Software Defect Classification: A Comparative Study of Rough-Neuro-fuzzy Hybrid Approaches with Linear and Non-linear SVMs

Rajen Bhatt[1], Sheela Ramanna[2], and James F. Peters[3]

[1] Samsung India Software Center,
Noida-201305, Uttar Pradesh, India
rajen.bhatt@gmail.com
[2] Department of Applied Computer Science
University of Winnipeg,
Winnipeg, Manitoba R3B 2E9 Canada
s.ramanna@uwinnipeg.ca
[3] Computational Intelligence Laboratory
Electrical and Computer Engineering Department
University of Manitoba,
Winnipeg, Manitoba R3T 5V6 Canada
jfpeters@ee.umanitoba.ca

Summary. This chapter is an extension of our earlier work in combining and comparing rough hybrid approaches with neuro-fuzzy and partial decision trees in classifying software defect data. The extension includes a comparison of our earlier results with linear and non-linear support vector machines (SVMs) in classifying defects. We compare SVM classification results with partial decision trees, neuro-fuzzy decision trees(NFDT), LEM2 algorithm based on rough sets, rough-neuro-fuzzy decision trees(R-NFDT), and fuzzy-rough classification trees(FRCT). The analyses of the results include statistical tests for classification accuracy. The experiments were aimed at not only comparing classification accuracy, but also collecting other useful software quality indicators such as number of rules, number of attributes (metrics) and the type of metrics (design vs. code level). The contribution of this chapter is a comprehensive comparative study of several computational intelligence methods in classifying software defect data. The different methods also point to the type of metrics data that ought to be collected and whether the rules generated by these methods can be easily interpreted.

Keywords: Classification, fuzzy-rough classification trees, neuro-fuzzy decision trees, rough sets, software defects, support vector machines.

1 Introduction

In the context of software defect classification, the term data mining refers to knowledge-discovery methods used to find relationships among defect data and the extraction of rules useful in making decisions about defective modules either during development or during post-deployment of a software system. A software

A. Abraham, R. Falcón, and R. Bello (Eds.): Rough Set Theory, SCI 174, pp. 213–231.
springerlink.com

defect is a product anomaly (e.g, omission of a required feature or imperfection in the software product) [19]. As a result, defects have a direct bearing on the quality of the software product and the allocation of project resources to program modules. Software metrics make it possible for software engineers to measure and predict quality of both the product and the process.

There have been several studies in applying computational intelligence techniques such as rough sets [18], fuzzy clustering [8, 29], neural networks [13] to software quality data. Statistical predictive models correlate quality metrics to number of changes to the software. The predicted value is a numeric value that gives the number of changes (or defects) to each module. However, in practice, it is more useful to have information about modules that are highly defective rather than knowing the exact number of defects for each module.

This chapter is an extension of our earlier work [20, 21] in comparing rough hybrid approaches in classifying software defect data. Significant enhancements include i) the comparison with linear and non-linear support vector machines (SVMs) ii) Using rough set based LEM2 algorithm [27] iii) preprocessing of data and experimenting with several iterations.

We compare SVM classification results with partial decision trees [25], neuro-fuzzy decision trees [1], rough-neuro-fuzzy decision trees, and fuzzy-rough classification trees [3, 4]. The analyses of the results include statistical tests for classification accuracy. In [20], the hybrid approach was limited to employing the strength of rough sets to attribute reduction as the first step in classification with neuro-fuzzy decision trees. In [21], Fuzzy-Rough Classification Trees that employ fuzzy-rough dependency degree [9, 2] for the induction of FRCT. Other data mining methods reported in this chapter are from rough set theory [17] and fuzzy decision trees [28].

In this work, the defect data consists of product metrics drawn from the PROMISE[1] Software Engineering Repository data set. The results are very promising in terms of how different methods point to the type of metrics data that ought to be collected and whether the rules generated by these methods can be easily interpreted.. In addition, we observed that the rule set with LEM2 was significantly smaller than our earlier reported result [20, 21]. The contribution of this chapter is a comprehensive comparative study of several computational intelligence methods in classifying software defect data.

This chapter is organized as follows. In Sect. 2, we give a brief overview of the various methods that were used in our experiments. The details of the defect data and classification methods are presented in Sect. 3. This is followed by an analysis of the classification results in Sect. 4.

2 Approaches

2.1 Neuro-Fuzzy Decision Trees

Fuzzy decision trees are powerful, top-down, hierarchical search methodology to extract easily interpretable classification rules [2]. However, they are often

[1] http://promise.site.uottawa.ca/SERepository

criticized for poor learning accuracy [26]. In [1] a Neuro-Fuzzy Decision Trees (NFDT) algorithm was proposed to improve the learning accuracy of fuzzy decision trees. In the forward cycle, NFDT constructs a fuzzy decision tree using the standard FDT induction algorithm fuzzy ID3 [28]. In the feedback cycle, parameters of fuzzy decision trees (FDT) have been adapted using stochastic gradient descent algorithm by traversing back from each leaf to root nodes. Forward cycle means construction of fuzzy decision tree by doing forward pass through data. Feedback cycle means tuning of FDT parameters using N-FDT algorithm. With the computation of mean-square-error, feedback regarding the classification performance of FDT is continuously available, which is being used to tune the FDT parameters. During the parameter adaptation stage, NFDT retains the hierarchical structure of fuzzy decision trees. A detailed discussion of NFDT algorithm with computational experiments using real-world datasets and analysis of results are available in [1].

We will now give a brief discussion about standard crisp decision trees and fuzzy decision trees. This will provide the useful reference for the study of neuro-fuzzy decision trees. The most important feature of decision trees is their capability to break down complex decision making method into a collection of locally optimal simple decisions through top-down greedy search technique. State-of-the-art survey of various crisp decision tree generation algorithms, including the most important and popular Quinlan's ID3 family and C4.5 [25], is given in [23, 24],and by Safavian and Landgrebe [22]. Although the decision trees generated by these methods are useful in building knowledge-based expert systems, they are often not capable of handling cognitive uncertainties consistent with human information processing, such as vagueness and ambiguity. In general, vagueness is related to the difficulty in making sharp classification boundaries. Ambiguity is associated with one-to-man mapping. To overcome these deficiencies, various researchers have developed fuzzy decision tree induction algorithms [28]. All fuzzy decision tree generation techniques evaluate classification abilities of fuzzified attributes using some suitable measure of uncertainty. Incorporating this measure in crisp decision tree generation algorithm like ID3, fuzzy decision trees can be constructed.

Figure 1 shows fuzzy decision tree using fuzzy ID3 algorithm for a toy dataset of two class classification problem. As shown in Fig. 1, fuzzy decision trees are composed of a set of internal nodes representing variables used in the solution of a classification problem, a set of branches representing fuzzy sets of corresponding node variables, and a set of leaf nodes representing the degree of certainty with which each class has been approximated. Patterns are classified by starting from the root node and then reaching to one or more leaf nodes by following the path of degree of memberships greater than zero. Each *path-m* is defined on the premise space composed of input features available in traversing from root node to m^{th}leaf node. In Fig. 1, *path-1*, *path-2*, and *path-3* are composed on the premise space $x1 = x2 = x3 = [x6, x2]$, where as *path-4* and *path-5* are composed on the premise space $x4 = x5 = [x6]$. Number of variables appearing on the path defines the length of that path. For example, in Fig. 1, length of

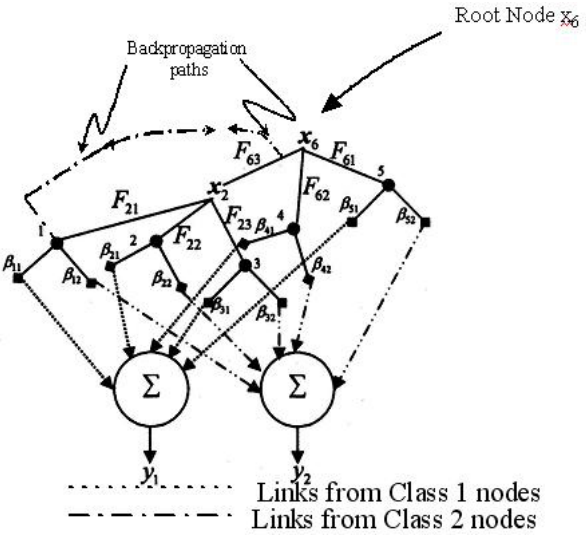

Fig. 1. Exemplary neuro-fuzzy decision tree

path-1, *path-2*, *path-3* = 2. Fuzzy decision tree being a hierarchical structure, share the membership functions along the paths leading to each leaf node from root node. In Fig. 1, membership function F_{63} has been shared by *path-1*, *path-2*, and *path-3*. With this preliminary discussion on fuzzy decision trees and its advantages over crisp decision trees, what follows is the formal notation and details of the neuro-fuzzy decision trees:

Figure 1 shows an exemplary NFDT with two summing nodes to carry out the inference process. There are five paths starting from root node to five leaf nodes. Root node is indicated by x_6. Leaf nodes are shown by dots and indexed as $m = 1, 2, , 5$. Training patterns are labeled as $\{x_i, y_i\}$ where $i = 1, ..., n$ and $y_i \in \{0, 1\}$ where where x_i represents the i^{th} object(pattern) and y_i represents the prediction certainty of the decision class for the i^{th} object. The input to NFDT is a decision table. Let β_{ml} be the certainty factor corresponding to m^{th} leaf node and decision *class-l*. From all the leaf nodes, certainty corresponding to decision *class-l* are summed up to calculate output y_l. Traversing path from root node x6 to second leaf node is represented by:

$$\begin{aligned} &path_2 = \mathrm{x}_6 \text{ is } \mathrm{F}_{63} \wedge \mathrm{x}_2 \text{ is } \mathrm{F}_{22} \\ &leaf_2 : y_1 = \mathrm{Class}_1(\beta_{21}), y_2 = \mathrm{Class}_2(\beta_{22}) \end{aligned} \tag{1}$$

The firing strength (FS) of *path-m* with respect to decision *class-l* for the i^{th} object is defined by (2)

$$FS_m = \mu^i_{path_m} \times \beta_{ml}, \tag{2}$$

where $\mu^i_{path_m}$ is the membership degree of i^{th} object for *path-m* and can be calculated as shown in (3)

$$\mu^i_{path_m} = \prod_j \mu_{F^m_j}\left(x^i_j\right). \tag{3}$$

where $\mu_{F^m_j}\left(x^i_j\right)$ is the degree of membership of the i^{th} pattern of the j^{th} input x_j into F^m_j. F^m_j is fuzzy membership function for j^{th} attribute on *path-m*. This way, $\mu^i_{path_m}$ is zero if for any of the input variable on m^{th} path the degree of membership of i^{th} pattern to the fuzzy membership function F^m_j is zero.

Firing strengths of all the paths for a particular decision *class-l* are summed up to calculate the prediction certainty y^i_l of i^{th} object(pattern) to l^{th} class through fuzzy decision tree as shown in (4)

$$y^i_l = \sum_{m=1}^{M} FS_m, \tag{4}$$

where $0 \leq y^i_l \leq 1$ and q is total number of classes. When classification to a unique class is desired, the class with the highest membership degree needs to be selected, i.e., classify given object(pattern) to class l_0, where

$$l_0 = \arg \max_{l=1,\ldots,q} \left\{y^i_l\right\}. \tag{5}$$

To fuzzify input attributes, we have selected Gaussian membership functions out of many alternatives due to its differentiability property (i.e., existence of the differentiation). For i^{th} object(pattern), membership degree of *path-m* can be calculated as shown in (6)

$$\mu^i_{path_m} = \prod_j \mu_{F^m_j}\left(x^i_j\right) = \prod_j \exp\left(\frac{\left(x^i_j - c_{jm}\right)^2}{2\sigma^2_{jm}}\right), \tag{6}$$

where c_{jm} and σ_{jm} are center and standard deviation (width) of Gaussian membership function of j^{th} attribute of input object(pattern) x_i on *path-m*, i.e., of F^m_j. We now briefly outline the strategy of NFDT by performing an adaptation of all types of parameters (centers, widths, and certainty factors) simultaneously on the structure shown in Fig. 1. We define as the error function of the fuzzy decision tree, the mean-square-error defined by (7)

$$MSE = \frac{1}{2n} \sum_{l=1}^{q} \sum_{i=1}^{n} \left(d^i_l - y^i_l\right)^2, \tag{7}$$

where n is the total number of training patterns and d^i_l and y^i_l are the desired prediction certainty and the actual prediction certainty of *class-l* for i^{th} object(pattern), respectively. At each epoch (iteration), the complete parameter $\mathrm{P} = \{c_{jm}, \sigma_{jm}, \beta_{ml} \mid m = 1, ..., M; l = 1, ..., q\}$ is moved by a small distance η in the direction in which MSE decreases most rapidly, i.e., in the direction of the negative gradient $-\frac{\partial E}{\partial \theta}$ where θ is the parameter vector constituted from the set P. This leads to the parameter update rule shown in (8)

$$\theta^{\tau+1} = \theta^{\tau} - \eta \frac{\partial E}{\partial \theta}, \tag{8}$$

where τ is the iteration index and η is the learning rate. The update equations for centers, widths, and certainty factors can be found in [1]. Parameter adaptation continues until error goes below certain small positive error goal ϵ or the specified number of training epochs has been completed. Neuro-fuzzy decision trees take as input the fuzzy decision tree structure and try to minimize the mean-square-error by tuning the centers and standard deviations of Gaussian membership functions along with certainty factors associated with each node. For example, if there is high degree of overlap (or very less overlap) between adjacent membership functions in the initial fuzzy decision tree structure if cluster centers are too close (or too far). Neuro-fuzzy decision tree algorithm tries to minimize the mean-square-error and in the process adjust the cluster centers and standard deviations.

What follows are details about the computational set-up for the experiments reported in this chapter with NFDT. All the attributes have been fuzzified using fuzzy c-means algorithm [5] into three fuzzy clusters. From the clustered row data, Gaussian membership functions have been approximated by introducing the width control parameter λ. The center of each gaussian membership function has been initialized by fuzzy cluster centers generated by the fuzzy c-means algorithm. To initialize standard deviations, we have used a value proportional to the minimum distance between centers of fuzzy clusters. For each numerical attribute x_j and for each gaussian membership function, the Euclidean distance between the center of F_{jk} and the center of any other membership function F_{jh} is given by $dc\,(c_{jk}, c_{jh})$, where $h \neq k$. For each k^{th} membership function, after calculating $dc_{\min}\,(c_{jk}, c_{jh})$, the standard deviation σ_{jk} has been obtained by (9)

$$\sigma_{jk} = \lambda \times dc_{\min}\,(c_{jk}, c_{jh})\,; 0 < \lambda \leq 1, \tag{9}$$

where λ is the width control parameter. For the computational experiments reported here, we have selected various values of $\lambda \in (0, 1]$ to introduce variations in the standard deviations of initial fuzzy partitions.

2.2 Fuzzy-Rough Classification Trees

Fuzzy-Rough Classification Trees (FRCT) integrate rule generation technique of fuzzy decision trees and rough sets. The measure used for the induction of FRCT is fuzzy-rough dependency degree proposed in [3, 4]. The dependency degree measure in the context of rough set theory has been proposed by Pawlak [17]. Fuzzy-rough dependency degree measure is an extension of Pawlak's measure to accommodate fuzzy data. Pawlak's measure is only applicable to crisp partitions of feature space. In [3, 4], we have shown that our measure of fuzzy-rough dependency degree is more general one and covers Pawlak's measure as a limiting case when partitions are crisp rather than fuzzy. In this sect., we briefly outline the steps of computing the fuzzy-rough dependency degree.

Formally, a data (information) table IS is represented by a pair (X, A), where X is a non-empty, finite set of objects and A is a non-empty, finite set of attributes, where $a_j : X \rightarrow Va_j$ for every $a_j \in A$ where a_j is the j^{th} input of attribute a and Va_j is the value set of a_j. A decision table is represented by a pair (X, C, D), where $C, D \subseteq A$. In other words, the attribute set A is partitioned into: condition attributes C and decision attribute D. Let $|C| = p$ i.e., the total number of input of variables are p. Let $|D| = q$, i.e., the total number of classes are q. In other words, each input pattern is classied into one of the q classes. Let F_{jk} be the k^{th} fuzzy set of attribute a_j. The fuzzy set represents overlapping and non-empty partitions of real-valued attributes $a_j \in C$ where $(1 \leq j \leq p)$ on the set of training set $T \subseteq X$. For notational convenience, we will use j instead of a_j for attributes.

The membership function of the lower approximation of an arbitrary *class-l* of F_{jk} for the i^{th} object(pattern) denoted by $x_j^i \in T$ with decision d^i is given by:

$$\mu_{\underline{l}}\left(F_{jk}\right) = \inf_{\forall i \in U} \max\left\{1 - \mu_{F_{jk}}\left(x_j^i\right), \mu_l\left(d^i\right)\right\}$$

The dependency degree $\gamma_{x_j}(d)$ for the ith object (pattern) and jth attribute can be calculated as follows:

- Calculate the lower approximation member function $\mu_{\underline{l}}\left(F_{jk}\right)$ using the above definition
- Calculate fuzzy positive region $\mu_{POS}\left(F_{jk}\right) = \sup_{l=1,..,q} \left\{\mu_{\underline{l}}\left(F_{jk}\right)\right\}$
- Calculate the degree of membership of the i^{th} pattern to the fuzzy positive region
  $\mu_{POS}\left(x_j^i\right) = \sup_{l=1,..,q} \min\left\{\mu_{F_{jk}}\left(x_j^i\right), \mu_{POS}\left(F_{jk}\right)\right\}$
- Calculate the dependency degree $\gamma_{x_j}(d) = \frac{\sum_{i=1}^{n} \mu_{POS}\left(x_j^i\right)}{n}$

Fuzzy-rough dependency degree of attribute x_j, denoted here as γ_{x_j} lies between 0 and 1, i.e., $0 \leq \gamma_{x_j}(d) \leq 1$. $d = 1$ indicates that decision attribute d completely depends on input attribute x_j, in other words, x_j alone is sufficient to approximate all the decisions given in decision attribute d. $d = 0$ indicates that decision attribute d is not completely dependent on on input attribute x_j. Any value of $\gamma_{x_j}(d)$ that is in (0,1) indicates partial dependency. Partial dependency means *addition* of other input attributes is required to completely approximate all the decisions given in decision attribute d. This property of fuzzy-rough dependency degree makes it a good choice as an attribute selection criterion for the induction of fuzzy decision trees. We call fuzzy decision trees wherein fuzzy-rough dependency degree is used as an attribute selection criterion, a fuzzy-rough classification trees.

Given fuzzy partitions of feature space, leaf selection threshold β_{th}, and fuzzy-rough dependency degree γ as expanded attribute (attribute to represent each node in fuzzy decision tree) selection criterion, the general procedure for generating fuzzy decision trees using FRCT algorithm is outlined in Alg. 1.

Algorithm 1. Algorithm for generating fuzzy decision trees using FRCT

Require: fuzzy partitions of feature space, β_{th}, γ
Ensure: fuzzy decision trees
1: **while** $\exists$ candidate nodes **do**
2: Select node with highest γ; ▷ dependency degree
3: Generate its child nodes; ▷ root node will contain attribute with highest γ
4: **if** $\beta_{child-node} \geq= \beta_{th}$ **then**
5: child-node = leaf-node
6: **else**
7: Search continues with child-node as new root node
8: **end if**
9: **end while**

Before training the initial data, the α cut is usually used for the initial data [4]. Usually, α is in the interval $(0, 0.5]$. A detailed description of fuzzy-rough dependency degree is available in [3]. The cut of a fuzzy set F is defined as:

$$\mu_{F_\alpha}(a) = \begin{cases} \mu_F(a); \mu_F(a) \geq \alpha \\ 0; \mu_F(a) < \alpha \end{cases}.$$

In the case of FRCT experiments, fuzzy partitioning of the feature space has been generated by the following method. Fuzzy c-means [5] algorithm has been utilized to fuzzify continuous attributes into three fuzzy clusters. The triangular approximation of the clustered raw data is done in two steps. First, the convex hull of the original clustered data is determined through MATLAB® function "convhull", and then the convex hull is approximated by a triangular membership function. We mention here that three fuzzy clusters have been chosen only to report experimental results. Choosing different number of clusters may affect the result. In general, one should iterate from a few minimum to maximum number of clusters, construct fuzzy-rough classification trees, and choose one which gives best classification accuracy with acceptable number of rules.

2.3 Support Vector Machines

In this sect., we give a brief discussion of linear and nonlinear Support Vector Machines (SVMs) used in our computational experiments [6]. Linear and nonlinear SVMs trained on non separable (and separable) data results in a quadratic programming problem.

Linear SVM

Let training patterns are labeled as $\{x_i, y_i\}$, where $i = 1, ..., n$, $y_i \in \{-1, +1\}$, $x_i \in \Re$. Let there exist some separating hyper plane which separates the positive from the negative patterns. The points x, which lie on the hyper plane satisfy

$$x_i \cdot \mathbf{w} + b = 0 \tag{10}$$

where $\mathbf{w}$ is normal to the hyper plane $\frac{|b|}{|\mathbf{w}|}$ is the perpendicular distance from the hyper plane to the origin, and $\|\mathbf{w}\|$ is the Euclidean norm of $\mathbf{w}$. For the linearly separable case, the SVM algorithm simply looks for the separating hyper plane with largest margin i.e., $\mathbf{w}$ and b which can maximize the margin. Once optimal b and $\mathbf{w}$ are obtained, we simply have to determine on which side of the decision boundary a given test pattern x lies and assign the corresponding class label, i.e., we take the class of x to be $sgn(\mathbf{w} \cdot x + b)$ given as

$$x_i \cdot \mathbf{w} + b \geq 1, \text{ for } y_i = 1, \; x_i \cdot \mathbf{w} + b \leq -1, \text{ for } y_i = -1. \tag{11}$$

Equation 11 can be combined into one set of inequalities:

$$y\,(x_i \cdot \mathbf{w} + b) - 1 \geq 1, \text{for } i \tag{12}$$

However, in practice, it is difficult to find problems with perfectly linearly separable case. The actual SVM formulation described above for the linearly separable case is modified by introducing positive slack variables ζ_i where i=1,...,n in the constraints. Equation 11 can be be rewritten as:

$$x_i \cdot \mathbf{w} + b \geq 1 - \zeta_i, \text{ for } y_i = 1, \tag{13}$$

$$x_i \cdot \mathbf{w} + b \leq -1 - \zeta_i, \text{ for } y_i = -1 \; \textit{where} \; \zeta_i \geq 0 \text{ for } \; i \tag{14}$$

NonLinear SVM

To handle cases where the decision function is not a linear function of the data, a nonlinear version of the SVM is normally used. In this case, we first map the data to some other (possibly infinite dimensional) Euclidean space H using a mapping ϕ where $\phi : \Re \rightarrow H$. SVM training algorithm would only depend on the data through dot products in H i.e., on functions of the form $\phi\,(x_i) \cdot \phi\,(x_j)$.

Now if there is a 'Kernel function' K such that, $K\,(x_i) \cdot K\,(x_j) = \phi\,(x_i) \cdot \phi\,(x_j)$. we would only need to use K in the training algorithm, and would never need to explicitly know the value(s) for ϕ. One such example is Gaussian kernel, used in the computational experiments reported here. A detailed discussion of linear and nonlinear SVMs for separable and non-separable cases, with interesting mathematical results can be found in [6].

3 Software Defect Data

The PROMISE data set includes a set of static software metrics about the *product* as a predictor of defects in the software. The data includes measurements for 145 modules (objects). There are a total of 94 attributes and one decision attribute (indicator of defect level). The defect level attribute value is TRUE if the class contains one or more defects and FALSE otherwise. The metrics at the *method level* are primarily drawn from Halstead's Software Science metrics [10] and McCabe's Complexity metrics [15]. The metrics at the *class level*, include such standard measurements as Weighted Methods per Class (WMC),

a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22
0	0	24	4	100	0	0	2	110	73	0	1	0	0	1	1	1	0	5.8	1.5	17	0
0	0	19	4	100	0	0	3	78	30	0	1	0	0	1	1	1	0	0	0	0	0
100	0	13	1	88	0	0	0	99	99	0	1	0	0	1	1	1	1	0	0	0	0
0	0	21	4	100	0	0	2	68	30	0	1	0	0	1	1	1	0	7.7	1.5	17	0
5	0	17	2	90	0	0	1	69	36	0	1	0	0	1	1	1	0	0	0	0	0
0	0	10	1	88	0	0	0	22	22	0	1	0	0	1	1	1	1	0	0	0	0
0	0	19	3	94	0	0	3	53	17	0	1	0	0	1	1	1	0	4.4	1.5	12	0
0	0	14	2	90	0	0	1	37	25	0	1	0	0	1	1	1	0	0	0	0	0
0	0	18	2	100	1	1	1	38	13	0	1	0	0	1	1	1	0	0	0	0	0
0	0	18	2	100	1	1	1	38	13	0	1	0	0	1	1	1	0	0	0	0	0
0	0	9	1	0	4	0	0	7	7	0	1	0	0	1	1	1	1	0	0	0	0
0	0	6	2	59	0	0	1	33	25	0	1	0	0	1	1	1	0	0	0	0	0
0	0	5	1	0	0	0	0	7	7	0	1	0	0	1	1	1	0	0	0	0	0
100	0	4	1	75	2	0	0	8	8	0	1	0	0	1	1	1	0	4.8	1	4.8	0
0	0	4	1	0	0	0	0	6	6	0	1	0	0	1	1	1	1	0	0	0	0
100	0	24	4	96	0	0	3	140	35	0	1	0	0	1	1	1	1	0	0	0	0
0	0	23	2	92	0	0	1	53	25	0	1	0	0	1	1	1	0	5.3	1	8	0
0	0	9	2	28	0	0	1	31	23	0	1	0	0	1	1	1	1	0	0	0	0
0	0	2	2	94	0	0	1	52	17	0	1	0	0	1	1	1	0	0	0	0	0
0	0	6	1	80	0	0	0	15	15	0	1	0	0	1	1	1	1	0	0	0	0
0	0	5	1	25	0	0	0	9	9	0	1	0	0	1	1	1	1	0	0	0	0
0	0	0	1	100	0	0	0	0	0	0	1	0	0	1	1	1	0	5.3	1.5	12	0
0	0	0	1	100	0	0	0	0	0	0	1	0	0	1	1	1	0	5.3	1.5	12	0
0	0	8	1	58	0	0	0	37	34	0	1	0	0	1	1	1	0	5.3	1.5	12	0
0	0	15	1	77	0	0	0	12	12	0	1	0	0	1	1	1	1	0	0	0	0
0	0	11	1	76	0	0	0	28	28	0	1	0	0	1	1	1	0	5.3	1.5	12	0
0	0	12	1	55	0	0	0	24	20	0	1	0	0	1	1	1	2	6.2	2	39	0
100	0	4	1	100	0	0	0	11	11	0	1	0	0	1	1	1	0	5.3	1.5	12	0
17	0	14	4	94	0	0	1	198	24	0	1	0	0	1	1	1	0	5.3	1.5	12	0
100	0	2	2	85	0	0	1	11	7	0	1	0	0	1	1	1	0	0	0	0	0

Fig. 2. Exemplary data set

Depth of Inheritance Tree (DIT), Number of Children (NOC), Response For a Class (RFC), Coupling Between Object Classes (CBO), and Lack of Cohesion of Methods (LCOM) [7]. A sample data set of 30 modules with 22 attributes are shown in Fig. 2.

In this chapter, for the purposes of illustration we have given a brief description of the first 22 attributes. Since the defect prediction is done at a class-level, all method level features were transformed to the class level. Transformation was achieved by obtaining min, max, sum, and avg values over all the methods in a class. Thus this data set includes four features for each method-level features.

- a1: PERCENT-PUB-DATA . The percentage of data that is public and protected data in a class.
- a2: ACCESS-TO-PUB-DATA. The amount of times that a class's public and protected data is accessed.
- a3: COUPLING-BETWEEN-OBJECTS. The number of distinct non-inheritance-related classes on which a class depends.

- a4: DEPTH. The level for a class. For instance, if a parent has one child the depth for the child is two.
- a5: LACK-OF-COHESION-OF-METHODS. For each data field in a class, the percentage of the methods in the class using that data field. This metric indicates low or high percentage of cohesion.
- a6: NUM-OF-CHILDREN. The number of classes derived from a specified class.
- a7: DEP-ON-CHILD. Whether a class is dependent on a descendant.
- a8: FAN-IN. This is a count of calls by higher modules.
- a9: RESPONSE-FOR-CLASS. A count of methods implemented within a class plus the number of methods accessible to an object class due to inheritance.
- a10: WEIGHTED-METHODS-PER-CLASS. A count of methods implemented within a class rather than all methods accessible within the class hierarchy.
- a11: minLOC-BLANK. Lines with only white space or no text content.
- a12: minBRANCH-COUNT. This metric is the number of branches for each module.
- a13: minLOC-CODE-AND-COMMENT. Lines that contain both code and comment.
- a14: minLOC-COMMENTS. Minimum lines with comments.
- a15: minDESIGN-COMPLEXITY. Design complexity is a measure of a module's decision structure as it relates to calls to other modules.
- a17: minESSENTIAL-COMPLEXITY. Essential complexity is a measure of the degree to which a module contains unstructured constructs.
- a18: minLOC-EXECUTABLE. Minimum Source lines of code that contain only code and white space.
- a19: minHALSTEAD-CONTENT. Complexity of a given algorithm independent of the language used to express the algorithm.
- a20: minHALSTEAD-DIFFICULTY. Minimum Level of difficulty in the program.
- a21: minHALSTEAD-EFFORT. Minimum estimated mental effort required to develop the program.
- a22: minHALSTEAD-ERROR-EST. Estimated number of errors in the program.

Data Preprocessing

Since all the data is real-valued with a wide variation in the values for attributes from 0.1 to 10^5, it was necessary to normalize the attribute values for experiments that were not based on fuzzy sets. What follows are some data preprocessing tasks that were performed on the defect data:

- All attributes were normalized using the WEKA[2] unsupervised instance based filter method.

[2] `http://www.cs.waikato.ac.nz/ml/weka`

- For 10-Fold CV, pairs of testing-training data sets were generated independently and used in all our experiments with different methodologies.
- All attribute values were discretized in experiments with LEM2 and J48. The version of LEM2 included in RSES[3] works only with discretized data. Discretization alogrithm implemented in WEKA was used with J48. Discretization is by simple binning in the unsupervised mode.

Experimental Setup - Discretization in RSES

Whenever the domain of a real-valued condition attribute is exceptionally large, then the number of decision rules that are generated can be unmanageably large. In such cases, discretization of attribute value sets provides a mechanism for reducing the domain space without significantly altering the quality of the rules that are derived. That is, we need to obtain approximate knowledge of a continuum by considering parts of the continum for an attribute. Discretization of a continuum entails the partition of the interval of a real-valued attribute into subintervals of reals. Let $DT = (U, A \cup d)$ be a decision table where $U = \{x_1, \ldots, x_n\}$. In addition, assume that the value set V_a for each attribute of DT is a subset of the reals. For example, consider the interval of reals $V_a = [la, ra]$ for values of an attribute $a \in A$ in a decision system $DT = (U, A \cup d)$ where $la \leq a(x) \leq ra$. Discretization of V_a entails searching for a partition P_a of V_a for any a $a \in A$ (i.e., discovering a partition of the value sets of conditional attributes into intervals). A partition of V_a is defined by a sequence of what are known as cuts $l_a = v_1 < v_2 < v_3 < \ldots << v_{n-1} < v_n = r_a$ so that $V_a = [l_a, v_2)[v_2, v_3) \ldots [v_{n-1}, r_a)$. The search for partitions of attributes into subintervals of the reals is carried out on consistent decision tables. In rough set theory, discretization leads to partitions of value sets so that if the name of the interval containing an arbitrary object is substituted for any object instead of its original value in DT, a consistent decision system is also obtained. The discretization concept defined by cuts has been generalized by using oblique hyperplanes. The boundary between each pair of decision classes is a linear plane called a hyperplane. The quality of a hyperplane has been treated by a number of measures. Measure values are viewed as energy levels of a hyperplane. During discretization of a set of numeric attributes, the search of hyperplanes is carried out using simulated annealing. A detailed description of this approach to discretization is outside the scope of this article. A complete presentation containing the details about discretization in the context of rough sets is given in [16].

Experimental Setup - Rough Set-based LEM2 and J48

LEM2 based on rough set theory learns the concept with the smallest set of rules [27]. In the results reported in [20, 21], rule-set used by the rough set classifier in RSES was quite large (average number of rules was 280). We also compared our results with a classical partial decision tree-based method (J48) method in WEKA using a variant of the well-known C4.5 revision 8 algorithm [25].

[3] `http://logic.mimuw.edu.pl/~rses`

Experimental Setup - Hybrid Methods

In all our hybrid methods involving fuzzy sets, the attribute values were not normalized. The Rough-NFDT method included i) generating reducts from rough set methods ii) using the data from the reduced set of attributes to run the NFDT algorithm. For the NFDT algorithm, after attribute fuzzification, the fuzzy ID3 algorithm with cut $\alpha = 0$ and leaf selection threshold $\beta_{th} = 0.75$ was used. The fuzzy decision trees have been tuned using the NFDT algorithm for 500 epochs with the target MSE value 0.001.

For the computational experiments reported with the FRCT, all parameters were set as described in Sect. 2.2.

Experimental Setup - SVM

The data set was classified using nonlinear SVM(SVM-NL) as well as linear SVM (SVM-L). The tuning parameters involved with nonlinear SVM are penalty parameter C and Gaussian Kernel parameter μ. These two parameters were tuned based on grid search [12]. The range for C used is 2^{-12} to 2^{12}. The range for μ used is 2^{-35} to 2^{4}. The best values of C and μ are 2^{-1} and 2^{-29}. All algorithms were implemented in MATLAB 7.3.0 (R2006b) [14] environment on a PC with Intel Core2Duo processor (2.13GHz), 1GB RAM running Ms-Windows XP operating system. The dual quadratic programming problems arising in SVM were solved using Mosek optimization toolbox [4] for MATLAB which implements fast interior point based algorithms.

4 Analysis of Classification Results

A comparison of pairs of differences in classification accuracy during one-fold of a 10Fold CV and a paired t-test is also discussed in this sect. Table 1 gives a summary of computational experiments using seven methods and Table 2 gives the average size of the rule set (and support vectors). Percentage classification accuracy has been calculated by $\frac{n_c}{n} \times 100\%$, where n is the total number of test patterns, and n_c is the number of test patterns classified correctly.

Figure 3 gives a sample LEM2 rule set for a single run of the 10fold CV. It can be observed that the most frequestly used attributes (metrics) are:

a4(DEPTH), a5(LACK-OF-COHESION-OF-METHODS),
a23(minHALSTEAD-LENGTH), a35(maxLOC-COMMENTS),
a53(avgLOC-BLANK), a56(avgLOC-COMMENTS) and a63(avgHALSTEAD-LENGTH).

4.1 T-Test

In this sect., we discuss whether there is any difference between the various methods in terms of classification accuracy (and the number of rules) statistically

[4] `http://www.mosek.com`

Table 1. Defect data classification I

	%Accuracy						
Run	NFDT	R-NFDT	LEM2	FRCT	J48	SVM-NL	SVM-L
1	93	71	85	86	86	76	71
2	83	93	82	86	79	76	86
3	64	71	55	71	57	57	57
4	71	71	80	93	86	86	71
5	64	57	44	71	28	64	64
6	79	79	57	79	57	93	71
7	86	71	67	86	50	64	50
8	71	79	67	71	79	64	79
9	93	100	86	93	93	93	100
10	89	89	85	89	84	89	89
Avg.Acc	80	78	71	83	73	77	74

Table 2. Defect data classification II

Average Number of Rules and Support Vectors						
NFDT	R-NFDT	LEM2	FRCT	J48	SVM-NL	SVM-L
7.2	5.6	26	17.3	12	123	49

Decision rules

(a35="(-Inf,1.29E-4)")&(a56="(-Inf,9.5E-6)")&(a4="(3.575E-4,Inf)")=>(decision={F[37]})
(a4="(-Inf,6.8E-5)")&(a35="(-Inf,1.29E-4)")&(a56="(-Inf,9.5E-6)")&(a53="(-Inf,1.65E-5)")=>(decision={T[23]})
(a4="(-Inf,6.8E-5)")&(a35="(-Inf,1.29E-4)")&(a23="(-Inf,4.2E-5)")&(a63="(-Inf,0.0758755)")&(a56="(-Inf,9.5E-6)")&(a53="(-Inf,1.65E-5)")=>(decision={T[14]})
(a53="(1.65E-5,Inf)")&(a35="(-Inf,1.29E-4)")&(a63="(0.0758755,Inf)")&(a4="(6.8E-5,3.575E-4)")=>(decision={F[10]})
(a4="(-Inf,6.8E-5)")&(a56="(9.5E-6,Inf)")&(a84="(0.913131,Inf)")&(a53="(1.65E-5,Inf)")&(a23="(-Inf,4.2E-5)")&(a35="(-Inf,1.29E-4)")=>(decision={T[7]})
(a4="(-Inf,6.8E-5)")&(a35="(-Inf,1.29E-4)")&(a23="(-Inf,4.2E-5)")&(a53="(1.65E-5,Inf)")&(a63="(-Inf,0.0758755)")&(a56="(9.5E-6,Inf)")=>(decision={T[6]})
(a53="(1.65E-5,Inf)")&(a56="(9.5E-6,Inf)")&(a84="(-Inf,0.913131)")&(a23="(4.2E-5,Inf)")&(a4="(6.8E-5,3.575E-4)")&(a35="(-Inf,1.29E-4)")=>(decision={F[5]})
(a56="(9.5E-6,Inf)")&(a35="(1.29E-4,Inf)")&(a53="(1.65E-5,Inf)")&(a23="(-Inf,4.2E-5)")&(a63="(-Inf,0.0758755)")=>(decision={F[5]})
(a53="(1.65E-5,Inf)")&(a4="(-Inf,6.8E-5)")&(a56="(9.5E-6,Inf)")&(a63="(0.0758755,Inf)")&(a84="(-Inf,0.913131)")&(a23="(4.2E-5,Inf)")=>(decision={T[5]})
(a53="(1.65E-5,Inf)")&(a56="(9.5E-6,Inf)")&(a23="(-Inf,4.2E-5)")&(a84="(-Inf,0.913131)")&(a4="(-Inf,6.8E-5)")&(a35="(1.29E-4,Inf)")&(a63="(-Inf,0.0758755)")=>(decision={F[3]})
(a53="(1.65E-5,Inf)")&(a63="(-Inf,0.0758755)")&(a84="(0.913131,Inf)")&(a4="(6.8E-5,3.575E-4)")&(a23="(4.2E-5,Inf)")&(a35="(-Inf,1.29E-4)")=>(decision={F[3]})
(a53="(1.65E-5,Inf)")&(a4="(-Inf,6.8E-5)")&(a23="(-Inf,4.2E-5)")&(a35="(-Inf,1.29E-4)")&(a56="(-Inf,9.5E-6)")&(a63="(0.0758755,Inf)")=>(decision={F[3]})
(a53="(1.65E-5,Inf)")&(a56="(9.5E-6,Inf)")&(a23="(4.2E-5,Inf)")&(a35="(1.29E-4,Inf)")&(a63="(0.0758755,Inf)")&(a84="(0.913131,Inf)")=>(decision={T[3]})
(a53="(1.65E-5,Inf)")&(a63="(-Inf,0.0758755)")&(a4="(-Inf,6.8E-5)")&(a56="(-Inf,9.5E-6)")&(a23="(-Inf,4.2E-5)")&(a35="(-Inf,1.29E-4)")&(a84="(-Inf,0.913131)")=>(decision={T[3]})
(a63="(-Inf,0.0758755)")&(a4="(-Inf,6.8E-5)")&(a35="(1.29E-4,Inf)")&(a84="(0.913131,Inf)")&(a23="(4.2E-5,Inf)")=>(decision={T[3]})
(a56="(9.5E-6,Inf)")&(a35="(1.29E-4,Inf)")&(a23="(4.2E-5,Inf)")&(a53="(1.65E-5,Inf)")&(a63="(-Inf,0.0758755)")&(a4="(-Inf,6.8E-5)")&(a84="(-Inf,0.913131)")=>(decision={F[2]})
(a56="(9.5E-6,Inf)")&(a35="(1.29E-4,Inf)")&(a63="(0.0758755,Inf)")&(a4="(3.575E-4,Inf)")&(a23="(4.2E-5,Inf)")&(a53="(-Inf,1.65E-5)")=>(decision={F[2]})
(a56="(9.5E-6,Inf)")&(a35="(1.29E-4,Inf)")&(a63="(0.0758755,Inf)")&(a84="(-Inf,0.913131)")&(a4="(3.575E-4,Inf)")=>(decision={F[2]})
(a53="(1.65E-5,Inf)")&(a35="(1.29E-4,Inf)")&(a56="(9.5E-6,Inf)")&(a23="(4.2E-5,Inf)")&(a63="(-Inf,0.0758755)")&(a4="(6.8E-5,3.575E-4)")&(a84="(-Inf,0.913131)")=>(decision={T[2]})
(a23="(-Inf,4.2E-5)")&(a4="(-Inf,6.8E-5)")&(a35="(1.29E-4,Inf)")&(a63="(-Inf,0.0758755)")&(a53="(-Inf,1.65E-5)")=>(decision={T[2]})
(a53="(1.65E-5,Inf)")&(a23="(-Inf,4.2E-5)")&(a35="(1.29E-4,Inf)")&(a4="(-Inf,6.8E-5)")&(a56="(-Inf,9.5E-6)")=>(decision={T[1]})
(a53="(1.65E-5,Inf)")&(a4="(6.8E-5,3.575E-4)")&(a23="(-Inf,4.2E-5)")&(a35="(-Inf,1.29E-4)")&(a56="(-Inf,9.5E-6)")=>(decision={T[1]})

Fig. 3. Exemplary rule set

Table 3. T-test results

	Accuracy		
Pairs	Avg. Diff.	Std. Deviation	t-stat
R-NFDT/NFDT	−1.43	9.99	−0.45
R-NFDT/LEM2	7.43	11.07	2.12
R-NFDT/J48	8.33	14.85	1.77
NFDT/LEM2	8.86	9.21	3.04
NFDT/J48	9.76	16.83	1.83
LEM2/J48	0.90	9.31	0.31
FRCT/R-NFDT	4.29	10.76	1.26
FRCT/NFDT	2.86	7.68	1.18
FRCT/LEM2	11.71	9.01	4.11
FRCT/J48	12.61	16.41	2.43
FRCT/SVM-NL	5.71	9.48	1.91
SVM-NL/NFDT	−2.86	11.33	−0.80
SVM-NL/R-NFDT	−1.43	11.65	−0.39
SVM-NL/LEM2	6.00	12.93	1.47
SVM-NL/J48	6.90	17.03	1.28
SVM-L/NFDT	−5.86	13.45	−1.38
SVM-L/R-NFDT	−4.43	8.32	−1.68
SVM-L/LEM2	3.00	12.68	0.75
SVM-L/J48	3.90	14.58	0.85

using the well-known t-test. This is done by formulating the hypothesis that the mean difference in accuracy between any two classification learning algorithms is zero. Table 3 gives the t-test results.

Let μ_d denote the mean difference in accuracy during a 10-fold classification of software defect data. Let H0 denote the hypothesis to be tested (i.e., $H0 : \mu_d = 0$). This is our null hypothesis. The paired difference t-test is used to test this hypothesis and its alternative hypothesis ($HA : \mu_d \neq 0$). Let $\overline{d}$, S_d^2 denote the mean difference and variance in the error rates of a random sample of size n from a normal distribution N(μ_d, σ^2), where μ_d and σ^2 are both unknown. The t statistic used to test the null hypothesis is as follows:

$$t = \frac{\bar{d} - \mu_d}{S_d/\sqrt{n}} = \frac{\bar{d} - 0}{S_d/\sqrt{n}} = \frac{\bar{d}\sqrt{n}}{S_d}$$

Table 4. Null-hypothesis results for accuracy

Accept H0 ($u_d = 0$) if $\|t\ value\| < 2.262$		
Pairs	t-stat(Acc.)	Acc/Rej H0
R-NFDT/NFDT	−0.45	*Accept*
R-NFDT/LEM2	2.12	*Accept*
R-NFDT/J48	1.77	*Accept*
NFDT/LEM2	3.04	**Reject**
NFDT/J48	1.83	*Accept*
LEM2/J48	0.31	*Accept*
FRCT/R-NFDT	1.26	*Accept*
FRCT/NFDT	1.18	*Accept*
FRCT/LEM2	4.11	**Reject**
FRCT/J48	2.43	**Reject**
FRCT/SVM-NL	1.91	*Accept*
SVM-NL/NFDT	−0.80	*Accept*
SVM-NL/R-NFDT	−0.39	*Accept*
SVM-NL/LEM2	1.47	*Accept*
SVM-NL/J48	1.28	*Accept*
SVM-L/NFDT	−1.38	*Accept*
SVM-L/R-NFDT	−1.68	*Accept*
SVM-L/LEM2	0.75	*Accept*
SVM-L/J48	0.85	*Accept*

where t has a student's t-distribution with n-1 degrees of freedom [11]. In our case, $n - 1 = 9$ relative to 10 sample error rates. The significance level α of the test of the null hypothesis H0 is the probability of rejecting H0 when H0 is true (called a Type I error). Let t_{n-1}, $\alpha/2$ denote a t-value to right of which lies $\alpha/2$ of the area under the curve of the t-distribution that has n-1 degrees of freedom. Next, formulate the following decision rule with $\alpha/2 = 0.025$:

Decision Rule: Reject $H0 : \mu_d = 0$ at significance level α if, and only if $|t - value| > 2.262$
Pr-values for t_{n-1}, $\alpha/2$ can be obtained from a standard t-distribution table. It should be noted that we repeated the experiments 30 times and averages have

remained consistent. However, for the purposes of analysis, we have restricted the reporting to 10 experiments.

4.2 Analysis

In terms of the t-test for accuracy, in general the three hybrid methods (FRCT, R-NFDT and NFDT) and SVM methods are comparable in that there is no significant difference in any of the methods based on the null hypothesis. In contrast, there is a $difference$ in accuracy between three pairs of methods outlined above(FRCT and LEM2, FRCT and J48 and NFDT and LEM2). This result corroborates our earlier result reported in [21]. Also of note is that SVM based methods have not had any effect in terms of classification accuracy. In terms of rules, in this chapter, we have only reported average number of rules over 10 runs (see Table 2). It is clear that the hybrid R-NFDT classifier has the smallest rule set.

The other important observation is the role that *reducts* play in defect data classification. On an average, only 6 attributes out of 95 were used by LEM2 in its rules with no significant reduction in classification accuracy. The R-NFDT and NFDT method uses an average of 4 out of 95 attributes resulting in a minimal number of rules with comparable accuracy.

The metrics at the method level that are most significant for R-NFDT and NFDT classifiers include: i) Halstead's metric of *essential complexity* which is a measure of the degree to which a module contains unstructured constructs, ii) Halstead's metric of *level* which is the level at which the program can be understood, iii) Halstead's metric of *number of unique operands* which includes variables and identifiers, constants (numeric literal or string) function names when used during calls iv) total lines of code v) *number of unique operators* is the number of branches for each module. These metrics were the most frequently occurring attributes in the rule set that contribute to the highest classification accuracy.

The metrics at the class-level that are most significant for R-NFDT, NFDT and LEM2 classifiers include: Depth of Inheritance Tree (DIT), Coupling Between Object Classes (CBO) and Lack of Cohesion of Methods (LCOM).

5 Conclusion

This chapter has presented a combination of hybrid and native methods based on rough sets, fuzzy sets, neural networks and support vector machines to classification of software defect data. The t-test shows that there is no significant difference between any of the hybrid methods in terms of accuracy at the 95% confidence level. However, in terms of rules, there is a difference between these methods. The experiments were aimed at not only comparing classification accuracy, but also collecting other useful software quality indicators such as number of rules, number of attributes (metrics) and the type of metrics (design vs. code level). In conclusion, the R-NFDT classifier seems to be the most desired classifier in terms of

comparable accuracy, average number of attributes used and smallest rule set. The Rough-NFDT method consists of generating reducts (reduced set of attributes) from rough set theory and then using the data from the reduced set of attributes to run the NFDT algorithm. The desired metrics (attributes) are: COUPLING-BETWEEN-OBJECTS, DEPTH, LACK-OF-COHESION-OF-METHODS max NUM-OPERATORS, max. NUM-UNIQUE-OPERANDS, max. HALSTEAD-LEVEL and min. LOC-TOTAL.

Acknowledgments

The research of Sheela Ramanna and James F. Peters is supported by NSERC Canada grant 194376 and 185986.

References

1. Bhatt, R.B., Gopal, M.: Neuro-fuzzy decision trees. International Journal of Neural Systems 16(1), 63–78 (2006)
2. Bhatt, R.B.: Fuzzy-Rough Approach to Pattern Classification- Hybrid Algorithms and Optimization, Ph.D. Dissertation, Electrical Engineering Department, Indian Institute of Technology Delhi, India (2006)
3. Bhatt, R.B., Gopal, M.: On the Extension of Functional Dependency Degree from Crisp to Fuzzy Partitions. Pattern Recognition Letters 27(5), 487–491 (2006)
4. Bhatt, R.B., Gopal, M.: Induction of Weighted and Interpretable Fuzzy Classification Rules for Medical Informatics. International Journal of Systemics, Cybernetics, and Informatics 3(1), 20–26 (2006)
5. Bezdek, J.C.: Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum, New York (1981)
6. Burges, C.J.: A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery 2(2), 955–974 (1998)
7. Chidamber, S.R., Kemerer, F.C.: A metrics suite for object-oriented design. IEEE Trans. Soft. Eng. 20(6), 476–493 (1994)
8. Dick, S., Meeks, A., Last, M., Bunke, H., Kandel, A.: Data mining in software metrics databases. Fuzzy Sets and Systems 145, 81–110 (2004)
9. Dubois, D., Prade, H.: Rough Fuzzy Sets and Fuzzy Rough Sets. Internation Journal of General Systems 17(2-3), 191–209 (1990)
10. Halstead, M.H.: Elements of Software Science. Elsevier, New York (1977)
11. Hogg, R.V., Tanis, E.A.: Probability and Statistical Inference. Macmillan Publishing Co., Inc., New York (1977)
12. Hsu, C.W., Lin, C.J.: A Comparison on methods for multi-class support vector methods, Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan (2001)
13. Khoshgoftaar, T.M., Allen, E.B.: Neural networks for software quality prediction. In: Pedrycz, W., Peters, J.F. (eds.) Computational Intelligence in Software Engineering, pp. 33–63. World Scientific, River Edge (1998)
14. MATLAB User's Guide. The Mathworks, Inc., Natick, MA 01760 (1994-2007)
15. McCabe, T.: A complexity measure. IEEE Trans. on Software Engineering SE 2(4), 308–320 (1976)

16. Nguyen, H.S., Nguyen, S.H.: Discretization methods in data mining. In: Polkowski, L., Skowron, A. (eds.) Rough Sets in Knowledge Discovery, vol. 1, pp. 451–482. Physica-Verlag, Berlin (1998a)
17. Pawlak, Z.: Rough sets. International J. Comp. Information Science 11(3), 341–356 (1982)
18. Peters, J.F., Ramanna, S.: Towards a software change classification system: A rough set approach. Software Quality Journal 11, 121–147 (2003)
19. Peters, J.F., Pedrycz, W.: Software Engineering: An Engineering Approach. John Wiley and Sons, New York (2000)
20. Ramanna, S., Bhatt, R., Biernot, P.: A rough-hybrid approach to software defect classification. In: An, A., Stefanowski, J., Ramanna, S., Butz, C.J., Pedrycz, W., Wang, G. (eds.) RSFDGrC 2007. LNCS, vol. 4482, pp. 79–86. Springer, Heidelberg (2007)
21. Ramanna, S., Bhatt, R.: Software defect classification: A comparative study with rough hybrid approaches. In: Kryszkiewicz, M., Peters, J.F., Rybinski, H., Skowron, A. (eds.) RSEISP 2007. LNCS, vol. 4585, pp. 630–638. Springer, Heidelberg (2007)
22. Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier methodology. IEEE Transactions on Systems, Man and Cybernetics 21(3), 660–674 (1991)
23. Quinlan, J.R.: Induction of decision trees. Machine Learning 1(1), 81–106 (1986)
24. Quinlan, J.R.: Decision trees and decision making. IEEE Transactions on Systems, Man and Cybernetics 20(2), 339–346 (1990)
25. Quinlan, J.R.: C.4.5 Programs for machine learning. Morgan Kauffmann, San Mateo (1993)
26. Tsang, E.C.C., Yeung, D.S., Lee, J.W.T., Huang, D.M., Wang, X.Z.: Refinement of generated fuzzy production rules by using a fuzzy neural network. IEEE Trans. on Transactions on Systems, Man and Cybernetics 34(1), 409–418 (2004)
27. Gryzmala-Busse, J.W.: A New Version of the Rule Induction System LERS. Fundamenta Informaticae 31(1), 27–39 (1997)
28. Wang, X.Z., Yeung, D.S., Tsang, E.C.C.: A comparative study on heuristic algorithms for generating fuzzy decision trees. IEEE Trans. on Transactions on Systems, Man and Cybernetics 31, 215–226 (2001)
29. Yuan, X., Khoshgoftaar, T.M., Allen, E.B., Ganesan, K.: An application of fuzzy clustering to software quality prediction. In: Proc. 3rd IEEE Symp. on Application-Specific Software Engineering Technology, pp. 85–90 (2000)
30. Yuan, Y., Shaw, M.J.: Induction of fuzzy decision trees. Fuzzy Sets and Systems 69, 125–139 (1995)