Chapter 1
Introduction

1.1 Overview

In our thesis work we have trained a network with some sample data and then test it with at different circumstances. Our goal is to optimize the system by trial and error method.

We suggest an optimization approach of cluster-based under sampling to select appropriate instances. This approach can solve the data imbalance problem, which can lead to knowledge extraction for improving the performance of existing data mining techniques. Although data mining techniques among various big data analytics technologies have been successfully applied and proven in terms of classification performance in various domains, such as marketing, accounting and finance areas, the data imbalance problem has been regarded as one of the most important issues to be considered. We have examined the effectiveness of a hybrid method using a clustering technique and Back-Propagation algorithms based on the artificial neural networks model to balance the proportion between the minority class and majority class. The objective of this paper is to constitute the best suitable training dataset for both decreasing data imbalance and improving the classification accuracy. We extracted the properly balanced dataset composed of optimal or near-optimal instances for the artificial neural networks model. The main contribution of the proposed method is that we extract explorative knowledge based on recognition of the data structure and categorize instances through the clustering technique while performing simultaneous optimization for the artificial neural networks modeling. In addition, we can easily understand why the instances are selected by the rule-format knowledge representation increasing the expressive power of the criteria of selecting instances. The proposed method is successfully applied to the bankruptcy prediction

problem using financial data for which the proportion of small- and medium-sized bankruptcy firms in the manufacturing industry is extremely small compared to that of non-bankruptcy firms. We suggest an optimization approach of cluster-based under sampling to select appropriate instances. This approach can solve the data imbalance problem, which can lead to knowledge extraction for improving the performance of existing data mining techniques. Although data mining techniques among various big data analytics technologies have been successfully applied and proven in terms of classification performance in various domains, such as marketing, accounting and finance areas, the data imbalance problem has been regarded as one of the most important issues to be considered. We examined the effectiveness of a hybrid method using a clustering technique and genetic algorithms based on the artificial neural networks model to balance the proportion between the minority class and majority class. The objective of this paper is to constitute the best suitable training dataset for both decreasing data imbalance and improving the classification accuracy. We extracted the properly balanced dataset composed of optimal or near-optimal instances for the artificial neural networks model. The main contribution of the proposed method is that we extract explorative knowledge based on recognition of the data structure and categorize instances through the clustering technique while performing simultaneous optimization for the artificial neural networks modeling. In addition, we can easily understand why the instances are selected by the rule-format knowledge representation increasing the expressive power of the criteria of selecting instances. The proposed method is successfully applied to the bankruptcy prediction problem using financial data for which the proportion of small- and medium-sized bankruptcy firms in the manufacturing industry is extremely small compared to that of non-bankruptcy firms.

Deep neural networks require massive amounts of data to be trained. In large-scale datasets, supervised methods have been successfully trained over the past few years due to the advances in parallel computing (Simonyan & Zisserman, 2014; Szegedyetal, 2014). Popular datasets such as ImageNet (Dengetal, 2009) contain more than a million labeled

samples, and even larger datasets are already sought after by researchers in the field. Further pushing the boundaries, video datasets are becoming increasingly important in the context of deep neural networks for event recognition tasks. In all such cases, labeling is necessary so that a supervised training algorithm can be used. However, the task of labeling data is quite expensive and time-consuming, requiring tedious work. For example, several hundreds of hours were spent to create ImageNet, and thousands of hours maybe needed to annotate even the simplest video dataset (Russakovsky et al., 2015). To circumvent this problem, the research community recognizes that a large breakthrough lies in the use of unlabeled data, which is freely available in abundant quantities.

Over the last few decades, extensive research has been dedicated to learning feature hierarchies for deep learning in the context of image understanding. Examples include unsupervised, supervised, and semi-supervised learning. Such deep learning techniques use hierarchy of layers, which use "filters" to extract multiple input features and "connections" to combine extracted features together into inputs for the next layer. In earlier studies in the field, unsupervised pre-training was required for training deep networks by supervised learning methods. Recent advances in Convolutional Neural Networks (ConvNets) combined with abundant amounts of labeled data have shown great promises in object recognition tasks to remedy this issue (Krizhevsky et al., 2012). On the other hand, unsupervised learning algorithms, such as k-means clustering, also increased the number of parameters in the network and achieved state-of-the-art results when labeled data are limited. Although unsupervised learning techniques using k-means algorithm were commonly used to train filters in several studies (Coates & Ng, 2011b; Bo et al., 2013), the network encoding structures present many similarities with ConvNets, such as the use of convolution and pooling in each layer.

We will do the analysis with clustering technique which means we will simply use a clustering algorithm first on the dataset. Once clustered we will train with the same network and with same test data we will check the performance of the network. Of course it will be better or worse than without clustering performance.

There are some other factors involved in clustering that the number of clusters, training the networks and many more. We will also check the number of cluster that the networks better and optimal too. For example : let's say on a data if we apply 3 number of cluster it performs good and if apply 4 number it will perform same so we will chose 3 as more cluster will make the system complicated.

Once we are done with K-means clustering we will try to check same things with some other clustering algorithms like fuzzy c-mean and more.

Our main goal is to analyze the clustering techniques in neural network and find out the optimal solution for different cases such as number of clusters, clustering techniques etc.

Before choosing this topic we have search for related works and what we found is that maximum time when someone needs to use neural network and clustering they just choose a clustering algorithm and number of cluster without thinking how many cluster will be optimal for their work or which algorithm they should choose. But with our system it will be much easier to choose the best algorithm and number of cluster.

Chapter 2

Literature Review

2.1 Neural Network Basics

Neural networks are members of a family of computational architectures inspired by biological brains (e.g., McClelland et al., 1986; luger and Stubblefield, 1993). Such architectures are commonly called “connectionist systems”, and are composed of interconnected and interacting components called nodes or neurons (these terms are generally considered synonyms in connectionist terminology, and are used interchangeably here). Neural networks are characterized by a lack of explicit representation of knowledge; there are no symbols or values that directly correspond to classes of interest. Rather, knowledge is implicitly represented in the patterns of interactions between network components (Lugar and Stubblefield, 1993). A graphical depiction of a typical feedforward neural network is given in Figure 2.1. The term “feedforward” indicates that the network has links that extend in only one direction. Expect during training, there are no backward links in a feedforward network; all links in feedforward network; all links proceed from input nodes toward output nodes.

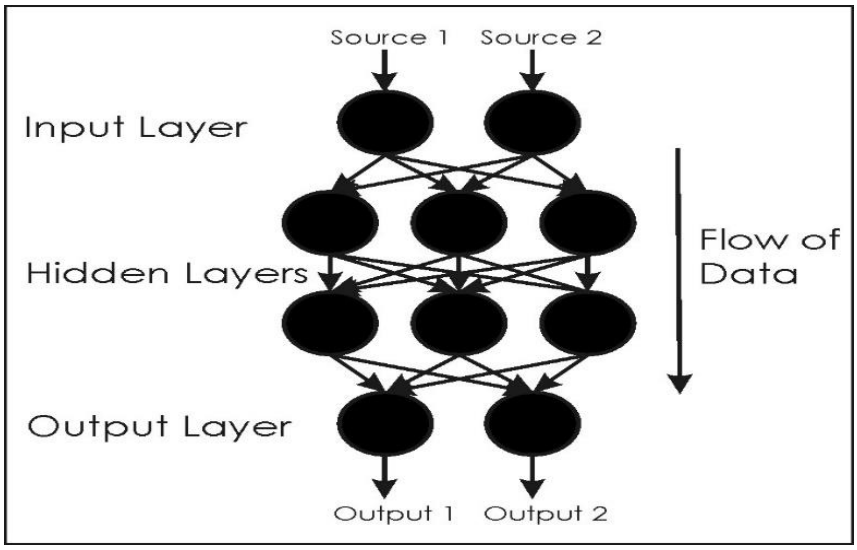


Figure 2.1: A typical feedforward neural network.

Individual nodes in a neural network emulate biological neurons by taking input data and performing simple operations on the data, selectively passing the results on to other neurons (Figure 2.2). The output of each node is called its "activation" (the terms "node values" and "activations" are used interchangeably here). Weight values are associated with each vector and node in the network, and these values constrain how input data (e.g., satellite image values) are related to output data (e.g., land-cover classes). Weight values associated with individual nodes are also known as biases. Weight values are determined by the iterative flow of training data through the network (i.e., weight values are established during a training phase in which the network learns how to identify particular classes by their typical input data characteristics). A more formal description of the foundations of multi-layer, feed forward, back propagation neural networks are given in next chapters.

Once trained, the neural network can be applied toward the classification are performed by trained networks through 1) the activation of network input nodes by relevant data sources [these data sources must directly match those used in the training of the network], 2) the forward flow of these data through the network, and 3) the ultimate activation of the output nodes. The pattern of activation of the networks output nodes determines the outcome of each pixels classification. Useful summaries of fundamental neural network principles are

given by Rumelhart et al. (1986), McClelland and Rumelhart (1988), Rich and Knight (1991), Winston (1991), Anzai (1992), Lugar and Stubblefield (1993), Gallant (1993), and Richards and Jia (2005). Parts of this web page draw on these summaries. A brief historical account of the development of connectionist theories is given Gallant (1993).

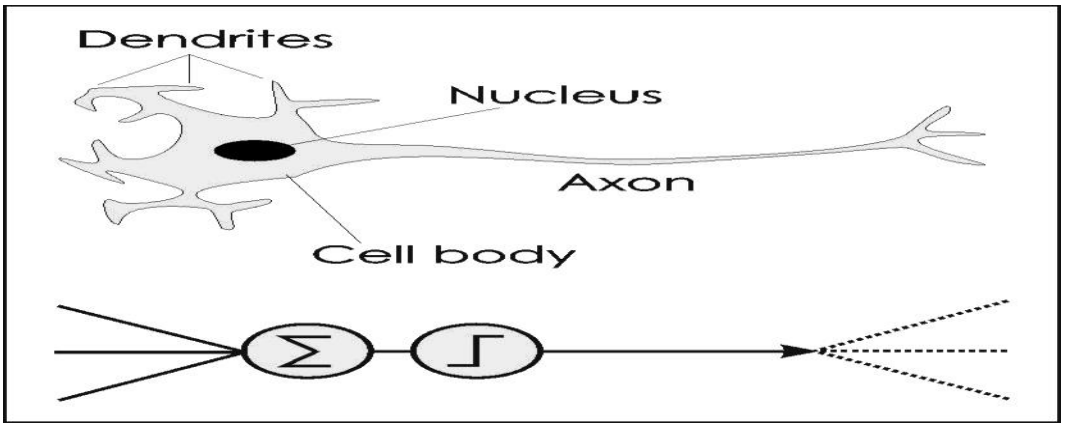


Figure 2.2 Schematic comparison between a biological neuron and an artificial neuron

2.1.1 McCulloch-Pitts Networks

Neural computing began with the development of the McCulloch-pitts network in the 1940's (McCulloch and Pitts, 1943; Luger and Stubblefield, 1993). These simple connectionist networks, shown in Figure 2.3, are stand-alone"decision machines" that take a set of inputs, multiply these inputs by associated weights, and output a value based on the sum of these products. Input values (also known as input activations) are thus related to output values (output activations) by simple mathematical operations involving weights associated with network links. McCulloch- Pitt's networks are strictly binary; they take as input and produce as output only 0's or 1's. These 0's and 1's can be thought of as excitatory or inhibitory entities, respectively (Luger and Stubblefield, 1993). If the sum of the products of the inputs and their respective weights is greater than or equal to 0, then the output nodes return a 1 (otherwise, a 0 is return). The value of 0 is thus a threshold that

must be exceeded if the output of the system is to be 1. The above rule, which governs the manner in which an output node maps input values to output values, is known as an activation function. McCulloch-Pitts networks can be constructed to compute logical functions (for example, in the "X AND Y" case, no combination X=Y=1). McCulloch-Pitts networks do not learn, and thus the weight values must be determined in advance using other mathematical or heuristic means. Nevertheless, these networks did much to inspire further research into connectionist models during the 1950's (Luger and Stubblefield, 1993).

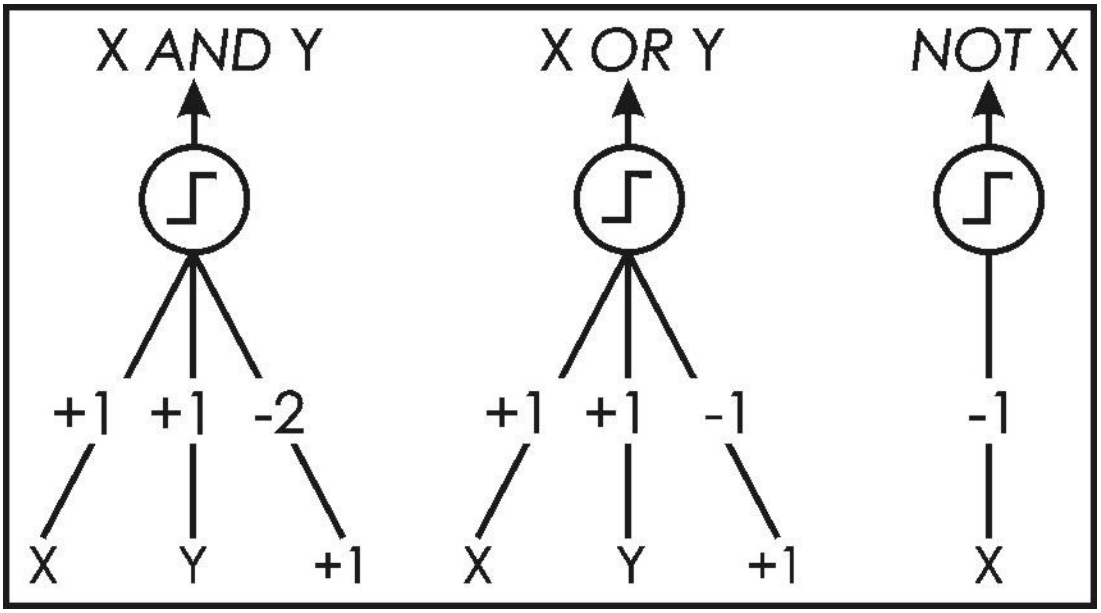


Figure 2.3: McCulloch-Pitts networks (after Luger and Stubblefield, 1993).

2.1.2 Architecture of Neural Networks

A neural network is composed of a number of nodes, or units, connected by links. Each link has a numeric weight associated with it. Weights are the primary means of long-term storage in neural networks, and learning usually takes place by updating the weights. Some of the units are connected to the external environment, and can be designated as input or

output units. The weights are modified so as to try to bring the network's input/output behavior more into line with that of the environment providing the inputs.

Each unit has a set of input links from other units, a set of output links to other units, a current activation level, and a means of computing the activation level at the next step in time, given its inputs and weights.

2.1.3 A simple Neuron

An artificial neuron is a device with many inputs and one output. The neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.

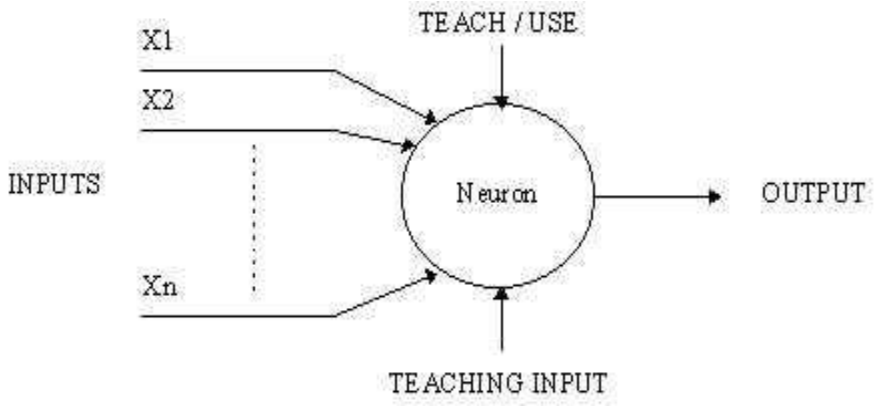


Figure 2.4: A simple Neuron.

2.1.4 Perceptron

The development of a connectionist system capable of limited learning occurred in the late 1950's, when Rosenblatt created a system known as a perceptron (see Rosenblatt, 1962; luger and Stubblefield, 1993). Again, this system consists of binary activations (inputs and outputs) (see Figure 2.4). In common with the McCulloch-Pitts neuron described above, the perceptron's binary output is determined by summing the products of inputs and their respective weight values. In the perceptron implementation, a variable threshold values is used (whereas in the McCulloch-Pitts network, this threshold is fixed at 0): if the linear sum of the input/weight products is greater than a threshold value (theta), the output of the system is 1 (otherwise, a 0 is returned). The output unit is thus said to be, like the perceptron output unit, a linear threshold unit. To summarize, the perceptron "classifies" input values as either 1 or 0, according to the following rule, referred to as the activation function:

PERCEPTRON OUTPUT = 1

If (sum of products of inputs and weights) > theta

(Otherwise, PRECEPTRON OUTPUT = 0) …. (Equation 2.1)

The perceptron is trained (i.e., the weights and threshold values are calculated) based on an iterative training phase involving training data. Training data are composed of a list of input values and their associated desired output values. In the training phase, the inputs and related outputs of the training data are repeatedly submitted to the perceptron. The perceptron calculates an output value for each set of input values. If the output of a particular training case is labeled 1 when it should be labeled 0, the threshold value (theta) is increased by 1, and all weight values associated with inputs of 1 are decreased by 1. The opposite is performed if the output of training case is labeled 0 when it should be labeled 1. No changes are made to the threshold value or weights if a particular training case is correctly classified. This set of training rules is summarized as:

If OUTPUT is correct, then no changes are made to the threshold or weights…….

(Equation 2.2a)

If OUTPUT = 1, but should be 0

then {theta = theta + 1}

and {weight = weight - 1, if input = 1} …(Equation 2.2b)

If OUTPUT = 0, but should be 1

then {theta = theta -1}

and {weight = weight +1, if input = 1} … (Equation 2.2c)

Where the subscript x refers to a particular input-node and weight pair. The effect of the above training rules is to make it less likely that a particular error will be made in subsequent training iterations. For example, in equation (2.2b), increasing the threshold values serves to make it less likely that the same sum of products will exceed the threshold in later training iterations, and thus makes it less likely that an output value of 1 will be produced when the same inputs are presented. Also, by modifying only those weights that are associated with input values of 1, only those weights that could have contributed to the error are changed (weights associated with input values of 0 are not considered to have contributed to error). Once the network is trained, it can be used to classify new data sets whose input/output associations are similar to those that characterized the training data set. Thus, through an iterative training stage in which the weights and threshold gradually migrate to useful values (i.e., values that minimize or eliminate error),

The perceptron can be said to “learn” how to solve simple problems.

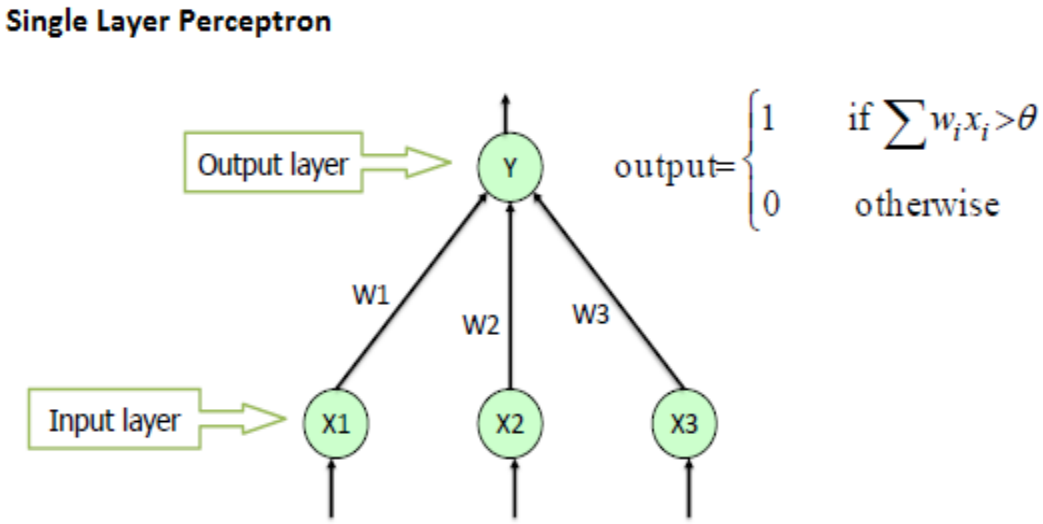


Figure 2.5: An example of a perceptron.

The system consists of binary activations. Weights are identified by w’s and inputs are identified by x’s. A variable threshold value (theta) is used at the output node.

2.1.5 Multi-Layer Networks and Backpropagation

Eventually, despite the apprehensions of earlier workers, a powerful algorithm for apportioning error responsibility through a multi-layer network was formulated in the form of the backpropagation algorithm (Rumelhart et al., 1986). The essential idea of backpropagation is to combine a non-linear multi-layer perceptron-like system capable of making decisions with the objective error function of the Delta Rule (McClelland and Rumelhart, 1988).

2.1.5.1 Network Terminology

A multi-layer, feedforward, backpropagation neural network is composed of

1) An output layer of nodes.

2) One or more intermediate (hidden) layers of nodes, and

3) An output layer of nodes (Figure 2.1).

The output layer can consist of one or more nodes, depending on the problem at hand. In most classification applications, there will either be a single output node (the value of which will identify a predicated class), or the same number of nodes in the output layer as there are classes (under this latter scheme, the predicted class for a given set of input data will correspond to that class associated with the output node with the highest activation). It is important to recognize that the term "multi-layer" is often used to refer to multiple layers of weights. This contrasts with the usual meaning of "layer", which refers to a row of nodes (Vemuri, 1992). For clarity, it is often best to describe a particular network by its number of nodes in each layer (e.g., a "4-3-5" network has an input layer with 4 nodes, a hidden layer with 3 nodes, and an output layer with 5 nodes).

2.1.5.2 The Sigma Function

The use of a smooth, non-linear activation function is essential for use in a multi-layer network employing gradient-descent learning. An activation function commonly used in backpropagation networks is the sigma (or sigmoid) function:

$$a_{j_m} = \frac{1}{\left(1+e^{-S_{j_m}}\right)} \text{ Where} S_{j_m} = \sum_{x=0}^{n} w_{i\,j_x} a_{i_x}$$

…… (Equation 2.3)

where a_j sub m is the activation of a particular "receiving" node m in layer j, Sj is the sum of the products of the activations of all relevant "emitting" nodes (i.e., the nodes in the preceding layer i) by their respective weights, and w_{ij} is the set of all weights between layers i and j that are associated with vectors that feed into node m of layer j. This function maps all sums into [0,1] (Figure 10) (an alternate version of the function maps activations into [-1, 1]; e.g., Gallant 1993, pp. 222-223). If the sum of the products is 0, the sigma function returns 0.5. As the sum gets larger the sigma function returns values closer to 1, while the function returns values closer to 0 as the sum gets increasingly negative. The derivative of the sigma function with respect to Sj sub m is conveniently simple, and is given by Gallant (1993, p.213) as:

$$\frac{d}{d(S_{j_m})}\left(1+e^{-S_{j_m}}\right)^{-1} = -1\left(1+e^{-S_{j_m}}\right)^{-2}e^{-S_{j_m}}(-1) = \frac{1}{1+e^{-S_{j_m}}}1-\left(\frac{1}{1+e^{-S_{j_m}}}\right) = a_{j_m}\left(1-a_{j_m}\right)$$

… (Equation 2.3a)

The sigma function applies to all nodes in the network, except the input nodes, whose values are assigned input values. The sigma function superficially compares to the threshold function (which is used in the perceptron) as shown in Figure 10. Note that the derivative of the sigma function reaches its maximum at 0.5, and approaches its minimum with values approaching 0 or 1. Thus, the greatest change in weights will occur with values near 0.5, while the least change will occur with values near 0 or 1. McClelland and Rumelhart (1988) recognize that it is these features of the equation (i.e., the shape of the function) that contribute to the stability of learning in the network; weights are changed most for units whose values are near their midrange, and thus for those units that are not yet committed to being either "on" or "off".

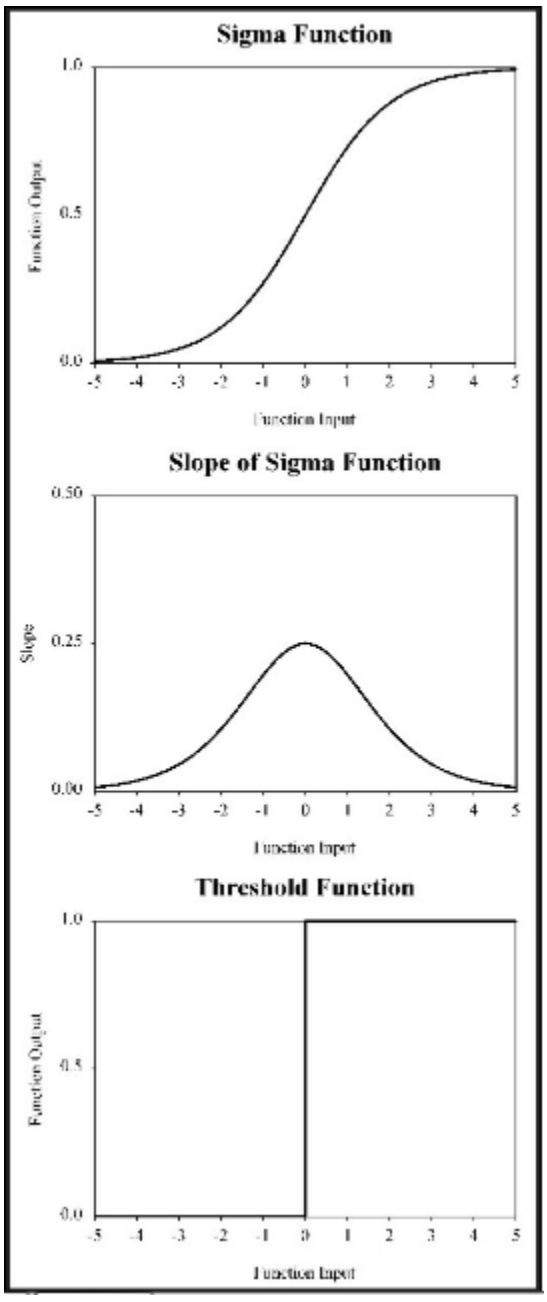


Figure 2.6: Graphs of sigma function, derivative and threshold function.

2.1.5.3 The Backpropagation Algorithm

In the employment of the backpropagation algorithm, each iteration of training involves the following steps: 1) a particular case of training data is fed through the network in a forward direction, producing results at the output layer, 2) error is calculated at the output nodes based on known target information, and the necessary changes to the weights that lead into the output layer are determined based upon this error calculation, 3) the changes to the weights that lead to the preceding network layers are determined as a function of the properties of the neurons to which they directly connect (weight changes are calculated, layer by layer, as a function of the errors determined for all subsequent layers, working backward toward the input layer) until all necessary weight changes are calculated for the entire network. The calculated weight changes are then implemented throughout the network, the next iteration begins, and the entire procedure is repeated using the next training pattern. In the case of a neural network with hidden layers, the backpropagation algorithm is given by the following three equations (modified after Gallant, 1993), where i is the "emitting" or "preceding" layer of nodes, j is the "receiving" or "subsequent" layer of nodes, k is the layer of nodes that follows j (if such a layer exists for the case at hand), ij is the layer of weights between node layers i and j, jk is the layer of weights between node layers j and k, weights are specified by w, node activations are specified by a, delta values for nodes are specified by d, subscripts refer to particular layers of nodes (i, j, k) or weights (ij, jk), "sub-subscripts" refer to individual weights and nodes in their respective layers, and epsilon is the learning rate:

$$\Delta W_{ij_m} = \varepsilon \delta_{j_p} a_{i_q} \quad \ldots\ldots\ldots\ldots \qquad \text{(Equation 2.4a)}$$

where $\delta_{j_p} = a_{j_p}\left(1 - a_{j_p}\right)\left(t_{j_p} - a_{j_p}\right)$ if output node …

(Equation 2.4b)

Where $\delta_{j_p} = a_{j_p}\left(1 - a_{j_p}\right)\Sigma_{x=0}^{n}\delta_{k,}w_{jk}$ if intermediate node

.....(Equation 2.4c)

Being based on the generalized Delta Rule, it is not surprising that Equation (2.4a) has the same form. Equation (2.4a) states that the change in a given weight m located between layers i and j is equal to the products of: 1) the learning rate (epsilon); 2) the delta value for node p in layer j [where node p is the node to which the vector associated with weight m leads]; and 3) the activation of node q in layer i [where node q is the node from which the vector associated with weight m leads]. In practice, the learning rate (epsilon) is typically given a value of 0.1 or less; higher values may provide faster convergence on a solution, but may also increase instability and may lead to a failure to converge (Gallant, 1993). The delta value for node p in layer j in Equation (2.4a) is given either by Equation (2.4b) or by Equation (2.4c), depending on the whether or not the node is in an output or intermediate layer.

2.1.5.4 Bias

Equations (2.4a), (2.4b), and (2.4c) describe the main implementation of the backpropagation algorithm for multi-layer, feedforward neural networks. It should be noted, however, that most implementations of this algorithm employ an additional class of weights known as biases. Biases are values that are added to the sums calculated at each node (except input nodes) during the feedforward phase. That is, the bias associated with a particular node is added to the term S_j .The negative of a bias is sometimes called a threshold (Bishop, 1995a).

For simplicity, biases are commonly visualized simply as values associated with each node in the intermediate and output layers of a network, but in practice are treated in exactly the

same manner as other weights, with all biases simply being weights associated with vectors that lead from a single node whose location is outside of the main network and whose activation is always 1. The change in a bias for a given training iteration is calculated like that for any other weight [using Equations (2.4a), (2.4b), and (2.4c)], with the understanding that a_i sub m in Equation (2.4a) will always be equal to 1 for all biases in the network. The use of biases in a neural network increases the capacity of the network to solve problems by allowing the hyper planes that separate individual classes to be offset for superior positioning. More specific discussions on the utility of biases in neural networks are given by, e.g., Gallant (1993, pp.65-66), Bishop (1995a, p.78), and Reed and Marks (1999, pp.15-17).

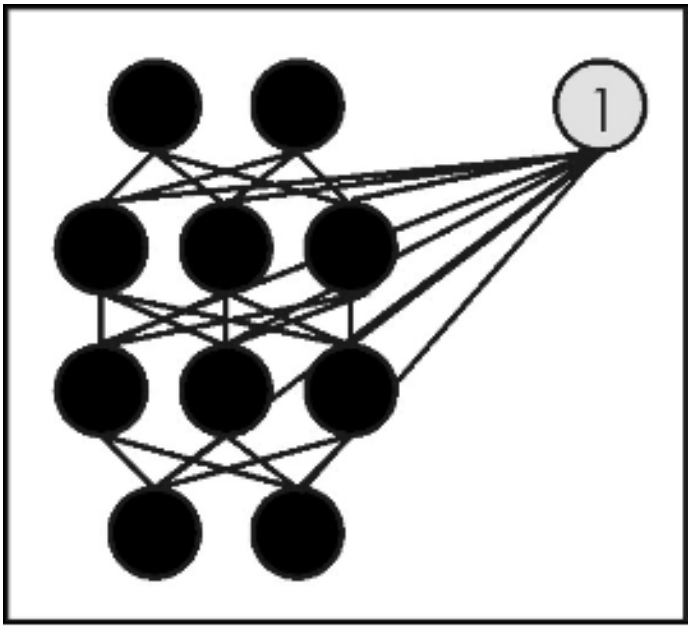


Figure 2.7: Biases are weights associated with vectors that lead from a single node whose location is outside of the main network and whose activation is 1.

2.1.5.5 Network Topology

The precise network topology required to solve a particular problem usually cannot be determined (Figure 12; Castellano et al., 1997; Tamura and Tateishi, 1997; Reed and Marks, 1999; Richards and Jia, 2005), although research efforts continue in this regard. This is a critical problem in the neural-network field, since a network that is too small or too large for the problem at hand may produce poor results. This is analogous to the problem of curve fitting using polynomials: a polynomial with too few coefficients cannot evaluate a function of interest, while a polynomial with too many coefficients will fit noise data.

General "rules of thumb" regarding network topology are commonly used. At least one intermediate layer is always used; as noted in Section 4.2, even simple problems such as the exclusive-OR problem cannot be solved without intermediate layers. Many applications of the backpropagation algorithm involve the use of networks consisting of only one intermediate layer of nodes, although the use of two intermediate layers can generate superior results for certain problems in which higher order functions are involved (Gallant, 1993; Reed and Marks, 1999, p.38). The number of nodes used in each intermediate layer is typically between the number of nodes used for the input and output layers (e.g., Richards and Jia, 2005). Ultimately, the only method that can be confidently used to determine the appropriate number of layers in a network for a given problem is the trial and error.

An experimental means for determining an appropriate topology for solving a particular problem involves the training of a larger-than-necessary network, and the subsequent removal of unnecessary weights and nodes during training. This approach, called pruning, requires advance knowledge of initial network size, but such upper bounds may not be difficult to estimate. An alternative means for determining appropriate network topology involves algorithms which start with a small network and build it larger; such algorithms are known as constructive algorithms. Additionally, much neural network research remains

focused on the use of evolutionary and genetic algorithms, based on simplified principles of biological evolution, to determine network topology, weights, and overall network behavior.

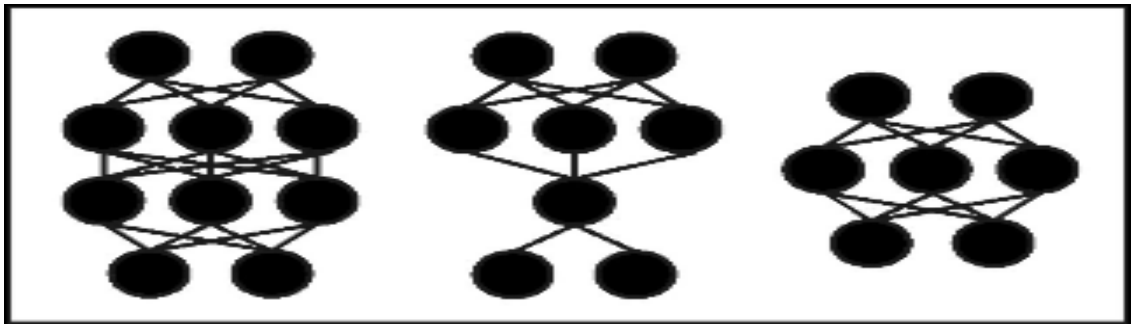


Figure 2.8: Three of an infinite number of possible network topologies that could be used to relate two inputs to two outputs.

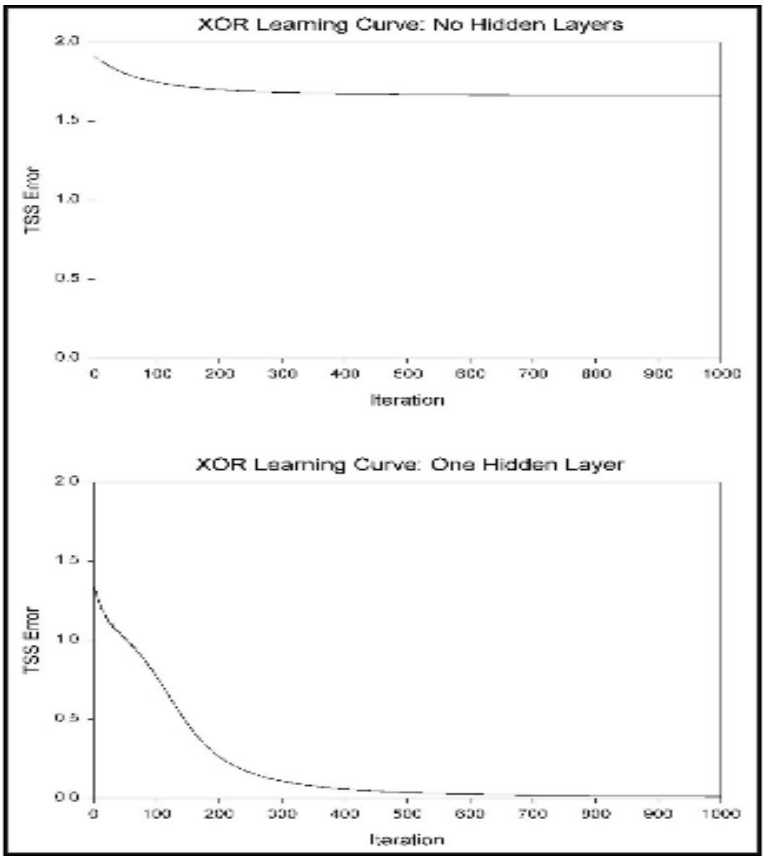


Figure 2.9: Learning curves for the exclusive-OR (XOR) problem (Leverington, 2001).

A neural network with no hidden layers cannot solve the XOR problem (top curve), but a neural network with a single hidden layer can solve the problem with a relatively small number of iterations (bottom curve).

2.1.5.6 Initialization of Network Weights

Although the ideal initial values for weights (i.e., those that will maximize the effectiveness and speed with which a neural network learns) cannot yet be determined theoretically (Thimm and Fiesler, 1997), it is general practice to assign randomly-generated positive and negative quantities as the initial weight values. Such a random distribution can help minimize the chances of the network becoming stuck in local minima (Gallant, 1993). Typically, values are selected from a range [-a,+a] where $0.1 < a < 2$ (Reed and Marks, 1999, p.57). The reason for using random initial weights is to break symmetry, while the reason for using small initial weights is to avoid immediate saturation of the activation function (Reed and Marks, 1999, p.97). Further discussions regarding the benefits of the use of small initial weights are given by Reed and Marks (1999, p.116 and p.120).

2.1.5.7 Momentum

The speed of convergence of a network can be improved by increasing the learning rate epsilon. Unfortunately, increasing e will usually result in increasing network instability, with weight values oscillating erratically as they converge on a solution. Instead of changing e, most standard backpropagation algorithms employ a momentum term in order to speed convergence while avoiding instability. The momentum term is added to Equation 2.4a, and is equal to the product of some fraction 0 <= alpha <= 1 by the change in weight that occurred in the previous weight change. By adding fractions of previous weight changes, weight changes can be kept on a faster and more even path (Gallant, 1993). Three learning curves, each generated using a different momentum setting. The first

learning curve was generated with momentum set to zero (that is, no momentum). The second learning curve was generated using momentum on all non-bias weights (here termed half-momentum); note that although the learning curve is characterized by greater iteration-to-iteration variation in tss error, the network using half-momentum converged to a better solution (lower tss error) using a smaller number of iterations, compared with the network using zero momentum. The third learning curve applied momentum to all weights, included biases (here termed full-momentum); the network using full-momentum did not learn as quickly or as successfully as the network using half-momentum, and the iteration-to-iteration variation in tss error was greatly increased.

The value for alpha is typically set to 0.9, with the learning rate set to 0.1. A useful discussion of considerations relevant to the choice of both learning rate epsilon and momentum alpha is given by Reed and Marks (1999, pp.74-77 and 87-90).

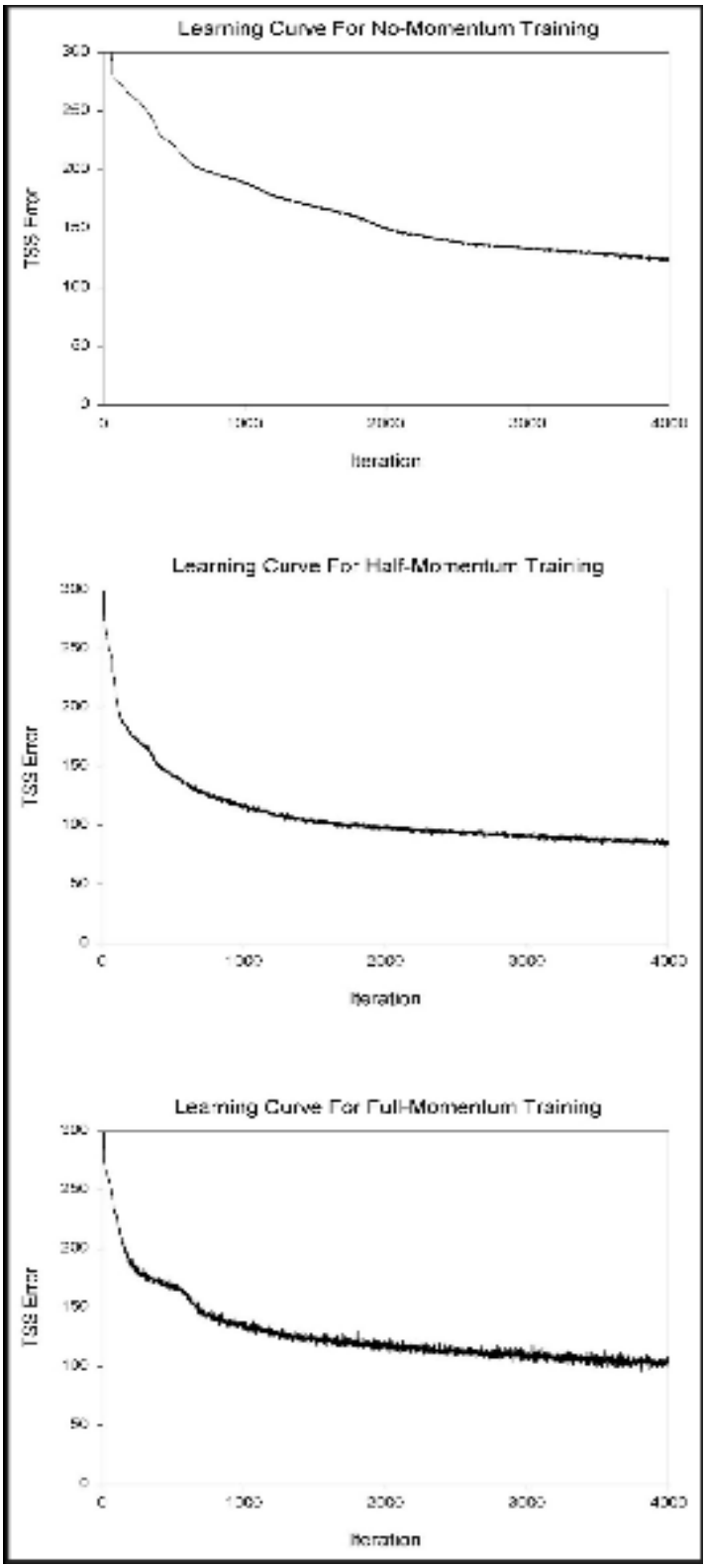


Figure 2.10: Learning curves produced using three different momentum settings.

2.1.5.8 Over-Generalization and Training with Noise

The purpose of training a feedforward backpropagation network is to modify weight values iteratively such that the weights, over time, converge on a solution that relates inputs to outputs in a manner that is useful to the user. It is normally desirable in training for a network to be able to generalize basic relations between inputs and outputs based on training data that do not consist of all possible inputs and outputs for a given problem. A difficulty that can arise in the training of a neural network involves the adaptation of weight values so closely to training data that the utility of the network in processing new data is diminished; this problem is called over-generalization (or over-training) (e.g., Bishop, 1995a; Reed and Marks, 1999; Karystinos and Pados, 2000). In over-generalization, the network has ceased to be able to relate general ranges of values to classes, and instead relates input values to classes in a manner that is contrary to the more general relation that is desired. While early stages of learning generally result in the successful evaluation of the main features of the underlying function that properly maps inputs to outputs, later stages of learning may incorporate features of the training dataset that are uncharacteristic of the data as a whole (e.g., due to the use of incomplete or noisy training data) (Reed and Marks, 1999). Measured error in such a situation will continually be decreasing during training, but the generalization capabilities of the network will also be decreasing (as the network modifies weights to suit peculiarities of the training data being used).

One approach used to prevent over-generalization is the termination of training before over-generalization can occur (Reed and Marks, 1999). That is, for a given network, training data, and learning algorithm, there may be an optimal amount of training that produces the best generalization. While potentially effective, this solution is often not useful, since algorithms that provide information on approximately when to stop training are not necessarily dependable; such algorithms may, for example, be fooled into the premature termination of training by factors such as, e.g., local minima in the error surface. A premature halt to training will result in a network that is not trained to its highest

potential, while a late halt to training can result in a network whose operation is characterized by over-generalization (Reed and Marks, 1999).

Another solution that is sometimes used to help combat over-generalization is the use of jitter (i.e., the addition of a small amount of artificial noise to training data while a network is being trained) (e.g., Bishop, 1995a, pp.346-349; Bishop, 1995b; Reed and Marks, 1999, pp.277-289). This is performed by adding a random vector to each training pattern each time it is submitted to the network. The addition of noise to training data allows values that are proximal to true training values to be taken into account during training; as such, the use of jitter may be thought of as a means of extrapolating training data to proximal values. The use of jitter helps to fill out otherwise sparse training datasets by relating a greater variety of input values (and combinations of input values) to output classes, allowing the network's weights to converge on a solution that relates general ranges of values to class labels, rather than relating specific values to class labels (and thus potentially avoiding over-generalization).

2.1.5.9 Combining Neural Network Results

The random initialization of network weights prior to each execution of the neural network training algorithm can in some cases cause final classification results to vary from execution to execution, even when all other factors (e.g., training data, learning rate, momentum, network topology) are kept constant. Particularly when working with very limited training datasets, the variation in results can be large. Under such circumstances, it is best to expand training data on the basis of improved ground truth. If this is not possible, generation of optimum results can sometimes be made through combination of the results of multiple neural network classifications. For example, multiple neural network results can be combined using a simple consensus rule: for a given pixel, the class label with the largest number of network "votes" is that which is assigned (that is, the results of the

individual neural-network executions are combined through a simple majority vote) (Hansen and Salamon, 1990). The reasoning behind such a consensus rule is that a consensus of numerous neural networks should be less fallible than any of the individual networks, with each network generating results with different error attributes as a consequence of differing weight initializations (Hansen and Salamon, 1990). Of interest in the neural network community is the use of consensus algorithms to generate final results that are superior to any individual neural network classification.

2.1.6 Activation Function of ANN

In computational networks, the activation function of a node defines the output of that node given an input or set of inputs. A standard computer chip circuit can be seen as a digital network of activation functions that can be "ON"(1) or "OFF"(0), depending on input. This is similar to the behavior of the linear perceptron in neural networks. However, it is the nonlinear activation function that allows such networks to compute nontrivial using only a small number of nodes.

A neural networks loosely models biological synapse and neurons. Neural network (NN) classifiers have two activation functions. One activation function is used when computing the values of nodes in the middle, hidden layer, and one function is used when computing the value of the nodes final, output layer.

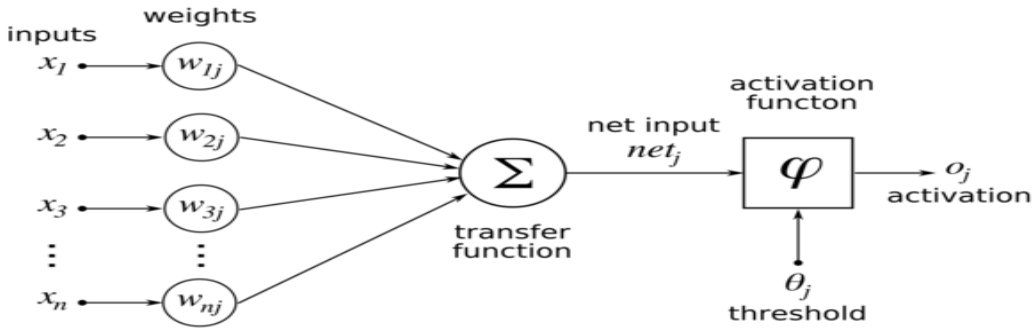


Figure 2.11: Process of activation function.

The activation function of a node in a neural network defines the output of the node, given a set of predetermined inputs. We can conceptualize ad activation function by using the example of something as simple as an AND gate in electronics. Note that the combinational nature of an AND gate implies that the output will either be 0 or 1, depending solely on the inputs. The root of such a function is biologically inspired; the function is a representation of action potential firing in a neuron. In its simplest form, this function is binary as demonstrated above. This introduces non-linarites that are desirable in multi-layer networks in order to detect non-linear features in the data. If only linear activation function would be applied, then the network would just apply a couple of linear functions which is again a linear function. Inspired by the neurons work in the brain, they either fire or not, hence a binary output resembles this property. We also need the function for practical purposes – to limit values to a probabilistic setup between -1 and 1 in a network. The commonly used model in multilayer perceptron is then a sigmoidal activation function, defined in the form of a hyperbolic tangent, normalized from -1 to 1 and vertically translated between 0 and 1.

2.1.6.1 Step Function

A step function is a function that is used by the original Perceptron. The output is a certain value, A1, if the input sum is above a certain threshold and A0 is the input sum is below a certain threshold. The values used by the Perceptron were A1 = 1 and A0 = 0.

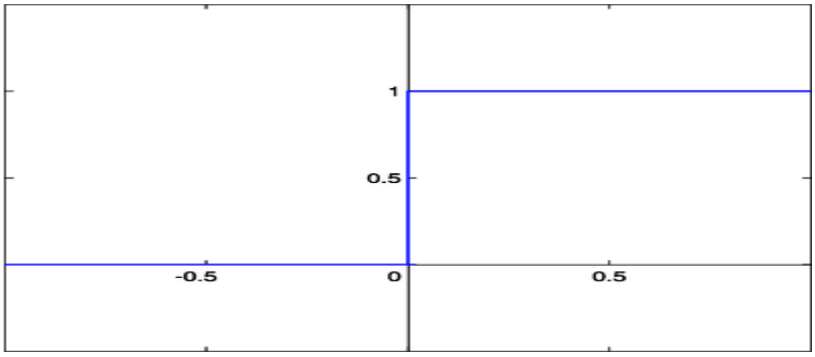


Figure 2.12: Step Function.

These kinds of step activation are useful for binary classification schemes. In order words, when we want to classify an input pattern into one of two groups, we can use a binary classifier with a step activation function. Another use for this would be to create a set of small identifiers. Each identifier would be a small network that would output a 1 if a particular input feature is present and a 0 otherwise. Combining multiple feature detectors into a single network would allow a very complicated clustering or classification problem to be solved.

Computing the values of an NN classifier's output nodes always uses the activation function because the sum of the output values is 1,0 and so the values can be interpreted as probabilities for each class. For example, if your goal is to predict the political leaning of a person based on predictor features such as age, annual income, state of residence and so on, then the NN classifier will have three output nodes. A set of output values might be something like (0.20, 0.70, 0.10), which would indicate the predicted political leaning is moderate.

There are two common activation functions used for NN hidden layer nodes, the logistic sigmoid function and the hyperbolic tangent function. But using other hidden layer activation functions is possible.

2.1.6.2 The Effect of Activation Function on Training

Training an NN is the process of finding values for the weights and biases. This is done by examining different weight and bias values to find the values where the difference between computed output values and the known target output values in the training data is minimized.

There are several different NN training algorithms, but by far the most common technique is called back-propagation. Back-propagation uses the calculus derivative of the hidden layer activation function. College calculus is likely to be a bit rusty. Suppose a math function is y = 4x^3. The derivative of the function is y' =12x^2. The point is that the derivative of a function almost always depends on x, the input value. The log-sigmoid function is y = 1 / (1 + e^-x) where e is the special math constant 2.71828. So you'd expect the derivative to have the term x in it. However, due to some algebra coincidences, the derivative of log-sigmoid is (y)(1-y). In other words, the derivative of log-sigmoid can be expressed in terms of the output value y. As we'll see shortly, this is very convenient when coding.

Similarly, the tanh function is y = (e^x – e^-x) / (e^x + e^-x). It's derivative is (1 - y)(1 + y). Again, by an algebra coincidence, the derivative of the tanh function can be expressed in terms of the output value y.

All reasonable alternative NN activation functions have derivatives that contain x. For example, the arctan function is y = 1 / 2 * log((1 + x) / (1 - x)). It's derivative is 1 / (1 – x^2).

The so-called stochastic version of the back-propagation algorithm, expressed in high-level pseudo-code with a few important details omitted, is:

Algorithm 1: Stochastic version of the back-propagation algorithm

1. Loop until satisfied
2. For each training item
3. Peel off training input values and target output values
4. Compute NN output values using current weights
5. Compute derivative of output layer activation function
6. Compute derivative of hidden layer activation function
7. Use derivatives to update weights and biases
8. End-for
9. End-loop
10. Return final weight and bias values.

If we examine the pseudo-code you'll notice the derivative of the hidden layer activation function is needed. But just before the derivative is needed, the hidden nodes values are calculated as part of the process that computes the values of the output nodes. These hidden node values are exactly what are needed to compute the derivative of the log-sigmoid function or the tanh function. This makes coding relatively easy.

2.1.7 Why do we use ANN

Neural Networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of an "expert" in the category of information it has been given to analyze. This

expert can then be used to provide projections given new situations of interest and answer "what if" questions.

Other advantages include:

Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

Self-Organization: An ANN can create its own organization or representation of the information it receives during learning time.

Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

2.2 What is Clustering?

Clustering is the process of grouping data in clusters, where objects within each cluster have high similarity, but are dissimilar to the objects in other clusters. Similarities are assessed based on the attribute values that best describes the object. Often distance measures are used for the purpose. Clustering has its roots in many areas, including data mining, statistics, biology, and machine learning. Among the various clustering algorithms, K-Means (KM) is one of the most popular methods used in data analysis due to its good computational performance. However, it is well known that KM might converge to a local optimum, and its result depends on the initialization process, which randomly generates the initial clustering. In other words, different runs of KM on the same input data might produce different results.

2.2.1 K-Means Clustering

K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. K-means clustering aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both algorithms. Additionally, they both use cluster centers to model the data; however, *k*-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has a loose relationship to the k-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with k-means because of the *k* in the name. One can apply the 1-nearest neighbor classifier on the cluster centers obtained by *k*-means to classify new data into the existing clusters. This is known as nearest centroid classifier or Rocchio algorithm.

K-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriority. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group

age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this algorithm aims at minimizing an objective function know as squared error function given by:

$$J(V) = \sum_{i=1}^{C} \sum_{j=1}^{C_i} (\|x_i - v_j\|)^2$$

where,

'$||x_i - v_j||$' is the Euclidean distance between x_i and v_j.

'c_i' is the number of data points in i^{th} cluster.

'c' is the number of cluster centers.

Algorithm 2: Steps for k-means clustering

Let $X = \{x_1, x_2, x_3, \ldots\ldots, x_n\}$ be the set of data points and $V = \{v_1, v_2, \ldots\ldots, v_c\}$ be the set of centers.

1) Randomly select *'c'* cluster centers.

2) Calculate the distance between each data point and cluster centers.

3) Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.

4) Recalculate the new cluster center using:

$$v_{i=(1/c_i)} \sum_{j=1}^{c_i} x_i$$

Where *'c_i'* represents the number of data points in i^{th} cluster.

5) Recalculate the distance between each data point and new obtained cluster centers.

6) If no data point was reassigned then stop, otherwise repeat from step 3).

2.2.2 Advantages of K-Means Clustering

1) Fast, robust and easier to understand.

2) Relatively efficient: O(tknd), where n is # objects, k is # clusters, d is # dimension of each object, and t is # iterations. Normally, k, t, d << n.

3) Gives best result when data set are distinct or well separated from each other.

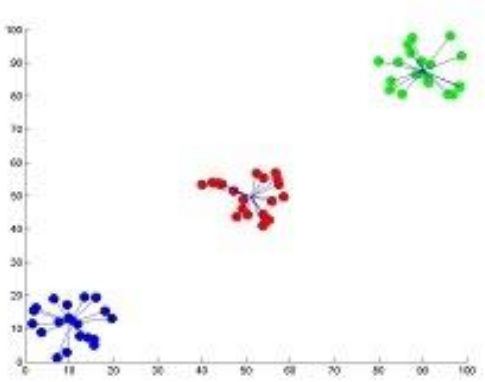


Figure 2.13: Showing the result of k-means for 'N' = 60 and 'c' = 3

2.2.3 Disadvantages of K-Means Clustering

1) The learning algorithm requires a priori specification of the number of cluster centers.

2) The use of exclusive assignment - If there are two highly overlapping data then k-means will not be able to resolve that there are two clusters.

3) The learning algorithm is not invariant to non-linear transformations i.e. with different representation of data we get different results.

4) Euclidean distance measures can unequally weight underlying factors.

5) The learning algorithm provides the local optima of the squared error function.

6) Randomly choosing of the cluster center cannot lead us to the fruitful result.

7) Applicable only when mean is defined i.e. fails for categorical data.

8) Unable to handle noisy data and outliers.

9) Algorithm fails for non-linear data set.

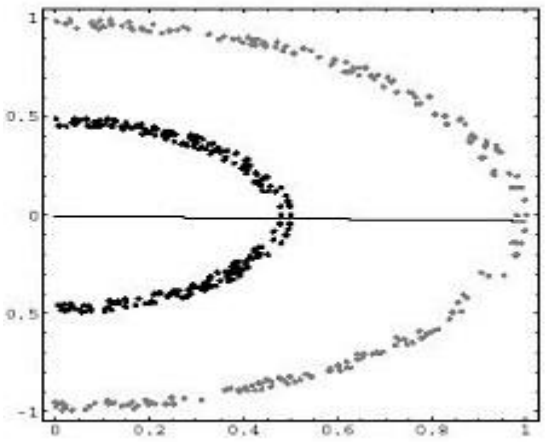


Figure 2.14: Showing the non-linear data set where k-means algorithm fails.

2.2.4 Fuzzy C-Mean Clustering

Fuzzy clustering (also referred to as soft clustering) is a form of clustering in which each data point can belong to more than one cluster. The fuzzy c-means algorithm is very similar to the *k*-means algorithm. Fuzzy c-means clustering works in a way that choose a number of clusters, assign randomly to each point coefficients for being in the clusters, repeat until the algorithm has converged, Compute the centroid for each cluster, For each point, compute its coefficients of being in the clusters. Any point x has a set of coefficients giving the degree of being in the kth cluster $w_k(x)$. With fuzzy c-means, the centroid of a cluster is the mean of all points, weighted by their degree of belonging to the cluster.

Algorithm 3: Steps for Fuzzy C-Mean clustering

Let X = {x_1, x_2, x_3 ..., x_n} be the set of data points and V = {v_1, v_2, v_3 ..., v_c} be the set of centers.

1) Randomly select *'c'* cluster centers.

2) Calculate the fuzzy membership *'μ_{ij}'* using:

$$\mu_{ij} = 1/\sum_{k=1}^{c} \left({}^{d_{ij}}/_{d_{ik}} \right)^{(2lm-1)}$$

3) Compute the fuzzy centers *'v_j'* using:

$$v_j = \left(\sum_{i=1}^{n} (\mu_{ij})^m x_i\right) / \left(\sum_{i=1}^{n} (\mu_{ij})^m\right), \forall_{j=1,2.....c},$$

4) Repeat step 2) and 3) until the minimum *'J'* value is achieved or $||U^{(k+1)} - U^{(k)}|| < \beta$.

where,

'k' is the iteration step.

'β' is the termination criterion between [0, 1].

'$U = (\mu_{ij})_{n*c}$' is the fuzzy membership matrix.

'J' is the objective function.

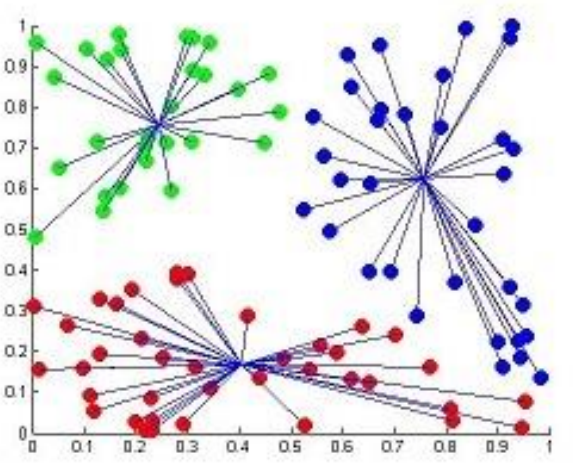


Figure 2.15: Result of Fuzzy c-means clustering.

2.2.5 Fuzzy C-Mean Advantages

1) Gives best result for overlapped data set and comparatively better then k-means algorithm.

2) Unlike k-means where data point must exclusively belong to one cluster center here data point is assigned membership to each cluster center as a result of which data point may belong to more than one cluster center.

2.2.6 Fuzzy C-Mean Disadvantages

1) A priori specification of the number of clusters.

2) With lower value of β we get the better result but at the expense of more number of iteration.

3) Euclidean distance measures can unequally weight underlying factors.

2.3 MATLAB

MATLAB is a multi-paradigm numerical computing environment and fourth-generation programming language. A propriety programming language developed by MathWorks, MATLAB allows matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python.

Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems.

2.3.1 Neural Network Tools

Neural network tools or "nntool" is basically a simulator that we have used in our work. Unlike the research simulators, nntool simulators are intended for practical applications of artificial neural networks. Their primary focus is on data mining and forecasting. Data analysis simulators usually have some form of preprocessing capabilities. Unlike the more general development environments data analysis simulators use a relatively simple static neural network that can be configured. A majority of the data analysis simulators on the market use backpropagating networks or self-organizing maps as their core. The advantage of this type of software is that it is relatively easy to use.

Our name of the neural network is 'Feed-Forward Neural Network' & we have custom info for the implementation purpose. Our NN dimensions (simply get from command line by writing net.particular_dimension) have some following properties:

#net.numInputs: Neural network numInputs property defines the number of inputs receives. It can be set to 0 or a positive integer. The number of network inputs and the size of a network input are not the same thing. The number of inputs defines how many sets of vectors the network receives as input. The size of each input (i.e., the number of elements in each input vector) is determined by the input size (net.inputs{i}.size).

We have net.numInputs = 1 & net.inputs{1}.size = 11.

Any change to this property results in a change in the size of the matrix defining connections to layers from inputs (net.inputConnect), the cell array of input subobjects (net.inputs), and the cell array of weight values (net.1W).

#net.numLayers: Neural network numLayers property defines the number of layers a network has. It can be set to 0 or a positive interger. For this case we have 4 layers.

Any change to this property changes the size of each of these Boolean matrices that define connections to and from layers (such as biasConnect, inputConnect, outputConnect) and changes the size of each cell array of subobject structures whose size depends on the number of layers (such as biases, layerWeights, outputs). Most importantly its change has an impact on the size of each of the networks's adjustable parameter's properties (net.1W, net.LW, net.b).

#net.numOutputs: This read-only property indicates how many outputs the network has. It is always equal to the number of 1s in net.OutputConnect. we will discuss about this connection part.

#net.numInputDelays: This is also read-only property which indicates the number of time steps past inputs that must be supplied to simulate the network. It is always set to the maximum delay value associated with any of the network's input weights. In this case it is 0.

#net.numLayerDelays: Another read-only property which is the number of time steps of past layer outputs that must be supplied to simulate the network. It is always set to the maximum delay value associated with any of the network's layer weights. In this case it is 0.

#net.numFeedbackDelays: This property indicates the number of time steps ahead a dynamic network predicts outputs. It is always set to the maximum delay value associated with any of the network's outputs (net.outputs{i}.feedbackDelay). It is basically used in dynamic networks to format time series data properly.

#net.numWeightElements: This read-only property indicates the number of weight and bias values in the network. It is the sum number of elements in the matrices stored in the two cell arrays, such as net.IW, net.LW, net.b . We have 432 values for this network structure.

#net.sampleTime: This property specifies the sample time to be used for dynamic network updates in Simulink block diagrams generated with gensim.

Connections section have also some properties as follows:

net.biasConnect property defines which layers have biases. It can be set to any N-by-1 matrix of Boolean values, where *Nl* is the number of network layers (net.numLayers). The presence of a bias to the *ith* layer is indicated by a 1 at net.biasConnect(i). In this case we have [1; 1].

net.inputConnect defines which layer have weights coming from inputs. It can be set to any *Nl x Ni* matrix of Boolean values, where *Nl* is the number of network layers (net.numLayers), and *Ni* is the number of network inputs (net.*numInputs*). The presence (or absence) of a weight going to the *ith* layer from the *jth* input is indicated by a 1 (or 0)

at net.inputConnect(i,j). in this case we have [1: 0] . Any change on this structure has an impact on input weight sub-objects (net>inputWeights) & input weight matrix (net.IW).

net.layerConnect This property defines which layers have weights coming from other layers. It can be set to any *Nl x Ni* matrix of Bolean values, where *Nl* is the number of network layers (net.numLayers). We have this [2x2 Boolean]

0 0

1 0

While changing it has effect of net.layerWeights & net.LW of the network.

net.outputConnect defines which layers generate network outputs. It can be set to any *1 x Nl* matrix of Boolean values, where *Nl* is the number of network layers (net.numLayers). Here we have [0 1].

Subobject has also some properties like inputs, layers, outputs, biases, inputWeights, layerWeights. These can be discussed briefly.

net.inputs – This property holds structures of properties for each of the network's inputs. It is always an *Ni x 1* cell array of input structures, where *Ni* is the number of network inputs.

net.layers – This property holds structures of properties for each of the network's layers. It is always an *Nl x 1* cell array of layer structures, where *Nl* is the number of network inputs.

net.outputs – This property holds structures of properties for each of the network's outputs. It is always *a1 x N1* cell array, where *Nl* is the number of network outputs.

net.biases – This property holds structures of properties for each of the network's biases. It is always *Nl x 1* cell array, where *Nl* is the number of network layers

We have different functions used here & also have some effects:

divideFnc defines the data division function to be used when the network is trained using a supervised algorithm, such as backpropagation. Whenver this property is altered, the network's adaption parameters (net.divideParam) are set to contain the parameters and default values of the new finction. Here we use dividerand which means divide targets into three sets using random indices.

[trainInd, valInd, testInd] = divideratio(Q, trainRatio, valRatio, testRatio)

Where input arguments means Number of targets to divideup, Ratio of vectors for training (default 0.7), validation (default 0.15) & testing (default 0.15) respectively. And it also returns indices of training, validation & testing respectively.

divideParam defines the parameters and values of the current data division function. It has three parameters:

***net.divideParam.trainRatio** is a data division function parameter. It must be a strictly positive scalar. It is the relative number of training samples to be selected from all samples by a data division function, compared to the relative numbers of validation samples valRatio, and test samples testRatio.

***net.divideParam.valRatio & *net.divideParam.testRatio** has also similar properties like as above, for validation ratio & test ratio function parameter.

We have **adapt function (**which means the adapt function to update the network during adaptive simulation with the function adapt**)** named **'adaptwb'** which is a network function which updates each weight and bias according to its learning function.

net.layers{i}.initFunc - Function for layer by layer network initialization called 'initlay' is a network initialization function that initializes each layer I according to its own initialization function. And also return the network with each layer updated.

net.performFunc – It defines the performance function used to measure a network's usefulness during training with train and for direct performance calculations with perform. For this, Mean squared normalized error performance function called 'mse' is used which is a network performance function. It measures the network's performance according to the mean of squared errors. It returns the mean squared error. It has a threshold value for which the variation occurs.

net.performParam – property defines the parameters and values of the current performance function net.performFunc. This function has two optional parameters, which are associated with networks whose net.trainFunc is set to this function:

‘regularization’ can be set to any value between 0 and 1. The greater the regularization value, the more squared weights and biases are included in the performance calculation relative to errors. The default is 0, corresponding to no regulation.

‘normalization’ can be set to ‘none’ (the default); ‘standard’, which normalizes errors between -2 and 2, corresponding to normalizing outputs and targets between -1 and 1; and ‘percent’, which normalizes errors between -1 and 1. This feature is useful for networks with multi-element outputs. It ensures that the relative accuracy of output elements with differing target value ranges are treated as equally important, instead of prioritizing the relative accuracy of the output element with the largest target value range.

net.trainFunc – have a function called ‘trainlm’ which is a network training function that updates weight & bias values according to Levenberg- Marquardt optimization.

Chapter 3

Proposed Model

3.1 Data Selection

For our research we have used three sets of different real life benchmark dataset. The Wisconsin Breast Cancer dataset was initially created to conduct experiments that were to prove the usefulness of automation of find needle aspiration cytological diagnosis. It contains 699 instances of cytological analysis of fine needle aspiration from breast tumors, each contains 11 attributes.

Our next experimental dataset is the Pima Indians Diabetes dataset used to test different kind of symptoms of the diabetes. It contains 768 instances of cytological analysis of fine needle aspiration from diabetes, each contains 10 attributes.

Our next experimental dataset is Hungarian Institution of cardiology, Budapest Cardio dataset used to test different kind of symptoms of the Cardio attack. It contains 920 instances of cytological analysis of fine needle aspiration from Cardio attack, which specify the condition of a Cardio. Each contains 76 attributes. But for our experiment we have a reduced dataset contains 920 instances having 37 attribute of each.

3.2 Cancer Data Set Attribute

The Wisconsin Breast Cancer dataset consists of 11 attributes. In our research we have ignored case ID column and separated the 9 attributes (inputs) and to decide whether its benign or malignant 2 output variables are being used. And to get a better result the column range were set to (0-1).

Table 3.1: Cancer Data Set Attributes

#	Attributes	Range
Input 1	Clump Thickness	0-1
Input 2	Uniformity Of Cell Size	0-1
Input 3	Uniformity Of Cell Shape	0-1
Input 4	Marginal Adhesion	0-1
Input 5	Single Epithetical Cell Size	0-1
Input 6	Bare Nuclei	0-1
Input 7	Bland Chromatin	0-1
Input 8	Normal Nucleoli	0-1
Input 9	Mitoses	0-1
Output 1	Bening Or Maligant	0 or 1
Output 2	Bening Or Maligant	0 or 1

We have divided the whole Cancer dataset into training and test dataset. For training we have used 525 for training, 174 for testing and 0 for validation.

3.3 Diabetes Data Set Attributes

The Pima Indians Diabetes dataset consist of 10 attributes, among which 8 attribute in the input parameters and two attributes specify the output parameters. The Column range set to (0-1).

Table 3.2: Diabetes Data Set Attributes

#	Attributes	Range
Input 1	Number of times pregnant	0-1
Input 2	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	0-1
Input 3	Diastolic blood pressure (mm Hg)	0-1
Input 4	Triceps skin fold thickness(mm)	0-1
Input 5	2-Hour serum insulin (mu U/ml)	0-1
Input 6	Body mass index (weight in kg/(height in m)^2)	0-1
Input 7	Diabetes pedigree function	0-1
Input 8	Age(years)	0-1
Output 1	Class Variable	0 or 1
Output 2	Class Variable	0 or 1

We have divided the whole Diabetes dataset into training and test dataset. For training we have used 576 for training, 192 for testing and 0 for validation.

3.4 Cardio Data Set Attributes

The Hungarian Institution of cardiology, Budapest Cardio dataset consists of 76 Attributes, by applying PCA its dimension was reduced. We have used the reduced data set consists of 37 attributes among which 35 attributes are the input parameters and 2 attributes are the output parameters.

Table 3.3: Cardio Data Set Attributes

#	Attributes	Range
Input 1	Age in years	0-1
Input 2	Sex	0-1
Input 3	Chest pain location	0-1
Input 4	Relieved after rest	0-1
Input 5	Chest Pain Type	0-1
Input 6	Resting blood pressure	0-1
Input 7	Serum cholesterol	0-1
Input 8	Number of years as a smoker	0-1
Input 9	Fasting blood sugar>120 mg/dl	0-1
Input 10	History of diabetes	0-1
Input 11	Family History of coronary artery disease	0-1
Input 12	Resting electrocardiographic results	0-1
Input 13	Month of exercise ECG reading	0-1
Input 14	Day of exercise ECG reading	0-1
Input 15	Year of exercise ECG reading	0-1
Input 16	Digitalis used funning exercise ECG	0-1
Input 17	Beta blocker used during exercise ECG	0-1
Input 18	Nitrates used during exercise ECG	0-1
Input 19	Calcium channel blocker used during exercise ECG	0-1
Input 20	Diuretic used during exercise ECG	0-1
Input 21	Peak exercise blood pressure	0-1
Input 22	Resting blood pressure	0-1
Input 23	Exercise induced angina	0-1
Input 24	ST depression induced by exercise relative to rest	0-1
Input 25	The slope of the peak exercise ST segment	0-1

Input 26	Diagnosis of heart disease	0-1
Input 27	Month of cardiac Cath	0-1
Input 28	Day of cardiac Cath	0-1
Input 29	Year of cardiac Cath	0-1
Input 30	lmt	0-1
Input 31	ladprox	0-1
Input 32	laddist	0-1
Input 33	Diag	0-1
Input 34	cxmain	0-1
Input 35	ramus	0-1
Output 1	Class Variable	0 or 1
Output 2	Class Variable	0 or 1

3.5 Network Selection

We have fixed hidden layer which is always 8 in the case of all three sets of data. So for Cancer data our network is 9-8-2, for the Diabetes data our network is 8-8-2 and for Cardio data our network is 35-8-2. So we have used three different networks for three dataset.

3.5.1 Initial Weight and Bias

In our test we used the default weight allocation technique and have selected default bias and then saved the network for further studies. Thus each time we load the network it has all the fixed weight and bias which is our requirement. If the weight and bias for a network is fixed then each time the program runs, it will give the same result as it gives at earlier runs.

3.5.2 Train the Network

After creating the initial networks we trained the networks with the training datasets. After training we saved all the final networks for further use. For all three datasets we have repeated the process.

3.5.3 Test the Network

As mentioned earlier we have three types of test data set, we have used them to test their corresponding network and analyze the result.

3.5.4 Calculate MAD, MSE and RMSE

Mean Absolute Deviation, Mean Square Error and Root Mean Square Error is the final outcome of our experiment by which we can compare how many cluster is suited for our individual datasets and find the optimal number of clusters.

We establish an Algorithm to calculate MAD (Mean Absolute Deviation), MSE (Mean Square Error) and RMSE (Root Mean Square Error). The Algorithm is given below:

Algorithm 3: Steps of MAD, MSE and RMSE calculation:

1) Initialize sum = 0, sum1 =0;
2) Initialize n=LENGTH (test_set);
3) For I = 1 to n do
4) sum=sum+(ABSOLUTE (actual_output(i)-forecast(i)/
   ABSOLUTE (actual_output))));
5) sum=sum /n;
6) MAD = sum;
7) For J =1 to n do
8) sum1=sum1 + SQUARE((ABSOLUTE(actual_output(i)-forecasr(i)/
   ABSOLUTE (actual_output))));
9) sum1=sum1/n;
10) MSE= sum1;
11) RMSE= ROOT(MSE);

In this way by running the program we can compare the error and can minimize the number of error by clustering our data set and select the result which will minimize minimum error.

3.6 Cluster Selection

For our training dataset we have clustered the data into several groups. For our clustering we have chosen the k-means clustering technique. Before the clustering we have decided that we will divide the training datasets into several cluster and calculate the errors and select the cluster with the minimum error. We have chosen the centroid based built in k-means function of Mat lab to determine the clustered data. The test set data is also divided into several cluster. To determine which test data belongs to which cluster we have used the following algorithm.

Algorithm 4: Steps by which we can get which test data belongs to which cluster:

1) For i =1 to LENGTH(test_set) do

2) For j = 1 to Number_OF_CLUSTER

3) Distance(i,j) = NORM(test_set(i,:) – centroid(j,:));

4) For i = 1 to LENGTH(test_set) do

5) MIN_VALUE = INFINITY

6) For j = 1 NUMBER_OF_CLUSTER

7) If MIN_VALUE > diatance(i,j);

8) TEST_SET_CLUSTER(i) = j;

3.6.1 Train the Network Using Clustered Data

The loaded network is trained separately by the clustered test set which are divided based on the cluster in the previous step. The training technique is same as the previous one.

3.6.2 Test the Network Using Clustered Data

Just like training of the clustered data set, we have tested each individual datasets and simulated separately for the forecasted value which is then compared with the corresponding record of the actual output to compute the result.

3.6.3 Calculate MAD, MSE and RMSE

Just like the previous MAD, MSE and RMSE calculation the error is calculated for the k-means clustering technique and record it into the excel sheet.

3.7 Hybrid Technique

Just like section 3.6 this technique is operated just the difference is: instead of loading the cloned network which has only the prior fixation of weight and bias, here we loaded the hybrid network which has not only have the weight and bias but also other attributes which is fixed after the training of the normal cloned network. After that clustered train sets is trained individually into the network and then the three test set of cluster is passed through the network for testing and the forecasted data are taken into account for the calculation of MAD, MSE and RMSE.

The steps of proposed model is given below:

Algorithm 5: Steps of proposed model

1) Create a network of 8 hidden layer with fixed bias, weight and no validation.

2) Select and preprocess the working data

3) For i = 1 to 300 do

4) Train the network with the training dataset.

5) Test the network with the test dataset.

6) Calculate the MAD, MSE and RMSE.

7) End-for

8) Divide the training data into N number of clusters.

9) Select which test data belongs to with cluster.

10) For i = 1 to N do

11) For j = 1 to 300 do

12) Train the network with clustered dataset

13) Test the network with clustered test set

14) Calculate MAD, MSE and RMSE

15) End-for

16) End-for

17) Compare Before and after cluster error result.

18) Repeat the whole process for all the dataset and compare result to find the optimal cluster with the minimum error.

Here is a Flowchart for the given algorithm,

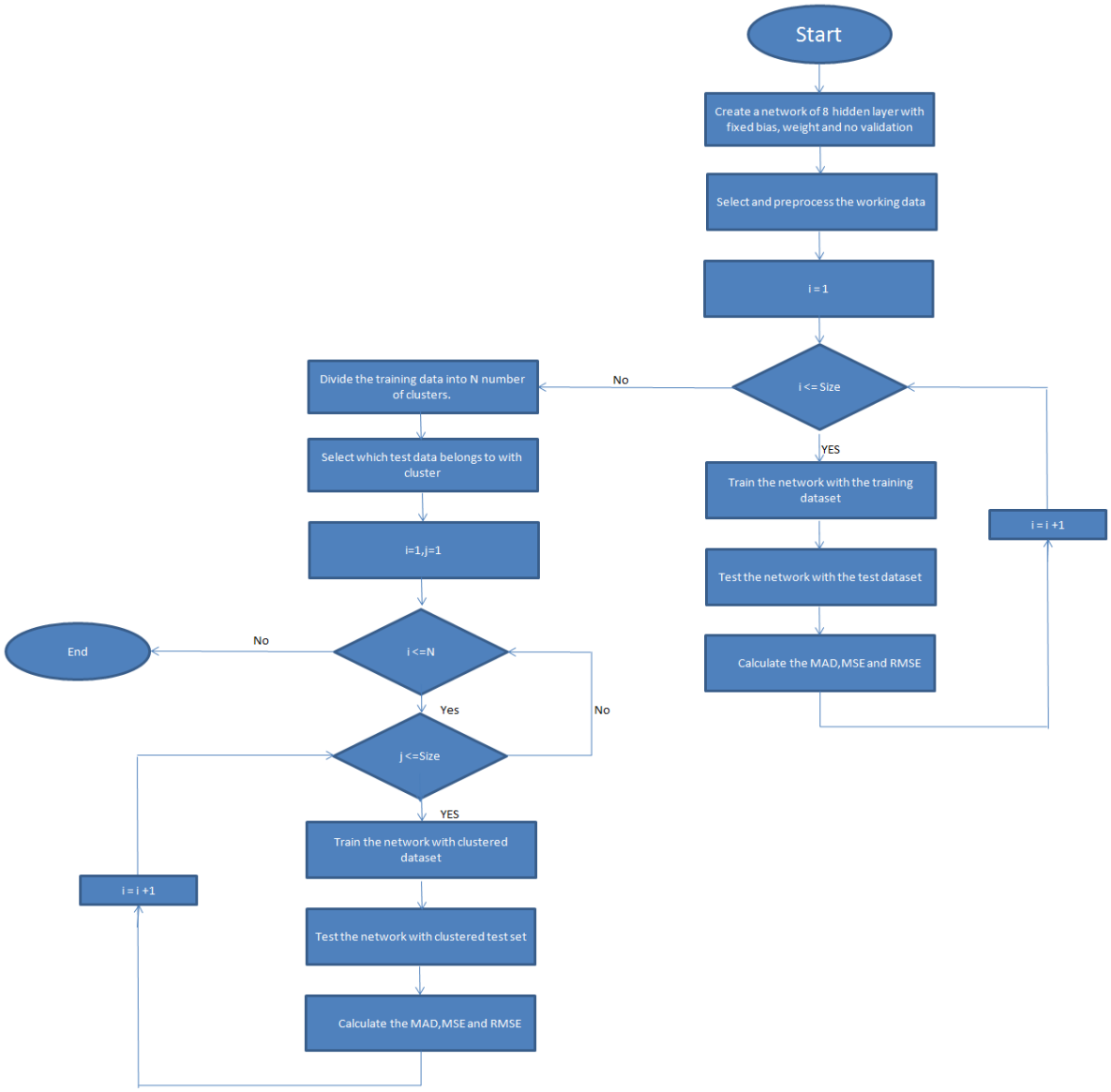


Figure 3.1: Flowchart for the hybrid model

3.8 Observation

We will be applying all this on 3 dataset and see how well our algorithm performs and if our hybrid model is good enough for usage.

Chapter 4
Implementation

4.1 Sample Data Set

Cancer data set which contains 699 of data. Few sample data are given below:

Table 4.1.1: Sample Data Set

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.2	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.2	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	1	0
0.5	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.5	0.4	0.6	0.8	0.4	0.1	0.8	1	0.1	0	1
0.5	0.3	0.3	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.2	0.3	0.1	0.1	0.3	0.1	0.1	0.1	0.1	1	0
0.3	0.5	0.7	0.8	0.8	0.9	0.7	1	0.7	0	1
1	0.5	0.6	1	0.6	1	0.7	0.7	1	0	1
0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.1	1	0
0.1	0.4	0.3	1	0.4	1	0.5	0.6	0.1	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.2	0.1	0.1	0.2	0.3	0.1	0.2	0.1	0.1	1	0
0.4	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.6	0.3	0.2	0.1	0.3	0.4	0.4	0.1	0.1	0	1
0.5	0.1	0.2	0.1	0.2	0.1	0.1	0.1	0.1	1	0
0.7	0.5	0.6	1	0.5	1	0.7	0.9	0.4	0	1
0.6	1	1	1	0.8	1	0.7	1	0.7	0	1
0.5	0.7	1	1	0.5	1	1	1	0.1	0	1
0.1	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0

4.2 Result of Without Clustering (Cancer data Set)

we have selected 60% (525) data for training the Neural Network.

Table 4.2.1: Without Clustering Training Data

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.2	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.2	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	1	0
0.5	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.5	0.4	0.6	0.8	0.4	0.1	0.8	1	0.1	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.2	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	1	0
0.5	0.8	0.9	0.4	0.3	1	0.7	0.1	0.1	0	1
0.1	0.1	0.1	0.3	0.2	0.3	0.1	0.1	0.1	1	0
0.4	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.1	1	0

Table 4.2.2: Without Clustering Test Data (targeted output)

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.4	0.2	0.1	0.1	0.2	0.1	0.1	0.1	0.1	1	0
0.5	0.4	0.3	0.1	0.2	0.35	0.2	0.3	0.1	1	0
0.4	0.1	0.1	0.1	0.2	0.3	0.2	0.1	0.1	1	0
0.1	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. .	. . .	. . .
0.7	0.5	0.6	1	0.5	1	0.7	0.9	0.4	0	1
0.6	1	1	1	0.8	1	0.7	1	0.7	0	1
0.5	0.7	1	1	0.5	1	1	1	0.1	0	1
0.1	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0

Table 4.2.3: Without Clustering Test Data (forecasted Output)

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.4	0.2	0.1	0.1	0.2	0.1	0.1	0.1	0.1	0.766016	0.232902
0.5	0.4	0.3	0.1	0.2	0.35	0.2	0.3	0.1	0.583734	0.415503
0.4	0.1	0.1	0.1	0.2	0.3	0.2	0.1	0.1	0.715676	0.283339
0.1	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	0.810972	0.18785
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.7	0.5	0.6	1	0.5	1	0.7	0.9	0.4	0.044263	0.955793
0.6	1	1	1	0.8	1	0.7	1	0.7	-0.05928	1.05947
0.5	0.7	1	1	0.5	1	1	1	0.1	-0.05465	1.054882
0.1	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	0.823182	0.175618

4.3 Result of n-Clustering for Cancer Data Set

We have created n number of cluster for 525 data using K-means clustering algorithm. Also 174 test data clustered accordingly. They are sorted below.

4.3.1 Two Cluster System for Cancer Data Set

Cluster1: 348 training data and 114 test data

Cluster2: 177 training data and 60 test data

4.3.1.1 Cluster 1 test data results

Table 4.3.1.1(a): Cluster 1 test data (targeted output) of 2- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.4	0.2	0.1	0.1	0.2	0.1	0.1	0.1	0.1	1	0
0.5	0.4	0.3	0.1	0.2	0.35	0.2	0.3	0.1	1	0
0.4	0.1	0.1	0.1	0.2	0.3	0.2	0.1	0.1	1	0
0.1	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.4	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.6	0.3	0.2	0.1	0.3	0.4	0.4	0.1	0.1	0	1
0.5	0.1	0.2	0.1	0.2	0.1	0.1	0.1	0.1	1	0
0.1	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0

Table 4.3.1.1(b): Cluster 1 test data (forecasted output) of 2- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.4	0.2	0.1	0.1	0.2	0.1	0.1	0.1	0.1	0.893265	0.076081
0.5	0.4	0.3	0.1	0.2	0.35	0.2	0.3	0.1	0.758016	0.213589
0.4	0.1	0.1	0.1	0.2	0.3	0.2	0.1	0.1	0.890951	0.07871
0.1	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	0.912483	0.056343
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.4	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	0.90215	0.066962
0.6	0.3	0.2	0.1	0.3	0.4	0.4	0.1	0.1	0.777465	0.194173
0.5	0.1	0.2	0.1	0.2	0.1	0.1	0.1	0.1	0.891841	0.077506
0.1	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	0.920856	0.047623

4.3.1.2 Cluster 2 test data results

Table 4.3.1.2(a): Cluster 2 test data (targeted output) of 2- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
1	0.6	0.3	0.6	0.4	1	0.7	0.8	0.4	0	1
0.5	1	1	0.6	1	1	1	0.6	0.5	0	1
0.5	1	1	1	1	1	1	0.1	0.1	0	1
0.7	0.6	0.4	0.8	1	1	0.9	0.5	0.3	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.1	0.4	0.3	1	0.4	1	0.5	0.6	0.1	0	1
0.7	0.5	0.6	1	0.5	1	0.7	0.9	0.4	0	1
0.6	1	1	1	0.8	1	0.7	1	0.7	0	1
0.5	0.7	1	1	0.5	1	1	1	0.1	0	1

Table 4.3.1.2(b): Cluster 2 test data (forecasted output) of 2- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
1	0.6	0.3	0.6	0.4	1	0.7	0.8	0.4	0.142348	0.926301
0.5	1	1	0.6	1	1	1	0.6	0.5	0.103193	0.874617
0.5	1	1	1	1	1	1	0.1	0.1	0.122127	0.858023
0.7	0.6	0.4	0.8	1	1	0.9	0.5	0.3	0.162047	0.839952
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.1	0.4	0.3	1	0.4	1	0.5	0.6	0.1	0.250789	0.764253
0.7	0.5	0.6	1	0.5	1	0.7	0.9	0.4	0.142864	0.851766
0.6	1	1	1	0.8	1	0.7	1	0.7	0.063503	0.863136
0.5	0.7	1	1	0.5	1	1	1	0.1	0.115832	0.890159

4.3.2 Three Cluster System for Cancer Data Set

Cluster1:210 training data and 59 test data

Cluster2:169 training data and 57 test data

Cluster3:146 training data and 58 test data

4.3.2.1 Cluster 1 test data Results

Table 4.3.2.1(a): Cluster 1 test data (targeted output) of 3- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.1	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	1	0
0.1	0.1	0.3	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	0.7	1	0
0.3	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.1	0.1	0.2	0.1	0.2	0.2	0.4	0.2	0.1	1	0
0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.1	1	0
0.2	0.1	0.1	0.2	0.3	0.1	0.2	0.1	0.1	1	0
0.1	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0

Table 4.3.2.1(b): Cluster 1 test data (forecasted output) of 3- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.1	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	1	0
0.1	0.1	0.3	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	0.7	1	0
0.3	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.1	0.1	0.2	0.1	0.2	0.2	0.4	0.2	0.1	1	0
0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.1	1	0
0.2	0.1	0.1	0.2	0.3	0.1	0.2	0.1	0.1	1	0
0.1	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0

4.3.2.2 Cluster 2 test data Results

Table 4.3.2.2(a): Cluster 2 test data (targeted output) of 3- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
1	0.6	0.3	0.6	0.4	1	0.7	0.8	0.4	0	1
0.5	1	1	0.6	1	1	1	0.6	0.5	0	1
0.5	1	1	1	1	1	1	0.1	0.1	0	1
0.7	0.6	0.4	0.8	1	1	0.9	0.5	0.3	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.1	0.4	0.3	1	0.4	1	0.5	0.6	0.1	0	1
0.7	0.5	0.6	1	0.5	1	0.7	0.9	0.4	0	1
0.6	1	1	1	0.8	1	0.7	1	0.7	0	1
0.5	0.7	1	1	0.5	1	1	1	0.1	0	1

Table 4.3.2.2(b): Cluster 2 test data (forecasted output) of 3- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
1	0.6	0.3	0.6	0.4	1	0.7	0.8	0.4	0.076018	0.907757
0.5	1	1	0.6	1	1	1	0.6	0.5	0.065139	0.918036
0.5	1	1	1	1	1	1	0.1	0.1	0.06781	0.915513
0.7	0.6	0.4	0.8	1	1	0.9	0.5	0.3	0.100882	0.884248
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.1	0.4	0.3	1	0.4	1	0.5	0.6	0.1	0.189181	0.800758
0.7	0.5	0.6	1	0.5	1	0.7	0.9	0.4	0.090894	0.893692
0.6	1	1	1	0.8	1	0.7	1	0.7	0.062687	0.920354
0.5	0.7	1	1	0.5	1	1	1	0.1	0.055966	0.926711

4.3.2.3 Cluster 3 test data Results

Table 4.3.2.3(a): Cluster 3 test data (targeted output) of 3- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.4	0.2	0.1	0.1	0.2	0.1	0.1	0.1	0.1	1	0
0.5	0.4	0.3	0.1	0.2	0.35	0.2	0.3	0.1	1	0
0.4	0.1	0.1	0.1	0.2	0.3	0.2	0.1	0.1	1	0
0.5	0.2	0.2	0.2	0.1	0.1	0.2	0.1	0.1	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.6	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.4	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.6	0.3	0.2	0.1	0.3	0.4	0.4	0.1	0.1	0	1
0.5	0.1	0.2	0.1	0.2	0.1	0.1	0.1	0.1	1	0

Table 4.3.2.3(b): Cluster 3 test data (forecasted output) of 3- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.4	0.2	0.1	0.1	0.2	0.1	0.1	0.1	0.1	0.993945	0.005989
0.5	0.4	0.3	0.1	0.2	0.35	0.2	0.3	0.1	0.705208	0.29472
0.4	0.1	0.1	0.1	0.2	0.3	0.2	0.1	0.1	0.952838	0.047096
0.5	0.2	0.2	0.2	0.1	0.1	0.2	0.1	0.1	0.915005	0.084929
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.6	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	0.931495	0.068438
0.4	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	0.995695	0.004239
0.6	0.3	0.2	0.1	0.3	0.4	0.4	0.1	0.1	0.658305	0.34162
0.5	0.1	0.2	0.1	0.2	0.1	0.1	0.1	0.1	0.965924	0.03401

4.3.3 Four Cluster System for Cancer Data Set

Cluster1:341 training data and 112 test data

Cluster2:64 training data and 15 test data

Cluster3:70 training data and 24 test data

Cluster4:50 training data and 23 test data

4.3.3.1 Cluster 1 test data Results

Table 4.3.3.1(a): Cluster 1 test data (targeted output) of 4- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.4	0.2	0.1	0.1	0.2	0.1	0.1	0.1	0.1	1	0
0.5	0.4	0.3	0.1	0.2	0.35	0.2	0.3	0.1	1	0
0.4	0.1	0.1	0.1	0.2	0.3	0.2	0.1	0.1	1	0
0.1	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.4	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0
0.6	0.3	0.2	0.1	0.3	0.4	0.4	0.1	0.1	0	1
0.5	0.1	0.2	0.1	0.2	0.1	0.1	0.1	0.1	1	0
0.1	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	1	0

Table 4.3.3.1(b): Cluster 1 test data (forecasted output) of 4- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.4	0.2	0.1	0.1	0.2	0.1	0.1	0.1	0.1	0.873476	0.104926
0.5	0.4	0.3	0.1	0.2	0.35	0.2	0.3	0.1	0.73751	0.246936
0.4	0.1	0.1	0.1	0.2	0.3	0.2	0.1	0.1	0.875431	0.102908
0.1	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.1	0.892625	0.084913
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.4	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	0.886176	0.091647
0.6	0.3	0.2	0.1	0.3	0.4	0.4	0.1	0.1	0.777186	0.20559
0.5	0.1	0.2	0.1	0.2	0.1	0.1	0.1	0.1	0.869362	0.109231
0.1	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.1	0.898315	0.078941

4.3.3.2 Cluster 2 test data Results

Table 4.3.3.2(a): Cluster 2 test data (targeted output) of 4- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
1	0.6	0.3	0.6	0.4	1	0.7	0.8	0.4	0	1
0.9	1	1	0.1	1	0.8	0.3	0.3	0.1	0	1
1	0.8	0.8	0.2	0.3	0.4	0.8	0.7	0.8	0	1
1	1	1	0.8	0.6	0.1	0.8	0.9	0.1	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.8	0.7	0.8	0.2	0.4	0.2	0.5	1	0.1	0	1
0.4	0.6	0.6	0.5	0.7	0.6	0.7	0.7	0.3	0	1
0.8	0.7	0.8	0.5	0.5	1	0.9	1	0.1	0	1
0.9	0.7	0.7	0.5	0.5	1	0.7	0.8	0.3	0	1

Table 4.3.3.2(b): Cluster 2 test data (forecasted output) of 4- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
1	0.6	0.3	0.6	0.4	1	0.7	0.8	0.4	0.149331	0.850655
0.9	1	1	0.1	1	0.8	0.3	0.3	0.1	0.162386	0.8376
1	0.8	0.8	0.2	0.3	0.4	0.8	0.7	0.8	0.045185	0.954799
1	1	1	0.8	0.6	0.1	0.8	0.9	0.1	0.079961	0.920024
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.8	0.7	0.8	0.2	0.4	0.2	0.5	1	0.1	0.12005	0.879935
0.4	0.6	0.6	0.5	0.7	0.6	0.7	0.7	0.3	0.29483	0.705157
0.8	0.7	0.8	0.5	0.5	1	0.9	1	0.1	0.011858	0.988126
0.9	0.7	0.7	0.5	0.5	1	0.7	0.8	0.3	0.078574	0.92141

4.3.3.3 Cluster 3 test data Results

Table 4.3.3.3(a): Cluster 3 test data (targeted output) of 4- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
1	0.2	0.2	0.1	0.2	0.6	0.1	0.1	0.2	0	1
0.5	0.2	0.3	0.4	0.2	0.7	0.3	0.6	0.1	0	1
0.6	0.5	0.5	0.8	0.4	1	0.3	0.4	0.1	0	1
1	0.4	0.4	0.6	0.2	1	0.2	0.3	0.1	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
1	0.4	0.3	1	0.4	1	1	0.1	0.1	0	1
0.5	0.4	0.6	1	0.2	1	0.4	0.1	0.1	0	1
0.9	0.5	0.8	0.1	0.2	0.3	0.2	0.1	0.5	0	1
0.1	0.4	0.3	1	0.4	1	0.5	0.6	0.1	0	1

Table 4.3.3.3(b): Cluster 3 test data (forecasted output) of 4- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
1	0.2	0.2	0.1	0.2	0.6	0.1	0.1	0.2	0.17455	0.82545
0.5	0.2	0.3	0.4	0.2	0.7	0.3	0.6	0.1	0.234516	0.765484
0.6	0.5	0.5	0.8	0.4	1	0.3	0.4	0.1	0.204061	0.795939
1	0.4	0.4	0.6	0.2	1	0.2	0.3	0.1	0.120114	0.879886
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
1	0.4	0.3	1	0.4	1	1	0.1	0.1	-0.02396	1.023963
0.5	0.4	0.6	1	0.2	1	0.4	0.1	0.1	0.177356	0.822644
0.9	0.5	0.8	0.1	0.2	0.3	0.2	0.1	0.5	0.228255	0.771745
0.1	0.4	0.3	1	0.4	1	0.5	0.6	0.1	0.266649	0.733351

4.3.3.4 Cluster 4 test data Results

Table 4.3.3.4(a): Cluster 4 test data (targeted output) of 4- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.5	1	1	0.6	1	1	1	0.6	0.5	0	1
0.5	1	1	1	1	1	1	0.1	0.1	0	1
0.7	0.6	0.4	0.8	1	1	0.9	0.5	0.3	0	1
0.9	0.9	1	0.3	0.6	1	0.7	1	0.6	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.3	1	0.3	1	0.6	1	0.5	0.1	0.4	0	1
0.7	0.5	0.6	1	0.5	1	0.7	0.9	0.4	0	1
0.6	1	1	1	0.8	1	0.7	1	0.7	0	1
0.5	0.7	1	1	0.5	1	1	1	0.1	0	1

Table 4.3.3.4(b): Cluster 4 test data (forecasted output) of 4- Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Output 1	Output 2
0.5	1	1	0.6	1	1	1	0.6	0.5	0	1
0.5	1	1	1	1	1	1	0.1	0.1	0	1
0.7	0.6	0.4	0.8	1	1	0.9	0.5	0.3	0	1
0.9	0.9	1	0.3	0.6	1	0.7	1	0.6	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.3	1	0.3	1	0.6	1	0.5	0.1	0.4	0	1
0.7	0.5	0.6	1	0.5	1	0.7	0.9	0.4	0	1
0.6	1	1	1	0.8	1	0.7	1	0.7	0	1
0.5	0.7	1	1	0.5	1	1	1	0.1	0	1

4.4 Sample Data Set for Diabetes

Diabetes data set which contains 768 data. Few sample data are given below:

Table 4.4.1: Sample Data Set

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.352941	0.83	0.606557	0	0	0.396423	0.096499	0.75	0	1
0.352941	0.77	0.606557	0.32	0.228132	0.436662	0.324936	0.3	0	1
0.117647	0.46	0.622951	0.2	0	0.360656	0.691716	0.116667	0	1
0.117647	0.555	0.491803	0	0	0.390462	0.113151	0.033333	0	1
0.470588	0.325	0.590164	0.23	0	0.4769	0.222886	0.35	0	1
0.058824	0.53	0.57377	0.28	0.159574	0.509687	0.027327	0.016667	0	1
0	0.49	0.672131	0.15	0.099291	0.375559	0.094364	0.016667	0	1
0.235294	0.475	0.57377	0.32	0	0.47839	0.22801	0.05	0	1
0.411765	0.405	0.639344	0.4	0.056738	0.695976	0.078138	0.35	0	1
0.176471	0.565	0.360656	0.13	0	0.33383	0.026473	0.016667	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.117647	0.985	0.57377	0.45	0.641844	0.454545	0.034159	0.533333	1	0
0.058824	0.425	0.540984	0.29	0	0.396423	0.116567	0.166667	0	1
0.352941	0.67	0.57377	0.23	0.153664	0.527571	0.198121	0.133333	1	0
0.176471	0.955	0.557377	0.15	0.153664	0.460507	0.094364	0.216667	0	1
0	0.805	0.409836	0	0	0.326379	0.075149	0.733333	0	1
0.529412	0.61	0.459016	0	0	0.496274	0.442357	0.2	1	0
0.058824	0.755	0.491803	0	0	0.388972	0.043126	0.016667	0	1
0.352941	0.435	0.655738	0	0	0.345753	0.002562	0.183333	0	1

4.5 Result of without Clustering for Diabetes Data Set

Table 4.5.1: Without clustering training data

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.352941	0.83	0.606557	0	0	0.396423	0.096499	0.75	0	1
0.352941	0.77	0.606557	0.32	0.228132	0.436662	0.324936	0.3	0	1
0.117647	0.46	0.622951	0.2	0	0.360656	0.691716	0.116667	0	1
0.117647	0.555	0.491803	0	0	0.390462	0.113151	0.0333333	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.588235	0.34	0.868852	0.23	0.05792	0.529061	0.088386	0.433333	0	1
0	0.695	0.508197	0.17	0.248227	0.329359	0.055081	0	0	1
0.294118	0.495	0.442623	0.28	0.098109	0.506706	0.179761	0.15	0	1
0	0.595	0	0	0	0.482861	0.0269	0.05	1	0

Table 4.5.2: Without clustering test data (targeted output)

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.117647	0.54	0.508197	0.1	0.328605	0.377049	0.342869	0.016667	0	1
0.470588	0.54	0.57377	0	0	0.454545	0.374466	0.2	1	0
0.058824	0.595	0.442623	0.13	0.059102	0.33234	0.054227	0.05	0	1
0.352941	0.515	0.590164	0.32	0.224586	0.561848	0.105038	0.566667	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0	0.805	0.409836	0	0	0.326379	0.075149	0.733333	0	1
0.529412	0.61	0.459016	0	0	0.496274	0.442357	0.2	1	0
0.058824	0.755	0.491803	0	0	0.388972	0.043126	0.016667	0	1
0.352941	0.435	0.655738	0	0	0.345753	0.002562	0.183333	0	1

Table 4.5.3: Without clustering test data (forecasted output)

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.117647	0.54	0.508197	0.1	0.328605	0.377049	0.342869	0.016667	0.4828	0.601417
0.470588	0.54	0.57377	0	0	0.454545	0.374466	0.2	0.508926	0.560248
0.058824	0.595	0.442623	0.13	0.059102	0.33234	0.054227	0.05	0.466478	0.63561
0.352941	0.515	0.590164	0.32	0.224586	0.561848	0.105038	0.566667	0.536121	0.526895
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0	0.805	0.409836	0	0	0.326379	0.075149	0.733333	0.521995	0.574444
0.529412	0.61	0.459016	0	0	0.496274	0.442357	0.2	0.52261	0.542154
0.058824	0.755	0.491803	0	0	0.388972	0.043126	0.016667	0.485579	0.631055
0.352941	0.435	0.655738	0	0	0.345753	0.002562	0.183333	0.474264	0.611711

4.6 Result of n-Clustering for Diabetes Data Set

We have created n number of cluster for 576 data using K-means clustering algorithm. Also 192 test data clustered accordingly. They are sorted below.

4.6.1 Two Cluster System for Diabetes Data Set

Cluster1: 192 training data and 63 test data

Cluster2: 348 training data and 192 test data

4.6.1.1 Cluster 1 test data results

Table 4.6.1.1(a): Cluster 1 test data (targeted output) of 2-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.470588	0.54	0.57377	0	0	0.454545	0.374466	0.2	1	0
0.352941	0.515	0.590164	0.32	0.224586	0.561848	0.105038	0.566667	0	1
0.294118	0.52	0.606557	0	0	0.42921	0.032024	0.45	0	1
0.411765	0.975	0.57377	0.33	0.171395	0.374069	0.036294	0.566667	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.235294	0.625	0.57377	0.18	0.144208	0.4307	0.455167	0.4	1	0
0.117647	0.985	0.57377	0.45	0.641844	0.454545	0.034159	0.533333	1	0
0	0.805	0.409836	0	0	0.326379	0.075149	0.733333	0	1
0.529412	0.61	0.459016	0	0	0.496274	0.442357	0.2	1	0

Table 4.6.1.1(b): Cluster 1 test data (forecasted output) of 2-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.470588	0.54	0.57377	0	0	0.454545	0.374466	0.2	0.522092	0.472794
0.352941	0.515	0.590164	0.32	0.224586	0.561848	0.105038	0.566667	0.442848	0.551804
0.294118	0.52	0.606557	0	0	0.42921	0.032024	0.45	0.381848	0.612672
0.411765	0.975	0.57377	0.33	0.171395	0.374069	0.036294	0.566667	0.655785	0.339437
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.235294	0.625	0.57377	0.18	0.144208	0.4307	0.455167	0.4	0.558227	0.436711
0.117647	0.985	0.57377	0.45	0.641844	0.454545	0.034159	0.533333	0.698208	0.297084
0	0.805	0.409836	0	0	0.326379	0.075149	0.733333	0.479628	0.515122
0.529412	0.61	0.459016	0	0	0.496274	0.442357	0.2	0.596173	0.398902

4.6.1.2 Cluster 2 test data results

Table 4.6.1.2(a): Cluster 2 test data (targeted output) of 2-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.117647	0.54	0.508197	0.1	0.328605	0.377049	0.342869	0.016667	0	1
0.058824	0.595	0.442623	0.13	0.059102	0.33234	0.054227	0.05	0	1
0.058824	0.58	0.639344	0.29	0.212766	0.538003	0.17848	0.066667	0	1
0.058824	0.9	0	0	0	0.645306	0.087105	0.333333	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.352941	0.67	0.57377	0.23	0.153664	0.527571	0.198121	0.133333	1	0
0.176471	0.955	0.557377	0.15	0.153664	0.460507	0.094364	0.216667	0	1
0.058824	0.755	0.491803	0	0	0.388972	0.043126	0.016667	0	1
0.352941	0.435	0.655738	0	0	0.345753	0.002562	0.183333	0	1

Table 4.6.1.2(b): Cluster 2 test data (forecasted output) of 2-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.117647	0.54	0.508197	0.1	0.328605	0.377049	0.342869	0.016667	0.295344	0.711133
0.058824	0.595	0.442623	0.13	0.059102	0.33234	0.054227	0.05	0.29214	0.714335
0.058824	0.58	0.639344	0.29	0.212766	0.538003	0.17848	0.066667	0.348541	0.657274
0.058824	0.9	0	0	0	0.645306	0.087105	0.333333	0.71963	0.281659
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.352941	0.67	0.57377	0.23	0.153664	0.527571	0.198121	0.133333	0.395829	0.609446
0.176471	0.955	0.557377	0.15	0.153664	0.460507	0.094364	0.216667	0.573094	0.429987
0.058824	0.755	0.491803	0	0	0.388972	0.043126	0.016667	0.371629	0.633844
0.352941	0.435	0.655738	0	0	0.345753	0.002562	0.183333	0.223086	0.784284

4.6.2 Three Cluster System for Diabetes Data Set

Cluster 1:337 training data and 114 test data

Cluster 2:115 training data and 39 test data

Cluster 3:124 training data and 39 test data

4.6.2.1 Cluster 1 test data results

Table 4.6.2.1(a): Cluster 1 test data (targeted output) of 3-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.117647	0.54	0.508197	0.1	0.328605	0.377049	0.342869	0.016667	0	1
0.058824	0.595	0.442623	0.13	0.059102	0.33234	0.054227	0.05	0	1
0.058824	0.58	0.639344	0.29	0.212766	0.538003	0.17848	0.066667	0	1
0.294118	0.43	0.557377	0.28	0.083924	0.450075	0.122118	0.05	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.176471	0.415	0.47541	0.31	0.021277	0.511177	0.110162	0.066667	0	1
0.058824	0.425	0.540984	0.29	0	0.396423	0.116567	0.166667	0	1
0.176471	0.955	0.557377	0.15	0.153664	0.460507	0.094364	0.216667	0	1
0.058824	0.755	0.491803	0	0	0.388972	0.043126	0.016667	0	1

Table 4.6.2.1(b): Cluster 1 test data (forecasted output) of 3-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.117647	0.54	0.508197	0.1	0.328605	0.377049	0.342869	0.016667	0.254558	0.736636
0.058824	0.595	0.442623	0.13	0.059102	0.33234	0.054227	0.05	0.231354	0.759006
0.058824	0.58	0.639344	0.29	0.212766	0.538003	0.17848	0.066667	0.291033	0.701724
0.294118	0.43	0.557377	0.28	0.083924	0.450075	0.122118	0.05	0.186242	0.802519
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.176471	0.415	0.47541	0.31	0.021277	0.511177	0.110162	0.066667	0.212641	0.777136
0.058824	0.425	0.540984	0.29	0	0.396423	0.116567	0.166667	0.257989	0.733643
0.176471	0.955	0.557377	0.15	0.153664	0.460507	0.094364	0.216667	0.424654	0.573399
0.058824	0.755	0.491803	0	0	0.388972	0.043126	0.016667	0.252215	0.738886

4.6.2.2 Cluster 2 test data results

Table 4.6.2.2(a): Cluster 2 test data (targeted output) of 3-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.352941	0.515	0.590164	0.32	0.224586	0.561848	0.105038	0.566667	0	1
0.411765	0.975	0.57377	0.33	0.171395	0.374069	0.036294	0.566667	1	0
0.588235	0.645	0.508197	0.36	0	0.614009	0.154996	0.283333	1	0
0.352941	0.645	0.737705	0.07	0.385343	0.292101	0.215201	0.65	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.529412	0.595	0.655738	0.35	0	0.432191	0.078992	0.133333	1	0
0.235294	0.625	0.57377	0.18	0.144208	0.4307	0.455167	0.4	1	0
0.117647	0.985	0.57377	0.45	0.641844	0.454545	0.034159	0.533333	1	0
0.352941	0.67	0.57377	0.23	0.153664	0.527571	0.198121	0.133333	1	0

Table 4.6.2.2(b): Cluster 2 test data (forecasted output) of 3-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.352941	0.515	0.590164	0.32	0.224586	0.561848	0.105038	0.566667	0.645512	0.47213
0.411765	0.975	0.57377	0.33	0.171395	0.374069	0.036294	0.566667	0.671535	0.470193
0.588235	0.645	0.508197	0.36	0	0.614009	0.154996	0.283333	0.629862	0.485696
0.352941	0.645	0.737705	0.07	0.385343	0.292101	0.215201	0.65	0.665584	0.470418
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.529412	0.595	0.655738	0.35	0	0.432191	0.078992	0.133333	0.597563	0.528842
0.235294	0.625	0.57377	0.18	0.144208	0.4307	0.455167	0.4	0.65822	0.46439
0.117647	0.985	0.57377	0.45	0.641844	0.454545	0.034159	0.533333	0.634264	0.460232
0.352941	0.67	0.57377	0.23	0.153664	0.527571	0.198121	0.133333	0.626261	0.491105

4.6.2.3 Cluster 3 test data results

Table 4.6.2.3(a): Cluster 3 test data (targeted output) of 3-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.470588	0.54	0.57377	0	0	0.454545	0.374466	0.2	1	0
0.058824	0.9	0	0	0	0.645306	0.087105	0.333333	1	0
0.294118	0.52	0.606557	0	0	0.42921	0.032024	0.45	0	1
0.235294	0.6	0.557377	0	0	0.441133	0.269428	0.216667	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.176471	0.305	0.672131	0.28	0	0.512668	0.070453	0.416667	0	1
0	0.805	0.409836	0	0	0.326379	0.075149	0.733333	0	1
0.529412	0.61	0.459016	0	0	0.496274	0.442357	0.2	1	0
0.352941	0.435	0.655738	0	0	0.345753	0.002562	0.183333	0	1

Table 4.6.2.3(b): Cluster 3 test data (forecasted output) of 3-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.470588	0.54	0.57377	0	0	0.454545	0.374466	0.2	0.345952	0.64524
0.058824	0.9	0	0	0	0.645306	0.087105	0.333333	0.39478	0.545401
0.294118	0.52	0.606557	0	0	0.42921	0.032024	0.45	0.32773	0.626752
0.235294	0.6	0.557377	0	0	0.441133	0.269428	0.216667	0.354391	0.609844
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.176471	0.305	0.672131	0.28	0	0.512668	0.070453	0.416667	0.272326	0.665383
0	0.805	0.409836	0	0	0.326379	0.075149	0.733333	0.379456	0.55134
0.529412	0.61	0.459016	0	0	0.496274	0.442357	0.2	0.358388	0.639267
0.352941	0.435	0.655738	0	0	0.345753	0.002562	0.183333	0.312286	0.637979

4.6.3 Four Cluster System for Diabetes Data Set

Cluster1:138 training data and 48 test data,

Cluster2:172 training data and 59 test data,

Cluster3:238 training data 77 test data,

Cluster4:28 training data and 8 test data.

4.6.3.1 Cluster 1 test data results

Table 4.6.3.1(a): Cluster 1 test data (target output) of 4-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.058824	0.58	0.639344	0.29	0.212766	0.538003	0.17848	0.066667	0	1
0	0.59	0.688525	0.47	0.271868	0.682563	0.201964	0.166667	1	0
0.117647	0.635	0.47541	0.24	0.325059	0.412817	0.649872	0.066667	0	1
0.294118	0.545	0.508197	0.41	0.152482	0.533532	0.186166	0.066667	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0	0.585	0.540984	0.31	0.222222	0.459016	0.177199	0.016667	0	1
0.117647	0.985	0.57377	0.45	0.641844	0.454545	0.034159	0.533333	1	0
0.352941	0.67	0.57377	0.23	0.153664	0.527571	0.198121	0.133333	1	0
0.176471	0.955	0.557377	0.15	0.153664	0.460507	0.094364	0.216667	0	1

Table 4.6.3.1(b): Cluster 1 test data (forecasted output) of 4-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.058824	0.58	0.639344	0.29	0.212766	0.538003	0.17848	0.066667	0.331967	0.655258
0	0.59	0.688525	0.47	0.271868	0.682563	0.201964	0.166667	0.379354	0.606367
0.117647	0.635	0.47541	0.24	0.325059	0.412817	0.649872	0.066667	0.414445	0.568548
0.294118	0.545	0.508197	0.41	0.152482	0.533532	0.186166	0.066667	0.257122	0.733556
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0	0.585	0.540984	0.31	0.222222	0.459016	0.177199	0.016667	0.357704	0.627847
0.117647	0.985	0.57377	0.45	0.641844	0.454545	0.034159	0.533333	0.71285	0.256349
0.352941	0.67	0.57377	0.23	0.153664	0.527571	0.198121	0.133333	0.327896	0.658975
0.176471	0.955	0.557377	0.15	0.153664	0.460507	0.094364	0.216667	0.652123	0.318949

4.6.3.2 Cluster 2 test data results

Table 4.6.3.2(a): Cluster 2 test data (targeted output) of 4-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.470588	0.54	0.57377	0	0	0.454545	0.374466	0.2	1	0
0.352941	0.515	0.590164	0.32	0.224586	0.561848	0.105038	0.566667	0	1
0.294118	0.52	0.606557	0	0	0.42921	0.032024	0.45	0	1
0.411765	0.975	0.57377	0.33	0.171395	0.374069	0.036294	0.566667	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.529412	0.595	0.655738	0.35	0	0.432191	0.078992	0.133333	1	0
0.235294	0.625	0.57377	0.18	0.144208	0.4307	0.455167	0.4	1	0
0	0.805	0.409836	0	0	0.326379	0.075149	0.733333	0	1
0.529412	0.61	0.459016	0	0	0.496274	0.442357	0.2	1	0

Table 4.6.3.2(b): Cluster 2 test data (forecasted output) of 4-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.470588	0.54	0.57377	0	0	0.454545	0.374466	0.2	0.465444	0.526357
0.352941	0.515	0.590164	0.32	0.224586	0.561848	0.105038	0.566667	0.436461	0.561136
0.294118	0.52	0.606557	0	0	0.42921	0.032024	0.45	0.365367	0.62773
0.411765	0.975	0.57377	0.33	0.171395	0.374069	0.036294	0.566667	0.631511	0.366205
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.529412	0.595	0.655738	0.35	0	0.432191	0.078992	0.133333	0.452432	0.545378
0.235294	0.625	0.57377	0.18	0.144208	0.4307	0.455167	0.4	0.544155	0.449664
0	0.805	0.409836	0	0	0.326379	0.075149	0.733333	0.455914	0.534438
0.529412	0.61	0.459016	0	0	0.496274	0.442357	0.2	0.506639	0.484388

4.6.3.3 Cluster 3 test data results

Table 4.6.3.3(a): Cluster 3 test data (targeted output) of 4-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.117647	0.54	0.508197	0.1	0.328605	0.377049	0.342869	0.016667	0	1
0.058824	0.595	0.442623	0.13	0.059102	0.33234	0.054227	0.05	0	1
0.294118	0.43	0.557377	0.28	0.083924	0.450075	0.122118	0.05	0	1
0.117647	0.625	0.491803	0.2	0.165485	0.503726	0.00427	0.166667	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.176471	0.415	0.47541	0.31	0.021277	0.511177	0.110162	0.066667	0	1
0.058824	0.425	0.540984	0.29	0	0.396423	0.116567	0.166667	0	1
0.058824	0.755	0.491803	0	0	0.388972	0.043126	0.016667	0	1
0.352941	0.435	0.655738	0	0	0.345753	0.002562	0.183333	0	1

Table 4.6.3.3(b): Cluster 3 test data (forecasted output) of 4-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.117647	0.54	0.508197	0.1	0.328605	0.377049	0.342869	0.016667	0.156201	0.846462
0.058824	0.595	0.442623	0.13	0.059102	0.33234	0.054227	0.05	0.117209	0.885977
0.294118	0.43	0.557377	0.28	0.083924	0.450075	0.122118	0.05	0.168189	0.834692
0.117647	0.625	0.491803	0.2	0.165485	0.503726	0.00427	0.166667	0.191117	0.81155
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.176471	0.415	0.47541	0.31	0.021277	0.511177	0.110162	0.066667	0.145002	0.858046
0.058824	0.425	0.540984	0.29	0	0.396423	0.116567	0.166667	0.147568	0.855402
0.058824	0.755	0.491803	0	0	0.388972	0.043126	0.016667	0.16331	0.839576
0.352941	0.435	0.655738	0	0	0.345753	0.002562	0.183333	0.252798	0.749656

4.6.3.4 Cluster 4 test data results

Table 4.6.3.4(a): Cluster 4 test data (targeted output) of 4-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.058824	0.9	0	0	0	0.645306	0.087105	0.333333	1	0
0	0.655	0	0	0	0.643815	0.081981	0.083333	1	0
0.176471	0.705	0	0	0	0.447094	0.291631	0.1	1	0
0.117647	0.37	0	0	0	0	0.010248	0.016667	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.117647	0.435	0	0.23	0	0.4307	0.296755	0.066667	0	1
0.411765	0.525	0	0	0	0	0.096926	0.05	0	1
0.235294	0.66	0	0	0	0.490313	0.095645	0.033333	1	0
0	0.495	0	0	0	0.372578	0.074723	0.016667	0	1

Table 4.6.3.4(b): Cluster 4 test data (forecasted output) of 4-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.058824	0.9	0	0	0	0.645306	0.087105	0.333333	0.596192	0.352525
0	0.655	0	0	0	0.643815	0.081981	0.083333	0.744416	-0.01048
0.176471	0.705	0	0	0	0.447094	0.291631	0.1	0.716335	0.335963
0.117647	0.37	0	0	0	0	0.010248	0.016667	-0.4729	0.593225
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.117647	0.435	0	0.23	0	0.4307	0.296755	0.066667	0.566749	0.492319
0.411765	0.525	0	0	0	0	0.096926	0.05	-0.52655	0.719081
0.235294	0.66	0	0	0	0.490313	0.095645	0.033333	0.327472	0.240608
0	0.495	0	0	0	0.372578	0.074723	0.016667	0.307757	0.102799

4.6.4 Five Cluster System for Diabetes Data Set

Cluster1:105 training data and 30 test data

Cluster2:254 training data and 76 test data

Cluster3:98 training data and 37 test data

Cluster4:27 training data and 8 test data

Cluster5:92 training data and 41 test data

4.6.4.1 Cluster 1 test data results

Table 4.6.4.1(a): Cluster 1 test data (targeted output) of 5-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.352941	0.515	0.590164	0.32	0.224586	0.561848	0.105038	0.566667	0	1
0.411765	0.975	0.57377	0.33	0.171395	0.374069	0.036294	0.566667	1	0
0.588235	0.645	0.508197	0.36	0	0.614009	0.154996	0.283333	1	0
0.529412	0.76	0.639344	0.34	0.202128	0.509687	0.347993	0.2	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.764706	0.765	0.721311	0.37	0.165485	0.605067	0.467976	0.3	0	1
0.411765	0.92	0.688525	0.33	0	0.529061	0.118275	0.333333	1	0
0.529412	0.595	0.655738	0.35	0	0.432191	0.078992	0.133333	1	0
0.235294	0.625	0.57377	0.18	0.144208	0.4307	0.455167	0.4	1	0

Table 4.6.4.1(b): Cluster 1 test data (forecasted output) of 5-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.352941	0.515	0.590164	0.32	0.224586	0.561848	0.105038	0.566667	-0.00525	0.271721
0.411765	0.975	0.57377	0.33	0.171395	0.374069	0.036294	0.566667	0.323284	0.045881
0.588235	0.645	0.508197	0.36	0	0.614009	0.154996	0.283333	0.002699	0.22333
0.529412	0.76	0.639344	0.34	0.202128	0.509687	0.347993	0.2	0.36064	-0.01601
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.764706	0.765	0.721311	0.37	0.165485	0.605067	0.467976	0.3	0.360602	-0.14974
0.411765	0.92	0.688525	0.33	0	0.529061	0.118275	0.333333	0.38058	-0.10603
0.529412	0.595	0.655738	0.35	0	0.432191	0.078992	0.133333	-0.03765	0.357083
0.235294	0.625	0.57377	0.18	0.144208	0.4307	0.455167	0.4	0.377247	0.081719

4.6.4.2 Cluster 2 test data results

Table 4.6.4.2(a): Cluster 2 test data (targeted output) of 5-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.117647	0.54	0.508197	0.1	0.328605	0.377049	0.342869	0.016667	0	1
0.058824	0.595	0.442623	0.13	0.059102	0.33234	0.054227	0.05	0	1
0.294118	0.43	0.557377	0.28	0.083924	0.450075	0.122118	0.05	0	1
0.117647	0.625	0.491803	0.2	0.165485	0.503726	0.00427	0.166667	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0	0.585	0.540984	0.31	0.222222	0.459016	0.177199	0.016667	0	1
0.176471	0.415	0.47541	0.31	0.021277	0.511177	0.110162	0.066667	0	1
0.058824	0.425	0.540984	0.29	0	0.396423	0.116567	0.166667	0	1
0.058824	0.755	0.491803	0	0	0.388972	0.043126	0.016667	0	1

Table 4.6.4.2(b): Cluster 2 test data (forecasted output) of 5-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.117647	0.54	0.508197	0.1	0.328605	0.377049	0.342869	0.016667	0.163157	0.797219
0.058824	0.595	0.442623	0.13	0.059102	0.33234	0.054227	0.05	0.114264	0.836275
0.294118	0.43	0.557377	0.28	0.083924	0.450075	0.122118	0.05	0.148781	0.808158
0.117647	0.625	0.491803	0.2	0.165485	0.503726	0.00427	0.166667	0.167564	0.790514
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0	0.585	0.540984	0.31	0.222222	0.459016	0.177199	0.016667	0.138145	0.819759
0.176471	0.415	0.47541	0.31	0.021277	0.511177	0.110162	0.066667	0.140481	0.814414
0.058824	0.425	0.540984	0.29	0	0.396423	0.116567	0.166667	0.1543	0.801453
0.058824	0.755	0.491803	0	0	0.388972	0.043126	0.016667	0.119945	0.828366

4.6.4.3 Cluster 3 test data results

Table 4.6.4.3(a): Cluster 3 test data (targeted output) of 5-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.470588	0.54	0.57377	0	0	0.454545	0.374466	0.2	1	0
0.294118	0.52	0.606557	0	0	0.42921	0.032024	0.45	0	1
0.235294	0.6	0.557377	0	0	0.441133	0.269428	0.216667	0	1
0.117647	0.79	0.737705	0	0	0.470939	0.310418	0.75	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.235294	0.73	0.754098	0	0	0.464978	0.19684	0.666667	1	0
0	0.805	0.409836	0	0	0.326379	0.075149	0.733333	0	1
0.529412	0.61	0.459016	0	0	0.496274	0.442357	0.2	1	0
0.352941	0.435	0.655738	0	0	0.345753	0.002562	0.183333	0	1

Table 4.6.4.3(b): Cluster 3 test data (forecasted output) of 5-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.470588	0.54	0.57377	0	0	0.454545	0.374466	0.2	0.341588	0.677162
0.294118	0.52	0.606557	0	0	0.42921	0.032024	0.45	0.323932	0.693027
0.235294	0.6	0.557377	0	0	0.441133	0.269428	0.216667	0.392223	0.628582
0.117647	0.79	0.737705	0	0	0.470939	0.310418	0.75	0.423497	0.598405
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.235294	0.73	0.754098	0	0	0.464978	0.19684	0.666667	0.395773	0.624641
0	0.805	0.409836	0	0	0.326379	0.075149	0.733333	0.422926	0.597938
0.529412	0.61	0.459016	0	0	0.496274	0.442357	0.2	0.369709	0.650327
0.352941	0.435	0.655738	0	0	0.345753	0.002562	0.183333	0.302566	0.713638

4.6.4.4 Cluster 4 test data results

Table 4.6.4.4(a): Cluster 4 test data (targeted output) of 5-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.058824	0.9	0	0	0	0.645306	0.087105	0.333333	1	0
0	0.655	0	0	0	0.643815	0.081981	0.083333	1	0
0.176471	0.705	0	0	0	0.447094	0.291631	0.1	1	0
0.117647	0.37	0	0	0	0	0.010248	0.016667	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.117647	0.435	0	0.23	0	0.4307	0.296755	0.066667	0	1
0.411765	0.525	0	0	0	0	0.096926	0.05	0	1
0.235294	0.66	0	0	0	0.490313	0.095645	0.033333	1	0
0	0.495	0	0	0	0.372578	0.074723	0.016667	0	1

Table 4.6.4.4(b): Cluster 4 test data (forecasted output) of 5-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.058824	0.9	0	0	0	0.645306	0.087105	0.333333	0.584093	0.369787
0	0.655	0	0	0	0.643815	0.081981	0.083333	0.548134	0.409789
0.176471	0.705	0	0	0	0.447094	0.291631	0.1	0.556515	0.404215
0.117647	0.37	0	0	0	0	0.010248	0.016667	0.389059	0.611269
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.117647	0.435	0	0.23	0	0.4307	0.296755	0.066667	0.377708	0.62564
0.411765	0.525	0	0	0	0	0.096926	0.05	0.461397	0.525224
0.235294	0.66	0	0	0	0.490313	0.095645	0.033333	0.545347	0.417054
0	0.495	0	0	0	0.372578	0.074723	0.016667	0.489036	0.483679

4.6.4.5 Cluster 5 test data results

Table 4.6.4.5(a): Cluster 5 test data (targeted output) of 5-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.058824	0.58	0.639344	0.29	0.212766	0.538003	0.17848	0.066667	0	1
0	0.59	0.688525	0.47	0.271868	0.682563	0.201964	0.166667	1	0
0.117647	0.635	0.47541	0.24	0.325059	0.412817	0.649872	0.066667	0	1
0.294118	0.695	0.52459	0.35	0.165485	0.42623	0.142186	0.083333	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.117647	0.615	0.393443	0.32	0.195035	0.627422	0.188728	0.083333	0	1
0.117647	0.985	0.57377	0.45	0.641844	0.454545	0.034159	0.533333	1	0
0.352941	0.67	0.57377	0.23	0.153664	0.527571	0.198121	0.133333	1	0
0.176471	0.955	0.557377	0.15	0.153664	0.460507	0.094364	0.216667	0	1

Table 4.6.4.5(b): Cluster 5 test data (forecasted output) of 5-Cluster System

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
0.058824	0.58	0.639344	0.29	0.212766	0.538003	0.17848	0.066667	0.498695	0.491275
0	0.59	0.688525	0.47	0.271868	0.682563	0.201964	0.166667	0.504831	0.492606
0.117647	0.635	0.47541	0.24	0.325059	0.412817	0.649872	0.066667	0.492005	0.491007
0.294118	0.695	0.52459	0.35	0.165485	0.42623	0.142186	0.083333	0.476668	0.432303
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.117647	0.615	0.393443	0.32	0.195035	0.627422	0.188728	0.083333	0.516764	0.466562
0.117647	0.985	0.57377	0.45	0.641844	0.454545	0.034159	0.533333	0.605414	0.335461
0.352941	0.67	0.57377	0.23	0.153664	0.527571	0.198121	0.133333	0.467261	0.439355
0.176471	0.955	0.557377	0.15	0.153664	0.460507	0.094364	0.216667	0.577384	0.316821

4.7 Sample Data Set for Cardio

Cardio data set which contains 920 rows of data. Few sample data are given below:

Table 4.7.1: Sample Data Set

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.54	1	0	0	……	0	0	0	1	0	1
0.26	0	0	1	……	1	0	0	0	1	0
0.4	0	0	0	……	0	0	0	1	1	0
0.3	1	1	0	……	0	0	0	1	0	1
0.42	1	0	0	……	0	0	0	1	0	1
0.52	1	0	1	……	0	0	0	1	1	0
0.84	1	0	0	……	1	0	0	0	0	1
0.92	1	1	0	……	0	0	0	1	0	1
0.46	1	0	0	……	0	0	0	1	0	1
0.74	0	0	0	……	1	0	0	0	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.78	1	0	0	……	0	0	1	0	0	1
0.36	1	0	0	……	1	0	0	0	0	1
0.48	1	0	0	……	0	0	0	1	0	1
0.64	1	0	0	……	0	0	0	1	0	1
0.26	0	0	0	……	1	0	0	0	1	0
0.74	1	1	0	……	1	0	0	0	0	1
0.4	1	0	0	……	0	0	0	1	0	1
0.76	1	0	0	……	0	1	0	0	1	0

4.8 Result of without Clustering for Cardio Data Set

Table 4.8.1: Without clustering training data

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.54	1	0	0	……	0	0	0	1	0	1
0.26	0	0	1	……	1	0	0	0	1	0
0.4	0	0	0	……	0	0	0	1	1	0
0.3	1	1	0	……	0	0	0	1	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.64	1	0	0	……	0	0	1	0	0	1
0.24	1	0	0	……	0	0	1	0	0	1
0.06	1	0	0	……	0	0	0	1	0	1
0.34	0	0	0	……	1	0	0	0	1	0

Table 4.8.2: Without clustering test data (targeted output)

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.48	1	0	0	…….	1	0	0	0	1	0
0.56	1	0	0	…….	0	0	1	0	0	1
0.72	1	0	0	…….	1	0	0	0	0	1
0.52	1	0	0	…….	0	0	0	1	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.26	0	0	0	…….	1	0	0	0	1	0
0.74	1	1	0	…….	1	0	0	0	0	1
0.4	1	0	0	…….	0	0	0	1	0	1
0.76	1	0	0	…….	0	1	0	0	1	0

Table 4.8.3: Without clustering test data (forecasted output)

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.48	1	0	0	……	1	0	0	0	0.607314	0.381634
0.56	1	0	0	……	0	0	1	0	0.299646	0.83322
0.72	1	0	0	……	1	0	0	0	0.171843	0.8238
0.52	1	0	0	……	0	0	0	1	0.155524	0.939335
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.26	0	0	0	……	1	0	0	0	0.736531	0.303683
0.74	1	1	0	……	1	0	0	0	0.420291	0.654296
0.4	1	0	0	……	0	0	0	1	0.073602	0.857092
0.76	1	0	0	……	0	1	0	0	0.391347	0.541496

4.9 Result of n-Clustering for Cardio Data Set

We have created n number of cluster for 630 data using K-means clustering algorithm. Also 290 test data clustered accordingly. They are sorted below.

4.9.1 Two Cluster System for Cardio Data Set

Cluster1: 283 training data and 99 test data

Cluster2: 407 training data and 131 test data

4.9.1.1 Cluster 1 test data results

Table 4.9.1.1(a): Cluster 1 test data (targeted output) of 2-Cluster System

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.48	1	0	0	…….	1	0	0	0	1	0
0.56	1	0	0	…….	0	0	1	0	0	1
0.72	1	0	0	…….	1	0	0	0	0	1
0.5	0	0	0	…….	1	0	0	0	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.36	1	0	0	…….	1	0	0	0	0	1
0.26	0	0	0	…….	1	0	0	0	1	0
0.74	1	1	0	…….	1	0	0	0	0	1
0.76	1	0	0	…….	0	1	0	0	1	0

Table 4.9.1.1(b): Cluster 1 test data (forecasted output) of 2-Cluster System

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.48	1	0	0	…….	1	0	0	0	0.795988	0.182603
0.56	1	0	0	…….	0	0	1	0	0.289376	0.790112
0.72	1	0	0	…….	1	0	0	0	0.245594	0.807859
0.5	0	0	0	…….	1	0	0	0	0.710515	0.295124
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.36	1	0	0	…….	1	0	0	0	0.564746	0.442009
0.26	0	0	0	…….	1	0	0	0	0.850809	0.143052
0.74	1	1	0	…….	1	0	0	0	0.627978	0.397768
0.76	1	0	0	…….	0	1	0	0	0.444216	0.581579

4.9.1.2 Cluster 2 test data results

Table 4.9.1.2(a): Cluster 2 test data (targeted output) of 2-Cluster System

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.52	1	0	0	……	0	0	0	1	0	1
0.74	1	0	0	……	0	0	1	0	0	1
0.72	1	0	0	……	0	0	0	1	0	1
0.5	1	0	1	……	0	0	0	1	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.44	1	0	0	……	0	0	0	1	1	0
0.48	1	0	0	……	0	0	0	1	0	1
0.64	1	0	0	……	0	0	0	1	0	1
0.4	1	0	0	……	0	0	0	1	0	1

Table 4.9.1.2(b): Cluster 2 test data (forecasted output) of 2-Cluster System

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.52	1	0	0	……	0	0	0	1	-0.05791	0.990099
0.74	1	0	0	……	0	0	1	0	0.306538	0.732259
0.72	1	0	0	……	0	0	0	1	0.043486	0.95268
0.5	1	0	1	……	0	0	0	1	0.423146	0.620197
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.44	1	0	0	……	0	0	0	1	0.559463	0.49898
0.48	1	0	0	……	0	0	0	1	0.047	0.914612
0.64	1	0	0	……	0	0	0	1	0.025074	0.92129
0.4	1	0	0	……	0	0	0	1	0.027327	0.914525

4.9.2 Three Cluster System for Cardio Data Set

Cluster1: 216 training data and 77 test data

Cluster2: 243 training data and 80 test data

Cluster3: 231 training data and 73 test data

4.9.2.1 Cluster 1 test data results

Table 4.9.2.1(a): Cluster 1 test data (targeted output) of 3-Cluster System

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.48	1	0	0	……	1	0	0	0	1	0
0.56	1	0	0	……	0	0	1	0	0	1
0.72	1	0	0	……	1	0	0	0	0	1
0.5	0	0	0	……	1	0	0	0	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.36	1	0	0	……	1	0	0	0	0	1
0.26	0	0	0	……	1	0	0	0	1	0
0.74	1	1	0	……	1	0	0	0	0	1
0.76	1	0	0	……	0	1	0	0	1	0

Table 4.9.2.1.2(b): Cluster 1 test data (forecasted output) of 3-Cluster System

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.48	1	0	0	……	1	0	0	0	0.906175	0.143651
0.56	1	0	0	……	0	0	1	0	0.187051	0.588717
0.72	1	0	0	……	1	0	0	0	0.406008	0.49145
0.5	0	0	0	……	1	0	0	0	0.699604	0.212622
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.36	1	0	0	……	1	0	0	0	0.785376	0.26148
0.26	0	0	0	……	1	0	0	0	0.843898	0.033535
0.74	1	1	0	……	1	0	0	0	0.66754	0.191794
0.76	1	0	0	……	0	1	0	0	0.397874	0.470114

4.9.2.2 Cluster 2 test data results

Table 4.9.2.2(a): Cluster 2 test data (targeted output) of 3-Cluster System

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.74	1	0	0	……	0	0	1	0	0	1
0.5	1	0	1	……	0	0	0	1	1	0
0.6	1	0	0	……	0	0	0	1	0	1
0.18	0	0	1	……	0	0	0	1	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.5	0	0	1	……	0	0	0	1	1	0
0.54	0	0	1	……	0	0	0	1	1	0
0.16	1	0	1	……	0	0	0	1	1	0
0.44	1	0	0	……	0	0	0	1	1	0

Table 4.9.2.2(b): Cluster 2 test data (forecasted output) of 3-Cluster System

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.74	1	0	0	……	0	0	1	0	0.108073	0.876799
0.5	1	0	1	……	0	0	0	1	0.503984	0.491967
0.6	1	0	0	……	0	0	0	1	0.5162	0.471727
0.18	0	0	1	……	0	0	0	1	1.034234	-0.03602
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.5	0	0	1	……	0	0	0	1	0.849389	0.148866
0.54	0	0	1	……	0	0	0	1	0.878816	0.115094
0.16	1	0	1	……	0	0	0	1	1.004652	-0.00687
0.44	1	0	0	……	0	0	0	1	0.676596	0.319086

4.9.2.3 Cluster 3 test data results

Table 4.9.2.3(a): Cluster 3 test data (targeted output) of 3-Cluster System

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.52	1	0	0	……	0	0	0	1	0	1
0.72	1	0	0	……	0	0	0	1	0	1
0.66	1	0	0	……	0	1	0	0	0	1
0.82	1	0	0	……	0	1	0	0	0	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.56	1	0	0	……	0	0	1	0	1	0
0.48	1	0	0	……	0	0	0	1	0	1
0.64	1	0	0	……	0	0	0	1	0	1
0.4	1	0	0	……	0	0	0	1	0	1

Table 4.9.2.3(b): Cluster 3 test data (forecasted output) of 3-Cluster System

Input 1	**Input 2**	**Input 3**	**Input 4**	**……**	**Input 32**	**Input 33**	**Input 34**	**Input 35**	**Output 1**	**Output 2**
0.52	1	0	0	……	0	0	0	1	0.246434	0.871067
0.72	1	0	0	……	0	0	0	1	0.209028	0.943278
0.66	1	0	0	……	0	1	0	0	0.255253	0.747218
0.82	1	0	0	……	0	1	0	0	0.087545	0.771856
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0.56	1	0	0	……	0	0	1	0	0.261729	0.646882
0.48	1	0	0	……	0	0	0	1	0.165052	0.9384
0.64	1	0	0	……	0	0	0	1	0.350588	0.791652
0.4	1	0	0	……	0	0	0	1	0.232527	0.863004

4.10 Error Calculation

After generating the tables we needed to calculate the errors to see if the ANN prediction with clustering is better or not. We have calculated mean absolute deviation (MAD), Mean squared error (MSE) and Root mean squared error (RMSE) to check the performance of the network.

The process we followed is,

First we tested the network with the test data as we seen before and then we had the actual data and the predicted data. We subtracted the predicted data from actual output and then calculated the MAD, MSE and RMSE. The results are given below.

4.10.1 Error calculation for Cancer dataset

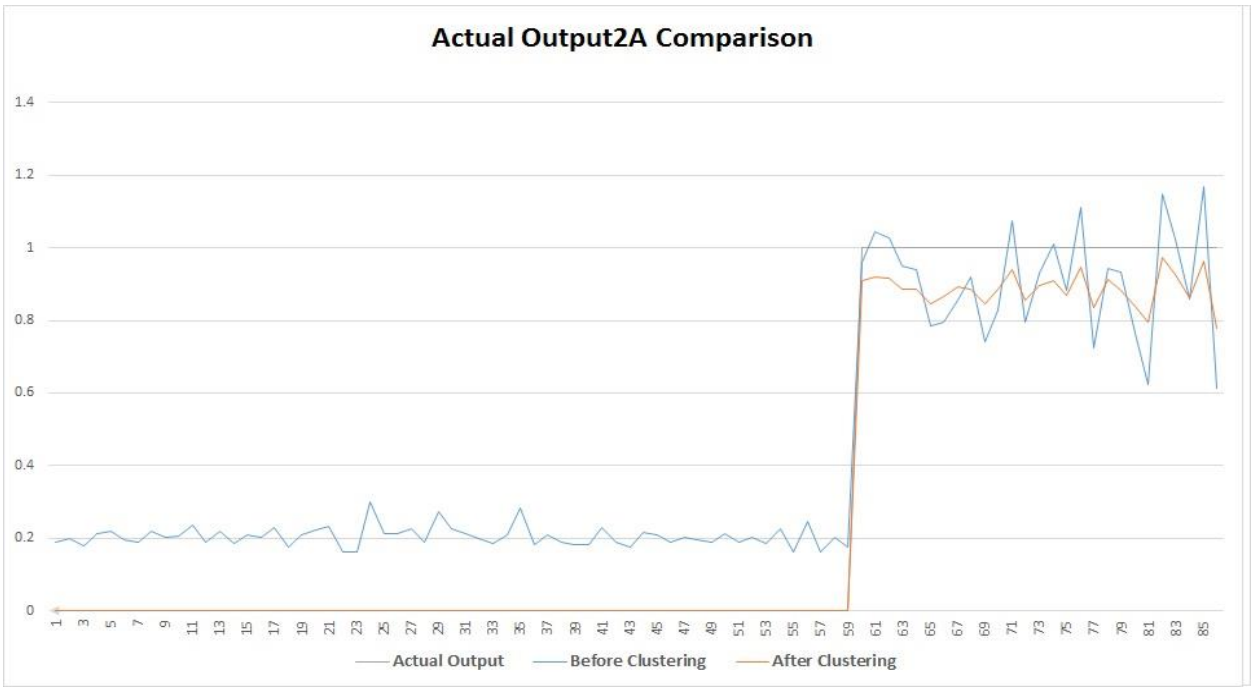
As we have told before that we have calculated the MAD, MSE and RMSE for all the dataset here is the result table for cancer dataset:

Table 4.10.1: Error Calculation for Cancer Dataset

		Output 1		**Output 2**			
ERROR	Cluster	Before	After	Before	After	Average before	Average after
MAD	2	0.21575	0.15146	0.21515	0.1145	0.21545	0.13298
	3	0.21575	0.07728	0.21515	0.08215	0.21545	0.079715
	4	0.21575	0.12863	0.21515	0.11545	0.21545	0.12204
MSE	2	0.05767	0.0370076	.0.05741	0.02813737	0.05767	0.032572485
	3	0.05767	0.02129	.0.05742	0.02227	0.05767	0.02178
	4	0.05767	0.02825	.0.05743	0.02552	0.05767	0.026885
	5	0.05767	0.02728	.0.05744	0.03128	0.05767	0.02928
RMSE	2	0.240151	0.19237361	0.239595	0.16774198	0.239873	0.180057791
	3	0.240151	0.145913	0.239595	0.149239	0.239873	0.147576

Now here is data comparison for cluster 3 which is better from the above table

Figure 4.1: output 1 comparison for 3 cluster system on cancer dataset



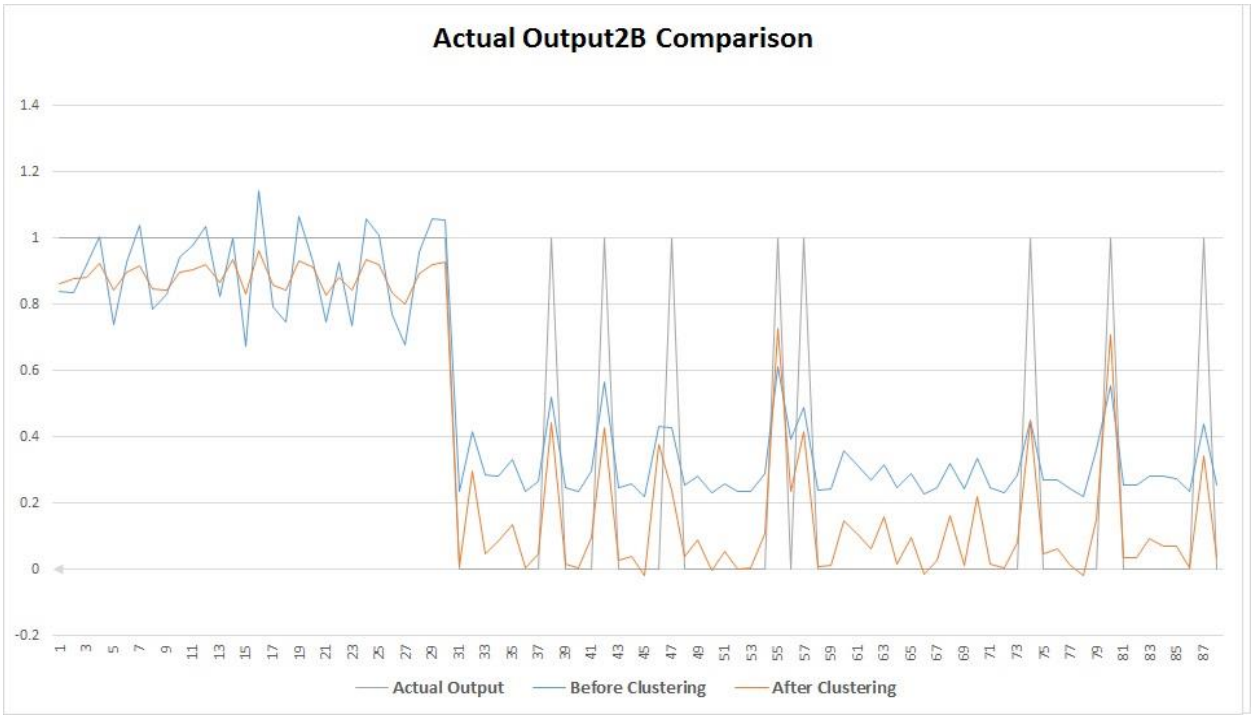
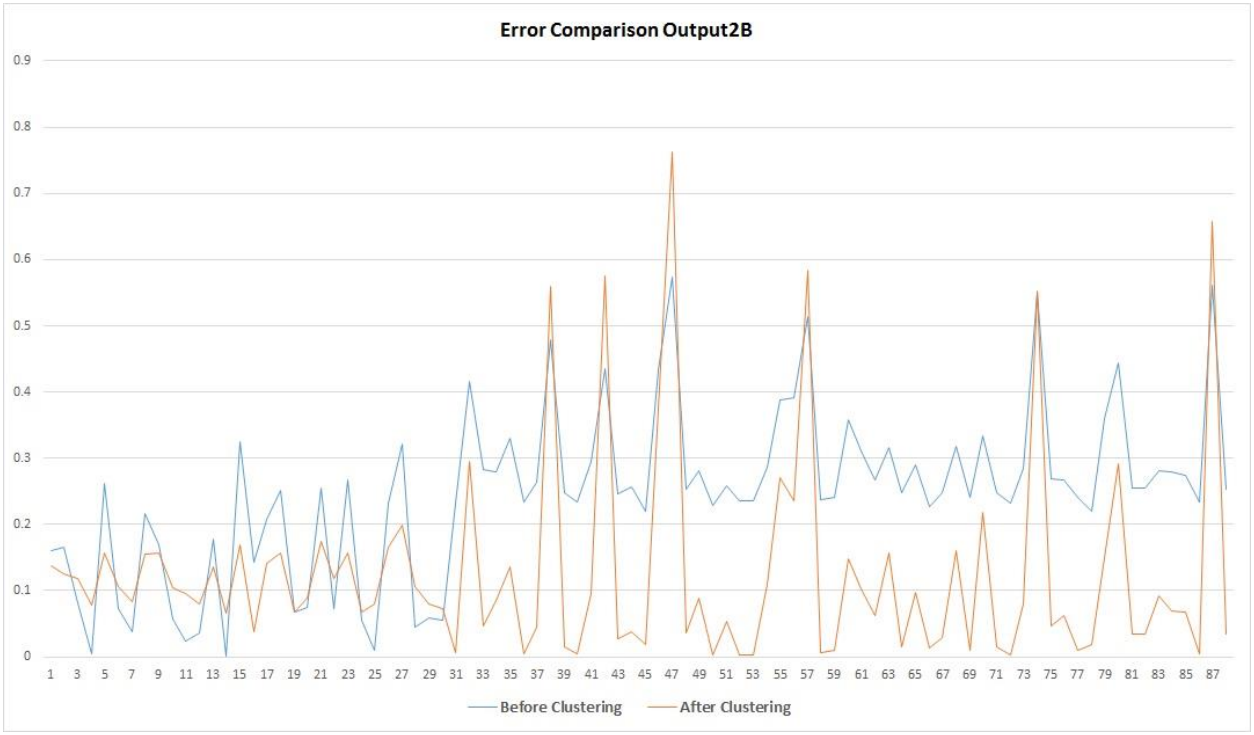


Figure 4.2: output 2 comparison for 3 cluster system on cancer dataset

Here is the error comparison graph for the same dataset and cluster



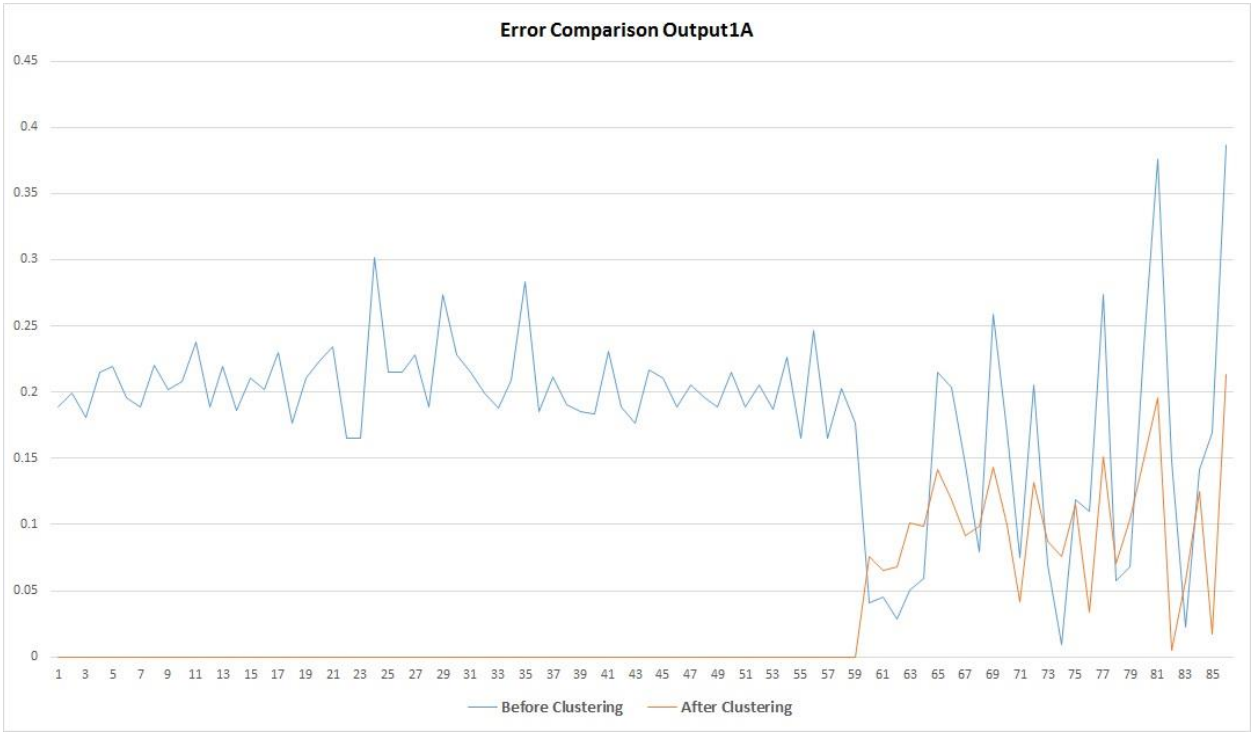
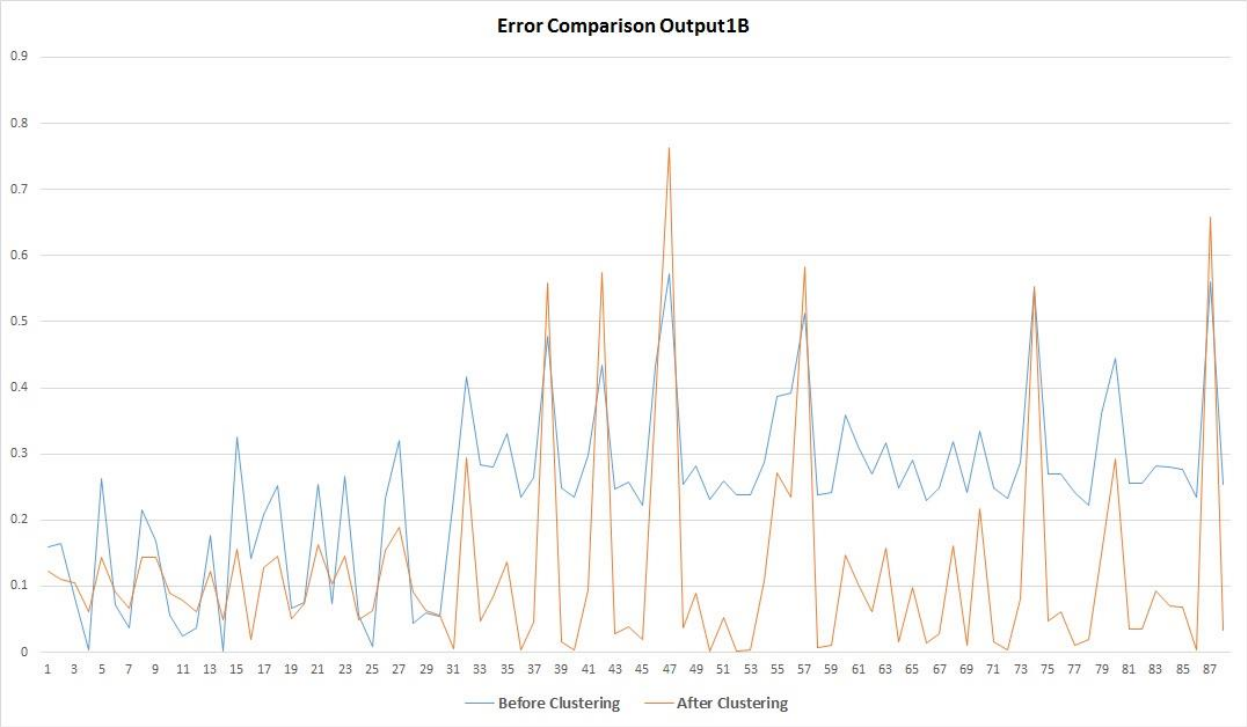


Figure 4.3: error comparison graph for cancer test data output 1



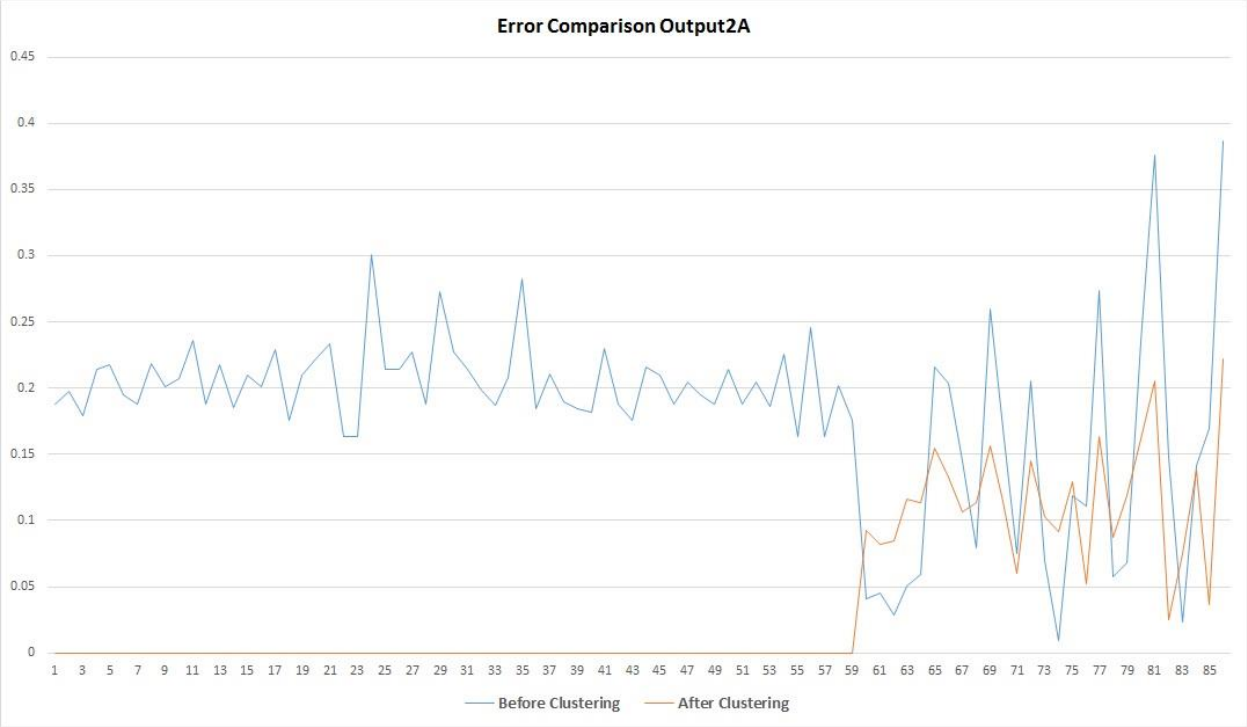


Figure 4.4: error comparison graph for cancer test data output 2

So from all the error calculation methods we can conclude that for 9-8-2 structured network the cancer dataset we provide us best result if we create 3 cluster on the dataset

4.10.2 Error calculation for Diabetes dataset

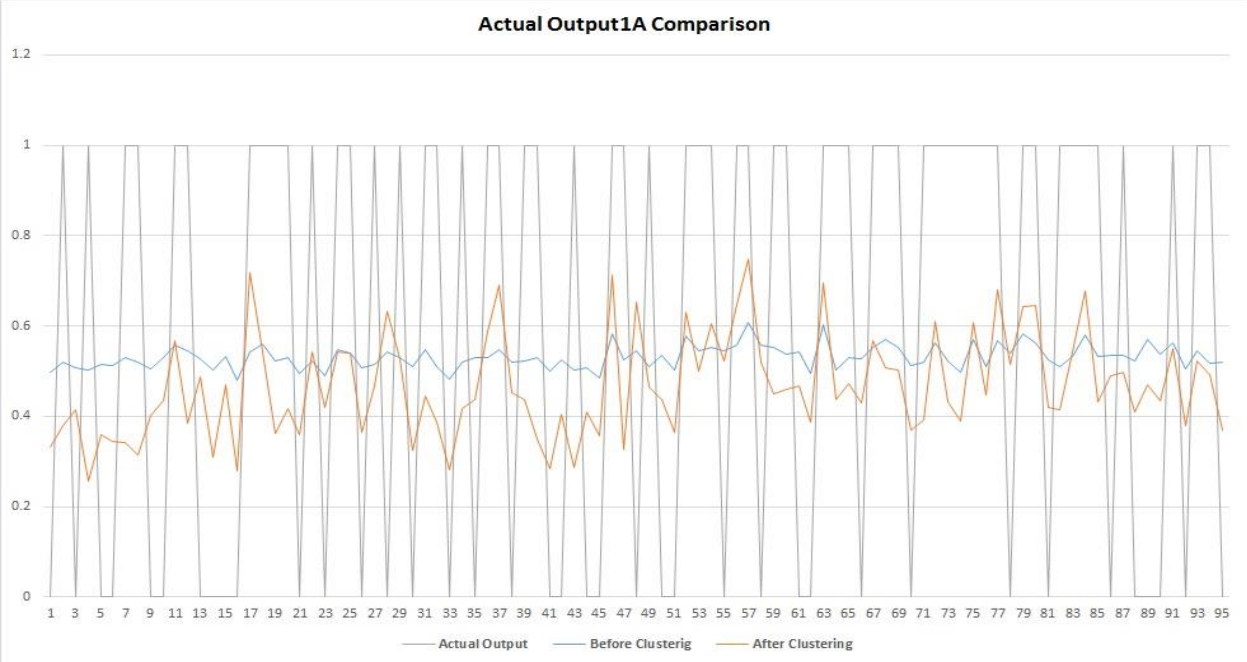
Here is the data table for error calculation for the Diabetes dataset,

Table 4.10.2: Error Calculation for diabetes dataset

		Output 1		**Output 2**			
ERROR	Cluster	Before	After	Before	After	Average before	Average after
MAD	2	0.481557	0.401735	0.461525	0.39894	0.471541	0.4003375
	3	0.481557	0.4037202	0.461525	0.414469	0.471541	0.409094475
	4	0.481557	0.375750405	0.461525	0.371499	0.471541	0.373624462
	5	0.481557	0.3452797	0.461525	0.671795	0.471541	0.508537269
MSE	2	0.232951	0.17937621	0.21980028	0.17811	0.2263754	0.178743105
	3	0.232951	0.1963104	0.21980028	0.198352	0.2263754	0.19733142
	4	0.232951	0.17683785	0.21980028	0.175093	0.2263754	0.175965345
	5	0.232951	0.14474824	0.21980028	0.474649	0.2263754	0.309698535
RMSE	2	0.482649	0.423528286	0.46882862	0.422027	0.4757390	0.422777698
	3	0.482649	0.443069295	0.46882862	0.445368	0.4757390	0.444218525
	4	0.482649	0.420520927	0.46882862	0.418441	0.4757390	0.419480947
	5	0.482649	0.380457928	0.46882862	0.688948	0.4757390	0.534702776

From this data table we can see 4 cluster system is better and it's performance is best.

Now here is output comparison for cluster 3 which is better from the above table



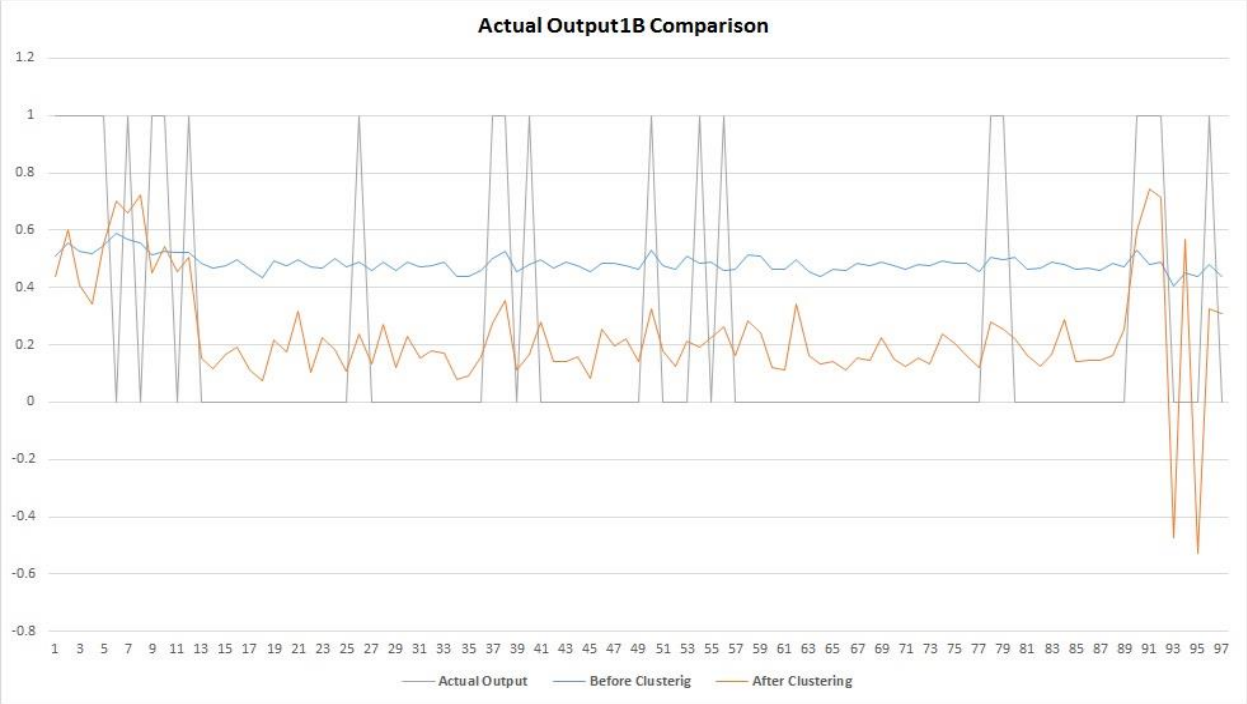
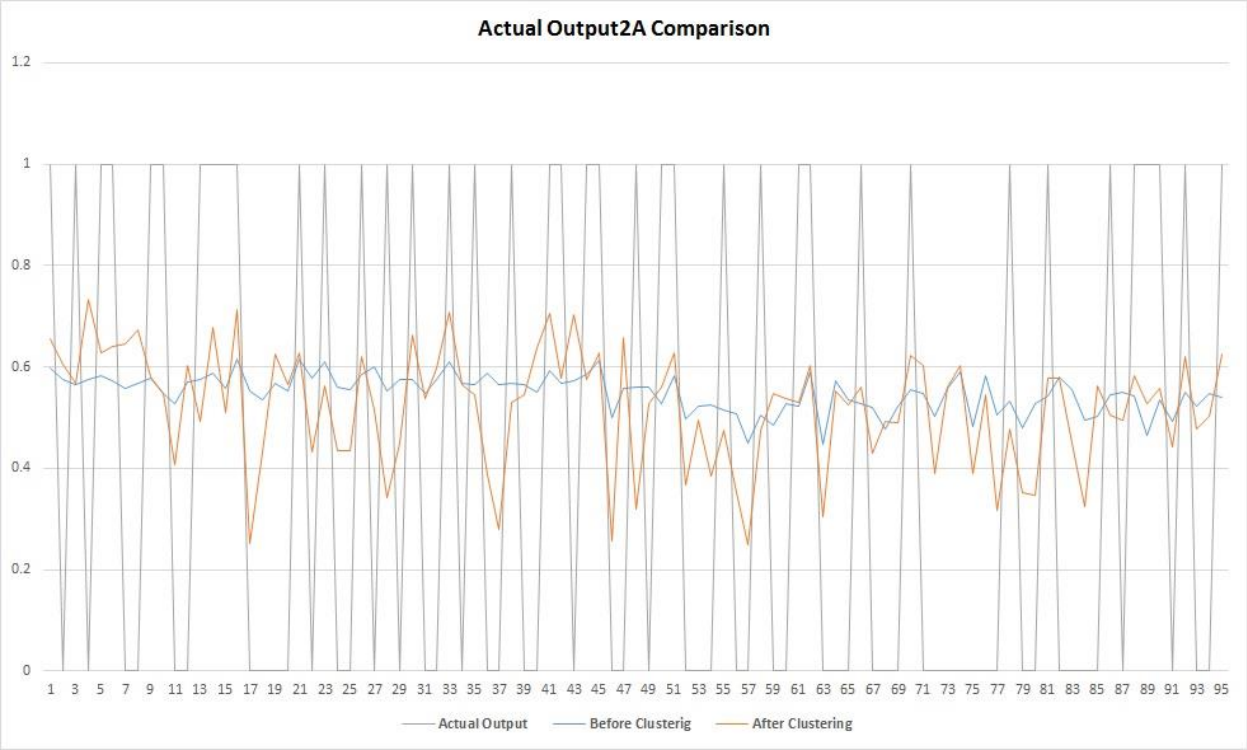


Figure 4.5: output 1 comparison for 4 cluster system on Diabetes dataset



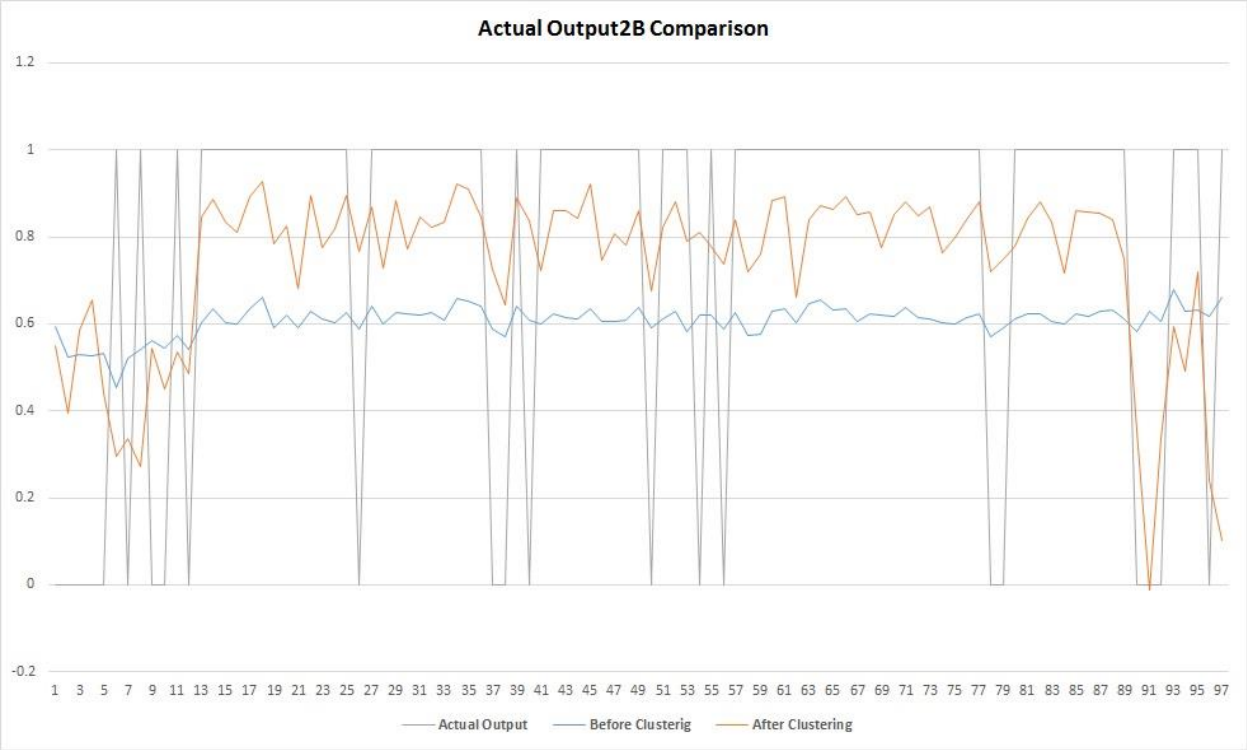
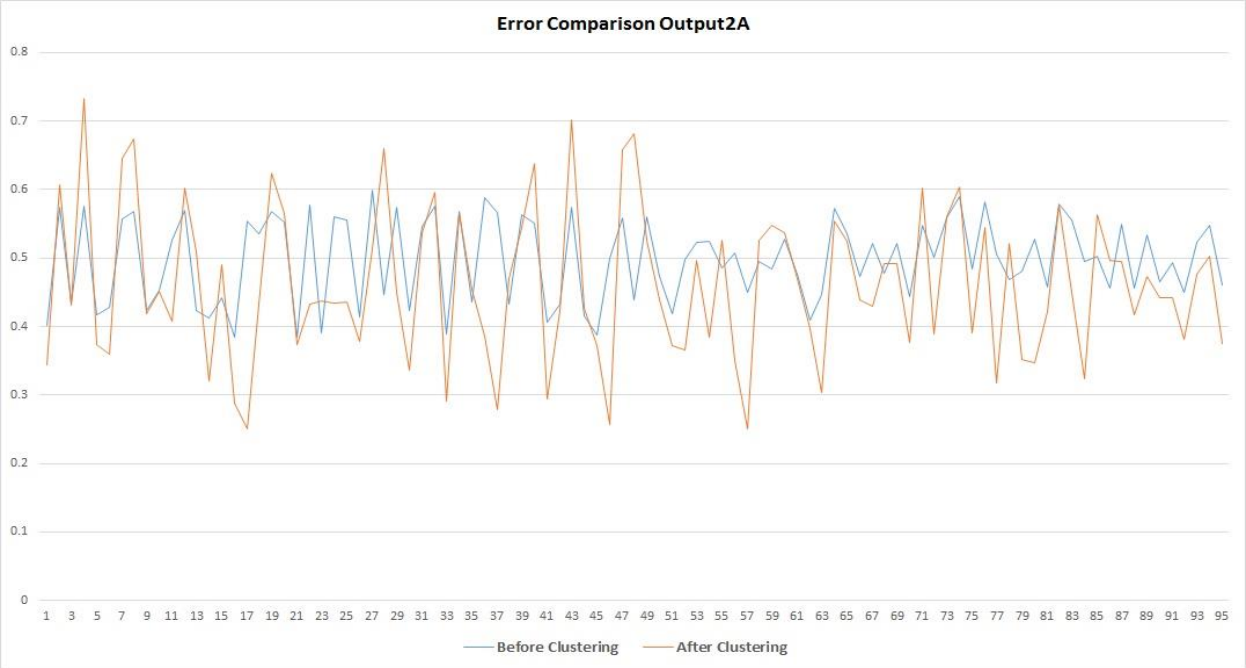


Figure 4.6: output 2 comparison for 4 cluster system on Diabetes dataset

Here is the error calculation graph,

Figure 4.7: error comparison graph for Diabetes test data output 1



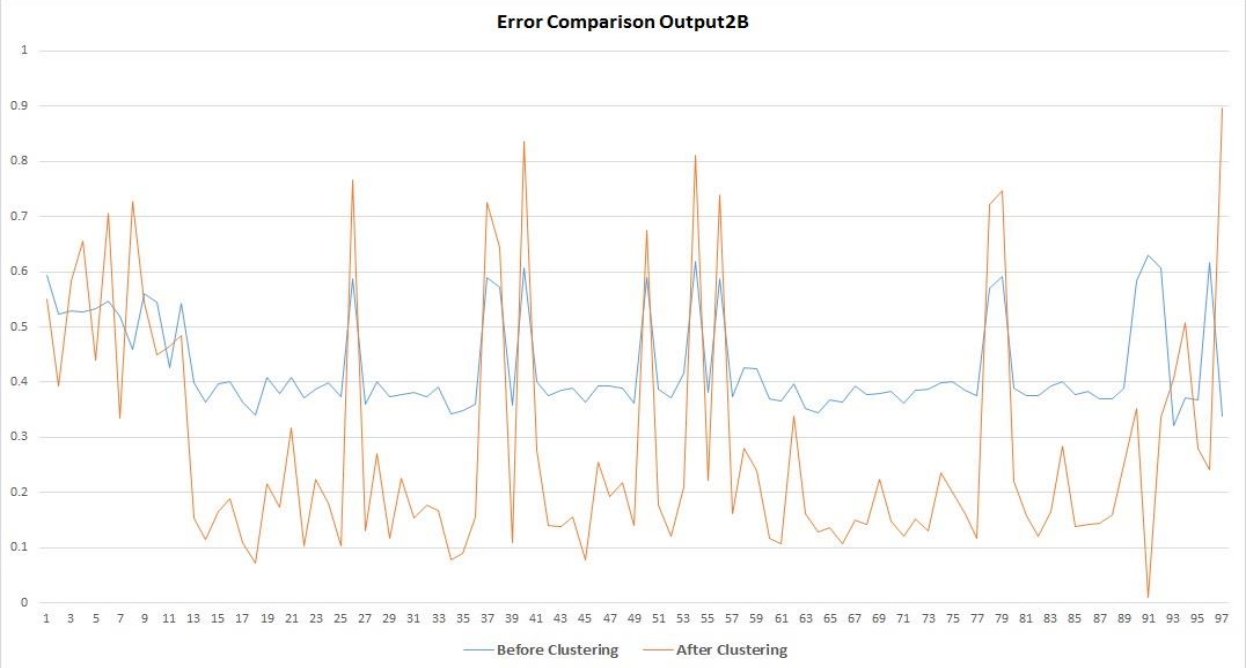


Figure 4.8: error comparison graph for Diabetes test data output 2

So we can conclude that if we have the diabetes dataset and 8-8-2 network structure then 4 cluster will provide us better performance.

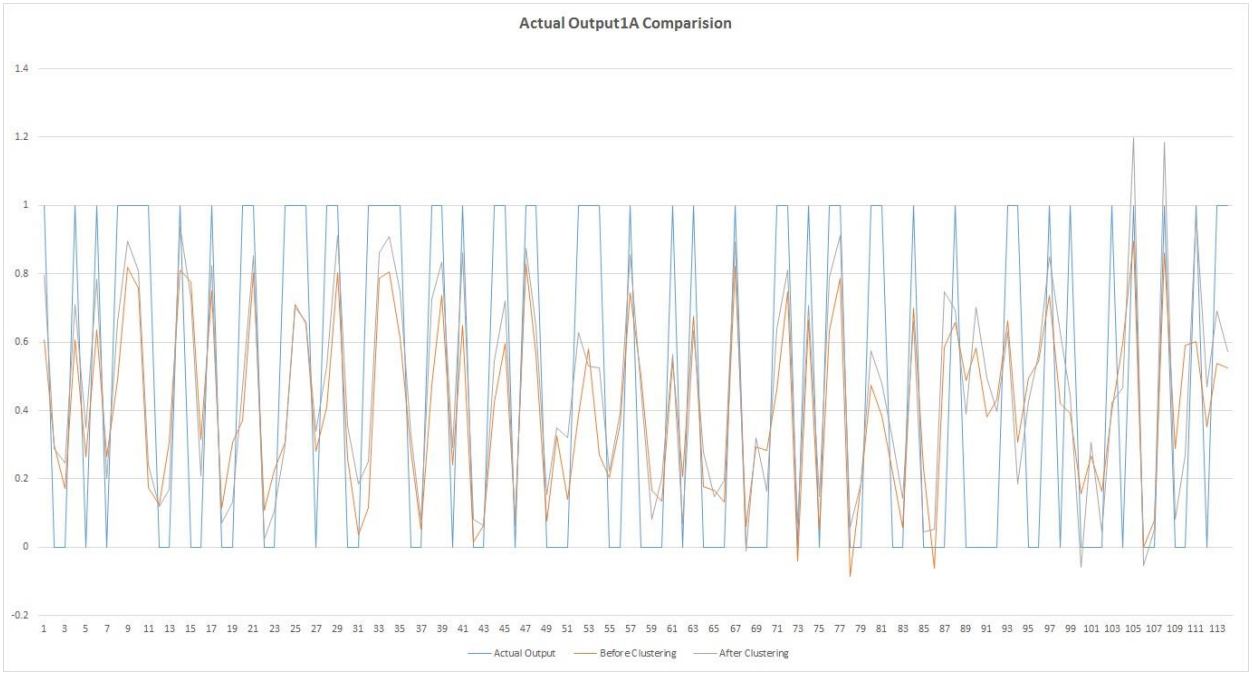
4.10.3 Error calculation table for Cardio data

Table 4.10.3: Error Calculation for Cardio Dataset

		Output 1		**Output 2**			
ERROR	Cluster	Before	After	Before	After	Average after	Average before
MAD	2	0.3314	0.28731179	0.35496314	0.280571243	0.343180965	0.283941518
	3	0.3314	0.2986338	0.35496314	0.300692212	0.343180965	0.299663005
MSE	2	0.15666	0.13767458	0.17100002	0.134125573	0.163830027	0.135900078
	3	0.15666	0.13882273	0.17100002	0.143961339	0.163830027	0.141392036
RMSE	2	0.3958	0.37104526	0.41352147	0.366231583	0.404662245	0.36863842
	3	0.3958	0.37258923	0.41352147	0.379422375	0.404662245	0.376005802

From this data table we can see that before clustering and after clustering results are changed a little.

Here is the Output comparison graph for output 1 and 2



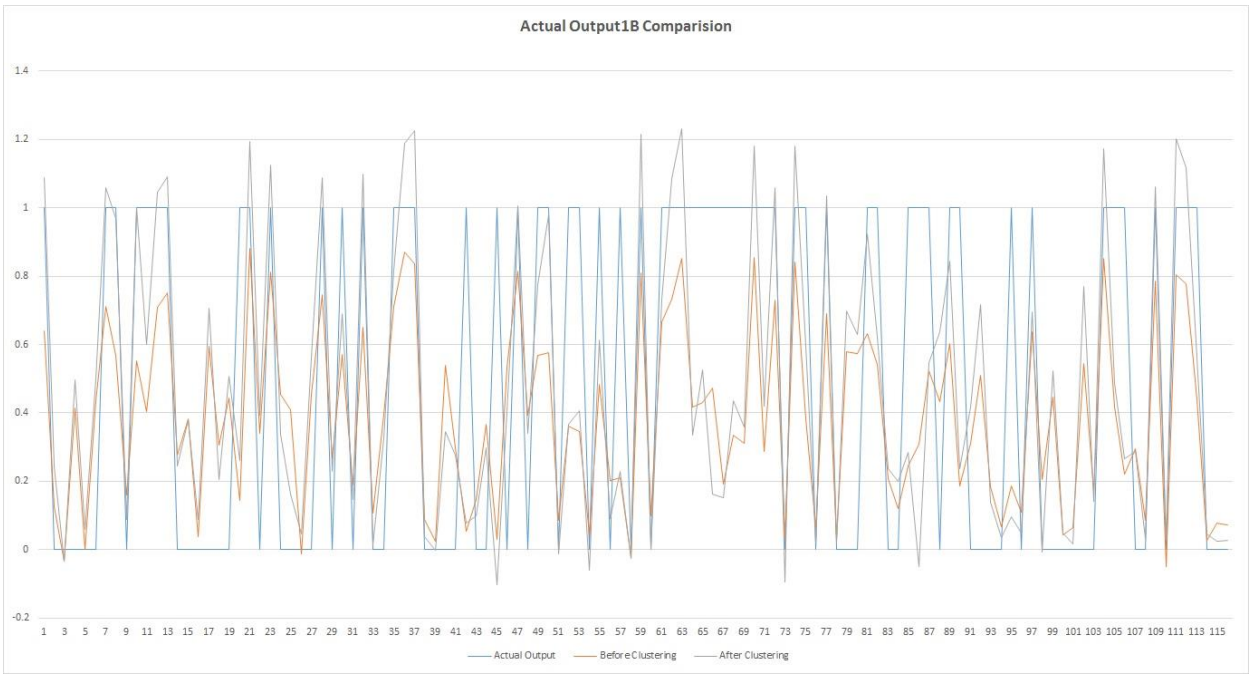
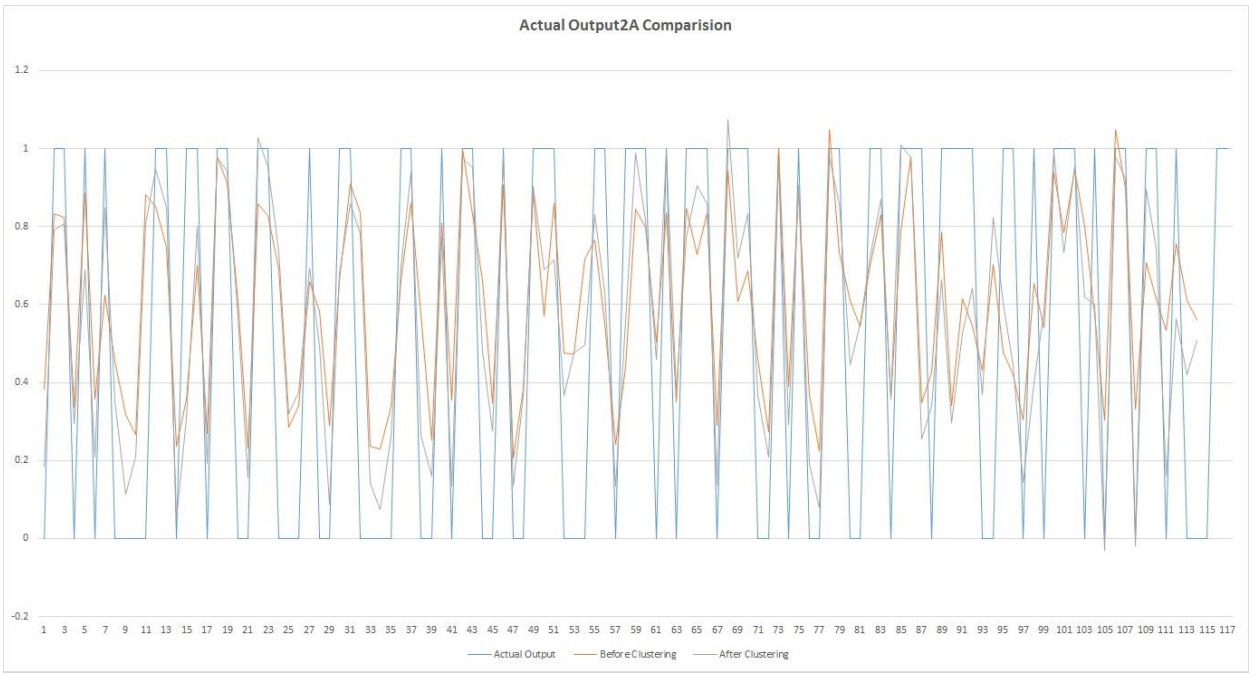


Figure 4.9: Output 1 Comparison graph for 2 cluster system on Cardio Dataset



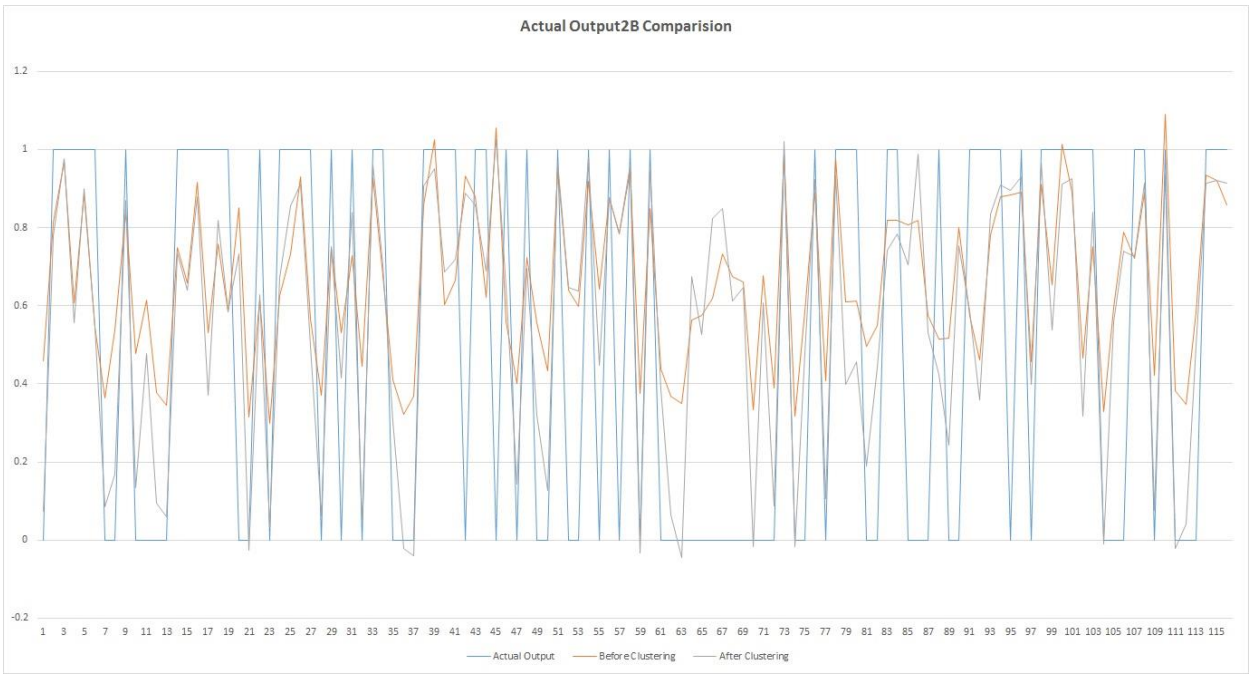
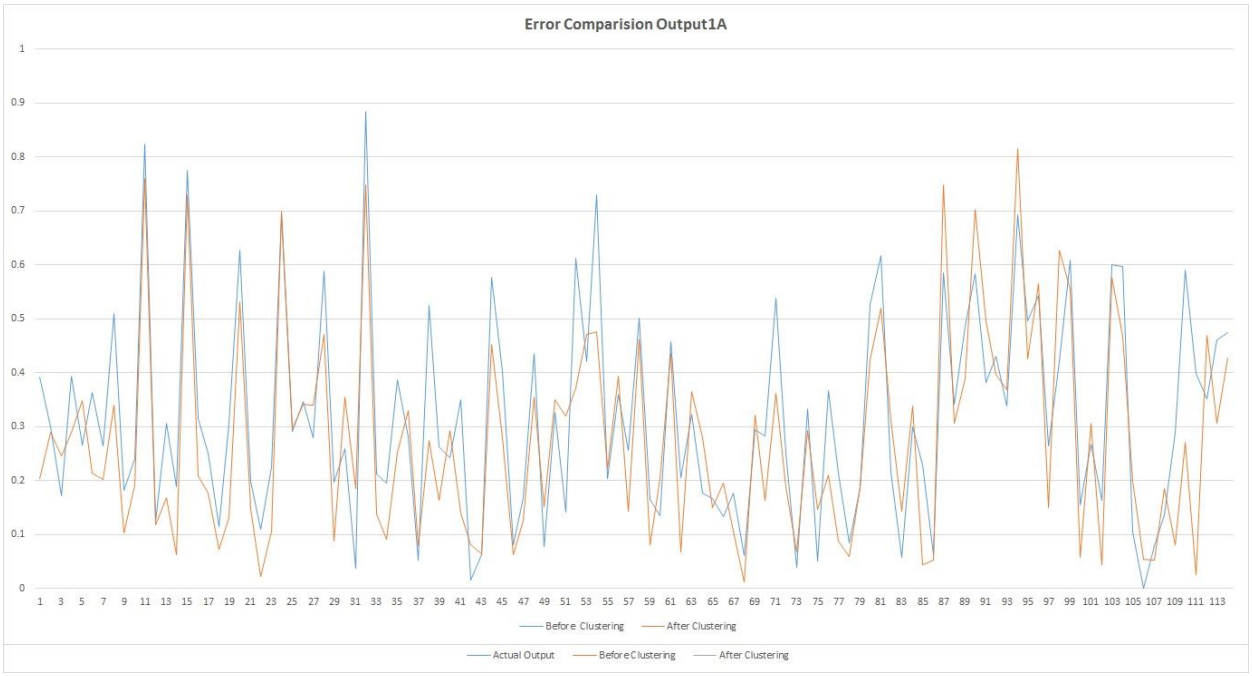


Figure 4.10: Output 1 Comparison graph for 2 cluster system on Cardio Dataset

And here is the error calculation graph,



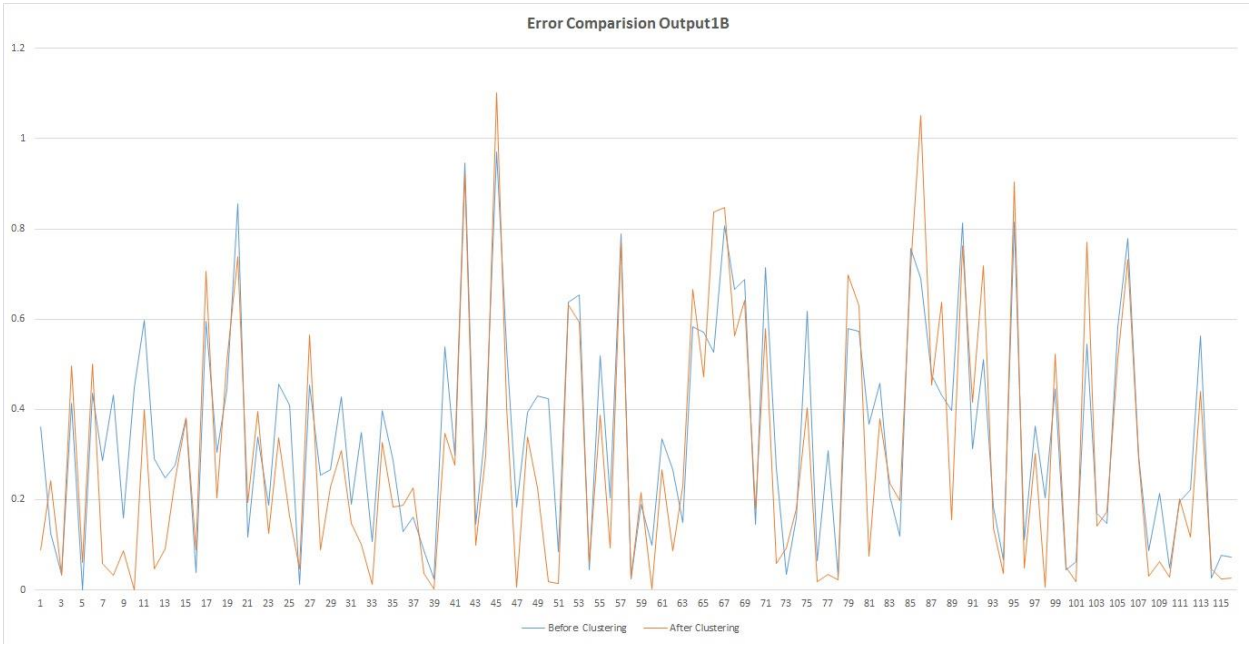


Figure 4.11: error comparison graph for Cardio test data output 2

Figure 4.12: error comparison graph for Cardio test data output 2

So we can conclude that 2 cluster system is better for Cardio dataset and 35-8-2 structured ANN.

4.11 Final Decision

Finally we have seen that MAD, MSE, RMSE of all the dataset and we can conclude that clustering technique gives us much better result for neural network system. For the 3 kind of dataset we have seen that each dataset works best for different number of cluster. So it's not necessary that a fixed number cluster will give us better result. The number of cluster may vary from data type and dataset. After using clustering technique the prediction for diseases are much more efficient without clustering technique.

Chapter 5
Conclusion

5.1 Summary

By our proposed model a disease detection can be forecasted and predicted if someone has a particular disease or not. In our Trial and error method we calculated the MAD, MSE and RMSE to calculate the efficiency and we found the error is lower for clustering technique than without clustering. So we have successfully optimized our dataset with clustering technique

5.2 Future Work

Some of the work we will improve in future is given bellow,

a. In our work we have just scaled some input columns and by trial and error approach fix the final input combination (how many input columns are taken as ANN's input) but we can minimize the dimensionality of the input data by removing the correlation among themselves by PCA or any other techniques like LDA, ICA etc. In future we will do these steps to reduce the dataset for better performance.
b. We have worked with a single network structure for all the dataset where there is 8 hidden layer. In future we will change the structure to measure its performance.
c. In our research we did not normalize [-1:1] data which theoretically fits best for this kind of problem so in future we will make experimental data feasible enough for better approximation.
d. Due to the shortage of time we could not work with lots of data so in future we will gather more data and experiment for a better result
e. Mainly we have done our experiment on k-means clustering technique where there are some more clustering technique like fuzzy c-mean, hierarchical etc. In future,

we will work with this technique for different clustering algorithm to visualize and analyze our research in greater extent.

References

[1] J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations, *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*", Berkeley, University of California Press, 1:281-297.

[2] P.Fishwick, "Neural network models in simulation: A comparison with traditional modeling approaches." Working Paper, University of Florida, Gainesville, FL, 1989.

[3] Dr. John A. Bullinaria, "Introduction to Neural Networks- 2nd Year UG, MSc in Computer Science: Lecture Series".

[4] K. Mehrota, C. Mohan, and S. Ranka. *Elements of Artificial Neural Networks*. MIT Press, 1996.

[5] D.E Rumelhart, G.E. Hinton, and R.J. Williams. "Learning internal representations by error propagation, "D.E. Rumelhart and J. McClelland, editors, *Parallel Data Processing*, Vol.1.

[6] The M.I.T. Press, Cambridge, A, 1986, PP. 318-362.

[7] M.T. Hagan, H.B. Demuth, and M.H. Beale, *Neural Introduction*.

[8] De Jesus, O., J.M. Horn, and M.T. Hagan, "Analysis of Recurrent Network Training and Suggestions for Improvements," *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC, July 15-19, 2001, pp. 2632-2637.

[9] De Jesus, O., and M.T. Hagan, "Forward Perturbation Algorithm for a General Class of Recurrent Network," *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC, July 15-19, 2001, pp. 2626-2631.

[10] *DARPA Neural Network Study*, Lexington, MA: M.I.T. Lincoln Laboratory, 1988.

[11] James A. Anderson. A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14: 197-220, 1972.

[12] A. Barton, R. Sutton, and C. Anderson. Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5): 834-846, September 1983.

[13] G. Cybenko. Approximation by super positions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303-314, 1989.

[14] K. Fukushima, S. Miyake, and T. Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5): 826-834, September/October 1983.

[15] M. Ankerst, M. M. Breuning, H.-P. Krieger, and J. Sander. OPTICS: Ordering Points to Identify the Clustering Structure. In Proc. Of 1999 ACM-SIGMOID Intl. Conf. on Management of Data, pages 49-60, Philadelphia, Pennsylvania, June 1999. ACM Press.

[16] P. Arabie, L. Hubert, and G. D. Soete. An overview of combinatorial data analysis. In P. Arabie, L. Hubert, and G. D. Soete, editors, Clustering and Classification, pages 188-217. World Scientific, Singapore, January 1996.

[17] M. S. Aldenderfer and R. K. Blashfield. Cluster Analysis. Sage Publications, Los Angeles, 1985.

[18] M. R. Anderberg. Cluster Analysis for Applications. Academic Press, New York, December 1973.

[19] Fahim A. M., Salem A. M., Torkey F. A. and Ramadan M. A., "An efficient enhanced k-means clustering algorithm," Journal of Zhejiang University Science A., pp. 1626-1633, 2006.

[20] H. Zha, C. Ding, M. Gu, X. He and H. D. Simon. "Spectral Relaxation for K-means Clustering", Neural; Information Processing Systems vol.14 (NIPS 2001). PP. 1057-1064, Vancouver, Canada. Dec. 2001.

[21] J. A. Hartigan (1975) "Clustering Algorithms". Wiley.

[22] J. A. Hartigan and M. A. Wong (1979) “A K-Means Clustering Algorithm”, Applied Statistics, Vp;.28, No. 1,pp. 100-108.

[23] D. Arthur, S. Vassilvitskii (2006): “How Slow is the k-means Method?.”

Chapter 5
Conclusion

5.1 Summary

By our proposed model a disease detection can be forecasted and predicted if someone has a particular disease or not. In our Trial and error method we calculated the MAD, MSE and RMSE to calculate the efficiency and we found the error is lower for clustering technique than without clustering. So we have successfully optimized our dataset with clustering technique

5.2 Future Work

Some of the work we will improve in future is given bellow,

f. In our work we have just scaled some input columns and by trial and error approach fix the final input combination (how many input columns are taken as ANN's input) but we can minimize the dimensionality of the input data by removing the correlation among themselves by PCA or any other techniques like LDA, ICA etc. In future we will do these steps to reduce the dataset for better performance.
g. We have worked with a single network structure for all the dataset where there is 8 hidden layer. In future we will change the structure to measure its performance.
h. In our research we did not normalize [-1:1] data which theoretically fits best for this kind of problem so in future we will make experimental data feasible enough for better approximation.
i. Due to the shortage of time we could not work with lots of data so in future we will gather more data and experiment for a better result
j. Mainly we have done our experiment on k-means clustering technique where there are some more clustering technique like fuzzy c-mean, hierarchical etc. In future, we will work with this technique for different clustering algorithm to visualize and analyze our research in greater extent.

References

[1] J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations, *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*", Berkeley, University of California Press, 1:281-297.

[2] P.Fishwick, "Neural network models in simulation: A comparison with traditional modeling approaches." Working Paper, University of Florida, Gainesville, FL, 1989.

[3] Dr. John A. Bullinaria, "Introduction to Neural Networks- 2nd Year UG, MSc in Computer Science: Lecture Series".

[4] K. Mehrota, C. Mohan, and S. Ranka. *Elements of Artificial Neural Networks*. MIT Press, 1996.

[5] D.E Rumelhart, G.E. Hinton, and R.J. Williams. "Learning internal representations by error propagation, "D.E. Rumelhart and J. McClelland, editors, *Parallel Data Processing*, Vol.1.

[6] The M.I.T. Press, Cambridge, A, 1986, PP. 318-362.

[7] M.T. Hagan, H.B. Demuth, and M.H. Beale, *Neural Introduction*.

[8] De Jesus, O., J.M. Horn, and M.T. Hagan, "Analysis of Recurrent Network Training and Suggestions for Improvements," *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC, July 15-19, 2001, pp. 2632-2637.

[9] De Jesus, O., and M.T. Hagan, "Forward Perturbation Algorithm for a General Class of Recurrent Network," *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC, July 15-19, 2001, pp. 2626-2631.

[10] *DARPA Neural Network Study*, Lexington, MA: M.I.T. Lincoln Laboratory, 1988.

[11] James A. Anderson. A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14: 197-220, 1972.

[12] A. Barton, R. Sutton, and C. Anderson. Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5): 834-846, September 1983.

[13] G. Cybenko. Approximation by super positions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303-314, 1989.

[14] K. Fukushima, S. Miyake, and T. Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5): 826-834, September/October 1983.

[15] M. Ankerst, M. M. Breuning, H.-P. Krieger, and J. Sander. OPTICS: Ordering Points to Identify the Clustering Structure. In Proc. Of 1999 ACM-SIGMOID Intl. Conf. on Management of Data, pages 49-60, Philadelphia, Pennsylvania, June 1999. ACM Press.

[16] P. Arabie, L. Hubert, and G. D. Soete. An overview of combinatorial data analysis. In P. Arabie, L. Hubert, and G. D. Soete, editors, Clustering and Classification, pages 188-217. World Scientific, Singapore, January 1996.

[17] M. S. Aldenderfer and R. K. Blashfield. Cluster Analysis. Sage Publications, Los Angeles, 1985.

[18] M. R. Anderberg. Cluster Analysis for Applications. Academic Press, New York, December 1973.

[19] Fahim A. M., Salem A. M., Torkey F. A. and Ramadan M. A., "An efficient enhanced k-means clustering algorithm," Journal of Zhejiang University Science A., pp. 1626-1633, 2006.

[20] H. Zha, C. Ding, M. Gu, X. He and H. D. Simon. "Spectral Relaxation for K-means Clustering", Neural; Information Processing Systems vol.14 (NIPS 2001). PP. 1057-1064, Vancouver, Canada. Dec. 2001.

[21] J. A. Hartigan (1975) "Clustering Algorithms". Wiley.

[22] J. A. Hartigan and M. A. Wong (1979) “A K-Means Clustering Algorithm”, Applied Statistics, Vp;.28, No. 1,pp. 100-108.

[23] D. Arthur, S. Vassilvitskii (2006): “How Slow is the k-means Method?.”