

3.2. Our RvC implementation for software defect prediction

Given the three different unsupervised discretization methods that were previously described, one problem that we are faced with in our implementation of the RvC framework, is which method to use for discretizing the faults. None of these three methods is better than the rest for all cases.

A second problem is that all discretization methods have a common requirement: a pre-determined number of classes k. So, even if we have selected one of these methods, we would then have to determine the number of classes. Obviously this number will have a direct effect on the subsequent classification learning task.

In order to deal with the above problems of determining the actual parameters of the discretization process of the RvC framework, we decided to use a wrapper approach (Kohavi, 1995b). The wrapper approach is used for determining the best parameters for a machine learning method, and it has been used in the past for classifier parameter selection and feature selection. It evaluates the different configurations of an approach by performing cross-validation and selects the configuration with the best accuracy.

Applying the wrapper approach, we run the discretization process using all three methods and experiment with the number of classes in the range 2 to $1 + 3.3\log(n)$, where n is the number of instances. The upper bound of the number of classes was proposed in Sturge (1926) for selecting the number of classes for discretization purposes. However, this is just a statistical proposal for the number of classes, that does not take into account any knowledge about the domain and tends to propose a rather large number of classes. For this reason we used it as an upper bound in the wrapper approach.

In total, our implementation evaluates $3 * (1 + 3.3\log(n) - 2) = 9.9\log(n) - 3$ different configurations of the discretization process using 10-fold cross-validation (Kohavi, 1995a). The 10-fold cross-validation process splits the data into 10-equal disjoint parts and uses nine of these parts for training the RvC framework and one for testing. This is done 10 times, each time using a different part of data for testing. The training data are used initially to discretize the faults (using one of the configurations) and then to train a classification algorithm. The learned model is then applied to the test data. For the transformation of the output of the classification model back to a numeric estimate we use the median of the values in each interval, as it is usually a more robust centrality measure than the mean. So, for each test instance we calculate the absolute difference of the number of faults in this instance with the median value of the predicted class interval. The average of these differences for all test instances is the Mean Absolute Error performance metric for numeric prediction. The configuration with the lowest average Mean Absolute Error over all the 10-folds of the cross-validation is selected as the configuration to use.

4. Data sets and learning algorithms

We firstly describe here the two data sets that were used in the experiments. We then present the learning algorithms that were used for RvC and ordinary regression on these data sets.

4.1. Data sets

Both data sets include data concerning the methods, tools, requirements and people that are involved in a software project. This information will be used by the algorithms for making an estimation of possible defects. Pekka data set used in the study was selected as a small data set containing projects maintained by a single company. ISBSG data set was selected as a large multi-organizational data set indicative of software development in companies all over world. Descriptive statistics of the two data sets are presented in Table 1.

4.1.1. The Pekka data set

The data in the Pekka data set come from a big commercial bank in Finland, which began to collect development and maintenance data as early as 1985. The data were collected by Pekka Forselious and are presented in Maxwell (2002). Between 1987 and 1995, 250 IBM applications were developed that moved applications from a Bull mainframe environment to a three-tier architecture system constructed of PCs, local servers and IBM mainframes. From the 250 projects of the database, a subset of 67 applications was presented in Maxwell (2002) that had accurate, complete and valid size, defect and effort data as mentioned in the book. The results coming from the statistical analysis when considering the whole data set pointed out the project with ID 55, which presented 163 defects, as an outlier. Almost all classification methods created a fault class with that project as a single member while the rest of the projects were classified into another class. In order to avoid the above situation the models were recreated omitting the

Table 1
Descriptive statistics for the number of faults

Data set	No. of projects	Minimum	Maximum	Mean	Std. Dev.
Pekka	66	0	59	7.106	11.848
ISBSG	91	0	151	9.813	21.31

project with ID = 55. The final data set used in the study included 66 projects. Tables 2 and 3 present the variables of Pekka data set used in our analysis.

This data set was selected because it contained projects with many defect data and risk factors whose values have been carefully assessed. In addition, it came from only one company so the results, would be indicative of the predictive accuracy of the suggested estimation models within a company.

4.1.2. The ISBSG data set

The ISBSG (International Software Benchmarking Standards Group) data set release 7 contains 1238 projects that cover the software development industry from 1989 to 2001 (International Software Benchmarking Standards Group). The group maintains, develops and exploits a repository of international software project metrics to help developers with project estimation and benchmarking. The data set contains over 50 fields involving the nature of the project, the type of the product and the development environment, methods and tools. The projects come from 20 different countries. Over 70 languages are represented in the data set. As a consequence, ISBSG data set covers a wide range of possible project attributes and application environments.

The selection of this data set for creating and comparing fault estimation models was motivated by the following questions:

1. Is it feasible to create accurate fault estimation models from multi-organizational data?
2. Different applications, in different environments have the same causes of faults?
3. What are the differences between estimates derived from multi-company data and estimates derived from company specific data?

Due to the fact that the ISBSG repository is a large heterogeneous data set, a data selection, transformation and preparation process was required before applying any data

Table 2
Variable description for Pekka data set

Name	Variable description	Values
Borg	Business organization type	BigCorp = Big Corporation, Corp = Other corporations, Group = Accounting/management, ITServ = IT services, InHServ = In-House services, Retail = Retail/people
Morg	Internal business unit	Account = Accounting, BUC = Business unit counting, Common = Banking service, CustInt = Customer interconnecting, Decsup = Decision support, Deposit = Deposit ITInfra = IT infrastructure, ITServ = IT services, ITSupp = IT technical support, IntlBank = International banking, LetCred = Letter of credit, Loan = Loan security, Payment = Payment, Person = Personel, Treasury = Treasury, Resto = In-house restaurant, SecTrade = Securities trading system
Apptype	Application type	BackOff = BackOffice database, Connect = Customer interconnection service, Core = Core banking business system, InfServ = Information service/decision support
DBMS	Database management system	DB2, ISDN
Tpms	Transaction processing management system	BATCH, IIMS, IMS, PTCICS, RECICS
r1	Number of users	Values of risk factors range from 1 to 5 1 = Least risky situation 5 = Most risky situation
r2	Configuration	
r3	Change management	
r4	Structural flexibility	
r5	Documentation quality	
r6	People dependence	
r7	Shutdown constraints	
r8	Online transaction processing integration	
r9	Batch processing integration	
r10	Capacity flexibility	

Table 3
Descriptive statistics for the variables with continuous values for Pekka data set

Name	Variable description	Minimum	Maximum	Mean	Std. Dev.
FP	Function points	18	2328	465.985	517.08
Pcobol	Percentage of code written in cobol	0	1	0.377	0.296
Ptelon	Percentage of code written in telon	0	0.87	0.072	0.19
Peasy	Percentage of code written in easy	0	0.52	0.095	0.134
Pjcl	Percentage of code written in jcl	0	1	0.456	0.264
Ageend	Total months maintained	8	85	39.288	20.74
Avetrans	Average transactions per 24 h	0	345	14.227	47.19
Disksp	Disk space used	0	39,012	1838.015	6063.651
Cpu	Cpu usage	0	2197	298.106	526.563

mining methods. Initially only the projects that had values for the minor, major and extreme defects were selected (97 projects). Because several projects had no defects reported for all three defect categories we decided that it would be more appropriate to create one more defect field. This field represents the total number of faults detected in an application and is the sum of the minor, major and extreme defects that are recorded in the data set. In order to avoid any mistakes from data collection or measurement scope, only projects with data quality A and B in the corresponding field were selected. Finally variables containing redundant or aggregated information were excluded (such as the basic elements of function points analysis). Also for attributes like Organization type that had many discrete levels, levels for which less than 4 observations were observed were put under category *Other*. After the pre-processing stage, 91 projects were used for applying the data mining methods. Tables 4 and 5 present the variables of ISBSG data set.

4.2. Learning algorithms

We used the WEKA machine learning library (Witten & Frank, 1999) as the source of algorithms for experimentation. For the RvC framework we used the following classification algorithms as implemented in WEKA with default parameters unless otherwise stated:

- IBk: The k nearest neighbor algorithm (Aha, Kibler, & Albert, 1991), using cross-validation to select the best k value.
- JRip: The RIPPER rule learning algorithm (Cohen, 1995).
- PART: The PART rule learning algorithm (Witten & Frank, 1998).
- J48: The C4.5 decision tree learning algorithm (Quinlan, 1993).
- SMO: The sequential minimal optimization algorithm for training a support vector classifier using RBF kernels (Platt, 1998).

For ordinary regression we used the following algorithms as implemented in WEKA with default parameters unless otherwise stated:

- Linear: A least median squared linear regression algorithm (Rousseeuw & Leroy, 1987).

Table 4
Variable description for ISBSG data set

Name	Variable description	Values
DT	Development type	New development, re-development, enhancement
LT	Language type	3GL, 4GL, ApG
PPL	Primary programming language	C, COBOL, NATURAL, ORACLE, PL/I, SQL, TELON other4GL, other
OT	Organisation type	Communication, electricity/gas/water, financial, business, manufacturing, other, public administration, transport and storage, energy
DBMS	Database management system	ADABAS, DB2, IDMSX, ORACLE, other
DP	Development platform	MF, MR, PC
HMA	How methodolgy acquired	Developed/purchased, developed inhouse, purchased
AT	Application type	DSS, MIS, Office.I.S, other, process control, transaction/production
BAT	Business area type	Accounting, banking, engineering, financial, fineenforcement, manufacturing, other
UCU	Upper case tool used	Yes, no
LCU	Lower case tool used	Yes, no
ICU	Integrated case tool used	Yes, no
MA	Methodology acquisition	Inhouse, purchased, combined
PC	Package customization	Yes, no

Table 5
Descriptive statistics for the variables with continuous values for ISBSG data set

Name	Variable description	Minimum	Maximum	Mean	Std. Dev.
FP	Function points	11	5684	763.385	957.72
mts	Max team size	1	65	7.868	11.936
Date	Implementation date	1992	1995	1993	0.716
EI	External inquiry	4	2221	345.5	405.493
EO	External output	0	1216	207.6	238.541
EE	External inquiry	0	820	135.775	152.659
ILF	Internal logical files	0	1137	202.75	213.737
EIF	External interface files	0	317	32.325	68.545
Added	Added lines of code	0	1481	171	298.932
Changed	Changed lines of code	0	556	113.429	133.596
Deleted	Deleted lines of code	0	270	23.829	61.5

- MLP: An algorithm for training a multi-layer perceptron (Bishop, 1995).
- Reg-IBk: The k nearest neighbor algorithm (Aha et al., 1991), using cross-validation to select the best k value.
- SMOreg: The sequential minimal optimization algorithm of Smola and Scholkopf (1998) for support vector regression using RBF kernels.
- M5P: An algorithm for generating M5 model trees (Quinlan, 1992; Witten & Frank, 1999). This algorithm is used twice, one for the production of a model tree and one for the production of a regression tree.
- REPTree: A fast regression tree learner that uses information variance reduction and reduced-error pruning (Witten & Frank, 1999).

5. Results and discussion

In this section we first present the evaluation results and then the classification models that were extracted from the two data sets will be presented and discussed.

5.1. Comparative evaluation of RvC

In order to evaluate RvC and the regression algorithms, 10-fold cross-validation was used, which is one of the most widely used and acceptable method for evaluating machine learning approaches (Kohavi, 1995a). The performance of the approaches was measured by their average Mean Absolute Error for the 10-folds of the cross-validation process. In addition, for RvC we calculated the average classification accuracy of the algorithms, the average number of fault classes and the percentage that each of the three discretization methods was used.

Table 6 shows the average Mean Absolute Error of all the approaches on the two datasets. We firstly notice that RvC actually manages to get better regression error than the standard regression approaches on both datasets. The best performance in both data sets is obtained with RvC and the SMO algorithm, while the SMOreg algorithm for regression is the second best. This result agrees with the idea of support vector machines as very effective machine learning approaches. However, support vector machines do not produce comprehensible models. Fortunately, relatively good performance is also obtained by the symbolic algorithms (RIPPER, C4.5 and PART). We will present the comprehensible models produced by these algorithms in the following section.

Another thing that must be noted is the fact that RvC achieves better performance overall than regression approaches, even though it uses a rough estimation of the actual numbers. One reason for this could be the difficulty of the datasets (few instances, high dimensionality) as a regression learning problem which leads to poor numeric estimates.

Table 7 shows the accuracy of the RvC classification algorithms, the mean number of classes in the 10-folds of the cross-validation and the percentage of times that each of the three methods (M1: equal-width, M2: equal-frequency, M3: k-means) was used for discretizing the number of defects. We first notice that the most accurate algorithms are SMO and PART and this has certainly contributed to the corresponding low regression error of RvC. However, RvC with RIPPER managed to achieve low regression error even though the classification accuracy of RIPPER was not very high. This shows that apart from the classification accuracy, the actual discretization of defects into intervals is also important for the regression error. It also shows that the end user should trust more the comprehensible models produced by PART and C4.5 rather than that of RIPPER.

Looking at the average number of classes we see that for the Pekka data set, it varies a lot from one algorithm to the other and ranges from 2 to 5.4. In the ISBSG data set it varies less and ranges from 2.5 to 3.7. The most successful discretization method seems to be the simplest one, i.e. equal-width binning.

Table 6
Mean Absolute Error of RvC and regression approaches

	RvC					Regression						
	SMO	RIPPER	PART	C4.5	IBk	SMOreg	Linear	REPTree	M5P regression tree	M5P model tree	IBk	MLP
PEKKA	6.69	7.15	7.70	8.53	7.88	7.07	7.96	7.72	7.71	7.28	8.27	7.22
ISBSG	9.08	9.77	9.56	9.36	10.17	9.81	10.17	10.80	10.87	12.20	12.99	15.35

Table 7
Accuracy, mean number of classes and percentage of each discretization method

	PEKKA					ISBSG				
	Acc	Av. C	M1	M2	M3	Acc	Av. C	M1	M2	M3
SMO	0.94	2.00	1.00	0.00	0.00	0.94	3.10	0.80	0.00	0.20
PART	0.72	4.40	0.60	0.10	0.30	0.86	3.70	0.70	0.10	0.20
IBk	0.69	2.40	0.40	0.50	0.10	0.83	2.50	0.70	0.20	0.10
C4.5	0.67	3.90	0.60	0.10	0.30	0.62	3.30	0.20	0.40	0.40
RIPPER	0.46	5.40	0.40	0.60	0.00	0.64	3.70	0.30	0.40	0.30

Observing the results in Tables 6 and 7 some conclusions can be drawn from the comparison of models created by multi-company data set (ISBSG data set) with the ones created by company specific data set (Pekka data set). There is a diversity between the results presented in Tables 6 and 7. When the two models are compared in terms of Mean Absolute Error it seems that company specific model outperforms the muti company model. When the two models are compared in terms of classification accuracy, the multi-company model perform slightly better than the company specific model. The difference in the performance of the two models between classification and regression results can be explained by the number of classes considered by the methods. The algorithms indicated a relatively small number of classes probably due to the fact that in the data set not sufficient evidence appeared that would allow the creation of more classes and the successful estimation of them. This had as a result a better classification accuracy of multi-company model due to the few number of classes but a slightly worse regression accuracy due to the amplitude of the classes. A tentative conclusion that can be drawn from this fact is that in our case a model from multi-company data can only provide a rough fault class estimation. Note that the generality of the conclusions is not guaranteed, as they are based on the two specific data sets that we have used in this study.

5.2. Model analysis

Here we will present the comprehensible models produced by the three symbolic algorithms C4.5, PART and RIPPER. The first algorithm outputs a decision tree, while the other two output a set of classification rules. Each rule has a *body*, which consists of one or more conditions under which the rule will fire, and a *head* which consists of the predicted class of faults. We also present two quantitative measures of the rule's quality: (a) *support*, which is the ratio between the number of records that satisfy the rule body and the total number of records in the database, and (b) *confidence*, which is the ratio between the number of database records that satisfy both the rule body and head and the number of records that satisfy just the rule body.

We must note here that RIPPER and PART belong to the separate-and-conquer family of rule learning algorithms. These algorithms learn one rule, remove the examples that this rule covers and proceed with the next rule. Any remaining uncovered examples, are handled by a *default* rule that fires without any conditions and predicts the most frequent class among the remaining examples. Therefore the support and confidence of each rule is reported based on the subset of the examples that remained for that rule. This also implies that the rules are presented in the order that they are discovered, and during execution they are considered in this order.

5.2.1. Pekka data set

Initially, the RvC and regression algorithms have been applied to the 66 projects included in our analysis. Table 8 presents the rule sets that was produced by RvC with the RIPPER and PART algorithms, while Fig. 1 presents the decision tree of C4.5. The first rule coming from the PART decision list can be explained as following: In the training data set there are 13 projects with fp $\leqslant$ 939 and internal business sector ACCOUNT from which 12 present equal or less to 8.43 faults. Applying this rule only one project is misclassified to another fault class. Support value for this rule is (13/66) 19.7% and confidence value is (12/13) 92.3%.

The decision tree of Fig. 1 has two splitting nodes. The number of function points is the splitting criterion for both nodes. The decision tree is interpreted as follows: If the number of function points of an unknown project is equal or less than 1361 then the number of defects will be less than 29. Else if the number of function points is equal or less than 1979 then the number of faults will be between 29 and 59, otherwise the number of faults will be less than 29. For each suggested class in the leaf the average number

Table 8
Rule list produced by CvR with PART and JRIP on the Pekka data set

Body	Head	Point estimate	Confidence	Support
JRIP list of rules				
FP $\leqslant$ 986	$9.5 < D \leqslant 59$	20	87.5	12.12
cpu $\geqslant$ 292	$2.5 < D \leqslant 9.5$	5	60	15.15
FP $\geqslant$ 671	$2.5 < D \leqslant 9.5$	5	80	7.58
	$0 \leqslant D \leqslant 0.5$	0	39.53	65.15
PART list of rules				
FP $\leqslant$ 939 and morg = ACCOUNT	$0 \leqslant D \leqslant 8.43$	1	92.31	19.70
FP $\leqslant$ 939 and borg = RETAIL	$0 \leqslant D \leqslant 8.43$	1	85.71	21.21
morg = PAYMENT and r1 = 5	$0 \leqslant D \leqslant 8.43$	1	85.71	10.61
cpu $\leqslant$ 506 and r6 = 2	$0 \leqslant D \leqslant 8.43$	1	85.86	21.42
r5 = 1	$0 \leqslant D \leqslant 8.43$	1	63.26	5.48
morg = deposit	$50.57 < D \leqslant 59$	53	100.00	3.03
Ageend $\leqslant$ 40	$8.43 < D \leqslant 16.86$	11	66.67	4.55
	$16.86 < D \leqslant 25.29$	20	74.55	3.39

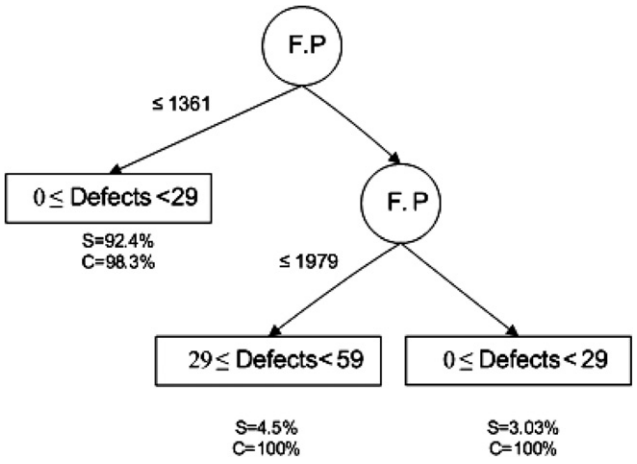

Fig. 1. Decision tree C4.5 for Pekka data set.

of the class is indicated as the most probable fault number of the class along with support and confidence values.

A project variable that appears often in the results of the three classification methods is Function Points. This is reasonable, as function points is a metric indicative of the size of a software application, and as the size of a software project grows so does its complexity. Software complexity is widely accepted as a major cause of defects. An interesting rule is the one indicated by PART decision list that application that are destined for deposit units tend to present a large number of faults. Also the applications that have low CPU usage seem to be less fault-prone. A surprise to us was that only risk factors r1, r5 and r6 (number of users, documentation quality and people dependence) appeared from the risk factors.

In this data set the classes of defects suggested by JRip and PART algorithms are relatively tight. JRip predicts three classes of faults with most probable values, 0, 5 and 20 faults. The model estimates whether a software project will present few number of faults (0 or 1), nominal number of faults (3–9) or relatively large number of faults (10–59). PART decision list estimates four probable number of faults, one fault, 11 faults, 20 faults and 53 faults.

The decision tree suggests a rough estimate of fault classes, predicting only 2 fault classes. The first fault class predicts from 0 to 28 faults with most probable value 2 faults and the second class is between 29 and 59 faults with most probable number 39 faults. Even though the suggested classes are relatively large the regression performance of C4.5 considering the median point of each class is competitive and even better than traditional regression models.

5.2.2. Isbsg data set

For ISBSG data set the models that are suggested by the framework of Regression via Classification are presented in Table 9, and in Fig. 2. For example the first rule of Table 9 can be interpreted as follows: If the function points of a project are equal or less than 303 and the added lines of code are equal or less than 153 then the total number of defects found will be more than 0.5 and less or equal to 3.5 with a most probable fault value of 1.8 faults. Though the number of the added lines of code cannot be estimated easily, it is a variable often used in models. This rule in the training set classifies correctly four of the projects that present the above values of function points and added lines of code into the correct class of faults. None of the projects of the training data set that comply with this rule is misclassified to a wrong fault class. The rules coming from PART decision list are interpreted in the same way.

The decision tree presented in Fig. 2 has the number of function points as a splitting node. If the number of function points is less than 722 the faults will be equal or more than 10 and less than 61. This rule coming from the decision tree classifies correctly 55 of the 60 projects whose function points are equal or less than 722. If the number of function points of a software application is more than 722 then the number of faults will be more than 61 and equal or less than 151, this rule classifies correctly 16 out of 31 projects.

As expected, function points appeared often in the results. Also Office and Management Information Systems are pointed out as fault-prone applications probably due to their increased functionality. An interesting rule is the one

Table 9
JRIP and PART list of rules for ISBSG dataset

Body	Head	Point estimate	Confidence	Support
JRIP list of rules				
FP <= 303 and ALC $\geqslant$ 153	$0.5 < D \leqslant 3.5$	2	100.00	4.39
MTS $\geqslant$ 4 and LT = 4GL	$9.5 < D \leqslant 151$	14	75	17.58
FP $\geqslant$ 499 and PL = COBOL	$9.5 < D \leqslant 151$	14	85.71	7.69
N_EO $\geqslant$ 71 and OT = other	$9.5 < D \leqslant 151$	14	100	4.4
	$0 < D \leqslant 0.5$	0	36.67	65.93
PART list of rules				
FP > 722 and ID <= 94	$3.5 < D \leqslant 151$	10	89.63	31.79
DT = Enhancement and Up.CASE = no	$0 < D \leqslant 3.5$	0	83.36	32.56
AT = Office information system	$3.5 < D \leqslant 151$	10	83.33	6.59
OT = Electricity and gas and water	$0 < D \leqslant 3.5$	0	100.00	4.39
AT = Management information system	$3.5 < D \leqslant 151$	10	73.84	9.03
AT = Transaction/production and N_EO > 57	$3.5 < D \leqslant 151$	10	69.44	6.33
	$0 < D \leqslant 3.5$	0	87.35	9.30

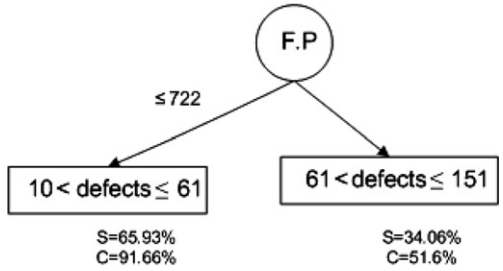

Fig. 2. Decision tree for ISBSG data set.

included in the PART decision list and involves implementation date. It seems that applications developed 1994 and earlier with a high number of function points are fault-prone. In general as indicated by the decision tree a high number of function points is itself a reason for fault existence. But the rule that involves implementation date as well provides an additional information. Software size and complexity seemed more troublesome and difficult to handle before 1995. Probably sufficient experience is gained in the last years that gradually minimizes the appearance of faults.

In general, in both data sets in many cases large defect intervals are estimated. Some could say that the estimate is too general and fuzzy and therefore has reduced practical use. But we argue that it is preferable to have a large interval with a high confidence value within which the actual fault number will fall, than a small interval with a low degree of confidence, or no estimate at all. Probably in larger data sets the appearance of few and large fault classes would be minimized as more information would be included in the models.

Regarding the causes of defects, they remain the same both when the data come from the multi-company data set and the company-specific data set. In both cases in our study the size of a software application is the main reason of faults. Also the type of the application seems to have an influence on software fault-proness.

6. Conclusions and future work

In this paper the framework of Regression via Classification (RvC) was applied to the problem of early fault prediction. Our motivation was to exploit the advantages of classification algorithms in order to solve the main drawbacks of regression algorithms, such as the incomprehensibility of the produced models and their inability to provide a good point estimate of faults. RvC provides a complete framework for defect prediction producing as an output a fault class into which the actual fault number may fall in, along with a specific most probable fault number within this class. The representation of the fault knowledge can be in the form of rules and decision trees which are among the most expressive and human readable representations for learned hypotheses.

In general RvC as a data mining method offers a convenient way to solve problems that are not explained purely logically but rather probabilistically. Software fault estimation is one of these problems: we are not sure of the factors that affect directly the existence of faults and we expect a support from statistical methods to point out the underlying relationships that appear in fault data. Some of the results of the application of RvC technique were expected and confirmed by intuition like the influence of a software application size on the existence of faults. The success of the method is that it provides a framework for discovering potential cause–effect relationships that can be surprising like the one in ISBSG data set that implies that when a CASE tool is not used the number of faults is reduced. This is controversial to the fact that the use of a CASE tool is considered to prevent faults and aid software development. However the same rule may also imply that with the use of CASE tools more faults are triggered or detected. Further analysis is needed to investigate this phenomenon.

In addition, we must stress the very good results of RvC in terms of regression error. Despite the fact that RvC outputs the median of an entire interval as its point estimate of faults, it manages to outperform most of the regression approaches in predictive accuracy.

Future work may involve the application of RvC in predefined fault intervals in order to avoid the extraction of models estimating large fault classes. Also we intend to apply the proposed methodology to other software data sets involving other software quality attributes in addition to defects so as the results of the RvC analysis will guide the testing process. We will also experiment with methods that combine different classification algorithms such as Stacking (Wolpert, 1992) and Effective Voting (Tsoumakas, Katakis, & Vlahavas, 2004) for the purpose of increasing the predictive performance of RvC.

References

Aha, D., Kibler, D. W., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*(1), 37–66.

Bellini, P., Bruno, I., Nesi, P., & Rogai, D. (2005). Comparing fault-proneness estimation models. ICECCS 2005, pp. 205–214.

Bibi, S., Tsoumakas, G., Stamelos, I., & Vlahavas, I. (2006). Software defect prediction using Regression via Classification. In *The proceedings of 4th ACS/IEEE international conference on computer systems and applications (AICCSA 2006)* (pp. 330–337), Dubai/Sharjah, 8–11 March.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Briand, L., Melo, W., & Wust, J. (2002). Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Transactions on Software Engineering, 28*(7), 706–720.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the 12th international conference on machine learning* (pp. 115–123). Morgan Kaufman.

Cohen, W. W., & Devanbu, P. (1999). Automatically exploring hypotheses about fault prediction: A comparative study of inductive logic programming methods. *International Journal of Software Engineering and Knowledge Engineering, 9*(5), 519–546.

Dougherty, J., Kohavi, R. & Sahami, N. (1995). Supervised and unsupervised discretization of continuous features. In: *Proceedings of the 12th international conference on machine learning* (pp. 194–202), Tahoe City, US.

Emam, K. E., Melo, W., & Machado, J. C. (2001). The prediction of faulty classes using object-oriented design metrics. *The Journal of Systems and Software, 56*, 63–75.

Fenton, N., & Neil, M. (1999). A critique of software defect prediction models. *IEEE Transactions on Software Engineering, 25*(5), 675–689.

Fenton, N., & Neil, M. (2001). Making decisions: Using Bayesian nets and MCDA. *Knowledge-Based Systems, 14*, 307–325.

Gaffney, J. R. (1984). Estimating the number of faults in code. *IEEE Transactions on Software Engineering, 10*(4).

Halstead, M. H. (1975). *Elements of software science*. Elsevier, North-Holland.

International Software Benchmarking Standards Group, http://www.isbsg.org.

Kamiya, T., Kusumoto, S., & Inoue, K. (1999). Prediction of fault-proneness at early phase in object-oriented development. In *2nd International symposium on object-oriented real-time distributed computing (ISORC '99)* (pp. 53–258), Saint Malo, France, 2–5 May 1999. IEEE Computer Society.

Khoshgoftaar, T. M., & Seliya, N. (2002). Tree-based software quality estimation models for fault prediction. *IEEE Metrics*, 203.

Kohavi, R. (1995a). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th international joint conference on artificial intelligence*, IJCAI 95 (pp. 1137–1145), Montréal, Canada.

Kohavi, R. (1995b). Wrappers for performance enhancement and oblivious decision graphs. PhD thesis. Department of Computer Science, Stanford University.

Lanubile, F., Lonigro, A., & Visaggio, G. (1995). Comparing models for identifying fault-prone software components. In: *Proceedings of the 7th international conference on software engineering and knowledge engineering* (pp. 312–319), Washington, DC, June.

Lanubile, F., & Visaggio, G. (1997). Evaluating predictive quality models derived from software measures: Lessons learned. *The Journal of Systems and Software, 38*, 225–234.

Lipow, M. (1982). Number of faults per line of code. *IEEE Transactions on Software Engineering, 8*(4), 437–439.

Maxwell, K. (2002). *Applied statistics for software managers*. NJ: Prentice-Hall.

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

Neumann, R., & Bibi, S. (2004). Building fault prediction models from abstract cognitive complexity metrics-analyzing and interpreting fault related influences. In *International workshop on software measurement/Metrikon 2004*, Berlin, Germany, 2–5 November.

Ostrand, T., Weyuker, E., & Bell, R. (2005). Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering, 31*(4), 340–355.

Platt, J. (1998). Fast training of support vector machines using sequential minimal optimization. In B. Scholkopf, C. Burges, & A. Smola (Eds.), *Advances in kernel methods – Support vector learning*. MIT Press.

Quah, T. S., & Thwin, M. (2004). Prediction of software development faults in PL/SQL files using neural network models. *Information and Software Technology, 46*(8), 519–523.

Quinlan, J. R. (1992). Learning with continuous classes. In *Proceedings of Australian joint conference on AI* (pp. 343–348). Singapore: World Scientific.

Quinlan, R. J. (1993). *C4.5: Programs for machine learning*. Morgan Kaufman.

Rousseeuw, P. J., & Leroy, A. M. (1987). *Robust regression and outlier detection*. New York: Wiley.

Smola, A.J. & Scholkopf, B. (1998). A tutorial on support vector regression. NeuroCOLT2 Technical Report Series – NC2-TR-1998-030.

Sturge, H. (1926). The choice of class interval. *Journal of American Statistical Association*, 65–66.

Tomaszewski, P., Hakansson, J., Lundberg, L., & Grahn, H. (2006). The accuracy of fault prediction in modified code, statistical model vs. expert estimation. *ECBS*, 334–343.

Torgo, L., & Gama, J. (1997). Regression using classification algorithms. *Intelligent Data Analysis, 1*(4), 275–292.

Tsoumakas, G., Katakis, I., & Vlahavas, I. (2004). Effective voting of heterogeneous classifiers. In *Proceedings of 15th European conference on machine learning, ECML 04* (pp. 465–476), Pisa, Italy, September.

Weiss, S., & Indurkhya, N. (1995). Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research (JAIR), 3*, 383–403.

Witten, I. H., & Frank, E. (1998). Generating accurate rule sets without global optimization. In *Proceedings of the 15th international conference on machine learning*, ICML98 (pp. 144–151).

Witten, I. H., & Frank, E. (1999). *Data mining: Practical machine learning tools with Java implementations*. San Francisco: Morgan Kaufman.

Wolpert, D. (1992). Stacked generalization. *Neural Networks* (5), 241–259.

Wooff, D. A., Goldstein, M., & Cohen, F. P. A. (2002). Bayesian graphical models for software testing. *IEEE Transactions on Software Engineering, 28*(5), 510–524.

Zhong, S., Khoshgoftaar, T. M., & Seliya, N. (2004). Unsupervised learning for expert-based software quality estimation. In *8th IEEE international symposium on high-assurance systems engineering* (HASE 2004) (pp. 149–155), Tampa, FL, USA, 25–26 March.

Zhong, S., Khoshgoftaar, T. M., & Seliya, N. (2004b). Analyzing software measurement data with clustering techniques. *IEEE Intelligent Systems, 19*(2), 20–27.