DETECTING FAULT MODULES USING BIOINFORMATICS TECHNIQUES

GREGOR STIGLIC

Laboratory for System Design, University of Maribor, Smetanova 17,
2000 Maribor, Slovenia
gregor.stiglic@uni-mb.si

MATEJ MERTIK

Laboratory for System Design, University of Maribor, Smetanova 17,
2000 Maribor, Slovenia
matej.mertik@uni-mb.si

PETER KOKOL

Laboratory for System Design, University of Maribor, Smetanova 17,
2000 Maribor, Slovenia
kokol@uni-mb.si

MAURIZIO PIGHIN

Department of Mathematics and Informatics, University of Udine, Via delle Scienze 206,
33100 Udine, Italy
maurizio.pighin@uniud.it

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Many software reliability studies attempt to develop a model for predicting the faults of a software module because the application of good prediction models provides important information on significant metrics that should be observed in the early stages of implementation during software development. In this article we propose a new method inspired by a multi-agent based system that was initially used for classification and attribute selection in microarray analysis. Best classifying gene subset selection is a common problem in the field of bioinformatics. If we regard the software metrics measurement values of a software module as a genome of that module, and the real world dynamic characteristic of that module as its phenotype (i.e. failures as a disease symptoms) we can borrow the established bioinformatics methods in the manner first to predict the module behavior and second to data mine the relations between metrics and failures.

Keywords: Software engineering; fault-modules prediction; machine learning; bioinformatics.

1. Introduction

As the use of increasingly large and increasingly complex software systems is growing dramatically, the possibility of crises from failure is exponentially multiplying. The consequence is that the dependence on reliability and quality is becoming more and more essential [1] and thus software reliability is becoming a major concern not only for software engineers and computer scientists, but also for the society as a whole. Indeed, the use of software in high-assurance and mission-critical systems increases the need to develop and quantify measures of software quality. Unfortunately, high software reliability often involves prohibitive consumption of time and monetary resources for software development. Reliability improvement techniques may include more rigorous design and code reviews, more exhaustive testing, and focused re-engineering of high-risk segments of a software system.

More cost-effective strategy for software reliability improvement is identifying software modules that are likely to have more faults, before system tests and operations [2, 3, 4] in the manner to target quality improvement efforts only on such high-risk modules. Thereafter if faulty modules are detected or identified early during the development life-cycle [5], the number of faults likely to be detected during operations can be largely reduced resulting in more reliable and easy to maintain systems.

Software metrics-based [6] quality estimation models have been used to predict, for a software module, the number of faults expected or it's risk class such as fault-prone or not fault-prone [7, 8]. In literature, the former is referred to as software quality prediction modeling, whereas the latter is referred to as software quality classification modeling. A software quality classification model is composed of independent variables (software metrics) measured and collected during the software life-cycle, and a dependent variable (risk class) recorded later in the life cycle, such as during operations. Classification-based modeling for software quality estimation is a proven technique in achieving better software quality control and was widely discussed [9, 5, 1] and include techniques like optimal set reduction [10], logistic regression [11, 12], decision trees [13, 14, 15, 4], neural networks [3, 16, 17], statistical methods [4], data mining [18] and case-based reasoning [19]. Various approaches were compared and discussed in Pighin et al. [4] and Khoshgoftaar et al. [20]. Like in many other disciplines it has been found that no method outperforms all other in all experiments and all domains (so called »no free lunch theorem«).

Another problem shown by Pfleeger [21] is that it is not possible to estimate software quality from single software metric, such as lines of code, but that metrics like lines of code are only a part of the complexity picture. It is therefore necessary to capture a sufficiently large number of metrics related to quality to allow a valid prediction or classification, which further limits the usability of above methods. But when using many metrics we encounter still another problem called multicolinearity, which occurs, when the measured variables are too highly inter-correlated which is often the case in software metrics [22].

To summarize in developing a new software prediction classification method we should solve three problems:

- No free lunch theorem
- Analysis of a lot of metrics values
- Multicolinearity

Going through possible approaches we were inspired by a multi-agent based system presented in [23]. This system was initially built for identification of significant genes in microarray databases. Gene subset selection is a common problem in the field of bioinformatics. The recently developed microarray technology allows measurement of expression levels for thousands of genes simultaneously. Using classification or clustering techniques we are able to search for significant uncorrelated genes that can help us identifying different clinical states of the patient. The problem of multidimensionality, which is an integral part of microarray analysis, also opens up a new area in machine learning methods research. Assuming that we can easily adapt the multi-agent based system from bioinformatics for solving the classification problem of software reliability we can simply swap software metrics and genes.

We can see a problem of complexity metrics selection as a very similar to the above described problems from bioinformatics. During the last twenty years hundreds of metrics have been proposed for the software reliability assessment. The problem of a software engineer, who is aware of software quality control importance, is which of these metrics to include in checking the reliability of the developed software. If we regard the software measurements values of a software module as a genome of that module, and the real world dynamic characteristic of that module as its phenotype (i.e. failures as a disease symptoms) we can borrow the established bioinformatics methods in the manner first to predict the module behavior and second to data mine the relations between metrics and failures. So our approach tries to overcome the enumerated problems in the following way:

1. No free lunch theorem with a hybrid approach including genetic algorithms, case based reasoning and intelligent agent technology.
2. The approach can analyze thousands of different metrics.
3. The approach selects uncorrelated metrics and in addition we introduced a new uncorrelated metric called alpha [4].

2. Application in Bioinformatics

Here, we first present our research made in the field of bioinformatics, which will be used later in the fault software module prediction problem.

2.1. *Multi-Agent Based Classification*

Multi-agent systems offer a new paradigm to organize applications in the field of artificial intelligence and also machine learning. We can see agents as intelligent computational entities that are able to collaborate and solve problems in groups. In our approach we employ some ideas from agents and multi-agent systems to solve the problem of gene expression attribute selection and classification.

Each agent in the system has to follow some basic rules of acting in the large search space. In our case every possible pair of genes defines a point in the search space. This means that we would have to evaluate n^2 points in the search space for each agent, because each agent can have its own settings. To avoid searching through the whole search space we define basic rules that help our agent based method in solving the problem faster. Each cycle of agent's activity consists of two parts. In the first part an agent is exploring the search space using the Monte Carlo method. Among all numerical methods that rely on n-point evaluations in m-dimensional space to produce an approximate solution, the Monte Carlo method has absolute error of estimate that decreases as $n^{-1/2}$ whereas, in the absence of exploitable special structure all others have errors that decrease as $n^{-1/m}$ at best. Using this method an agent samples a pre-defined number of points in search space and stores the location of the best classifying pair of genes.

In the second part we search around the best solution by following vertical and horizontal lines from the best classifying position. This way an agent is using one of the genes selected by the best classifier so far and tries to improve the current best result of classification. At the end of the cycle agents exchange information about their best solutions and start the new cycle.

Fitness level of an agent is expressed by the ability of finding the best combination of genes for classification that is measured by the accuracy of classification using selected gene-pair. Each agent presents a pair of attributes (genes) in the final ensemble of classifying agents. We use the k-nearest neighbors (k-NN) algorithm to estimate the accuracy of classification for each agent.

2.2. *Results of Microarray Classification*

The most exciting result of microarray technology research in the past has been the demonstration that patterns of gene expression can distinguish between tumors of different anatomical origin.

Colon cancer database we were using was first presented and analyzed by Alon et al. in [24] and was later frequently used as a benchmark for testing the accuracy of gene expression classification methods. In this study, gene expression of more than 6500 human genes in 40 tumor and 22 normal tissue samples were analyzed using microarray scanner. In our database we used 2000 statistically most significant genes that were chosen in the above-mentioned paper [24].

From all investigated methods we found only one that managed to classify all samples correctly using LOOCV (Leave-one-out cross-validation). It was performed by Fujarewicz and Wiench in [25]. They achieved these results using a combination of recursive attribute selection (RFR) and support vector machines (SVM) methods.

Using evolutionary multi-agent based system combined with simple nearest neighbor classifiers we got 100% accuracy which is comparable to other much more complex methods. Our system is also very useful as an attribute selection method and can effectively reduce the number of needed genes for discrimination between tissue classes. We prove this by comparing our result of no misclassified samples in Colon cancer dataset to the result by Fujarewicz and Wiench [25] where support vector machines were used for classification. We matched their result using only three pairs of genes combined in an ensemble of nearest neighbor classifiers (Table 1).

Table 1. Ensembles classification accuracy (Colon Cancer Microarray Database).

#	Classification Accuracy	Classifier Used	Used Gene Ids
1	95.2 %	2-NN	1967, 1449
2	93.5 %	4-NN	323, 625
3	91.9 %	5-NN	1423, 1472
4	100.0 %	Ensemble	#1, #2, #3

3. Detecting Software Modules Faults

As mentioned earlier we used the multi-agent based method, which was developed for classification of multi-dimensional datasets like gene expression analysis, also in the field of software modules fault detection.

3.1. *Classification Method*

Because of the multi-dimensionality and type of data in analyzed database, we chose k-NN, which is a simple but effective classification algorithm. It is widely used in machine learning and has numerous variations [26, 27]. Given a test sample of unknown label, it finds k nearest neighbors in the training set using Euclidean distance (*d*) and assigns the label of the test sample according to the labels of those neighbors. We use five different types of classifiers ranging for 1 to 5 nearest neighbors used for classification. To ensure a majority in the voting process we use weighting technique where neighbors vote is weighted by its distance to the test sample. The weight assigned to each vote is *1/d*.

As mentioned earlier we also want to use our system for attribute selection purposes. Our aim is to select the smallest possible set of genes and still achieve the highest possible accuracy of classification. Therefore we try to combine the best agents in an ensemble of agents.

Ho introduced the idea of ensembles of k-Nearest Neighbors (k-NN) classifiers where the variety in the ensemble is generated by selection of different attribute subsets for each ensemble in [28]. Since she generates these attribute subsets randomly she refers to these different subsets as random subspaces. She points to the ability of ensembles of k-NN classifiers based on different attribute subsets to improve on the accuracy of individual k-NN classifiers because of the simplicity and accuracy of the k-NN approach. She shows that an ensemble of k-NN classifiers based on random subsets improves on the accuracies of individual classifiers on a hand-written character recognition problem.

3.2. *Experimental Design*

In our research we tested our proposed method on the database consisting of software modules extracted from a large Management System which is still in use and was developed, in a cycle of life of more that 10 years, by a medium-size Italian software house. The modules were classified as dangerous (high occurrence of errors during the software usage) and normal (lower or no errors during run-time). The threshold between "dangerous" and "normal" modules was set based on distribution of samples in the database. This way we got two classes with approximately the same number of samples. Application was written in C programming language using relation database and running on UNIX operating system. Number of lines of code was defined by standardized rules of the software house and is in the range of 300 to 600 lines for most of the files. Database consisted of 217 samples (modules) and 168 attributes (metrics). We obtained metrics using Resource Standard Metrics (RSM) version 6.10 and our own parser for metrics measurements. Metrics were classified into different groups such as: number of constants, variables, preprocessor instructions, functions, comments, control structures and standardized metrics used in software engineering.

When considering which testing method to use, most papers use a hold-out procedure, using a part of the samples to train a predictive model and using the rest of instances as a test set to estimate classifier accuracy. Another possibility is using n-fold cross-validation, which is typically implemented by running the same learning system n times, each time on a different training set of size $(n-1)/n$ times the size of the original data set. Because of this computational cost, cross-validation is sometimes avoided, even when it is agreed that the method would be useful. This is often the case in usual machine learning problems while in microarray classification the computational cost is usually not a problem because of a small number of samples. That is why a lot of authors use leave-one-out cross-validation method, in which one sample in the training set is withheld, the remaining samples of the training set are used to build a classifier to predict the class of withheld sample, and the cumulative error is calculated. LOOCV was often criticized, because of higher error variance in comparison to 5 or 10-fold cross-validation [29]. A recent study by Braga-Neto and Dougherty [30] shows that LOOCV can still be considered as the most optimal classifier for microarray datasets, because they have not been able to verify the substantial difference in performance among mentioned methods. In all our tests we use LOOCV for classifier accuracy estimation.

3.3. *Learning Algorithms*

To compare classification results of our proposed method we used 3 different classification methods that were used for comparison of the obtained classification accuracy results.

J48 Decision Tree is WEKA toolkit [31] implementation of C4.5 decision tree algorithm proposed by Quinlan [32]. It is used for inducing classification rules in the form of decision trees from a set of given examples. We are given a set of records where each record has the same structure, consisting of a number of attribute/value pairs. One of these attributes represents the category of the record. The problem is to determine a decision tree that on the basis of answers to questions about the non-category attributes predicts correctly the value of the category attribute also called class attribute.

Neural Networks represent the "black-box" classifiers where inner structure of the classification method cannot be observed. The idea of neural networks dates back to the 1940s, when McCulloch and Pitts [33] developed the first neural model. This was followed in by the perceptron model, devised by Rosenblatt [34], which generated much interest because of its ability to solve some simple pattern classification problems. We use multi-layer perceptron type of neural network that consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer.

Logistic Regression with Ridge Estimator is a WEKA toolkit classifier used for building two-class multinomial logistic regression models with a ridge estimator. It is based on the original logistic regression ridge estimator by le Cessie and van Houwelingen [35].

3.4. *Results*

Because of nearest neighbour classification, we have to normalize all data in the database so that we can easily compare pairs of attributes. This way we also prevent attributes with initially large ranges from outweighing attributes with initially smaller ranges.

Our first experiment followed the example from the bioinformatics where usually leave-one-out cross-validation is used for classification accuracy estimation. This is done because datasets in microarray analysis consist of a very small number of samples (usually less than 100).

In table 2 we can see the results of the best classifying agents on the software modules fault detection database. From this table we can see that lines without comment count metric is the one that is present in almost every combination of two metrics used by best agents.

Table 2. Classification accuracy of the best classifying agents.

#	Classification Accuracy	Classifier Used	Used Metrics
1	88.0 %	5-NN	lines_without_comment, functional_instructions
2	87.1 %	3-NN	str_function_parameter, lines_without_comment
3	86.6 %	3-NN	lines_without_comment, int
4	85.2 %	3-NN	lines_without_comment, alpha
5	85.2 %	1-NN	implemented_functions, lines_without_comment
6	84.8 %	3-NN	significant_comments, pointers

7	84.3 %	4-NN	lines_without_comment, functions_with_control_structure
8	84.3 %	1-NN	lines_without_comment, casting_operators
9	83.9 %	3-NN	pointers, words_in_comments
10	83.9 %	4-NN	or, lines_without_comment

In the next step we tried to find the optimal combination of best performing agents. Therefore we constructed different ensembles from ten best classifying agents. Table 3 represents the results of classification using different combinations of agents. By combining different three classifiers from the ten best classifiers we got the best solution that was able to classify the samples with the accuracy of over 90 percent.

Table 3. Classification accuracy of the best ensembles.

#	Classification Accuracy	Agents Used	Ensemble
1	88.9 %	1 – 3	Top 3
2	89.9 %	1 – 5	Top 5
3	87.6 %	1 – 7	Top 7
4	89.4 %	1 – 9	Top 9
5	91.2 %	2, 3, 4	Combination of 3

Observing the best classifying ensembles we can notice that the best performing ensemble consists of agents number 2, 3 and 4. All of those also include the already mentioned most important metric that counts the number of lines without comment (LWC). It is obvious that this attribute holds some important information, because it was chosen by our method so often. We could compare the same attribute with comment lines count (CLC) metric (Fig. 1). Both are linearly correlated in fault or normal classified software modules. We can only spot the difference in number of lines in both cases. In fault-prone modules we have on average twice as many lines with or without comment as in non-problematic modules. Observing both charts we can notice that the slope of the trend line is lower in the right (non-fault modules) chart compared to the fault-prone modules chart. Therefore we can say the lower the slope the better-commented (i.e. less error-prone) module we have.

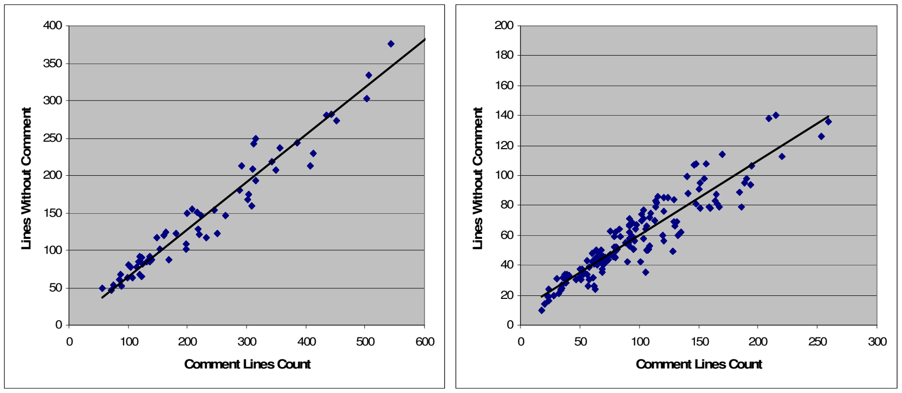

Fig. 1. Comparison of linear correlation between LWC and CLC metrics within fault-prone and normal modules.

In our second experiment we compared the accuracy of our method to other classification methods using 10 and 20-fold cross-validation. Our results were compared to the following methods from the WEKA toolkit:

- J48 (WEKA implementation of C4.5 decision trees)
- Neural Networks
- Logistic Regression

If not otherwise stated we used default settings of the toolkit. Results of classification accuracy comparison are summarized in table 4.

Table 4. Classification accuracy comparison.

Method	10-fold CV Accuracy	20-fold CV Accuracy
Multi-agent	78.64 %	77.73 %
C4.5 Trees	76.96 %	77.88 %
Neural Networks	79.80 %	79.26 %
Logistic	76.04 %	76.04 %

Although the accuracy of our method is not the highest we show that the results of our method can match other classification methods. The difference between 10 and 20-fold cross-validation results is small, therefore we cannot say which method is better according to different cross-validation method. We have to know that our multi-agent based system selected only six best classifying attributes in each fold while other methods used much more if not all (e.g. neural networks) the attributes. Neural networks method also had a significantly higher computation time.

4. Discussion

Accuracy of a software quality classification (two-group) model is measured in terms of its Type I and Type II misclassification error rates. A Type I error occurs when a not fault-prone module is classified as fault-prone, whereas a Type II error occurs when a fault-prone module is classified as not fault-prone. Thus, accuracy is indicated by how well the class predictions for software modules are realized later in the life cycle, i.e., during testing or operations. Furthermore, the balance between the two misclassification error rates may affect the usefulness of the calibrated classification model.

Recent research has indicated that published software quality classification modeling methods may not consistently produce a useful balance between the Type I and Type II error rates. If either type of misclassification rate is high, the model is generally not useful to guide software improvement efforts. In the case when the Type I error rate is high, then targeting of enhancement efforts will be inefficient. On the other hand, if the Type II error rate is very high, then many fault-prone modules will not be detected for enhancements. Therefore, a preferred balance, as per the needs of the project, between the two types of misclassifications is desired. This is especially important for high-assurance systems that usually have very few faulty modules. In such cases, software quality classification modeling methods would be inappropriate for obtaining a preferred balance between the two error rates.

The medical literature has recognized above problem and they widely use so called Receiver Operating Characteristic (ROC) plots to asses the quality of various classifiers. The ROC technique developed in signal processing and the term Receiver Operating Characteristic refers to the performance (the 'operating characteristic') of a human or mechanical observer (the 'receiver') engaged in assigning cases into dichotomous classes [36]. To make a ROC plot we need to calculate the sensitivity that is a measure of how good the classifier is at picking out patients affected with disease (It is simply the Type I error from above) and specificity which is the ability of the classifier to pick out patients who do NOT have the disease (Type II error). The ROC plot shows the relation between sensitivity and specificity. The lesser the area is under the ROC curve, the less useful is the classifier in discriminating between the two populations. We could also say that area under ROC is a measure for general quality of a classifier.

To find out the quality of our classifier we performed tests comparing our multi-agent based system and decision trees using C4.5 algorithm. The tests were done using 25 runs of the C4.5 and our proposed method on bootstrapped data using 10-fold cross-validation. In the Fig. 2 and Fig. 3 we can observe ROC curves for all classification systems used in our accuracy tests.

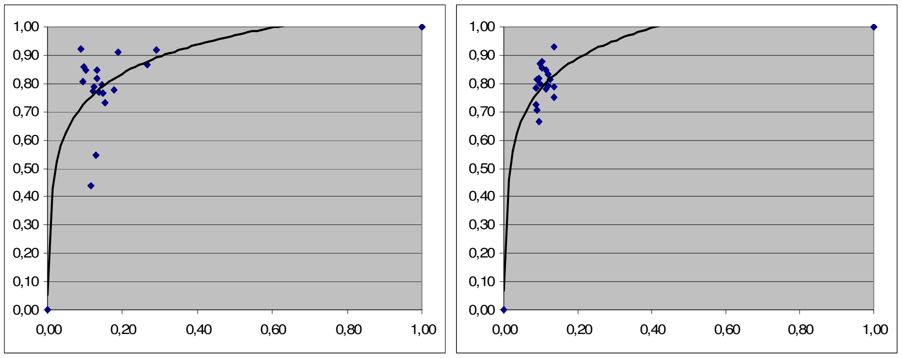

Fig. 2 ROC curve charts for C4.5 decision trees (left) and multi-agent based classification (right).

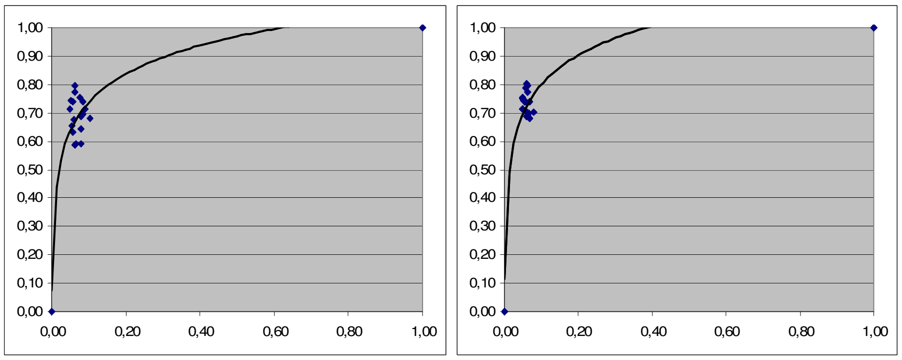

Fig. 3 ROC curve charts for neural networks (left) and linear logistic regression based classification (right).

When we calculate the area under the ROC curve, we can see that multi-agent based classification system achieved a result of 0.94 compared to 0.89, 0.91 and 0.94 that were achieved by decision trees, neural networks and logistic regression methods. An area of 1 represents a perfect test; an area of 0.5 represents a worthless test. A rough guide for classifying the accuracy of such a test is the traditional academic point system, which classifies all the results better than 0.8 as good, while all results of over 0.9 are considered excellent.

5. Conclusions and Further Work

Our paper proposes multi-agent based classification method using simple k-NN classifiers for prediction of the correlation between software metrics and software fault-proneness. The proposed technique tries to minimise the number of used metrics to estimate the fault-proneness of the software module. Our approach demonstrates its efficiency and effectiveness in dealing with high dimensional data for classification. But following »no free lunch theorem« mentioned in the beginning of the paper, we can also observe that at the cost of effectiveness shown in small number of metrics used in final result, we can still find methods that gain better accuracy on the same dataset.

In the future we plan to do some more research on the current database and try to look for some attribute dependency and not only classification accuracy. We would also like to improve our method by using more sophisticated classifiers incorporated in agents. This way time complexity will increase but we may expect to get useful solutions using only three or four attributes. The presented method can help a software engineer in selection of software complexity metrics at the validation of software modules reliability. One of our future plans is also presentation of the results in a more user-friendly way, so that software engineers will easily understand why they were selected.

Acknowledgments

This work has been supported by the bilateral project MERCOS (SLO-ITA 8A/2002-2005) – Measuring Reusability by Complexity Similarity. Part of the research was financed by Grant No. 3311-04-828131 (project MAGSIS) from the Slovenian Ministry of Higher Education, Science and Technology and Slovenian Research Agency.

References

1. N. F. Schneidewind, Software metrics validation: Space shuttle flight software example. Annals of Software Engineering 1 (1995), 287–309.
2. C. Ebert, Classification techniques for metric-based software development, Software Quality Journal 5(4) (1996) 255–272.
3. T. M. Khoshgoftaar, E. B. Allen, J. P. Hudepohl and S. J. Aud, Applications of neural networks to software quality modeling of a very large telecommunications system. IEEE Transactions on Neural Networks 8(4) (1997) 902–909.
4. M. Pighin, V. Podgorelec and P. Kokol, Fault Thresholds Definition with Linear Programming Methodologies, Empirical Software Engineering, Kluwer Academic Publication, Norwell-MA (USA), Vol. 8, Number 2, June 2003, pp. 117-138.
5. M.C. Ohlsson, M. Helander and C. Wohlin, Quality improvement by identi.cation of fault-prone modules using software design metrics, in Proceedings of International Conference on Software Quality, Ottawa, Ontario, Canada, 1997, pp. 1–13.
6. V. R. Basili, L. C. Briand and W. L. Melo, A validation of object-oriented design metrics as quality indicators, IEEE Transactions on Software Engineering 22(10) (1996) 751–761.
7. L. C. Briand, W. L. Melo and J. Wust, Assessing the applicability of fault-proneness models across object-oriented software projects, IEEE Transactions on Software Engineering 28(7) (2002) 706–720.
8. Y. Ping, T. Systa and H. Muller, Predicting fault-proneness using OO metrics, an industrial case study, in Proceedings: 6th European Conference on Software Maintenance and Reengineering. Budapest, Hungary, March, 2002, pp. 99–107.
9. M. C. Ohlsson and P. Runeson, Experience from replicating empirical studies on prediction models. In Proceedings: 8th International SoftwareMetrics Symposium. Ottawa, Ontario, Canada: IEEE Computer Society, June, 2002, pp. 217–226.

10. L. C. Briand, V. R. Basili and C. J. Hetmanski, Developing interpretable models with optimized set reduction for identifying high-risk software components, IEEE Transactions on Software Engineering 19(11) (1993) 1028–1044.
11. T. M. Khoshgoftaar and E. B. Allen, Logistic regression modeling of software quality, International Journal of Reliability, Quality and Safety Engineering 6(4) (1999) 303–317.
12. N. F. Schneidewind, Investigation of logistic regression as a discriminant of software quality, in Proceedings of 7th International Software Metrics Symposium, London UK: IEEE Computer Society, April, 2001, pp. 328–337.
13. T. M. Khoshgoftaar, E. B. Allen, W. D. Jones and J. P. Hudepohl, Classification tree models of software-quality over multiple releases, IEEE Transactions on Reliability 49(1) (2000) 4–11.
14. A. Suarez and J. F. Lutsko, Globally optimal fuzzy decision trees for classification and regression, Pattern Analysis and Machine Intelligence 21(12) (1999) 1297–1311.
15. R. Takahashi, Y. Muraoka and Y. Nakamura, Building software quality classification trees: Approach, experimentation, evaluation, in Proceedings of 8th International Symposium on Software Reliability Engineering, Albuquerque, NM, USA: IEEE Computer Society: November, 1997, pp. 222–233.
16. R. Paul, Metric-based neural network classification tool for analyzing large-scale software, in Proceedings of International Conference on Tools with Artificial Intelligence, Arlington, Virginia, USA: IEEE Computer Society, November, 1992, pp. 108–113.
17. N. J. Pizzi, A. R. Summers and W. Pedrycz, Software quality prediction using median-adjusted class labels. In Proceedings of International Joint Conference on Neural Networks, volume 3, Honolulu, Hawaii, USA: IEEE Computer Society, May, 2002, pp. 2405–2409.
18. D. Scott, A. Meeks, M. Last, H. Bunke and A. Kandel, Data mining in software metrics databases, Fuzzy Sets and Systems, 145(1) (2004) 81-110.
19. T.M. Khoshgoftaar, N. Seliya, Analogy-Based Practical Classification Rules for Software Quality Estimation, Empirical Software Engineering, 8 (2003) 325–350.
20. T.M. Khoshgoftaar, N. Seliya, Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study, Empirical Software Engineering, 9 (2004) 229–257.
21. S. L. Pfleeger, Software Engineering: Theory and Practice. Prentice-Hall Inc., 1998.
22. N. Hanebutte, C. S. Taylor and R. R. Dumke, Techniques of Successful Application of Factor Analysis in Software Measurement, Empirical Software Engineering, 8 (2003), 43–57.
23. G. Stiglic and P. Kokol, Using Multi-Agent System for Gene Expression Classification, in Proceedings of 26 th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS 2004), San Francisco, CA, 2004, 2952-2955.
24. U. Alon et al., Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays, Proc. Natl. Acad. Sci., 96(1), 6745-6750.
25. K. Fujarewicz and M. Wiench, Selecting differentially expressed genes for colon tumor classification, Int. J. Appl. Math. Comput. Sci., 13(3), 327-335.
26. R. Duda, P.E. Hart and D.G. Stork, Pattern classification, Wiley, 2001.
27. C. Yeang, Molecular Classification of Multiple Tumor Types, Bioinformatics, 17(l) (2001), 316-322.
28. T.K. Ho, Nearest Neighbours in Random Subspaces, in Proceedings of 2nd International Workshop on Statistical Techniques in Pattern Recognition, 1998, 640-648.
29. T. Hastie, R. Tibshirani and J. Friedman, The Elements of Statistical Learning, Springer, 2001.
30. M. Braga-Neto and E.R. Dougherty, Is cross-validation valid for small-sample microarray classification?, Bioinformatics, 20(3) (2004) 374-380.
31. I.H. Witten and E.Frank, Data Mining: Practical machine learning tools with Java implementations, Morgan Kaufmann, San Francisco, 2000.

32. J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kauffman, 1993.
33. W.S. McCulloch and W.H. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics, 5 (1943) 115-133.
34. F. Rosenblatt, A comparison of several perceptron models, Self-Organizing Systems, Spartan Books, Washington, DC, 1962.
35. S. le Cessie, J.C. van Houwelingen, Ridge Estimators in Logistic Regression, Applied Statistics, 41 (1)(1992) 191-201.
36. M. H. Zweig and G. Campbell, Receiver-operating characteristic (ROC) plots: A fundamental evaluation tool in clinical medicine, Clin. Chem. 39 (1993) 561–577.