

# CHAPTER 1

## Introduction

### 1.1 Introduction

One of the data mining problems is classification. Various classification algorithms have been designed to tackle the problem by researchers in different fields such as mathematical programming, machine learning, and statistics. Recently, there is a surge of data mining research in the database community. The classification problem is re-examined in the context of large databases. Unlike researchers in other fields, database researchers pay more attention to the issues related to the volume of data. They are also concerned with the effective use of the available database techniques, such as efficient data retrieval mechanisms. With such concerns, most algorithms proposed are basically based on decision trees. The general impression is that the neural networks are not well suited for data mining. The major criticisms include the following:

- 1) Neural networks learn the classification rules by many passes over the training data set so that the learning time of a neural network is usually long.
- 2) A neural network is usually a layered graph with the output of one node feeding into one or many other nodes in the next layer. The classification process is buried in both the structure of the graph and the weights assigned to the links between the nodes. Articulating the classification rules becomes a difficult problem.
- 3) For the same reason, available domain knowledge is rather difficult to be incorporated to a neural network. On the other hand, the use of neural networks in classification is not uncommon in machine learning community. In some cases, neural networks give a lower classification error rate than the decision trees but require longer learning time. In this project, we present our results

from applying neural networks to mine classification rules for large databases with the focus on articulating the classification rules represented by neural networks.

## **1.2 Aim and Objective**

The contributions of our study include the following:

Different from previous research work that excludes the neural network based approaches entirely, we argue that those approaches should have their position in data mining because of its merits such as low classification error rates and robustness to noise. With our rule extraction algorithms, symbolic classification rules can be extracted from a neural network. The rules usually have a comparable classification error rate to those generated by the decision tree based methods. For a data set with a strong relationship among attributes, the rules extracted are generally more concise.

A data mining system based on neural networks is developed. The system successfully solves a number of classification problems in the literature.

## **1.3 Scope**

The project will enable us to predict things which are required for many applications like following.[8]

- **Biological Data Analysis:** In recent times, we have seen a tremendous growth in the field of biology such as genomics, proteomics, functional Genomics and biomedical research. Biological data mining is a very important part of Bioinformatics.
- **Telecommunication Industry:** Today the telecommunication industry is one of the most emerging industries providing various services such as fax, pager, cellular phone, internet messenger, images, e-mail, web data transmission, etc. Due to the development of new computer and communication technologies, the telecommunication industry is rapidly expanding. This is the reason why data mining is become very important to help and understand the business. Data mining in telecommunication industry helps in identifying the telecommunication patterns, catch fraudulent activities, make better use of resource, and improve quality of service.

- Retail Industry: Data Mining has its great application in Retail Industry because it collects large amount of data from on sales, customer purchasing history, goods transportation, consumption and services. It is natural that the quantity of data collected will continue to expand rapidly because of the increasing ease, availability and popularity of the web. Data mining in retail industry helps in identifying customer buying patterns and trends that lead to improved quality of customer service and good customer retention and satisfaction.

## **CHAPTER 2**

### **Review of literature**

#### **2.1 Domain Explanation**

Data mining (the analysis step of the "Knowledge Discovery in Databases" process, or KDD), an interdisciplinary subfield of computer science, is the computational process of discovering patterns in large data sets ("big data") involving methods at the intersection of artificial intelligence, machine learning, statistics, and database systems. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use. Aside from the raw analysis step, it involves database and data management aspects, data pre-processing, model and inference considerations, interestingness metrics, complexity considerations, post-processing of discovered structures, visualization, and online updating.[7]

The term is a misnomer, because the goal is the extraction of patterns and knowledge from large amount of data, not the extraction of data itself. It also is a buzzword and is frequently applied to any form of large-scale data or information processing (collection, extraction, warehousing, analysis, and statistics) as well as any application of computer decision support system, including artificial intelligence, machine learning, and business intelligence. The popular book "Data mining: Practical machine learning tools and techniques with Java" (which covers mostly machine learning material) was originally to be named just "Practical machine learning", and the term "data mining" was only added for marketing reasons. Often the more general terms "(large scale) data analysis", or "analytics" – or when referring to actual methods, artificial intelligence and machine learning – are more appropriate.[7]

The actual data mining task is the automatic or semi-automatic analysis of large quantities of data to extract previously unknown, interesting patterns such as groups of data records (cluster analysis), unusual records (anomaly detection), and dependencies (association rule mining). This usually involves using database techniques such as spatial indices. These patterns can then be seen as a kind of summary of the input data, and may be used in further analysis or, for example, in machine learning and predictive analytics. For example, the data mining step might identify multiple groups in the data, which can then be used to obtain more accurate prediction results by a decision support system. Neither the data collection, data preparation, nor result interpretation and reporting are part of the data mining step, but do belong to the overall KDD process as additional steps.[7]

The related terms data dredging, data fishing, and data snooping refer to the use of data mining methods to sample parts of a larger population data set that are (or may be) too small for reliable statistical inferences to be made about the validity of any patterns discovered. These methods can, however, be used in creating new hypotheses to test against the larger data populations.[7]

Data mining involves six common classes of tasks:

- Anomaly detection (Outlier/change/deviation detection) – The identification of unusual data records, that might be interesting or data errors that require further investigation.
- Association rule learning (Dependency modelling) – Searches for relationships between variables. For example, a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis.
- Clustering – is the task of discovering groups and structures in the data that are in some way or another "similar", without using known structures in the data.
- Classification – is the task of generalizing known structure to apply to new data. For example, an e-mail program might attempt to classify an e-mail as "legitimate" or as "spam".
- Regression – attempts to find a function which models the data with the least error.
- Summarization – providing a more compact representation of the data set, including visualization and report generation.

## 2.2 EXISTING SOLUTION

Systems that construct classifiers are one of the commonly used tools in data mining. Such systems take as input a collection of cases, each belonging to one of a small number of classes and described by its values for a fixed set of attributes, and output a classifier that can accurately predict the class to which a new case belongs.

These notes describe C4.5, a descendant of CLS and ID3. Like CLS and ID3, C4.5 generates classifiers expressed as decision trees, but it can also construct classifiers in more comprehensible ruleset form [3].

### 2.2.1 Decision trees

Given a set  $S$  of cases, C4.5 first grows an initial tree using the divide-and-conquer algorithm as follows:

- If all the cases in  $S$  belong to the same class or  $S$  is small, the tree is a leaf labeled with the most frequent class in  $S$ .
- Otherwise, choose a test based on a single attribute with two or more outcomes. Make this test the root of the tree with one branch for each outcome of the test, partition  $S$  into corresponding subsets  $S_1, S_2, \dots$  according to the outcome for each case, and apply the same procedure recursively to each subset.

There are usually many tests that could be chosen in this last step. C4.5 uses two heuristic criteria to rank possible tests: information gain, which minimizes the total entropy of the subsets  $\{S_i\}$  (but is heavily biased towards tests with numerous outcomes), and the default gain ratio that divides information gain by the information provided by the test outcomes.

Attributes can be either numeric or nominal and this determines the format of the test outcomes. For a numeric attribute  $A$  they are  $\{A \leq h, A > h\}$  where the threshold  $h$  is found by sorting  $S$  on the values of  $A$  and choosing the split between successive values that maximizes the criterion above. An attribute  $A$  with discrete values has by default one outcome for each value, but an option allows the values to be grouped into two or more subsets with one outcome for each subset.[3][4]

The initial tree is then pruned to avoid overfitting. The pruning algorithm is based on a pessimistic estimate of the error rate associated with a set of  $N$  cases,  $E$  of which do not belong to the most frequent class. Instead of  $E/N$ , C4.5 determines the upper limit of the binomial probability when  $E$  events have been observed in  $N$  trials, using a user-specified confidence whose default value is 0.25.[3][4]

Pruning is carried out from the leaves to the root. The estimated error at a leaf with  $N$  cases and  $E$  errors is  $N$  times the pessimistic error rate as above. For a subtree, C4.5 adds the estimated errors of the branches and compares this to the estimated error if the subtree is replaced by a leaf; if the latter is no higher than the former, the subtree is pruned. Similarly, C4.5 checks the estimated error if the subtree is replaced by one of its branches and when this appears beneficial the tree is modified accordingly. The pruning process is completed in one pass through the tree.[3][4]

### **2.2.2 Rule set classifiers**

Complex decision trees can be difficult to understand, for instance because information about one class is usually distributed throughout the tree. C4.5 introduced an alternative formalism consisting of a list of rules of the form “if A and B and C and ... then class X”, where rules for each class are grouped together. A case is classified by finding the first rule whose conditions are satisfied by the case; if no rule is satisfied, the case is assigned to a default class.[3]

C4.5 rule sets are formed from the initial (unpruned) decision tree. Each path from the root of the tree to a leaf becomes a prototype rule whose conditions are the outcomes along the path and whose class is the label of the leaf. This rule is then simplified by determining the effect of discarding each condition in turn. Dropping a condition may increase the number  $N$  of cases covered by the rule, and also the number  $E$  of cases that do not belong to the class nominated by the rule, and may lower the pessimistic error rate determined as above. A hill-climbing algorithm is used to drop conditions until the lowest pessimistic error rate is found. To complete the process, a subset of simplified rules is selected for each class in turn. These class subsets are ordered to minimize the error on the training cases and a default class is chosen. The final ruleset usually has far fewer rules than the number of leaves on the pruned decision tree.[3]

The principal disadvantage of C4.5's rule sets is the amount of CPU time and memory that they require. In one experiment, samples ranging from 10,000 to 100,000 cases were drawn from a large dataset. For decision trees, moving from 10 to 100K cases increased CPU time on a PC from 1.4 to 61 s, a factor of 44. The time required for rule sets, however, increased from 32 to 9,715 s, a factor of 300.[4]

## **2.3 H/W and S/W Requirements**

### ➤ **HARDWARE REQUIREMENTS**

- CPU:-Intel 3.0 GHz and above
- RAM:-4GB and above
- STORAGE:- 1GB and above

### ➤ **SOFTWARE REQUIREMENTS**

- Windows 7/XP/8/8.1 32bit/64 bit
- Eclipse Development Platform
- Java Development Kit 1.6



## **CHAPTER 3**

### **Analysis**

#### **3.1 Functional Requirements**

The system will classify the data from the given database. This more accurate classified data can be used for prediction like weather prediction, statistics, employees salary distribution, medical diagnosis etc.

#### **3.2 Non-Functional Requirements**

##### **3.2.1 Reliability and accuracy**

This approach is very reliable and accurate as compared to traditional data mining approach. If traditional system requires 10 classification rules then this approach will require only 2-3 classification rules. It has low classification error rates and robustness to noise.

##### **3.2.2 Scalability**

With rule extraction algorithm, symbolic classification rules can be extracted from neural network. The rules usually have a comparable classification error rate to those generated by the decision tree based methods. For a data set with strong relationship among attributes, the rules extracted are generally more concise.

### 3.2.3 Performance issues

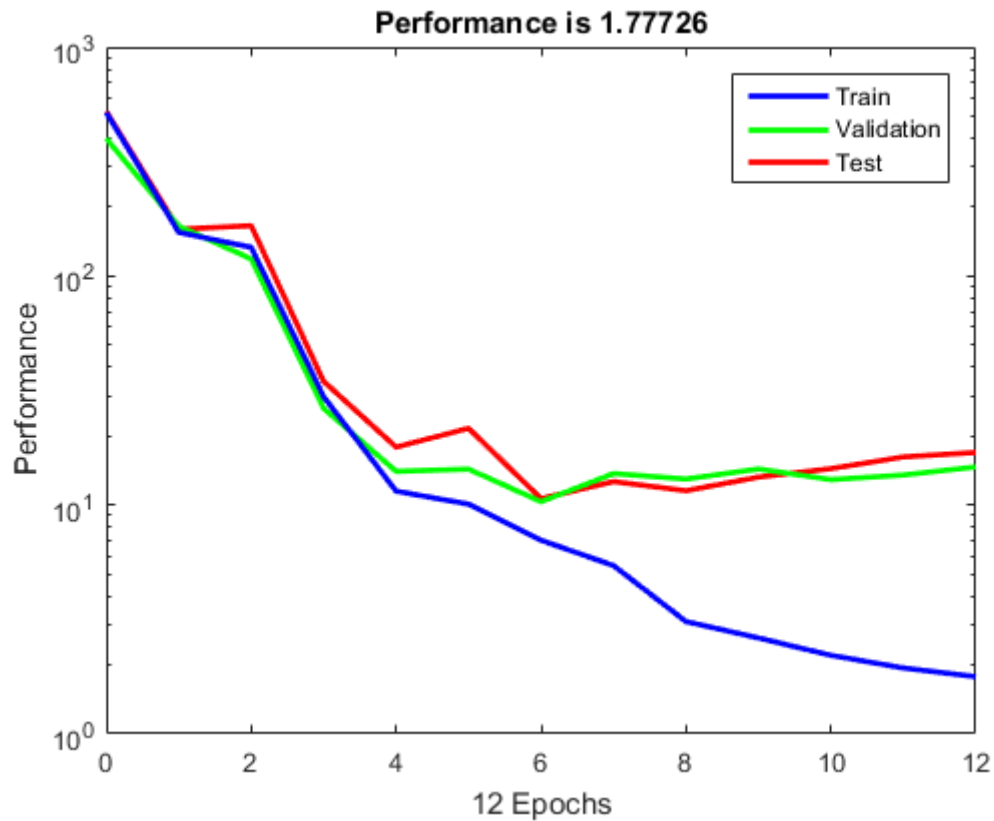


Fig 2.1

Once trained the system shows very good performance.[2]

- Determine your target error rate,  $e$
- Success rate is  $1 - e$
- Typical training set approx.  $n/e$ , where  $n$  is the number of weights in the net
- Trained until 95% correct training set classification should produce 90% correct classification on testing set (typical)
- Example:
  - $e = 0.1$ ,  $n = 80$  weights
  - training set size 800

### 3.3 Proposed system

The proposed system is basically a research based system which classifies the given database using rules that are extracted from neural network. At present, data mining is a new and important area of research, and neural network itself is very suitable for solving the problems of data mining because its characteristics of good robustness, self-organizing adaptive, parallel processing, distributed storage and high degree of fault tolerance. The combination of data mining method and neural network model can greatly improve the efficiency of data mining methods, and it has been widely used.

#### 3.3.1 Neural Networks (Backpropagation)

In machine learning and cognitive science, artificial neural networks (ANNs) are a family of statistical learning models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.[5]

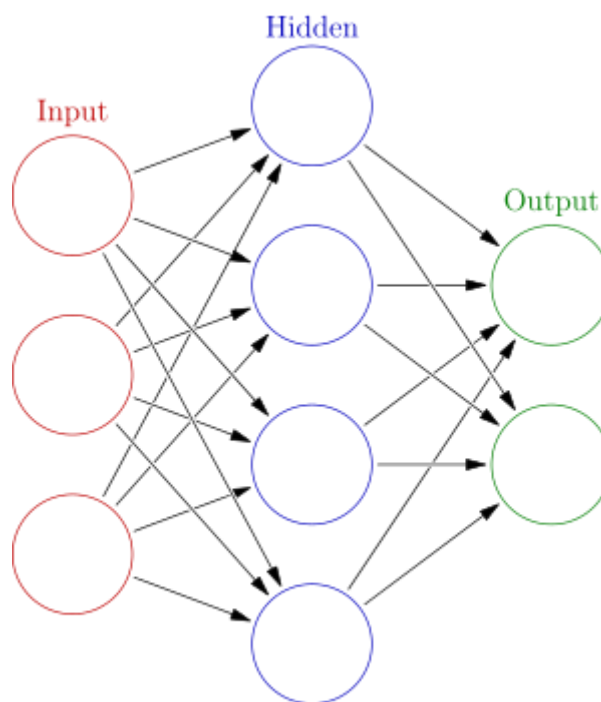


Fig 3.2 Neural Network

Backpropagation, an abbreviation for "backward propagation of errors", is a common method of training artificial neural networks used in conjunction with an optimization method such as gradient descent. The method calculates the gradient of a loss function with respect to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function.[6]

Backpropagation requires a known, desired output for each input value in order to calculate the loss function gradient. It is therefore usually considered to be a supervised learning method, although it is also used in some unsupervised networks such as auto encoders. It is a generalization of the delta rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer. Backpropagation requires that the activation function used by the artificial neurons (or "nodes") be differentiable.[6]

The backpropagation learning algorithm can be divided into two phases: propagation and weight update.

#### Phase 1: Propagation

Each propagation involves the following steps:

1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.
2. Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas (the difference between the input and output values) of all output and hidden neurons.

#### Phase 2: Weight update

For each weight-synapse follow the following steps:

1. Multiply its output delta and input activation to get the gradient of the weight.
2. Subtract a ratio (percentage) of the gradient from the weight.

This ratio (percentage) influences the speed and quality of learning; it is called the *learning rate*. The greater the ratio, the faster the neuron trains; the lower the ratio, the more accurate the training is. The sign of the gradient of a weight indicates where the error is increasing.

This is why the weight must be updated in the opposite direction.[6] Repeat phase 1 and 2 until the performance of the network is satisfactory.

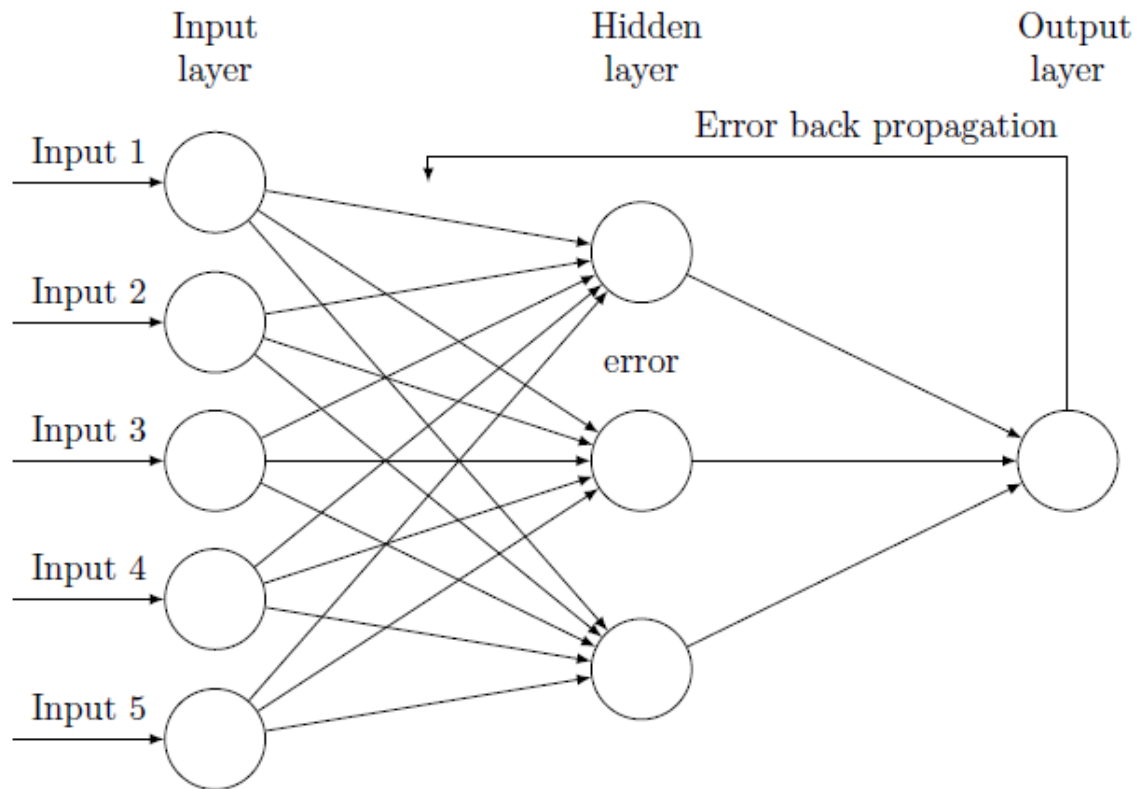


Fig 3.3 Backpropagation

## CHAPTER 4

### Design

#### 4.1 Design Consideration

- **Reliability:** The reliability of this program depends how the data has been encoded to train the network. The network won't be reliable if poor and ambiguous encoding technique is used.
- **Re-usability:** The current neural network has two classes for classification. This can be changed to support more number of classes for classification.
- **Robustness:** Since the algorithm is trained over a large number of data sets, the network can perform under stress and can give accurate input in case of some invalid data.
- **Usability:** The user interface of this program would be simple and easy to understand so that it can be used by target audience.

#### 4.2 Design Details

The Data Flow Diagram and the Sequence Diagram shown below describes how and in what sequence the action and data flow occurs in the program. Both the diagram contains three entities: Neural Network, Database and Prediction function.

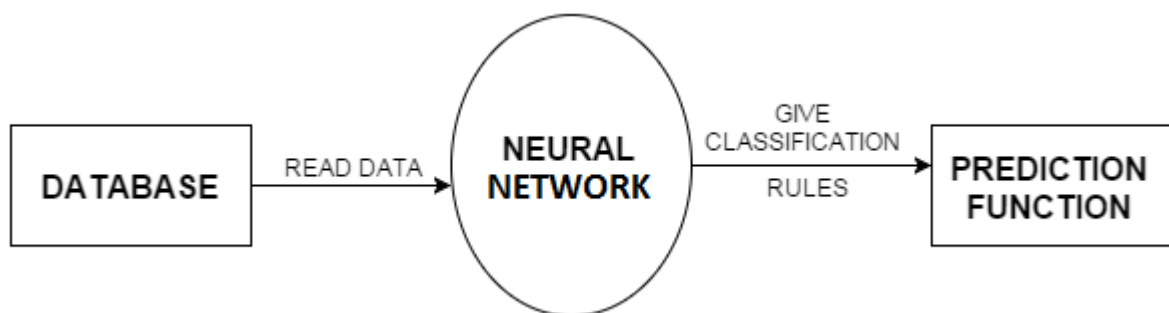


Fig 4.1 Level 0 DFD

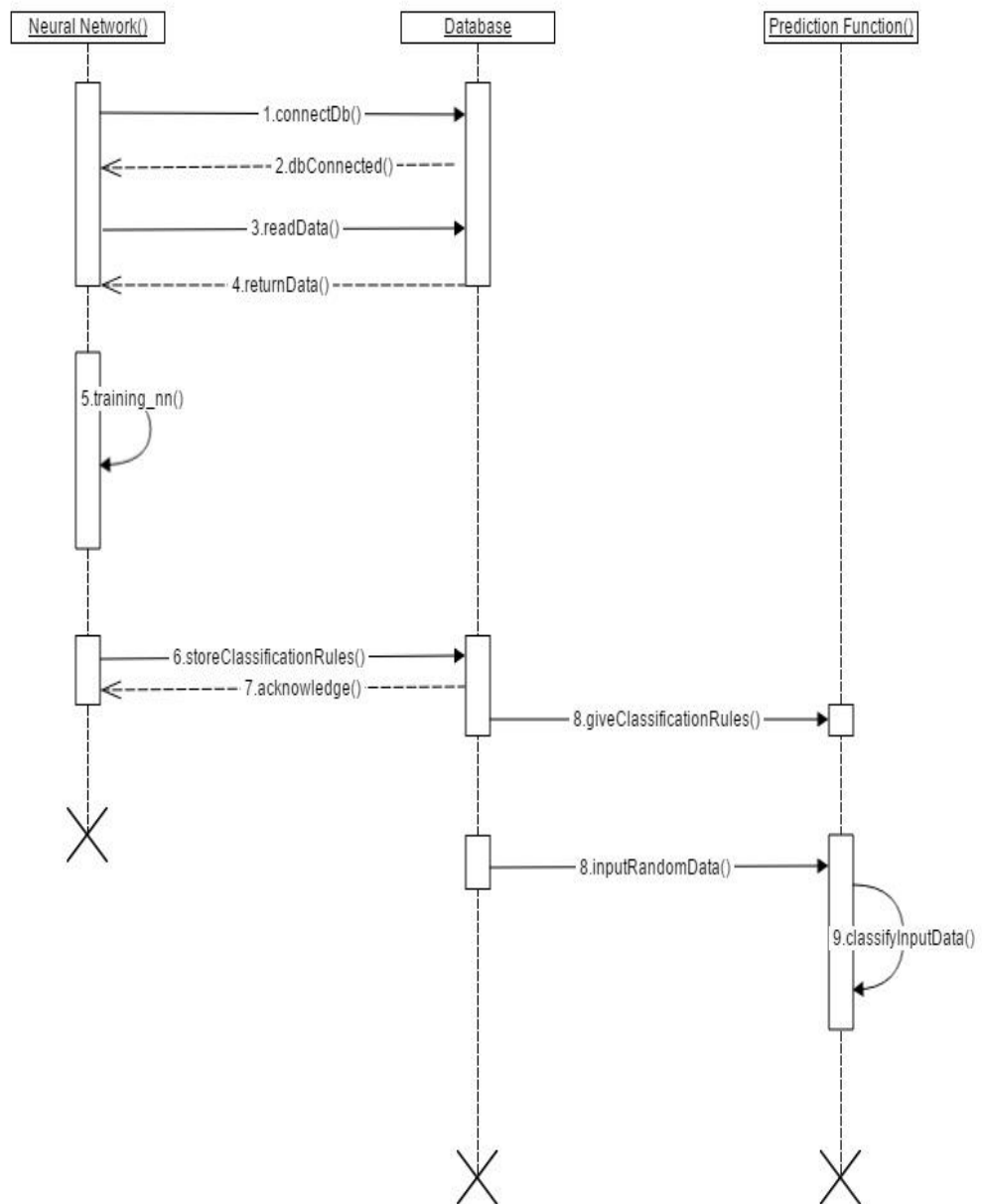


Fig 4.2 Sequence Diagram

## 4.3 GUI DETAILS

The below figure shows the proposed GUI of the program that would be designed. Based on the input and the output and how the program works the GUI contents are as follows:

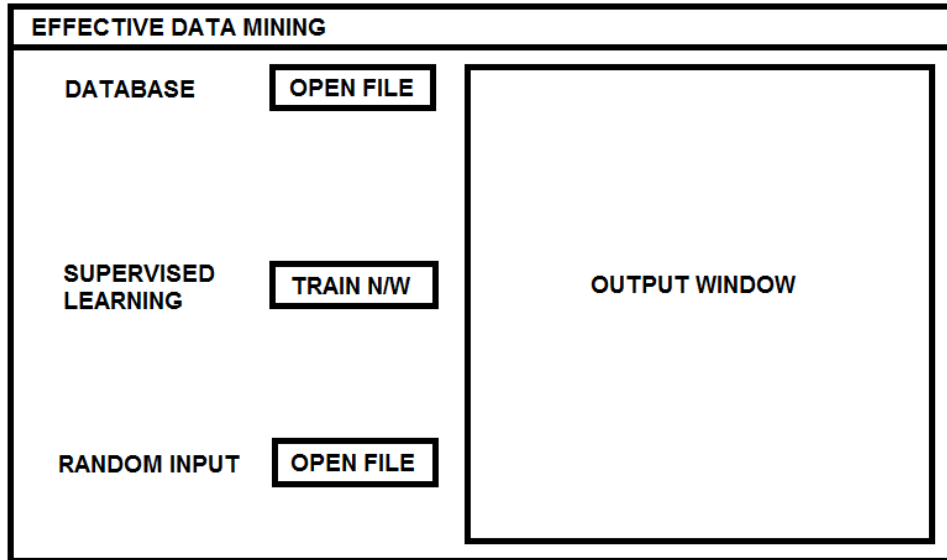


Fig 4.3 Proposed GUI

### 4.3.1 Buttons

- OPEN FILE (SUPERVISED LEARNING): This button will be used to access the database which will be used to train the neural network.
- TRAIN NETWORK: This button will initiate the network training process.
- OPEN FILE (RANDOM INPUT): This button will be used to give any random input via a database file to the network.

### 4.3.2 Output Window

The output window would be used to show the result after the processing of random input data using the classification rules. The window will be also used to show messages regarding completion or failure of operations.



## **CHAPTER 5**

### **Implementation Methodology**

Here we will study how the entire project idea was implemented into a run able code which produces a tangible output, i.e. a running version of software. Hence in this chapter we will learn implementation of the project. Implementation is vital part in software building and designing. Our project is implemented in Java. We have used Eclipse to build our Project. We are going to use MySQL database to store data. DBC will serve as an interface between the program and database.

#### **5.1 Module Description**

##### **5.1.1 Designing a Neural Network**

A neural network has been designed using Backpropagation algorithm which is a supervised learning algorithm. The Backpropagation neural network is a multilayered, feedforward neural network and is by far the most extensively used. It is also considered one of the simplest and most general methods used for supervised training of multilayered neural networks. Backpropagation works by approximating the non-linear relationship between the input and the output by adjusting the weight values internally.[1]

##### **5.1.2 Creating/Gathering training Data sets**

We create/gather the suitable training data sets for which the neural network is to be trained. The training dataset should be large enough to train the network for almost all the probable combinations of data.

### 5.1.3 Thermometer Encoding

Before we feed the data to the network for its training, the data needs to be encoded in a format which the neural network can easily work with. For this, we use thermometer encoding technique. To solve the problem, a neural network was first constructed. Each attribute value was coded as a binary string for use as input. to the network The e strings used to represent the attribute values are in The thermometer coding scheme was used for binary representations of the continuous attributes Each bit of a string was either 0 or 1 depending on which subinterval the original value was located For example, a salary value of 140k would be coded as (1, 1, 1, 1,1,1) and a value of 100k as (0, 1, 1, 1, 1, 1}. For the discrete attribute, elevel, for example, an elevel of 0 would be coded as (0, 0, 0, 0} , 1 as {0, 0, 0,1}, etc.[1]

Table : Coding of the attributes for neural network input

Attribute	No. of inputs	Subintervals
Salary	6	[20k,25k],[25k,50k],[50k,75k],[75k,100k],[100k,125k],[125k,150k]
Commission	3	[0k,25k],[25k,50k],[50k,75k]
Age	6	[20,30],[30,40],[40,50],[50,60],[60,70],[70,80]
Elevel	4	[0],[1],[2],[3],[4]

### 5.1.4 Training the Neural Network

In supervised training, both the inputs and the outputs are provided. The network then processes the inputs and compares its resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights which control the network. This process occurs over and over as the weights are continually tweaked. During the training of a network the same set of data is processed many times as the connection weights are ever refined. A limiting value is set so that the training stops on reaching the limit. A learning rate is initialized which is set to either low(if the dataset is variable and large) or high(if the dataset is small).[6]

### 5.1.5 Generate Classification Rules

Based on the corrected weights to the hidden layers, classification rules are generated which is stored in the another database for future reference.

### 5.1.6 Testing the Neural Network against Random data

Once the training part is completed, the network is given random sets of data as input. The network will classify the given input into the appropriate group/class based on how effectively the network is trained. The network will be 80-90% accurate in classifying the random sets of data.[1]

## 5.2 Flowchart

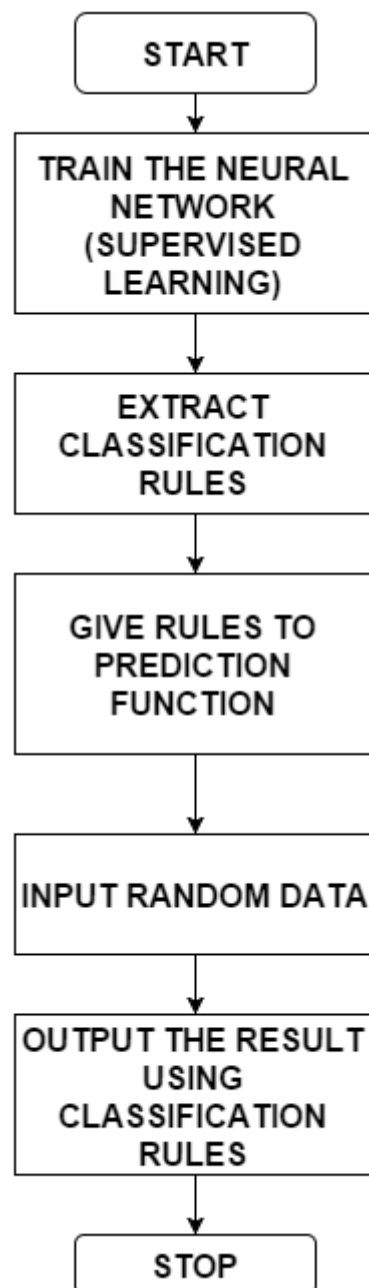


Fig 5.1

## 5.3 Code

```
import java.util.Random;
import java.text.DecimalFormat;

public class Example_4x6x16
{
    private static final int INPUT_NEURONS = 4;
    private static final int HIDDEN_NEURONS = 6;
    private static final int OUTPUT_NEURONS = 14;

    private static final double LEARN_RATE = 0.2; // Rho.
    private static final double NOISE_FACTOR = 0.45;
    private static final int TRAINING_REPS = 10000;

    // Input to Hidden Weights (with Biases).
    private static double wih[][] = new double[INPUT_NEURONS +
1][HIDDEN_NEURONS];

    // Hidden to Output Weights (with Biases).
    private static double who[][] = new double[HIDDEN_NEURONS +
1][OUTPUT_NEURONS];

    // Activations.
    private static double inputs[] = new double[INPUT_NEURONS];
    private static double hidden[] = new double[HIDDEN_NEURONS];
    private static double target[] = new double[OUTPUT_NEURONS];
    private static double actual[] = new double[OUTPUT_NEURONS];

    // Unit errors.
    private static double erro[] = new double[OUTPUT_NEURONS];
    private static double errh[] = new double[HIDDEN_NEURONS];
```

```
private static final int MAX_SAMPLES = 14;
```

```
private static int trainInputs[][] = new int[][] { {1, 1, 1, 0},  
                                                    {1, 1, 0, 0},  
                                                    {0, 1, 1, 0},  
                                                    {1, 0, 1, 0},  
                                                    {1, 0, 0, 0},  
                                                    {0, 1, 0, 0},  
                                                    {0, 0, 1, 0},  
                                                    {1, 1, 1, 1},  
                                                    {1, 1, 0, 1},  
                                                    {0, 1, 1, 1},  
                                                    {1, 0, 1, 1},  
                                                    {1, 0, 0, 1},  
                                                    {0, 1, 0, 1},  
                                                    {0, 0, 1, 1}};
```

```
private static int trainOutput[][] = new int[][]  
    { {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},  
      {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},  
      {0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},  
      {0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},  
      {0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},  
      {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},  
      {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0},  
      {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},  
      {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},  
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0},  
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0},  
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0},  
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},  
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1}};
```

```

private static void NeuralNetwork()
{
    int sample = 0;
    assignRandomWeights();
    // Train the network.
    for(int epoch = 0; epoch < TRAINING_REPS; epoch++)
    {
        sample += 1;
        if(sample == MAX_SAMPLES){
            sample = 0;
        }
        for(int i = 0; i < INPUT_NEURONS; i++)
        {
            inputs[i] = trainInputs[sample][i];
        } // i

        for(int i = 0; i < OUTPUT_NEURONS; i++)
        {
            target[i] = trainOutput[sample][i];
        } // i

        feedForward();
        backPropagate();

    } // epoch
    getTrainingStats();
    System.out.println("\nTest network against original input:");
    testNetworkTraining();
    System.out.println("\nTest network against noisy input:");
    testNetworkWithNoise1();
    return;
}

```

```

private static void getTrainingStats()
{
    double sum = 0.0;
    for(int i = 0; i < MAX_SAMPLES; i++)
    {
        for(int j = 0; j < INPUT_NEURONS; j++)
        {
            inputs[j] = trainInputs[i][j];
        } // j

        for(int j = 0; j < OUTPUT_NEURONS; j++)
        {
            target[j] = trainOutput[i][j];
        } // j

        feedForward();

        if(maximum(actual) == maximum(target)){
            sum += 1;
        }else{
            System.out.println(inputs[0] + "\t" + inputs[1] + "\t" + inputs[2] + "\t" + inputs[3]);
            System.out.println(maximum(actual) + "\t" + maximum(target));
        }
    } // i

    System.out.println("Network is " + ((double)sum / (double)MAX_SAMPLES * 100.0) +
"% correct.");
    return;
}

private static void testNetworkTraining()
{
    // This function simply tests the training vectors against network

```

```

for(int i = 0; i < MAX_SAMPLES; i++)
{
    for(int j = 0; j < INPUT_NEURONS; j++)
    {
        inputs[j] = trainInputs[i][j];
    } // j
    feedForward();
    for(int j = 0; j < INPUT_NEURONS; j++)
    {
        System.out.print(inputs[j] + "\t");
    } // j
    System.out.print("Output: " + maximum(actual) + "\n");
} // i
return;
}

```

```

private static void testNetworkWithNoise1()
{
    // This function adds a random fractional value to all the training
    // inputs greater than zero.
    DecimalFormat dfm = new java.text.DecimalFormat("###0.0");
    for(int i = 0; i < MAX_SAMPLES; i++)
    {
        for(int j = 0; j < INPUT_NEURONS; j++)
        {
            inputs[j] = trainInputs[i][j] + (new Random().nextDouble() * NOISE_FACTOR);
        } // j
        feedForward();
        for(int j = 0; j < INPUT_NEURONS; j++)
        {
            System.out.print(dfm.format(((inputs[j] * 1000.0) / 1000.0)) + "\t");
        } // j
    }
}

```



```

System.out.print("Output: " + maximum(actual) + "\n");
    } // i
return;
}

private static int maximum(final double[] vector)
{
    // This function returns the index of the maximum of vector().
    int sel = 0;
    double max = vector[sel];

    for(int index = 0; index < OUTPUT_NEURONS; index++)
    {
        if(vector[index] > max){
            max = vector[index];
            sel = index;
        }
    }
    return sel;
}

private static void feedForward()
{
    double sum = 0.0;

    // Calculate input to hidden layer.
    for(int hid = 0; hid < HIDDEN_NEURONS; hid++)
    {
        sum = 0.0;
        for(int inp = 0; inp < INPUT_NEURONS; inp++)
        {
            sum += inputs[inp] * wih[inp][hid];
        } // inp
    }
}

```

```

sum += wih[INPUT_NEURONS][hid]; // Add in bias.
    hidden[hid] = sigmoid(sum);
} // hid

// Calculate the hidden to output layer.
for(int out = 0; out < OUTPUT_NEURONS; out++)
{
    sum = 0.0;
    for(int hid = 0; hid < HIDDEN_NEURONS; hid++)
    {
        sum += hidden[hid] * who[hid][out];
    } // hid

    sum += who[HIDDEN_NEURONS][out]; // Add in bias.
    actual[out] = sigmoid(sum);
} // out
return;
}

private static void backPropagate()
{
    // Calculate the output layer error (step 3 for output cell).
    for(int out = 0; out < OUTPUT_NEURONS; out++)
    {
        erro[out] = (target[out] - actual[out]) * sigmoidDerivative(actual[out]);
    }

    // Calculate the hidden layer error (step 3 for hidden cell).
    for(int hid = 0; hid < HIDDEN_NEURONS; hid++)
    {
        errh[hid] = 0.0;

```

```

for(int out = 0; out < OUTPUT_NEURONS; out++)
{
    errh[hid] += erro[out] * who[hid][out];
}
errh[hid] *= sigmoidDerivative(hidden[hid]);
}

// Update the weights for the output layer (step 4).
for(int out = 0; out < OUTPUT_NEURONS; out++)
{
    for(int hid = 0; hid < HIDDEN_NEURONS; hid++)
    {
        who[hid][out] += (LEARN_RATE * erro[out] * hidden[hid]);
    } // hid
    who[HIDDEN_NEURONS][out] += (LEARN_RATE * erro[out]); // Update the bias.
} // out

// Update the weights for the hidden layer (step 4).
for(int hid = 0; hid < HIDDEN_NEURONS; hid++)
{
    for(int inp = 0; inp < INPUT_NEURONS; inp++)
    {
        wih[inp][hid] += (LEARN_RATE * errh[hid] * inputs[inp]);
    } // inp
    wih[INPUT_NEURONS][hid] += (LEARN_RATE * errh[hid]); // Update the bias.
} // hid
return;
}

private static void assignRandomWeights()
{
    for(int inp = 0; inp <= INPUT_NEURONS; inp++) // Do not subtract 1 here.

```

```

{
    for(int hid = 0; hid < HIDDEN_NEURONS; hid++)
    {
        // Assign a random weight value between -0.5 and 0.5
        wih[inp][hid] = new Random().nextDouble() - 0.5;
    } // hid
} // inp

for(int hid = 0; hid <= HIDDEN_NEURONS; hid++) // Do not subtract 1 here.
{
    for(int out = 0; out < OUTPUT_NEURONS; out++)
    {
        // Assign a random weight value between -0.5 and 0.5
        who[hid][out] = new Random().nextDouble() - 0.5;
    } // out
} // hid
return;
}

private static double sigmoid(final double val)
{
    return (1.0 / (1.0 + Math.exp(-val)));
}

private static double sigmoidDerivative(final double val)
{
    return (val * (1.0 - val));
}

public static void main(String[] args)
{
    NeuralNetwork();
    return;
}
}

```

## CHAPTER 6

### Implementation Plan For Semester VIII

#### 6.1 Gantt Chart

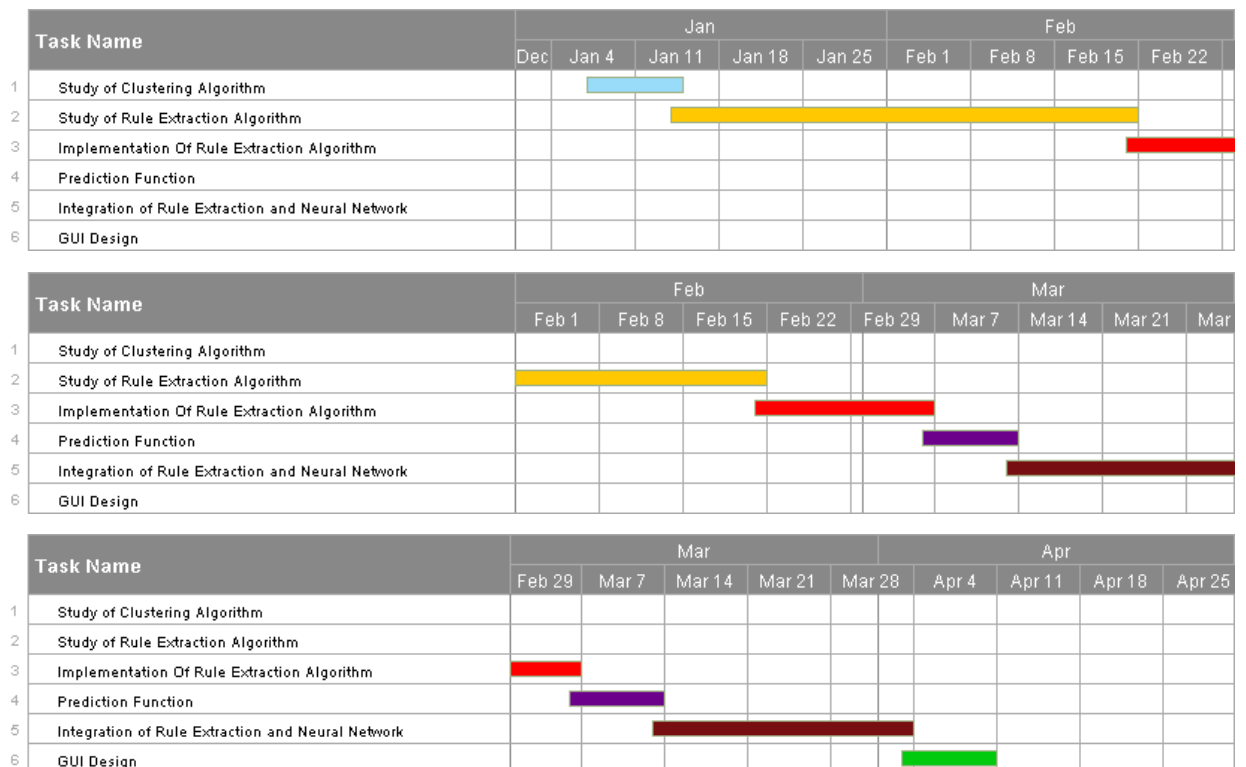
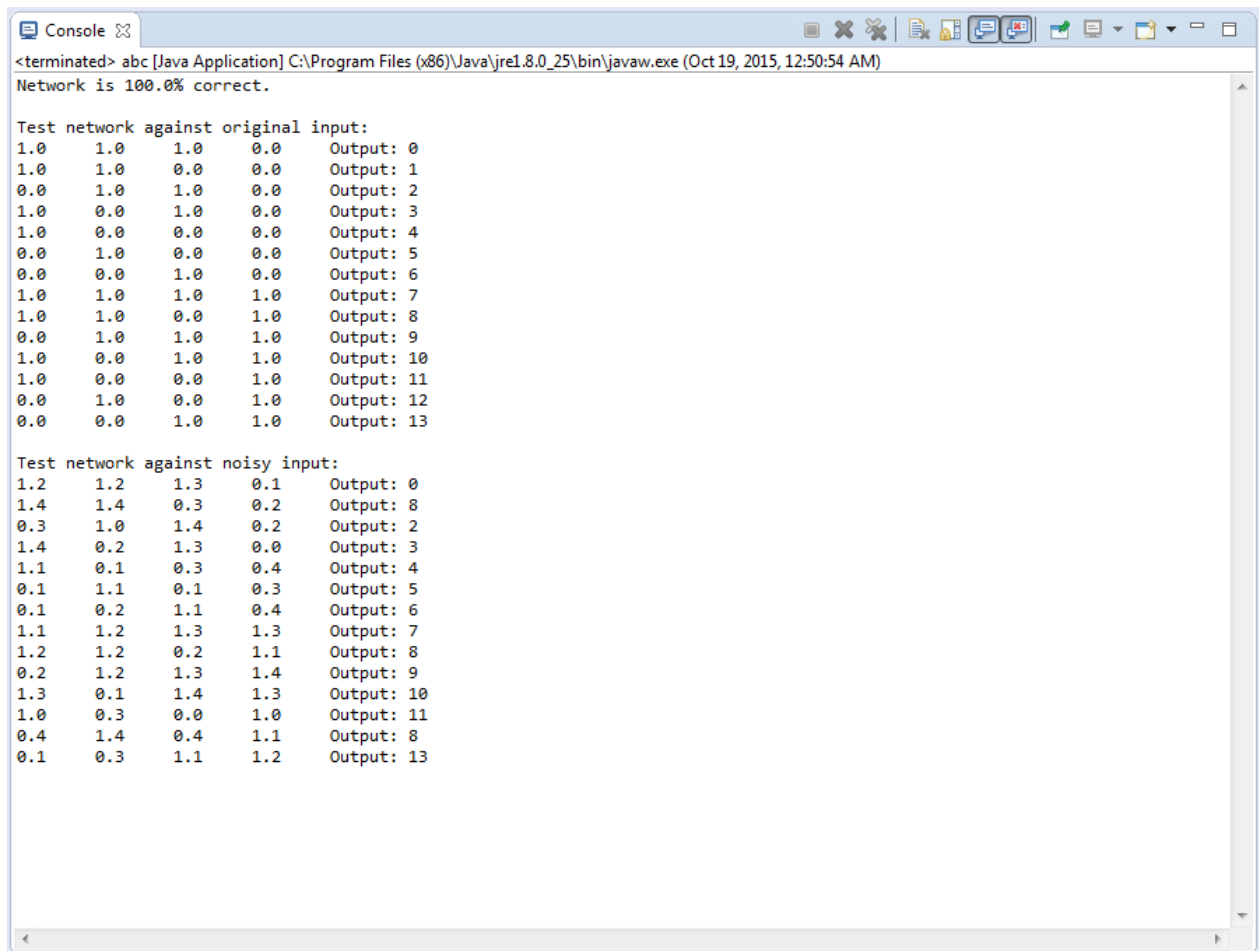


Fig 5.2

# CHAPTER 7

## Result

### 7.1 Screenshot



The screenshot shows a Java console window titled "Console" with the following output:

```
<terminated> abc [Java Application] C:\Program Files (x86)\Java\jre1.8.0_25\bin\javaw.exe (Oct 19, 2015, 12:50:54 AM)
Network is 100.0% correct.

Test network against original input:
1.0 1.0 1.0 0.0 Output: 0
1.0 1.0 0.0 0.0 Output: 1
0.0 1.0 1.0 0.0 Output: 2
1.0 0.0 1.0 0.0 Output: 3
1.0 0.0 0.0 0.0 Output: 4
0.0 1.0 0.0 0.0 Output: 5
0.0 0.0 1.0 0.0 Output: 6
1.0 1.0 1.0 1.0 Output: 7
1.0 1.0 0.0 1.0 Output: 8
0.0 1.0 1.0 1.0 Output: 9
1.0 0.0 1.0 1.0 Output: 10
1.0 0.0 0.0 1.0 Output: 11
0.0 1.0 0.0 1.0 Output: 12
0.0 0.0 1.0 1.0 Output: 13

Test network against noisy input:
1.2 1.2 1.3 0.1 Output: 0
1.4 1.4 0.3 0.2 Output: 8
0.3 1.0 1.4 0.2 Output: 2
1.4 0.2 1.3 0.0 Output: 3
1.1 0.1 0.3 0.4 Output: 4
0.1 1.1 0.1 0.3 Output: 5
0.1 0.2 1.1 0.4 Output: 6
1.1 1.2 1.3 1.3 Output: 7
1.2 1.2 0.2 1.1 Output: 8
0.2 1.2 1.3 1.4 Output: 9
1.3 0.1 1.4 1.3 Output: 10
1.0 0.3 0.0 1.0 Output: 11
0.4 1.4 0.4 1.1 Output: 8
0.1 0.3 1.1 1.2 Output: 13
```

Fig.7.1

## **CHAPTER 8**

### **Conclusion**

This semester we have implemented following modules.

- Design of Neural Network
- Database Connectivity Module

We will implement following modules for the Semester VIII

- Implementing Rule Extraction
- Integration of ANN module, database connectivity module, Rule Extraction module
- GUI module

## **CHAPTER 9**

### **References**

- [1] Effective Data Mining Using Neural Network by Huan Liu, Effective Data Mining Using Neural Network by Huan Liu, IEEE Transactions on Knowledge and Data Engineering (Impact Factor: 2.07). 01/1997; 8(6):957 - 961. DOI: 10.1109/69.553163
- [2] Bekir Karlik and A. Vehbi Olgac International Journal of Artificial Intelligence And Expert Systems (IJAE), Volume (1): Issue (4) 111 Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks
- [3] A comparative study of decision trees ID3 and C4.5, Badr HSSINA, Abdelkarim MERBOUHA
- [4] Top 10 Algorithms in Data Mining, Xindong Wu, Vinay Kumar
- [5] [https://en.wikipedia.org/wiki/Artificial\\_neural\\_network](https://en.wikipedia.org/wiki/Artificial_neural_network)
- [6] <https://en.wikipedia.org/wiki/Backpropagation>
- [7] [https://en.wikipedia.org/wiki/Data\\_mining](https://en.wikipedia.org/wiki/Data_mining)
- [9] [http://www.tutorialspoint.com/data\\_mining/dm\\_applications\\_trends.htm](http://www.tutorialspoint.com/data_mining/dm_applications_trends.htm)