This chapter proposes generic neural network models for knowledge mining in generic databases. This chapter stands as another milestone to demonstrate the generality of histograms and regression lines as generic objects to model conventional and symbolic features. The generality of these features are proved through Histogram Neural Networks and the low complexity Regression Line Neural Networks. The theory of histogram/regression line arithmetic is used to propose a Supervised Model (A Back Propagation Network) and an Unsupervised Model (Self Organizing Feature Map) respectively for classification and clustering of histogram/regression objects. The chapter also signifies various other learning perspectives like learning from granules/chunks of data samples, classifying chunks/group of objects and so on. The generic modeling procedure is demonstrated through experiments with 700x3 data and iris data [Appendix]. The utility/applicability of these Neural Networks is validated through classification and clustering of above data.

3.1 Introduction

Artificial neural networks are popular because they have a proven track record in many data mining and decision-support applications. They have been applied across a broad range of industries, from identifying financial series to diagnosing medical conditions, from identifying clusters of valuable customers to identifying fraudulent credit card transactions, from recognizing numbers written on checks to predicting the failure rates of engines (Sushmita et al., 2002). Humans are good at generalizing models from experience and thus the computers are expected to excel by following explicit instructions over and over. The appeal of neural networks is that they bridge
this gap by modeling, on a digital computer, the neural connections in human brains. When used in well-defined domains, their ability to generalize and learn from data mimics our own ability to learn from experience. This ability is useful for data mining and it also makes neural networks an exciting area for research, promising new and better results in the future.

The combination of topology, learning paradigm (supervised or non-supervised learning), and learning algorithm define a neural network model (Yegnanarayana, 1999). There is a wide selection of popular neural network models. For data mining, perhaps the back propagation network and the Kohonen feature map are the most popular. However, there are many different types of neural networks in use. Some are optimized for fast training, others for fast recall of stored memories, others for computing the best possible answer regardless of training or recall time. But the best model for a given application or data mining function depends on the data and the function required.

Selecting which neural network model to use for a particular application is based on the following criteria (Robert, 1997):

(a) **Data type and quantity**: When the input data has real values, then fuzzy Adaptive Resonance Network (ART) or Kohonen maps should be used.

(b) **Training requirements**: In general, whenever we want online learning, then training speed becomes the overriding factor in determining which neural network model to use. Back propagation and recurrent back propagation train quite slowly and so are almost never used in real-time or online learning situations. ART and radial basis function networks, however, train quite fast, usually in a few passes over the data.

(c) **Functional requirements**: Based on the function required, some models can be used. For example, ART and Kohonen feature maps are used for clustering. Back Propagation Network can be used for classification or time-series forecasting.
3.2 Neural Networks for Data Mining: State of the Art

Neural Networks were earlier thought to be unsuitable for data mining because of their inherent black-box nature. The first reason is no information was available from them in comprehensible form, suitable for verification or interpretation by humans. The second is because of the large training time taken to learn from huge data sets. Recently there has been a widespread activity in redressing the former situation, by extracting the embedded knowledge in trained networks in the form of rules. This serves to identify the attributes that, either individually or in a combination, are the most significant determinants of the decision or classification. But still not much has been addressed with the second situation to design neural network models for learning from huge samples within less amount of time. This chapter makes an important contribution in this respect by attempting to learn from the characteristics of data represented by histograms and regression lines.

The main contribution of neural nets towards data mining stems from rule extraction, classification and clustering. Classification is one of the data mining problems receiving great attention recently in the database community. Neural networks have been thought not suitable for data mining because the classifications made were not explicitly stated as symbolic rules that are suitable for verification or interpretation by humans. Hongjun Lu, Rudy Setiono (1996) proposed an approach with which concise symbolic rules with high accuracy can be extracted from a neural network. The network is first trained to achieve the required accuracy rate. Redundant connections of the network are then removed by a network pruning algorithm. The activation values of the hidden units in the network are analyzed, and classification rules are generated using the result of this analysis. LiMin Fu (1994) showed the mechanism to interpret neural network knowledge in symbolic form. In his work he lays down the required definitions for this treatment, formulate the interpretation algorithm, and formally verify its soundness. The main result is a formalized relationship between a neural network and a rule-based system. In addition, it has been demonstrated that the neural network generates rules of better performance than the decision tree approach in noisy conditions. In spite of the effective learning capability as a uniform function approximator, the multilayered NN suffers from unreadability, i.e., it is difficult for
the user to interpret or understand the "knowledge" that the NN has by looking at the connection weights and thresholds obtained by backpropagation (BP). This unreadability comes from the distributed nature of the knowledge representation in the NN. Narazaki, Watanabe, Yamamoto (1996) propose a method that reorganizes the distributed knowledge in the NN to extract approximate classification rules. The rule extraction method is based on the analysis of the function that the NN has learned, rather than on the direct interpretation of connection weights as correlation information. More specifically, the method divides the input space into "monotonic regions" where a monotonic region is a set of input patterns that belongs to the same class with the same sensitivity pattern. Approximate classification rules are generated by projecting these monotonic regions.

Clustering is another important model fit in data mining. Clustering using the unsupervised self-organizing map has been successful. Vesanto et al. (2000) employ a step-wise strategy by partitioning the data with a self-organizing map (SOM), followed by its clustering. Alahakoon et al. (2000) perform hierarchical clustering of SOMs, based on a spread factor which is independent of the dimensionality of the data. Shalvi and DeClaris (1998) have designed a data mining technique, combining Kohonen's self-organizing neural network with data visualization, for clustering a set of pathological data containing information regarding the patients' drugs, topographies (body locations) and morphologies (physiological abnormalities). Kohonen et al. (2000) have demonstrated the utility of a huge self-organizing map (SOM) with more than one million nodes to partition a little less than seven million patent abstracts where the documents are represented by 500-dimensional feature vectors.

Many successful applications have been reported by using Neural Networks for classification. Shuxiang Xu, Ming Zhang (2005) proposed a new adaptive neural network model: a feed-forward neural network with a new activation function called neuron-adaptive activation function for financial analysis. Mingkun Li, Shuo Feng (2003) presented the preliminary results of a data mining study of a production line involving hundreds of variables related to mechanical, chemical, electrical and magnetic processes involved in manufacturing coated glass. The study was

The sections that follow is intended to provide an intuitive understanding of the two major neural networks (Back Propagation Network and Self Organizing Feature Map) proposed for learning from histogram and regression line data.

3.3 A Generic Back Propagation Histogram Network

A back propagation neural network uses a feed-forward topology, supervised learning, and the back propagation learning algorithm. This algorithm was responsible in large part for the reemergence of neural networks in the mid 1980s. Back propagation is a general purpose learning algorithm. It is powerful but also expensive in terms of computational requirements for training. A back propagation network with a single hidden layer of processing elements can model any continuous
function to any degree of accuracy (given enough processing elements in the hidden layer). There are literally hundreds of variations of back propagation in the neural network literature, and all claim to be superior to “basic” back propagation in one way or the other. Indeed, since back propagation is based on a relatively simple form of optimization known as gradient descent, mathematically astute observers soon proposed modifications using more powerful techniques such as conjugate gradient and Newton’s methods. However, “basic” back propagation is still the most widely used variant. Its two primary virtues are that it is simple and easy to understand, and it works for a wide range of problems.

The basic back propagation algorithm consists of three steps as shown in fig 3.1. The input pattern is presented to the input layer of the network. These inputs are propagated through the network until they reach the output units. This forward pass produces the actual or predicted output pattern. Because back propagation is a supervised learning algorithm, the desired outputs are given as part of the training vector. The actual network outputs are subtracted from the desired outputs and an error signal is produced. This error signal is then the basis for the back propagation step, whereby the errors are passed back through the neural network by computing the contribution of each hidden processing unit and deriving the corresponding adjustment needed to produce the correct output. The connection weights are then adjusted and the neural network has just “learned” from an experience.
As mentioned earlier, back propagation is a powerful and flexible tool for data modeling and analysis. In case of linear regression a back propagation network with no hidden units can be easily used to build a regression model relating multiple input parameters to multiple outputs or dependent variables. This type of back propagation network actually uses an algorithm called the delta rule, first proposed by Widrow and Hoff (1960). Adding a single layer of hidden units turns the linear neural network into a nonlinear one, capable of performing multivariate logistic regression, but with some distinct advantages over the traditional statistical technique. Using a back propagation network to do logistic regression allows us to model multiple outputs at the same time. Confounding effects from multiple input parameters can be captured in a single back propagation network model. Back propagation neural networks can be used for classification, modeling, and time-series forecasting.

For classification problems, the input attributes are mapped to the desired classification categories. The training of the neural network amounts to setting up the correct set of discriminate functions to correctly classify the inputs. For building models or function approximation, the input attributes are mapped to the function output. This could be a single output such as a pricing model, or it could be complex models with multiple outputs such as trying to predict two or more functions at once. In this section, we propose a Histogram Back Propagation Network which is capable of accepting histograms as input attributes and correspondingly map the sample/object characterized by the histogram into the desired classification category. The weight vectors which characterize the discriminant functions are also modeled as histograms. As demonstrated in chapter 2, with histograms possessing the distinctiveness of being a generic feature, a neural network model that is competent to handle histogram inputs could also be termed as a Generic Neural Network. The generic nature is demonstrated with a set of experiments.

### 3.3.1 Problem Statement

There are m samples in n-dimensional space. Each feature $f_i$ of sample $j$ is of symbolic distribution type (histogram), ie $f_{ij} = H$ where $1 \leq j \leq m$ and $1 \leq i \leq n$. It is required to define a neural network which would accept histogram inputs and get trained. Each histogram feature is constituted with $B$ number of bins.
For describing the computational details of histogram neural network let us consider a data space with data set D.

### Table 3.1 Histogram Symbolic Data Set

<table>
<thead>
<tr>
<th>Dataset D</th>
<th>F₁</th>
<th>F₂</th>
<th>...</th>
<th>Fₘ</th>
<th>O₁</th>
<th>O₂</th>
<th>...</th>
<th>Oₖ</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>H₁₁</td>
<td>H₁₂</td>
<td>...</td>
<td>H₁ₘ</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>S₂</td>
<td>H₂₁</td>
<td>H₂₂</td>
<td>...</td>
<td>H₂ₘ</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>S₃</td>
<td>H₃₁</td>
<td>H₃₂</td>
<td>...</td>
<td>H₃ₘ</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>S₄</td>
<td>H₄₁</td>
<td>H₄₂</td>
<td>...</td>
<td>H₄ₘ</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Sₙ₋₁</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Sₙ</td>
<td>Hₙ₁</td>
<td>Hₙ₂</td>
<td>...</td>
<td>Hₙₘ</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
</tr>
</tbody>
</table>

where \( \{S₁, S₂, S₃, ..., Sₙ\} \) are the samples \( \{F₁, F₂, ..., Fₘ\} \) are the histogram features, \( \{O₁, O₂, ..., Oₖ\} \) are the output class labels.

### 3.3.2 A Histogram Back propagation Feed Forward Network

The network under consideration has the general architecture as shown in fig 3.2. Although the present discussion applies to a histogram neural network with three layers of neurons, the algorithm is extendable to networks with a large number of layers. Input layer neurons are linear, whereas neurons in the hidden and output layers have sigmoidal signal functions.

![General Architecture of a Histogram Neural Network](image-url)
Based on the gradient descent based learning the basic procedure of back propagation algorithm is outlined as below (John Hertz, 1991):

i. Select a Sample $S_k$ from the training set $D$, and present it to the network.

ii. Compute activations and signals of input, hidden and output neurons in that sequence.

iii. Compute the error over the output neurons by comparing the generated outputs with the desired outputs.

iv. Use the error calculated in step (ii) to compute the change in the hidden to output layer weights, and the change in input to hidden layer weights including bias weights, such that a global error measure gets reduced.

v. Update all weights of the network in accordance with the changes computed in step iv.

\[
\text{Hidden to output layer weights} \\
W_{hj}^{k+1} = W_{hj}^k + \Delta W_{hj}^k
\]  

(3.1)

\[
\text{Input to hidden layer weights} \\
W_{ih}^{k+1} = W_{ih}^k + \Delta W_{ih}^k
\]  

(3.2)

where $\Delta W_{hj}^k$ and $\Delta W_{ih}^k$ are weight changes computed in step iv.

vi. Repeat steps (i) through (iv) until the global error falls below a predefined threshold.

Now we proceed to look into the computational aspect of the histogram neural network

### 3.3.3 Computational Aspect of Histogram NN

#### Computation of Neuron Signals

1. **Input Layer**

The input to the neural network is a vector of histograms $S_i = [H_{i1}, H_{i2}, \ldots, H_{ij}, \ldots, H_{iM}]$ where $M$ is the number of features. ‘$i$’ stands for $i^{th}$ sample and $j$ represents the $j^{th}$ feature. Thus the network consists of $M$ input neurons with optional bias neuron
A GENERIC NEURAL NETWORK MODEL FOR CLASSIFICATION AND CLUSTERING IN KNOWLEDGE MINING

whose input is set to 1. The basic assumption is that all input histograms consist of equal number of bins and let it be denoted as B. Let the signal from the kth input neuron with ith input sample be represented as \( \delta_k^i \).

2. Hidden Layer

Let there be K number of hidden neurons. The weight vectors \( W_{ih} \) between the input layer and the hidden layer neurons are also considered to be histograms with B number of bins. Thus the input to the hidden layer represented by \( \zeta_h^i \) where \('h' represents hth hidden layer neuron with \('i'\)th input signal, is given by

\[
\zeta_h^i = \sum_{i=1}^{M} w_{ih} \cdot \delta_k^i
\]

\[
= \sum_{i=1}^{M} w_{ih}(b) \cdot \delta_k^i(b) \text{ where } b = \{1,2,...,10\} \quad (3.3)
\]

Here we observe the simplification resulting out of histogram arithmetic where, the input-hidden layer weight histograms and the input signal histograms are multiplied along the corresponding bins resulting in another histogram which is the input to the hidden layer.

The output of the hidden layer represented by \( \xi_h^i \) (ith input sample and hth hidden neuron) is obtained with a sigmoidal function

\[
\xi_h^i = \frac{1}{1+\exp(-\zeta_h^i)} \quad (3.4)
\]

The exponential function with histogram as input would result in a histogram with exponential function applied on the individual elements of the input histogram. Thus \( \xi_h^i \) is a histogram signal from the hidden neurons.

3. Output Layer

Let there be C number of output neurons corresponding to C classes. The weight vectors \( W_{hc} \) between the hidden layer and the output layer neurons are also considered to be histograms with B number of bins. Thus the input to the output layer represented by \( \Theta_c^i \) (where \('c' represents cth output layer neuron with \('i'\)th input signal) is given by

\[
\Theta_c^i = \sum_{h=1}^{K} w_{hc} \cdot \xi_h^i
\]
A GENERIC NEURAL NETWORK MODEL FOR CLASSIFICATION AND CLUSTERING IN KNOWLEDGE MINING

\[ k = \sum_{h=1}^{K} w_{hc}(b) \ast f_h(b) \text{ where } b = \{1, 2, ..., 10\} \] (3.5)

Even here we observe the simplification resulting out of histogram arithmetic where, the hidden-output layer weight histograms and the hidden layer signal histograms are multiplied along the corresponding bins resulting in another histogram which is the input to the output layer.

The output of the output layer represented by \( \Omega^i_c \) (i\textsuperscript{th} input sample and c\textsuperscript{th} output neuron) is obtained with a sigmoidal function

\[ \Omega^i_c = \frac{1}{1+\exp(-\Theta^i_c)} \] (3.6)

The exponential function with histogram as input would result in a histogram with exponential function applied on the individual elements of the input histogram. Thus \( \Omega^i_c \) is a histogram signal from the output neurons.

**Computation of Deltas/Errors**

1. **Output Neuron**

   The actual output is denoted by \( \Omega^i_c \) and the desired output is single valued. The difference is computed by simple scalar addition/subtraction by considering histogram as an array or vector. For example the resultant of 1 - H is another histogram \( H^* \) obtained by subtracting all the individual components of the histogram H from 1. Similarly for 0 - H each individual element of H is subtracted from 0. Thus the error in the output neuron is computed as given below

   \[ E^i_c = d^i_c - \Omega^i_c \] (3.7)

   where \( d^i_c \) is the desired output in the c\textsuperscript{th} neuron and \( \Omega^i_c \) is the actual output in the c\textsuperscript{th} neuron. \( E^i_c \) is the error in the c\textsuperscript{th} output neuron. Thus \( E^i_c \) is also a histogram. Now this error histogram is scaled by the slope. The scaled error is represented by SE. Thus

   \[ SE^i_c = E^i_c \ast \Omega^i_c (1 - \Omega^i_c) \] (3.8)

   is a histogram based on the defined histogram arithmetic. Now the hidden-output layer weights are updated with \( \Delta w_{hc} \) where
\[ \Delta w_{hc} = \eta \ast SE_{c}^i \ast \Omega_c^i \]  

Thus \( \Delta w_{hc} \) is a weight update histogram.

2. Hidden Neuron

In order to calculate the extent of contribution of the \( h \) hidden neuron to the output error, scaled errors \( SE_{c}^i \) are back propagated through weights \( w_{hc} \). In other words the error contribution of the \( h \) hidden node is

\[ E_{h}^i = \sum_{c=1}^{C} SE_{c}^i \ast w_{hc} \]

implies

\[ E_{h}^i = \sum_{c=1}^{C} SE_{c}^i (b) \ast w_{hc}(b) \text{ where } b = \{1, 2, ..., 10\} \]  

where \( b \) represents the histogram bin.

Thus the scaled error histogram is obtained as

\[ SE_{h}^i = E_{h}^i \ast \mathcal{L}_h^i (1 - \mathcal{L}_h) \]  

Now the input-hidden layer histogram weights are updated with

\[ \Delta w_{ih} = \eta \ast SE_{h}^i \ast \delta_l \]  

As can be understood the computational procedure of histogram neural network demands correspondence between the bins i.e. the bins should be centered at the same position for all histograms. It can be achieved by first normalizing the values along each feature and then synthesizing the histogram symbolic data set. This helps us in maintaining the histogram spread between 0 and 1 for all histograms. This gives a straightforward approach to define a neural network that could process histogram data. The only constraint due to the requirement of bin correspondence is that all histograms have to span equal spread or equal number of bins. In our experimentation set up we have chosen histograms with 10 bins.

As the computational procedure proves, the bin correspondence in the architecture of a histogram neural network can be exploited to define a parallel architecture for faster processing. The parallel architecture demands for \( B \) number of neural networks with the \( k \) neural network learning the data corresponding to \( k \) bin of the input histogram. The weights of the \( k \) neural network correspond to the \( k \) bin of the histogram weight vectors defined previously. The class labels are duplicated \( B \) times.
for B neural networks. All B neural networks are independent neural networks without any interconnection between them.

3.3.4 Experimentation

Experimentation results: 700x3 Data

In order to demonstrate the working of the above formulations we have considered a few symbolic dataset synthesized from conventional data sets (a) 700x3 data (Generated by expanding the data table introduced in (Nagabhushan,1995;Nagabhushan,1998) and (b) iris data.

(a) The 700x3 is a data set with 700 samples and three features. This data is a conventional data with 5 classes where there is a clear overlap among the feature values requiring all three features for data modeling. This dataset consists of 700 samples with first 140 belonging to class I, 141-280 to class II, 281-420 to class III, 421 to 560 belonging to class IV and 561-700 to class V. Now in each class, the sample set is divided into seven sample packets with 20 samples in each packet. Histograms are generated for each feature in the sample packet. Thus for each class we obtain seven symbolic objects with histogram features. This results in 35 symbolic objects with three histogram features.

Figure 3.3 portrays the 700x3 Histogram dataset with 35 samples and 3 histogram features. The class labels are shown in table 3.2

<table>
<thead>
<tr>
<th>Samples</th>
<th>1 - 7</th>
<th>8 - 14</th>
<th>15 - 21</th>
<th>22 - 28</th>
<th>29 - 35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Labels</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C4</td>
<td>C5</td>
</tr>
<tr>
<td>1 - 7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8 - 14</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15 - 21</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22 - 28</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>29 - 35</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The initial weight vector histogram before training is shown in figure 3.4a, 3.5a and the weight vector histogram after training is shown in figure 3.4b, 3.5b. The goal was set to an error value of 0.001. The test vector histograms consisted of 10 samples with 2 samples belonging to each class. The test vector input is shown in figure 3.6
and the corresponding output histograms are shown in figure 3.7. The corresponding class labels from the output histogram are obtained by fixing a hard limiter. The hard limiter would assign the output to 1 if the area of the histogram (computed by adding all the bin values of the histogram) is more than a threshold $T$. In our experimentation $T = 5$. The hard limited output is shown in figure 3.8.

Fig 3.3a 700x3 Histogram dataset – Samples 1-12

Fig 3.3b 700x3 Histogram dataset – Samples 13-24
A GENERIC NEURAL NETWORK MODEL FOR CLASSIFICATION AND CLUSTERING IN KNOWLEDGE MINING

Fig 3.3c 700x3 Histogram dataset – Samples 25-35

Fig 3.3 700x3 Histogram dataset

Fig 3.4 a Before Training  Fig 3.4b After Training

Fig 3.4 Histogram Weight Vectors Between Input – Hidden Layer
A GENERIC NEURAL NETWORK MODEL FOR CLASSIFICATION AND CLUSTERING IN KNOWLEDGE MINING

Fig 3.5a Before Training  
Fig 3.5b After Training  
Fig 3.5 Histogram Weight Vectors Between Hidden - Output Layer

Fig 3.6 700x3 Input Test Data Set

Fig 3.7 700x3 Output Histograms for the corresponding Input Test Data Set
Experimentation results: IRIS Data

The iris dataset consists of 150 samples with 4 features. This data is again a conventional data with 3 classes namely setosa, versicolour and virgínica. The first 50 samples belong to class I, second 50 to class II and the third 50 to class III. Now from each class 30 samples were drawn randomly each time and histograms were generated for four features of the drawn samples. This random sampling was done 5 times with respect to each class and thus we obtain 15 symbolic objects with 4 histogram feature type. The sampling was tuned to cover all 50 samples belonging to each class. Even in this case the values were normalized along each feature to generate uniform histogram spreads between 0 and 1. The class labels are shown in table 3.3.

Table 3.3 IRIS Histogram Dataset Class Labels

<table>
<thead>
<tr>
<th>Samples</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6-10</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11-15</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Figure 3.9 depicts the 15 histogram symbolic data set with 15 samples and 4 histogram features. The weight vector histogram before and after training are shown in figure 3.10 and 3.11. The goal was set to an error value of 0.001. The test vector histograms consisted of 6 samples with 2 samples belonging to each class. The test vector input is shown in figure 3.12 and the corresponding output histograms are shown in figure 3.13. The corresponding class labels from the output histogram are obtained by fixing a hard limiter. The hard limiter would assign the output to 1 if the area of the histogram (computed by adding all the bin values of the histogram) is more than a threshold T. In this experimentation T = 5.

Fig 3.9 IRIS Histogram Data
A GENERIC NEURAL NETWORK MODEL FOR CLASSIFICATION AND CLUSTERING IN KNOWLEDGE MINING

Fig 3.10a Before Training
Fig 3.10b After Training
Fig 3.10 Histogram Weight Vectors Between Input – Hidden Layer

Fig 3.11a Before Training
Fig 3.11b After Training
Fig 3.11 Histogram Weight Vectors Between Hidden – Output Layer
3.4 Regression Line Neural Network

The Histogram Neural Network theory demonstrates the requirement of a bin-wise correspondence among the histograms which implies that for B number of bins we need B neural networks. Thus the computational complexity directly depends upon the number of bins. More the number of bins more parallelism is required. So in order to avoid this complexity we present a regression based neural network model which can be more generic than histogram NN with almost producing the same results. Obviously there will be a minor error factor which doesn’t affect the end result in anyway. In this section we illustrate the Regression Line Neural Network. Regression line NN has the advantage of less computational complexity over
histogram NN. It is because, corresponding to B number of bins B sets of NN will have to be constructed as indicated by the histogram NN theory and histogram arithmetic. But as defined in this section, only two sets of NN computation is good enough for Regression line NN.

3.4.1 Problem Statement
There are m samples in n-dimensional space. Each feature $f_i$ of sample $j$ is of symbolic regression line type, ie $f_{ij} = L$ where $1 \leq j \leq m$ and $1 \leq i \leq n$. It is required to define a neural network which would accept regression line inputs and get trained. Each regression line feature can be represented with \{slope, intercept\} or line coordinates \{[$x_1 \ x_2$] [$y_1 \ y_2$]\}.

Before describing the computational details of regression neural network let us consider a data space with data set $D$.

<table>
<thead>
<tr>
<th>Table 3.4 Regression Line Symbolic Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset $D$</td>
</tr>
<tr>
<td>$S_1$            $F_1$  $L_{11}$ $L_{12}$ ... $L_{1M}$ $O_1$ 0 ... $O_C$</td>
</tr>
<tr>
<td>$S_2$            $F_2$  $L_{21}$ $L_{22}$ ... $L_{2M}$ $O_1$ 0 ... $O_C$</td>
</tr>
<tr>
<td>$S_3$            $F_3$  $L_{31}$ $L_{32}$ ... $L_{3M}$ $O_1$ 1 ... $O_C$</td>
</tr>
<tr>
<td>$S_4$            $F_4$  $L_{41}$ $L_{42}$ ... $L_{4M}$ $O_1$ 1 ... $O_C$</td>
</tr>
<tr>
<td>...             ...  ...  ...  ...  ...  ...  ...  ...</td>
</tr>
<tr>
<td>$S_N$            $F_N$  $L_{N1}$ $L_{N2}$ ... $L_{NM}$ $O_1$ 0 0 ... $O_C$</td>
</tr>
</tbody>
</table>

where \{$S_1, S_2, S_3, ..., S_N$} are the samples \{$F_1, F_2, ..., F_M$} are the histogram features, \{$O_1, O_2, ..., O_C$} are the output class labels.

3.4.2 Computational Aspect of Regression Line NN

Computation of Neuron Signals

1. Input Layer

The input to the neural network is a vector of regression lines $S_i = [L_{i1} \ L_{i2} \ ... \ L_{ij} \ ... \ L_{iM}]$ where $M$ is the number of features. ‘i’ stands for $i^{th}$ sample and $j$ represents the $j^{th}$ feature. Thus the network consists of $M$ input neurons with optional
bias neuron whose input is set to 1. The basic assumption is that all input regression lines is represented by line co-ordinate vectors \((X,Y)\). Let the signal from the \(k^{th}\) input neuron with \(i^{th}\) input be represented as \(\delta_k^i\).

2. Hidden Layer

Let there be \(K\) number of hidden neurons. The weight vectors \(w_{ih}\) between the input layer and the hidden layer neurons are also considered to be regression lines. Thus the input to the hidden layer represented by \(\zeta_h^i\) where 'h' represents \(h^{th}\) hidden layer neuron with 'i' input signal is given by

\[
\zeta_h^i = \sum_{i=1}^{M} w_{ih} \cdot \delta_k^i
\]

\[
= \sum_{i=1}^{M} w_{ih}(X) \cdot \delta_k^i(x)
\]

for \(x = \{x_{min} \text{ at } y_1 = 0 \text{ and } x_{max} \text{ at } y_2 = 1\} \) (3.13)

Here we observe the simplification resulting out of regression line arithmetic where, the input-hidden layer weight regression lines and the input signal regression lines are multiplied along the corresponding \(x_{min}, x_{max}\) which are the co-ordinate values of \(X\) at \(y_1 = 0\) and \(y_2 = 1\). This results in another regression line which is the input to the hidden layer.

The output of the hidden layer represented by \(\xi_h^i\) (\(i^{th}\) input sample and \(h^{th}\) hidden neuron) is obtained with a sigmoidal function

\[
\xi_h^i = 1/(1+\exp(-\zeta_h^i))
\]

(3.14)

The exponential function with regression line as input would result in a regression line with exponential function applied on \(\{x_{min}, x_{max}\}\) values of the input regression line with \(y_1\) and \(y_2\) fixed. Thus \(\xi_h^i\) is a regression line signal from the hidden neurons.

3. Output Layer

Let there be \(C\) number of output neurons corresponding to \(C\) classes. The weight vectors \(w_{hc}\) between the hidden layer and the output layer neurons are also considered to be regression lines. Thus the input to the output layer represented by \(\Theta_c^i\) (where 'c' represents \(c^{th}\) output layer neuron with 'i' input signal) is given by
\[ \Theta_c = \sum_{h=1}^{K} w_{hc} \ast \mathcal{L}_h \]

\[ = \sum_{h=1}^{K} w_{hc}(x) \ast \mathcal{L}_h(x) \]

for \( x = \{x_{\text{min}} \text{ at } y_1 = 0 \text{ and } x_{\text{max}} \text{ at } y_2 = 1\} \) \hspace{1cm} (3.15)

Even here we observe the simplification resulting out of regression line arithmetic where, the hidden-output layer weight regression lines and the hidden layer signal regression lines are multiplied along the corresponding \( \{x_{\text{min}}, x_{\text{max}}\} \) resulting in another regression line which is the input to the output layer.

The output of the output layer represented by \( \Omega_c^i \) (\( i^{th} \) input sample and \( c^{th} \) output neuron) is obtained with a sigmoidal function

\[ \Omega_c^i = \frac{1}{1+\exp(-\Theta_c^i)} \] \hspace{1cm} (3.16)

The exponential function with regression line as input would result in a regression line with exponential function applied on the \( \{x_{\text{min}}, x_{\text{max}}\} \) of the input regression line. Thus \( \Omega_c^i \) is a regression line signal from the output neurons.

**Computation of Deltas/Errors**

1. **Output Neuron**

The actual output is denoted by \( \Omega_c^i \) and the desired output is single valued. The difference is computed by simple scalar addition/subtraction by considering regression line as an \([x_{\text{min}}, x_{\text{max}}]\) vector at \([y_1, y_2]\) correspondingly. For example the resultant of \( 1 - L \) is another regression line \( L^* \) obtained by subtracting \([x_{\text{min}}, x_{\text{max}}]\) by 1 and \( y_1 \) and \( y_2 \) fixed to 0 and 1 respectively. Similarly for \( 0 - L \) each individual element of \( L \) (i.e.\([x_{\text{min}}, x_{\text{max}}]\)) is subtracted from 0. Thus the error in the output neuron is computed as given below

\[ E_c^i = d_c^i - \Omega_c^i \] \hspace{1cm} (3.17)

Where \( d_c^i \) is the desired output in the \( c^{th} \) neuron and \( \Omega_c^i \) is the actual output in the \( c^{th} \) neuron. \( E_c^i \) is the error in the \( c^{th} \) output neuron. Thus \( E_c^i \) is also a regression line. Now this error regression line is scaled by the slope. The scaled error is represented by \( SE \). Thus
is a regression line based on the defined regression arithmetic. Now the hidden-output layer weights are updated with $\Delta w_{hc}$ where

$$\Delta w_{hc} = \eta \ast SE_c^i \ast \Omega_c^i$$

Thus $\Delta w_{hc}$ is a weight update regression line.

### 2. Hidden Neuron

In order to calculate the extent of contribution of the $h^{th}$ hidden neuron to the output error, scaled errors $SE_c^i$ are back propagated through weights $w_{hc}$. In other words the error contribution of the $h^{th}$ hidden node is

$$E_h^i = \sum_{c=1}^{C} SE_c^i \ast w_{hc}$$

implies

$$E_h^i = \sum_{c=1}^{C} SE_c^i(x) \ast w_{hc}(x)$$

for $x = \{x_{min} \ast y_1 = 0 \text{ and } x_{max} \ast y_2 = 1\}$

Thus the scaled error regression line is obtained as

$$SE_h^i = E_h^i \ast \mathcal{L}_h^i (1 - \mathcal{L}_h^i)$$

Now the input-hidden layer regression line weights are updated with

$$\Delta w_{ih} = \eta \ast SE_h^i \ast \delta_l$$

As can be understood the computational procedure of regression neural network overcomes the demand for correspondence between the bins. Regression based approach is more generic by helping us in dealing with histograms of varied spread i.e Histograms with any number of bins can be represented by a corresponding regression line. This gives a straightforward approach to define a neural network that could process symbolic data.

The computational procedure proves that the whole mechanism depends on only two x-coordinate values by fixing $y_1$ and $y_2$ as illustrated or with two y-coordinate values by fixing $x_1$ and $x_2$. The architecture of regression line neural network can be exploited to define a parallel architecture for faster processing. The parallel architecture demands for 2 parallel neural networks with the 1st neural network...
learning the data corresponding to xmin of the input regression lines and the second learning the input corresponding to xmax. Thus the two neural networks are independent neural networks without any interconnection between them.

3.4.3 Experimentation

Experimentation results: 700x3 Data

Figure 3.14 shows the 700x3 Regression Line dataset with 35 samples and 3 Regression Line features. The class labels are shown in table 3.5

<table>
<thead>
<tr>
<th>Samples</th>
<th>Class Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1</td>
</tr>
<tr>
<td>1-7</td>
<td>1</td>
</tr>
<tr>
<td>8-14</td>
<td>0</td>
</tr>
<tr>
<td>15-21</td>
<td>0</td>
</tr>
<tr>
<td>22-28</td>
<td>0</td>
</tr>
<tr>
<td>29-35</td>
<td>0</td>
</tr>
</tbody>
</table>

The initial weight vector regression lines before training is shown in figures 3.15a,3.16a and the weight vector regression lines after training is shown in figures 3.15b,3.16b. The goal was set to an error value of 0.001. The test vector regression lines consisted of 10 samples with 2 samples belonging to each class. The test vector input is shown in figure 3.17 and the corresponding output regression lines are shown in figure 3.18. The corresponding class labels from the output regression lines are obtained by fixing a hard limiter. The hard limiter would assign the output to 1 if the sum of xmin and xmax (xmin is the x-value corresponding to y = 0 and xmax is the x value corresponding to y = 1) is more than a threshold T. In our experimentation T = 1. The hard limited output is shown in figure 3.19.
A GENERIC NEURAL NETWORK MODEL FOR CLASSIFICATION AND CLUSTERING IN KNOWLEDGE MINING

Fig 3.14a 700x3 Regression Line Data: 1 to 12

Fig 3.14b 700x3 Regression Line Data 13 to 24

Fig 3.14c 700x3 Regression Line Data 25 to 35

Fig 3.14 700x3 Regression Line Data
A GENERIC NEURAL NETWORK MODEL FOR CLASSIFICATION AND CLUSTERING IN KNOWLEDGE MINING

Fig 3.15a Before Training  Fig 3.15b After Training
Fig 3.15 Regression Line Weight Vectors Between Input-Hidden Layer

Fig 3.16a Before Training  Fig 3.16b After Training
Fig 3.16 Regression Line Weight Vectors Between Hidden – Output Layer

Fig 3.17 700x3 Regression Line Input Test Data Set
Fig 3.18 700x3 Output Regression Line for the Corresponding Input Test Data Set

Fig 3.19 Hard limited output of 700x3 output regression lines

Experimentation results: IRIS Data

Figure 3.20 depicts the 15 regression line symbolic data set with 4 regression line features. The initial weight vector regression lines before training and after training is
shown in figures 3.21a, 3.22a and figures 3.21b, 3.22b respectively. The goal was set to an error value of 0.001. The test vector regression lines consisted of 6 samples with 2 samples belonging to each class. The test vector input is shown in figure 3.23 and the corresponding output regression lines are shown in figure 3.24. The corresponding class labels from the output regression lines are obtained by fixing a hard limiter as before.

Table 3.6 - IRIS Regression Line Dataset Class Labels

<table>
<thead>
<tr>
<th>Samples</th>
<th>Class Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1</td>
</tr>
<tr>
<td>1-5</td>
<td>1</td>
</tr>
<tr>
<td>6-10</td>
<td>0</td>
</tr>
<tr>
<td>11-15</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig 3.20 IRIS Regression Line Data
Fig 3.21 a Before Training  
Fig 3.21b After Training
Fig 3.21 Regression Line Weight Vectors Between Input-Hidden Layer

Fig 3.22a Before Training  
Fig 3.22b After Training
Fig 3.22 Regression Line Weight Vectors Between Hidden - Output Layer
3.5 Generic Symbolic Neural Network

Now having defined the Histogram/Regression NN and also with the possibility of Histograms/Regression lines as a generic representation of other symbolic data the plausibility of overcoming the curse of generic dimensions through Histogram NN and Regression Line NN is explored. So along with the above demonstration with histogram data and regression data, a bunch of experiments by modeling a set of single valued data, interval data and multi-valued data has been conducted to prove
Histogram and Regression NN as a prospective mechanism to model a generic
database by overcoming the curse of generic dimensions. The experimentation results
pertaining to Single Valued and Interval Valued are presented here.

3.5.1 Experimentation I – Single Valued Data
We have sampled the 700x3 data to create a 35x3 single valued dataset. The dataset
was normalized along each feature. Thus the histogram was defined over the spread 0
to 1 with 10 bins. The single valued data is converted to histogram data by making
the bin into which each value falls as 1 and the remaining 0. So the histogram is a
binary histogram where only the bin into which the single value falls will be 1. Thus
we generate a 35x3 histogram dataset and the corresponding regression dataset as
shown in figure 2.44 of chapter 2.

After training the histogram and regression data with the corresponding neural
network, we test the outcome with few test vectors of both (histogram and
regression) types. The classification results with histogram neural network are shown
in figure 3.25 and figure 3.26. The classification results with regression line neural
network are shown in figures 3.27 and 3.28.

Fig 3.25 700x3 Single Valued Histogram Input Test Data Set
A GENERIC NEURAL NETWORK MODEL FOR CLASSIFICATION AND CLUSTERING IN KNOWLEDGE MINING

![Fig 3.26 700x3 Single Valued Output Histogram for the Corresponding Input Test Data Set](image1)

![Fig 3.27 700x3 Single Valued Regression Line Input Test Data Set](image2)
3.5.2 Experimentation II – Interval Valued Data
Again in this experiment we consider a variant of the 700x3 data representing intervals. This is an interval dataset with 35 samples and 3 interval features (Nagabhushan, 1996). The dataset is first normalized by considering the maximum and minimum along the corresponding features. Now the histograms are defined over the spread 0 to 1 with 10 bins. The interval valued data is converted into histogram data by making the bins into which the interval falls as 1 and the remaining bins as 0. So the histogram is a binary histogram where only the bins into which the interval falls are 1. Thus we generate a 35x3 histogram dataset and the corresponding regression dataset as shown in figure 2.49 of chapter 2. After training the histogram and regression data with the corresponding neural network, we test the outcome with few test vectors of both (histogram and regression) types. The classification results with histogram neural network are shown in figure 3.29 and figure 3.30. The classification results with regression line neural network are shown in figures 3.31 and 3.32.
A GENERIC NEURAL NETWORK MODEL FOR CLASSIFICATION AND CLUSTERING IN KNOWLEDGE MINING

Fig 3.29 700x3 Interval Valued Histogram Input Test Data Set

Fig 3.30 700x3 Interval Valued Output Histogram for the Corresponding Input Test Data Set
3.6 A Generic Kohonen Feature Maps

Kohonen feature maps are feed-forward networks that use an unsupervised training algorithm, and through a process called self-organization, configure the output units into a topological or spatial map. Kohonen’s (1988) work was reevaluated during the late 1980s, and the utility of the self-organizing feature map was recognized. A feature map neural network consists of two layers of processing units with the input
layer fully connected to a competitive output layer. There are no hidden units. When an input pattern is presented to the feature map, the units in the output layer compete with each other for the right to be declared the winner. The winning output unit is typically the unit whose incoming connection weights are the closest to the input pattern (in terms of Euclidean distance). Thus the input is presented and each output unit computes its closeness or match score to the input pattern. The output that is deemed closest to the input pattern is declared the winner and so earns the right to have its connection weights adjusted. The connection weights are moved in the direction of the input pattern by a factor determined by a learning rate parameter. This is the basic nature of competitive neural networks.

The Kohonen feature map creates a topological mapping by adjusting not only the winner’s weights, but also adjusting the weights of the adjacent output units in close proximity or in the neighborhood of the winner. So not only does the winner get adjusted, but the whole neighborhood of output units gets moved closer to the input pattern. Starting from randomized weight values, the output units slowly align themselves such that when an input pattern is presented, a neighborhood of units responds to the input pattern. As training progresses, the size of the neighborhood radiating out from the winning unit is decreased. Initially large numbers of output units will be updated, and later on smaller and smaller numbers are updated until at the end of training only the winning unit is adjusted. Similarly, the learning rate will decrease as training progresses, and in some implementations, the learn rate decays with the distance from the winning output unit.

From a data mining perspective, two sets of useful information are available from a trained feature map. Similar customers, products, or behaviors are automatically clustered together or segmented so that marketing messages can be targeted at homogeneous groups. The information in the connection weights of each cluster defines the typical attributes of an item that falls into that segment. This information lends itself to immediate use for evaluating what the clusters mean. When combined with appropriate visualization tools and/or analysis of both the population and segment statistics, the makeup of the segments identified by the feature map can be analyzed and turned into valuable business intelligence.
3.6.1 Histogram Feature Map Algorithm

The operational summary of a histogram SOFM algorithm is illustrated as below.

**Input:** A stream of training vectors \( \{ X_k \}_{k=1}^Q \) drawn uniformly from a possibly unknown probability distribution \( p(X) \). The training vectors are samples with histogram features. Let us consider histograms with \( B \) number of bins.

**Initialize:**
- Weights \( W_{ij}(0) \) to some small random histogram vectors with \( B \) number of bins.
- Value of the neighborhood \( N^k \)
- Learning rate \( \eta_0 \)

**Iteration:** Repeat

\[
\begin{align*}
\text{Selection:} & \quad \text{Pick a histogram sample } X_k \\
\text{Similarity Matching:} & \quad \text{Find the winning neuron } (I, J). \text{ The } AB \\
& \quad \text{regression based histogram distance measure introduced in section } \\
& \quad 2.3.6 \text{ is used for computing the winning neuron.} \\
\text{Adaptation:} & \quad \text{Update synaptic vectors of ONLY the winning cluster.} \\
& \quad w_{ij}^{k+1}(b) = w_{ij}^k(b) + \eta_k (x_i^k(b) - w_{ij}^k(b)) \quad i, j \in N^k \\
& \quad \text{where } b = \{1,2,3 \ldots B\} \quad (3.23)
\end{align*}
\]
Update: Update $\eta_k$, $N^k_u$

} until (there is no observable change in the map)

3.6.2 Regression Line Feature Map Algorithm
The operational summary of a regression line SOFM algorithm is illustrated as below

**Input:** A Stream of training vectors \{X_k\}_{k=1}^Q drawn uniformly from a possibly unknown probability distribution $p(X)$. The training vectors are samples with regression line features.

**Initialize:** Weights $W_{ij(0)}$ to some small random regression line vectors $(x_{\text{min}}, x_{\text{max}})$. Value of the neighborhood $N^k_u$ Learning rate $\eta_0$

**Iteration:** Repeat

{  

**Selection:** Pick a regression line sample $X_k$

**Similarity Matching:** Find the winning neuron $(I, J)$. The AB regression distance measure introduced in section 2.3.6 is used for computing the winning neuron.

**Adaptation:** Update synaptic vectors of ONLY the winning cluster.

$$w^{k+1}_{i,j}(x) = w^k_{i,j}(x) + \eta_k (x^k_i(x) - w^k_{i,j}(x)) \quad i,j \in N^k_u$$

for $x = \{x_{\text{min}} \text{ at } y_1 = 0 \text{ and } x_{\text{max}} \text{ at } y_2 = 1\}$ (3.24)

**Update:** Update $\eta_k$, $N^k_u$

} until (there is no observable change in the map)

3.6.3 Experimentation results
The above SOFM model was implemented and simulated on MATLAB 7. The input dataset considered are the IRIS and 700x3 shown in fig 3.3/3.13 and fig 3.8/3.18 respectively. The cluster results obtained using histogram and regression line SOFM are tabulated in table 3.7 and 3.8.
Table 3.7: 700x3 Dataset

<table>
<thead>
<tr>
<th>Cluster 1</th>
<th>{1, 2, 3, 4, 5, 6, 7}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 2</td>
<td>{8, 9, 10, 11, 12, 13, 14}</td>
</tr>
<tr>
<td>Cluster 3</td>
<td>{15, 16, 17, 17, 18, 19, 20, 21}</td>
</tr>
<tr>
<td>Cluster 4</td>
<td>{22, 23, 24, 25, 26, 27, 28}</td>
</tr>
<tr>
<td>Cluster 5</td>
<td>{29, 30, 31, 32, 33, 34, 35}</td>
</tr>
</tbody>
</table>

Table 3.8: IRIS Dataset

<table>
<thead>
<tr>
<th>Cluster 1</th>
<th>{1, 2, 3, 4, 5}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 2</td>
<td>{6, 7, 8, 9, 10}</td>
</tr>
<tr>
<td>Cluster 3</td>
<td>{11, 12, 13, 14, 15}</td>
</tr>
</tbody>
</table>

3.6 Few Interesting Interpretation

The Back propagation histogram/regression network and the histogram/regression SOFM could thus be used as generic network models to accept single valued data, interval data, multi-valued data, histogram data, regression data and so on. The generic nature of the network model could also be exploited for inter operability. Inter-operability means, say, the network could be trained with the histogram features of a data set and can be used for classifying a test sample with single valued feature or an interval valued feature. As demonstrated in chapter 2 we have clearly shown the technique for converting single valued, interval valued, multi valued features into histograms. And also, the methodology for deriving histograms from chunks/groups of samples is demonstrated. An interesting interpretation is derived from this demonstration.

**Interpretation I:**

In case of supervised learning the data set is broken into chunks/groups of samples. The corresponding histogram of each chunk/group is derived. The chunking and
generating the corresponding histogram dataset is performed on the entire data set. Now the histogram neural network is used to learn from the histogram data set. Let us assume that we need to classify a sample with single valued feature. The single valued features can be transformed into histogram features and can be classified with the already trained histogram network. As indicated earlier histogram network can be trained in a parallel mode which could reduce the training time drastically. On the other hand the computation can also be further optimized by converting the input data set into regression features. The experimentation results provided in section 3.3.5 and section 3.4.3 demonstrate this interpretation. In other words we learn from the characteristics (represented by histograms / regression lines) of the data set rather than the data itself. The network which has learnt from the data characteristic is now used to classify the test sample data.

**Interpretation II:**

The histogram network which has learnt from histogram dataset can also be used for classifying chunks of data belonging to same class. For example, let us assume that the group/chunk of test samples belong to the same class. i.e the fact that the test samples belong to same class is known but the actual class to which they belong is unknown. Now it may result in two situations

(i) The number of samples in the group is more: Then the histogram of the corresponding chunk can be derived and could be classified using the histogram network.

(ii) The number of samples in the group is small: Then the range of each feature can be derived to obtain an interval test sample. The interval test sample can be transformed into histogram and then classified using histogram network.

It can always be argued that a single test sample TS1 can be classified and by virtue of other samples belonging to the same class of TS1, will also fall into the same class. But by virtue of the histogram network having learnt from the characteristics of chunks/groups, the accuracy of classification with the characteristics of the test group will be high compared to that of classifying a single valued data.
3.7 Conclusion
Apart from being generic, histograms are also frequently derived features in case of image and temporal data analysis. Histogram features are always better representative features than a single valued feature in case of huge sampled data like temporal or image data. Neural Network learning models for dataset represented by histogram features is a novel contribution of this chapter. Most of the time the computational complexity of learning from such histogram based dataset is intensive. And that is one of the reasons why neural network models have not been prominently used for learning from image/temporal dataset. This chapter introduced the regression based neural network models through which such computational complexities can be drastically reduced. The application of these novel contributions are demonstrated in chapters 7 and 8 through multi channel signal data mining and image mining.

Chapter 2 and Chapter 3 demonstrated the generic nature of histogram and regression feature type. The chapters also illustrated a dimensionality reduction model (histogram/regression PCA), classification model (back propagation histogram/regression network) and a clustering model (histogram/regression SOFM). The demonstration and illustrations were strongly supported with experiments to handle many feature type (single valued, interval valued, multi-valued etc) transformed into the generic histogram/regression features. The other interesting interpretations of the models were also discussed. Thus both the chapters were dedicated to propose generic models for conventional object features and symbolic features.

Now on, Chapter 4 and chapter 5 widely discuss about the temporal data. These chapters discuss few interesting models for mining temporal data using wavelet transform and its variant WaveSim transform. They suggest techniques to extract histogram features from temporal data. In chapter 6 and 7 these histogram features are used further to model multi-channel signal data set in which the feature itself is a signal/temporal data.