Software Defect Prediction Using Regression via Classification

Bibi S., Tsoumakas G., Stamelos I., Vlahavas I
Department of Informatics, Aristotle University of Thessaloniki,54124 Thessaloniki, Greece
{sbibi,greg,stamelos,vlahavas}@csd.auth.gr

Abstract

In this paper we apply a machine learning approach to the problem of estimating the number of defects called Regression via Classification (RvC). RvC initially automatically discretizes the number of defects into a number of fault classes, then learns a model that predicts the fault class of a software system. Finally, RvC transforms the class output of the model back into a numeric prediction. This approach includes uncertainty in the models because apart from a certain number of faults, it also outputs an associated interval of values, within which this estimate lies, with a certain confidence. To evaluate this approach we perform a comparative experimental study of the effectiveness of several machine learning algorithms in a software dataset. The data was collected by Pekka Forselious and involves applications maintained by a bank of Finland.

1. Introduction

Although there is diversity in the definition of software quality, it is widely accepted that a project with many defects lacks quality. Knowing the causes of possible defects as well as identifying general software process areas that may need attention from the initialization of a project could save money, time and work. The possibility of early estimating the potential faultiness of software could help on planning, controlling and executing software development activities.

A low cost 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 [7], [14]. Using this knowledge one should ideally be able to: a) Identify potential fault-prone software, b) Estimate the specific number of faults, and c) Discover the possible causes of faults.

Several data mining methods have been proposed for defect analysis in the past [5], [9], [15], [28] but few of them manage to deal successfully with all of the above issues. Regression models estimates are difficult to interpret and also provide the exact number of faults which is too risky, especially in the beginning of a project when too little information is available. On the other hand classification models that predict possible faultiness can be comprehensible, but not very useful, because they give no clue about the actual number of faults.

These issues led us to the proposal of a different data mining approach, called Regression via Classification (RvC) [22], 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's predictions back into specific numerical estimates.

To our knowledge, RvC has not been applied for software fault prediction in the past, despite the many benefits that it offers. It is a method that considers uncertainty, produces comprehensible results and is a reasonable alternative to regression problems that need a logical explanation. Additionally the method performs all of the three tasks of defect prediction, estimates a particular number, estimates a fault class and suggests potential causes of faults.

In order to evaluate RvC in terms of its prediction accuracy, we make a comparative evaluation of several classification algorithms for the implementation of the RvC framework 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. For the evaluation of all these approaches a data set was used that involves maintenance data from bank applications [14]. It contains data about the size and defects of each application. The results coming from the application of RvC methods show that both regression and classification accuracy of the models is competitive to those of regression models and in most cases RvC outperforms them.

The rest of 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 dataset and the learning algorithms applied to the data sets are found 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 fault prediction focused on establishing relationships between software complexity, usually measured in lines of code, and defects. Widely known metrics introduced during 70s is Halstead's theory [6] and McCabe's cyclomatic complexity [13]. The usual drawback of complexity metrics is that they indicate software size as the only predictor of faults. Therefore in 80s and afterwards research has tried to relate software complexity to sets of different metrics, deriving multivariate regression models [12], [9], [15]. Regression models on the other hand presented the disadvantage of providing results difficult to interpret that ignored causal effects. In the 90s classification models were adopted to solve this problem. Clustering [28], logistic regression [4], [8] and Bayesian nets [5] are applied for the estimation of fault-proneness. Most of the above studies estimate potential fault proneness of software components without providing particular fault numbers.

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 but each time 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 [12], while MARS regression method and classification methods such as rules, CART and Bayesian networks are compared in [15]. Fenton and O'Neil [5] 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 [5] 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.

3. Regression via Classification

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 x1, x2, …, xn. If y takes real 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 been recently studied by some researchers [22], [24].

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.

Three methods for discretization are equal-interval binning, equal-frequency binning and K-means clustering [23]. The first one divides the range of values of a numerical attribute into a predetermined number of equal intervals. The second one divides the range of values into a predetermined number of intervals that contain equal number of instances. The k-means clustering 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.

Once the discretization process has been completed, any 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 summarizes the values of the training instances within each interval.

3.1 Our RvC implementation on the problem of software defect prediction.

In this study in order to determine the actual parameters of the discretization process of the RvC framework, we decided to use a wrapper approach [11].

The wrapper approach evaluates the different configurations of an approach by performing cross-validation and selects the configuration with the best accuracy. Similar to that approach, we run the discretization process using all three methods and experiment with the number of classes in the range 2 to

1+3.3log(n). The upper bound of the number of classes was proposed in [21] 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+3.3log(n)-2)=9.9log(n) different configurations of the discretization process using 10-fold cross-validation [10]. The 10-fold cross-validation process splits the data into 10-equal disjoint parts and uses 9 of these parts for training the RvC framework and 1 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 data set that was used in the experiments. We then present the learning algorithms that were used for RvC and ordinary regression on this data set.

4.1 Pekka data set

The data set used in this study is the Pekka data set that comes from a big commercial bank in Finland, which began to collect development and maintenance data as early as 1985 until 1995. The data were collected by Pekka Forselious and are presented in [14].

From the 250 projects of the database, a subset of 67 applications was presented in [14] and used in the evaluation. The variables of the data set used in our analysis are presented in table 1. The data set involves classification and quantitative variables along with 10 variables called risk factors. Target of the study is, based on existing knowledge of historical data, to provide a prediction model for the number of faults that will appear during the maintenance of software applications.

Table 1: Variables of Pekka data set.

Classification variables	Borg:Business organization type
	Morg: Internal business unit
	Apptype: Application Type
	Dbms: Database system
	Tpms: Transaction Processing management system
Risk Factors Values of risk factors range from 1 to 5. 1= least risky situation 5= most risky situation	r1: Number of users
	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
Quantitative variables	F.P (function points)
	Pcobol (% of code in cobol)
	Ptelon (% of code in telon)
	Peasy ((% of code in easy)
	T (recovery capability)
	Ageend (total months maintained)
	Disksp (disk space used)
	Avetrans (average transactions/ 24 h)
	Cpu (cpu usage)
	Pjcl (% of code in jcl)
	Appdef (number of defects) target variable

4.2 Learning algorithms

We used the WEKA machine learning library [26] 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 [1].
- JRip: the RIPPER rule learning algorithm [3].
- PART: the PART rule learning algorithm [25].
- J48: the C4.5 decision tree learning algorithm [18].
- SMO: the sequential minimal optimization algorithm for training a support vector classifier using RBF kernels [16].

We will further analyze PART, RIPPER and C4.5 algorithms as the results of these algorithms are presented in section 4.

C4.5 outputs a decision tree, while the other two (PART and RIPPER) 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 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.

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 [19].

- MLP: an algorithm for training a multi-layer perceptron [2].
- Reg-IBk: the k nearest neighbor algorithm [1], using cross-validation to select the best k value.
- SMOreg: the sequential minimal optimization algorithm of [20] for support vector regression using RBF kernels.
- M5P: an algorithm for generating M5 model trees [17], [26]. 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 [26].

5. Results and discussion

In this section we first present the evaluation results and then the classification models that were extracted from the data set will be presented and discussed. The performance of the approaches was measured by their average Mean Absolute Error (MAE) for the 10 folds of the cross-validation process. The MAE function is given by:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | Pi - Ei |$$

Where n is the number of instances in the test set, P is the actual fault number and E is the predicted one.

In addition, for RvC we calculated the average classification accuracy of the algorithms which provides the percentage of projects for which the correct fault class has been successfully estimated, the average number of fault classes and the percentage of times that each of the 3 discretization methods was used.

Table 2: Mean absolute error of RvC and regression approaches

	Pekka data set	
RvC	SMO	6,69
	RIPPER	7,15
	PART	7,70
	C4.5	8,53
	IBk	7,88
Regression	SMOreg	7,07
	Linear	7,96
	REPTree	7,72
	M5P regression tree	7,71
	M5P model tree	7,28
	IBk	8,27
	MLP	7,22

Table 2 shows the average Mean Absolute Error of all the approaches on the Pekka dataset. We firstly notice that RvC actually manages to get better regression error than the standard regression approaches. Indeed within the top three performers we find two RvC approaches (SMO, RIPPER) and only one regression approach (SMOreg). The best average performance is obtained with RvC and the SMO algorithm, while the SMOreg algorithm for regression is the second best. Relatively good performance is also obtained by the symbolic algorithms (RIPPER, C4.5 and PART) that produce comprehensible models. Another thing that must be noted is the fact that RvC achieves improved performance overall than regression approaches, even though it uses a rough estimation of the actual numbers.

Table 3 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 relatively low. This shows that apart from the classification accuracy, the actual discretization of defects into intervals is also important for the regression error.

Initially, the RvC and regression algorithms have been applied to the whole data set (67 projects). The results 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 create meaningful models whose results could be exploited the models were recreated omitting the project with ID =55.

Table 3: Accuracy, mean number of classes and percentage of each discretization method

	PEKKA				
	Acc	Av. C	M1	M2	M3
SMO	0,94	2,00	1,00	0,00	0,00
PART	0,72	4,40	0,60	0,10	0,30
IBk	0,69	2,40	0,40	0,50	0,10
C4.5	0,67	3,90	0,60	0,10	0,30
RIPPER	0,46	5,40	0,40	0,60	0,00

Table 4 presents the rule sets that were produced by RvC with the RIPPER and PART algorithms accompanied with a point estimate in brackets and confidence and support values. In the results of the RIPPER approach function points and CPU usage are the sole predictors of defects. In the results of PART list of rules the type of the organization and the unit in which the application is destined play important role in the estimation. The decision tree of Figure 1 has two splitting nodes. The number of function points is the splitting criterion for both nodes. For each suggested class the median number of the class is indicated as the most probable fault number of the class as a point estimate (PE) along with support and confidence values.

Table 4: Rule list produced by RvC with PART and JRIP on the Pekka data set.

JRIP list of rules		
Body	Head	(C) (S)
F.P ≤ 986	9.5< D ≤59 (20)	87.5 12.12
cpu ≥ 292	2.5< D ≤9.5 (5)	60.0 15.15
F.P ≥ 671	2.5< D ≤9.5 (5)	80.0 7.58
	0≤ D ≤0.5 (0)	39.5 65.15
PART list of rules		
Body	Head	(C) (S)
F.P ≤ 939 and morg =ACCOUNT	0≤ D ≤8.43 (1)	92.31 19.7
F.P ≤ 939 and borg = RETAIL	0≤ D ≤8.43 (1)	85.71 21.2
morg= PAYMENT and r1=5	0≤ D ≤8.43 (1)	85.71 10.6
cpu ≤ 506 and r6=2	0≤ D ≤8.43 (1)	85.86 21.4
r5=1	0≤ D ≤8.43 (1)	63.26 5.48
morg= deposit	50.57< D ≤59 (53)	100.0 3.03
Ageend ≤ 40	8.4<D ≤16.86 (11)	66.67 4.55
	16.7< D ≤25.3 (20)	74.55 3.39

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 the primary- cause of defects. An interesting rule is the one indicated by PART decision list that application that are destined for deposit units tend to appear a large number of faults. Probably this can be explained by the fact that the requirements for applications for these units are relatively demanding and strict as a single fault could cause loss of money. Even small defects that otherwise would be ignored in such applications are recorded and fixed. Also the applications that have low CPU usage seem to be less fault prone. Another rule that can be confirmed intuitively is the one that supports that application with equal or less than 40 months of maintenance tend to present many faults. A surprise to us was that only r1, r5 and r6 (number of users, documentation quality and people dependence) appeared from the risk factors.

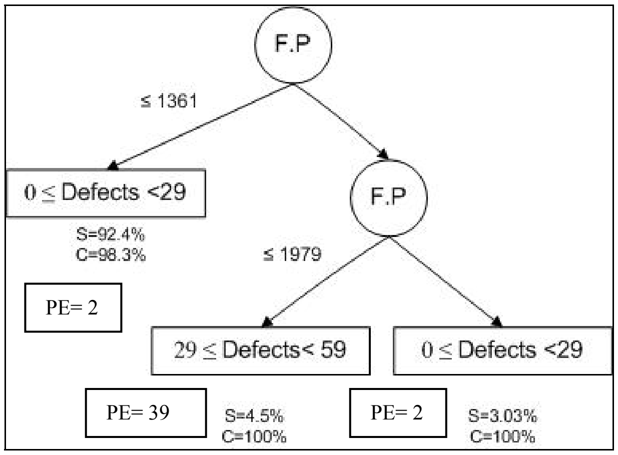

Figure 1: Decision tree C4.5 for Pekka data set.

One could argue that the predicted fault classes of RvC are large and therefore contain fuzzy information. This argument can be confronted with the fact that RvC even when two few fault classes are considered succeeds comparable and even lower regression error from traditional regression models when considering the median value of a class as a point estimate. There are though several advantages by that type of 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.

6. Conclusions and Future work

In this paper the framework of Regression via Classification (RvC) was applied to the problem of 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 particular 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 causes of faults that are not profound like the one that implies that applications for deposit organizations are fault-prone.

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.

In the future we intend to apply the proposed methodology to other software data sets [7] involving other software quality attributes in addition to defects. We will also experiment with methods that combine different classification algorithms such as Stacking [27] and Effective Voting [23] for the purpose of increasing the predictive performance of RvC.

7. References

[1] Aha, D., Kibler, D.W., and Albert, M.K., "Instance-based learning algorithms", *Machine Learning*, Vol. 6, 1991, pp. 37-66.

[2] Bishop, C.M., *"Neural Networks for Pattern Recognition"*, Oxford University Press, 1995.

[3] Cohen, W.W., "Fast Effective Rule Induction", In *Proceedings of the 12th International Conference on Machine Learning*, Morgan Kaufmann, 1995, pp. 115–123.

[4] Emam, K.E, Melo, W., Machado, J.C., "The Prediction of Faulty Classes Using Object-Oriented Design Metrics*", Journal of Systems and Software,* Vol. 56, 2001, pp. 63-75.

[5] Fenton, N., Neil, M., "A Critique of Software Defect Prediction Models", *IEEE Transactions on Software Engineering,* Vol 25(5), 1999, pp.675-689.

[6] Halstead, M.H, *"Elements of Software Science"*, Elsevier, North-Holland,1975.

[7] International Software Benchmarking Standards Group, *http://www.isbsg.org.*

[8] Kamiya, T., Kusumoto, S., Inoue, K., "Prediction of Fault-proneness at Early Phase in Object-Oriented Development", In *Proceedings of the 2nd International Symposium on Object-Oriented Real-Time Distributed Computing*, IEEE Computer Society,1999, pp. 253-258.

[9] Khoshgoftaar, T.M., Seliya, M., "Tree-Based Software Quality Estimation Models For Fault Prediction", In *Proceedings of the 8th IEEE International Conference on Software Metrics*, 2002, pp. 203-215

[10] Kohavi, R., "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection"`, In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995, pp. 1137-1145

[11] Kohavi, R, *"Wrappers for Performance Enhancement and Oblivious Decision Graphs"*. PhD Thesis. Department of Computer Science, 1995, Stanford University.

[12] Lanubile, F., Lonigro, A., Visaggio, G., "Comparing models for identifying fault-prone software components", In *Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering*, 1995, pp.312-319.

[13] McCabe, T.J, "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. 2 (4), 1976, pp. 308-320.

[14] Maxwell, K., *"Applied Statistics for Software Managers"*, Prentice-Hall, 2002.

[15] Neumann, R., Bibi, S., "Building fault prediction models from abstract cognitive complexity metrics- analyzing and interpreting fault related influences", *In Proceedings of the International Workshop on Software Measurement/Metrikon*, 2004, pp. 575-587.

[16] Platt, J., *"Fast Training of Support Vector Machines using Sequential Minimal Optimization"*, In Advances in Kernel Methods - Support Vector Learning, (Eds) B. Scholkopf, C. Burges, and A. Smola, MIT Press, 1998.

[17] Quinlan, J.R., "Learning with continuous classes", In *Proceedings. of Australian Joint Conf. on Artificial Intelligence*, 343-348, World Scientific, 1992, pp.343- 348.

[18] Quinlan, R.J., "*C4.5: Programs for Machine Learning*", Morgan Kaufman, 1993.

[19] Rousseeuw, P.J. and Leroy, A. M., *"Robust Regression and Outlier Detection"*, Wiley, 1997, New York.

[20] Smola, A.J, Scholkopf , B., *"A Tutorial on Support Vector Regression"*, NeuroCOLT2 Technical Report Series - NC2-TR-1998-030, 1998.

[21] Sturge, H., "The choice of Class Interval", *Journal of American Statistical Association*, pp 65-66, 1926.

[22] Torgo, L. and Gama, J., "Regression Using Classification Algorithms", *Journal of Intelligent Data Analysis*, Vol 1(4), pp. 275-292.

[23] Tsoumakas, G., Katakis, I. and Vlahavas, I. "Effective Voting of Heterogeneous Classifiers", In *Proceedings of the 15th European Conference on Machine Learning*, Italy, pp 465-476.

[24] Weiss, S. and Indurkhya, N., "Rule-based Machine Learning Methods for Functional Prediction". *In Journal of Artificial Intelligence Research*, Vol 3, 1995, pp. 383-403.

[25] Witten, I.H, and Frank, E., "Generating Accurate Rule Sets Without Global Optimization", *Proceedings of the 15th International Conference on Machine Learning*, 1998, pp. 144-151.

[26] Witten, I.H., and Frank E., J., *"Data Mining: Practical machine learning tools with Java implementations"*, Morgan Kaufmann, 1999, San Francisco.

[27] Wolpert, D., "Stacked Generalization", *Neural Networks*, No. 5, , 1992, pp. 241-259.

[28] Zhong, S., Khoshgoftaar, T.M., and Seliya, N., "Analyzing Software Measurement Data with Clustering Techniques", *IEEE Intelligent Systems*, Special issue on Data and Information Cleaning and Preprocessing, Vol (2), 2004, pp. 20-27.