A Multivariate Analysis of Static Code Attributes for Defect Prediction

Burak Turhan, Ayşe Bener
Department of Computer Engineering, Bogazici University
34342, Bebek, Istanbul, Turkey
{turhanb, bener}@boun.edu.tr

Abstract

Defect prediction is important in order to reduce test times by allocating valuable test resources effectively. In this work, we propose a model using multivariate approaches in conjunction with Bayesian methods for defect predictions. The motivation behind using a multivariate approach is to overcome the independence assumption of univariate approaches about software attributes. Using Bayesian methods gives practitioners an idea about the defectiveness of software modules in a probabilistic framework rather than the hard classification methods such as decision trees. Furthermore the software attributes used in this work are chosen among the static code attributes that can easily be extracted from source code, which prevents human errors or subjectivity. These attributes are preprocessed with feature selection techniques to select the most relevant attributes for prediction. Finally we compared our proposed model with the best results reported so far on public datasets and we conclude that using multivariate approaches can perform better.

Keywords: Defect prediction, Software Metrics, Naïve Bayes.

Topics: Software Quality, Methods and Tools.

1. Introduction

Testing is the most costly and time consuming part of software development lifecycle, regardless of the development process used. Therefore effective testing leads to significant decrease in project costs and schedules. The aim of defect prediction is to give an idea about the testing priorities, so that exhaustive testing is prevented. Using an automated model may help project managers to allocate testing resources effectively. These models can predict the degree of defectiveness if relevant features of software are supplied to them. These relevant features are achieved by using software metrics.

Researchers usually prefer focusing on the selection of a subset of available features [10]. Feature subset selection is mainly preferred because of its interpretability, since the selected features correspond to actual and in some cases controllable measurements from software. This gives the ability to generate rules about the desired values of metrics for 'good' software. It is easier to explain such rules to programmers and managers [6].

This is also the answer to why most of the studies use decision trees as predictors. Decision trees can be interpreted as a set of rules and they can be understood by less technically involved people [6]. But decision trees are hard classification methods that can predict a module as either defective or non-defective. Alternatively, Bayesian approaches provide a probabilistic framework and yield soft classification methods with posterior probabilities attached to the predictions [1]. This is why we employed Bayesian approaches in this work.

On the other hand, feature subset selection requires an exhaustive search for choosing the optimal subset. Thus, feature selection algorithms use greedy approaches like backward or forward selection [7]. In forward selection, one starts with an empty set of features, and a feature is selected only if it increases the performance of the predictor, otherwise it is discarded. Backward selection is similar in the sense that one starts with all features, and a feature is removed if it does not affect the performance of the predictor. These approaches evaluate the features one at a time and they do not consider the effects of features if taken as pairs, triples and n-tuples. While a single feature may not affect the estimation performance significantly, pairs, triples or n-tuples of features may [7]. In order to overcome this problem, this study employs feature extraction techniques and compares the results with a baseline study, where InfoGain algorithm is used to rank and select a subset of features [10].

Major contribution of this research is to incorporate multivariate approaches rather than univariate ones. Univariate approaches assume the independence of features whereas multivariate approaches take the relations between features into consideration. Obviously univariate models are simpler than multivariate models. While it is good practice to start modeling with simple models, the problem at hand should also be investigated by using more complex models. Then it should be validated by measuring performance whether using more complex models is worth the extra complexity introduced in the modeling. This research performs experiments with both simple and complex models and compares their performances.

In the following section, feature extraction methods used in this research are briefly described. Then, models used for defect prediction are explained. After describing the experimental design and the results, conclusions will be given.

2. Feature Extraction Methods

In feature extraction, new features are formed by combining the existing ones. These new set of features may not be interpreted easily as before [6]. On the contrary, there are cases where they turn out to be interpretable [5]. The new features may also lead to better prediction performances by removing irrelevant and non-informative features. An advantage of feature extraction methods used in this study is that they project data to an orthogonal feature space. One has to decide between ease of interpretability and better prediction performances in such cases. In this research authors prefer better performance and therefore they explore feature extraction methodologies.

Principal Component Analysis (PCA) has been used in other defect prediction studies [11], [13], [8], [14],[2]. We also use PCA in this research. PCA reveals the optimum linear structure of data points. But it is unable to find nonlinear relations, if there exists such relations in data. In order to investigate non-linear relations, we use Isomap algorithm as another feature extraction technique.

2.1. Isomap

Isomap inherits the advantages of PCA and extends them to learn nonlinear structures that are hidden in high dimensional data. Computational efficiency, global optimality, and guarantee of asymptotic convergence are its major features [16].

In general, Euclidean distance is used to calculate the similarity of two instances. However, the use of the Euclidean distance to represent pair wise distances makes the model unable to preserve the intrinsic geometry of the data. Two nearby points, in terms of Euclidean distance, may indeed be distant, because their actual distance is the path between these points along the manifold. The length of the path along the manifold is referred to as the *geodesic distance* [16]. A 2-D spiral is an example of a manifold, which is actually a 1-D line that is folded and embedded in 2-D (See Figure 1, adapted from [9]). Applying Isomap on the spiral unfolds it to its true structure. Isomap simply performs classical Multidimensional Scaling [4] on pair wise geodesic-distance matrix.

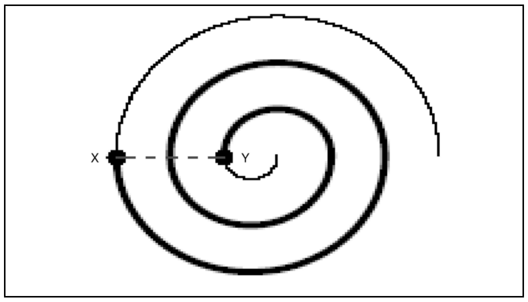


Figure 1. Geodesic distance metric: Points X and Y are at distinct ends of the spiral. Using Euclidean distance, the true structure of spiral, i.e. 1-D line folded and embedded in 2-D, can not be revealed.

Geodesic distance represents similar (or different) data points more accurately than the Euclidean distance, but the question is how to estimate it? Here the local linearity principle is used and it is assumed that neighboring points lie on a linear patch of the manifold, so for nearby points the Euclidean distances correctly estimate the geodesic distances. For distant points, the geodesic distances are estimated by adding up neighboring distances over the manifold using a shortest-path algorithm.

Isomap finds the true dimensionality of nonlinear structures. The interpretation of projection axes can be meaningful in some cases [5]. Isomap uses a single parameter to define the neighborhood for data points i.e. for k-nearest neighbors of a data point, pair wise geodesic distances are assumed to be equivalent to Euclidean distances. This parameter should be fine tuned, preferably by cross-validation, to obtain optimum results. Data sample is transformed to have a linear structure in the new projection space; e.g. the spiral is unfolded to a line.

3. Predictor Models

This section explains predictor models used for defect prediction. As a baseline, the Naive Bayes classifier is taken since it is shown to acquire best results obtained so far [10]. We remove the assumptions of the Naive Bayes classifier one at a time and construct the linear and quadratic discriminants. The assumption in Naive Bayes is that the features of data sample are independent, thus it employs the univariate normal distribution. We believe this assumption is not valid for software data and since there are correlations between software data features. So we use a multivariate normal distribution to model the correlations among features. In the next section univariate and multivariate normal distributions are briefly explained.

3.1. Univariate vs. Multivariate Normal Distribution

In univariate normal distribution, $x \sim N(\mu, \sigma^2)$, x is said to be normal distributed with mean μ and standard deviation σ and the probability distribution function (pdf) is defined as:

$$p(x) = \frac{1}{\sqrt{(2\pi)}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right) \quad \textbf{(1)}$$

The term inside the exponential term in Equation 1 is the normalized Euclidean distance, where the distance of a data sample *x* to the sample mean μ is measured in terms of standard deviations σ. This ensures to scale the distances of different features in case feature values vary significantly. This measure does not consider the correlations among features.

In the multivariate case, *x* is a d-dimensional vector that is normal distributed, $x \sim N(\vec{\mu}, \Sigma)$, and the pdf of a multivariate normal distribution is defined as:

$$p(\vec{x}) = \frac{1}{\sqrt{(2\Pi)^{d/2} \Sigma^{1/2}}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu}) \right) \quad \textbf{(2)}$$

Where Σ is the covariance matrix and μ is the mean vector. The term inside the exponential term in Equation 2 is another distance function and called the M*ahalanobis distance* [1]. In this case, the distance to the mean vector is normalized by the covariance matrix and the correlations of features are also considered. This results in less contribution of highly correlated features and features with high variance.

Our assumption is that software data features are correlated and a multivariate model would be more appropriate than the univariate model. Besides, multivariate normal distribution is analytically simple, tractable and robust to departures from normality [1]. As *no free lunch* theorem states [17], nothing comes for free and using a multivariate model increases the number of parameters to estimate. In the univariate case, only 2 parameters, μ and Σ are estimated, while in the multivariate case, d parameters for μ and d.d parameters for Σ need to be estimated.

3.2. Multivariate Classification

In software defect prediction, one aims to *discriminate* classes C_0 and C_1 where samples in C_0 are non defective and samples in C_1 are defective. We combine the multivariate normal distribution and the Bayes rule, use different assumptions, and achieve different discriminants with different complexity levels (See Table 1). We prefer discriminant point of view, since it is geometrically interpretable. A discriminant in general is a hyper plane that separates d-dimensional space into 2 disjoint subspaces. General structure of a discriminant is explained next.

Table 1. Complexities of predictors in a K-class problem with d features.

Predictor	# Parameters
QD	(K x (d x d)) + (K x d) + (K)
LD	(d x d) + (K x d) + (K)
NB	(d) + (K x d) + (K)

Bayes theorem states that the posterior distribution of a sample is proportional to the prior distribution and the likelihood of the given sample. More formally:

$$P(C_i \mid x) = \frac{P(x \mid C_i) P(C_i)}{P(x)} \quad \textbf{(3)}$$

Equation 3 is read as:

"The probability of a given data instance x to belong to class C_i is equal to the multiplication of the likelihood that x is coming from the distribution that generates C_i and the probability of observing C_i 's in the whole sample, normalized by the evidence.

Evidence is given by:

$$P(x) = \sum_i P(x \mid C_i) P(C_i) \quad \textbf{(4)}$$

and it is a normalization constant for all classes, thus it can be safely discarded. Then Equation 3 becomes:

$$P(C_i \mid x) = P(x \mid C_i) P(C_i) \quad \textbf{(5)}$$

In a classification problem we compute the posterior probabilities $P(C_i|x)$ for each class and choose the one with the highest posterior. This is equivalent to defining a discriminant function $g_i(x)$ for class C_i and $g_i(x)$ is derived from Equation 6 by taking the logarithms for convenience.

$$g_i(x) = \log(P(x \mid C_i)) + \log(P(C_i)) \quad \textbf{(6)}$$

In order to achieve a discriminant value, one needs to compute the prior and likelihood terms. Prior probability $P(C_i)$ can be estimated from the sample by counting. The critical issue is to choose a suitable distribution for the likelihood term P(x|C$_i$). This is where the multivariate normal distribution takes place. In this study likelihood term is modeled by the multivariate normal distribution.

Computing discriminant values for each class and assigning the instance to the class with the highest value is equivalent to using Bayes Theorem for choosing the class with the highest posterior probability. For the 2-class case, it is sufficient to construct a single discriminant by g(x) = g$_0$(x) – g$_1$(x). Using discriminant point of view, we will explain different predictors in the following section. In all cases, an instance x is classified as C$_i$ such that $i = \arg\max_k (g_k(\vec{x}))$

3.3. Quadratic Discriminant

Assumption: Each class has distinct Σ_i and μ_i.
Derivation: Combining Equation 2 and Equation 6

$$g_i(\vec{x}) = \frac{1}{2}\log(|S_i|) - \frac{1}{2}(\vec{x}-\vec{m})^T S^{-1}(\vec{x}-\vec{m}) + \log(\hat{P}(C_i)) \quad \textbf{(7)}$$

and by defining new variables W$_i$, w$_i$ and w$_{i0}$, the quadratic discriminant is obtained as

$$g_i(\vec{x}) = \vec{x}^T W_i \vec{x} + w_i^T \vec{x} + w_{i0} \quad \textbf{(8)}$$

where

$$W_i = -\frac{1}{2} S_i^{-1} \quad \textbf{(9)}$$

$$w_i = S_i^{-1}\vec{m}_i \quad \textbf{(10)}$$

$$w_{i0} = -\frac{1}{2}\vec{m}_i^T S_i^{-1}\vec{m}_i - \frac{1}{2}\log(|S_i|) + \log(\hat{P}(C_i)) \quad \textbf{(11)}$$

and S$_i$, m$_i$ and *P*(C$_i$) are maximum likelihood estimates of Σ_i, μ_i and P(C$_i$) respectively.

Quadratic model considers the correlation of the features differently for each class. In case of K-classes, the number of parameters to estimate is K.(d.d) for covariance estimates and (K.d) for mean estimates. Also K prior probability estimations are needed.

3.4. Linear Discriminant

Assumption: Each class has a common Σ and distinct μ_i
Derivation: Assumption states that classes share a common covariance matrix. The estimator is found by either using the whole data sample or by the weighted average of class covariances which is given as

$$S = \sum_i \hat{P}(C_i) S_i \quad \textbf{(12)}$$

Placing this term in Equation 7 we get

$$g_i(\vec{x}) = -\frac{1}{2}(-2\vec{x}^T S^{-1}\vec{m}_i + \vec{m}_i^T S^{-1}\vec{m}_i) + \log(\hat{P}(C_i)) \quad \textbf{(13)}$$

which is now a linear discriminant in the form of

$$g_i(\vec{x}) = \vec{w}_i^T \vec{x} + w_{i0} \quad \textbf{(14)}$$

where

$$w_i = S_i^{-1}\vec{m}_i \quad \textbf{(15)}$$

$$w_{i0} = -\frac{1}{2}\vec{m}_i^T S_i^{-1}\vec{m}_i + \log(\hat{P}(C_i)) \quad \textbf{(16)}$$

This model considers the correlation of the features but assumes the variances and correlation of features are the same for both classes. The number of parameters to estimate for covariance matrix is now independent of K. For covariance estimates (d.d), for mean estimates (K.d) and for priors K parameters should be estimated.

3.5. Naïve Bayes

Assumption: Each class has a common Σ with off diagonal entries equal to 0, and distinct μ_i

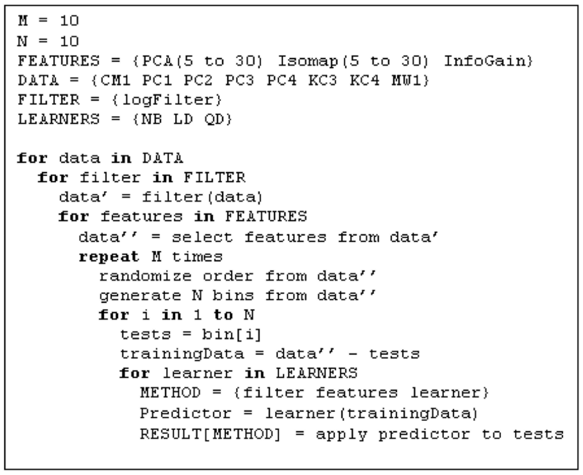
Derivation: Assumption states the independence of features by using a diagonal covariance matrix. Then the model reduces to a univariate model given in Equation 17.

$$g_i(x) = -\frac{1}{2}\sum_{j=1}^{d}\left(\frac{x_j^t - m_{ij}}{s_j}\right)^2 + \log(P(C_i)) \quad \textbf{(17)}$$

This model does not take the correlation of the features into account and it measures the deviation from the mean in terms of standard deviations. For Naive Bayes, (d) covariance, (K.d) mean and K prior parameters should be estimated.

4. Experiments and Results

Design of experiments and evaluation of results in software defect prediction problems have particular importance. Most of the experiment designs have important flaws such as self tests and insufficient performance measures as reported in [10]. Most research reported only the accuracy of predictors as a performance indicator. Examining defect prediction datasets, it is easily seen that they are not balanced. In other words, the number of defective instances is much less than the number of nondefective instances. As pointed out in [10], one can achieve 95% accuracy on a 5% defective dataset by building a dummy classifier that always classifies instances as nondefective. A framework of MxN experiment design, which means M replications of N holdout (cross validation) experiments, is also given in [10] and additional performance measures are reported, such as probability of detection (pd) and probability of false alarm (pf). This research follows the same notation.



```
M = 10
N = 10
FEATURES = {PCA(5 to 30) Isomap(5 to 30) InfoGain}
DATA = {CM1 PC1 PC2 PC3 PC4 KC3 KC4 MW1}
FILTER = {logFilter}
LEARNERS = {NB LD QD}

for data in DATA
  for filter in FILTER
    data' = filter(data)
    for features in FEATURES
      data'' = select features from data'
      repeat M times
        randomize order from data''
        generate N bins from data''
        for i in 1 to N
          tests = bin[i]
          trainingData = data'' - tests
          for learner in LEARNERS
            METHOD = {filter features learner}
            Predictor = learner(trainingData)
            RESULT[METHOD] = apply predictor to tests
```

Figure 2. Experiment Design.

The experiments conducted in [10] are replicated and extended in this study. Framework for experiment design in [10] is followed and updated as in Figure 2. In order to extract features, PCA and Isomap are performed on the log filtered data attributes. An advantage of log filtering is that it scales the features so that extreme values are handled. Another advantage of log filtering is that normal distribution better fits to data. In other words, data attributes are assumed to be lognormal distributed. 5 to 30 features are extracted for all datasets using PCA and Isomap. Best subset of features reported in [10] is also used in the experiments. This subset of features differs in each dataset. The best performing dimensionalities achieved by PCA and Isomap are also different for each dataset. These observations support the idea that there is no global set of features that describe the software. So, maximum possible metrics of software should be collected and analyzed as long as it is feasible to collect them.

10-fold cross-validation approach is used in the experiments. That is, datasets are divided into 10 bins, 9 bins are used for training and 1 bin is used for testing. Repeating these 10 folds ensures that each bin is used for training and testing while minimizing sampling bias. Each holdout experiment is also repeated 10 times and in each repetition the datasets are randomized to overcome any ordering effect and to achieve reliable statistics. Reported results are the mean values of these 100 experiments for each dataset. Quadratic discriminant (QD), linear discriminant (LD) and Naive Bayes (NB) are the predictors used in this research. As performance measures *pd*, *pf* and balance (*bal*) are reported. *pd* is a measure for correctly detecting defective modules and it is the ratio of the number of defective predicted modules to the number of actual defective modules. Obviously higher *pd*'s are desired. As the name suggests, *pf* is a measure for false alarms and it is interpreted as the probability of predicting a module as defective while it is not indeed. *pf* is desired to have low values. Balance measure is used to choose the optimal (*pd*, *pf*) pairs such that area under the ROC curve is maximized and it is defined as the normalized Euclidean distance from the desired point (0,1) to (*pd*, *pf*) in a ROC curve.

Table 2. Dataset Descriptions

Name	#Modules	DefectRate (%)
CM1	505	9
PC1	1107	6
PC2	5589	0.6
PC3	1563	10
PC4	1458	12
KC3	458	9
KC4	125	49
MW1	403	7

For evaluation, 8 different public datasets obtained from NASA MDP repository [12] are used. Sample sizes vary from 125 to 5589 modules. Each dataset has 38 features representing static code attributes. As seen in Table 2 defect rates are too low which consolidates the use of above mentioned performance measures. All implementations are done in MATLAB environment using standard toolboxes.

Results are tabulated in Table 3. Mean results of (*pd*, *pf*) pairs selected by the *bal* measure after 10x10 holdout experiments are given. For PCA and ISO labeled entries, these results are selected from 5 to 30 features obtained by PCA and Isomap respectively. For SUB labeled entries, the best subset of features

Table 3. Results

		Performances		
Data	Predictor	pd(%)	pf(%)	bal(%)
CM1	SUB+NB	**84**	32	74
PC1	PCA+NB	**68**	25	71
PC2	PCA+NB	72	13	78
PC3	PCA+LD	**76**	31	72
PC4	PCA+QD	88	20	83
KC3	ISO+NB	**81**	25	77
KC4	ISO+LD	78	27	75
MW1	ISO+LD	**73**	34	69
Average:		***77***	***25***	***76***

obtained by InfoGain are used as reported in [10]. In Table 3, results indicated in bold face are statistically significant than other methods with α = 0.05 after applying a t-test, considering *pd* performance measure.

Subset selection is better than feature extraction methods in only 1 out of 8 datasets (CM1). In the remaining datasets, best performances are obtained either by applying PCA or Isomap instead of InfoGain. In PC1, PC2, PC3 and PC4, best mean performances are achieved applying PCA, and in KC3, KC4 and MW1 Isomap yielded better results. It is observed that Isomap gives the best performances on relatively small datasets. As the module sizes increase PCA performs better.

Except PC3 dataset, our replicated results are similar to reported mean results in [10]. But variances of replicated experiments (i.e. subsetting) are larger than PCA and Isomap approach especially for *pf* measure. NB and LD are observed to behave similarly whereas QD results are different than NB and LD in terms of performance. It is observed for QD, that as the number of features increase, performances get worse especially for *pf* measure and the variances increase. Possible reason for this is the complexity of the model (i.e. too many parameters to estimate).

As for the predictors, Naive Bayes (NB) is chosen 4 times, linear discriminant (LD) is chosen 3 times and quadratic discriminant (QD) is chosen only once. From these results, it can be concluded that claims stating any of these predictors as the 'globally' correct one, should be avoided. As expected, no specific configuration of a feature selection and a predictor is always better than the others. Even though NB is the majority winner, it is clearly seen that performances on some datasets are increased by using multivariate methods: QD and LD. Applying QD gives the best result in PC4 dataset, but it is not statistically significant. It can be concluded that QD can be discarded because of its complexity. In cases where LD wins, statistical significances are observed, so the additional complexity introduced can be justified. There may be other predictors performing better than these. Constructing better predictors is an open ended problem and as better results are reported, the problem gets more difficult due to *ceiling effect* i.e. it is harder to confirm the hypothesis that predictor A performs better than predictor B, when A and B perform maximum achievable performance or close to it [3].

Overall performance of the approach improves on the best results reported so far [10]. Previous research reported mean (*pd*, *pf*) = (71,25) which yields *bal* = 72 averaged over all datasets. Replication of these experiments yield mean (*pd*, *pf*) = (64, 19) and *bal* = 71. After experimenting with all possible combinations of InfoGain, PCA, Isomap with NB, LD and QD, an improvement is observed by picking the best combinations for all datasets. Improved results yield mean (*pd*, *pf*) = (77, 25) where *bal* = 76. While no change in pf measure is observed, *pd* measure is improved by 6%.

A final comment should be made about the running times of algorithms. As expected, QD takes more time than LD and NB. However this difference is not too significant. The dominant factor that affects the running times are the sample sizes.

5. Conclusions and Future Work

In this research software defect prediction is considered as a data mining problem. Several experiments are conducted, including the replication of previous research on publicly available datasets from NASA repository. Performances of different predictors together with different feature extraction methods are evaluated. Results are compared with the best performances reported so far and some improvements are observed.

The previous research advices that one should not seek for globally best subset of features, rather to focus on building predictors that combines information from multiple features. In addition, authors also believe that research should focus on a balanced combination of those. In other words, building successful predictors depends on how useful information is supplied to them. While making research on better predictors, research on obtaining useful information from features should also be carried out. A contribution of this research is using linear and nonlinear feature extraction methods in order to combine information from multiple features. In software defect prediction there is more research on feature subset selection than feature extraction. Results suggest that it is worth to explore more to deepen our knowledge on feature extraction studies.

Another contribution of this research is the modeling of correlations among features. Improved results are obtained by using multivariate statistical methods. Furthermore, the probabilities of predictions are provided by employing Bayesian approaches, which can give project managers and practitioners a better understanding of the defectiveness of software modules.

Further research should investigate the validation of the log normal distribution assumption of software data used in this research. It is better practice to apply *goodness of fit* tests, rather than assuming a normal distribution. Other exponential family distributions should also be investigated. Another research area is to investigate filters to transform data into suitable distributions.

Acknowledgements

This research is supported in part by Bogazici University research fund under grant number BAP-06HA104. Authors would like to thank Koray Balcı, who has contributed to the earlier versions of this manuscript.

References

[1] E. Alpaydin, *Introduction to Machine Learning*, The MIT Press, October 2004.

[2] E. Ceylan, F. O. Kutlubay, and A. B. Bener, "Software defect identification using machine learning techniques", In *Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, IEEE Computer Society, Washington, DC, USA, 2006, pp. 240–247.

[3] P. R. Cohen. *Empirical Methods for Artificial Intlligence,* The MIT Press, London, England, 1995.

[4] T. Cox and M. Cox, *Multidimensional Scaling*. Chapman & Hall, London, 1994.

[5] V. de Silva and J. B. Tenenbaum, "Global versus local methods in nonlinear dimensionality reduction", In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems,* 15, MIT Press, Cambridge, MA, 2003, pp. 705–712.

[6] N. E. Fenton and M. Neil, "A critique of software defect prediction models", *IEEE Transactions. on Software. Engineering*., 25(5), 1999, pp. 675–689.

[7] Guyon and Elisseff, "An introduction to variable and feature selection", *Journal of Machine Learning Research*, 3, 2003, pp 1157–1182.

[8] T. M. Khoshgoftaar and J. C. Munson, "Predicting software development errors using software complexity metrics", *IEEE Journal on Selected Areas in Communications*, 8(2), Feb. 1990, pp. 253–261.

[9] J. A. Lee, A. Lendasse, N. Donckers, and M. Verleysen., "A robust nonlinear projection method", In *Proceedings of ESANN 2000, European Symposium on Artificial Neural Networks*, Bruges (Belgium), 2000, pp. 13– 20.

[10] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors", *IEEE Transactions on Software Engineering*, 33(1), 2007, pp. 2–13.

[11] J. Munson and Y. M. Khoshgoftaar, "Regression modelling of software quality: empirical investigation", *J. Electron. Mater.*, 19(6), 1990, pp. 106–114

[12] NASA/WVU IV&V Facility, Metrics Data Program, available from http://mdp.ivv.nasa.gov.

[13] M. Neil, "Multivariate assessment of software products", *Softw. Test., Verif. Reliab.*, 1(4), 1992, pp. 17–37.

[14] D. E. Neumann, "An enhanced neural network technique for software risk analysis", *IEEE Tranactions on. Software Engineering*, 28(9), 2002, pp. 904–912.

[15] G. Boetticher, T. Menzies and T. Ostrand, PROMISE Repository of empirical software engineering data http://promisedata.org/repository, West Virginia University, Department of Computer Science, 2007

[16] J. B. Tenenbaum, V. de Silva, and J. C. Langford, „A global geometric framework for nonlinear dimensionality reduction", *Science*, 290, 2000, pp. 2319– 2323

[17] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization" *IEEE Transactions on Evolutionary Computation*, 1(1), April 1997, pp. 67–82