Information Sciences 229 (2013) 40–57

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

# The design of polynomial function-based neural network predictors for detection of software defects

Byoung-Jun Park [a], Sung-Kwun Oh [b,*], Witold Pedrycz [c,d]

[a] BT Convergence Technology Department, IT Convergence Technology Research Laboratory, Electronics and Telecommunications Research Institute (ETRI), 138 Gajeongno, Yuseong-gu, Daejeon 305-700, South Korea
[b] Department of Electrical Engineering, The University of Suwon, Hwaseong-si, Gyeonggi-do, South Korea
[c] Department of Electrical & Computer Engineering, University of Alberta, Edmonton AB, Canada T6R 2G7
[d] Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland

ARTICLE INFO

*Article history:*
Received 7 March 2008
Received in revised form 29 October 2010
Accepted 17 January 2011
Available online 23 January 2011

*Keywords:*
Software defect
Neural networks
Pattern classification
Fuzzy clustering
Two-class discrimination
Imbalanced data

ABSTRACT

In this study, we introduce a design methodology of polynomial function-based Neural Network (pf-NN) classifiers (predictors). The essential design components include Fuzzy C-Means (FCM) regarded as a generic clustering algorithm and polynomials providing all required nonlinear capabilities of the model. The learning method uses a weighted cost function (objective function) while to analyze the performance of the system we engage a standard receiver operating characteristics (ROC) analysis. The proposed networks are used to detect software defects. From the conceptual standpoint, the classifier of this form can be expressed as a collection of "if-then" rules. Fuzzy clustering (Fuzzy C-Means, FCM) is aimed at the development of premise layer of the rules while the corresponding consequences of the rules are formed by some local polynomials. A detailed learning algorithm for the pf-NNs is presented with particular provisions made for dealing with imbalanced classes encountered quite commonly in software quality problems. The use of simple measures such as accuracy of classification becomes questionable. In the assessment of quality of classifiers, we confine ourselves to the use of the area under curve (AUC) in the receiver operating characteristics (ROCs) analysis. AUC comes as a sound classifier metric capturing a tradeoff between the high true positive rate (TP) and the low false positive rate (FP). The performance of the proposed classifier is contrasted with the results produced by some "standard" Radial Basis Function (RBF) neural networks.



## 1. Introduction

The construction of effective predictive model of software defect estimation is one of the key challenges of quantitative software engineering. A wide range of prediction models have been proposed on a basis of product metrics by applying different modeling techniques, cf. [2,5,7,13,22,29,32]. Most defect predictors have used size and complexity metrics when attempting to predict defects of software. The earliest study along this line is the one reported by Akiyama [2], which dealt with the system developed at Fujitsu, Japan. Software metrics have been used as quantitative means of assessing software development process as well as the quality of software products. Many researchers have studied the correlation between software design metrics and the likelihood of occurrence of software faults [6,11,15,18,30,33].

As the size and complexity of software systems increases, software industry is highly challenged to deliver high quality, reliable software on time and within budget. Although there is a diversity of definitions of software quality, it is widely

* Corresponding author. Tel.: +82 31 229 8162; fax: +82 31 220 2667.
*E-mail address:* ohsk@suwon.ac.kr (S.-K. Oh).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.01.026

accepted that a project with many defects lacks quality. Defects are commonly defined as deviations from specifications or expectations that might lead to future failures in operation [5,13]. Knowing the causes of possible defects as well as identifying general software management decisions and practices that may need attention since the beginning of a project could save money, time and work. The estimation of the potential fault-prone of software is an indicator of quality and can help planning, controlling and executing software development and maintenance activities [5].

A variety of statistical techniques are used in software quality modeling. Models often exploit statistical relationships between measures of quality and software metrics. However, relationships between static software metrics and quality factors are often complex and nonlinear, limiting the accuracy of conventional modeling approaches (such as e.g., linear regression models). An efficient method for software quality (defect) analysis is to learn from past mistakes and acquired experience, namely, to use machine learning approaches. Neural networks are adept at modeling nonlinear functional relationships that are difficult to model when exploiting other techniques, and thus, form an attractive alternative for software quality modeling. However, traditional machine learning approaches may be faced with a difficulty of building a classification model (classifier) with binary class imbalanced data in software quality engineering [14]. The class imbalanced dataset has a class that outnumbers greatly the other classes. One of the commonly encountered techniques to alleviate the problems associated with class imbalance is data sampling that can be accomplished through undersampling [20], oversampling [8] or both of them [21].

The goal of this study is to develop a design environment for polynomial-based neural network predictors (classifiers) for the prediction of defects in software engineering artifacts. Neural networks (NNs) have been widely used to deal with pattern classification problems. It has been shown that the NNs can be trained to approximate complex discriminant functions [24]. NN classifiers can deal with numerous multivariable nonlinear problems for which an accurate analytical solution is difficult to derive or does not exist [3]. It is found however, that the quality and effectiveness of NN classifiers depend on several essential parameters whose values are crucial to the accurate predictions of the properties being sought. The appropriate NN architecture, the number of hidden layers and the number of neurons in each hidden layer are the important design issues that can immediately affect the accuracy of the prediction. Unfortunately, there is no direct method to identify these factors as they need to be determined on an experimental basis [3]. In addition, it is difficult to understand and interpret the resulting NNs. These difficulties increase once the number of variables and the size of the networks start to increase [34,38].

To alleviate the problems highlighted above, we propose to consider polynomial function based Neural Network (pf-NN) classifiers exploiting a direct usage of Fuzzy C-Means clustering and involving polynomials in the description of relationships of the models. We also describe the use a suitable learning methodology based on the weighted cost function with intention to cope with imbalanced classes. The proposed classifier is expressed as a collection of "if-then" rules. In its design, the premise parts of the rules are constructed with the use of the FCM. In the consequence (conclusion) part of the network we consider the use of polynomials which serve as the corresponding local models. The aggregation involves mechanisms of fuzzy inference. According to the type of polynomial, pf-NNs can be categorized into linear function-based NNs (lf-NNs) and quadratic function-based NNs (qf-NNs). For this category of the neural architectures, we construct a detailed learning algorithm. Generally speaking, most learning algorithms lead to high classification accuracy under assumption that the distribution of classes is almost equal. However, defect problems in software engineering have large differences between classes, called a class imbalance problem. Given this, an evaluation of the performance of the classifier using only accuracy of classification becomes questionable and has to be replaced by a more suitable criterion. To cope with the problem, we use a weighted cost function for learning and the area under curve (AUC) in the receiver operating characteristics (ROCs) analysis. AUC is a classifier metric which offers an important tradeoff between the high true positive rate (TP) and the low false positive rate (FP) [16].

In order to demonstrate the usefulness of the proposed pf-NN in software engineering, we use two software engineering datasets (CM1 and DATATRIEVE) coming from the PROMISE repository of empirical software engineering data (http://promisedata.org/repository/). The performance of the proposed classifier is contrasted with the results produced by the radial basis function (RBF) NNs given that these networks are one of the most widely applied category of neural classifiers [34,35].

This paper is organized as follows. We outline the underlying general topology of polynomial function-based neural networks in Section 2. Learning method of pf-NNs is given in Section 3 where we elaborate on all necessary development issues. Section 4 covers the assessment of performance of the proposed predictors. Extensive experimental studies are presented in Section 5 while Section 6 offers some concluding comments.

## 2. Design of polynomial function-based neural network predictors for software defect detection

In what follows, we discuss a general topology of polynomial function-based neural networks (pf-NNs).

### 2.1. Architecture of polynomial function-based neural networks

Neural Networks (NNs) have been widely used to deal with pattern classification problems. The generic topology of NNs consists of three layers as shown in Fig. 1. One of most widely applied neural classifiers are RBF NNs [34,35]. RBF NNs exhibit

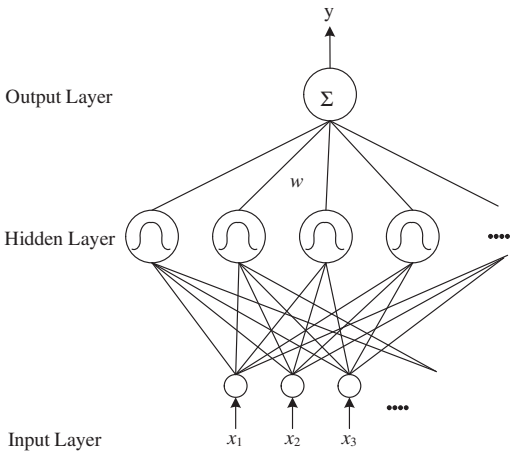


**Fig. 1.** General architecture of neural networks.

some advantages including global optimal approximation and classification capabilities, and rapid convergence of the learning procedures, cf. [12,19].

From the structural perspective, RBF NNs come with a single hidden layer. Each node in the hidden layer associates with a receptive field $\Theta(\mathbf{x})$ and generates a certain activation level depending on some incoming input **x.** The output $y(\mathbf{x})$ is a linear combination of the activation levels of the receptive fields:

$$y(\mathbf{x}) = \sum_{i=1}^{c} w_i \Theta_i(\mathbf{x}). \tag{1}$$

In the case of the Gaussian type of RBFs, we have

$$\Theta_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{v}_i\|^2}{2\sigma_i^2}\right). \tag{2}$$

Where $\mathbf{x}$ is the $n$-dimensional input vector $[x_1,\ldots x_n]^T$, $\mathbf{v}_i = [v_{i1},\ldots v_{in}]^T$ is the center of $i$th basis function $\Theta_i(\mathbf{x})$ and $c$ is the number of the nodes of the hidden layer. Typically the distance shown in (2) is the Euclidean one [36].

The proposed pf-NNs exhibit a similar topology as the one encountered in RBF NNs. However the functionality and the associated design process exhibit some evident differences. In particular, the receptive fields do not assume any explicit functional form (say, Gaussian, ellipsoidal, etc.), but are directly reflective of the nature of the data and come as the result of fuzzy clustering. Considering the prototypes $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c$ formed by the FCM method, the receptive fields are expressed in the following way

$$\Theta_i(\mathbf{x}) = A_i(\mathbf{x}) = \frac{1}{\sum_{j=1}^{c}\left(\frac{\|\mathbf{x}-\mathbf{v}_i\|^2}{\|\mathbf{x}-\mathbf{v}_j\|^2}\right)}. \tag{3}$$

In the addition, the weights between the output layer and the hidden layer are not constants but come in the form of polynomials of the input variables, namely

$$w_i = f_i(\mathbf{x}). \tag{4}$$

The neuron located at the output layer completes a linear combination of the activation levels of the corresponding receptive fields hence (1) can be rewritten as follows:

$$y(\mathbf{x}) = \sum_{i=1}^{c} f_i(\mathbf{x}) A_i(\mathbf{x}). \tag{5}$$

The above structure of the classifier can be represented through a collection of fuzzy rules

$$\text{If } \mathbf{x} \textit{ is } A_i \text{ then } f_i(\mathbf{x}), \tag{6}$$

where the fuzzy set $A_i$ is the $i$-cluster (membership function) of the $i$th fuzzy rule, $f_i(\mathbf{x})$ is a polynomial function.

As shown in Fig. 2 and describerd by (6), the proposed pf-NNs has three layers, say a premise layer (If), a consequence layer and an aggregation node in the output layer.

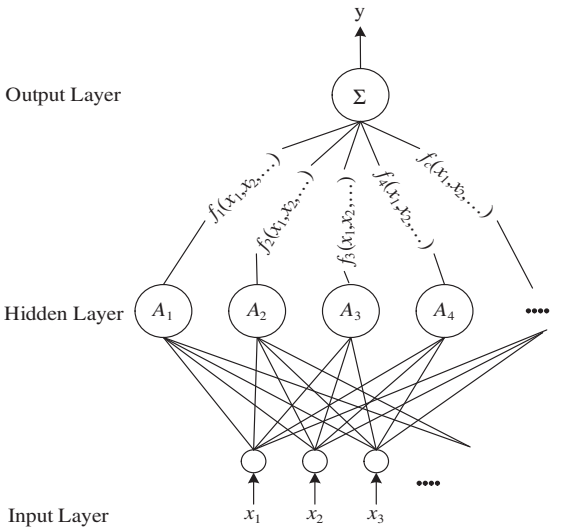


**Fig. 2.** Topology of polynomial function-based neural networks.

### 2.2. Three layers of polynomial function-based neural networks

The premise layer of pf-NNs is formed by means of the Fuzzy C-Means (FCM) clustering. In this section, we briefly review the objective function-based fuzzy clustering with the intent of highlighting it key features being useful here. The FCM algorithm comes as a standard mechanism aimed at the formation of '*c*' fuzzy sets (relations) in $\mathbf{R}^n$. The objective function *Q* guiding the clustering process is expressed as a sum of the distances of individual data from the prototypes $\mathbf{v}_1, \mathbf{v}_2, \ldots,$ and $\mathbf{v}_c$,

$$Q = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^{m} \|\mathbf{x}_k - \mathbf{v}_i\|^2. \tag{7}$$

Here, ||| denotes a certain distance function; '*m*' stands for a fuzzification factor (coefficient), $m > 1.0$. The commonly used value of "*m*" is equal to 2. *N* is the number of patterns (data). The resulting partition matrix is denoted by $U = [u_{ik}], i = 1,2,\ldots,c;\ k = 1,2,\ldots,N$. While there is a substantial diversity as far as distance functions are concerned, here we adhere to a weighted Euclidean distance taking on the following form

$$\|\mathbf{x}_k - \mathbf{v}_i\|^2 = \sum_{j=1}^{n} \frac{(x_{kj} - v_{ij})^2}{\sigma_j^2} \tag{8}$$

with $\sigma_j$ being a standard deviation of the *j*th variable. While not being computationally demanding, this type of distance is still quite flexible and commonly used.

The minimization of *Q* is realized in successive iterations by adjusting both the prototypes and entries of the partition matrix, min $Q(U, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c)$. The corresponding formulas leading to the iterative algorithm read as follows:

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left( \frac{\|\mathbf{x}_k - \mathbf{v}_i\|}{\|\mathbf{x}_k - \mathbf{v}_j\|} \right)^{\frac{2}{m-1}}}, \quad 1 \leqslant k \leqslant N, 1 \leqslant i \leqslant c \tag{9}$$

and

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} u_{ik}^{m} \mathbf{x}_k}{\sum_{k=1}^{N} u_{ik}^{m}}, \quad 1 \leqslant i \leqslant c. \tag{10}$$

The properties of the optimization algorithm are well documented in the literature, cf. [1,4]. In the context of our investigations, we note that the resulting partition matrix is a clear realization of '*c*' fuzzy relations with the membership functions $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_c$ forming the corresponding rows of the partition matrix *U*, that is $U = [\mathbf{u}_1^T \mathbf{u}_2^T \ldots \mathbf{u}_c^T]$.

Polynomial functions are dealt with in the consequence layer. In the Fig. 2 and (6), $f_i(\mathbf{x})$ is represented as the polynomial of the following forms.

$$\text{Linear function-based NN;}\quad f_i(\mathbf{x}) = a_{i0} + \sum_{j=1}^{n} a_{ij}x_j, \tag{11}$$

$$\text{Quadratic function-based NN;}\quad f_i(\mathbf{x}) = a_{i0} + \sum_{j=1}^{n} a_{ij}x_j + \sum_{j=1}^{n}\sum_{k=j}^{n} a_{ijk}x_j x_k. \tag{12}$$

These functions are activated by partition matrix and leads to local regression models for the consequence layer in each linguistic rule. As shown in (11) and (12), the proposed pf-NNs are separated into linear function-based NN (lf-NN) and quadratic function-based NN (qf-NN). In case of the quadratic function, the dimensionality of the problem goes up for a larger number of inputs, namely, a large number of combinations of the pairs of the input variables. In anticipation of the computational difficulties, we confine ourselves to a reduced quadratic function

$$\text{Reducedqf}-\text{NN};\quad f_i(\mathbf{x}) = a_{i0} + \sum_{j=1}^{n} a_{ij}x_j + \sum_{k=1}^{n} a_{ijk}x_k^2. \tag{13}$$

Let us consider the pf-NNs structure by considering the fuzzy partition (formed by the FCM) as shown in Fig. 2. The family of fuzzy sets $A_i$ forms a partition (so that the sum of membership grades sum up to one at each point of the input space). The "$\sum$" neuron realizes the sum as governed by (5). The output of pf-NNs can be obtained through a general scheme of fuzzy inference based on a collection of "if-then" fuzzy rules [17,28]. More specifically, we obtain

$$y = g(\mathbf{x}) = \sum_{i=1}^{c} \frac{u_i f_i(\mathbf{x})}{\sum_{j=1}^{c} u_j} = \sum_{i=1}^{c} u_i f_i(\mathbf{x}) \tag{14}$$

where, $u_i = A_i(\mathbf{x})$ [4].$g(\mathbf{x})$ is a concise description of the pf-NNs regarded as some discriminant function.

Note that based on local presentations (polynomials), the global characteristics of the pf-NNs result through the composition of their local relationships on the aggregation layer of pf-NNs.

### 2.3. Polynomial function-based neural network predictor for software defects

In this section, we consider the use of pf-NNs as software defect predictors and show how their functionality gives rise to highly nonlinear classifier.

A defect predictor is a two-class classifier with software modules being fault free (class $\omega_+$) and faulty (class $\omega_-$). The discriminant function gives rise to the following classification rule

$$\text{Decide } \omega_+ \text{ if } g(\mathbf{x}) > 0;\ \text{otherwise decide } \omega_-. \tag{15}$$

The final output of networks, (14), is used as a discriminant function $g(\mathbf{x})$ and can be rewritten by a linear combination

$$g(\mathbf{x}) = \mathbf{a}^{\mathrm{T}}\mathbf{fx}, \tag{16}$$

where, $\mathbf{a}$ is a vector of coefficients of polynomial functions used in the consequence layer of the rules in (11)–(13). More specifically, we have

(i) lf-NNs; $\mathbf{a}^T = [a_{10},\ldots,a_{c0},a_{11},\ldots,a_{c1},\ldots,a_{cn}]$ $\quad \mathbf{fx} = [u_1,\ldots,u_c,u_1x_1,\ldots,u_cx_1,\ldots,u_cx_n]^T$

(ii) qf-NNs; $\mathbf{a}^T = [a_{10},\ldots,a_{c0},a_{11},\ldots,a_{c1},\ldots,a_{cn},\ldots,a_{cnn}]$ $\quad \mathbf{fx} = [u_1,\ldots,u_c,u_1x_1,\ldots,u_cx_1,\ldots,u_cx_n,\ldots,u_cx_nx_n]^T$

(iii) Reduced qf-NNs; $\mathbf{a}^T = [a_{10},\ldots,a_{c0},a_{11},\ldots,a_{c1},\ldots,a_{cn},\ldots,a_{cnn}]$ $\quad \mathbf{fx} = [u_1,\ldots,u_c,u_1x_1,\ldots,u_cx_1,\ldots,u_cx_n,\ldots,u_cx_n^2]^T$

For the discriminant function coming in the form of (16), a two-class classifier implements the decision rule expressed by (15). Namely, $\mathbf{x}$ is assigned to $\omega_+$ if the inner product $\mathbf{a}^T\mathbf{fx}$ is greater than zero and to $\omega_-$ otherwise. The equation $g(\mathbf{x}) = 0$ defines the (nonlinear) decision surface that separates the two classes of the software modules.

## 3. Learning of pf-NNs using weighted linear discriminant analysis

In order to identify the parameters of the consequence layer of the pf-NNs classifier, we take care of a learning algorithm using a weighted cost function analysis.

Consider that a set of $N$ patterns $\mathbf{fx}_1,\ldots,\mathbf{fx}_N$ is given and these are labeled as $\omega_+$ or $\omega_-$. We want to use these patterns to determine the coefficients $\mathbf{a}$ in a discriminant function (16). A pattern $\mathbf{x}_k$ is classified correctly if $\mathbf{a}^T\mathbf{fx}_k > 0$ and $\mathbf{fx}_k$ is labeled $\omega_+$ or if $\mathbf{a}^T\mathbf{fx}_k < 0$ and $\mathbf{fx}_k$ is labeled $\omega_-$. Here, these can be incorporated into the formulas to simplify the treatment of the two-class case, namely, the replacement of all patterns labeled $\omega_-$ by their minus. Thus we look for a coefficient vector, $\mathbf{a}$, such that $\mathbf{a}^T\mathbf{fx}_k > 0$ for all patterns. Seeking a vector making all of the inner products $\mathbf{a}^T\mathbf{fx}_k$ positive, $\mathbf{a}^T\mathbf{fx}_k = b_k$ can be considered cf. [9]. Here $b_k$ is some arbitrarily specified positive constant; generally one could assume that $b_k = 1$. Then the problem is to determine a coefficient vector $\mathbf{a}$ satisfying the system of linear equations

$$\mathbf{Xa} = \mathbf{b}, \tag{17}$$

where $\mathbf{X} = [\mathbf{fx}_1, \mathbf{fx}_2, \ldots, \mathbf{fx}_N]^T$ and $\mathbf{b} = [b_1, \ldots, b_N]^T$ is called a margin vector.

The coefficient vector $\mathbf{a}$ standing in the consequence layer of pf-NNs classifier can be determined by the standard least square method. Namely, the coefficient can be estimated by solving the optimization problem

$$\mathrm{Min}_a J(\mathbf{a}, N), \tag{18}$$

where the minimized performance index $J$ comes in the form

$$J(\mathbf{a}, N) = \|\mathbf{Xa} - \mathbf{b}\|^2 = \sum_{i=1}^{N} (\mathbf{a}^T \mathbf{fx}_i - b_i)^2. \tag{19}$$

The advantage of using the standard least squares estimator is that the classifier can be easily determined. However, the disadvantage is that a preferred classifier is not derived by optimizing the results of classification or the cost of classification (19) for imbalanced data sets. Note that parameter estimators that directly optimize classification performance are generally difficult to find due to the factors such as unknown probability function of the data distribution, or possibly non-differentiable objective functions [16].

A common scenario in learning realized in presence of imbalanced data sets is that the trained classifier tends to classify all examples to the major class [23]. In practice, it is often very important to have accurate classification for the minority class, e.g., in the application of abnormality detection. This issue is apparent in software defect analysis. A major problem in predicting software quality using the number of component defects as a direct metric is the highly skewed distribution of faults because the majority of components have no defects or very few defects [22]. In this case, standard least squares learning methods with equal weighting for all data points will produce unfavorable classification results for the minority class (defect), which has a much smaller influence than the majority class on the resulting parameter estimates.

In order to learn parameters which reflect upon the minority class, we use the weighed least squares learning method based on the following cost function that is sensitive to data sample's importance for classification [16,23].

$$J = \gamma_+ \sum_{\mathbf{x}_i \in \omega_+} e_i^2 + \gamma_- \sum_{\mathbf{x}_i \in \omega_-} e_i^2, \tag{20}$$

where, $e$ is classification error, and $\gamma_+$ and $\gamma_-$ are weights for $\omega_+$ and $\omega_-$, respectively. If $\gamma_+ = \gamma_- = 1$, then the cost function for parameter learning becomes the same as being used in (19).

In this paper, the weights for $\omega_+$ and $\omega_-$ are defined following the number of patterns in each class, viz.

$$\gamma_+ = \frac{N}{N_+}, \quad \gamma_- = \frac{N}{N_-}, \tag{21}$$

where, $N_+$ is the number of patterns of minority class $\omega_+$, and $N_-$ is the number of patterns belonging to $\omega_-$. We have $N = N_+ + N_-$. Following (21), the performance index (20) assigns more weight to data points positioned in the minority class and this helps to alleviate the potential problem of classifying all examples to the major class [16].

We introduce the following matrix form

$$\mathbf{\Gamma} = \begin{bmatrix} \gamma_1 & 0 & \cdots & 0 \\ 0 & \gamma_2 & & \\ & \ddots & & \vdots \\ \vdots & & \gamma_i & \\ & & & \ddots \; 0 \\ 0 & 0 & \cdots & \gamma_N \end{bmatrix}, \text{in which } \gamma_i = \begin{cases} \gamma_+ & \text{if } \mathbf{x}_i \in \omega_+, \\ \gamma_- & \text{if } \mathbf{x}_i \in \omega_-. \end{cases} \tag{22}$$

Then (20) can be expressed as follows:

$$J(\mathbf{a}, N, \mathbf{\Gamma}) = \mathbf{e}^T \mathbf{\Gamma} \mathbf{e} = [\mathbf{Xa} - \mathbf{b}]^T \mathbf{\Gamma} [\mathbf{Xa} - \mathbf{b}] = [\mathbf{\Gamma}^{1/2}\mathbf{Xa} - \mathbf{\Gamma}^{1/2}\mathbf{b}]^T [\mathbf{\Gamma}^{1/2}\mathbf{Xa} - \mathbf{\Gamma}^{1/2}\mathbf{b}] = [\tilde{\mathbf{X}}a - \tilde{\mathbf{b}}]^T [\tilde{\mathbf{X}}a - \tilde{\mathbf{b}}] = \left\|\tilde{\mathbf{X}}a - \tilde{\mathbf{b}}\right\|^2$$
$$= \sum_{i=1}^{N} (\mathbf{a}^T \mathbf{f}\tilde{\mathbf{x}}_i - \tilde{b}_i)^2. \tag{23}$$

Given this form of the performance index, its global minimum is produced by the least square method. This leads to the optimal solution in the well-known format

$$\mathbf{a} = (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \tilde{\mathbf{b}}, \tag{24}$$

where, $\tilde{\mathbf{X}} = [\mathbf{f}\tilde{\mathbf{x}}_1, \mathbf{f}\tilde{\mathbf{x}}_2, \ldots, \mathbf{f}\tilde{\mathbf{x}}_N]^T = [\sqrt{\gamma_1}\mathbf{fx}_1, \sqrt{\gamma_2}\mathbf{fx}_2, \ldots, \sqrt{\gamma_N}\mathbf{fx}_N]^T \tilde{\mathbf{b}} = \left[\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_N\right]^T = [\sqrt{\gamma_1}b_1, \sqrt{\gamma_2}b_2, \ldots, \sqrt{\gamma_N}b_N]^T$

As mentioned in the previous section, the use of the reduced quadratic function can be helpful in reducing the number of coefficients, which contributes to a substantial computing load when dealing with a large number of combinations of the pairs of the variables.

## 4. Assessing performance using the ROC plane and determining AUC

The performances of the proposed predictors of software defect are assessed by making use of the receiver operating characteristics (ROCs). Let us recall that the ROC analysis is a classical approach coming from signal detection theory [10,16,37]. Formally, a defect predictor gets for a signal that a software module is defect prone. Signal detection theory considers ROC curves as a method for assessing quality of different predictors [27].The ROC analysis has been widely used in medical diagnosis [37] and is receiving considerable attention in the machine learning research community [31]. It has been shown that the ROC approach is a powerful tool both for making practical choices and for drawing general conclusions. The cost of misclassification is described in [31], in which the ROC convex hull (ROCCH) method has been introduced to manage classifiers (via combining or voting). One of the performance metrics used in the ROC analysis is to maximize the area under curve (AUC) of the ROC, which is equivalent to the probability that for a pair of randomly drawn data samples from the positive and negative groups respectively, the classifier ranks the positive sample higher than the negative sample in terms of "being positive" [10,16,27,31,37].

Consider a two-class set $\omega = \{\omega_+, \omega_-\}$ consisting of $N$ $n$-dimensional patterns that belong to a two-class set. There are $N_+$ positive data samples which belong to $\omega_+$ and $N_-$ negative data patterns, $N = N_+ + N_-$. In this paper, the minority class, defect patterns, is referred to as the positive class $\omega_+$. Let a two-class classifier be formed using the data set. The performance of the classifier may be evaluated using the counts of patterns {A, B, C, D} collected in the confusion matrix shown in Table 1.

Clearly $N_+ = A + B$ and $N_-$=C + D. The true positive rate (TP) is the proportion of positive patterns that were correctly identified, as given by [16]. TP can be called as probability of detection (PD) or recall in software engineering [27].

$$\mathrm{TP} = \frac{\mathrm{A}}{\mathrm{A}+\mathrm{B}} = \frac{\mathrm{A}}{N_+}. \tag{25}$$

The false positive rate (FP) is the proportion of the negatives patterns that were incorrectly classified as positive. FP can be called as probability of a false alarm (PF).

$$\mathrm{FP} = \frac{\mathrm{C}}{\mathrm{C}+\mathrm{D}} = \frac{\mathrm{C}}{N_-}, \tag{26}$$

$$\mathrm{CA} = \frac{\mathrm{A}+\mathrm{D}}{\mathrm{A}+\mathrm{B}+\mathrm{C}+\mathrm{D}} = \frac{\mathrm{A}+\mathrm{D}}{N}. \tag{27}$$

Classification accuracy (CA) is defined as the percentage of true negatives and true positives. Maximizing CA is a commonly used performance metric, which is equivalent to minimizing the misclassification rate of the classifier.

Alternatively, a classifier can be mapped as a point in two-dimensional ROC plane with coordinates FP-TP, as shown in Fig. 3. The y-axis shows the true positive rate (TP) and the x-axis shows the false positive rate (FP) of a classifier. By definition, the ROC curve must pass through the points FP = TP = 0 and FP = TP = 1 (a predictor that never triggers never makes false alarms; a predictor that always triggers always generates false alarms).

Three interesting trajectories connect these points:

1. A straight line from (0,0) to (1,1) is of little interest since it offers no information; i.e., the probability of a predictor firing (Defect detection) is the same as it being silent (No defect detection).
2. Another trajectory is the negative curve that bends away from the ideal point. Elsewhere [26], we have found that if predictors negate their tests, the negative curve will transpose into a preferred curve.
3. The point (FP = 0, TP = 1) is the ideal position (called sweet spot) on ROC plane. This is where we recognize all errors and never make mistakes. Preferred curves bend up toward this ideal point.

In the ideal case, a predictor has a high probability of detecting a genuine fault (TP) and a very low probability of false alarm (FP). This ideal case is very rare.

The ROC analysis is commonly applied to the visualization of the model performance, decision analysis, and model combinations [31] with an extensive scope and numerous applications [10,37]. For instance, the AUC of 0.50 means that the diagnostic accuracy is equivalent to a pure random guess. The AUC of 1.00 states that the classier distinguishes class examples perfectly [16]. As the performance index of a classifier, the AUC of a ROC can be calculated as follows [16,31].

$$\mathrm{AUC} = \frac{1 + \mathrm{TP} - \mathrm{FP}}{2}. \tag{28}$$

**Table 1**
Confusion matrix of a classifier (counts of patterns).

| | Predicted positive (Defect detection) | Predicted negative (No defect detection) |
|---|---|---|
| Actual positive (Defect logs) | A | B |
| Actual negative (No defect logs) | C | D |

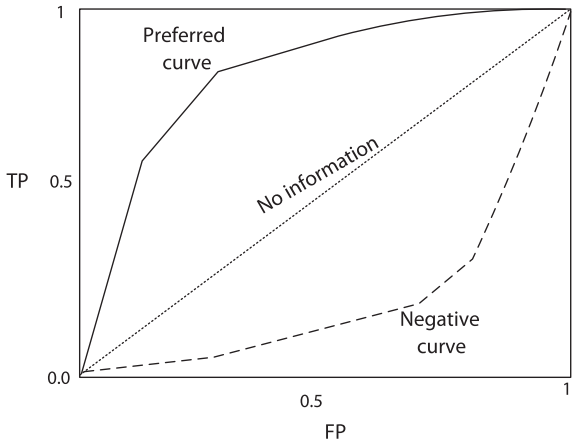


**Fig. 3.** Regions of a typical ROC curve.

Eq. (28) can be easily adjusted to cope with cost-sensitive classification if the misclassification costs are skewed [31], or other performance metrics derived from TP and FP are used. Clearly, the AUC of (28) is a classifier metric with a tradeoff between high TP and low FP. Note that if the data set is completely balanced with $N_+ = N_-$, then AUC = CA. However, for imbalanced data sets, this equivalence no longer holds. By separating the performance of a classifier into two terms that represent the performance for two classes, respectively, in comparison to the classification accuracy of (27) which only has a single term, this enables the possibility to manage the classification performance for imbalanced data.

For imbalanced datasets, there are several performance metrics widely used for balanced classification performance for two classes. All of them are based on values reported in the confusion matrix, see Table 1.

$$\mathrm{BAL} = 1 - \sqrt{\frac{(0-\mathrm{FP})^2 + (1-\mathrm{TP})^2}{2}}, \tag{29}$$

$$\mathrm{GM} = \sqrt{\mathrm{TP} \times (1-\mathrm{FP})}. \tag{30}$$

The index BAL [27] is based on Euclidean distance from the "sweet" spot FP = 0, TP = 1 to a pair of (FP, TP) and GM [16] is a performance metric as geometric mean for tradeoff between high TP and low FP.

## 5. Experimental studies

The experiments reported here involve synthetic two-dimensional data and CM1 and DATATRIEVE concerning defect problems coming from the PROMISE repository of empirical software engineering data (http://promisedata.org/repository/). Our objective is to quantify the performance of the proposed pf-NNs classifier (predictor) and compare it with the performance of some other classifiers reported in the literature. In the assessment of the performance of the classifiers, we use the AUC of the classifier.

For the software datasets, the experiments completed in this study are reported for the 10-fold cross-validation using a split of data into 70%–30% training and testing subsets, namely, 70% of the entire set of patterns is selected randomly for training and the remaining patterns are used for testing purposes.

### 5.1. Two-dimensional synthetic dataset

We start with a series of two-dimensional synthetic examples. Our primary objective is to illustrate the classification of the proposed pf-NNs classifier learned when using the weighted cost function (20) and to interpret the resulting partition space of inputs.

The collection of data involving 2 classes is shown in Fig. 4. The data set with two features x1 and x2 was generated following some Gaussian distribution, with the majority class ($\omega_-$) of mean vector $[0\ 0]^T$ and the covariance matrix as the identity matrix. The minority class ($\omega_+$) has a mean vector of $[2\ 2]^T$ and the same identity matrix as the covariance matrix. The synthetic dataset consists of 300 samples from $\omega_-$ and 30 patterns from $\omega_+$.

The classification results of the proposed pf-NN classifiers are shown in Tables 2 and 3. Table 2 shows the results of the pf-NNs learned with the use of (19) while the results of the pf-NNs trained on weighted cost function (20) are shown in Table 3. Table 2 shows the results with a high accuracy (96.1%) but a low probability of detection (66.7 or 63.3%). On the opposite end, Table 3 shows the results with lower accuracy 80.3 % and 78.8 %, but higher probability of detection 93.3 % and 96.7 % for lf-NN and qf-NN, respectively. Classification accuracy is a good measure of a learner's performance when the possible

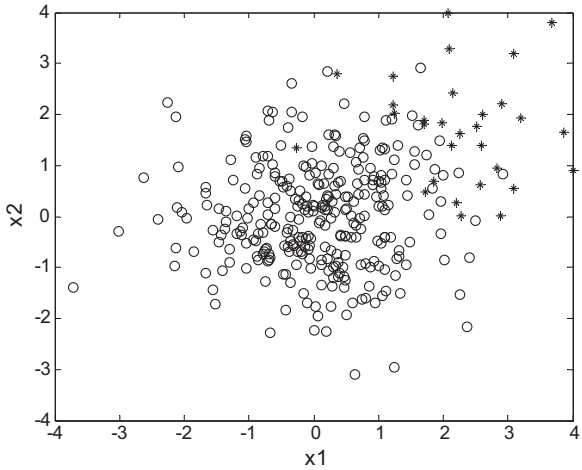


**Fig. 4.** Two-class synthetic datasets.

**Table 2**
Results of pf-NNs learned with the use of the standard cost function.

| Model | TP | FP | CA | AUC | BAL | GM |
|---|---|---|---|---|---|---|
| lf-NN | 0.667 | 0.01 | 0.961 | **0.828** | **0.764** | **0.812** |
| qf-NN | 0.633 | 0.007 | 0.961 | 0.813 | 0.741 | 0.793 |

**Table 3**
Results of pf-NNs learned with the use of the weighted cost function.

| Model | TP | FP | CA | AUC | BAL | GM |
|---|---|---|---|---|---|---|
| lf-NN | 0.933 | 0.21 | 0.803 | 0.862 | **0.844** | 0.859 |
| qf-NN | 0.967 | 0.23 | 0.788 | **0.868** | 0.836 | **0.863** |

outcomes occur with similar frequencies. However, the datasets used in this study have very uneven class distributions as shown in Fig. 4. Therefore, we should not be using the classification accuracy but consider AUC to be a sound vehicle to assess the performance of the classifiers. As shown in Tables 2 and 3, the learning method with the weighted cost function is better than those making use of the standard cost function in case of imbalanced data. In (20), $\gamma_{-} = 330/300 = 1.1$ and $\gamma_{+} = 330/30 = 10.1$ are used as the weight values in the minimized performance index.

For the pf-NN classifiers learned with the use of the standard and weighted objective functions, the boundaries of classification are shown in Figs. 5 and 6, respectively. When comparing Fig. 5 with Fig. 6, we note that the classification boundary

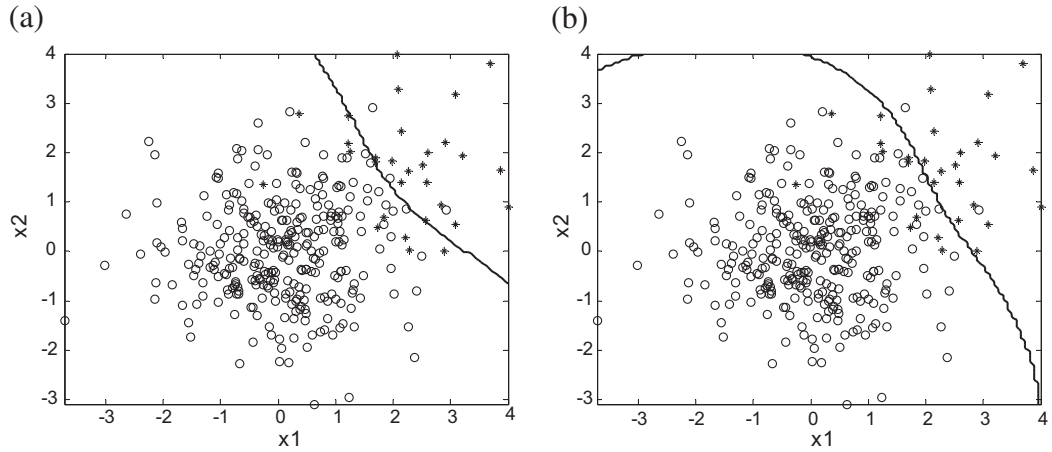


**Fig. 5.** Classification boundary of pf-NN classifiers learned with the use of the standard cost function: (a) lf-NN classifier and (b) qf-NN classifier.

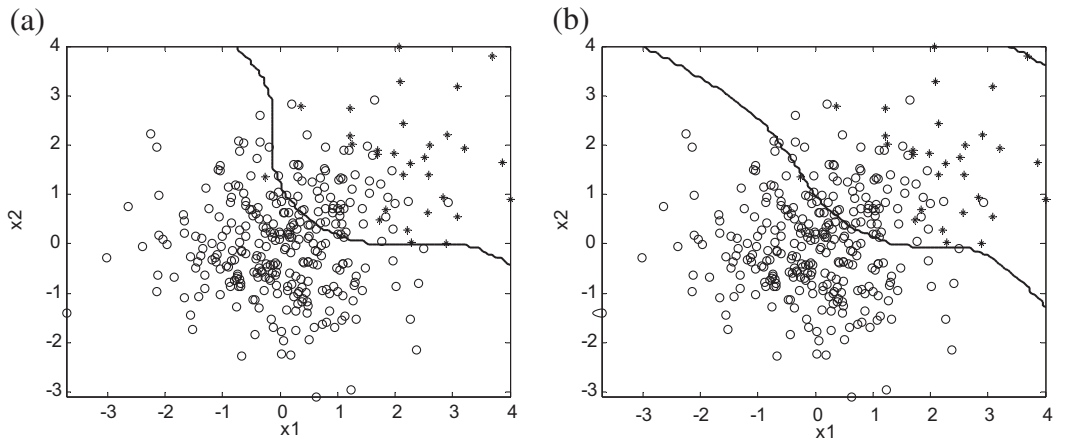


**Fig. 6.** Classification boundary of pf-NN classifiers learned with the use of the weighted cost function: (a) lf-NN classifier and (b) qf-NN classifier.

of the proposed pf-NN classifiers trained on the weighted objective function is reflective of the nature of the imbalanced dataset. For these results shown in Figs. 5 and 6, pf-NNs can be represented in the form of a series of rules. A partition $A_i$ of each rule on premise layer is illustrated by means of the contour shown in Fig. 7.

– For the standard cots function:

$$\begin{aligned} \text{lf}-\text{NN}: \quad & \text{If } \mathbf{x} \text{ is } A_1 \text{ then } y_1 = 0.75 + 0.069x_1 - 0.11x_2, \\ & \text{If } \mathbf{x} \text{ is } A_2 \text{ then } y_2 = 1.64 - 0.647x_1 - 0.35x_2, \end{aligned} \tag{31}$$

$$\begin{aligned} \text{qf}-\text{NN}: \quad & \text{If } \mathbf{x} \text{ is } A_1 \text{ then } y_1 = 0.91 + 0.11x_1 + 0.089x_2 + 0.22x_1^2 - 0.0339x_1x_2 + 0.17x_2^2, \\ & \text{If } \mathbf{x} \text{ is } A_2 \text{then } y_2 = 1.03 + 0.13x_1 + 0.235x_2 - 0.34x_1^2 - 0.0007x_1x_2 - 0.25x_2^2. \end{aligned} \tag{32}$$

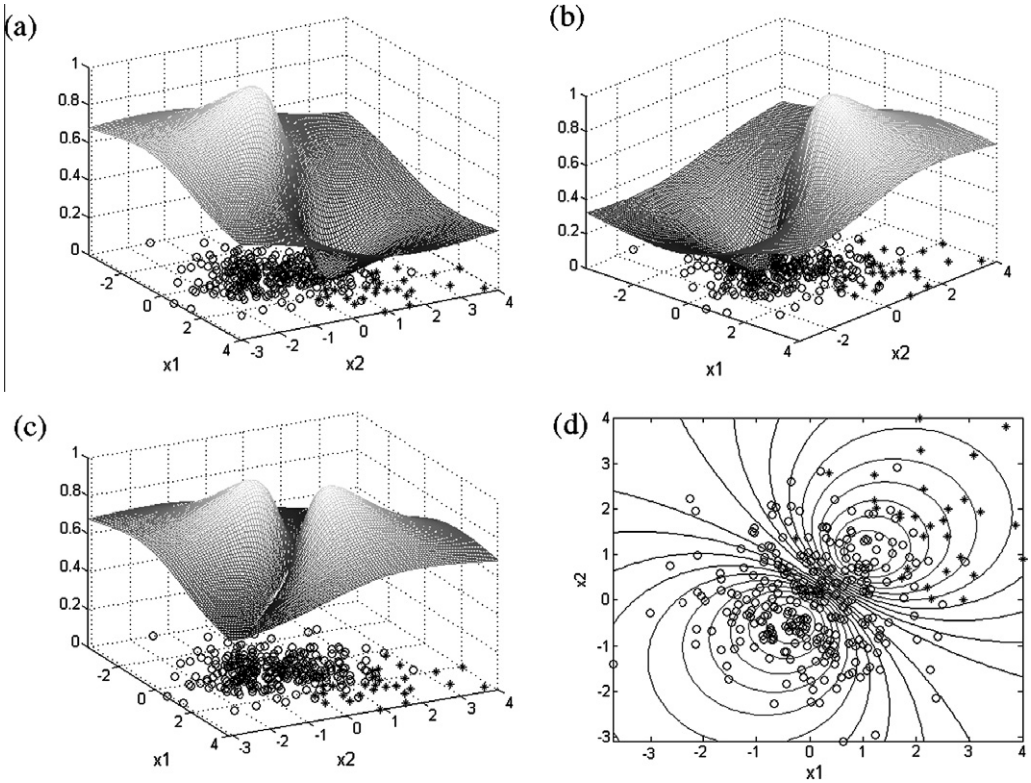


**Fig. 7.** Linguistic variables: (a) $A_1$, (b) $A_2$, (c) $A_1$ & $A_2$, and (d) contour of $A_1$ & $A_2$.

**Table 4**
Attribute information of CM1 dataset.

| Metric | Attribute | Description |
|---|---|---|
| loc | loc | McCabe's line count of code |
| | lOCode | Halstead's line count of code |
| | lOComment | Count of lines of comments |
| | lOBlank | Count of blank lines |
| | lOCodeAndCom | Line of code and comment |
| McCabe | v (g) | Cyclomatic complexity |
| | eV(g) | Essential complexity |
| | iv (g) | Design complexity |
| Basic Halstead | uniq_Op | Number of unique operators |
| | uniq_Opnd | Number of unique operands |
| | total_Op | Total number of operators |
| | total _Opnd | Total number of operands |
| Derived Halstead | *n* | Total number of operators and operands |
| | *v* | Volume |
| | *l* | Program length |
| | *d* | Difficulty |
| | *i* | Intelligence |
| | *e* | Effort |
| | *b* | Error estimator |
| | *t* | Time estimator |
| Other | branchCount | Branch count of the flow graph |

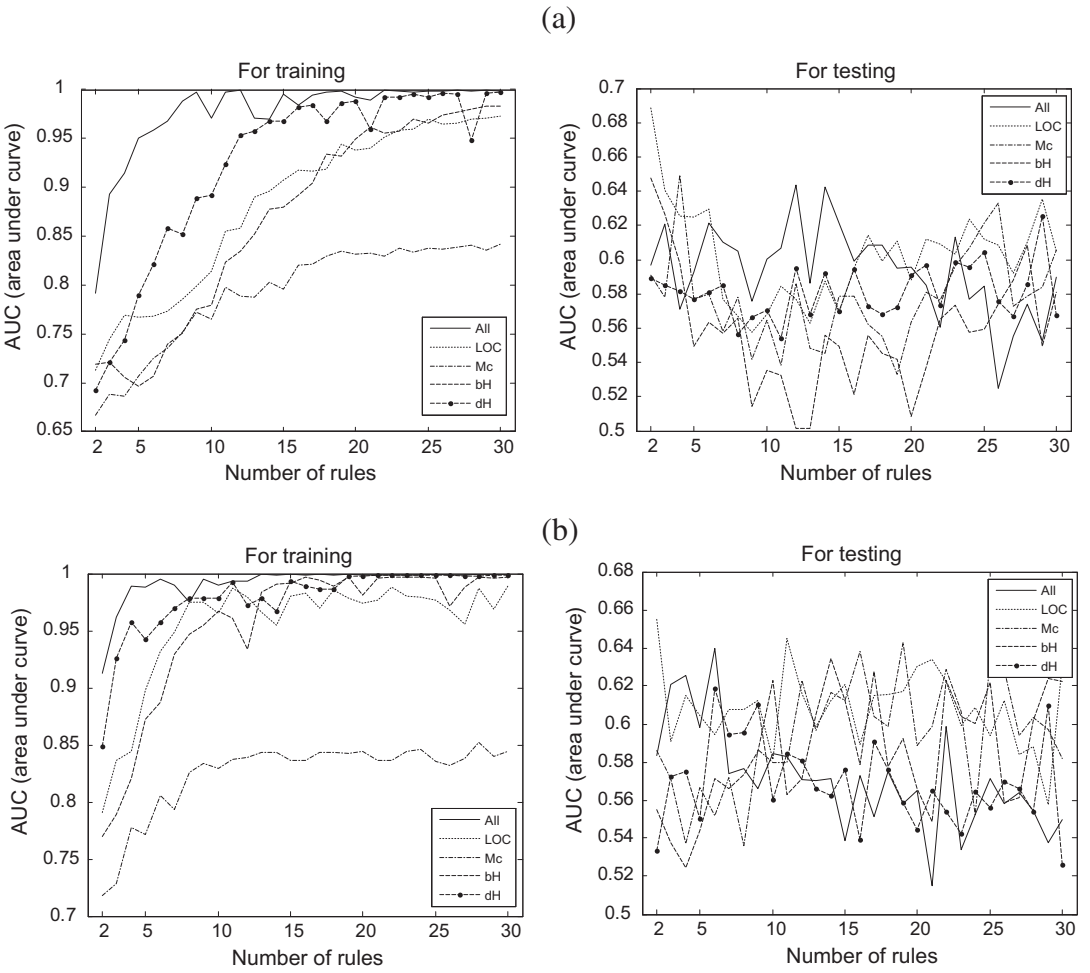


**Fig. 8.** Results of AUC of pf-NN predictors applied to subsets of the attributes: (a) AUC of lf-NN and (b) qf-NN predictors.

– For the weighted cots function:

$$\begin{aligned} \text{lf}-\text{NN}: \ & \text{If } \mathbf{x} \text{ is } A_1 \text{ then } y_1 = 0.75 - 0.187x_1 - 0.206x_2, \\ & \text{If } \mathbf{x} \text{ is } A_2 \text{ then } y_2 = -0.26 + 0.059x_1 + 0.066x_2, \end{aligned} \tag{33}$$

$$\begin{aligned} \text{qf}-\text{NN}: \ & \text{If } \mathbf{x} \text{ is } A_1 \text{ then } y_1 = 2.23 + 3.04x_1 + 5.27x_2 + 5.81x_1^2 + 0.074x_1x_2 + 6.35x_2^2, \\ & \text{If } \mathbf{x} \text{ is } A_2 \text{ then } y_2 = -12.9 + 12.1x_1 + 12.9x_2 - 5.79x_1^2 - 0.029x_1x_2 - 6.33x_2^2. \end{aligned} \tag{34}$$

### 5.2. CM1 dataset

CM1 dataset data comes from the PROMISE repository (http://promisedata.org/repository/). This dataset is a NASA spacecraft instrument written in C. The preprocessed data set has 21 attributes and one target attribute (defect or not), shown in Table 4 and included Halstead, McCabe, lines of code (loc), and other attributes. For the class distribution, the number of patterns is false 449 (90.16 %) and true (defects) 49 (9.84 %). The total number of patterns is 498. Halstead's theory which predicts the number of defects based on the language volume and McCabe's cyclomatic complexity which measures and controls the number of paths through a program are well known metrics in software defect prediction focused on establishing relationships between software complexity, usually measured in lines of code and defects [5,27]. Halstead estimates reading complexity by counting the number of operators and operands in a module; see the Basic Halstead attributes. Also, attributes related to the Halsted attributes were used to compute the eight Derived Halstead attributes shown in Table 4 [27]. Unlike Halstead, McCabe argued that the complexity of pathways between module symbols is more insightful than just a count of the symbols [27].

To generate defect predictors, we apply pf-NNs to CM1 dataset. Also, the pf-NNs are applied to subsets of the attributes containing just the loc (LOC), just McCabe (Mc), just basic Halstead (bH), just derived Halstead (dH) or all available attributes

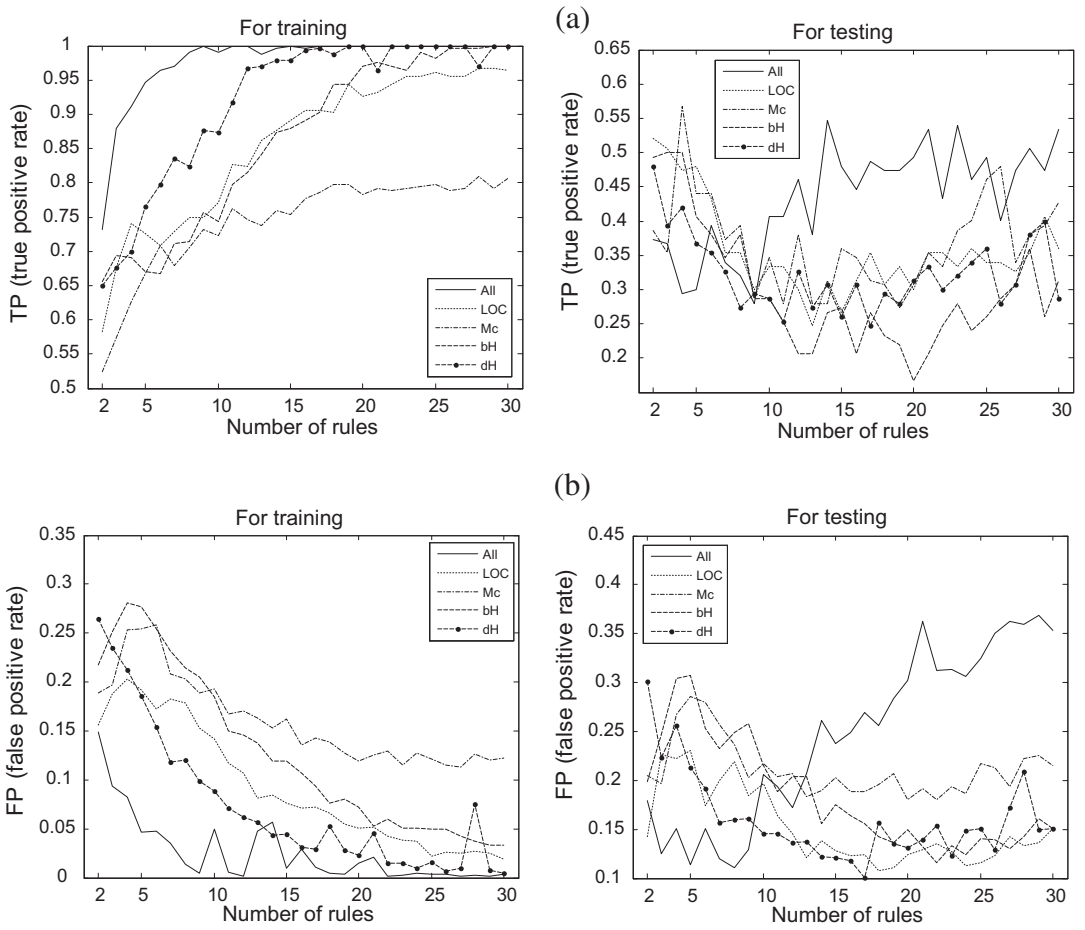


**Fig. 9.** Results of TP and FP of lf-NN; (a) TP and (b) FP.

(All). The experiments completed in this study are reported for the 10-fold cross-validation using a split of data into 70%-30% training and testing subsets. In this sense, the results pertain to the average and the standard deviation reported over the 10 experiments. Fig. 8 shows the results of AUC of pf-NN predictors according to number of rules. When results of training and testing are considered, the preferred number of rules is 10 to 15 rules in case of lf-NN and 5 to 10 rules in case of qf-NN. In addition, all available attributes or derived Halstead attribute are preferred as an input set of pf-NN. The results of TP and FP used to calculate AUC are shown in Figs. 9 and 10 for lf-NN and qf-NN, respectively. As shown in results, TP increase and FP

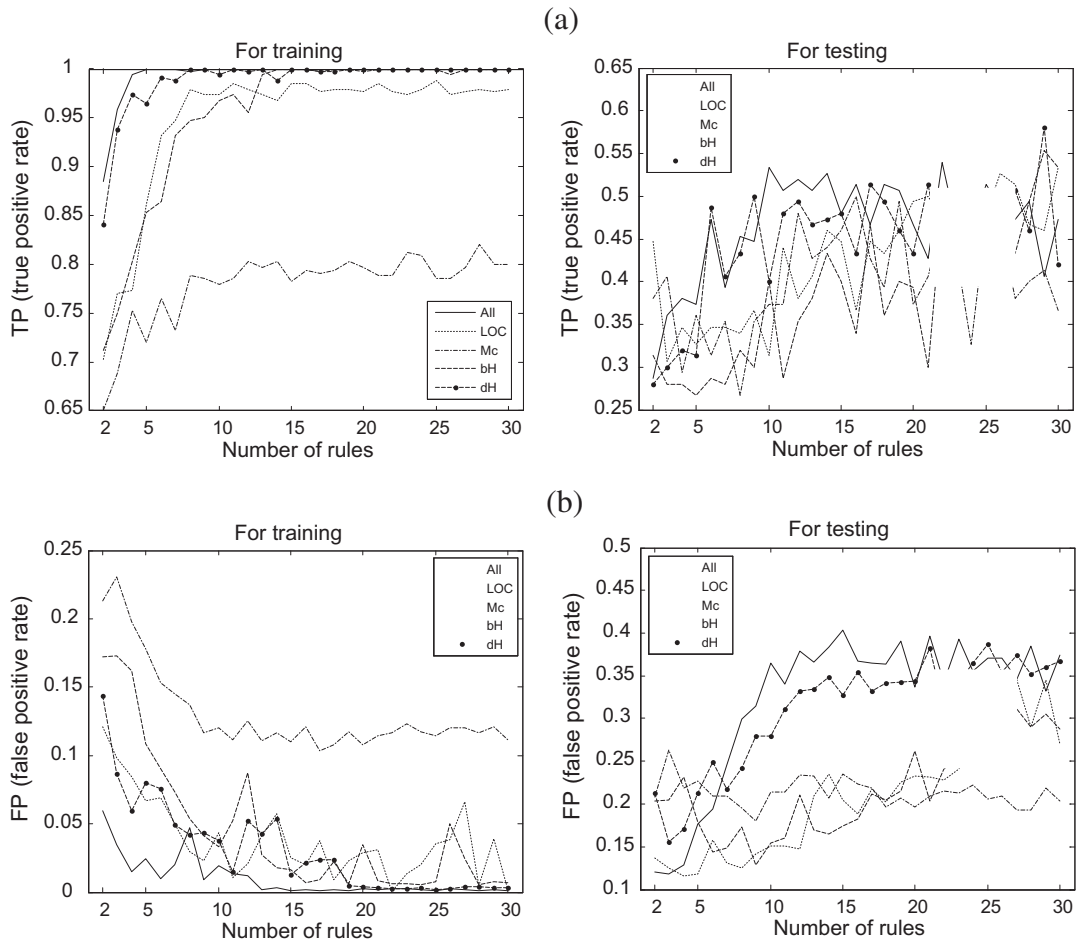


**Fig. 10.** Results of TP and FP of qf-NN; (a) TP and (b) FP.

**Table 5**
Results of comparative performance analysis.

| Methods | | TP | FP | CA | AUC | BAL | GM |
|---|---|---|---|---|---|---|---|
| lf-NN | TR | 1 ± 0 | 0.002 ± 0.002 | 0.998 ± 0.002 | 0.999 ± 0.001 | 0.999 ± 0.002 | 0.999 ± 0.001 |
| | TE | 0.46 ± 0.139 | 0.173 ± 0.023 | 0.791 ± 0.022 | 0.644 ± 0.067 | 0.598 ± 0.092 | 0.61 ± 0.095 |
| qf-NN | TR | 1 ± 0 | 0.01 ± 0.016 | 0.991 ± 0.015 | 0.995 ± 0.008 | 0.993 ± 0.011 | 0.995 ± 0.008 |
| | TE | 0.473 ± 0.127 | 0.194 ± 0.086 | 0.773 ± 0.08 | 0.64 ± 0.081 | 0.599 ± 0.092 | 0.612 ± 0.095 |
| RBF NN | TR | 0.756 ± 0.024 | 0.372 ± 0.018 | 0.64 ± 0.017 | 0.692 ± 0.017 | 0.685 ± 0.016 | 0.689 ± 0.017 |
| | TE | 0.793 ± 0.135 | 0.386 ± 0.072 | 0.632 ± 0.054 | 0.704 ± 0.041 | 0.676 ± 0.043 | 0.691 ± 0.043 |
| LSR | TR | 0.662 ± 0.044 | 0.144 ± 0.017 | 0.837 ± 0.016 | 0.759 ± 0.023 | 0.74 ± 0.029 | 0.752 ± 0.026 |
| | TE | 0.507 ± 0.126 | 0.162 ± 0.05 | 0.805 ± 0.051 | 0.672 ± 0.076 | 0.631 ± 0.089 | 0.647 ± 0.091 |
| MLP | TR | 0.018 ± 0.037 | 0.002 ± 0.003 | 0.902 ± 0.002 | 0.508 ± 0.017 | 0.305 ± 0.026 | 0.068 ± 0.119 |
| | TE | 0.02 ± 0.045 | 0.002 ± 0.004 | 0.9 ± 0.004 | 0.509 ± 0.022 | 0.307 ± 0.032 | 0.062 ± 0.133 |
| Blind Spot [25] | | 0.35 | 0.10 | 0.845 | 0.625 | 0.535 | 0.561 |
| Static Code [27] | | 0.71 | 0.27 | 0.729 | 0.72 | 0.72 | 0.72 |

decrease for training. To consider the testing results yields that 10 to 15 rules and 5 to 10 rules are preferred as the number of rules for the lf-NN and qf-NN, respectively. The values of the AUC, TP and FP shown in the figures refer to the mean of 10 results and the standard deviation as reported in Table 5.

In order to compare training results using the standard cost function (19) and the weighted cost function (20), Fig. 11 shows points of pairs of TP and FP on ROC plans for results of lf-NN with 10 to 15 rules and qf-NN with 5 to 10 rules on the derived Halstead. Here, we used $\gamma_- = 498/449 = 1.1091$ and $\gamma_+ = 498/49 = 10.163$ as the weighted values in the (20). As shown in the results, the points learned on the standard cost function are nearer to no information line than those on the weighted cost function.

Table 5 shows results of the proposed pf-NN and other methods applied to the weighted cost function. RBF NN (radial basis function neural network) is one of most widely applied neural classifiers [34,35] and LSR is linear standard regression

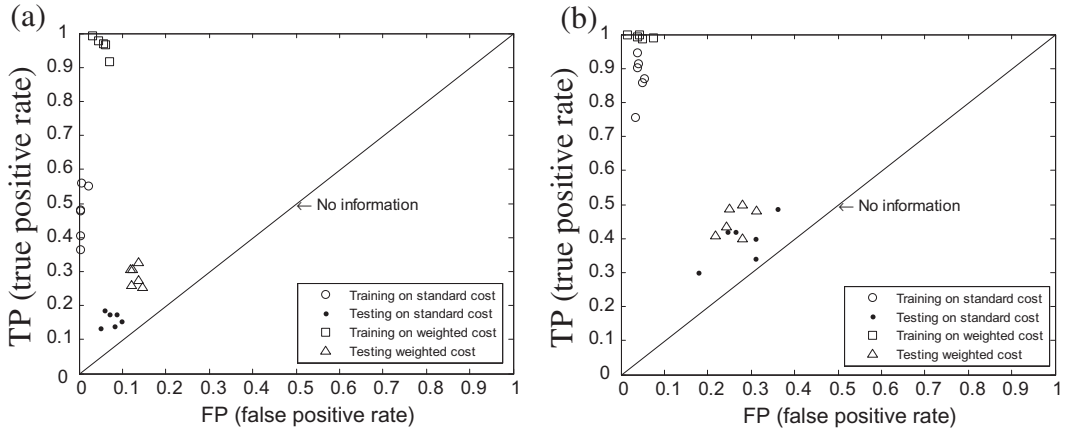


**Fig. 11.** Points of pairs of TP and FP on ROC plans: Results of (a) lf-NN and (b) qf-NN on the derived Halstead.

**Table 6**
Attribute information of DATATRIEVE dataset.

| No. | Attributes | Information |
|---|---|---|
| 1 | LOC6_0 | Number of lines of code of module m in version 6.0 |
| 2 | LOC6_1 | Number of lines of code of module m in version 6.1 |
| 3 | AddedLOC | Number of lines of code that were added to module m in version 6.1, i.e., they were not present in module m in version 6.0 |
| 4 | DeletedLOC | Number of lines of code that were deleted from module m in version 6.0, i.e., they were no longer present in module m in version 6.1 |
| 5 | DifferentBlocks | Number of different blocks module m in between versions 6.0 and 6.1 |
| 6 | ModificationRate | Rate of modification of module m, i.e., (AddedLOC + DeletedLOC)/ (LOC6.0 + AddedLOC) |
| 7 | ModuleKnowledge | Subjective variable that expresses the project team's knowledge on module m (low or high) |
| 8 | ReusedLOC | Number of lines of code of module m in version 6.0 reused in module m in version 6.1 |

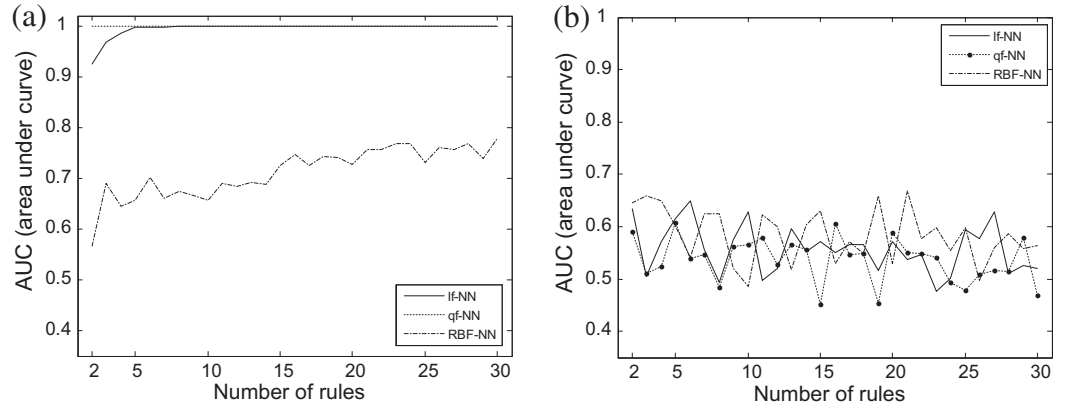


**Fig. 12.** AUC of pf-NN and RBF-NN: (a) training and (b) testing.

shown in [25]. The MLP (multi-layer perceptron) have been widely used to deal with pattern classification problem. For the comparison of the models, we consider their approximation and the generalization abilities. In Table 5, lf-NN has $0.999 \pm 0.001$ and $0.644 \pm 0.067$, qf-NN has $0.995 \pm 0.008$ and $0.64 \pm 0.081$ and RBF NN has $0.692 \pm 0.017$ and $0.704 \pm 0.041$ for the training (approximation) and testing (generalization), respectively. It seems that neither lf-NN nor qf-NN is better than RBF NN for the generalization of the model. However, when we consider the both the approximation and the generalization aspects, the proposed models are better than the previous models since the improvement of the approximation capabilities is higher than the deterioration of performance with regard to the generalization abilities themselves.

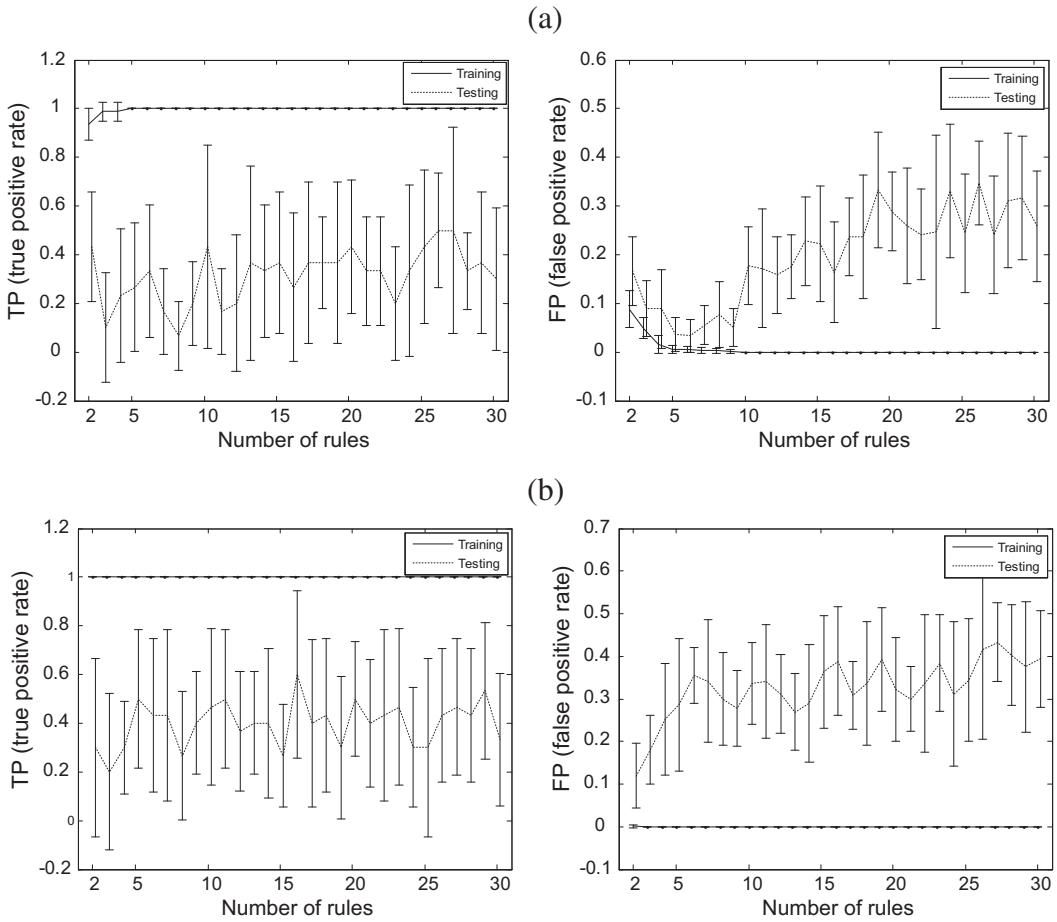


**Fig. 13.** TP and FP of pf-NN: (a) lf-NN and (b) qf-NN.

**Table 7**
Comparison of the assessing performance with other methods trained on the weighted cost function.

| Methods | | TP | FP | CA | AUC | BAL | GM |
|---|---|---|---|---|---|---|---|
| lf-NN | TR | 1 ± 0 | 0.005 ± 0.006 | 0.996 ± 0.006 | 0.998 ± 0.003 | 0.997 ± 0.004 | 0.998 ± 0.003 |
| | TE | 0.333 ± 0.272 | 0.033 ± 0.034 | 0.918 ± 0.036 | 0.65 ± 0.135 | 0.527 ± 0.192 | 0.467 ± 0.338 |
| qf-NN | TR | 1 ± 0 | 0 ± 0 | 1 ± 0 | 1 ± 0 | 1 ± 0 | 1 ± 0 |
| | TE | 0.5 ± 0.283 | 0.286 ± 0.156 | 0.697 ± 0.144 | 0.607 ± 0.158 | 0.565 ± 0.163 | 0.551 ± 0.238 |
| RBF NN | TR | 0.7 ± 0.105 | 0.187 ± 0.033 | 0.803 ± 0.033 | 0.757 ± 0.059 | 0.747 ± 0.065 | 0.753 ± 0.061 |
| | TE | 0.467 ± 0.358 | 0.131 ± 0.052 | 0.838 ± 0.048 | 0.668 ± 0.174 | 0.593 ± 0.221 | 0.551 ± 0.328 |
| LSR | TR | 0.75 ± 0.212 | 0.212 ± 0.1 | 0.785 ± 0.077 | 0.769 ± 0.074 | 0.734 ± 0.092 | 0.754 ± 0.087 |
| | TE | 0.4 ± 0.263 | 0.206 ± 0.111 | 0.764 ± 0.095 | 0.597 ± 0.117 | 0.536 ± 0.159 | 0.489 ± 0.271 |
| MLP | TR | 0.113 ± 0.314 | 0.1 ± 0.316 | 0.831 ± 0.261 | 0.506 ± 0.02 | 0.302 ± 0.028 | 0.035 ± 0.112 |
| | TE | 0.1 ± 0.316 | 0.103 ± 0.315 | 0.836 ± 0.267 | 0.499 ± 0.004 | 0.293 ± 0 | 0 ± 0 |

**Table 8**
Comparison of the assessing performance with other methods trained on the standard cost function.

| Methods | | TP | FP | CA | AUC | BAL | GM |
|---|---|---|---|---|---|---|---|
| lf-NN | TR | 0.6 ± 0.154 | 0.001 ± 0.004 | 0.964 ± 0.015 | 0.799 ± 0.077 | 0.717 ± 0.109 | 0.769 ± 0.099 |
| | TE | 0.467 ± 0.172 | 0.056 ± 0.045 | 0.908 ± 0.044 | 0.706 ± 0.089 | 0.62 ± 0.121 | 0.654 ± 0.121 |
| qf-NN | TR | 1 ± 0 | 0 ± 0 | 1 ± 0 | 1 ± 0 | 1 ± 0 | 1 ± 0 |
| | TE | 0.533 ± 0.281 | 0.283 ± 0.092 | 0.703 ± 0.083 | 0.625 ± 0.137 | 0.591 ± 0.155 | 0.575 ± 0.227 |
| RBF NN | TR | 0.025 ± 0.053 | 0.002 ± 0.005 | 0.912 ± 0 | 0.511 ± 0.024 | 0.311 ± 0.037 | 0.07 ± 0.148 |
| | TE | 0.033 ± 0.105 | 0 ± 0 | 0.926 ± 0.008 | 0.517 ± 0.053 | 0.316 ± 0.075 | 0.058 ± 0.183 |
| LSR | TR | 0.113 ± 0.109 | 0.001 ± 0.004 | 0.921 ± 0.007 | 0.556 ± 0.053 | 0.372 ± 0.077 | 0.273 ± 0.204 |
| | TE | 0 ± 0 | 0.011 ± 0.019 | 0.913 ± 0.018 | 0.494 ± 0.01 | 0.293 ± 0 | 0 ± 0 |
| MLP | TR | 0 ± 0 | 0 ± 0 | 0.912 ± 0 | 0.5 ± 0 | 0.293 ± 0 | 0 ± 0 |
| | TE | 0 ± 0 | 0 ± 0 | 0.923 ± 0 | 0.5 ± 0 | 0.293 ± 0 | 0 ± 0 |

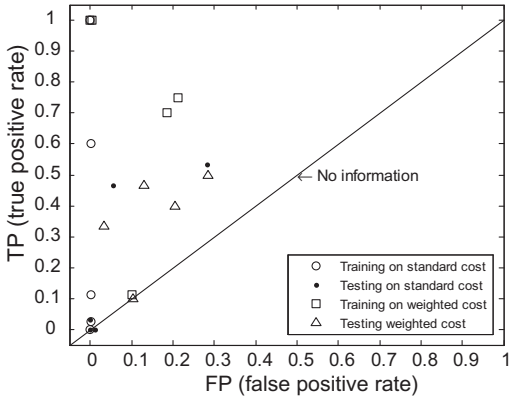


**Fig. 14.** ROC curve.

### 5.3. DATATRIEVE dataset

In this section, a pf-NN classifier is applied to the analysis of measurement data of a real-life maintenance project, the DATATRIEVE$^{TM}$ project carried out at Digital Engineering Italy. The DATATRIEVE$^{TM}$ product was originally developed in the BLISS language. BLISS is an expression language. It is block-structured, with exception handling facilities, co-routines, and a macro system. It was one of the first non-assembly languages for operating system implementation. Some parts were later added or rewritten in the C language. Therefore, the overall structure of DATATRIEVE$^{TM}$ is composed of C functions and BLISS subroutines [28]. DATATRIEVE dataset consists of 130 patterns with 8 condition attributes and 1 decision attribute shown in Table 6. Class distribution is 119 (91.54 %) and 11 (8.46 %) for 0 and 1. Its value is 0 for all those patterns in which no faults were found and its value is 1 for all other patterns. Therefore, $\gamma_{-} = 130/119 = 1.092$ and $\gamma_{+} = 130/11 = 11.818$ are used as the weighted values in (20).

Fig. 12 illustrates the AUC results obtained by lf-NN and qf-NN. Also, the AUCs of RBF NN trained on the weighted cost function are added for comparison with pf-NNs. As shown in this figure, lf-NN and qf-NN have bigger values of AUC than those of RBF NN for training; however, the similar trend of AUC is shown for testing. TP and FP of pf-NN are shown in Fig. 13. The bar denotes the standard deviation and the center value of a bar represents an average coming from the set of the 10 experiments. For the comparison of the results learned on the weighted and standard cost functions, Tables 7 and 8 are shown respectively. In case of the standard cost function, classification accuracy is higher but TP (probability of detection) is very low. The proposed lf-NN predictors show the better results than other methods. Fig. 14 illustrates relations between TP and FP (probability of false alarm) in Tables 7 and 8. As shown in Fig. 14, the points learned on the weighted cost function are far from the "no information" line. As the same results are shown in Section 5.2, the pf-NN models are preferred as a predictor in the detection of software defects given approximation and generalization aspects of the model.

## 6. Conclusions

In this paper, we proposed and design classifiers in the form of polynomial function-based neural networks (pf-NNs) for prediction of defects in software engineering. This type of classifiers is expressed in the form of "if-then" rules. The premise layer of the rules is developed with the use of fuzzy clustering. The consequence layer comes in the form of polynomials. The

proposed pf-NNs comes with two categories that is linear function-based NNs (lf-NNs) and quadratic function-based NNs (qf-NNs) where the taxonomy of the networks depends on the type of the polynomial in the conclusion part of the rule. The learning algorithm used to in the development of the consequence layer of the rules takes advantage of the linear discriminant analysis. In addition, the weighted cost function is used to handle imbalanced classes. The area under curve (AUC) in the receiver operating characteristics (ROC) analysis is used to evaluate the performance in presence of imbalanced datasets.

The pf-NNs were experimented with synthetic data and selected software engineering datasets. In this suite of experiments, we investigated and quantified the impact coming from the number of rules and the values of the AUC trained on the weighted cost function. In general, we have achieved substantially high values of TP (probability of detection) and low values of FP (probability of false alarm) than those for other methods and the standard cost function. Classifying software modules into faulty and non-faulty ones that is dealt with in this paper, is one of the first pursuits of software quality assurance. Further activities dealing with the localization and elimination of software defects could engage the use of other hybrid techniques of Computational Intelligence and machine learning.

## Acknowledgements

This research was supported by the Converging Research Center Program funded by the Ministry of Education, Science and Technology (No. 2011K000655) and supported by the GRRC program of Gyeonggi province [GRRC SUWON2012-B2, Center for U-city Security & Surveillance Technology] and also National Research Foundation of Korea Grant funded by the Korean Government (NRF-2012-003568).

## References

[1] A. Aiyer, K. Pyun, Y.Z. Huang, D.B. O'Brien, R.M. Gray, Lloyd clustering of Gauss mixture models for image compression and classification, Signal Processing: Image Communication 20 (2005) 459–485.
[2] F. Akiyama, An example of software system debugging, Information Processing 71 (1971) 353–379.
[3] Y. Al-Assaf, H. El Kadi, Fatigue life prediction of composite materials using polynomial classifiers and recurrent neural networks, Composite Structures 77 (2007) 561–569.
[4] J.C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum Press, New York, 1981.
[5] S. Bibi, G. Tsoumakas, I. Stamelos, I. Vlahavas, Regression via classification applied on software defect estimation, Expert Systems with Applications 34 (2008) 2091–2101.
[6] L.C. Briand, W.L. Melo, J. Wust, Assessing the applicability of fault-proneness models across object-oriented software projects, IEEE Transactions on Software Engineering 28 (2002) 706–720.
[7] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, Information Sciences 179 (8) (2009) 1040–1058.
[8] N.V. Chawla, L.O. Hall, K.W. Bowyer, W.P. Kegelmeyer, SMOTE: synthetic minority oversampling technique, Journal Artificial Intelleginee Research 16 (2002) 321–357.
[9] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, second ed., Wiley-Interscience, 2000.
[10] J.P. Egan, Signal Detection Theory and ROC Analysis, Academic, New York, 1975.
[11] K.E. Emam, W. Melo, C.M. Javam, The prediction of faulty classes using object-oriented design metrics, Journal of Systems and Software 56 (1) (2001) 63–75.
[12] M.J. Er, S.Q. Wu, J.W. Lu, H.L. Toh, Face recognition with radical basis function (RBF) neural networks, IEEE Transactions on Neural Networks 13 (5) (2002) 697–710.
[13] N.E. Fenton, M. Neil, A critique of software defect prediction models, IEEE Transactions on Software Engineering 25 (5) (1999) 675–689.
[14] A. Fernández, M. José del Jesus, F. Herrera, On the 2-tuples based genetic tuning performance for fuzzy rule based classification systems in imbalanced data-sets, Information Sciences 180 (8) (2010) 1268–1291.
[15] T.L. Graves, A.F. Karr, J.S. Marron, H. Siy, Predicting fault incidence using software change history, IEEE Transactions on Software Engineering 26 (7) (2000) 653–661.
[16] X. Hong, S. Chen, C.J. Harris, A kernel-based two-class classifier for imbalanced data sets, IEEE Transactions on Neural Networks 18 (1) (2007) 28–41.
[17] R.A. Hooshmand, M. Ataei, Real-coded genetic algorithm based design and analysis of an auto-tuning fuzzy logic PSS, Journal of Electrical Engineering and Technology 2 (2) (2007) 178–187.
[18] A. James, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, G. Succi, Identification of defect-prone classes in telecommunication software systems using design metrics, Information Sciences 176 (2006) 3711–3734.
[19] X.-Y. Jing, Y.-F. Yao, D. Zhang, J.-Y. Yang, M. Li, Face and palmprint pixel level fusion and Kernel DCV-RBF classifier for small sample biometric recognition, Pattern Recognition 40 (2007) 3209–3224.
[20] T.M. Khoshgoftaar, K. Gao, Feature selection with imbalanced data for software defect prediction, in: Proceeding of the 2009 International Conference on Machine Learning and Applications, 2009, pp. 235–240.
[21] T.M. Khoshgoftaar, J.V. Hulse, A. Napolitano, Supervised neural network modeling: an empirical investigation into learning from imbalanced data with labeling errors, IEEE Transactions on Neural Networks 21 (5) (2010) 813–830.
[22] F. Lanubile, G. Visaggio, Evaluating predictive quality models derived from software measures: lessons learned, Journal of Systems and Software 38 (3) (1997) 225–234.
[23] J. Leskovec, J. Shawe-Taylor, Linear programming boost for uneven datasets, in: Proceedings of International Conference on Machine Learning (ICML), 2003, pp. 456–463.
[24] R.P. Lippman, An introduction to computing with neural nets, IEEE ASSP Magazine 4 (2) (1981) 4–22.
[25] T. Menzies, J.S. Di Stefano, How good is your blind spot sampling policy?, Proceedings of IEEE International Symposium on High Assurance Systems Engineering (2004) 129–138
[26] T. Menzies, J.S. Di Stefano, M. Chapman, K. Mcgill, Metrics that Matter, in: Proceedings of 27th NASA SEL Workshop Software Engineering, 2002, pp. 51–57.
[27] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Transactions on Software Engineering 33 (1) (2007) 2–13.
[28] S. Morasca, G. Ruhe, A hybrid approach to analyze empirical software engineering data and its application to predict module fault-proneness in maintenance, The Journal of Systems and Software 53 (2000) 225–237.

[29] J.E. Munoz-Exposito, S. Garcia-Galan, N. Ruiz-Reyes, P. Vera-Candeas, Adaptive network-based fuzzy inference system vs. other classification algorithms for warped LPC-based speech/music discrimination, Engineering Applications of Artificial Intelligence 20 (2007) 783–793.
[30] S.-K. Oh, W. Pedrycz, B.-J. Park, Self-organizing neurofuzzy networks in modeling software data, Fuzzy Sets and Systems 145 (2004) 165–181.
[31] F. Provost, T. Fawcett, Robust classification for imprecise environments, Machine Learning 42 (2001) 203–231..
[32] T.-S. Quah, Estimating software readiness using predictive models, Information Science 179 (4) (2009) 430–445.
[33] T.-S. Quah, M.M.T. Thwin, Prediction of software development faults in PL/SQL files using neural network models, Information and Software Technology 46 (2004) 519–523.
[34] F. Ros, M. Pintore, J.R. Chretien, Automatic design of growing radial basis function neural networks based on neighborhood concepts, Chemometrics and Intelligent Laboratory Systems 87 (2007) 231–240.
[35] H. Sarimveis, P. Doganis, A. Alexandridis, A classification technique based on radial basis function neural networks, Advances in Engineering Software 37 (2006) 218–221.
[36] A. Staiano, R. Tagliaferri, W. Pedrycz, Improving RBF networks performance in regression tasks by means of a supervised fuzzy clustering, Neurocomputing 69 (2006) 1570–1581.
[37] J.A. Swets, Signal Detection Theory and ROC Analysis in Psychology and Diagnostics: Collected Papers, Lawrence Erlbaum Associates, Mahwah, NJ, 1996.
[38] C. Zhang, J. Jiang, M. Kamel, Intrusion detection using hierarchical neural networks, Pattern Recognition Letters 26 (2005) 779–791.