Chapter 2

Related Works

In this chapter, I summarize the literature about the rule extraction from the ANN and other associates.

2.1 Cluster Analysis

2.1.1 Chi2 Algorithm (Liu and Setiono, 1995)

Chi2 (Liu and Setiono, 1995) is an improved and automated version of Chi-Merge (Kerber, 1992). Instead of specifying a c^2 threshold, Chi2 wraps up ChiMerge with a loop that automatically increments the c^2 threshold. A consistency checking is also introduced as a stopping criterion in order to guarantee that the discretized data instance set accurately represents the original one. With these two new features, Chi2 automatically determines a proper c^2 threshold that keeps the fidelity of the original data.

At the end of Chi2, if an attribute is merged to only one value, it simply means that this attribute is not relevant in representing the original data instance set. As a result, when discretization ends, feature selection is accomplished.

Given a data instance set where each pattern is described by the values of the continuous attributes x_1, x_2, .., x_m and its class label is known. Chi2 finds discrete representations of the data instance set. With the c^2 statistic, Chi2 divides the range of the each attribute into subintervals and assigns all values that fall in a subinterval (a unique discrete value). The outline of the algorithm is as follows:

Table 2.1: Chi2 Algorithm.

1. Let Chi-0 be an initial critical value.
2. For each attribute x_i:

a. Sort the instances according the input values of attribute x_i.

b. Form an initial set of intervals such that each interval contains only one unique value.

3. Initialize all attributes as "unmarked".

4. For each unmarked attribute x_i:

a. For each adjoining pairs of subintervals, compute their c^2 values whose degree of the freedom is one less than the number of classes:

$$c^2 = \sum_{l=1}^{2} \sum_{j=1}^{k} \frac{(J_{ij} - E_{ij})^2}{E_{ij}} \tag{2.1}$$

where

k: The number of classes;

J_{lj}: The number of instances in the l-th interval of the adjoining pairs of subintervals of the j-th class;

R_l: The number of instances in the l-th interval and $R_l = \sum_{j=1}^{k} J_{lj}$;

C_j: The number of instances in the j-th class and $C_j = \sum_{l=1}^{2} J_{lj}$;

N: The total number of instances in the adjoining pairs of subintervals and $n = \sum_{i=1}^{2} R_l$;

E_{lj}: The expected frequency of J_{lj}, $E_{lj} = R_l C_j / N$, and E_{lj} is set to 0.1 if either R_l or C_j is 0.

b. Find two subintervals with the lowest c^2 value.

c. If this value is less than Chi-0 and it does not introduce the rate of conflicting data[1] (inconsistency rate) > d to merge the two subintervals, then

- Merge the subintervals and go to step 4.a.

Else if it is going to introduce the inconsistency rate > d, then

- Label attribute as "mark".

5. If there is still an unmarked attribute, then Chi-0 is increased and go to step 4.

1. Conflicting data occur when there are two or more instances from different classes with the same dicretized attribute values.

2.2 Rule Extraction Techniques

2.2.1 NeuroLinear (Setiono and Liu, 1997)

Setiono and Liu (1997) present NeuroLinear, a system for extracting oblique decision rules from a trained ANN for the classification problem. Each condition of an oblique decision rule corresponds to a partition of the **x**-space (the input space) by a hyper-plane that is not necessarily axis-parallel. Allowing a set of such hyper-planes to form the boundaries of the decision regions leads to a significant reduction in the number of rules generated while maintaining the accuracy rates of the ANN.

The steps of extracting oblique decision rules from an ANN are as follows:

1. Train and select an ANN to meet a pre-specified accuracy requirement. Remove the redundant connections in the network by pruning while maintaining the accuracy.
2. Discretize the hidden node activation values of the ANN. Extract rules that describe outputs of the ANN in terms of the discretized activation values. For each discretized hidden node activation value, generate a rule with respect to input attributes of the ANN. Merge the two sets of rules obtained above.

The details of these steps are listed below.

Neural Network Training and Pruning

The basic structure of the ANN is a standard three-layer feed-forward network, which consists of an input layer, a hidden layer, and an output layer. Two approaches to determine a suitable number of hidden nodes have been described in the literature. The first approach begins with a minimal number of hidden nodes, say one or two, and more hidden nodes are added as they are needed to increase the accuracy of the ANN. The second approach begins with an oversized network and removes redundant connections in the network by pruning. In the process, hidden nodes that are not connected to any input nodes or output nodes can be removed as well. Setiono and Liu

(1997) adopt the second approach since they are interested in removing input nodes that are irrelevant to the classification. During the pruning phase, irrelevant input nodes and hidden nodes can be identified and removed from the network.

Given a m-dimensional pattern ${}_c\mathbf{x}$, $c \in \{1, 2, ..., n\}$ as input. Let ${}_2w_{ji}$ be the weight of the connection from input node i, $i \in \{1, 2, ..., m\}$ to hidden node j, $j \in \{1, 2, ..., p\}$; ${}_3w_{jk}$ be the weight of the connection from hidden node j to output node k, $k \in \{1, 2, ..., q\}$. The k-th output of the ANN for pattern ${}_c\mathbf{x}$ is obtained by computing

$$ {}_c y_k = s\left(\sum_{j=1}^{p} {}_3w_{jk}\, {}_c h_j\right), \tag{2.2} $$

where

$$ s(t) = \text{sigmoid}(t) = 1/(1+e^{-t}), \tag{2.3} $$

$$ {}_c h_j = d\left(\sum_{i=1}^{m} {}_c x_{i\,2} w_{ji} + {}_2q_j\right), \tag{2.4} $$

$$ d(t) = \tanh(t) = (e^{t} - e^{-t})/(e^{t} + e^{-t}). \tag{2.5} $$

The desirous output for a pattern ${}_c\mathbf{x}$ that belongs to the k-th class is a q-dimensional vector ${}_c\mathbf{d}$, where ${}_cd_k = 1$ and others equal 0. The back-propagation algorithm is applied to update the weights (${}_2\mathbf{W}$, ${}_3\mathbf{W}$, ${}_2$q) and minimize the following function:

$$ \mathrm{E}({}_2\mathbf{W}, {}_3\mathbf{W}, {}_2\mathrm{q}) = \mathrm{F}({}_2\mathbf{W}, {}_3\mathbf{W}, {}_2\mathrm{q}) + \mathrm{P}({}_2\mathbf{W}, {}_3\mathbf{W}, {}_2\mathrm{q}), \tag{2.6} $$

where ${}_2\mathbf{W}$, ${}_3\mathbf{W}$ and ${}_2$q denote these parameter matrixes consisting of ${}_2w_{ji}$, ${}_3w_j$ and ${}_2q_j$ respectively; F(${}_2\mathbf{W}$, ${}_3\mathbf{W}$, ${}_2$q) is the cross entropy function (Van Ooyen and Nienhuis, 1992):

$$ \mathrm{F}({}_2\mathbf{W}, {}_3\mathbf{W}, {}_2\mathrm{q}) = -\sum_{c=1}^{n}\sum_{k=1}^{q}\left({}_cd_k \log {}_cy_k + (1-{}_cd_k)\log(1-{}_cy_k)\right) \tag{2.7} $$

and P(${}_2\mathbf{W}$, ${}_3\mathbf{W}$, ${}_2$q) is a penalty term used for weight decay (Hertz et al., 1991):

$$ \mathrm{P}({}_2\mathbf{W}, {}_3\mathbf{W}, {}_2\mathrm{q}) = e_1\left(\sum_{j=1}^{p}\frac{b({}_2q_j)^2}{1+b({}_2q_j)^2} + \sum_{i=1}^{m}\sum_{j=1}^{p}\frac{b({}_2w_{ji})^2}{1+b({}_2w_{ji})^2} + \sum_{j=1}^{p}\sum_{k=1}^{q}\frac{b({}_3w_{jk})^2}{1+b({}_3w_{jk})^2}\right) $$

$$+e_2\left(\sum_{j=1}^{p}({}_2q_j)^2+\sum_{i=1}^{m}\sum_{j=1}^{p}({}_2w_{ji})^2+\sum_{j=1}^{p}\sum_{k=1}^{q}({}_3w_{jk})^2\right) \quad (2.8)$$

where e_1, e_2, and β are positive decay parameters.

One of the advantages of having an ANN with only the relevant connections is that the behavior of the ANN can be explained by a simple set of rules (Karnin, 1990). The pruning algorithm removes connections in the ANN based on their magnitude. The details of this algorithm and the experimental results on a number of well-known classification problems are given in (Setiono, 1997). The effectiveness of the pruning algorithm is shown by the fact that for the many problems tested, the final pruned ANN has only the relevant connections left regardless of the number of initial hidden nodes in the ANN.

Rule Generation

The range of the hidden nodes activation values of the ANN is the interval (-1, 1), since they have been computed as equation (2.4). In order to extract rules from the ANN, it is necessary for these values to be grouped into a few clusters while the accuracy of the ANN is preserved. Chi2 (Liu and Setiono, 1995), an improved and automated version of ChiMerge (Kerber, 1992), is the algorithm used for this purpose.

After Chi2 discretizes hidden node activation values, rules are generated in two phases. First, rules that describe the classification are obtained in terms of the discretized hidden node activation values. Second, rules that describe each discretized hidden node activation values are obtained in terms of the original attributes of the data instance set.

The first phase implement an efficient rule generator called X2R (Liu and Tan, 1995). It generates a set of rules which covers all the data instances with an error rate not exceeding the inconsistency rate present in the dataset. It is particularly suitable

for moderate sized datasets with discrete attribute values. When few hidden nodes are left in the pruned network where each of these hidden nodes has a relatively small number of different discrete activation values, it may be possible to find a set of classification rules without the help of any computer program.

As for the second phase, they obtain the rules that describe the relation between the discretized activation values and the input attributes. The discretized activation values at all the remaining hidden nodes are not the only output of the Chi2 algorithm. Chi2 also provides the boundaries of the subintervals of the activation values after the merging process has terminated. Suppose n_j is the number of clusters or subintervals found by Chi2 for the activation values of hidden node j. There are $n_j + 1$ real numbers h_{D0}, h_{D1}, h_{D2}, …, h_{Dn_j}, such that $-1 = h_{D0} < h_{D1} < h_{D2} < \ldots < h_{Dn_{gj}-1} < h_{Dn_j} = 1$, which form these n_j clusters. An activation value h_j of an input pattern will be discretized into the l-th cluster if $h_{Dl-1} < h_j < h_{Dl}$. The activation value h_j is obtained by applying equation (2.7). Hence, an activation value falls into subinterval $[h_{Dl-1}, h_{Dl}]$ if its net input $({}_2\mathbf{w}_j^t\mathbf{x} + {}_2q_j)$ satisfies the condition

$$\tanh^{-1}(h_{Dl-1}) \le {}_2\mathbf{w}_j^t\mathbf{x} + {}_2q_j < \tanh^{-1}(h_{Dl}) \tag{2.9}$$

where ${}_2\mathbf{w}_j$ is the weight vector from the input layer to one hidden node j and $\tanh^{-1}(t)$ is the inverse of the $\tanh(t)$ function

$$\tanh^{-1}(t) = \log((1 + t)/(1-t))/2. \tag{2.10}$$

The equation (2.9) describes the half-spaces (in the case of $l = 1$ or $l = n_j$) or the intersection of two half-spaces (in the case of $l = 2, 3, \ldots, n_j-1$) in the original input space of the dataset where a pattern will have a discretized hidden node activation value located in the l-th subinterval.

Note that in order to obtain the rules that describe the relation between hidden node activation values and the class labels, the weights of the connections between the hidden nodes and the output nodes are not required. On the other hand, rule conditions

that determine the subintervals of the discretized activation values are specified by the weights of the ANN connections from the input nodes to the hidden nodes. Combining the rules and conditions obtained from the two phases, Setiono and Liu (1997) find decision boundaries in terms of the original attributes that classify the patterns of the dataset. So the rules finally generated is like the following: "If $-x_1 + x_2 = 0$ AND $1.51x_1 - x_2 = 0.25$, then $d_1 = 1$ else $d_1 = 0$."

2.2.2 STARE (Zhou et al., 2000)

Data Generation

Suppose that a trained ANN $\mathcal{N}$ for the classification problem has been obtained. If an input pattern ${}_c\mathbf{x}^t = ({}_cx_1, {}_cx_2, \ldots, {}_cx_m)$ is fed to $\mathcal{N}$, its corresponding output pattern ${}_c\mathbf{y}^t = ({}_cy_1, {}_cy_2, \ldots, {}_cy_q)$ is derived from the output nodes. Through combining ${}_c\mathbf{x}$ and ${}_c\mathbf{y}$, an instance $({}_c\mathbf{x}, {}_c\mathbf{y})$ which could be generated by $\mathcal{N}$ and represent the function of $\mathcal{N}$ on the specific point ${}_c\mathbf{x}$ in the instance space is got. Considering the data instance set $\mathcal{S}$ which could be generated by $\mathcal{N}$ through gradually sliding ${}_c\mathbf{x}$ in the instance space, that is, varying each ${}_cx_i$ ($i = 1, 2, \ldots, m$) across its value range so that diversified input patterns could appear as more as possible, Zhou et al. (2000) believe that the function of $\mathcal{N}$ is encoded in $\mathcal{S}$. Thus, if a comprehensible rule set $\mathcal{R}$ can be extracted from $\mathcal{S}$, the function of $\mathcal{R}$ will approach that of $\mathcal{N}$ while the size of $\mathcal{S}$ approaches infinity, that is,

$$\lim_{|\mathcal{S}|\to\infty} func(\mathcal{R}) = func(\mathcal{N})$$

For a category attribute, it is easy to be done through making all the possible values appear in turn. For a continuous attribute, it is necessary to sample it across its value range with a high frequency. For example, we get 100 patterns for an attribute whose value range is [0, 1]. In other words, we sample it with the interval 0.01.

Continuous Attribute Processing

After the data instance set $\mathcal{S}$ has been created, the rule set $\mathcal{R}$ could be extracted via STARE. Attention should be paid to that STARE always extracts rules from category attributes. Only when new rules cannot be extracted out, STARE resorts to the continuous attribute that has the best clustering effect, that is, the number of discretized clusters is the least. The discretized continuous attribute by Chi2 will be regarded as a new category attribute, and used together with previous category attributes for rule extraction.

Comparing with the continuous attribute processing used by most rule extraction approaches which discretizes all the continuous attributes at the beginning, the processing of STARE has some advantages. First, in most cases, especially when there are lots of attributes, some continuous attributes will not appear in the extracted rules. Second, in STARE, since every time only one continuous attribute may be processed, different attributes could be discretized to different number of clusters. If they are discretized to equal number of clusters, some helpful information may be obliterated. Third, since the continuous attributes are discretized one by one, the computational complexity for discretization will gradually descend due to thr unique rule creation process, which will be elaborated later.

However, the continuous attribute processing of STARE also has shortcomings. Since continuous attributes are processed one by one, the relationship among multiple continuous attributes is ignored, which makes STARE work quite well on problems where there is little interaction between attributes.

Rule Creation

A subset $\mathcal{H}$ of input attributes will be identified. Based on $\mathcal{H}$, a rule will be created. At first, $\mathcal{H}$ is only composed of one attribute x_i that is randomly picked out from present category attributes. Consider the possible values of one attribute x_i. If there

exists a value u that all the instances possessing such value in $\mathcal{S}$ fall into a certain class C, a rule r will be created via regarding $x_i = u$ as the antecedent and regarding C as the consequent (if $x_i = u$, then C). If u does not exist, x_i will be replaced by another category attribute x_j, and a similar process searching for x_j's fitful value v occurs.

If no rule has been created after examining all the single attributes, another category attribute will be appended to $\mathcal{H}$. In other words, $\mathcal{H}$ is composed of two attributes now. Suppose they are x_i and x_j. The resulting rule will have two conjunctive antecedents, that is, $x_i = u$ AND $x_j = v$. Thus, the number of rule antecedents will increase while the number of attributes in $\mathcal{H}$ increases. Because of investigation of Zhou et al. that rules with more than three antecedents are nearly incomprehensible to human beings, they limit the maximum number of attributes in $\mathcal{H}$ to be 3. Accordingly, the limitation will result in the increase of the number of rules. However, they believe it is valuable because it will increase the comprehensibility of the extracted rules as well.

If no rule has been created after examining all the possible subsets of category attributes, a continuous attribute will be discretized and regarded as a new category attributes, as described in pre-section. If no rules could be created after all the continuous attributes being discretized, or $\mathcal{S}$ has been fully covered by $\mathcal{R}$, the rule creation process terminates.

Priority Formation

In STARE, if a new rule r is extracted, instances covered by it will be deleted from the data instance set $\mathcal{S}$. The part of instance space that has been covered by $\mathcal{R}$ is named as "known space", and the rest part of instance space is named as "unknown space". The known space is always growing and the unknown space is always shrinking while the rule set $\mathcal{R}$ matures. Namely, the number of extracted rules lifts. This leads the rules to be expressed in priority form. The earlier the rule being extracted, the higher its

priority should be. The reason for this is that the rules could be viewed as being extracted from a series of unknown spaces where the later ones are subsets of the earlier ones.

Such kind of rule form has some advantages because the rules should be matched by priority when the rule set is used. First, the rules could have concise appearances because the higher priority rules implicitly work as rule antecedents for the lower priority ones. Second, some rules could not be extracted without the help of earlier extracted ones because the number of explicit rule antecedents is limited to three in the rule creation process, as mentioned in pre-section. Third, when a common rule set is used, a conflict check should be executed so that specific rules will not be replaced by general ones. Since priority rules have already contained order information, the time-consuming check process could be omitted.

An important point is that the priority rule form is also the requirement of the continuous attribute processing of STARE. Since the continuous attributes are discretized one by one and the unknown space is always shrinking, the working areas of the rules involving different continuous attributes are different.

Fidelity Evaluation

STARE only creates rules from category attributes, so the dimension of the unknown space could be measured as the number of category attributes. Thus, discretizing continuous attributes could be viewed as incrementing the dimension of unknown space when the rule extraction task is too difficult to be accomplished. Such dimension incrementing technique has been proved to be valid in solving many problems (Vapnik, 1995). However, it also raises a new problem. Since the unknown space with lower dimension has been transformed to a space with higher dimension, the distribution of instances in S may be "distorted". Namely, the instances may not uniformly

distribute across the new unknown space. Thus, the extracted rules may only be valid for a part of the unknown space. In order to solve this problem, statistics is introduced.

The newly created rule r will be evaluated before coming into $\mathcal{R}$ as a member. It is used to examine the fidelity of r, that is, the percentage of instances for which rule r and neural network $\mathcal{N}$ make the same classification. First, n_u instances are generated by $\mathcal{N}$ in the same way described in pre-section. Among those instances, the ones covered by $\mathcal{R}$ will be repeatedly replaced by newly generated instances until none of the n_u instances is covered by $\mathcal{R}$. Then, rule r works to classify the n_u instances. If the accuracy of r is beyond a pre-set lower bound d, r will be accepted as a new member of $\mathcal{R}$. Else r will be rejected and another rule should be created. If consider the n_u instances that are generated by $\mathcal{N}$ and could represent the function of $\mathcal{N}$ in present unknown space, r's accuracy is actually its fidelity. Experiments show that when n_u equals the size of $\mathcal{S}$ of the time, accurate rules could be extracted.

What should be heeded is that the number of extracted rules could be determined by users through setting the value of d. Since large d will result in high frequency of rule rejection, and the smaller d will result in that the more rules are extracted, definitely.

In summary, the STARE algorithm is depicted in Table 2.2.

Table 2.2: STARE Algorithm

1. Generate a rule extraction instance set $\mathcal{S}$ by the trained ANN $\mathcal{N}$.
2. Create a new rule r based on an adequate category attribute subset $\mathcal{H}$, go to step 3. If there is no such subset, jump step 5.
3. Generate n_u instances by $\mathcal{S}$. If r's fidelity is beyond the pre-set lower bound d and r is appended to the extracted rule set $\mathcal{R}$, go to step 4. Else r is rejected, return step 2.
4. Delete the instances covered by r from $\mathcal{S}$. If $\mathcal{S}$ is empty, jump step 6. Else returns step 2.
5. If all continuous attributes have been discretized, go to step 6. Else the con-

tinuous attribute with the best clustering effect is discretized returns step 2.
6. Merge the rules that have the same rule consequent and successive priorities.

2.2.3 CREFANN (Gaweda et al., 2000)

Gaweda et al. propose an algorithm to extract rules from an ANN. We name it CREFANN (Constant Rule Extraction from Function Approximating Neural Networks). The algorithm is based on clustering of the hidden layer activations. The center of each cluster in the product space of the hidden layer activations is a point that represents a hyper-plane in the input space. These hyper-planes are defined by linear equations whose coefficients are the weights from the input nodes to the hidden nodes. The number of linear equations in the antecedent of a single rule is obviously equal to the number of nodes in the hidden layer. Rule consequences are found by multiplying cluster coordinates (hidden node activations) by the output weights. Clustering of the hidden node responses has been used in prior studies (Setiono and Liu, 1997) (Weijters and Bosch, 1998); however, the form of the rules extracted by the presented method is a new extension.

Rule Extraction Algorithm

Let $\mathbf{x}^t \equiv (x_1, x_2, \ldots, x_m)$ denote an input pattern of an ANN and assume, for simplicity, that the ANN has one linear output denoted by y (generalization to the case with multiple outputs is straightforward). Further, assume that the ANN has one hidden layer of p nodes with sigmoid activation functions. Let ${}_2\mathbf{w}_j^t \equiv ({}_2w_{j1}, {}_2w_{j2}, \ldots, {}_2w_{jm})$ denote the weight vector between the input layer and the j-th hidden node and ${}_2q_j$ be the bias of the j-th hidden node where $1 \le j \le p$. Let ${}_3\mathbf{w}^t \equiv ({}_3w_1, {}_3w_2, \ldots, {}_3w_p)$ denote the weight vector between the hidden layer and the output node and ${}_3q$ be the bias of the output node.

- Feed the training patterns to the network and search for clusters in the

p-dimensional product space of the responses of hidden nodes.

- For each cluster ${}_l\bar{h} \equiv ({}_lh_1, {}_lh_2, \ldots, {}_lh_p)$, find the corresponding network output

$$ {}_l\bar{y} = \sum_{j=1}^{p} {}_3w_j \, {}_lh_p + {}_3q \tag{2.11} $$

where $1 \le l \le n_g$ and n_g is the number of clusters.

- For each of p hidden nodes and each of n_g clusters, find the parameters of the input linear prototypes denoted here as $\mathrm{lp}_l({}_2\mathbf{w}_j)$:

$$ \mathrm{lp}_l({}_2\mathbf{w}_j) \equiv \sum_{i=1}^{m} {}_2w_{ji} \, x_i + {}_2q_j + \ln\left(\frac{1-{}_lh_j}{{}_lh_j}\right) \tag{2.12} $$

The rules have the following form:

$$ R_l\text{: If } \mathrm{lp}_l({}_2\mathbf{w}_1)=0 \text{ and } \mathrm{lp}_l({}_2\mathbf{w}_2)=0 \text{ and } \ldots \text{ and } \mathrm{lp}_l({}_2\mathbf{w}_p)=0, \text{ then } y={}_l\bar{y} \tag{2.13} $$

A set of rules of the form (2.13) can be regarded as a composition of local models. These models partition the input space into subspaces defined as linear combinations of the input variables, for which the output attains specific, constant values.

Rule-based Approximation Algorithm

For a single input vector ${}_c\mathbf{x}^t \equiv ({}_cx_1, {}_cx_2, \ldots, {}_cx_m)$

- Find the similarity between ${}_c\mathbf{x}$ and every linear prototype $\mathrm{lp}_l({}_2\mathbf{w}_j)$; $1 \le l \le n_g$ and $1 \le j \le p$. The similarity measure can be defined as the following function reciprocal to the distance between ${}_c\mathbf{x}$ and a hyper-plane corresponding to the linear prototype $\mathrm{lp}_l({}_2\mathbf{w}_j)$

$$ m(\mathrm{lp}_l({}_2\mathbf{w}_j), {}_c\mathbf{x}) \equiv \frac{1}{1+\exp(\mathrm{d}(\mathrm{lp}_l({}_2\mathbf{w}_j), {}_c\mathbf{x}))} \tag{2.14} $$

where $\mathrm{d}(\mathrm{lp}_l({}_2\mathbf{w}_j), {}_c\mathbf{x})$ is the following distance measure

$$ \mathrm{d}(\mathrm{lp}_l({}_2\mathbf{w}_j), {}_c\mathbf{x}) \equiv \frac{\left| \sum_{i=1}^{m} {}_2w_{ji} \, {}_cx_i + q_j + \ln\left(\frac{1-{}_lh_j}{{}_lh_j}\right) \right|}{\sqrt{\sum_{i=1}^{m} {}_2w_{ji}^{\,2}}} \tag{2.15} $$

- For each rule find its activation defined as follows

$$a_l \equiv \sqrt{\sum_{j=1}^{p} m(\mathrm{lp}_l({}_2\mathbf{w}_j),\ {}_c\mathbf{x})^2} \tag{2.16}$$

- Select the best rule i.e. the one with the maximum activation

$$a_k = \max_{1<l<n_g}(a_l) \tag{2.17}$$

The actual output is the consequence of the rule R_k

$$y = {}_k\bar{y} \tag{2.18}$$

2.2.4 REFANN (Setiono et al., 2002).

Neural Network Training and Pruning Algorithm

To reduce the number of rules, redundant hidden nodes and irrelevant input nodes are first removed by a pruning method called N2PFA (Neural Networks Pruning for Function Approximate).

Give an available set of data instances $\{({}_c\mathbf{x}, {}_cd) | {}_c\mathbf{x} \in \mathcal{R}^m, {}_cd \in \mathcal{R}$ and $c = 1, 2, ..., n\}$. First, it is randomly divided into three subsets: the training, the cross-validation and the test sets. Using the training data set, an ANN with p hidden nodes is trained, so as to minimize the sum of squared errors $E({}_2\mathbf{W}, {}_3\mathbf{W}, {}_2\mathrm{q})$ augmented with a penalty term $P({}_2\mathbf{W}, {}_3\mathbf{W}, {}_2\mathrm{q})$.

$$E({}_2\mathbf{W}, {}_3\mathbf{W}, {}_2\mathrm{q}) = \sum_{c=1}^{n}({}_cd - {}_cy)^2 + P({}_2\mathbf{W}, {}_3\mathbf{W}, {}_2\mathrm{q}) \tag{2.19}$$

$$P({}_2\mathbf{W}, {}_3\mathbf{W}, {}_2\mathrm{q}) = e_1 \sum_{j=1}^{p}\left(\left(\sum_{i=1}^{m}\frac{b({}_2w_{ji})^2}{1+b({}_2w_{ji})^2}\right)+\frac{b({}_3w_j)^2}{1+b({}_3w_j)^2}+\frac{b({}_2q_j)^2}{1+b({}_2q_j)^2}\right)$$

$$+e_2\sum_{j=1}^{p}\left(\left(\sum_{i=1}^{m}{}_2w_{ji}^{\,2}\right)+{}_3w_j^{\,2}+{}_2q_j^{\,2}\right) \tag{2.20}$$

where ${}_2\mathbf{W}$, ${}_3\mathbf{W}$ and ${}_2\mathrm{q}$ denote these parameter matrixes consisting of ${}_2w_{ji}$, ${}_3w_j$ and ${}_2q_j$ respectively; e_1, e_2, and β are positive penalty parameters; ${}_2w_{ji}$ is the weight of the

connection from input node i, $i \in \{1, 2, ..., m\}$ to hidden node j, $j \in \{1, 2, ..., p\}$ and ${}_2q_j$ is the bias of hidden node j; ${}_3w_j$ is the weight of the connection from hidden node j to the output node. Then the hidden node activation value ${}_ch_j$ for input pattern ${}_c\mathbf{x}$ and its predicted value ${}_cy$ are computed as follows:

$$ {}_cy = \sum_{j=1}^{p} {}_3w_j \, {}_ch_j \,, \tag{2.21} $$

$$ {}_ch_j = h({}_cnet_j) = \tanh\left(\sum_{i=1}^{m} {}_2w_{ji} \, {}_cx_i + {}_2q_j\right), \tag{2.22} $$

Once the ANN has been trained, its hidden and input nodes are inspected as candidate for possible removal by a network pruning algorithm. This algorithm removes redundant and irrelevant nodes by computing them mean absolute error (MAE) of the network's prediction. In particular, ET and EX, respectively, the MAEs on the training set $\mathcal{T}$ and the cross-validation set $\mathcal{X}$, are used to determine when pruning should be terminated.

$$ ET = \frac{1}{|\mathcal{T}|} \sum_{({}_c\mathbf{x}, {}_ct) \in \mathcal{T}} |{}_cd - {}_cy| \tag{2.23} $$

$$ EX = \frac{1}{|\mathcal{X}|} \sum_{({}_c\mathbf{x}, {}_ct) \in \mathcal{X}} |{}_cd - {}_cy| \tag{2.24} $$

where $|\mathcal{T}|$ and $|\mathcal{X}|$ are the cardinality of training and cross-validation sets, respectively.

Table 2.3: N2PFA Algorithm

Given: Data set $({}_c\mathbf{x}, {}_cd)$, $c = 1,2, \ldots, n$.

Objective: Find an ANN with reduced number of hidden and input nodes that fits the data and generalizes well.

1. Split the data set into three subsets: training, cross-validation, and test sets.
2. Train an ANN with a sufficiently large number of hidden nodes to minimize the error equation (2.19).
3. Compute ET and EX, and set $ET_{best} = ET$, $EX_{best} = EX$, $E_{max} = \max\{ ET_{best}, EX_{best} \}$.
4. Remove redundant hidden nodes:
   a. For each $j = 1, 2, \ldots, p$, set ${}_3w_j = 0$ and compute the prediction errors ET_j.
   b. Retrain the ANN with ${}_3w_{l_h} = 0$ where $ET_{l_h} = \text{Min}\{ET_j, j=1, 2, ..., p\}$, and

compute ET and EX of the retrained ANN.

d. If $ET \le (1+\alpha)E_{max}$ and $EX \le (1+\alpha)E_{max}$, then

- Remove hidden node h.
- set $ET_{best} = \text{Min}\{ET, ET_{best}\}$, $EX_{best} = \text{Min}\{EX, EX_{best}\}$, and $E_{max} = \text{Max}\{ET_{best}, EX_{best}\}$
- Set $p = p - 1$ and go to step 4.a.

Else uses the previous setting of network weights.

5. Remove irrelevant inputs:

a. For each $i = 1, 2, \ldots, m$, set ${}_2w_{ji} = 0$ for all j and compute the prediction errors ET_i.

b. Retrain the ANN with ${}_2w_{jl} = 0$ for all j where $ET_l = \text{Min}\{ET_i, i=1, 2, ...,m\}$, and compute ET and EX of the retrained ANN.

d. If $ET \le (1+\alpha)E_{max}$ and $EX \le (1+\alpha)E_{max}$, then

- Remove input node l.
- set $ET_{best} = \text{Min}\{ET, ET_{best}\}$, $EX_{best} = \text{Min}\{EX, EX_{best}\}$, and $E_{max} = \text{Max}\{ET_{best}, EX_{best}\}$
- Set $m = m - 1$ and go to step 5.a.

Else uses the previous setting of network weights.

6. Report the accuracy of the ANN on the test data.

The value of E_{max} is used to determine if a network node can be removed. Typically, at the beginning of the algorithm when there are many hidden nodes in the network, the training mean absolute error ET will be much smaller than the cross-validation mean absolute error EX. The value of ET increases as more and more nodes are removed. As the network approaches its optimal structure, EX is expected to decrease. As a result, if only ET_{best} is used to determine whether a node can be removed, many redundant nodes are likely to remain in the network when the algorithm terminates because the initial value of ET_{best} tends to be small. On the other hand, if only EX_{best} is used, then the network would perform well on the cross-validation set but may not necessarily generalize well on the test set. This could be caused by the small number of samples available for cross-validation or an uneven distribution of the data in the training and cross-validation sets. Therefore, E_{max} is assigned the larger of ET_{best} and EX_{best} so as to remove as many redundant nodes as possible without sacri-

ficing the generalization accuracy. The parameter $a > 0$ is introduced to control the chances that a node will be removed. With a larger value of a, more nodes can be removed. However, the accuracy of the resulting network on the test data set may deteriorate. Setiono et al. have conducted extensive experiments to find a value for this parameter that works well for the majority of their test problems.

To Approximate Hidden Node Activation Function

Having produced the pruned network, Setiono et al. (2000) can proceed to extract rules that explain the network outputs as a collection of linear functions. The first step in their rule extraction method is to approximate the hidden node activation function. They approximate the activation function by either a three-piece linear function or a five-piece linear function.

Three-Piece Linear Approximation

Since $h(t)$ is anti-symmetric, it is sufficient to illustrate the approximation just for the nonnegative values of t. Suppose that the input ranges from zero to t_m. A simple

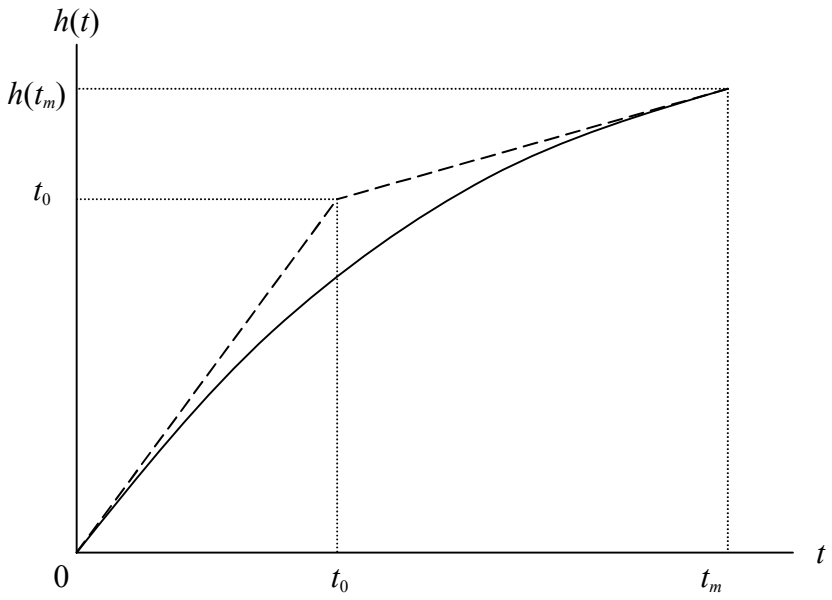

Figure 2.1: The tanh(t) **function (solid curve) for** $t \in [0, t_m]$ **is approximated by a two-piece linear function (dashed lines).**

and convenient approximation of $h(t)$ is to over-estimate it by the piecewise linear function shown in Figure 2.1. To ensure that $L(t)$ is larger than $h(t)$ everywhere between 0 to t_m, the line on the left should intersect the coordinate (0, 0) with a gradient of $h'(0)=1$, and the line on the right should intersect the coordinate $(t_m, h(t_m))$ with a gradient of $h'(t_m) = 1- h^2(t_m)$. Thus, $L(t)$ can be written as

$$L(t)=\begin{cases} t & 0 \le t \le t_0 \\ h'(t_m)(t-t_m)+h(t_m) & t \ge t_0 \end{cases} \quad (2.25).$$

The point of intersection x_0 of the two line segments is given by

$$t_0 = \frac{h(t_m)-t_m h'(t_m)}{h^2(t_m)} \quad (2.26)$$

The total error E_A of estimating $h(x)$ by $L(x)$ is as

$$\begin{aligned} E_A &= \int_0^{x_m} (L(t)-h(t))dt \\ &= \frac{1}{2}[t_0^2 + (t_m - t_0)(t_0+h(t_m)] - \ln\cosh(t) \\ &\to -\tfrac{1}{2} - \ln 0.5 \text{ as } t_m \to \infty \end{aligned} \quad (2.27)$$

Namely, the total error is bounded by a constant value.

Another simple linearization method of approximating is to under-estimate $h(t)$ by a three-piece linear function. It can be shown that the total error of the under-estimation method is unbounded and is larger than that of the over-estimation method for $t_m > 2.96$.

Five-Piece Linear Approximation

By having more line segments, the function $h(t)$ can be approximated with better accuracy. Figure 2.2 shows how this function can be approximated by a three-piece linear function for $t \in [0, t_m]$. The three dashed lines are given by

$$L(t)=\begin{cases} t & 0 \le t \le t_0 \\ h'(t_1)(t-t_1)+h(t_1) & t_0 \le t \le t_2 \\ h'(t_m)(t-t_m)+h(t_m) & t \ge t_2 \end{cases} \quad (2.28)$$

The underlying idea for this approximation is to find the point that minimizes the total

area A of the triangle and the two trapezoids

$$A = \frac{1}{2}[t_0^2 + (t_0 + y_2)(t_2 - t_0) + (y_2 + h(t_m))(t_m - t_2)] \tag{2.29}$$

where t_0, t_1 and y_2 are expressed in terms of a constant t_m and the free parameter t_1.

$$t_0 = \frac{h(t_1) - t_1 h'(t_1)}{1 - h^2(t_1)} \tag{2.30}$$

$$t_2 = \frac{t_1 h'(t_1) - t_m h'(t_m) - h(t_1) + h(t_m)}{h'(t_1) - h'(t_m)} \tag{2.31}$$

$$y_2 = \frac{h'(t_1)h'(t_m)(t_1 - t_m) + h'(t_1)h(t_m) - h'(t_m)h(t_1)}{h'(t_1) - h'(t_m)} \tag{2.32}$$

The bisection method (Gill, Mao and Li, 1981) for one-dimensional optimization problems is operated to find the optimal value of t_1. The total error E_A of estimating $h(x)$ by this linear approximation is computed to be as

$$\begin{aligned} E_A &= \int_0^{t_m} (L(t) - h(t))dt \\ &\to 0.071169 \text{ as } t_m \to \infty \end{aligned} \tag{2.33}$$

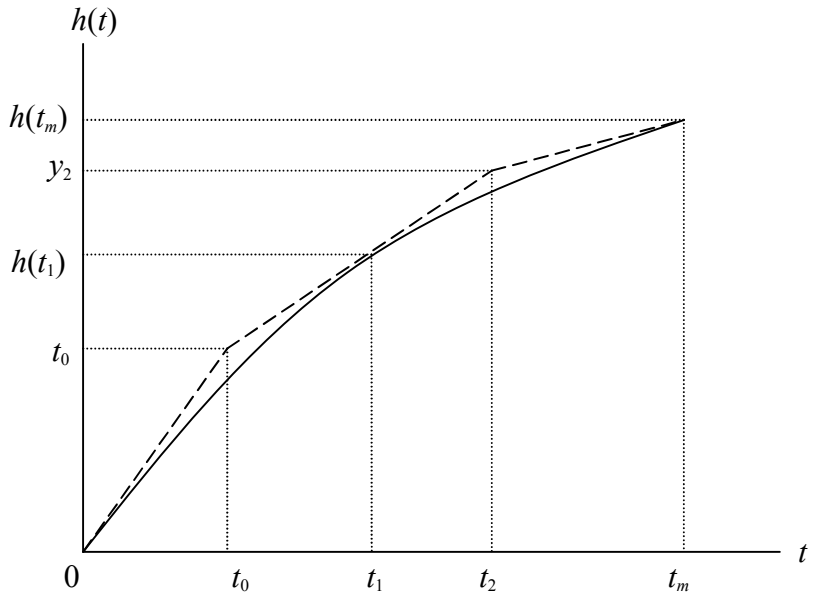

Figure 2.2: The tanh(t) **function (solid curve) for** $t \in [0, t_m]$ **is approximated by a three-piece linear function (dashed lines).**

Rule Generation

Setiono et al. (2002) introduce a method called REFANN (Rule Extraction from

Function Approximating Neural Networks). REFANN generates rules from a pruned ANN according to Table 2.4.

Table 2.4: REFANN Algorithm

Given: Data set ($_c\mathbf{x}$, $_cd$), c= 1,2, …, n and a pruned ANN with p hidden nodes.
Objective: Generate linear regression rules from the ANN.

1. Train and prune an ANN with one hidden layer and one output nodes.
2. For each hidden node j= 1, 2, …, p:
   a. Determine $t_{jm} = \text{Max}\{|_cnet_j|, c=1, 2, ..., n\}$ from the training instances.
   b. If the three-piece linear approximation is used:
      - Compute t_{j0}
      - Define the three-piece approximating linear function $L_j(t)$ as

$$L_j(t)=\begin{cases} h'(t_{jm})(t+t_{jm})-h(t_{jm}) & t\le -t_{j0} \\ t & -t_{j0}\le t\le t_{j0} \\ h'(t_{jm})(t-t_{jm})+h(t_{jm}) & t\ge t_{j0} \end{cases}.$$

      - Use the pair of point $-t_{j0}$ and t_{j0} of function $L_j(t)$ to divide the input space into 3^p subregions.
   c. Else if the five-piece linear approximation is used:
      - Use the bisection method to find t_{j1} and compute t_{j0} and t_{j2} according to equations (2.30) and (2.31).
      - Define the five-piece approximating linear function $L_j(t)$ as

$$L_j(x)=\begin{cases} h'(t_{jm})(t+t_{jm})-h(t_{jm}) & t\le -t_{j2} \\ h'(t_{j1})(t+t_{j1})-h(t_{j1}) & -t_{j2}\le t\le -t_{j0} \\ t & -t_{j0}\le t\le t_{j0} \\ h'(t_{j1})(t-t_{j1})+h(t_{j1}) & t_{j0}\le t\le t_{j2} \\ h'(t_{jm})(t-t_{jm})+h(t_{jm}) & t\ge t_{j2} \end{cases}$$

      - Use the points $-t_{j2}$, $-t_{j0}$, t_{j0} and t_{j2} divide the input space into 5^p subregions.
3. For each nonempty sub-region, generate a rule as follows:
   a. Define a linear equation that approximates the network's output for $_c\mathbf{x}$ in the sub-region as the consequent of the extracted rule

$$_cy'=\sum_{j=1}^{p} {}_3w_jL_j(_cnet_j) \tag{2.34}$$

$$_cnet_j=\sum_{i=1}^{m} {}_2w_{ji}\,{}_cx_i+{}_2q_j \tag{2.35}$$

b. Generate the rule condition: (C_1, C_2, and C_p), where C_j is either ${}_c net_j < -t_{j0}$, $-t_{j0} \le {}_c net_j \le -t_{j0}$, or ${}_c net_j > t_{j0}$ for the three-piece approximation approach; or C_j is either ${}_c net_j \le -t_{j2}$, $-t_{j2} < {}_c net_j < -t_{j0}$, , $-t_{j0} \le {}_c net_j \le t_{j0}$, , $t_{j0} < {}_c net_j < t_{j2}$,or ${}_c net_j \ge t_{j2}$ for the five-piece approximation approach.

4. (Optional) Apply C4.5 (Quinlan, 1993) to simplify the rule conditions.

In general, a rule condition is defined in terms of the weighted sum of the inputs (Towell and Shavlik, 1993) which corresponds to an oblique hyper-plane in the input space. This type of rule condition can be difficult for users to interpret. In some cases, the oblique hyper-planes can be replaced by hyper-planes that are parallel to the axes without affecting the prediction accuracy of the rules on the data set. Consequently, the hyper-planes can be defined in terms of the isolated inputs, and are easier for the users to understand. In some cases of real-life data, this enhanced interpretability would arrive at a possible cost of reduced accuracy. If the replacement of rule conditions is still desired, it can be performed by employing a classification method such as C4.5 (Quinlan, 1993) in the optional Step 4.

2.2.5 RN2 (Satio and Nakano, 2002)

Neural Network Training

Set (${}^{Q}x_1, \ldots, {}^{Q}x_{m_Q}, {}^{N}x_1, \ldots, {}^{N}x_{m_N}, d$) or (${}^{Q}\mathbf{x}, {}^{N}\mathbf{x}, d$) be a vector of attributes describing a instance, where ${}^{Q}x_l$ is a nominal attribute and ${}^{N}x_i$ is a numeric attribute. Here, by adding extra categories, if necessary, without losing generality, assume that ${}^{Q}x_l$ exactly matches the one category. Therefore, for each a dummy variable expressed by ${}^{D}x_{lr}$ is introduced as follows:

$$ {}^{D}x_{lr} = \begin{cases} 1 & \text{if } {}^{Q}x_l \text{ match the } r\text{ - th category.} \\ 0 & \text{otherwise} \end{cases} \tag{2.36} $$

Here $r = 1, 2, \ldots m_l$ and m_l is the number of distinct categories appearing in ${}^{Q}x_l$. Hereafter, ${}^{D}\mathbf{x}$ is used to denote a vector constructed by all dummy variables.

Using the training set of data instances, an ANN with p hidden nodes whose activation functions are $\exp(t)=e^t$ is trained. To improve both the generalization performance and the readability of the learning result, Satio and Nakano (2002) adopt a method to learn a distinct penalty factor for each weight as a minimization problem over the cross-validation error, called the minimum cross-validation (MCV) regularizer (Saito and Nakano, 2000).

Given an available set of data instances, $\mathcal{T} = \{({}^{\mathrm{D}}_{c}\mathbf{x}, {}^{\mathrm{N}}_{c}\mathbf{x}, {}_{c}d) \mid c = 1, 2, \ldots n\}$. An ANN is trained so that minimize the sum of squared error $\mathrm{E}({}^{\mathrm{D}}_{2}\mathbf{W}, {}^{\mathrm{N}}_{2}\mathbf{W}, {}_{3}\mathbf{W}, {}_{3}\mathrm{q})$.

$$\mathrm{E}({}^{\mathrm{D}}_{2}\mathbf{W}, {}^{\mathrm{N}}_{2}\mathbf{W}, {}_{3}\mathbf{W}, {}_{3}\mathrm{q}) = \frac{1}{2}\left(\sum_{c=1}^{n}({}_{c}d - {}_{c}y)^2 + \mathrm{P}({}^{\mathrm{D}}_{2}\mathbf{W}, {}^{\mathrm{N}}_{2}\mathbf{W}, {}_{3}\mathbf{W}, {}_{3}\mathrm{q})\right) \tag{2.37}$$

$$\mathrm{P}({}^{\mathrm{D}}_{2}\mathbf{W}, {}^{\mathrm{N}}_{2}\mathbf{W}, {}_{3}\mathbf{W}, {}_{3}\mathrm{q}) = \sum_{j=1}^{p}\sum_{l=1}^{m_{\mathrm{Q}}}\sum_{r=1}^{m_l} {}_{2}l_{jlr}({}^{\mathrm{D}}_{2}w_{jlr})^2 + \sum_{j=1}^{p}\sum_{i=1}^{m_{\mathrm{N}}} {}_{2}l_{ji}({}^{\mathrm{N}}_{2}w_{ji})^2$$

$$+ \sum_{j=1}^{p} {}_{3}l_j({}_{3}w_j)^2 + {}_{3}l({}_{3}q)^2 \tag{2.38}$$

where ${}^{\mathrm{D}}_{2}\mathbf{W}$, ${}^{\mathrm{N}}_{2}\mathbf{W}$, ${}_{3}\mathbf{W}$ and ${}_{3}\mathrm{q}$ denote these parameter matrixes consisting of ${}^{\mathrm{D}}_{2}w_{jlr}$, ${}^{\mathrm{N}}_{2}w_{ji}$, ${}_{3}w_j$ and ${}_{3}q$ respectively; ${}_{2}l_{jlr}$, ${}_{2}l_{ji}$, ${}_{3}l_j$ and ${}_{3}l$ are positive penalty parameters; ${}^{\mathrm{D}}_{2}w_{jlr}$ is the weight of the connection from nominal input node lr to hidden node j; ${}^{\mathrm{N}}_{2}w_{ji}$ is the weight of the connection from numeric input node i to hidden node j; ${}_{3}w_j$ is the weight of the connection from hidden node j to the output node and ${}_{3}q$ is the bias of the output node. Then the hidden node activation value ${}_{c}h_j$ for input $({}^{\mathrm{D}}_{c}\mathbf{x}, {}^{\mathrm{N}}_{c}\mathbf{x})$ and its predicted value ${}_{c}y$ are computed as follows:

$${}_{c}y = {}_{3}q + \sum_{j=1}^{p} {}_{3}w_j\,{}_{c}h_j\,, \tag{2.39}$$

$${}_{c}h_j = \exp({}_{c}net_j) = \exp\left(\sum_{l=1}^{m_{\mathrm{Q}}}\sum_{r=1}^{m_l} {}^{\mathrm{D}}_{2}w_{jlr}\,{}^{\mathrm{D}}_{c}x_{lr} + \sum_{i=1}^{m} {}^{\mathrm{N}}_{2}w_{ji}\ln({}^{\mathrm{N}}_{c}x_i)\right)$$

$$= \exp\left(\sum_{l=1}^{m_{\mathrm{Q}}} \sum_{r=1}^{m_l} {}_2^{\mathrm{D}}w_{jlr}\, {}_c^{\mathrm{D}}x_{lr}\right) \prod_{i=1}^{m_{\mathrm{N}}} {}_c^{\mathrm{N}}x_i^{\,{}_2^{\mathrm{N}}w_{ji}} \tag{2.40}$$

Method for Rule Extraction

Assume that a trained ANN has been obtained. In order to find a set of regression rules, a suitable efficient method is needed to extract the nominal conditions from the trained ANN.

Satio and Nakano (2002) can straightforwardly extract a regression rule for each training instance, and simply assemble them to obtain a rule set. Namely, the *j*-th hidden node calculates the following activation value from the values of encoded nominal attributes of the *c*-th training instance

$${}_cs_j = \exp\left(\sum_{l=1}^{m_{\mathrm{Q}}} \sum_{r=1}^{m_l} {}_2^{\mathrm{D}}w_{jlr}\, {}_c^{\mathrm{D}}x_{lr}\right) \tag{2.41}$$

Thus, by putting ${}_c^{\mathrm{D}}\mathbf{x}$, the following set of regression rules can be obtained from the training set of data instances and the trained ANN

$$\text{If } \bigwedge_{l=1}^{m_{\mathrm{Q}}} \bigvee_{{}_c^{\mathrm{D}}x_{lr} \in {}_cQ_l} {}_c^{\mathrm{D}}x_{lr} \quad \text{then } \mathrm{y} = {}_3q + \sum_{j=1}^{p} {}_3w_j\, {}_cs_j \prod_{i=1}^{m_{\mathrm{N}}} {}_c^{\mathrm{N}}x_i^{\,{}_2^{\mathrm{N}}w_{ji}}, \; c=1, 2, \ldots, n. \tag{2.42}$$

where ${}_cQ_l = \{ {}^{\mathrm{D}}x_{lr} \mid {}^{\mathrm{D}}x_{lr} = 1 \}$. However, the results of this naive method are still far from desirable because they contain a large number of similar rules, and each nominal condition is too specific to represent only one training instance.

Based on the above considerations, a new extraction method called RN2 is proposed by Satio and Nakano (2002). In RN2, the number of distinct polynomial equation is reduced by finding representative values of equation (2.41), and adequate number of representatives is determined by using a criterion for model selection, and a set of nominal conditions is determined by solving a standard classification problem by using decision trees.

In order to find representative vector, a set of vectors $\{{}_{c}\mathbf{s}^{t} = ({}_{c}s_1, {}_{c}s_2, \ldots, {}_{c}s_p) | c = 1, 2, \ldots, n\}$ calculated from the nominal inputs is quantized into a set of representative vectors $\{{}_{c_K}\mathbf{u}^{t} = ({}_{c_K}u_1, {}_{c_K}u_2, \ldots, {}_{c_K}u_p) | c_K = 1, 2, \ldots, n_K\}$, where n_K is the number of representatives. Among several vector quantization (VQ) algorithms, the K-means algorithm (Lloyd, 1982) is employed due to its simplicity.

In the K-means algorithm, all of the vectors are assigned simultaneously to their nearest representative vectors. Each representative vector is moved to the group's mean and this process is repeated until there is no further change in the grouping of representative vectors. Consequently, all of the n vectors are partitioned into n_K disjoint subset $\{\mathcal{G}_{c_K}: c_K=1, 2, \ldots, n_K\}$ so that the following sum of squares error function $\mathcal{VD}$ is minimized:

$$\mathcal{VD} = \sum_{c_K=1}^{n_K} \sum_{c \in \mathcal{G}_{c_K}} \sum_{j=1}^{p} ({}_{c}s_j - {}_{c_K}u_j)^2 \tag{2.43}$$

Let n_{c_K} be the number of vectors belonging to $\mathcal{G}_{c_K}$, then, each element of the representative vector is calculated as follows:

$${}_{c_K}u_j = \frac{1}{n_{c_K}} \sum_{c \in \mathcal{G}_{c_K}} {}_{c}s_j \tag{2.44}$$

For a given data instance set and a trained ANN, since the optimal number of distinct polynomial equations is unknown in advance, the plausibility of the number of representatives must be evaluated by changing n_K. For this purpose, the procedure of cross-validation (Stone, 1974) is employed. This procedure divides the data $\mathcal{T}$ at random into $n_{\mathcal{T}}$ distinct segments $\{\mathcal{T}_{c_{\mathcal{T}}} | c_{\mathcal{T}}=1, 2,\ldots, n_{\mathcal{T}}\}$. $n_{\mathcal{T}} - 1$ segments are used for the training, and the remaining one is used for the test. This process is repeated $n_{\mathcal{T}}$ time by changing the remaining segment. The extreme case of $n_{\mathcal{T}} = n$ is known as leave-one-out method, which is often used for a small-sized data instance set (Bishop, 1995).

Here, Satio and Nakano introduce a function $\mathrm{I}({}^{\mathrm{D}}_{c}\mathbf{x})$ that generates the index of the representative vector minimizing the distance.

$$\mathrm{I}({}^{\mathrm{D}}_{c}\mathbf{x}) = \arg\min_{c_{\mathrm{K}}} \sum_{j=1}^{p} ({}_{c}s_{j} - {}_{c_{\mathrm{K}}}u_{j})^{2} \tag{2.45}$$

By placing $\mathrm{I}({}^{\mathrm{D}}_{c}\mathbf{x})$ on the condition parts, they consider the following set of rules using the representative vectors:

$$\text{If } \mathrm{I}({}^{\mathrm{D}}_{c}\mathbf{x}) = c_{\mathrm{K}} \text{ then } \mathrm{y} = {}_{3}q + \sum_{j=1}^{p} {}_{3}w_{j\,c_{\mathrm{K}}} u_{j} \prod_{i=1}^{m_{\mathrm{N}}} {}^{\mathrm{N}}_{c}x_{i}^{\,{}^{\mathrm{N}}_{2}w_{ji}}, \; c_{\mathrm{K}} = 1, 2, \ldots, n_{\mathrm{K}}. \tag{2.46}$$

Since each element of **s** is calculated as

$$s_{j} = \exp\left(\sum_{l=1}^{m_{\mathrm{Q}}} \sum_{r=1}^{m_{I}} {}^{\mathrm{D}}_{2}w_{jlr}\, {}^{\mathrm{D}}_{c}x_{lr}\right) \tag{2.47}$$

Equation (2.46) can be applied to a new sample as well as the training sample. Thus, by using the final weights $({}^{\mathrm{D}}_{2}\mathbf{W}^{(c_{\mathcal{T}})}, {}^{\mathrm{N}}_{2}\mathbf{W}^{(c_{\mathcal{T}})}, {}_{3}\mathbf{W}^{(c_{\mathcal{T}})}, {}_{3}\mathrm{q}^{(c_{\mathcal{T}})})$ calculated from the cross-validation procedure excluding one segment $\mathcal{T}_{c_{\mathcal{T}}}$, the output value with respect to a test sample v can be calculated as

$${}_{v}\hat{y} = {}_{3}q^{(c_{\mathcal{T}})} + \sum_{j=1}^{p} {}_{3}w_{j}^{(c_{\mathcal{T}})}\, {}_{\mathrm{I}({}^{\mathrm{D}}_{v}\mathbf{x})}u_{j} \prod_{i=1}^{m_{\mathrm{N}}} {}^{\mathrm{N}}_{v}x_{i}^{\,{}^{\mathrm{N}}_{2}w_{ji}^{(c_{\mathcal{T}})}} \tag{2.48}$$

Therefore, they define the following cross-validation error function and it is helpful to determine the adequate n_{K}.

$$CV = \frac{1}{n} \sum_{c_{\mathcal{T}}=1}^{n_{\mathcal{T}}} \sum_{v \in \mathcal{T}_{c_{\mathcal{T}}}} ({}_{v}d - {}_{v}\hat{y})^{2} \tag{2.49}$$

Finally, the indexing function $\mathrm{I}({}^{\mathrm{D}}_{c}\mathbf{x})$ described in equation (2.46) must be transformed into a set of nominal conditions as described in equation (2.42). One reasonable approach is to perform this transformation by solving a simple classification problem whose training data instances are $\{({}^{\mathrm{D}}_{c}\mathbf{x}, \mathrm{I}({}^{\mathrm{D}}_{c}\mathbf{x}))| \; c = 1, 2, \ldots, n\}$ where $\mathrm{I}({}^{\mathrm{D}}_{c}\mathbf{x})$ indicates the class label of a training data instance ${}^{\mathrm{D}}_{c}\mathbf{x}$. For this classification problem,

the C4.5 decision tree generation program (Quinlan, 1993) is employed due to its wide availability.

In the induced decision tree, instances are passed from the root node to a terminal node that assigns the corresponding class label, with decisions being made at each non-terminal. By concatenating such decisions at non-terminal node for the path to each terminal node, Sation and Nakano produce a set of conjunctive condition as follows: ${}_{c_R}Q_l$ is initialized to $\{{}^{D}x_{lr} | r=1, ..., m_l\}$ for each l, then it is replaced with $\{{}^{D}x_{lr}\}$ (or ${}_{c_R}Q_l - \{{}^{D}x_{lr}\}$) according to each appearance of ${}^{D}x_{lr} = 1$ (or ${}^{D}x_{lr} = 0$) in the n_R conjunctive condition.

$$\text{if } \bigwedge_{l=1}^{m_Q} \bigvee_{{}^{D}x_{lr} \in {}_{c_R}Q_l} {}^{D}x_{lr} \quad \text{then } y = {}_3q + \sum_{j=1}^{p} {}_3w_j \, {}_{I_{c_K}(c_R)}u_j \prod_{i=1}^{m_N} {}^{N}x_i^{\,{}^{N}_{2}w_{ji}} \;, \; c_R = 1, 2, \ldots, n_R. \quad (2.50)$$

where $I_{c_K}(c_R)$ indicates the class label of the c_R terminal node, and n_R is the number of terminal nodes. In general, the number n_R of terminal nodes is greater than or equal to the number n_K of distinct polynomial equations because the same equation may appear on more than one action parts. Note that it is possible to perform some simplifications of equation (2.50) such as removal of $\bigvee_{r=1}^{m_l} {}^{D}x_{lr}$ and possible disjunctive integration of some nominal condition having the same polynomial equations. Moreover, by using a special condition "else", Sation and Nakano simplify one of nominal conditions and reduce the number of regression rules in many cases.

2.3 Discussion

Most prior studies in the related field focus on the ANN for classification problems and extract rules with the original or generated set of data instances. In other words, most of them do not use the function analysis to extracts rules from the ANN but the data analysis. Thus, they (Zhou et al., 2000) (Setiono et al., 2002) only generate a rule for each nonempty sub-region. We think it is better to extract rules with the

generated dataset because the original data instances set is usually smaller and, as Zhou et al. (2000) mentioned, the function of the ANN is encoded in the generated dataset while its size approaches infinity.

Most methods of prior studies are applied to the classification problem but not the regression problem. If they extract the regression rules from the ANN, they tend to approximate activation values of hidden nodes, for example, Setiono et al. (2002). Their approximations do not depend on the generated data instances set but the original one. It is more adequate to approximate activation values of hidden nodes with the generated data instances set. But, if the size of the generated dataset approaches infinity, it will resemble to approximating the activation functions ($\tanh(t)$) of hidden nodes without any dataset. That is to say, we extract rules from the ANN via function analysis.

If a method of rule extraction is not relative to the set of data instances but relative to the structure and weights of the ANN, it can not extract part rules presented by the dataset but all rules interesting us in the ANN. In addition, it is also more efficient in a large-sized data set.

Moreover, for regression problems, their regression rules extracted from ANN are usually linear. Although the linear regression rules are easy to be understood, they are hard to find the nonlinear characteristics embedded in the ANN. So the multivariate polynomial regression rules whose maximum power values are two may be more favored than linear. The similar multivariate polynomial regression rules have been proposed by Saito and Nakano (2002), but power values of their extracting rules are not restricted to integers and may great than two. The multivariate polynomial regression rules whose order are two are easy to analyze and can get some nonlinear characteristics.

In the following chapter, we propose a new method to extract multivariate poly-

nomial regression rules from the ANN with tanh(t) activation functions in hidden nodes.