Software Quality Prediction using Median-Adjusted Class Labels

Nicolino J. Pizzi[†] Arthur R. Summers[†] Witold Pedrycz[*]

[†]National Research Council Canada, Institute for Biodiagnostics
435 Ellice Avenue, Winnipeg, MB, R3B 1Y6, Canada (pizzi@nrc.ca)

[*]University of Alberta, Department of Electrical and Computer Engineering
Edmonton, AB, T6G 2G7, Canada (pedrycz@ee.ualberta.ca)

Abstract – Software metrics aid project managers in predicting the quality of software systems. A method is proposed using a neural network classifier with metric inputs and subjective quality assessments as class labels. The labels are adjusted using fuzzy measures of the distances from each class center computed using robust multivariate medians.

I. INTRODUCTION

Software quality has a direct impact on its validation and maintenance especially as the complexity of the underlying system increases. The manual process of quality assessment by project managers is slow, tedious, and prone to error and subjective misinterpretation. Any method that assesses software quality in an automated or semi-automated fashion would alleviate some of these concerns. One method is the translation of qualitative assessments into quantitative software measures as manifested in the methodology of quantitative software engineering [1][2].

A software metric is a mapping from a software object to a set of numerical values in order to quantify a software attribute [3]. These mappings may be used to assess the quality of targeted software attributes such as maintainability, extensibility, efficiency, and clarity. In essence, this is a problem of classification: can an object's quality ranking (class label) be predicted using only its corresponding metrics (input features)? While the metrics may be automatically collected, the quality class labels may be determined using expert software architects. Such a classifier could serve as a "quality filter" for objects. For example, a project manager could use such a filter to predict object quality in order to identify low quality objects for review and possible revision.

A problem with this approach is the inherent imprecision of the class labels due to the software architect's subjectivity in assessing object quality. This subjectivity may be compensated by incorporating into the quality class labels information gleaned from the software metrics themselves. *Median-adjusted class labels* is a neural network classification preprocessing strategy that compensates for the possible imprecision of class labels. Using training vectors (software metrics), robust measures of location using multivariate medians are computed for each class center. Based on distances from these centers, fuzzy sets are constructed that determine the degree to which each input vector belongs to each class. These membership values are then used to adjust class labels for the training vectors. In this context, one may view this method as a preprocessor used as a component of an entire learning environment developed for neural networks. This strategy is evaluated using (i) a multilayer perceptron, (ii) three multivariate medians, and (iii) a test set of metrics with non-adjusted quality class labels. The efficacy of this strategy is examined using objects from EvIdent®, a research-oriented biomedical data analysis system. Such systems must deal with voluminous data presented within straightforward graphical user interfaces.

II. NEURAL NETWORKS AS CLASSIFIERS

Given k classes, $\omega_1,\omega_1,\ldots,\omega_k$, the classification process is often an attempt to estimate an unknown function

$$f : \Re^n \rightarrow \{1, 2, \ldots, k\} \qquad (1)$$

from observed pair-wise samples, $(\boldsymbol{x}_1,y_1)$, $(\boldsymbol{x}_2,y_2)$,…, $(\boldsymbol{x}_N,y_N)$, where $\boldsymbol{x}_i \in \Re^n$ is an n-dimensional input vector and $y_i \in \{1,2,\ldots,k\}$ is a class label. The association between the classes and the labels is established by an external gold standard (e.g., a software architect). Each ω_i comprises a non-empty disjoint subset of N_i input vectors.

A supervised artificial neural network, attempts to approximate f using a subset (training set) of the input/output pairs, such that, for any given input vector, its corresponding class label is generated within some error tolerance. After the approximating function is found, its performance is analyzed using the remaining input/output pairs (test set). An input vector, $\boldsymbol{x}$, is correctly classified if the generated class label is the same as the desired class label.

The multilayer perceptron (MLP) requires the desired output for each input vector in order that it may be compared to the actual output generated [4]. Based on these comparisons, a learning strategy, such as the Quasi-Newton algorithm [5] (or the traditional back-propagation algorithm [6]), minimizes an error criterion in order to make incremental changes to the network. MLP has consistently demonstrated its effectiveness as a reliable nonlinear classifier and is sufficiently general for mappings of type (1).

The original labels are usually transformed (though not required) using 1-of-k encoding: $\boldsymbol{y}_i=[y_1,y_1,\ldots,y_k]$ where $y_j=1$, if the corresponding input vector, $\boldsymbol{x}_i$, belongs to ω_j, otherwise $y_j=0$. MLP successfully classifies an input vector, $\boldsymbol{x}_i$, belonging to ω_j, if the maximum coordinate of $\hat{\boldsymbol{y}}_i$ is $\hat{y}_j$.

0-7803-7278-6/02/$10.00 ©2002 IEEE

III. MEDIAN-ADJUSTED CLASS LABELS

The proposed method compensates for the possible imprecision of external gold standards by adjusting, if necessary, the class labels of the vectors in the training set. First, the center (medoid) of each class is computed using a multivariate median. Robust measures of location are used since they are insensitive to slight deviations from requisite normality assumptions about the underlying distribution. The medoid is determined using only data from the training set. Hence, the efficacy of this method is always measured against the original external reference test or gold standard.

Dispersion-adjusted distances are then computed between each training vector and each class medoid. A fuzzy set theoretic membership function uses these distances to adjust the class labels; in general, the further a training pattern is from a class medoid, the lower its membership value for that class. However, the class label for a training vector will only be adjusted if it is sufficiently distant from the medoid of its original class and sufficiently near another class medoid. A variant of this preprocessing method has been successfully used in several application areas [7][8].

A robust measure of dispersion is the median of the absolute deviations (MAD), τ

$$\tau(x) = \mu(|\, x - \mu(x)\,|) / 0.6745 \qquad (2)$$

where μ is a univariate median and x is a set of points (the constant ensures that as the error distribution becomes more normal, MAD estimate converges to the standard deviation) [9]. MAD may be extended to the multivariate case by using a multivariate median (see section IV) and computing, $\boldsymbol{\tau}_l=[\tau_{l1},\ldots,\tau_{ln}]$, the elements of which are coordinate dispersion measures for ω_l. Specifically, $\forall \boldsymbol{x}_i=[x_{i1},x_{i2},\ldots,x_{in}]\in\omega_l$ (i=1,…,N_l), τ_{lj} is the MAD, $\boldsymbol{\tau}_l$, of coordinate j (j=1,…,n).

The weighted distances between all training set input vectors and each class medoid may now be determined. Given, $\boldsymbol{\mu}_l=[\mu_{l1},\ldots,\mu_{ln}]$ and $\boldsymbol{\tau}_l=[\tau_{l1},\ldots,\tau_{ln}]$, the respective multivariate median and MAD of the ω_l vectors, the weighted distance, d_{il}, of an input vector, $\boldsymbol{z}_i=[z_{i1},\ldots,z_{in}]$, from the ω_l medoid is

$$d_{il} = \sum\nolimits_j |\,(z_{ij} - \mu_{lj}) / \tau_{lj}\,| \qquad (3)$$

A fuzzy set theoretic membership function of $\boldsymbol{z}_i$ in ω_c may be defined using (3) where $0\le f_c(\boldsymbol{z}_i)\le 1$

$$f_c(z_i) = [1 + d_{ic}]^{-1} \qquad (4)$$

Let $\boldsymbol{x}=[x_1,\ldots,x_n]$ be an input vector and $\boldsymbol{y}=[y_1,\ldots,y_k]$ its corresponding class label as originally assigned by the reference test using 1-of-k encoding as described in section II. The median-adjusted class label method encodes a new output vector $\boldsymbol{y}'=[y_1',\ldots,y_k']$ for $\boldsymbol{x}$ using (4) and contrast intensification [10]

$$y_c'(x) = \begin{cases} 2f_c^2(x) & \text{if } 0 \le f_c(x) \le 0.5 \\ 1 - 2(1 - f_c(x))^2 & \text{if } 0.5 \le f_c(x) \le 1.0 \end{cases} \qquad (1)$$

in order to increase values of $f_c(\boldsymbol{x})$ that are above 0.5 and reduce all other values. In general, the further $\boldsymbol{x}$ is from a class medoid, the lower its membership value for that class. While its original class label, $\boldsymbol{y}$, is crisp ($\boldsymbol{x}$ belongs to one and only one class with a degree of membership of 1), the median-adjusted class label, $\boldsymbol{y}'$, is fuzzy ($\boldsymbol{x}$ belongs to all classes to varying degrees).

IV. MULTIVARIATE MEDIANS

The popularity of the univariate median as a robust measure of univariate location is due to its high (50%) breakdown, and its affine equivariance. The breakdown point, which varies between 0 and 0.5, is a measure of the robustness of a location estimator and indicates which proportion of the data may be contaminated without seriously affecting the value of the median. The mean has a breakdown point approaching zero since any one excessive value data point can cause the mean to become correspondingly large. Affine equivariance refers to the fact that the univariate median is unaffected by changes in shift and scale. Since these qualities are desirable in a robust multivariate location estimator, the following multivariate analogs to the univariate median were chosen.

A. Component-Wise Median

The component-wise median [11] is the most easily computed multivariate median being the value, $z_1,\ldots,z_n$ which are the medians of the corresponding coordinate variates $p_1,\ldots,p_n$. It is invariant under shift and scale changes and has a 50% breakdown point but is not orthogonally invariant. Because it is a popular choice among researchers, it is included in this study for comparative purposes.

B. Spatial Median

The spatial median [12] (also known as the L1-median and mediancentre) is defined as a vector, $\boldsymbol{\Theta}_n$, that minimizes

$$\sum\nolimits_{i=1}^{n} \| X_i - \Theta \| \qquad (6)$$

where the X_i are multivariate data points and, for x=($x_1,\ldots,x_n$)

$$\| x \| = \sqrt{x_1^2 + \ldots + x_n^2} \qquad (7)$$

The spatial median is unique for dimensions two or greater and is both orthogonally invariant and shift invariant. It has a high breakdown point estimated as $(n+1)/2n$ but is not invariant to changes in scale. It is therefore not recommended for variables with components measured in different units. However, for this study, although the variables had different ranges of values, they were rescaled to vary between 0 and 100. A C++ implementation of the spatial median was developed in-house based on the algorithm described in [13].

C. *Halfspace Median*

The location depth [14] of a point Θ relative to a p-dimensional data set Z of size n is defined as the smallest number of data points in a closed halfspace with boundary through Θ. The halfspace median is defined to be the set of points {Θ} of maximal depth which is generally not a unique point. However, [15] has shown that this set of points is closed, bounded and convex. The halfspace median has affine equivariance and has a breakdown point of at least 1/(p+1) and is as high as 1/3 [16]. The original Fortran implementation of the halfspace median approximation ("deeploc") is described in [17]. This implementation was converted to a C++ program.

V. EVIDENT

EvIdent® [18] is a user-friendly, algorithm-rich, graphical environment for the detection, investigation, and visualization of novelty in a set of images as they evolve in time or frequency. Novelty is identified for a region of interest (ROI) and its associated characteristics. An ROI is a voxel grouping whose primary characteristic is its time course, which represents voxel intensity values over time. Trends due to motion and instrumentation drift can be detected.

EvIdent® uses a variant of the fuzzy c-means clustering algorithm where intra-cluster distance is minimized while inter-cluster distances are maximized (a cluster is a group of similar time courses) [19]. A cluster's centroid is a weighted average of the time courses that belong strongly to it. A membership map image represents the similarity of each active voxel to each cluster centroid (Figure 1). Centroids may be saved as time courses for further analysis and new ROIs from currently selected clusters may be created.

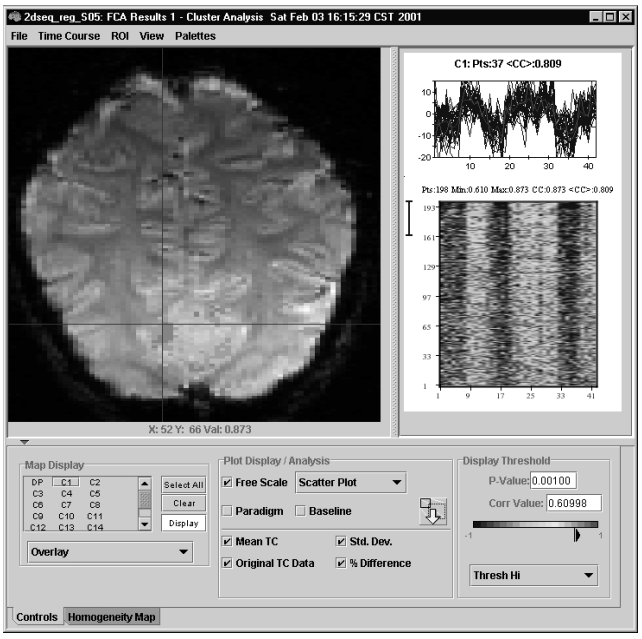

Figure 1: Example of EvIdent®'s cluster analysis display windows.

EvIdent® is written in Java and C++ and is based upon VIStA, an in-house application-programming interface (API). The VIStA API is written in Java and offers a generalized data model, an extensible algorithm framework, and a suite of graphical user interface constructs. Using the Java Native Interface, VIStA allows integration of native C++ algorithms. Java is computationally inefficient with numerical manipulations of large multi-dimensional datasets so all computationally intensive algorithms are implemented in C++.

VI. SOFTWARE METRICS AND QUALITY

Only Java objects were used in this study (C++ objects were excluded). For each of the N=312 objects, a system architect was asked to rank its quality, *Qual*, as either *low* (N_l=26), *medium* (N_m=132), or *high* (N_h=154), with respect to extensibility, maintainability, efficiency, and clarity. This ranking is used as the class label for the analysis described below. The architect was also asked to classify the objects as either *API* (application programming interface) objects or non-API objects. Finally, the architect classified each object according to its type. The *Type* indicates whether the object is a GUI object (*Type*=1), Data Model object (*Type*=2), or a Utility object (*Type*=3). Utility objects include functionality for I/O, native interface calls, and the algorithm framework.

An initial set of n=19 software metrics were compiled for each object. *Meth*, *IC*, *Kid*, *Sib*, and *Face* are the respective number of methods, inner classes, children, siblings, and implemented interfaces for each object. *DI* is the object's depth of inheritance (including interfaces). The features *rCC* and *rCR* are the respective ratios of lines of code to lines of comments and overloaded inherited methods to those that are not.

IOC is a measure of inter-object coupling, a count of the number of other objects to which the corresponding object is coupled. *RFO* is a measure of the response set for the object, that is, the number of methods that can be executed in response to a message being received by the object. Note that the standard Java hierarchy was not included in the calculations of *IC*, *Kid*, *Sib*, *Face*, *DI*, *IOC*, and *RFO*.

LCOM, the lack of cohesion in methods, is a measure that reveals the disparate nature of an object's methods. If I_j is the set of instance variables used by method j, and the sets P and Q are defined as

$$P = \{(I_i, I_j) \mid I_i \cap I_j = \varnothing\}$$
$$Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \varnothing\} \qquad (8)$$

then, if card(P)>card(Q), $LCOM$=card(P)–card(Q), otherwise $LCOM$=0 [20].

LOC, *TOK*, and *DEC* are the number of lines of code, tokens, and decisions, respectively, and *WD*, is a weighted

measure of the number of nested decisions (as the nesting increases, so does the associated weight). The final four metrics are the mean of the previous four metrics per method excluding getter and setter methods: *mLOC*, *mTOK*, *mDEC*, *mWD*, respectively.

VII. RESULTS AND DISCUSSION

For each experiment described below, MLP, using the quasi-Newton learning algorithm, was constructed with 19 input nodes (metrics) and either 3 (*Qual* or *Type*) or 2 (*API*) output nodes. Each architecture was allowed to run for 50,000 iterations. All metrics were scaled to the interval [0,100].

The datasets were randomly divided into training and test sets and test results are averaged over 10 runs of randomly resampled pairs of training and test sets. As a benchmark, the original unmodified (except for scaling to the interval [0,100]) metrics were also used for training. As an example of the disparate values for the medoids, Table I lists the spatial (S), halfspace (H), and component-wise (C) medians for both API and non-API objects metrics and compares them against the respective means (M).

TABLE I
MEDIANS OF API AND NON-API OBJECTS VERSUS MEAN

	API Objects				Non-API Objects			
	S	H	C	M	S	H	C	M
Meth	6.9	0.8	0.0	12.0	10.2	0.1	0.0	15.0
LOC	6.7	5.3	5.2	9.7	6.1	4.9	4.9	8.1
mLOC	3.3	2.5	2.5	6.0	2.7	1.9	1.9	4.5
TOK	12.6	12.6	12.5	15.7	9.3	8.8	8.8	13.8
mTOK	3.2	2.3	2.5	6.0	2.5	1.4	1.4	4.5
DEC	3.9	3.0	3.1	6.7	4.0	2.6	2.6	6.2
mDEC	2.2	1.0	1.0	4.4	2.1	0.5	0.5	4.2
WD	5.0	4.7	4.8	6.8	4.4	0.8	0.0	8.1
mWD	2.1	0.7	0.7	4.5	2.2	0.3	0.3	5.2
IC	2.0	1.1	1.1	2.9	1.9	1.1	1.1	4.5
Kid	2.2	0.4	0.0	4.3	1.3	0.0	0.0	2.6
Sib	8.9	0.4	0.0	14.4	4.6	0.1	0.0	9.5
Face	2.8	0.0	0.0	3.9	1.6	0.0	0.0	2.3
DI	5.0	0.1	0.0	9.8	1.9	0.0	0.0	3.5
rCC	6.8	0.3	0.0	9.7	7.7	0.2	0.0	8.8
rCR	3.2	2.2	2.2	3.2	2.5	1.8	1.8	3.2
IOC	1.2	0.0	0.0	1.9	1.2	0.0	0.0	2.7
RFO	8.1	9.1	9.1	11.0	4.8	0.2	0.0	7.6
LCOM	1.1	0.1	0.1	2.8	0.8	0.0	0.0	1.5

A. Quality Results

The test set for the quality assessment runs comprised 10 *low*, 40 *med*, and 50 *high Qual* objects. [The test set included 16 *low*, 92 *med*, and 104 *high Qual* objects.] Median-adjusted class labels using the component-wise median achieved a slightly poorer performance than the original metrics (Table II). However, the halfspace and spatial median versions performed significantly better with the spatial median achieving a 10% accuracy improvement (0.67 versus 0.74).

TABLE II
CLASSIFICATION RESULTS FOR QUALITY ASSESSMENT

		L	*M*	*H*	*Accuracy*
Orig	*L*	4	5	1	
	M	3	26	11	
	H	3	10	37	0.67
Comp	*L*	10	0	0	
	M	10	18	12	
	H	7	5	38	0.66
Half	*L*	7	3	0	
	M	4	25	11	
	H	3	7	40	0.72
Spat	*L*	8	2	0	
	M	3	33	4	
	H	5	12	33	0.74

B. Type Results

The test set for the type prediction runs included 20 GUI objects, *G*, 15 data model objects, *D*, and 15 algorithm framework objects, *A*. [The test set included 196 *G*, 36 *D*, and 30 *A Type* objects.] Overall, it is more difficult to predict the object type (*Type*) given only the software metrics. This is partly due to the semi-automated acquisition process for this feature (there was partial reliance on the location of object within the Java package hierarchy). Using the original metrics, the test set results were extremely poor (Table III). The classification performance using the component-wise median was also poor but the halfspace and spatial medians produced somewhat respectable results with the spatial median achieving a 59% improvement in accuracy with respect to the original metrics.

TABLE III
CLASSIFICATION RESULTS FOR TYPE PREDICTION

		G	*D*	*A*	*Accuracy*
Orig	*G*	20	0	0	
	M	15	0	0	
	A	12	1	2	0.44
Comp	*G*	16	1	3	
	D	2	12	1	
	A	2	12	1	0.58
Half	*G*	16	2	2	
	D	0	13	2	
	A	3	9	3	0.64
Spat	*G*	17	2	1	
	D	1	10	4	
	A	4	3	8	0.70

C. API Results

The test set for the prediction of whether or not an object was an API object included 50 API objects, *Y*, and 60 non-API objects, *N*. [The test set included 100 *Y* and 100 *N API* objects.] Unfortunately, none of the methods produced satisfactory results. Nevertheless, once again, the median-adjusted class label approaches were superior to the original metric with both the halfspace and spatial medians achieving a significant 19% improvement in classification accuracy (Table IV).

TABLE IV
CLASSIFICATION RESULTS FOR PREDICTION OF API

		Y	*N*	*Accuracy*
Orig	*Y*	17	33	
	N	21	41	0.52
Comp	*Y*	38	12	
	N	33	29	0.60
Half	*Y*	37	13	
	N	30	32	0.62
Spat	*Y*	37	13	
	N	30	32	0.62

VIII. CONCLUSION

The results demonstrate that median-adjusted class labels is an effective preprocessing method for multi-layer perceptron classifiers when used as a predictor of software object quality. By adjusting quality class labels, in a training set of software metrics, to reflect their proximity to all class medoids, a concomitant performance gain was realized with the underlying multilayer perceptron classifier. The spatial median provided the best and consistent classification results. One of the more popular multivariate medians, the component-wise median performed poorly. This may be partly due to its lack of orthogonal invariance: a more serious drawback in higher dimensional input spaces.

With respect to median-adjusted class labels, an avenue of future investigation would be to examine clustering strategies that would allow for several medoids for each class. Moreover, other quantile-based robust measures of dispersion may also be examined. With respect to software quality assessment, additional metrics need to be collected including more fine-grained divisions of existing metrics in order to improve overall discriminatory power.

ACKNOWLEDGEMENTS

We would like to acknowledge Aleksander Demko and Rodrigo Vivanco in the review of the collected software metrics. We would also like to thank Erica Moodie for the implementation of the halfspace median. The Natural Sciences and Engineering Research Council of Canada (NSERC) is gratefully acknowledged for their financial support of this investigation.

REFERENCES

[1] B.A. Kitchenham, R.T. Hughes, and S.G. Kinkman, "Modeling Software Measurement Data," *IEEE Trans Software Eng*, Vol. 27:9, pp. 788–804, 2001.

[2] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Trans Software Eng*, Vol. 20:6, pp. 476–493, 1994.

[3] N.E. Fenton and A.A. Kaposi, "Metrics and software structure," *J Inform Software Tech.*, Vol. 29, pp. 301–320, 1987.

[4] B.D. Ripley, "Neural networks and related methods for classification," *J Royal Stat Soc B*, Vol. 56, pp. 409–456, 1994.

[5] A.J. Shepherd, Second-Order Methods for Neural Network, Springer, New York, 1997.

[6] P. Werbos, Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University, 1974.

[7] N. Pizzi, "Fuzzy pre-processing of gold standards as applied to biomedical spectra classification," *Artif Intell Med*, Vol. 16, pp. 171–182, 1999.

[8] N. Pizzi, W. Pedrycz, "Fuzzy set theoretic adjustment to training set class labels using robust location measures," Intl Joint Conf Neural Networks, Como, Italy, pp. 109–112, 2000.

[9] P. J. Huber, "Robust estimation of a location parameter," *Annals Math Stat*, Vol. 35, pp. 73–101, 1964.

[10] L.A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans Syst Man Cybern*, Vol. 3, pp. 28–44, 1973.

[11] J.B.S. Haldane, "Note on the median of a multivariate distribution," *Biometrika*, Vol. 35, pp. 414–415, 1948.

[12] B.M. Brown, "Statistical use of spatial median," *J Roy Stat. Soc B*, Vol. 45, pp. 25–35, 1983.

[13] V. Yehuda, and C. Zhang, "The multivariate L1-median and associated data depth," *PNAS*, Vol. 97:4, pp. 1423–1426, 2000.

[14] J.W. Tukey, "Mathematics and picturing data," *Intl Congress of Mathematicians*, Vancouver, pp. 523–531, 1975.

[15] C.G. Small, "Measures of centrality of multivariate and directional distributions," *Can J Stat*, Vol. 15, pp. 31–39, 1987.

[16] D.L. Donoho, "Breakdown properties of multivariate location estimators," Ph.D. Qualifying Paper, Department of Statistics, Harvard University, 1982.

[17] A. Struyf, Dept. Mathematics and Computer Science, Univ. of Antwerp, Universiteitsplein 1, B-2610 Wilrijk-Antwerpen, Belgium. http://win-www.uia.ac.be/ u/statis/index.html.

[18] N.J. Pizzi, R.A. Vivanco and R.L. Somorjai, "EvIdent: a functional magnetic resonance image analysis system," *Artif Intell Med*, Vol. 21, pp 263–269, 2001.

[19] J. Bezdek, R. Ehrlich, and W. Full, "FCM: the fuzzy c-means clustering algorithm," *Comput Geo Sci*, Vol. 10, pp 191–203, 1984.

[20] J.F. Peters and W. Pedrycz, Software Engineering: An Engineering Approach, Wiley, New York, pp 505–551, 2000.

0-7803-7278-6/02/$10.00 ©2002 IEEE