Fuzzy Clustering of Software Metrics

Scott Dick
Department of Electrical & Computer Engineering
University of Alberta
Edmonton, AB, Canada
dick@ee.ualberta.ca

Abraham Kandel
Department of Computer Science & Engineering
University of South Florida
Tampa, FL, USA
kandel@csee.usf.edu

ABSTRACT- We investigate the use of fuzzy clustering for the analysis of software metrics databases. Software metrics are collected at various points during software development, in order to monitor and control the quality of a software product. We use fuzzy clustering to examine three collections of software metrics. This is one of the very few attempts to use unsupervised learning in the software metrics domain, even though unsupervised learning seems more appropriate for this application domain. Some characteristics of this application domain that have significant implications for machine learning are highlighted and discussed. Our results illustrate how unsupervised learning can be used in software quality control.

Keywords: Fuzzy clustering, machine learning, software metrics, software quality, unsupervised learning.[1]

I. INTRODUCTION

Software systems permeate every aspect of our lives, and are a critical part of the North American economy. They are the most complex technological systems man has ever constructed (up to 10^{20} states in a large system [1]), and the least reliable [2]. Software developers attempt to monitor and improve the quality of the software systems they are creating by collecting software metrics at various stages of the software development process. These metrics are then used to guide the allocation of additional development resources towards software modules that present significant development risks.

Each software metric quantifies some aspect of a program's source code. Simple counting metrics such as the number of lines of source code, or Halstead's software science metrics [3], simply count how many "things" there are in a program, on the assumption that the more "things" are present in a program, the more opportunity exists for errors to occur. More elaborate metrics such as McCabe's cyclomatic complexity [4], or the average nesting level of statements in a program (the *bandwidth* [5]), attempt to describe the complexity of a program. However, all of these metrics tend to be strongly correlated both to the number of failures in a module, and to each other, a phenomenon known as multicollinearity [6], [7]. Furthermore, there tend to be relatively few modules in any given system that are complex and have a high failure rate. As a result, any database of software metrics tends to be skewed towards small modules with a low failure rate. Finally, no one metric or combination of metrics is an adequate predictor of software quality, in the sense that a practical regression model can be built from them. The high correlation between software metrics and failure rates tells us that there is some relationship between metrics and quality, but there is no theory that lays out a model form for statistical regression. Thus, any of the uncountably infinite number of possible models might be a good candidate, and statistical regression does not work.

With statistical regression models unable to determine a precise relationship between software metrics and quality, some authors have turned to data mining technology to find such relationships. A call for the use of data mining to help in managing software projects is made in [8], and some candidate techniques such as neural networks and decision trees are discussed. The Goal-Question-Metric technique is used in [9], in concert with correlation analyses. A "software mining" approach is described in [10], in which awk scripts are used to crawl a reusable component library and extract software metrics. These metrics are exported to an Oracle database, where basic statistical moments are computed; the possibilities for using more advanced techniques are obvious. Khoshgoftaar et al. [11] report on the use of the Knowledge Discovery in Databases (KDD) framework in a database of software metrics from a large telecommunications system. The data mining tool used was the CART decision tree algorithm; an average classification accuracy of 75% was obtained in a tenfold cross-validation experiment.

Our main contribution in this paper is a fuzzy cluster analysis of 3 datasets of software metrics. The fuzzy c-means algorithm, a powerful and well-known unsupervised learning technique [27], is the basis of our work. Unsupervised learning algorithms look for patterns in a feature space, without requiring the correct classification of points in that feature space. This is a better fit for the software quality control problem than supervised learning algorithms, which require the correct classification of points in feature space as do statistical regression techniques. In the software quality control problem, each point in feature space is a module, and the correct classification of a point is the number of failures associated with that module. Clearly, this failure data will not be available early in the development process, but the module's features (i.e. metric values) will be. These features

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant no. PGSB222631-1999, and by the National Institute for Systems Test and Productivity under the USA Space and Naval Warfare Systems Command grant no. N00039-01-1-2248.

can be used in a Pareto analysis; the modules with the highest metric values will receive extra development resources, as these will have the highest level of risk associated with them. Previous machine learning approaches to software metrics analysis have almost all employed supervised learning approaches (see [11] – [16]). We suspect that fuzzy clustering would be a particularly effective tool in this area because there is no sharp dichotomy between low- and high-risk modules. There is instead a continuous transition from metrics and failure rates that indicate low risk to ones that indicate high risk. There was one previous attempt to use fuzzy clustering in this domain, involving the fuzzy subtractive clustering algorithm [17]. The problem with fuzzy subtractive clustering is that the expected characteristics of a cluster must be defined *a priori*.

The remainder of this paper is organized as follows. In Section two, we describe our three datasets and the previous work involving these datasets. In Section three, we present our fuzzy cluster analysis, and we offer a summary and discussion of future work in Section four.

II. Software Metrics Datasets

The datasets we examine were collected in the course of two Master's theses at the University of Wisconsin-Milwaukee. The MIS dataset was collected by Lind in [18], and widely disseminated in [19]. The datasets we have labeled 'OOSoft' and 'ProcSoft' were collected by DeVilbiss in [20]. For the reader's convenience, we will review the known characterisitics of these three datasets.

A. The MIS Dataset

The MIS dataset consists of 390 records, each representing one module sampled from a commercial medical imaging system with a total of 4,500 modules. 11 software metrics are collected for each module, along with the number of changes made to each module, as determined by the number of problem reports on file for that module. Lind [18] assumes that the number of changes represents the number of failures in a module, but this assumption was not verified. The measures used in this dataset are:

1. Lines of source code
2. Executable lines
3. Total characters
4. Lines of comments
5. Comment characters
6. Code characters
7. Halstead's N [3]
8. Halstead's Nh [3]
9. Jensen's Nf [21]
10. McCabe's cyclomatic complexity [4]
11. Bandwidth [5]

Lind conducted a correlation analysis of this dataset, using Pearson's correlation coefficient to measure the relationship between each metric and the number of changes per module. All metrics save the Bandwidth showed a strong positive correlation (all correlation coefficients above 0.6) between the metric and the number of changes per module. The Bandwidth metric had a correlation of just 0.26 with the number of changes per module. In addition to this thesis, Lind and Vairavan summarized these results in [7].

Two other works also analyzed the MIS dataset. In [22], Principal Components Analysis (PCA) [23] was used to reduce the number of features in the MIS dataset from 11 to 2. We have also used PCA in this dataset, and found that the number of principal components was 1 instead of 2. A key difference is that, in [22], the rule used to choose the number of principal dimensions was to select dimensions associated with any eigenvalue > 1. In contrast, [23] recommends that the eigenvalues be ranked, and a cut-off point be selected when there is a significant drop from one eigenvalue to the next in this ranking. Finally, the authors in [12] use a neural network to discriminate between low and high risk modules. A low risk module was one with zero or one changes, and a high risk module was one with ten or more changes. Those modules having between 2 and 9 faults were discarded.

B. The ProcSoft and OOSoft Datasets

The two remaining datasets were collected by DeVilbiss in [20]. These datasets are made up of 11 software metrics, collected from operator display applications. The first of these, ProcSoft, was programmed using structured programming and contains 422 modules. The second, OOSoft, was programmed using object-oriented techniques. A selection of 562 methods were analyzed, which implement the same functionality as the application underlying ProcSoft. Each method was treated as a module. The metrics collected in these datasets are:

1. n1 – number of unique operators [3]
2. n2 – number of unique operands [3]
3. N1 – number of operators [3]
4. N2 – number of operands [3]
5. Halstead's N [3]
6. Halstead's Nh [3]
7. Jensen's Nf [21]
8. VG1 – McCabe's complexity [4]
9. VG2 – enhanced McCabe's complexity
10. Lines of source code (LOC)
11. Comments (CMT)

DeVilbiss examined the linear correlations between each metric in the individual datasets. In the ProcSoft dataset, he reports that all the metrics correlated very strongly with each other, with the exception of CMT, which correlated well with only the LOC. In the OOSoft dataset, correlation values tend to be lower, as do the overall metric values. Change counts

were not collected in this study, and so no further information about the actual quality of the software systems is available. This is representative of the actual software quality control problem, and so we will use these datasets to study how our fuzzy clustering results might be applied in a practical setting.

III. Fuzzy Cluster Analysis

In this section, we describe our fuzzy cluster analysis. The fuzzy clustering technique we have used is the fuzzy c-means algorithm, as implemented in MATLAB® 6.0, using a fuzzifier exponent of 2.0, and a minimal improvement of 0.00001. The fuzzy c-means algorithm requires the number of clusters to be determined *a priori*; in keeping with established technique, we have repeated our fuzzy clustering experiments, varying the expected number of clusters in each experiment. Cluster validity metrics [24], [25] are then used to determine which of these experiments results in the 'best' fuzzy partition, and is therefore the correct number of clusters. In the MIS dataset, where we actually have a correct number of changes per module, we also use a prediction error as a cluster validity metric. This prediction error was computed using a tenfold cross-validation technique.

A. The MIS Dataset

In our experiments, we allowed the number of clusters in this dataset to range from 2 to 10. In Table 1, we present the cluster validity metrics for each of the resulting fuzzy partitions. The cluster validity metrics we have used are the compactness & separation of clusters (CS index) [24], a fuzzy separation index [25], and a prediction error obtained from a tenfold cross-validation technique. For the latter, we clustered the dataset using 9/10 of the dataset, and computed the sum of squared error (SSE) for the remaining 1/10 using the fuzzy nearest prototype algorithm [26]. We used the average SSE over all ten repetitions of the tenfold cross-validation as a cluster validity metric.

Table 1: Cluster Validity For MIS

Clusters	CS Index	Separation	Avg. SSE * 10^3
2	0.0064	9.5549	3.3596
3	0.0001	20.7707	2.7435
4	0.0003	20.8820	2.7124
5	0.0000	16.5433	2.5807
6	0.0005	29.1250	2.5836
7	**0.0006**	**23.9647**	**2.5753**
8	0.0001	41.6580	2.5878
9	0.0001	40.0919	2.5762
10	0.0009	26.0038	2.5821

In Table 1, maximum values of the CS index indicate the best partition, while minimal values of the Separation index and the average SSE indicate the best one. Both the CS and Separation indexes have their global optimum at 2 clusters, while the optimal value of the average SSE is at 7 clusters. The CS index and the Separation index also have local extrema at 7 clusters. Since the average SSE makes use of the true classification of each record, and since the CS index and Separation index both provide some additional support for 7 clusters, we have decided to accept 7 clusters as the correct value for this dataset.

In Table 2, we present a statistical characterization of the change counts in each cluster. We have first hardened the fuzzy clusters in classes using the method of maximum membership [27]. We determined the minimum, maximum, mean, median and standard deviation of the number of changes per module in each class, along with the number of records in each class. We have presented the classes in increasing order of the mean number of changes per module. As the reader will note, there is extensive overlap between the classes in the change dimension. We consider this to be evidence that there is no crisp dividing line between low and high-risk modules in a software system. We also wish to point out two important characteristics in this table. First, there is extensive skewness, both within each class and in the distribution of points between classes. The number of records in each class is monotonic decreasing with respect to the average number of changes, and the median number of changes is always less than the mean. Second, the variance is monotonic increasing with the mean. This indicates that the classes with the smallest number of examples also have the highest variance, which is a significant complication for machine learning algorithms.

Table 2: Class Characteristics In MIS

ID	Min	Max	Mean	Med.	STD	#
4	0	19	2.26	1	3.1	107
1	0	27	4.18	2	4.68	102
6	0	25	5.33	4	4.89	86
7	1	46	10.02	7	9.38	41
3	8	41	19.32	14	12.12	22
2	0	47	21.25	16.5	12.94	20
5	14	98	36.75	32.5	22.00	12

This kind of cluster analysis will be most useful in a Pareto analysis, in which the modules with the highest metric values are singled out for extra development effort. In the context of our fuzzy cluster analysis, the modules belonging to the *classes* with the highest metric values should be singled out for extra development. We therefore want to find a way to order the classes that does *not* depend on knowing the change counts. In Table 3, we examine the ordering of cluster centers for each metric. The second column is the cluster IDs, in increasing order of the cluster centers for that metric alone. What we notice is that the cluster centers in each dimension turn out to be ordered in the exact same way as the mean values of the change counts per class. Furthermore, the dataset is almost linear in nature; PCA indicates that there is only one principal dimension, and so this is a total ordering of

the classes. Thus, a Pareto analysis in this dataset can be conducted by determining the ordering of cluster centers for each dimension, and selecting the largest classes from this ordering. In the next section, we will use this technique to analyze the OOSoft and ProcSoft datasets, and attempt to determine if this idea of using cluster orderings in individual dimensions can be applied in other datasets as well.

TABLE 3: ORDERING OF CLUSTER CENTERS IN MIS

Feature	Ordering of Classes
Lines of code	4, 1, 6, 7, 3, 2, 5
Executable lines	4, 1, 6, 7, 3, 2, 5
Total characters	4, 1, 6, 7, 3, 2, 5
Lines of comments	4, 1, 6, 7, 3, 2, 5
Comment characters	4, 1, 6, 7, 3, 2, 5
Code characters	4, 1, 6, 7, 3, 2, 5
Halstead's N	4, 1, 6, 7, 3, 2, 5
Halstead's Nh	4, 1, 6, 7, 3, 2, 5
Jensen's Nf	4, 1, 6, 7, 3, 2, 5
McCabe's complexity	4, 1, 6, 7, 3, 2, 5
Bandwidth	4, 1, 6, 7, 3, 2, 5

B. *ProcSoft and OOSoft Datasets*

Since the ProcSoft and OOSoft datasets do not include change counts, we cannot use our average SSE metric, nor can we determine if any class ordering we find actually reflects a quality phenomenon. What we can do is run a PCA analysis to determine if the datasets are basically linear, and attempt to find a homogenous ordering of cluster centers by feature. We begin by presenting the cluster validity metrics for ProcSoft in Table 4, and OOSoft in Table 5.

TABLE 4: CLUSTER VALIDITY FOR PROCSOFT

Clusters	CS Index	Separation
2	0.0012	20.06
3	$1.46 * 10^{-4}$	39.61
4	$3.42 * 10^{-4}$	21.42
5	$4.23 * 10^{-5}$	33.16
6	$1.50 * 10^{-4}$	35.74
7	$2.32 * 10^{-4}$	13.52
8	$1.12 * 10^{-4}$	12.88
9	$1.73 * 10^{-4}$	13.06
10	$3.15 * 10^{-4}$	19.45

In Table 4, we find that the CS index and Separation index disagree; the CS index indicates that 2 clusters is the optimal value, and the Separation index indicates that 8 clusters are best. In order to decide between these two possibilities, we looked at the distribution of records between hardened classes for both partitions, and the ordering of cluster centers in each partition (see Table 6). In the 2-partition, we find that the classes are skewed *high*; class 2, which has the higher metric values, has 363 records, versus 59 records in class 1. This is in contrast to the overall dataset, which is skewed low. In the 8-partition, the distribution of records is skewed towards smaller clusters. We thus conclude that the 8-partition is correct for this dataset. A PCA analysis again showed that the dataset is almost linear, having only one principal dimension, while Table 6 indicates that there is indeed a homogenous ordering of cluster centers across all of the features. Thus, our cluster-based Pareto analysis does indeed work in the ProcSoft dataset.

TABLE 5: CLUSTER VALIDITY FOR OOSOFT

Clusters	CS Index	Separation
2	$1.02 * 10^{-4}$	229.29
3	$3.11 * 10^{-4}$	184.21
4	$3.11 * 10^{-4}$	95.89
5	$6.95 * 10^{-4}$	8.93
6	$4.95 * 10^{-4}$	16.10
7	$4.20 * 10^{-4}$	17.99
8	$1.07 * 10^{-4}$	17.29
9	$6.46 * 10^{-4}$	16.18
10	$6.46 * 10^{-4}$	17.86

TABLE 6: ORDERING OF CLUSTE4R CENTERS IN PROCSOFT

Feature	2 Clusters	8 Clusters
Halstead's n1	1,2	2,5,7,4,6,8,3,1
Halstead's n2	1,2	2,5,7,4,6,8,3,1
Halstead's N1	1,2	2,5,7,4,6,8,3,1
Halstead's N2	1,2	2,5,7,4,6,8,3,1
Halstead's N	1,2	2,5,7,4,6,8,3,1
Halstead's Nh	1,2	2,5,7,4,6,8,3,1
Jensen's Nf	1,2	2,5,7,4,6,8,3,1
VG1	1,2	2,5,7,4,6,8,3,1
VG2	1,2	2,5,7,4,6,8,3,1
LOC	1,2	2,5,7,4,6,8,3,1
CMT	1,2	2,5,7,4,6,8,3,1

In the OOSoft dataset, both the CS and Separation indexes indicate that 5 clusters is the optimal partition for this cluster. Again, a PCA analysis shows that the dataset is almost linear, with only one principal dimension. However, as can be seen in Table 7, there is no homogenous ordering of cluster centers by attribute; our technique of finding an ordering for each individual cluster breaks down in this dataset. We believe that the reason for this lies in the differences between ProcSoft and OOSoft. Both datasets were extracted from similar applications, and the same metrics were recorded for each. However, DeVilbiss [20] noticed that in the OOSoft dataset, metric values were generally lower than in ProcSoft. An analysis showed that much of the complexity in the ProcSoft modules was due to the use of SWITCH statements to determine the data type of arguments to functions. Plainly, such constructs are not needed in an object-oriented programming language. The removal of the SWITCH statements greatly simplified the program, although the lines of code underwent a smaller relative change. The usage of comments was also

significantly different than in the ProcSoft dataset. Examining Table 7, we see that it is precisely these two metrics that deviate from the ordering pattern set down by all of the other metrics. We also suspect that methods in an object-oriented program are not a natural target for software metrics developed for procedural-oriented programs; a new analysis using object-oriented metrics (such as the CK metrics [28]) would be a valuable exercise.

TABLE 7: ORDERING OF CLUSTER CENTERS IN OOSOFT

Feature	Cluster Centers
Halstead's n1	1,3,2,4,5
Halstead's n2	1,3,2,4,5
Halstead's N1	1,3,2,4,5
Halstead's N2	1,3,2,4,5
Halstead's N	1,3,2,4,5
Halstead's Nh	1,3,2,4,5
Jensen's Nf	1,3,2,4,5
VG1	1,3,2,4,5
VG2	1,3,2,4,5
LOC	1,2,4,3,5
CMT	1,2,4,5,3

IV. CONCLUSIONS

Unsupervised learning is a more natural fit to the software development process than supervised learning. In this work, we used fuzzy c-means clustering for the first time ever in the software metrics domain, and showed how the clusters we detected can be used in a Pareto analysis based on clusters of modules instead of individual modules. The Pareto principle tells us that 80% of software faults will be located in just 20% of the modules. The power of this technique is that we will select modules for additional work that would normally not be found by ranking individual modules. We have been able to replicate this analysis for the ProcSoft dataset, which also comes from a procedure-oriented program. However, we were not able to duplicate these results on the OOSoft dataset, which comes from an object-oriented program. One future study would be to use object-oriented metrics for object-oriented programs, and see if they behave in the linear fashion we observed in MIS and ProcSoft.

We can also offer two further points concerning the MIS dataset. Firstly, the dataset seems to be underdetermined. The divergence of the general cluster validity metrics from the average SSE shows that the clusters having the classically "best" quality do not actually represent this dataset. Secondly, in addition to the familiar problem of skewness, our analysis has revealed another unwelcome statistical quantity: *variance*. Our experiments showed that the mean and variance of changes per module increase monotonically with each other. The classes we are most interested in, the ones with the highest mean number of changes, were also the classes with the highest variance and the smallest number of examples. This will complicate machine learning in general, and be a serious problem for function approximation algorithms or statistical regression. This is a major challenge in the software metrics domain. Future work in this are will involve using resampling techniques to homogenize the class distribution in software metrics datasets, thus diminishing the problem of skewness.

ACKNOWLEDGMENTS

Our thanks to Dr. Vairavan of the University of Wisconsin-Milwaukee for providing the datasets in this paper.

REFERENCES

[1] Friedman, M.A.; Voas, J.M., *Software Assessment: Reliability, Safety, Testability*, New York: John Wiley & Sons, Inc., 1995.

[2] Jones, C., *Software Assessments, Benchmarks, and Best Practices*, New York: Addison-Wesley, 2000.

[3] Halstead, M., *Elements of Software Science*, New York: Elsevier, 1977.

[4] McCabe, T.J., "A Complexity Measure," *IEEE Trans. Soft. Eng.*, vol. 2 no. 4, Dec. 1976, pp. 308-20

[5] Peters, J.F.; Pedrycz, W., *Software Engineering: An Engineering Approach*, New York: John Wiley & Sons, 2000.

[6] Ebert, C.; Baisch, E., "Knowledge-based techniques for software quality management," in W. Pedrycz, W.; Peters, J.F., Eds., *Computational Intelligence in Software Engineering*, River Edge, NJ: World Scientific, 1998, pp. 295-320

[7] Lind, R.K.; Vairavan, K., "An Experimental Investigation of Software Metrics and Their Relationship to Software Development Effort," *IEEE Trans. Soft. Eng.*, vol. 15 no. 5, May 1989, pp. 649-653

[8] Paul, R.A.; Kunii, T.L.; Shinagawa, Y.; Khan, M.F., "Software metrics knowledge and databases for project management," *IEEE Trans. Knowledge & Data Eng.*, vol. 11 no. 1, Jan./Feb. 1999, pp. 255-264.

[9] Mendonca, M.G.; Basili, V.R.; Bhandari, I.S.; Dawson, J., "An approach to improving existing measurement frameworks," *IBM Systems Journal*, vol. 37 no. 4, 1998, pp. 484-501.

[10] McLellan, S.; Roesler, A.; Fei, Z.; Chandran, S.; Spinuzzi, C., "Experience using web-based shotgun measures for large-system characterization and improvement," *IEEE Trans. Soft. Eng.*, vol. 24 no. 4, April 1998, pp. 268-277.

[11] Khoshgoftaar, T.M.; Allen, E.B.; Jones, W.D.; Hudepohl, J.P., "Data Mining for Predictors of Software Quality," *Int. J. Soft. Eng. & Knowledge Eng.*, vol. 9 no. 5, 1999, pp. 547-563.

[12] Karunanithi, N.; Malaiya, Y.K. "Neural Networks for Software Reliability Engineering," in M.R. Lyu, Ed., *Handbook of Software Reliability Engineering*, New York: McGraw-Hill, 1996.

[13] Gray, A.R., "A simulation-based comparison of empirical modeling techniques for software metric models of development effort," in *Proceedings, 6th Int Conf. Neural Information Processing*, 1999, pp. 526-531.
[14] Mertoguno, J.S.; Paul, R,; Bourbakis, N.G.; Ramamoorthy, C.V., "A Neuro-Expert System for the Prediction of Software Metrics," *Eng. App. Artificial Intelligence*, vol. 9 no. 2, 1996, pp. 153-161.
[15] Khoshgoftaar, T.M.; Evett, M.P.; Allen, E.B.; Chien, P.-D., "An application of genetic programming to software quality prediction," in Pedrycz, W.; Peters, J.F., Eds., *Computational Intelligence in Software Engineering*, River Edge, NJ: World Scientific, 1998, pp. 176-195.
[16] Khoshgoftaar, T.M.; Allen, E.B.; Jones, W.D.; Hudepohl, J.P., "Classification-tree models of software-quality over multiple releases," *IEEE Trans. Rel.*, vol. 49 no. 1, March 2000, pp. 4-11
[17] Yuan, X.; Khoshgoftaar, T.M.; Allen, E.B.; Ganesan, K., "An application of fuzzy clustering to software quality prediction," in *Proceedings, 3rd IEEE Symp. App. Specific Soft. Eng. Tech.*, pp. 85-90, 2000.
[18] Lind, R.K,"An Experimental Study of Software Metrics and Their Relationship to Software Errors," Master's Thesis, University of Wisconsin-Milwaukee, 1986.
[19] Lyu, M.R., "Data and Tool Disk," in M.R. Lyu, Ed., *Handbook of Software Reliability Engineering*, New York: McGraw-Hill, 1996.
[20] DeVilbiss, W., "A Comparison of Software Complexity of Programs Developed Using Structured Techniques and Object-Oriented Techniques," Master's Thesis, University of Wisconsin-Milwaukee, 1993.
[21] Jensen, H.A.; Vairavan, K., "An Experimental Study of Software Metrics for Real-Time Software," *IEEE Trans. on Soft. Eng.*, vol. 11, no. 2, Feb. 1985, pp. 231-234.
[22] Munson, J.C.; Khoshgoftaar, T.M., "Software Metrics for Reliability Assessment," in M.R. Lyu, Ed., *Handbook of Software Reliability Engineering*, New York: McGraw-Hill, 1996.
[23] Duda, R.O.; Hart, P.E.; Stork, D.G., *Pattern Classification, 2nd Edition*, New York: John Wiley & Sons, Inc., 2001.
[24] Dunn, J.C., "A Fuzzy Relative of the ISODATA Process and its Use in Detecting Compact Well-Separated Clusters," *J. Cybernetics*, vol. 3 no. 3, 1973, pp. 32-57. Reprinted in Bezdek, J.C.; Pal, S.K., *Fuzzy Models for Pattern Recognition: Methods that Search for Structures in Data*, Piscataway, NJ: IEEE Press, 1992, pp. 82-101.
[25] Xie, X.L.; Beni, G., "A Validity Measure for Fuzzy Clustering," *IEEE Trans. Patt. Analysis & Mach. Int.*, vol. 13 no. 8, August 1991, 841-847. Reprinted in Bezdek, J.C.; Pal, S.K., *Fuzzy Models for Pattern Recognition: Methods that Search for Structures in Data*, Piscataway, NJ: IEEE Press, 1992, pp. 219-225.
[26] Keller, J.M.; Gray, M.R.; Givens, J.A., Jr., "A Fuzzy K-Nearest Neighbor Algorithm," *IEEE Trans. on Syst, Man and Cyb.*, vol. 15 no. 4, Jul/Aug 1985, pp. 580-585. Reprinted in Bezdek, J.C.; Pal, S.K. *Fuzzy Models for Pattern Recognition: Methods that Search for Structures in Data*, Piscataway, NJ: IEEE Press, 1992, pp. 258-263.
[27] Hoppner, F.; Klawonn, F.; Kruse, R.; Runkler, T., *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition*, New York: John Wiley & Sons, Inc., 1999.
[28] Chidamber, S.; Kemerer, C.F., "A Metrics Suite for Object Oriented Design", *IEEE Trans. Soft. Eng.*, vol. 20, no. 6, June 1994, pp. 476-493