CERTIFICATE

INDIAN INSTITUTE OF TECHNOLOGY DELHI
IRD Unit

No. IITD/IRD/MI01158/ 5472/2 Date: 6/6/17

Office Memorandum

Office memo for engagement of outside student under project/consultancy job as detailed below:

Name	Mr./Ms. Shreya K.R, Manipal Institute of Technology, Karnataka
Project No.	MI01158
Project Title	Microsoft Chair Professor Project of Dr. Jayadeva, Deptt. of Elect. Engg.
Budget Head	Contingency
Name of the Principal Investigator & Deptt/Centre	Prof./Dr. JAYADEVA , Dept. of Electrical Engineering
Period of Assignement	15-05-2017 to 02-07-2017
Honorarium to be paid	Rs. 8000.00 /-

This has approval of the Competent Authority.

Assistant Registrar (IRD)

Mr./Ms. Shreya K.R
(Through Prof./Dr. JAYADEVA
Dept. of Electrical Engineering)

AR (IRD A/Cs.)

CC: MI01158

ACKOWLEDGEMENT

I would like to express gratitude towards Prof. Jayadeva, of IIT Delhi, for accepting me as a student intern and for his patience in going through the details of my work. I was honored to work on an IEEE paper of which he was first author.

I would also like to thank Ms. Aashi Jindal and Mr. Prashant Gupta, present PhD students of the institute, for supervising my work and providing helpful insight multiple times.

I am grateful to Mr. Rakesh Kumar from the administrative section, for going to great lengths to allow me a smooth formal transition into the IIT system

Finally, I would like to thank my parents for their faith in me and my brother for his support.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This project deals with the implementation of the learning algorithm provided in the IEEE paper **Binary Classification by SVM Based Tree Type Neural Networks**. The Project was guided by the first author of the paper, **Professor Jayadeva** (IIT Delhi).

This algorithm is used to design an optimum pyramidal delayed perceptron architecture for an **artificial neural network** which is supposed to classify data into two classes, class 0 and class 1. It uses constructive addition of neurons at each stage to achieve this task. It makes uses of linear programming by the **simplex method** and a maximum margin classifier at each stage of this constructive addition.

The task involved reading and understanding the research paper, reading the relevant papers it used as reference and implementing the algorithm by breaking it into multiple functions. Finally the program was tested using various data sets. The algorithm was implemented on **Octave**, which is an open source software.

The duration of the internship was **7 weeks**. The internship was carried out at the **Indian Institute of Technology, Delhi** in the Electrical Department. The work was done in the Cyber lab of the department.

CHAPTER 2

BASIC THEORY

1. Artificial Neural Networks

Artificial neural networks are computational simulations of the networks found in the human brain. Each computational unit in a neural network is called a **neuron**. These neurons are modelled on the behavior of human brain cells, of the same name. Each neuron can take multiple inputs and provide a single output. The output of a neuron is dependent on the type of **activation function** used, which can be a linear, threshold or sigmoid function. The weighted sum of inputs to the network is given to the function as input. Multiple such neurons come together to form neural networks. The purpose of a neural network is to allow a system to learn. The network is provided with a large amount of data, called the training dataset. Then it learns the nature of the output desired with each input. This is called **supervised learning**. After learning with the provided dataset, it is tested with a test set to check for the accuracy of what it has learnt. If it has a good accuracy, it can then be used to predict outputs for data it hasn't been trained with. Thus, neural networks allow a system to decide the output on the basis of what it has learnt previously. This is called **soft computing**, as every decision isn't hard coded by the programmer, as seen in hard coding.

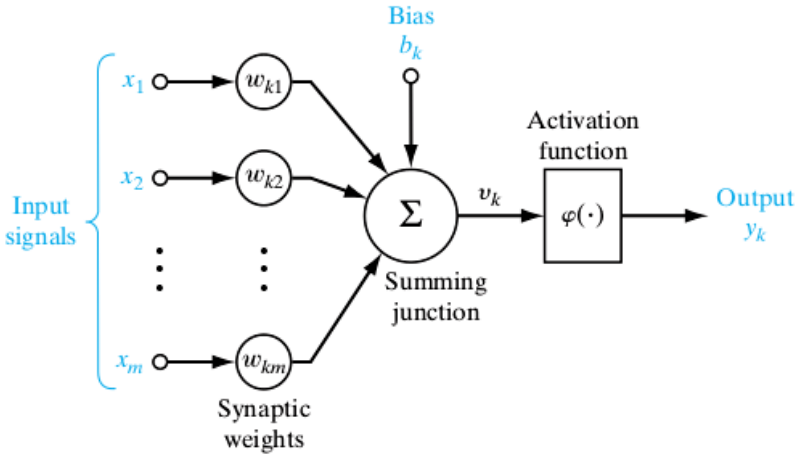

Fig 2.1 the structure of a computational neuron

From function approximation, to pattern recognition to data classification, neural networks have a wide array of applications in the artificial intelligence domain.

2. Multilayer Feedforward Neural Networks

A neural network can have one or more layers. Usually, for better quality of learning, a neural network has several layers one after the other, such that the output of one layer is fed as input to the consequent layer. Such neural networks are called **multilayer** neural networks. Neural Networks are of two types, feedforward and feedback.

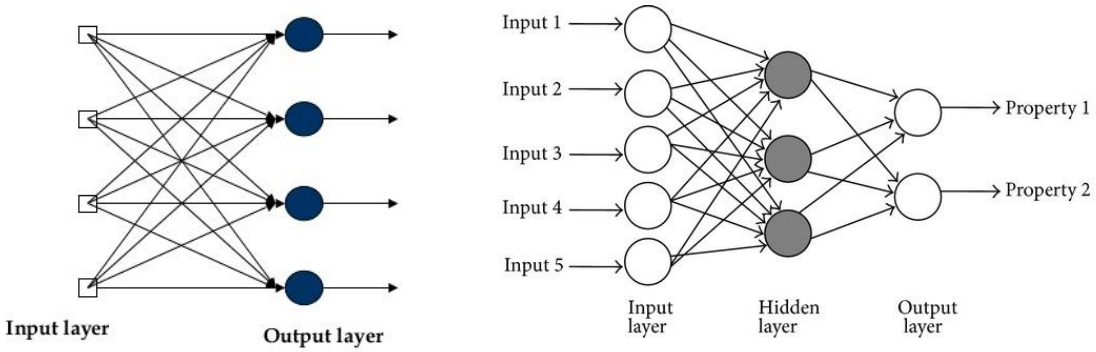

Fig 2.2 Single layer and multilayer neural networks

Feed-forward networks allow signals to travel one way only, from input to output. There is no feedback. The output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. **Feedback networks** can have signals travelling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organizations.

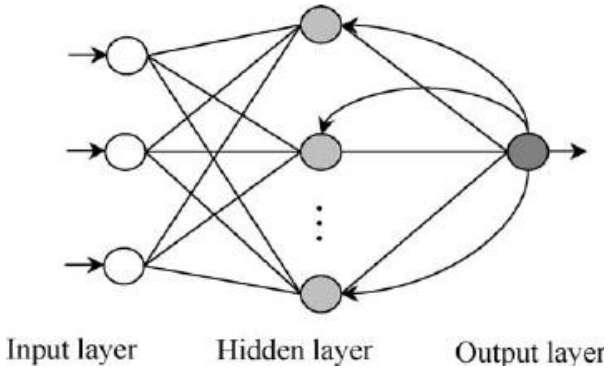

Fig 2.3 Feedback Neural Network

3. The Simplex Method

Linear programming is the optimization of an outcome based on some set of **constraints** using a linear mathematical model.

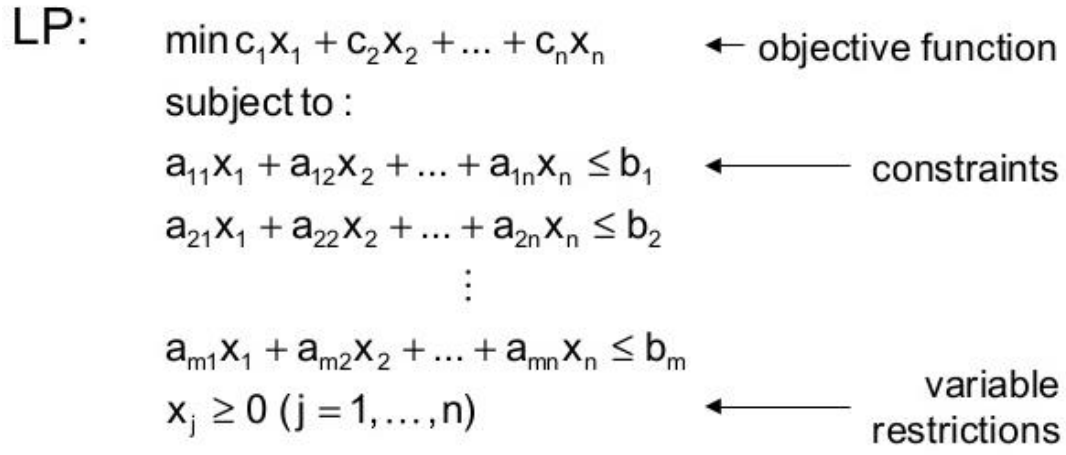

Fig 2.4 Linear Programming

The simplex method is a method used to solve a **linear programming problem**. To solve an LPP using the simplex method, the LPP must first be converted to its standard form by performing the following steps. In matrix form, consider matrices C,X,A and B.

- If the problem is min z, convert it to max z.
- If a constraint is $a_{i1}x_1+a_{i2}x_2+\ldots+a_{in}x_n <= b_i$, convert it into an equality constraint by adding a nonnegative slack variable s_i. The resulting constraint is $a_{i1}x_1 +a_{i2}x_2+\ldots+a_{in}x_n +s_i = b_i$, where $s_i >= 0$.
- If a constraint is $a_{i1}x_1 + ai_2x_2 + \ldots + a_{in}x_n >= b_i$, convert it into an equality constraint by subtracting a nonnegative surplus variable s_i. The resulting constraint is $a_{i1}x_1 + a_{i2}x_2 + \ldots+ a_{in}x_n\ s_i = b_i$, where $s_i >=0$.
- If some variable x_j is unrestricted in sign, replace it everywhere in the formulation by $x_j^+ - x_j^-$ where $x_j^+ >= 0$ and $x_j^- >=0$

After converting the LPP into the standard form, the **simplex algorithm** can be applied to find the optimum solution to the objective function, while keeping all the variables within the constraints.

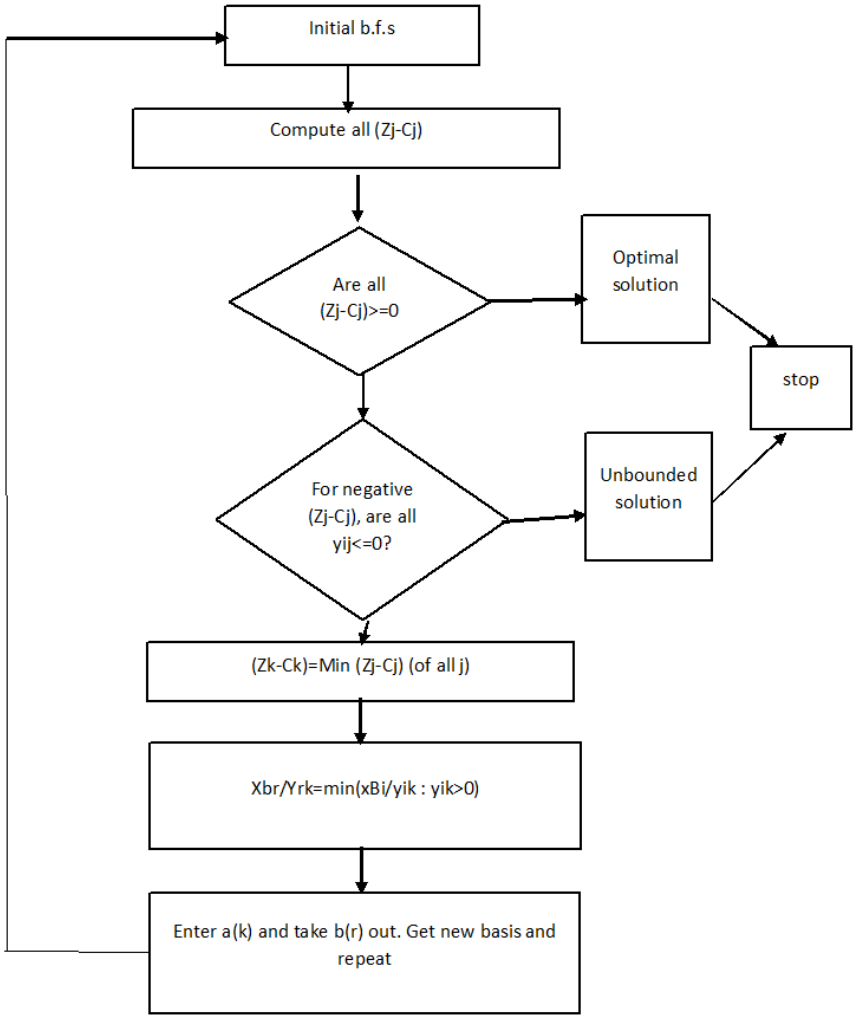

Fig 2.5 the simplex algorithm

- A b.f.s is a basic feasible solution which is a matrix of the size r x r where r is the rank of the matrix A. It takes any r columns from the matrix A initially. Usually an identity matrix is preferred.
- An initial tableau is constructed using this b.f.s. The variables corresponding to the columns of the b.f.s have an initial value of the matrix B, and all other variables are 0. The values of objective function Z are accordingly calculated.
- The initial tableau consists of a first column with matrix B, the next columns as matrix A. A row is appended to the bottom of the unit, where Z is calculated for each column, assuming the values of each column correspond to the variables selected in the b.f.s. For all columns other than the first, the corresponding coefficient of the objective function is subtracted from the final value (Zj-Cj).

4. Support Vector Machine

When we are provided with **linearly separable** data, often there are more than one lines that can divide the data into two. Deciding which of these lines to use is a challenge. It is, however, easy to understand that the line which can provide the maximum margins between itself and either data class, is the best separating **hyperplane**. This kind of maximum margin hyperplane can be obtained using the support vector method. In this method, we use quadratic programming to find the line that will give the maximum margin. The data points which lie on the margins are called **support vectors.**

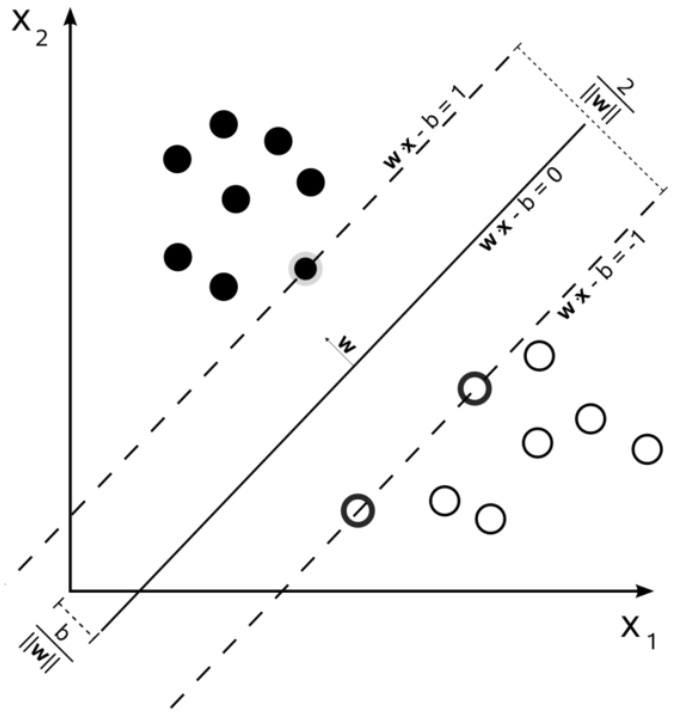

Fig 2.6 Using SVM to determine the best separating hyperplane

The image shows the equations of both the separating line and the margins. '**w**' is a vector perpendicular to the separating line, '**x**' is any point in the plane and b is the intercept. From the image, the margin is inversely proportional to the magnitude of vector w. Thus, to maximize the margin, we must minimize the magnitude of **w**. As magnitude involves the square root of the product of w, it involves quadratic optimization. The concept of SVM can be projected to higher dimensions using the same reasoning as the size of the vector w can be accordingly adjusted.

5. Optimizing The Architecture of a neural Network

There are two ways to develop a neural network for a particular application. One is to start off with far more neurons than necessary and remove unnecessary neurons at intermediate steps. This continues till an optimum structure is reached, beyond which removing further neurons will result in distorted output. This method is called **Network Pruning**. The other method involves starting off with one neuron, and adding more as and when required till a network that performs the desired task is reached. This method is called **Constructive addition**. Depending on the nature and complexity of the task at hand, one of these methods may be preferred.

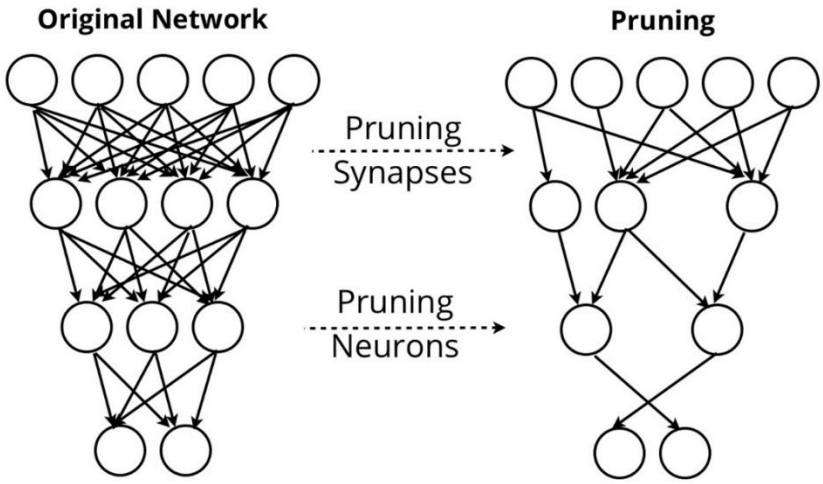

Fig 2.7 Neural Network pruning

CHAPTER 3

AGORITHM USED

1. Learning Algorithm

The constructive addition based algorithm to determine an optimum structure for a neural network to act as a binary classifier is as follows:-

1. Construct constraints for each data point in the training set, on the basis of whether it is supposed to be in class 0 or class 1. For Example, for a 2-d point (x1,x2) which belongs to class 0, the equation will be (i) and for a point (x1,x2) from class 1 will be (ii). W_1 and W_2 are the weights associated with inputs x1 and x2, and W_0 is the bias. They are split into W^+ and W^- as they are unconstrained variables, but in simplex, variables have to be positive.

   $W_0^+ - W_0^- + (W_1^+ - W_1^-)\, x1 + (W_2^+ - W_2^-)\, x2 <= -1$ (i)

   $W_0^+ - W_0^- + (W_1^+ - W_1^-)\, x1 + (W_2^+ - W_2^-)\, x2 >= 1$ (ii)

2. We have to minimize the averages of the classification errors of both classes. On adding surplus variables, the above constraints will become equations (iv) and (v).

   $W_0^+ - W_0^- + (W_1^+ - W_1^-)\, x1 + (W_2^+ - W_2^-)\, x2 + S_i^+ - S_i^- = -1$ (iii)

   $W_0^+ - W_0^- + (W_1^+ - W_1^-)\, x1 + (W_2^+ - W_2^-)\, x2 - S_j^+ + S_j^- = 1$ (iv)

   From these equations, we can see that S_i^- and S_j^- terms are what will lead to misclassification. Thus, we need to minimize the average sums of these. Thus, we can apply Simplex.

(LPP) $\quad \text{Minimize} \frac{1}{m}\sum_{i=1}^{m} s_i^{(-)} + \frac{1}{k}\sum_{j=1}^{k} s_j^{(-)}$

subject to the following constraints,

$$\sum_{d=0}^{n}\left(w_d^{(+)} - w_d^{(-)}\right)X_d^{(i)} = \Delta + \left(s_i^{(+)} - s_i^{(-)}\right) \text{ if } y^i = 1$$

$$\sum_{d=0}^{n}\left(w_d^{(+)} - w_d^{(-)}\right)X_d^{(j)} = -\Delta - \left(s_j^{(+)} - s_j^{(-)}\right) \text{ if } y^j = 0$$

Fig 3.1 Applying simplex to initial data

3. On the basis of the solution of the simplex, we obtain an initial weight matrix for our first neuron. Now, we will classify each of the data points into class 0 or class 1 on the basis of this weight matrix. After classification, we will check if the assigned class matches with the desired data, and on this basis, we will classify the data into four groups.

Table 3.1 Grouping data classified on simplex weights

Table I: CLASSIFICATION OF SAMPLES BASED ON OUTPUT			
Class	**Actual output**	**Desired output**	**No. of Samples**
C_1	0	0	N_1
C_2	1	1	N_2
C_3	0	1	N_3
C_4	1	0	N_4

4. If there are no samples in C3 or C4, proceed to step 8.

5. If there is a sample in C3, a type A child neuron needs to be added. Develop a training set for the type A neuron according to the table shown in figure 3.3.

6. If there is a sample in C3, a type B child neuron needs to be added. Develop a training set for the type B neuron according to the table shown in figure 3.3.

Table 3.2 Developing training sets for neurons A and B

Table II: REQUIRED OUTPUT OF NEURON A AND B		
Class	**Output of A**	**Output of B**
C_1	0	Don't care
C_2	Don't care	0
C_3	1	0
C_4	0	1

7. If neuron A or B or both have been added, augment the input dimensions of the parent neuron with the desired outputs of A/B/both.

8. Apply SVM i.e. Maximum Margin Classifier to the parent neuron.

9. If there is no value in C3 or C4, the final weight matrix for the vertex neuron is obtained.

CHAPTER 4

IMPLEMENTATION

1. Introduction

This algorithm was implemented using OCATAVE which is a free and open source software that functions along the lines of MATLAB, with the difference that it lacks several specialized toolboxes that MATLAB provides, for example, the machine learning toolbox. Thus, several functions have been coded from scratch which might otherwise have been built in, like support vector machine function.

The algorithm has been divided into 6 functions for implementation. Each function will be individually discussed in the forthcoming sections.

2. Function j1- The Simplex Function

This function accepts a matrix containing the data set to be classified along with corresponding class labels of each data point, as input and returns the optimum weight matrix for the neuron after applying the simplex algorithm to the provided data.

On first receiving the data, it initially constructs the coefficient matrix, the constant matrix and the objective function according to the task at hand. It then constructs the initial simplex tableau. After creating the initial tableau, it proceeds according to the algorithm till it obtains the final solution. It then returns the weight matrix. Some examples of this function in action are shown in figures 4.1, 4.2 and 4.3

```
%%INPUT
A=[1 1 1 0; 2 1 0 1];
C=[4 3 0 0];
K=[8;10];
```

```
Tableu
    8    1    1    1    0
   10    2    1    0    1
    0   -4   -3    0    0
```

Fig 4.1 The input and initial tableu for a 2D example

```
Diagonal Matrix

   1   0
   0   1

Tableu
    3.00000    0.00000    0.50000    1.00000   -0.50000
    5.00000    1.00000    0.50000    0.00000    0.50000
   20.00000    0.00000   -1.00000    0.00000    2.00000

Basis
   1   1
   0   2

Tableu
    6    0    1    2   -1
    2    1    0   -1    1
   26    0    0    2    1

Basis
   1   1
   1   2

optimal solution -

   2   6   0   0
>>
```

Command Window | Editor | Documentation

Fig 4.2 The Steps and optimal solution for the above example

```
Tableu
   14    2    1    1    1    0    0
   28    4    2    3    0    1    0
   30    2    5    5    0    0    1
    0   -1   -2    3    0    0    0
```

Octave

Command Window

```
    0   -1   -2    3    0    0    0

Basis
Diagonal Matrix

   1   0   0
   0   1   0
   0   0   1

Tableu
    8.00000    1.60000    0.00000    0.00000    1.00000    0.00000   -0.20000
   16.00000    3.20000    0.00000    1.00000    0.00000    1.00000   -0.40000
    6.00000    0.40000    1.00000    1.00000    0.00000    0.00000    0.20000
   12.00000   -0.20000    0.00000    5.00000    0.00000    0.00000    0.40000

Basis
   1   0   1
   0   1   2
   0   0   5

Tableu
    5.00000    1.00000    0.00000    0.00000    0.62500    0.00000   -0.12500
    0.00000    0.00000    0.00000    1.00000   -2.00000    1.00000    0.00000
    4.00000    0.00000    1.00000    1.00000   -0.25000    0.00000    0.25000
   13.00000    0.00000    0.00000    5.00000    0.12500    0.00000    0.37500

Basis
   2   0   1
   4   1   2
   2   0   5

optimal solution -

   5   4   0   0   0   0
>>
```

Command Window | Editor | Documentation

Fig4.3 A 3D example of the function j1

3. Function Classifyit –The Grouping Function

This function accepts both the initial data matrix (with all the data points and corresponding class labels) and the weight matrix generated by the function j1 as inputs. It returns a matrix called Cmatrix whose first column says which group it belongs to (1, 2, 3 or 4), the next n columns hold the coordinates of the n dimensional data point and the final column has the desired output. It also returns the number of samples in each group.

This function compares the output obtained by using the weight matrix generated by j1 to the desired output of each data point and then classifies it into C1, C2, C3 or C4 on the basis of table 3.1.

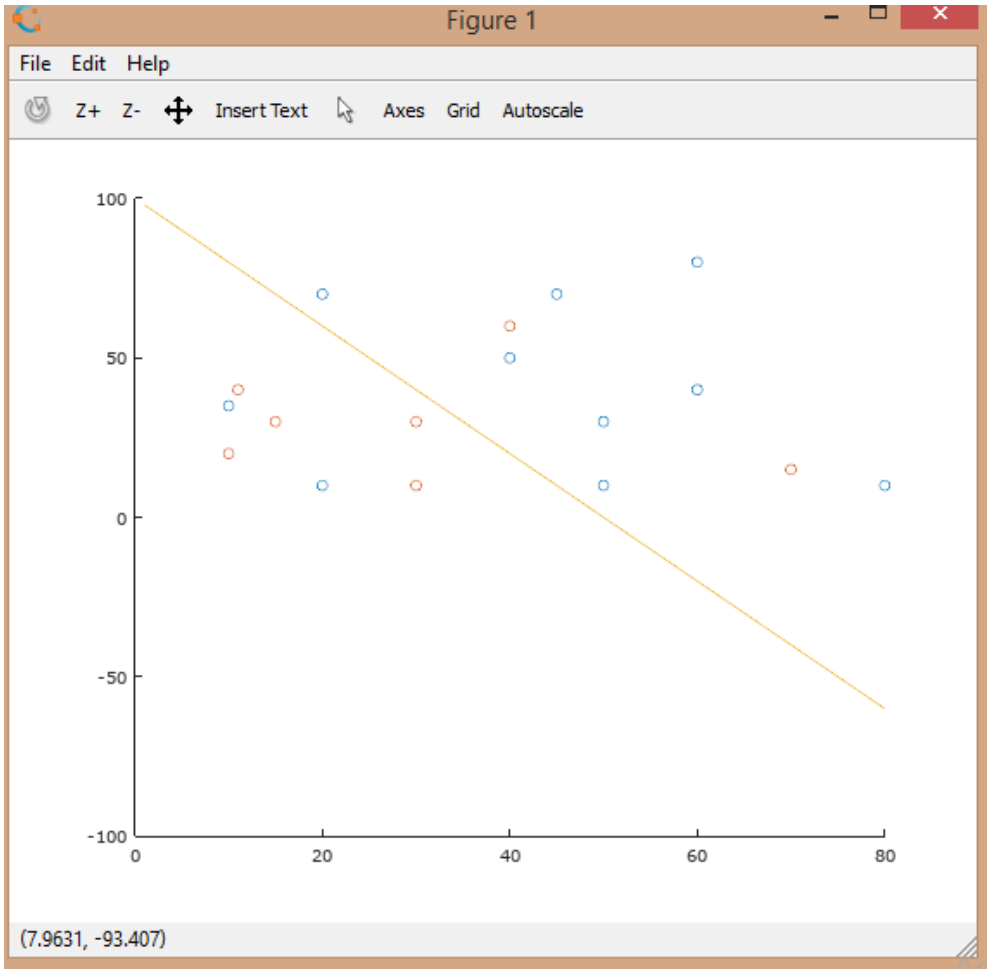

Fig 4.4 An Example Dataset to test the Classifyit function. The line represents the weight matrix obtained from j1.Samples belonging to class 0= blue and Samples belonging to class 1= Red

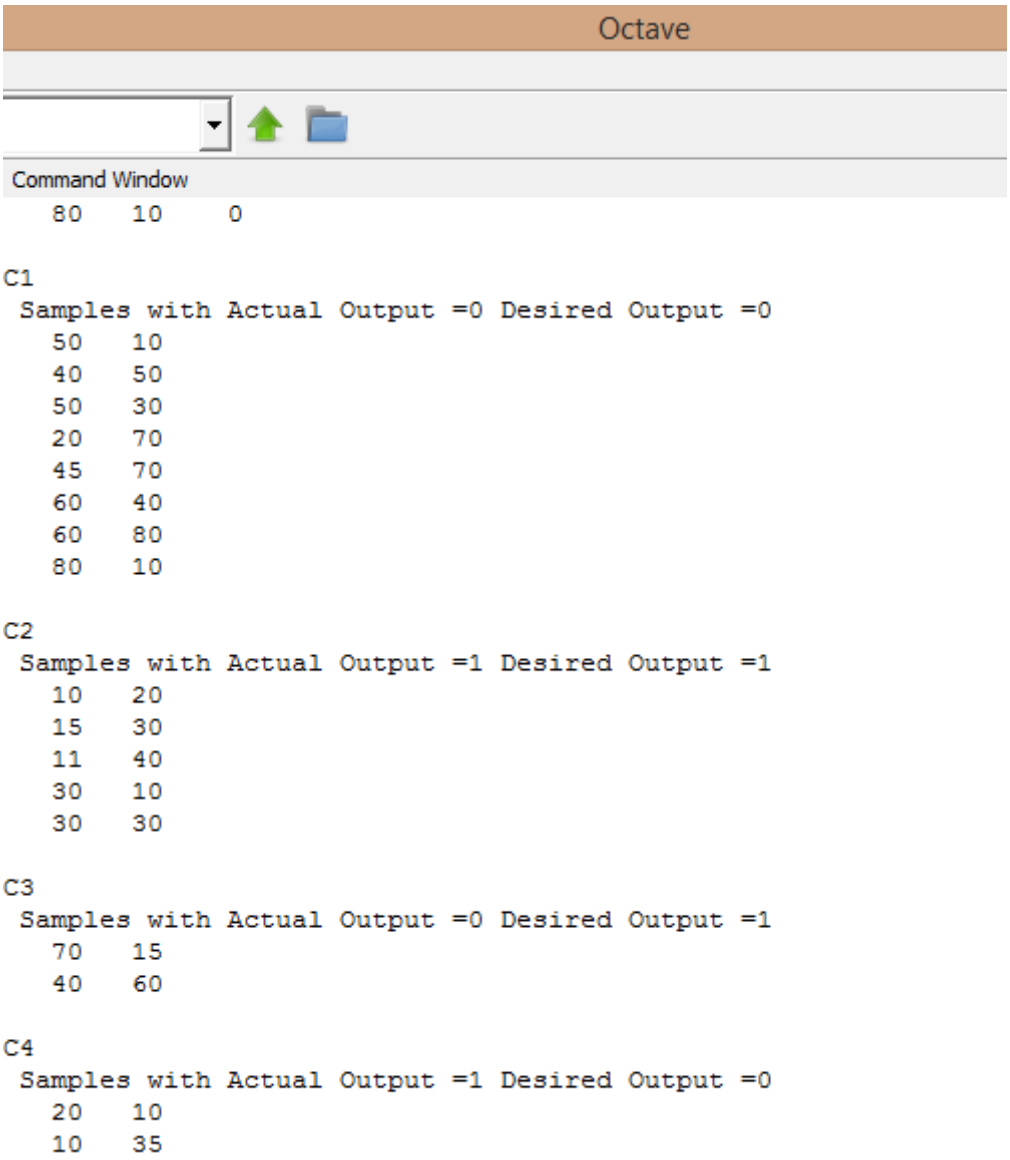

Fig 4.5 The Classification performed by Classifyit function

4. Function newin- Generates Augmented Input matrix for Vertex Neuron

This function takes Cmatrix, N3 and N4 (no. of samples in C3 and C4 respectively), all generated by Classifyit, as input and returns two matrices Cnew and Cdoll. Matrix Cnew has the augmented input matrix generated according to whether neuron A or B or both have been added, and the last column holds the corresponding desired output. Matrix Cdoll is different from Cnew only in the fact that its first column holds the group numbers (1,2, 3 and 4) for each sample.

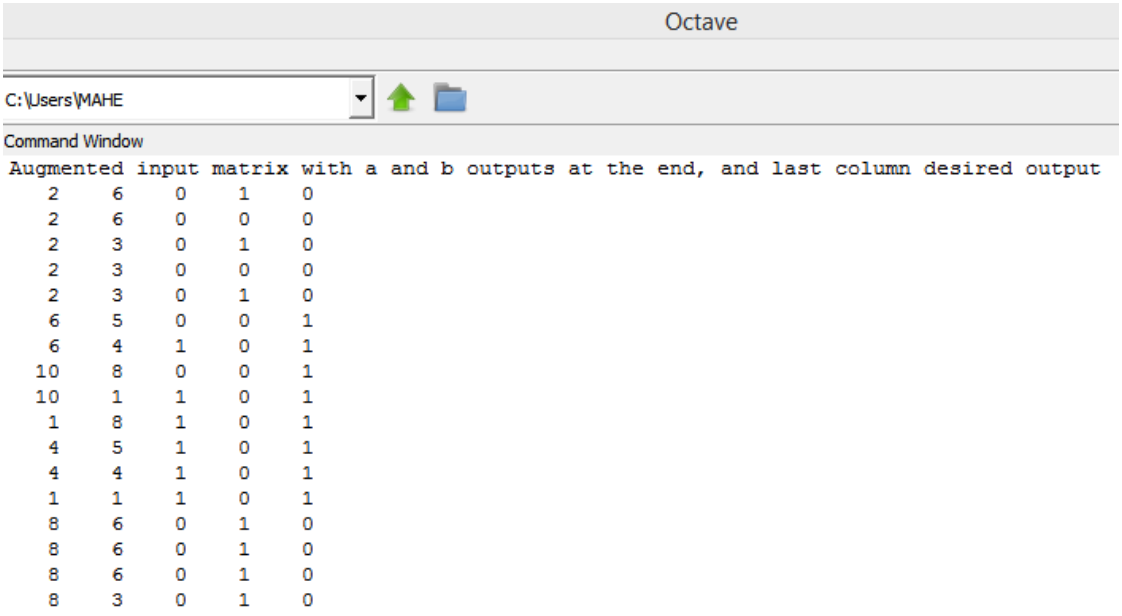

Fig 4.6 after augmentation, a 2D data set is extended to 4 dimensions as both A and B neurons have been added. The final column is the desired output.

5. Function svmap – The Support Vector Machine function

This function accepts a dataset with class labels and returns the final weight matrix after applying SVM. This is done by performing quadratic programming to minimize the magnitude of vector **w** to maximize the margin. Examples are shown in figures 4.7 and 4.8.

Table 4.1 a sample Dataset to perform SVM operation on

X1	X2	Y (class label)
3	1	0
4	1	0
5	3	0
6	2	0
7	1	0
7	2	0
8	2	0
8	3	0
1	3	1
1	4	1
2	5	1
3	6	1

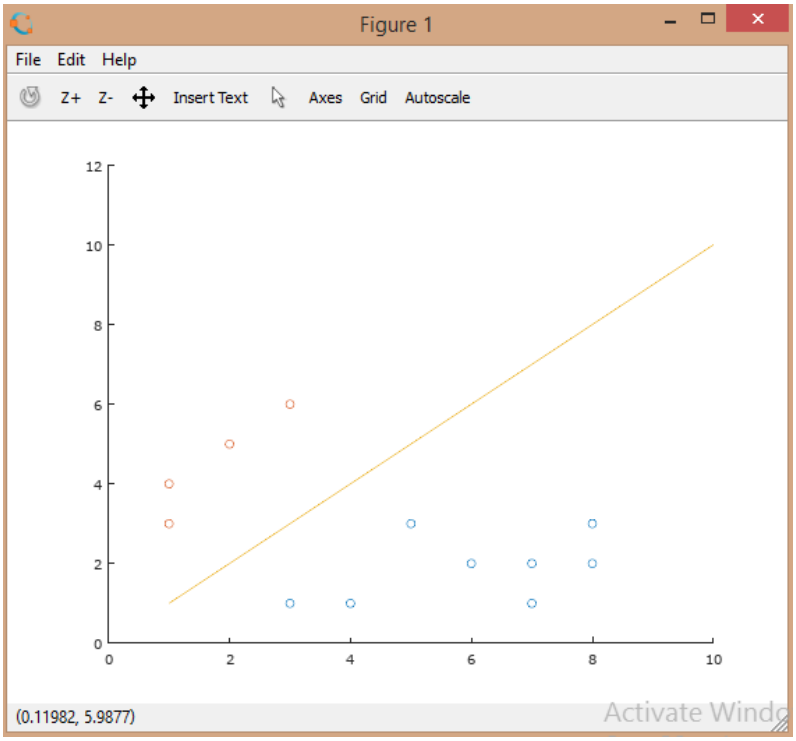

Fig 4.7 The resultant maximum margin hyperplane separating two classes. Optimum Weight Vector obtained, W= [0.00000; -0.50000; 0.50000]

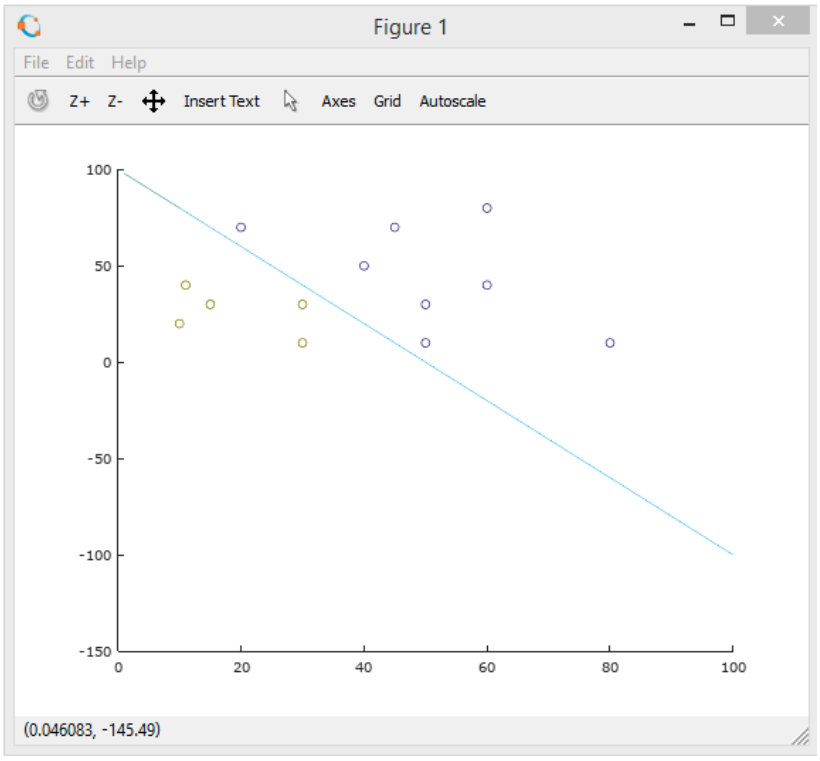

Fig 4.8 Another 2D example

6. Functions Ain and Binp – Designing training sets for child neurons

These function stake Cmatrix (generated by Classifyit) and N2 and N1 (no. of samples in C2 and C1) as input and return training sets ainp and binpi for neurons A and B respectively. Ain takes N2 specifically as input because the desired output of neuron A is a don't care value for members of group 2. Thus, N2 will be eliminated from the training set for A. Similarly N1 for B. This is in accordance with table 3.2.

7. Function Pyramid- A Recursive function to generate the final Pyramidal Network

Pyramid takes the dataset with class labels as input and returns the final weight matrix for that particular neuron. It encompasses all the above functions in sequence. It is a recursive function. It calls itself whenever there is a need to add child neurons A or B or both. This is the function that generates the final network.

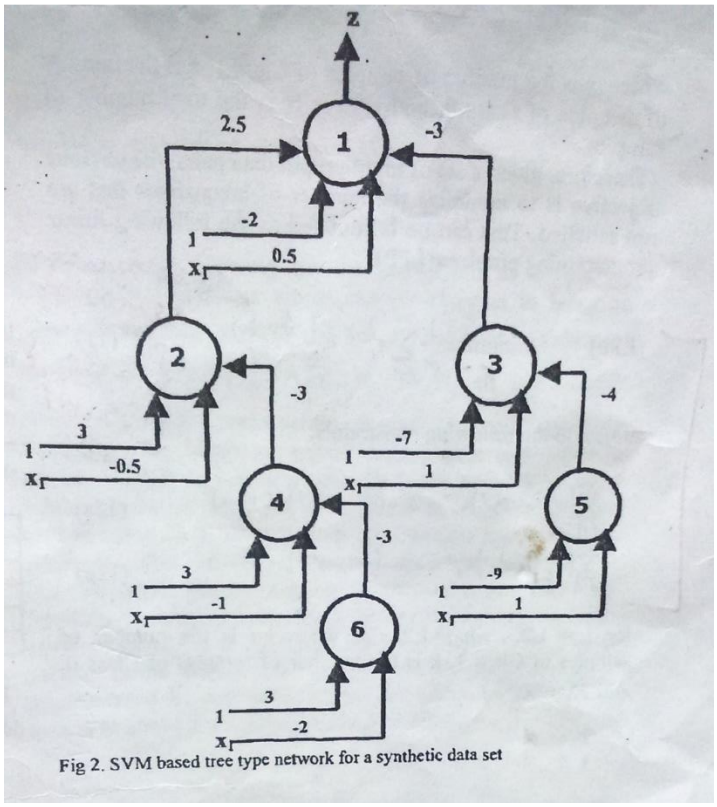

Fig 4.9 The network for the dataset [1 8 1; 4 5 1; 4 4 1; 1 1 1; 6 5 1; 6 4 1; 10 8 1; 10 1 1; 2 6 0; 2 3 0; 8 6 0; 8 3 0] where the third column denotes the class label generated by the pyramid function

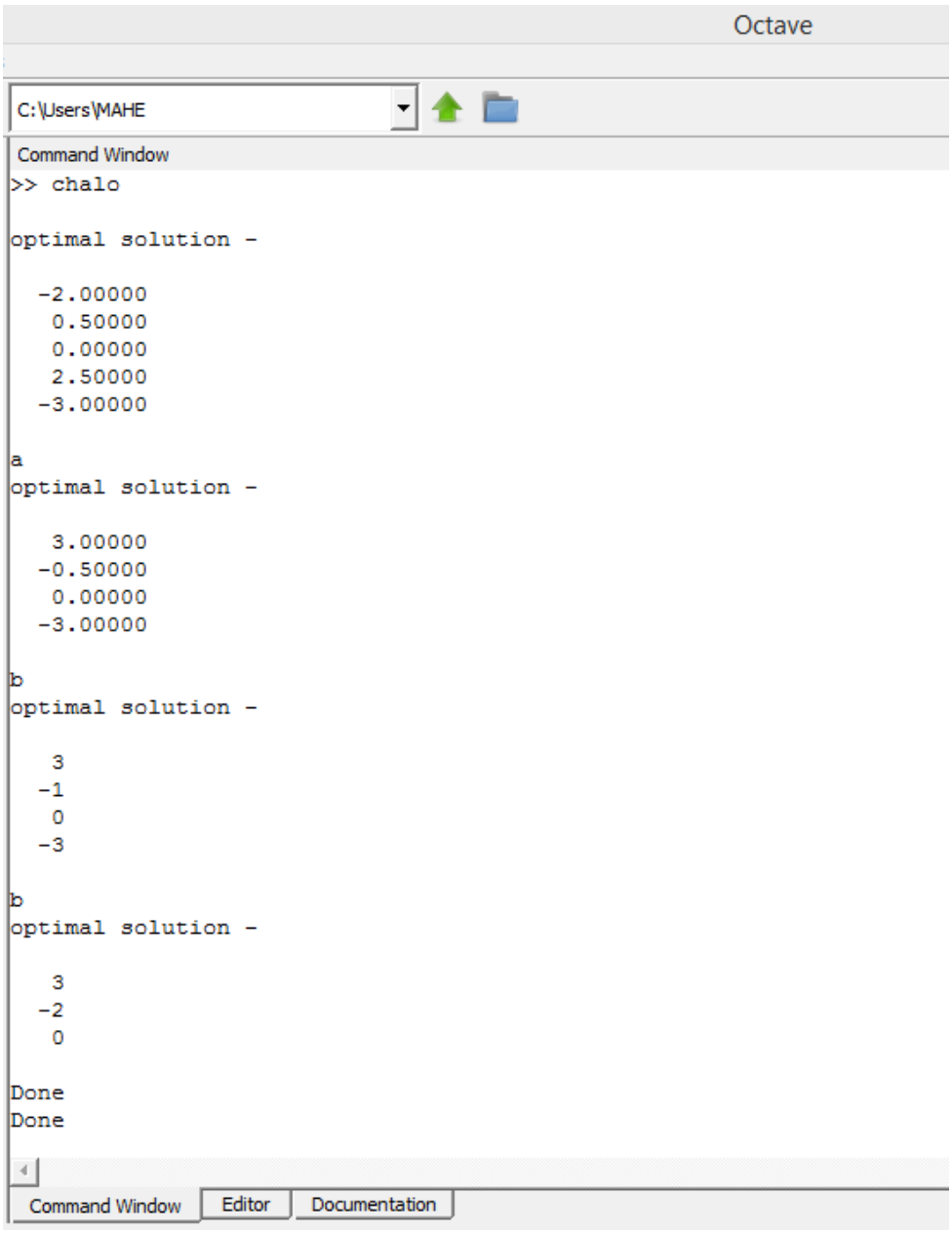

Fig 4.10 The obtained weights for the left half of the pyramid of above dataset

CHAPTER 5
SCOPE FOR FUTURE WORK

In this course of this project, the algorithm was tested primarily with 2-d and 3-d data. The program can be tested with **datasets of higher dimensions** to see how the speed of operation and the memory consumption are affected.

The project currently deals with binary classification. It can be extended to classify data into **multiple classes**.

It can be **compared** to existing algorithms for which determine the optimum architecture for binary classification neural networks. This can be done by providing the same dataset to different algorithms, and by repeating this with multiple datasets to see if there is a preference to certain kinds of data sets.

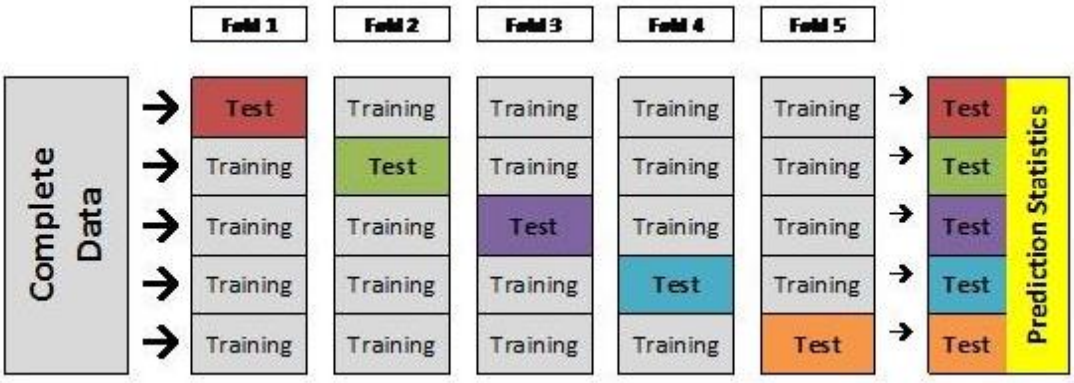

Fig 5.1: Schematic Representation of 5-fold Cross Validation

The accuracy of the program can be determined by means of **K-fold Cross validation**, wherein the dataset is divided into K units. K-1 units are used for training, and one is used for testing. This is repeated till each of the K units have been a test set once.

CHAPTER 6
DRAWBACKS

The project uses a **recursive function** to generate a Pyramidal Delayed Perceptron structure. Although a recursive function essentially means the code is more compact, and easier to debug, its **space requirements** are higher than that of an iterative function. This is because values of variables of each case have to be stored, till the base case is reached, after which memory is de-allocated. As the program was tested with datasets of small dimensions and datasets that produced trees of a small depth, this was insignificant, but it will gain importance with increase in the size of the tree, and with increase in dimensionality.

There is also an **imbalance of data** fed to the classifier. This means there are far higher samples of one class of data than the other. This may lead to overfitting in favor of the majority class.

Owing to the lack of specialized machine learning toolboxes in Octave, some functions were programmed from scratch, which contributes to a **longer program** than absolutely necessary.

Due to the choice of an **objective function with only surplus variables**, the simplex algorithm occasionally comes across a zero weight case with positive valued surplus variables, which in turn leads to a tree that alternates between two weights and never ends.

CHAPTER 7
CONCLUSION

The learning algorithm provided in the paper was successfully implemented using Octave. It was tested against a 2 dimensional example provided in the paper.

The program was made **scalable** so as to adapt to the dimensionality of any input data set given to it as input. So as to achieve this, each individual function used in the program is independently scalable.

Owing to the lack of the **specialized machine learning toolbox** in Octave, which is available in MATLAB, the SVM function was programmed from scratch using quadratic programming.

This algorithm provides an **efficient** way to generate an optimal neural network for binary classification, as it will add neurons only when needed, unlike in network pruning, which will consume a lot of memory initially. Also, since the neurons are continuously added till no values exist in misclassified groups, the **accuracy** of the final network in classification will be very high.

REFERENCES

1. Jayadeva, Alok Kanti Deb , and Suresh Chandra, “Binary Classification by SVM Based Tree Type Neural Networks” IEEE 2002

2. B. Hassibi and D.G Stork, "Second-order derivatives for network pruning: Optimal Brain surgeon," Advances in Neural Information Processing Systems (NIPS), Vol. 5, pp. 164-171, 1993.

3. G. Martinelli, L.P. Ricotti, S. Ragazzini and F.M. Mascioli,"A Pyramidal Delayed Perceptron", IEEE Transactions on Circuits and Systems, Vol. 37:9, pp. 1176-1181, 1990.

4. F.M.F. Mascioli and G. Martinelli, "A Constructive Algorithm for Binary Neural Networks: The Oil-Spot Algorithm," IEEE Transactions on Neural Networks," Vol. 6, No. 3, pp. 794-797, May 1995.

5. M. Muselli, "On Sequential Construction of Binary Neural Networks," IEEE Transactions on Neural Networks, Vol. 6, No. 3, pp. 678-690, May 1995.