Empirical Assessment of Machine Learning based Software Defect Prediction Techniques

Venkata U.B. Challagulla, Farokh B. Bastani, I-Ling Yen
Department of Computer Science
University of Texas at Dallas, TX 75083-0688
[uday, bastani, ilyen]@utdallas.edu

Raymond A. Paul
OASD/C31/Y2K
Department of Defense
Ray.Paul@osd.pentagon.mil

Abstract

The wide-variety of real-time software systems, including telecontrol/telepresence systems, robotic systems, and mission planning systems, can entail dynamic code synthesis based on runtime mission-specific requirements and operating conditions. This necessitates the need for dynamic dependability assessment to ensure that these systems will perform as specified and will not fail in catastrophic ways. One approach in achieving this is to dynamically assess the modules in the synthesized code using software defect prediction techniques. Statistical models, such as Stepwise Multi-linear Regression models and multivariate models, and machine learning approaches, such as Artificial Neural Networks, Instance-based Reasoning, Bayesian-Belief Networks, Decision Trees, and Rule Inductions, have been investigated for predicting software quality. However, there is still no consensus about the best predictor model for software defects.

In this paper, we evaluate different predictor models on four different real-time software defect data sets. The results show that a combination of 1R and Instance-based Learning along with the Consistency-based Subset Evaluation technique provides a relatively better consistency in accuracy prediction compared to other models. The results also show that "size" and "complexity" metrics are not sufficient for accurately predicting real-time software defects.

1. Introduction

With real-time systems becoming more complex and unpredictable, partly due to increasingly sophisticated requirements, traditional software development techniques might face difficulties in satisfying these requirements. Future real-time software systems may need to dynamically adapt themselves based on the run-time mission-specific requirements and operating conditions. This involves dynamic code synthesis that generates modules to provide the functionality required to perform the desired operations in real-time. Tele-control/telepresence, robotics, and mission planning systems are some of the examples that fall in this category. However, this necessitates the need to develop a real-time assessment technique that classifies these dynamically generated systems as being faulty/fault-free. Some of the benefits of dynamic dependability assessment include providing feedback to the operator to modify the mission-objectives if the dependability is low, the possibility of masking defects at run-time, and the possibility of pro-active dependability management. One approach in achieving this is to use software defect prediction techniques that can assess the dependability of these systems using defect metrics that can be dynamically measured.

A variety of software defect prediction techniques have been proposed, but none has proven to be consistently accurate. These techniques include statistical methods, machine learning methods, parametric models, and mixed algorithms. Obviously, there is a need to find the best prediction technique for a given prediction problem in context, and, perhaps, conclude that this problem is largely unsolvable.

The availability of a large number of divergent prediction techniques poses some interesting questions that need great attention to details. Some of these are,
1) What characteristics of the underlying defect data influence the prediction accuracy?
2) Can we systematically determine the relationship between the accuracy, choice of prediction system, and the data set characteristics?
3) How many times do we need to repeat the sampling and validation procedures before acquiring the required confidence intervals on the model?
4) Can we combine different techniques for better prediction? If so, what are the trade-offs?
5) Do software defect data have some characteristics (such as, standard deviation) in common?

This paper provides a critical review of software defect prediction techniques with special emphasis on machine learning based methods. These techniques are applied to three real-time defect data sets obtained from NASA's MDP (Metrics Data Program) data repository. Section 2 explains the probable characteristics of the defect data that influence the accuracy of the prediction techniques. Section 3 reviews some of the related works in this field. Section 4 details the software used and defines the contents of the data sets used for the experiment. Section 5 explains the different prediction techniques and their application to the above data sets. Section 6 compares the relative performances of the prediction techniques under scrutiny. Section 7 concludes by giving some future directions.

2. Characteristics of Software Data

All defects frequently exhibit non-normal characteristics, such as skewness, unstable variances, collinearity, and excessive outliers [3]. The following are some of the characteristics of the software data sets that are considered in our analysis:
1) Number of cases – a) Small ($n <= 500$), b) Medium ($500 < n < 10000$), c) Large ($n >= 10000$);
2) Number of features – a) Small ($p <= 6$), b) Medium ($6 < p < 20$), c) Large ($p >= 20$);
3) Distribution of values – a) Skewed, b) Outliers;
4) Independence of features – a) Independent, b) Multicollinearity;
5) Feature type – a) Discrete, b) Continuous.

3. Background and Related Work

Statistical, machine learning, and mixed techniques are widely used in the literature to predict software defects. Khoshgoftaar [4] used zero-inflated Poisson regression to predict the fault-proneness of software systems with a large number of zero response variables. He showed that zero-inflated Poisson regression is better than Poisson regression for software quality modeling. Munson and Khoshgoftaar [5, 6] also investigated the application of multivariate analysis to regression and showed that reducing the number of "independent" factors (attribute set) does not significantly affect the accuracy of software quality prediction. Lesley, Barbara, and Susan [3] investigated the efficacy of residual analysis, multivariate regression, and classification and regression trees (CART) for the analysis of software data. They found that multivariate regression analysis performed better if the data has only minor skewness, and residual analysis performed the best for data with severe heteroscedasticity. Yong Wang [7] developed PACE regression, and showed that it performs the best compared with other regression models for high dimensional data. Gaincarlo, Milorad, and Witold [8] investigated the application of Poisson regression model and binomial regression models to deal with the non-normally distributed software defect data. They concluded that negative binomial distribution dealt with over dispersion of data more effectively. Other statistical techniques that have been investigated include logistic regression techniques analyzed by Schneidewind [9]. He showed that the use of logistic regression techniques alone is of limited value. He proposed to combine logistic regression with Boolean discriminant functions for better software quality analysis.

Apart from the statistical methods described above, some researchers have recently started investigating the application of machine learning techniques to software quality analysis. Shepperd and Kododa [2] used simulation to compare software prediction using stepwise regression, rule induction, case-based reasoning (CBR), and artificial neural networks (ANN). They concluded that stepwise regression performed better with continuous target function, while the other machine learning approaches performed better for discontinuous target functions. They favored CBR since it appeared to be the best all round predictor by a small margin. Mair, Kadoda, Lefley, Schofield, Shepperd, and Webster [10] investigated the above prediction models on real software data. They compared these techniques in terms of accuracy, explanatory value, and configurability. They concluded that the explanatory value of case-based reasoning and rule induction gives them an advantage over neural nets, which have problems of configuration. If we just consider accuracy, neural nets performed better on their data sets. Aljahdali, Sheta, and Rine [11] compared regression with neural nets for prediction of software reliability and concluded that neural nets provided low errors than regression models in most of the cases. Menzies, Ammar, Nikora, and Stefano [12] compared decision trees, naïve Bayes, and 1-rule classifier on the NASA software defect data. A clear trend was not observed and different predictors scored better on different data sets. However, their proposed ROCKY classifier outscored all the above predictor models. Emam, Benlarbi, Goel, and Rai [13] compared different case-based reasoning classifiers and concluded that there is no added advantage in varying the combination of parameters (including varying nearest neighbor and using different weight functions) of the classifier to make the prediction accuracy better. Morasca and Ruhe [14] compared logistic regression and rough sets for prediction of software measurement data. They found that the prediction accuracy of both these techniques vary only within a small margin. They state that performing the analysis on any data set with both these classifiers simultaneously would increase our confidence in the prediction. Some other techniques include using self organizing maps along with genetic decision trees [15] and 1-rule classification [16].

Researchers have also investigated methods to combine machine learning and statistical methods. Neumann [17] used principal component analysis (PCA) to enhance the performance of neural networks. However, from [12], we find that feature subset selection (FSS) methods, such as correlation-based feature selection technique (CFS) [18], consistency-based subset evaluation technique (CBS) [19], information gain attribute ranking technique [20], and relief techniques [21] perform better than principal component analysis. Hence, the cumbersome procedure of calculating and deducing from principal components can be possibly eliminated.

While the above prediction models have their advantages and disadvantages, Fenton, Neil, and Krause [22] argued that these models can lead to inappropriate risk management decisions, since they do not effectively take dependencies between the "attributes" into consideration. So they proposed Bayesian belief networks (BBN) as an alternative to current approaches. However, constructing BBN topologies is not an easy task. They assert that a better science of software engineering can be achieved by investigating "software decompositions" that capture these casual relations effectively. In true sense, the Bayesian belief network method is gaining popularity as an effective approach for defect prediction. In fact, Zhang [23] lists BBN as the only valid approach for software defect prediction. Mitchell [24] provides a general comprehensive discussion of the machine learning algorithms discussed above.

4. Public Domain Defect Data

The real-time defect data sets used in this paper are shown in Table 1. They can be accessed from the NASA's MDP (Metric Data Program) data repository, available online at http://mdp.ivv.nasa.gov. The CM1 data is obtained from a spacecraft instrument, written in C, containing approximately 506 modules. The JM1 data is obtained from a predictive ground system project, written in C++, containing 10879 modules. The KC1 data is obtained from a science data processing project coded in C++, containing 2108 modules. The PC1 data is collected from a flight software system coded in C, containing 1108 modules. All these data sets varied in the percentage of defect modules, with the CM1 dataset containing the least number of defect modules and the KC1 dataset containing the largest. Table 2 shows the different types of predictor software metrics (independent variables) used in our analysis. These complexity and size metrics include well known metrics, such as Halstead, McCabe, line count, operator/operand count, and branch count metrics. Halstead metrics are sensitive to program size and help in calculating the programming effort in months. The different Halstead metrics include length, volume, difficulty, intelligent count, effort, error, and error estimate. McCabe metrics measure code (control flow) complexity and help in identifying vulnerable code. The different McCabe metrics include cyclometric complexity, essential complexity, design complexity and lines of code. The target metrics (dependent variables) are the "Error Count" and "Defects". "Error Count" refers to the number of module errors associated with the corresponding set of predictor software metrics, while "Defect" metric refers to whether the module is fault-prone or fault-free.

Regarding the software, we used the freely available WEKA machine learning tool kit and SAS tool to conduct these experiments. It is important to note that the prediction techniques investigated in this project are limited only by the available techniques in WEKA. SAS is primarily used to compute the characteristics of the data (including skewness and variance) and to compute the principal components.

5. Prediction Techniques

We evaluated empirically the accuracy of predicting the number of defects using machine learning and statistical prediction systems. We used 70% of the data as training data and 30% as the test data. Our input attributes (input data) are treated as continuous values, while the output takes discrete or continuous values depending on the classifier used. Table 3 shows the classification of classifiers based on the input/output data. Decision trees, Naïve-Bayes classifier, logistic regression, nearest neighbor, and 1-rule take both continuous and discrete values for independent metrics, but the target metric should be of discrete type. Regression also takes both continuous and discrete inputs, while outputting the desired metric in a continuous form.

Table 1. Data sets used in this study

Project	# modules	% with defects	Language
CM1	506	9.5%	C
JM1	10879	19.3 %	C++
KC1	2108	15.4 %	C++
PC1	1108	6.8 %	C

Table 2. Metrics group used in this study

Metric Type	Definition
McCabe	CC, EC, DC, ELOC
Halstead	N, V, D, I, E, B, L, T
Line Count	BLOC, CLOC, CCLOC
Operator	UNOD, UNOT, NOD, NOT
Branch Count	BC

CC – Cyclometric Complexity, EC – Essential Complexity, DC – Design Complexity, ELOC – Lines of Code, N – Length, V – Volume, D – Difficulty, I – Intelligent Content, E – Effort, B – Error Estimate, L – Level, T – Programming time, BLOC – Lines of Blank, CLOC – Lines of Comment, CCLOC – Lines of Code and Comment, UNOD – Unique Operands, UNOT – Unique Operators, NOD – Total Operands, NOT – Total Operators, BC – Branch Count

Table 3. Predictive models used in our analysis

Learning Method	Type of Input	Type of Output
Decision Trees, Naïve-Bayes, Logistic Regression, Nearest Neighbor, 1-Rule	C/D	D
Regression	C/D	C
Neural Networks	C/D	C/D

C – Continuous, D – Discrete

Neural networks can accept inputs and produce outputs in both continuous and discrete modes. The target metric "Error count" can be approximated as either continuous or discrete metric depending on the classifier used, while the "Defect" metric serves only the discrete case.

The WEKA software computes the mean absolute error, root mean squared error, relative absolute error, and root relative squared error. However, the most commonly reported error is the mean absolute error / root mean squared error. The root mean squared error is more sensitive to outliers in the data than the mean absolute error. In order to minimize the effect of outliers on the classification methods, we choose Mean Absolute Error as the standard error model in all the following analysis. The second column of Table 4 shows the error in accurately predicting the "Error Count" of the original CM1 data set by different predictor models. Due to space constraints, we use abbreviations in the table to represent different predictive methods: Learning Method (LM), Mean Absolute Error (MAE), Linear Regression (LR), Pace Regression (PR), Support Vector Regression (SVR), Neural Network for continuous goal field (NNC), Support Vector Logistic Regression (SVLR), Neural Network for discrete goal field (NND) with 12 hidden layers, Logistic Regression (LoR), Naïve Bayes (NB), Instance Based Learning for 10 nearest neighbors (IBL), J48 Trees (JDT), and 1-Rule (1R).

Table 4. Error shown by different models on predicting "Error Count" on CM1 data set

(LM)	MAE (Original)	MAE (PCA)	MAE (CFS)	MAE (CBS)
LR	0.2325	0.235	0.2369	0.2292
PR	0.2613	0.2385	0.2343	0.2325
SVR	0.2216	0.2346	0.2216	0.2192
NNC	0.2799	0.1773	0.3327	0.2406
SVLR	0.0569	0.2778	0.0584	0.0588
NND	0.0688	0.0634	0.067	0.0643
LoR	0.0712	0.0591	0.0626	0.0621
NB	0.138	0.2808	0.0925	0.1482
IBL	0.0543	0.0557	0.0527	0.053
JDT	0.0561	0.0591	0.0583	0.0553
1R	0.0351	0.0329	0.0329	0.0351

We also applied the prediction techniques to the prediction of faulty modules. While the "Error Count" attribute is approximated to a continuous or discrete variable, we felt that both these assumptions might not hold in practice. The range of values of the attribute is not very large making the continuous assumption invalid. Also, the number of data points in each of the discrete intervals is abnormally variant, making the discrete distribution widely skewed. Hence, the accuracy predicted by these discrete models might not be affected by the intervals with sparse data, thus giving better prediction results. We concluded that the better way is to analyze the modules as faulty or faultless. A module with one or more defects is considered faulty. In this way, we have only two classes of data which facilitates the application of classification prediction techniques. The second column of Table 5 shows the error of different prediction techniques in classifying the CM1 data as faulty or faultless. Comparing Tables 4 and 5, we notice that the error prediction in "Defect (faulty/faultless)" case is better than predicting the "Error Count" of the data sets. This validates our earlier proposal to analyze the modules as faulty/faultless rather than analyzing their absolute error count.

Table 5. Error shown by different models on predicting "Defect" on CM1 data set

(LM)	MAE (Original)	MAE (PCA)	MAE (CFS)	MAE (CBS)
SVLR	0.159	0.5	0.1671	0.5
NND	0.184	0.1833	0.188	0.1683
LoR	0.1569	0.1615	0.1626	0.1626
NB	0.1311	0.133	0.1316	0.1409
IBL	0.1567	0.158	0.152	0.1509
JDT	0.1752	0.1737	0.1737	0.1737
1R	0.1184	0.1184	0.1184	0.1053

Table 6. Error shown by different Models on the predicting "Defect" of the CM1, JM1, KC1, and PC1 data sets.

Learning Method (LM)	CM1 MAE	JM1 MAE	KC1 MAE	PC1 MAE
SVLR	0.159	0.2844	0.2088	0.111
NND	0.184	0.2648	0.1944	0.1123
LoR	0.1569	0.2834	0.2031	0.1059
NB	0.1311	0.191	0.1638	0.092
IBL	0.1567	0.269	0.1913	0.089
JDT	0.1752	0.275	0.2066	0.0895
1R	0.1184	0.2013	0.169	0.0631

The next step is to compare the consistency of the predictor models across different data sets of varying sizes and distribution. Of course, we want to compare the models with the data sets that

- belong to the same application domain meaning that they solve similar problems,
- they are developed with similar methods and implementation environments, and
- they are developed by teams with similar technical background and procedures.

Our data sets satisfy the above properties and, hence, are amenable to comparison. Table 6 shows the errors predicted by different predictor techniques on different data sets. We now proceed to the analysis of all these results.

6. Analysis of Results

We consider the results from the simulations conducted. Each subsection deals with a specific research issue associated with analysis of software defects.

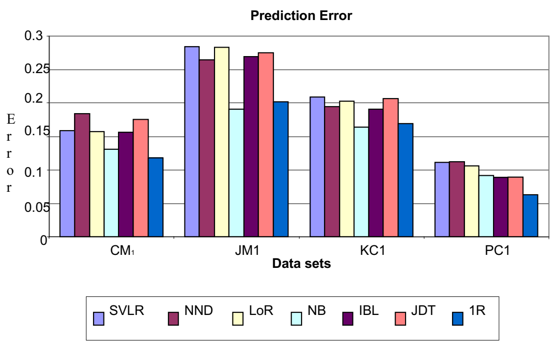
6.1. Predicting the Number of Defects in a Module versus Predicting it as Faulty

From Table 4, we see that the error predicted is less when we take the goal attribute to be a discrete distribution rather than a continuous one. In our data sets, the number of distinct data values taken by the goal attribute is very limited. So, we used discrete distribution for the goal attribute. If the data values range were very large, then we could have used continuous distribution for it. The distribution of all the predictor attributes were taken as continuous, which is obvious by the range of values they take. Thus, we find that prediction of the actual number of software defects in a module is much more difficult than predicting it as fault-prone.

6.2. Consistency of performance of Different Machine Learning Techniques

Figure 1, derived from Table 6, show the predicted error vs. prediction methodology for different techniques. However we notice that there is no consistency in the rank of learning techniques in terms of prediction accuracy across different data sets. This suggests that the best choice of a learning technique really depends on the data at hand at that particular moment.

Figure 1. Error Predicted on random JM1 data set by different Prediction techniques



6.3. Possible best choice of machine learning technique among different ML techniques

When we look at Figure 1, we see that Naïve Bayes performed better for JM1 and KC1 data and 1R performed better for CM1 and PC1 data sets. But 1R in general produced very good prediction, the reason being the continuous nature of the attributes and the smaller percentage of defects present. This makes 1R to form easy rules and, hence, yield better results. This suggests the use of the 1R rule as one of the foremost techniques for software defect data analysis. The following are the results obtained on four datasets for the other learning techniques:

- Naïve Bayes performed the best for 3 datasets and was third in the remaining data sets.
- IBL performed the best for one data set, while being second in two other data sets and third in the remaining data sets.
- Neural networks was second and third in two data sets, while it did not finish in the top three for the other two datasets.

Based on this analysis, we see that Naïve Bayes, IBL, and neural networks are the better prediction models compared to the other methods. However, it is certainly true that the Bayes classifier is highly affected by the percentage of the defect modules. The analysis in Table 7 shows the sensitivity of the Bayes classifier.

Also, the Bayes classifier assumes independence of attributes, which is not true as shown by the PCA analysis which is omitted here due to space constraints. Hence, we conclude that IBL and 1R are relatively best predictor techniques for software prediction. Memory-based reasoning is one of the IBL techniques used in our past research and this analysis provides a strong point in using MBR.

Table 7. Sensitivity of the Bayes Classifier.

Learning Method (LM)	CM1 (MAE / 9.5% DM)	CM1 (MAE / 13% DM)	Error Difference Percentage
SVLR	0.159	0.1892	19 %
NND	0.184	0.1808	1.7 %
LoR	0.1569	0.1843	17.46 %
NB	0.1311	0.1987	**51.56 %**
IBL	0.1567	0.1788	14.1 %
JDT	0.1752	0.1929	10.1 %
1R	0.1184	0.1349	13.93 %

6.4. How difficult is Software-Defect Prediction?

We have shown that software defect prediction models are affected by various factors, such as multicollinearity, among the attributes and the percentage of defects present in the data sets. While multicollinearity is eliminated by using PCA, it did not result in a decrease in error prediction for the learning models in most of the cases. There was no direct relationship between the skewness and the accuracy of prediction suggesting that machine learning techniques tend to be insensitive to skewness present in the data. Also, CFS and CBS methods performed better in most of the instances than the original one. This is particularly true for IBL prediction methodology. Hence, we can say that IBL is definitely one of the methods that need to be considered. Reduction of features by CBS and CFS definitely makes the prediction problem simpler and eliminates the use of cumbersome PCA analysis. Also, random reduction of attributes did not achieve any improved accuracy, suggesting that the attribute reduction process needs to be carried in a definite manner as in the case of CBS and CFS. We suggested that 1R and Instance based learning methods are the two forerunners among the choices of prediction models. But as a word of caution, we also notice that there is no particular trend in the accuracy prediction of different methodologies for different data sets. Hence, choosing the best result from a set of prediction models could be a better strategy.

One of the methods of choosing the best predictor methodology is to conduct the analysis on a set of different predictor models and choose the best one. While this is an efficient method, the time spent on evaluating these techniques might be a problem, especially for on-line decision making prediction techniques. However, the task of choosing the best predictor can be relegated to off-peak hours and, thus, make it feasible to evaluate many prediction techniques.

Although we have proposed a plausible solution to software defect prediction problem, more software defect data needs to be analyzed before asserting that software quality prediction is a feasible task.

6.5. Improving the Software Defect Prediction Process

We have seen above that lots of factors affect software defect prediction, resulting in inconsistencies among learning methods. Hence, there is a need to develop methods that could remove some of the randomness (complexity/uncertainties) in the data, leading us to a more definitive explanation of the error analysis. One of the steps in this direction is to capture the dependency among attributes using probabilistic models, rather than just using the "size" and "complexity" metrics. Bayesian belief networks (BBN) is one of the approaches along this direction.

7. Conclusion

We have compared different machine learning models for identifying faulty real-time software modules. We have shown that there is no particular learning technique that performs the best for all the data sets. However, IBL and 1R are the better methods that showed relatively better consistency in prediction accuracy compared with others.

We have tried to combine machine learning techniques with statistical techniques, such as PCA, but it did not add any additional advantages. In contrast, other simple FSS methods performed better than PCA. So we caution that, in case of software defects, using PCA might not be a good idea.

We also showed that "size" and "complexity" metrics are not sufficient attributes for accurate prediction. We need to include dependencies between these metrics in our analysis to improve the prediction models. In this respect, we believe that Bayesian belief networks method is one of the techniques that need to be explored to capture these causal relationships.

Of course, any conclusion needs to have solid scientific explanations. We need compelling and sophisticated theories that can explain the empirical observations. In the future, we aim to develop a more analytical approach to better explain the analysis results and investigate on BBN for software defect analysis. We conclude by saying that software defect analysis is not an easy task and more formal investigation is needed to relate software defect prediction with the inherent characteristics of the data set.

8. Acknowledgment

This research was supported in part by the National Science Foundation under Grant No. EIA-0103709.

9. References

[1] Fenton, N.E. and Neil, M, "A critique of software defect prediction models", Software Engineering, IEEE Transactions on, Volume: 25 Issue: 5, Sept.- Oct. 1999, Page(s): 675 -689.

[2] Shepperd, M. and Kadoda, G., "Comparing software prediction techniques using simulation", Software Engineering, IEEE Transactions on, Volume: 27 Issue: 11, Nov. 2001, Page(s): 1014 -1022.

[3] Pickard, L., Kitchenham, B. and Linkman, S., "An Investigation of Analysis Techniques for Software Datasets", Software Metrics Symposium, 1999. Proceedings. Sixth International, 4-6 Nov. 1999, Page(s): 130 -142.

[4] Khoshgoftaar, T.M., Gao, K.and Szabo, R.M., "An Application of Zero-Inflated Poisson Regression for Software Fault Prediction", Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on, 27-30 Nov. 2001, Page(s): 66 -73.

[5] Munson, J. and Khoshgoftaar, T., "Regression Modeling of Software Quality: An Empirical Investigation", Information and Software Technology, Volume: 32 Issue: 2, 1990, Page(s): 106 - 114.

[6] Khoshgoftaar, T.M. and Munson, J.C., "Predicting Software Development Errors using Complexity Metrics", Selected Areas in Communications, IEEE Journal on, Volume: 8 Issue: 2, Feb. 1990, Page(s): 253 -261.

[7] Wang, Y, "A New Approach for fitting Linear Models in high-dimensional Spaces", PhD Thesis (2000), Department of Computer Science, University of Waikato, New Zealand, www.cs.waikato.ac.nz/~ml/publications/2000/thesis.pdf

[8] Succi, G, Stefanovic, M. and Pedrycz, W., "Advanced Statistical Models for Software Data", Department of

Electrical and Computer Engineering, University of Alberta, Canada, http://www.unibz.it/web4archiv/objects/pdf/cs_library/2/AdvancedStatisticalModelsforSoftwaredata.pdf.

[9] Schneidewind, N.F., "Investigation of logistic regression as a discriminant of software quality", Software Metrics Symposium, METRICS 2001. Proceedings. Seventh International, 4-6 April 2001, Page(s): 328 -337.

[10] Mair, C., Kadoda, G., Leflel, M., Phapl, L., Schofield, K., Shepperd, M. and Webster, S., An Investigation of Machine Learning Based Prediction Systems, Systems Software, Journal of, Volume: 53 Issue: 1 , Nov. 2000, Page(s): 23 – 29.

[11] Aljahdali, S.H., Sheta, A., and Rine, D., "Prediction of software reliability: a comparison between regression and neural network non-parametric models", Computer Systems and Applications, ACS/IEEE International Conference on. 2001, 25-29 June 2001, Page(s): 470 -473.

[12] Menzies, T., Ammar, K., Nikora, A., and Stefano, S., "How Simple is Software Defect Prediction?", Submitted to Journal of Empirical Software Engineering, October 2003, http://menzies.us/pdf/03simpled.pdf.

[13] Eman, K., Benlarbi, S., Goel, N., and Rai, S., "Comparing case-based reasoning classifiers for predicting high risk software components", Systems Software, Journal of, Volume: 55 Issue: 3, Nov. 2001, Page(s): 301 – 310.

[14] Morasca, S., and Ruhe, G., "A Comparative study of two Techniques for Analyzing Software Measurement Data", Proceedings of Annual Meeting, ISERN, 1996, Sydney, Australia.

[15] Reformat, M., Pedrycz, W., and Pizzi, N.J., "Software quality analysis with the use of computational intelligence", Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on, Volume: 2, 12-17 May 2002
Page(s): 1156 -1161.

[16] Holte, R., "Very Simple Classification Rules Perform Well on Most Commonly used Datasets", Machine Learning, vol. 11, 1993, Page(s): 69 – 91.

[17] Neumann, D.E., "An enhanced neural network technique for software risk analysis", Software Engineering, IEEE Transactions on, Volume: 28 Issue: 9, Sept. 2002, Page(s): 904 -912.

[18] Hall, M.., "Correlation-based feature selection for machine learning", PhD Thesis, Department of Computer Science, University of Waikato, New Zealand.

[19] Liu, H., and Setino, R., "A probabilistic approach to feature selection: A filter solution", Machine Learning, Thirteenth International Conference on, 1996, Morgan Kaufmann, Page(s): 319 – 327.

[20] Dumais, S., Platt, J., Heckernam, D., and Sahami, M., "Inductive Learning Algorithms and representations for text categorization", Information and Knowledge Management, International Conference on, 1998, Page(s): 148 -155

[21] Kira, K., and Rendell, L., "A Practical Approach to Feature Selection", Machine Learning, Ninth International Conference on, 1992, Morgan Kaufmann, Page(s): 249 -256.

[22] Fenton, N., Neil, M., and Krause, P., "Software Measurement: Uncertainty and Causal Modeling", IEEE Software Magazine, Volume: 19 Issue: 4, July/Aug. 2002, Page(s): 116 -122.

[23] Zhang, D., "Applying Machine Learning Algorithms in Software Development", Modeling Software System Structures in a fastly moving Scenario, 2000 Monterey Workshop on, June 13 – 16, 2000, Santa Margerita Ligure, Italy.

[24] Mitchell, T., *Machine Learning*, ISBN 0070428077, McGraw-Hill Book Co – Singapore, 1997.