Contents

Chapter 1: Introduction

1.1. Classification

The problem of data classification has numerous applications in a wide variety of mining applications. This is because the problem attempts to learn the relationship between a set of feature variables and a target variable of interest. By definition, classification is the technique of analyzing a set of data and generating a set of grouping rules which can be used to classify future data. This is one of the most defining techniques in data-mining and widely used in various fields. The data analysis task is called classification where a model or classifier is constructed to predict categorical labels. Classification task begins with a data set in which the class assignments are known. Classes are discrete and do not imply order. A predictive model with a numerical target uses a regression algorithm, not a classification algorithm. The simplest type of classification problem is binary classification. In binary classification, the target attribute has only two possible values. In multiclass or multinomial classification targets have more than two values. In the model build (training) process, a classification algorithm finds relationships between the values of the predictors and the values of the target. Different classification algorithms use different techniques for finding relationships. These relationships are summarized in a model, which can then be applied to a different data set in which the class assignments are unknown.

1.2. Various Types of classification technique in data mining

- Decision tree
- Rule based classifier
- Naive Bayes Classifier
- K-Nearest Neighbour
- Artificial Neural Network
- Support Vector Machines

1.3. Basic concepts of neural network in human brain

The human brain is one of the most complicated organs which have been poorly understood. A human nervous system consists of billions of 'neurons' which is the basic unit of brain. Neuron has three major functional units i.e. cell body, dendrites and axon.

- Cell body-The cell body (soma) contains one nucleus. It is the source of most of the RNA which contains the heredity traits.
- Dendrites-Dendrites are long irregularly shaped filaments attached with cell body behaved like input channels. It receives the signals from neurons and passes the signals to next neuron.
- Axon-Axon is a long slender projection of a neuron. It is electrically active and serves as an output channel. It receives the signals from cell body and carries through synapse (very small gap) to the dendrites of the next neurons.
- Synapse-This is a microscopic gap that connects the axons with dendrite links of another neuron. It contains a neurotransmitter fluid. In the form of an electric signal an impulse travels through the cell body within the dendrites towards the pre-synaptic membranes of the synapse.

  A neurotransmitter chemical released from the vesicles which is proportional to the strength of the input signal at the time of arrival of the signal from pre-synaptic cell to post synaptic cell which is the receiving part of next neuron. Now the post-synaptic cell passes the signals through the dendrites to the main part of the neuron body i.e. in cell body. Now the signals from different dendrites are combined to produce the output signal which passes through the axon. The output signal is only produced if the input signals are of sufficient of strength to overcome some threshold values.

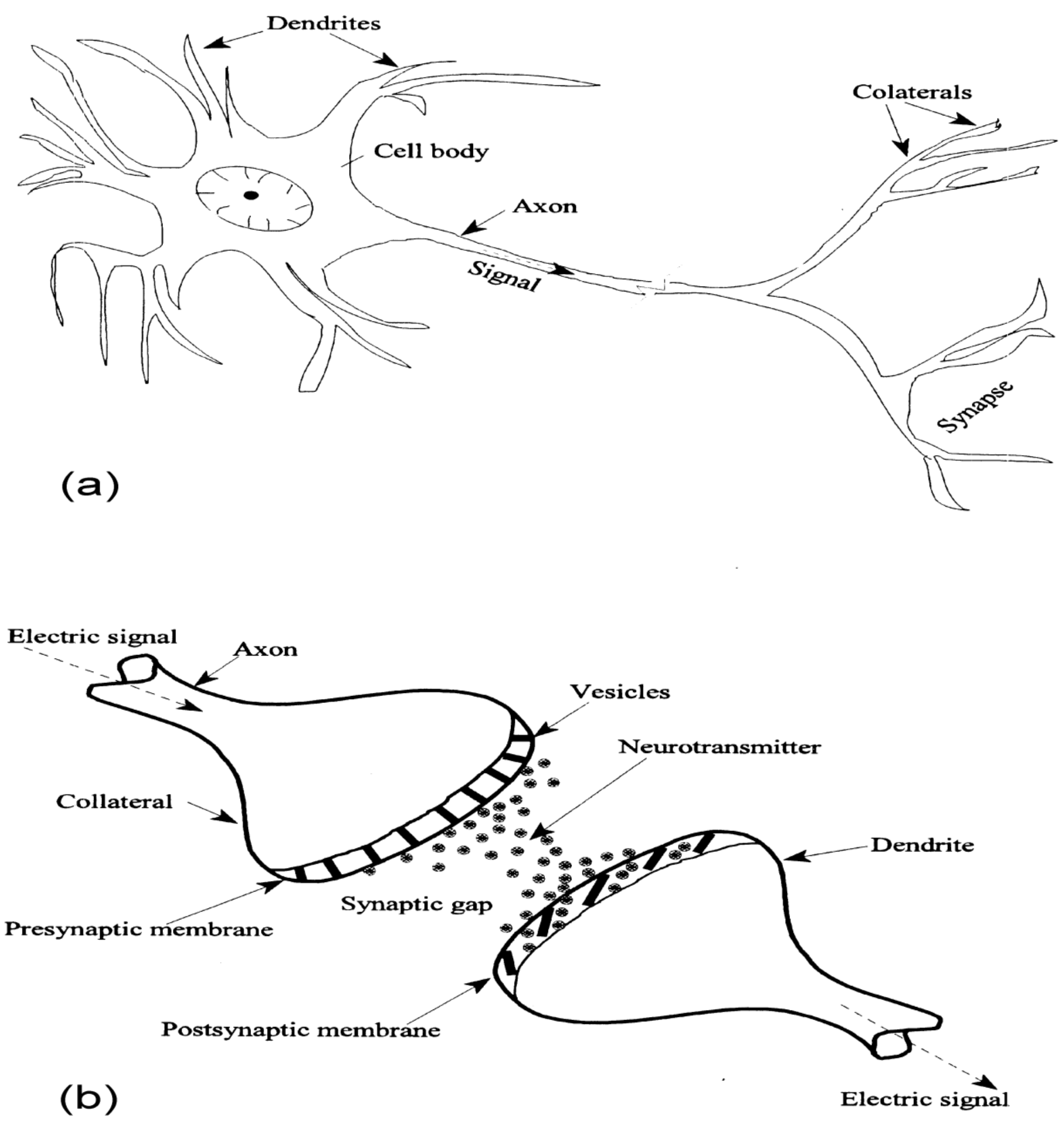


Figure 1.1: Structure of single neuron in the brain.

1.4. <u>An analogy to the biological neuron</u>:

The human brain is a highly complex structure with massive interconnected network of neurons. The characteristics of the neuron can be captured by a simple model. Each component of the model bears a direct analogy to the actual component of biological neuron. So this is called artificial neural network. In nervous system all neurons receive the inputs through dendrites and axons and produce the output if sum of the inputs is greater than some threshold value. Then the input signals pass on through synapse which may accelerate or decelerate the arriving signal. This acceleration or deceleration is modelled by weights. The synapse which transmits stronger signal will have larger weight and weak synapses will have smaller weights.

Table 1.1 Analogy between Biological and Artificial neurons.

Biological terminology	**ANN Terminology**
Neuron	Node/Processing Element
Dendrites	Combining Functions
Synapse	Weight
Axon	Output element

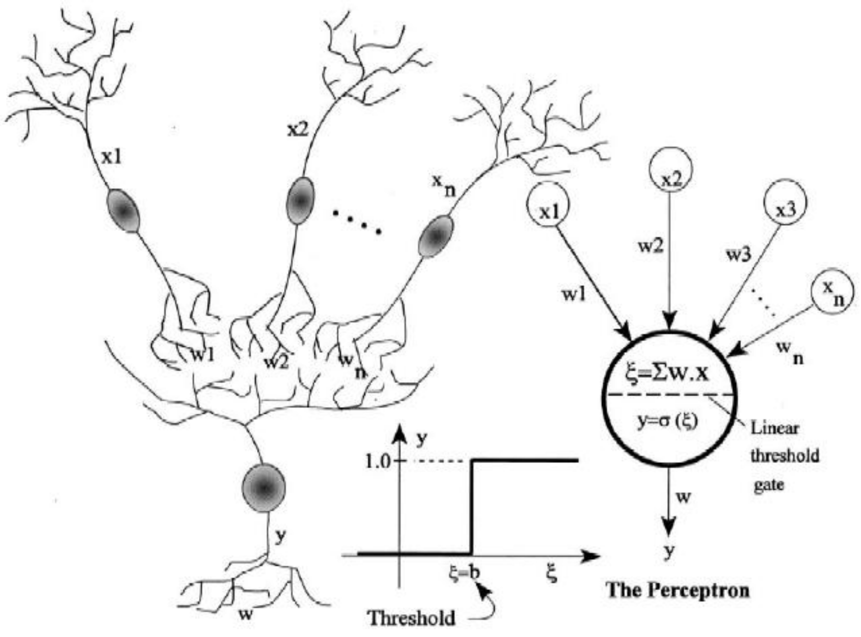


Fig 1.2: Signals from *n* neurons in brain and analogy to signal summing in an artificial neuron.

1.5. Artificial neural network:

Artificial Neural Networks are artificial adaptive systems which are inspired by the functioning processes of the human brain. They are systems that are able to modify their internal structure in relation to a functional objective. They are particularly suited for solving problems of the nonlinear type and for optimal solution for the problems.

Let $x_1, x_2, \dots, x_n$ are the n inputs in artificial neurons and $w_1, w_2, \dots, w_n$ are the corresponding weights.

Let I= $\sum_{i=1}^{n} x_i w_i$.

If I is greater than the threshold value, some function of I will be taken. I is called activation function.

i.e. $y = \phi(I)$.

1.5.1. **Terminologies:**

(a) **Node:** Node is the artificial neuron in ANN. There are three types of nodes, namely, input node, hidden node, output node.

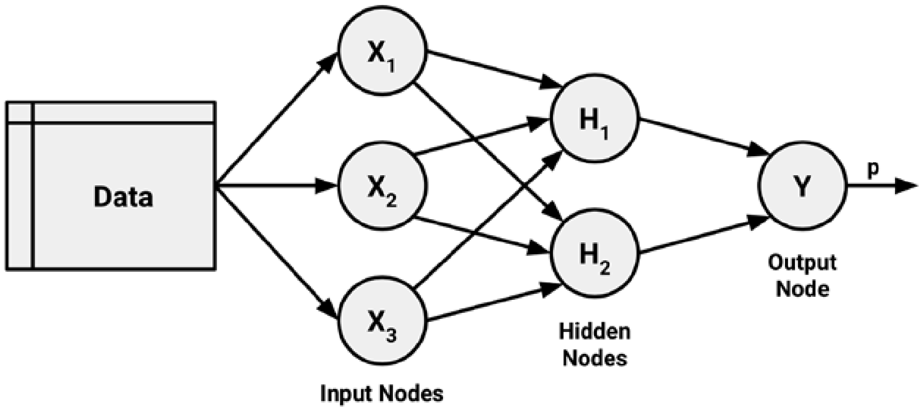


Fig 1.3: Types of nodes.

(b) **Activation function**: The activation function is a function by which the artificial neurons accept the incoming signals and possesses it to a single output signal. In biological neural network if the summation of all input signals crosses one threshold value then only neuron passes the signal otherwise not. Similarly in ANN this is called threshold activation function, which results in an output signal only once a specified input threshold has been attained. The summation value of weighted inputs is called activation value.

Types of activation function:

- **Threshold function**: In this function the sum of weighted inputs is compared with a threshold value θ. If the value of I is greater than θ then output is 1 else it is 0. Y = ϕ($\sum_{i=1}^{n} x_i w_i$ –θ). Its shape resembles stairs, and hence, is sometimes referred to as a **unit step activation function** also.

  Where ϕ is a step function such that,

$$\phi(I) = \begin{cases} 1, & I > 0 \\ 0, & I \leq 0 \end{cases}$$

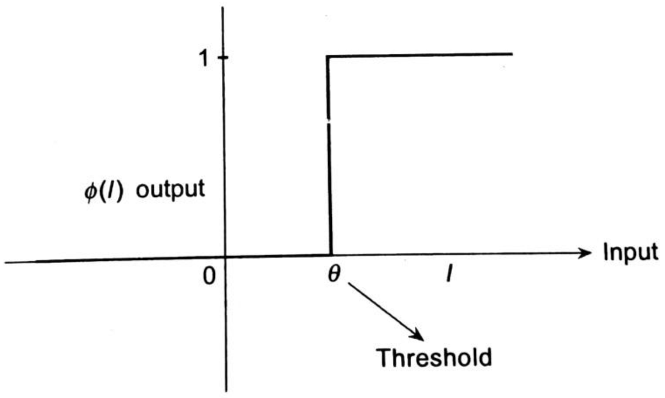


Fig 1.4: Threshold function

- Although the threshold function is interesting due to its parallels with biology, it is rarely used in artificial neural networks. Freed from the limitations of biochemistry, the ANN activation functions can be chosen based on their ability to demonstrate desirable mathematical characteristics and accurately model relationships among data.
- **<u>Signum Function</u>:**

  This function is known as Quantizer function.

  Here the function ϕ is defined as

$$\phi(I) = \begin{cases} 1, & I > \theta \\ -1, & I \leq \theta \end{cases}$$

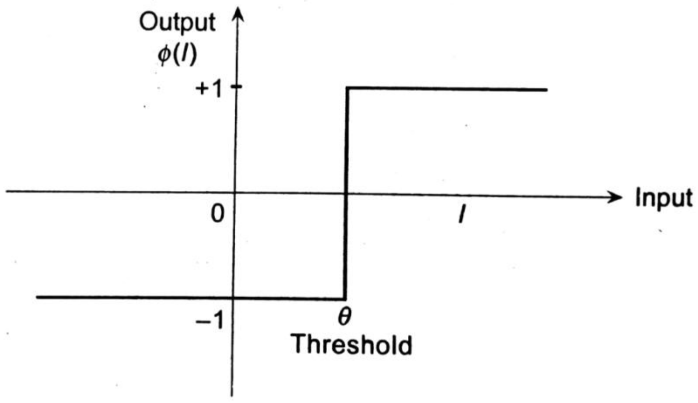


Fig 1.5: Signum Function

➢ **<u>Sigmoid function</u>:**

The most commonly used alternative is the **sigmoid activation function** (the *logistic* sigmoid) . Although it shares a similar step or "S" shape with the threshold activation function, the output signal is no longer binary; output values can fall anywhere in the range from 0 to 1or -1 and +1. This is given by,

$$\phi(\mathrm{I}) = \frac{1}{1+e^{-\alpha I}},$$

where α is the slope parameter, which adjusts the abruptness of the function as it changes between the two asymptotic values. This function is differentiable, i.e., it is possible to calculate the derivative across the entire range of inputs.

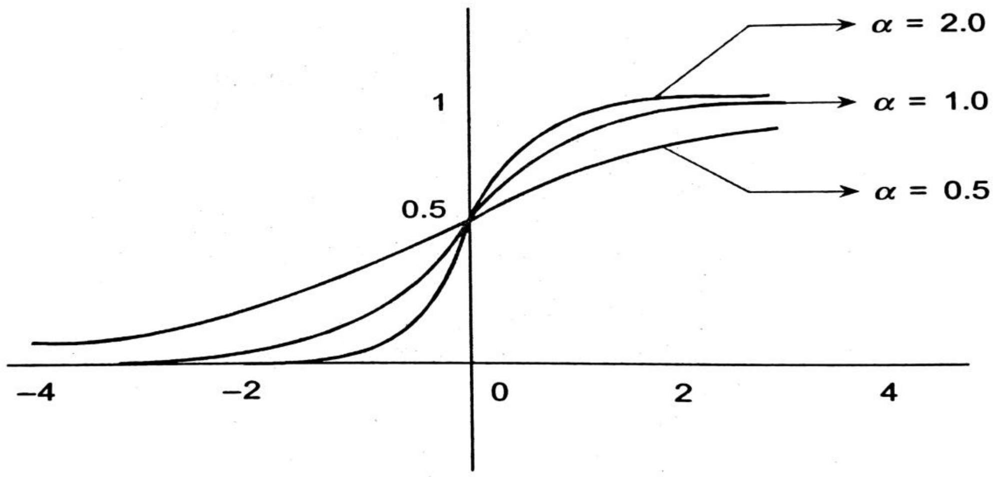


Fig 1.6: Sigmoid Function

Although sigmoid is perhaps the most commonly used activation function and is often used by default, some neural network algorithms allow a choice of alternatives. A selection of such activation functions is shown in the following figure:

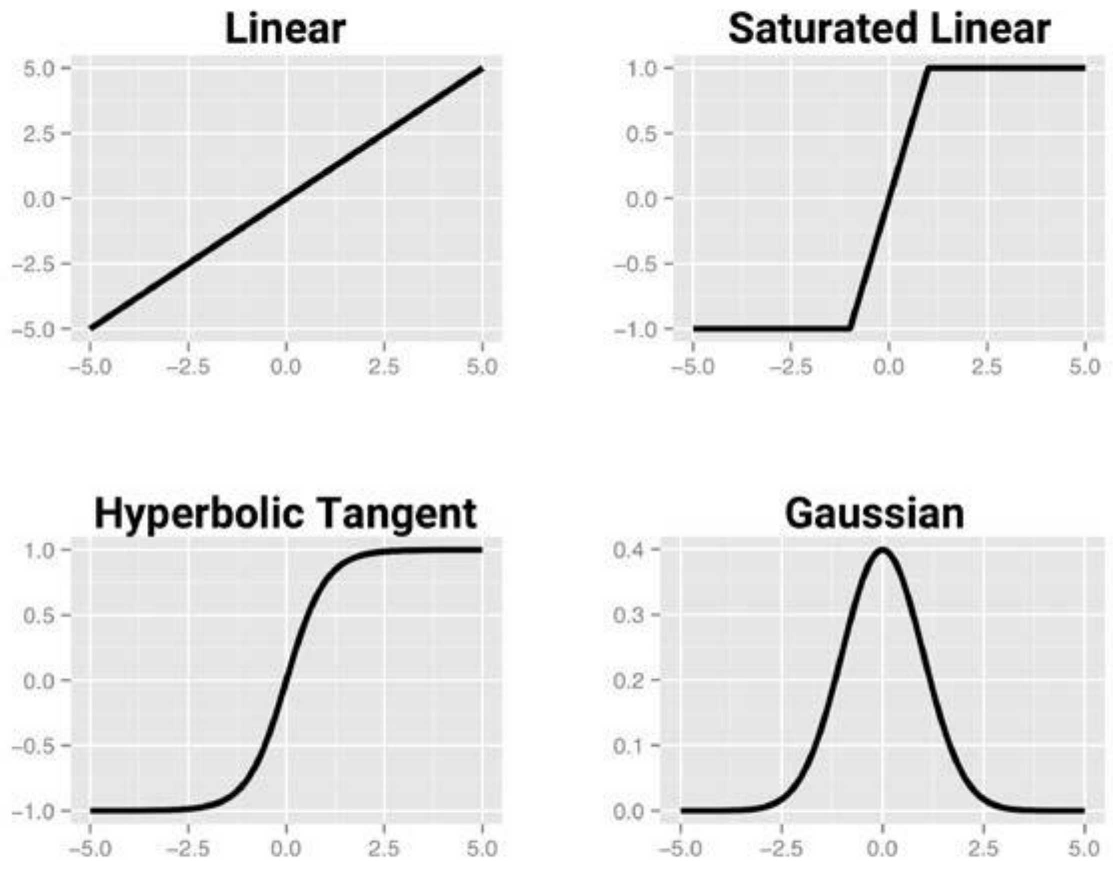


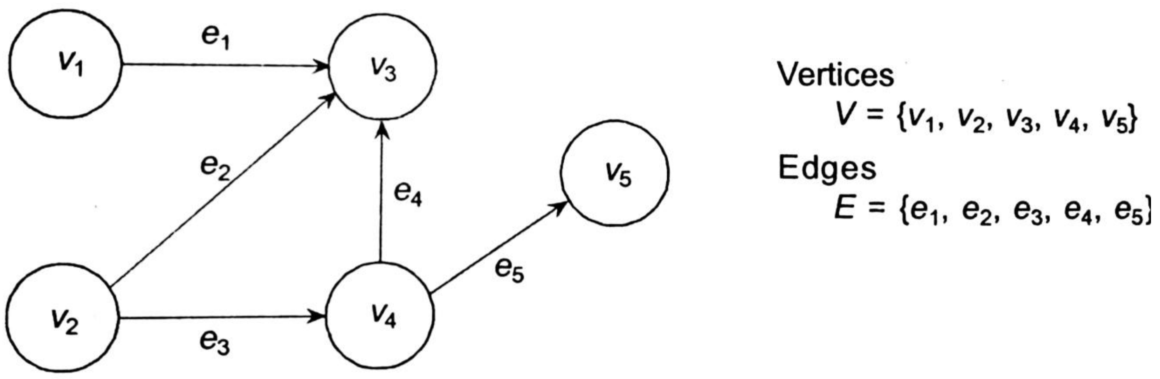
Fig 1.7: various types of activation function

Note: The choice of activation function is a very tough task because it biases the neural network such that it may fit certain types of data more appropriately, allowing the construction of specialized neural networks. For example a linear activation function results in ANN very similar to a linear regression model; while a Gaussian activation function results in a model called a Radial basis function (RBF) network. Each of these has its own strength better suited for certain learning tasks and not others.

(c) Neural Network Architectures:

An artificial Neural Network is defined as a data processing system consisting of a large number of simple highly interconnected processing elements. Generally an ANN can be represented using a direct graph. A graph G is an ordered 2-tuple (V, E) consisting of a set

V of vertices and a set E of edges. When each edge is assigned an orientation then the graph is directed and is called a direct graph or a digraph. The vertices of the graph may represent neurons, either input or output and the edge is the synaptic links. The edges are labelled by weights attached to the synaptic links.



v_1
e_1
v_3
e_2
e_4
v_5
e_5
v_2
v_4
e_3
Vertices
$V = \{v_1, v_2, v_3, v_4, v_5\}$
Edges
$E = \{e_1, e_2, e_3, e_4, e_5\}$

Fig 1.8: Vertices and edges in ANN

There are several classes of ANN classified according to their learning mechanisms. However there are fundamentally three different classes of networks.

- ***Single layer Feed-forward Network*** **:**
  This is the simplest network. It contains only two layers-input layer and output layer. The input layer neurons receive the input signals and the output layer neurons receive the output signals. The synaptic links carrying the weights connect every input neuron to the output neuron but on vice-versa. This type of network is called feed forward in type. Although there are two layers but for computation only output layer works so it is termed as single layer. The input layer only transits the signals to the output layer. Hence it is called single layer feed forward network. It can be used for basic pattern classification, particularly for patterns that are linearly separable.

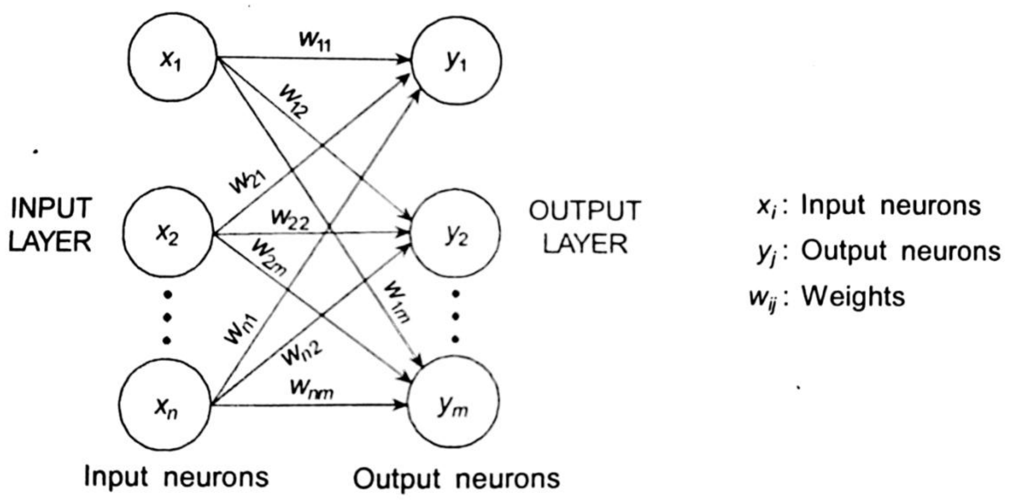


Fig 1.9: Depicting input, output layer and weights in single layer feed forward network.

- ***Multilayer Feed-forward network:***

  This network is made up of multiple layers. Besides possessing an input and an output layer this architecture has another layer called hidden layer. The computational units of hidden layer are known as hidden neurons or hidden units. The hidden layer helps in performing useful intermediary computations before detecting the input to output layer. The input layer neurons are linked to the hidden layer neurons and the weights of these links are referred to as input-hidden layer weights. Similarly the hidden layer neurons are linked with output layer neurons and the corresponding weights are called hidden-output layer weights. In this network more than one hidden layer can be present. For example, if there are l neurons in input layer,m_1 neurons in 1st idden layer, m_2 neurons in 2nd hidden layers (m_1+m_2 =m) and n number of neurons in output layer then this is called (i-m_1-m_2-n) configuration.

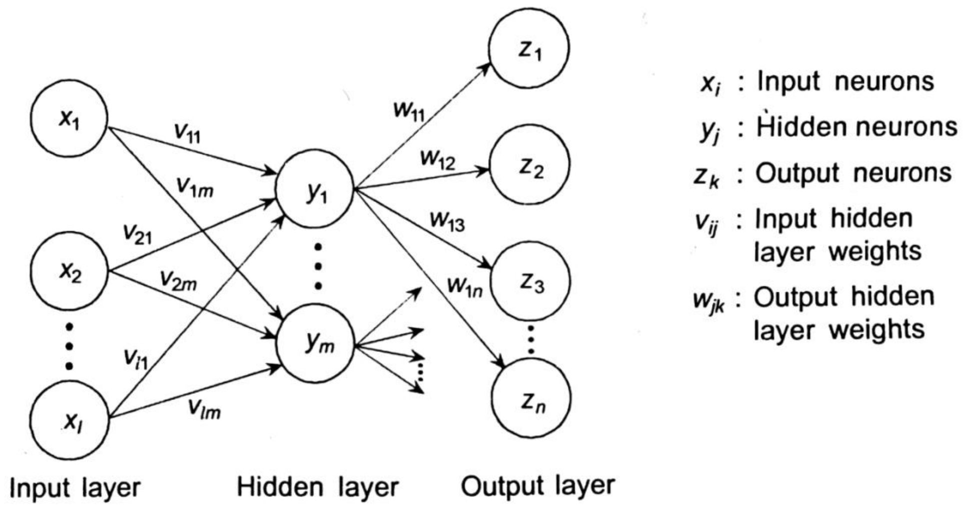


Fig 1.10: Multi-layer feed forward network.

➢ ***Recurrent Network*** *:*

This network is different from feed forward network because there is at least one feedback loop. Thus in this network there could exist one layer with feedback connections. There could also be neurons with self-feedback links i.e. in other words it can be said that the output of a neuron is feedback into itself as input.

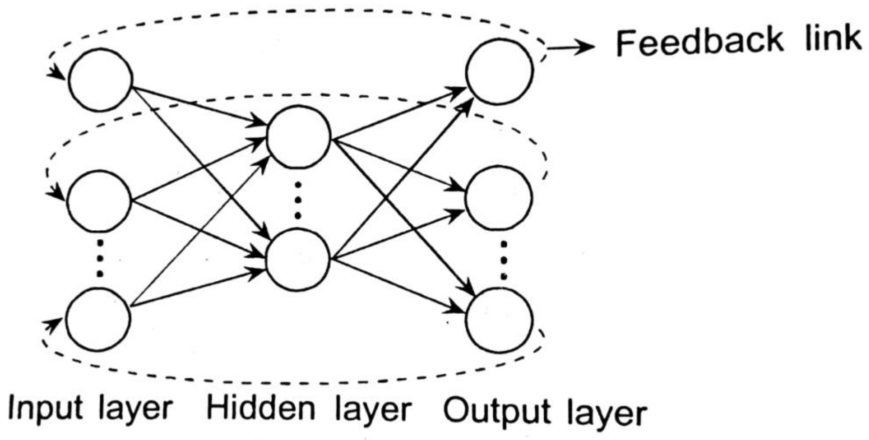


Fig 1.11 Recurrent network

(d) <u>Training the Network</u>:

The network architecture is a blank slate in that it can't learn itself. It must be trained with experience like a child's brain. A baby's brain develops by experience in environment. Similarly the neural network processes the input data then the connection between the neurons are going strong or weak. For training the data there are so many methods in ANN. Back-propagation is one of them.

1.5.2. Back-propagation

Back-propagation was invented in the 1970s as a general optimization method for performing automatic differentiation of complex nested functions. However, it wasn't until 1986, with the publishing of a paper by Rumelhart, Hinton, and Williams, titled "Learning Representations by Back-Propagating Errors," that the importance of the algorithm was appreciated by the machine learning community at large. Researchers had long been interested in finding a way to train multilayer artificial neural networks that could automatically discover good "internal representations," i.e. features that make learning easier and more accurate. Features can be thought of as the stereotypical input to a specific node that activates that node .Since a node's activation is dependent on its incoming weights and bias, researchers say a node has learned a feature if its weights and bias cause that node to activate when the feature is present in its input. By the 1980s, hand-engineering features had become the standard in many fields, especially in computer vision, since experts knew from experiments which features (e.g. lines, circles, edges, blobs in computer vision) made learning simpler. Back-propagation was one of the first methods able to demonstrate that artificial neural networks could learn good internal representations, i.e. their hidden layers learned nontrivial features. Experts examining multilayer feed-forward networks trained using back-propagation actually found that many nodes learned features similar to those designed by human experts and those found by neuroscientists investigating biological neural networks in mammalian brains. Even more importantly, because of the efficiency of the algorithm and the fact that domain experts were no longer required to discover appropriate features, backpropagation allowed artificial neural networks to be applied to a much wider field of problems that were previously off-limits due to time and cost constraints.

1.5.3. Resilient Back-propagation

With and without weight back-tracking :

The Algorithm RBP introduced by M. Riedmiller in 1993 is a local adaptive learning scheme, performing supervised batch learning in feed forward neural networks. The basic principlel of resilient back-propagation is to eliminate the harmful influence of size of partial derivatives on the weight step. As a consequence, only the sign of the derivative is considered to indicate the direction of the weight update.

1.6. Objectives of the project

1) The primary objective is to build a robust classification model for classifying breast cancer cells into benign and malignant cells using Artificial Neural Networks.
2) The secondary objective is to perform a comparative analysis of the three stated ANN algorithms via a case study.

Chapter 2: Methodology

2.1. What is Breast Cancer?

Breast cancer is the most common form of cancer and the second most common cause of cancer deaths among women in the world; the disease affects approximately 10% of all women at some stage of their life in the Western world. Breast cancer may be detected via a careful study of clinical history, physical examination, and imaging with either mammography or ultrasound. However, definitive diagnosis of a breast mass can only be established through fine needle aspiration (FNA) biopsy, core needle biopsy, or excision biopsy. Among these methods, FNA is the easiest and fastest method of obtaining a breast biopsy, and is effective for women who have fluid-filled cysts. Research works on the Wisconsin Diagnosis Breast Cancer (WDBC) data grew out of the desire to diagnose breast masses accurately based solely on FNA. To improve the accuracy and efficiency of the detection of breast cancer, a number of research projects are focusing on developing methods for Computer Aided Diagnosis (CAD) of breast cancer from FNA, including works on image analysis and computational intelligence.

2.2. Data description

The Wisconsin Diagnostic Breast Cancer (WDBC) dataset consist of 569 instances (357 benign – 212 malignant), where each one represents FNA test measurements for one diagnosis case. For this dataset each instance has 32 attributes, where the first two attributes correspond to a unique identification number and the diagnosis status (benign / malignant). The rest 30 features are computations for ten real-valued features, along with their mean, standard error and the mean of the three largest values ("worst" value) for each cell nucleus respectively. These ten real values, which are are computed from a digitized image of a fine needle aspirate (FNA) of breast tumour, describing characteristics of the cell nuclei present in the image and are recorded with four significant digits. The data-set is available presently at UCI machine learning repository.

2.2.1. Benign and Malignant cells

If the cells of the tumour are not cancerous then that is called *benign*. It will not spread to the other parts of the body. So it is less harmful unless it is pressing on the nearer tissues. If the tumour is made of cancer cells then it is *malignant*. It can invade the surrounding tissues. The cancerous cells are growing out of control and capable of metastasizing. Metastasization simply means that the cells of the tumour are able to leave the original tumour and travel to other parts of the body.

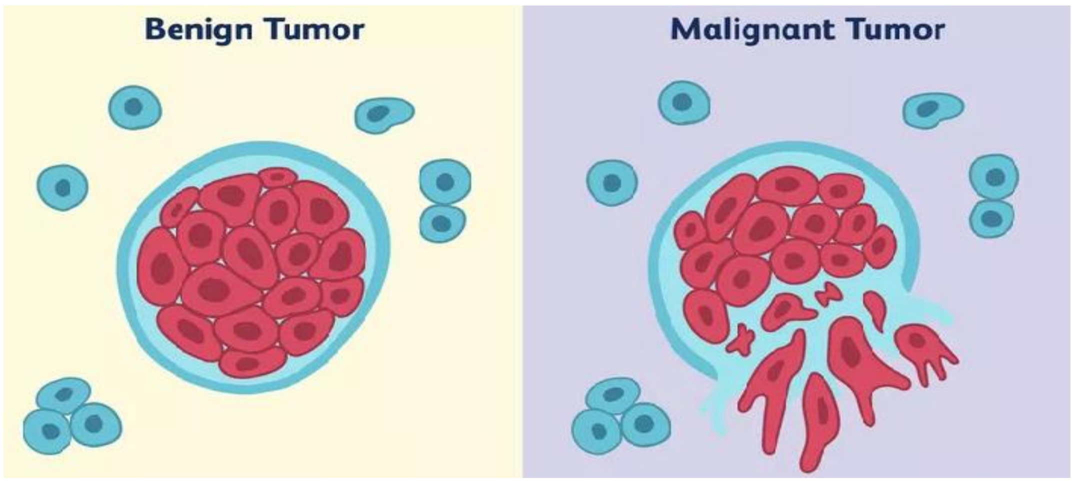


Fig 2.1: Benign and Malignant cells

2.2.2. <u>Variable description</u>

1) **Radius:** It is computed by averaging the length of radial line segments from the centre of mass of the boundary to each of the boundary points.
2) **Perimeter:** It is measured as the sum of the distances between consecutive boundary points.
3) **Area:** It is measured by counting the number of pixels on the interior of the boundary and adding one-half of the pixels on the perimeter, to correct for the error caused by digitization.
4) **Compactness:** It combines the perimeter and area to give a measure of the compactness of the cell.

$$\mathbf{Compactness} = \frac{\mathbf{Perimeter^2}}{\mathbf{Area}}$$

5) **Smoothness**: It is quantified by measuring the difference between the length of each radial line and the mean length of the two radial lines surrounding it,

$$\mathbf{Smoothness} = \frac{\sum_{\mathbf{Points}} |\mathbf{r_i} - \frac{\mathrm{r_i} + \mathrm{r_{i+1}}}{2}|}{\mathbf{perimeter}}$$

Where r_i is the length of the line from the centre of mass of the boundary to each boundary point.

6) **Concavity:** It is captured by measuring the size of any indentations in the boundary of the cell nucleus.

7) **Concave points:** It is similar to concavity, but counts only the number of boundary points lying on the concave regions of the boundary, rather than the magnitude of such concavities.

8) **Symmetry:** It is measured by finding the relative difference in length between pairs of line segments perpendicular to the major axis of the contour of the cell nucleus.

$$\mathbf{Symmetry} = \frac{\sum_{\mathbf{i}} |\mathbf{left_i} - \mathbf{right_i}|}{\sum_{\mathbf{i}}(\mathbf{left_i} + \mathbf{right_i}}$$

where left_i and right_i denote the lengths of perpendicular segments on the left and right of the major axis, respectively.

9) **Fractal dimension:** It is approximated using the "coastline approximation" described by Mandelbrot. The perimeter of the nucleus is measured using increasingly larger "rulers". As the ruler size increases, the precision of the measurement decreases, and the observed perimeter decreases. Plotting these values on a log-log scale and measuring the downward slope gives the negative of an approximation to the fractal dimension.

10) **Texture**: It is measured by finding the variance of the gray-scale intensities in the component pixels.

The mean value, standard error, and the extreme (largest or "worst") value of each characteristic were computed for each image, which resulted in 30 features of 569 images, yielding a database of 569×30 samples. 80% of the total sample was randomly selected for training and the remaining 20% of the sample for test but, in training and testing data set the ratio of malignant and benign is same.

2.3. The Back-propagation algorithm

Back-propagation algorithm iterates through many cycles of two processes. Each cycle is known as an epoch. The network does not contain a priori knowledge, the starting weights are random. Until a stopping criterion is reached, the algorithm iterates through the processes. Each epoch in the back-propagation algorithm includes:

- *Forward phase*: In this phase the neurons are activated in sequence from the input layer to the output layer applying each neuron's weights and activation function along the way. After reaching the final layer, an output signal is produced.
- *Backward phase*: In this phase the network's output signal resulting from the forward phase is compared to the true target value in the training data. The difference between the network's output signal and the true value results in an error that is propagated backwards in the network to modify the connection weights between neurons and reduce future errors. Then the network uses the information sent backward to reduce the total error of the network.

But the relationship between each neuron's inputs and outputs is complex, so we have to derive how much the weight should change. For this gradient descent technique is used in this algorithm. Then back-propagation algorithm uses the derivative of each neuron's activation function to identify the gradient in the direction of each of the incoming weights—hence the importance of having a differentiable activation function. The gradient suggests how steeply the error will be reduced or increased for a change in the weight. The algorithm will attempt to change the weights that result in the greatest reduction in error by an amount known as the learning rate. The greater the learning rate, the faster the algorithm will attempt to descend down the gradients, which could reduce the training time at the risk of overshooting the valley.

2.3.1. Strengths

1) Back-propagation algorithm can be adapted to classification or numeric prediction problems.

2) It is capable of modelling more complex patterns than nearly any algorithm.

3) It makes few assumptions about the data's underlying relationships.

2.3.2. <u>Limitations</u>

1) This algorithm extremely computationally intensive and slow to train, particularly if the network topology is complex.

2) It is very prone to over-fit training data.

3) It results in a complex black box model that is difficult, if not impossible, to interpret.

2.3.3. <u>Mathematical basis of Back-propagation algorithm</u>

Let for K-class classification there are k nodes in the output layer, with kth node modelling the probability of class k. There are K target measurements i.e. $\mathrm{Y_k, k = 1(1)K}$ each being coded as 0 to 1 variable for $\mathrm{K^{th}}$ class. Let $\mathrm{Z_m}$ be the linear combination of the inputs, then the target variable Y_k is modelled as a function of linear combination of the $\mathrm{Z_m}$. Where X is the input vectors comtaining m input elements.

$$\begin{aligned} \mathrm{Z_m} &= \sigma(\alpha_{0m} + \alpha_m{}^T X), \qquad m = 1, 2, \dots, M \\ \mathrm{T_k} &= \beta_{0k} + \beta_k{}^T Z, \qquad k = 1, 2, \dots, K \qquad \dots(1) \\ \mathrm{F_k(X)} &= \mathrm{G_k(T)}, \qquad k = 1, 2, \dots, K \end{aligned}$$

Where $\mathrm{Z = (Z_1, Z_2, \dots, Z_m)}$ and $\mathrm{T = (T_1, T_2, \dots, T_k)}$

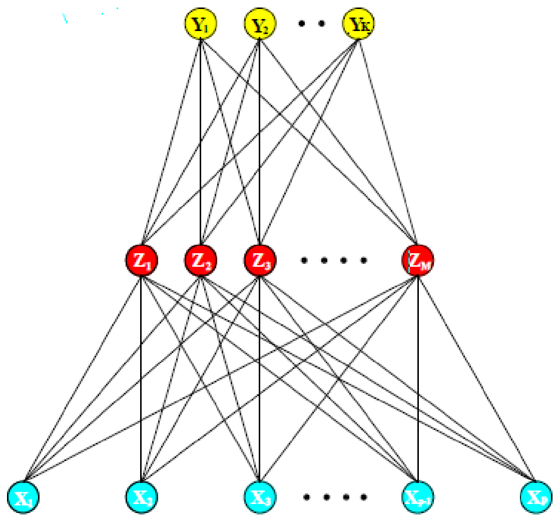


Fig 2.2: Single layer feed-forward neural network.

The activation function σ(v) is usually chosen to be the sigmoid

$$\sigma(v) = \frac{1}{1 + e^{-v}}$$

Neural network diagrams are sometimes drawn with an additional bias unit feeding into every unit in the hidden and output layers. Thinking of the constant "1" as an additional input feature, this bias unit captures the intercepts α_{0m} and β_{0k} in model. The output function $g_k(t)$ allows a final transformation of the vector of output T. For regression the identity function $g_k(t) = T_k$ can be used and for K-class classification also the identity function can be used but in maximum cases the soft-max function is more useful.

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{k} e^{T_l}} \qquad \text{... (2)}$$

2.3.4. **Fitting of neural networks**:

Now we have to fit the unknown parameters (weight).Here the total unknown parameter set is denoted by θ. It consists of

$\{ \alpha_{0m}, \alpha_m; m = 1, 2, \dots, M \}$ ... (3)

$\{\beta_{0k}, \beta_k; k = 1, 2, \dots, K\}$

For regression problem sum of squares can be used as a measure of fit.

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2 \qquad \text{... (4)}$$

For classification problem cross entropy (deviance) can be used

$$R(\theta) = -\sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} \log f_k(x_i) \qquad \text{... (5)}$$

Here the corresponding classifier is $G(x) = \text{argmax}_k f_k(x)$.

With the soft-max activation function and the cross-entropy error function, the neural network model is exactly a linear logistic regression model in the hidden units, and all the parameters are estimated by maximum likelihood. In this method we don't want the global minimum of R(θ), as that will over-fit the solution. This is achieved directly through a penalty term, or indirectly by early stopping. The approach to minimizing R(θ) is by gradient descent, called back-propagation in this situation. Because of the compositional form of the model, the gradient can be easily derived using the chain rule of differentiation.

Let $Z_{mi} = \sigma(\alpha_{0m} + \alpha_m{}^T X_i)$ where $Z_{mi} = (Z_{1i}, Z_{2i}, \dots, Z_{Mi})$

and $R(\theta)= \begin{cases} \sum_{k=1}^{K}\sum_{i=1}^{N}(y_{ik}-f_k(x_i))^2, & \text{for regression} \\ -\sum_{k=1}^{K}\sum_{i=1}^{N}y_{ik}\log f_k(x_i), & \text{for classification} \end{cases}$ ...(6)

With derivatives,

For regression,

$\frac{dR_i}{d\beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi}$,

$\frac{dR_i}{d\alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\,\sigma'(\alpha_m^T x_i)x_{il}$,

For classification,

$\frac{dR_i}{d\beta_{km}} = -\frac{y_{ik}}{f_k(x_i)}g_k'(\beta_k^T z_i)z_{mi}$ .... (7)

$$\frac{dR_i}{d\alpha_{ml}} = -\sum_{k=1}^{K}\frac{y_{ik}}{f_k(x_i)}g_k'(\beta_k^T z_i)\beta_{km}\,\sigma'(\alpha_m^T x_i)x_{il}$$

Given this derivatives gradient descent update at the (r + 1)st iteration has the form

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \delta_r \sum_{i=1}^{N}\frac{dR_i}{d\beta_{km}^{(r)}}$$

$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \delta_r \sum_{i=1}^{N}\frac{dR_i}{d\alpha_{ml}^{(r)}}$$

... (8)

Where $-\delta_r$ is the learning rate.Now it can be written as:

$$\frac{dR_i}{d\beta_{km}} = \delta_{ki}\, z_{mi}$$

$$\frac{dR_i}{d\alpha_{ml}} = s_{mi}\, x_{il}$$

...(9)

Here δ_{ki} and s_{mi} are "errors" from the current model at the output and hidden layer units, respectively. From their definitions, these errors satisfy

$s_{mi} = \sigma'(\alpha_m^T x_i)\sum_{k=1}^{K}\beta_{km}\,\delta_{ki}$ ... (10)

It is known as back propagation equation. Using this, the updates in (8) can be implemented with a two-pass algorithm. In the forward pass, the current weights are fixed and the predicted values $f_k(x_i)$ are computed from formula (1). In the backward pass, the errors δ_{ki} are computed, and then back-propagated via (10) to give the errors s_{mi}. Both sets of errors are then used to compute the gradients for the updates in (8), via (9). This two-pass procedure is known as back-propagation. The computational components for cross-entropy have the same form as those for the sum of squares error function.

2.4. Resilient Back Propagation Algorithm

2.4.1. Without weight back-tracking

The Algorithm RBP introduced by M. Riedmiller in 1993 is a local adaptive learning scheme, performing supervised batch learning in feed forward neural networks. The basic principal of resilient back-propagation is to eliminate the harmful influence of size of partial derivatives on the weight step. As a consequence, only the sign of the derivative is considered to indicate the direction of the weight update. To achieve this, we introduce for each ω_{ij}, its individual update value, $\Delta_{ij}(t)$, which solely determines the size of the weight-update. It introduces a second learning rule, which determines the evolution of the size update value $\Delta_{ij}(t)$. The estimation is based on the observed behaviour of the partial-derivative during two successive weight-steps:

$$\Delta_{ij}(t) = \begin{cases} \eta^{+}.\Delta_{ij}(t-1), & \text{if } \frac{\partial E}{\partial \omega_{ij}}(t).\frac{\partial E}{\partial \omega_{ij}}(t-1) > 0 \\ \eta^{-}.\Delta_{ij}(t-1), & \text{if } \frac{\partial E}{\partial \omega_{ij}}(t).\frac{\partial E}{\partial \omega_{ij}}(t-1) < 0 \\ \Delta_{ij}(t-1), & \text{Otherwise} \end{cases}$$

where, $0 < \eta^{-} < 1 < \eta^{+}$.

In words, the adaptation rule works as follows. Every time the partial derivative of the corresponding weight ω_{ij} changes its sign, which indicates that the last update was too big and the algorithm has jumped over a local minimum, the update value $\Delta_{ij}(t)$ is decreased by the factor η^{-} . If the derivative retains its sign, then the update-value is slightly increased in order to accelerate convergence in shallow regions.

Once the update value for each weight is adapted, the weight-update in itself follows a simple rule: If the derivative is positive (increasing error), the weight is decreased by its update value. If the derivative is negative, the weight is updated as:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

where,

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t), & \frac{\partial E}{\partial \omega_{ij}}(t) > 0 \\ \Delta_{ij}(t), & \frac{\partial E}{\partial \omega_{ij}}(t) < 0 \\ 0, & \text{else} \end{cases} \quad \forall\, i, j$$

2.4.2. With weight back tracking

If the partial derivative changes, i. e, the previous step was too large and the minimum was missed, the previous weight update is reverted.

$$w_{ij}(t) = -w_{ij}(t-1)$$

$$\text{if } \frac{\partial E}{\partial \omega_{ij}}(t).\frac{\partial E}{\partial \omega_{ij}}(t-1) < 0$$

Due to the 'back-tracking' weight stage, the derivative is supposed to change its sign once again in the following step. To avoid double punishment we set :

$$\frac{\partial E}{\partial \omega_{ij}}(t-1) = 0$$

2.5. k-fold Cross Validation

K-fold cross-validation uses part of the available data to fit the model, and a different part to test it. We split the data into K roughly equal-sized parts; for example, when K = 5, the scenario looks like this:

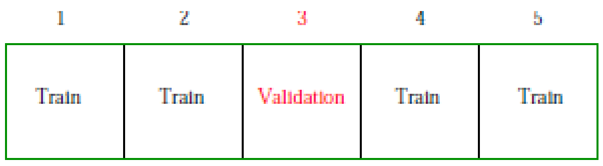


Fig 2.3: Splitting data into training and validation sets.

For the kth part (third above), we fit the model to the other K −1 parts of the data, and calculate the prediction error of the fitted model when predicting the kth part of the data. We do this for k = 1, 2, . . . ,K and combine the K estimates of prediction error. Here are more details. Let L : {1, . . . ,N} → {1, . . . ,K} be an indexing function that indicates the partition to which observation i is allocated by the randomization. Denote by $\hat{f}^{-k}(x)$ the fitted function, computed with the kth part of the data removed. Then the cross-validation estimate of prediction error is

$$\mathrm{C.V.}(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N} L(y_i,\ \hat{f}^{-k(i)}(x_i))$$

Typical choices of K are 5 or 10 (see below). The case K = N is known as leave-one-out cross-validation. In this case k(i) = i, and for the $it\ \ h$ observation the fit is computed using all the data except the ith.

Given a set of models f(x, α) indexed by a tuning parameter α, denote by $\hat{f}^{-k}(x, \alpha)$ the αth model fit with the kth part of the data removed. Then for this set of models we define,

$$\mathrm{C.V.}(\hat{f}, \alpha) = \frac{1}{N}\sum_{i=1}^{N} L(y_i,\ \hat{f}^{-k(i)}(x_i, \alpha))$$

The function $\mathrm{C.V.}(\hat{f},\ \alpha)$ provides an estimate of the test error curve, and we find the tuning parameter $\hat{\alpha}$ that minimizes it. Our final chosen model is $f(x, \hat{\alpha})$ which we then fit to all the data.

Chapter 3: Analysis

3.1. Importing and cleaning:

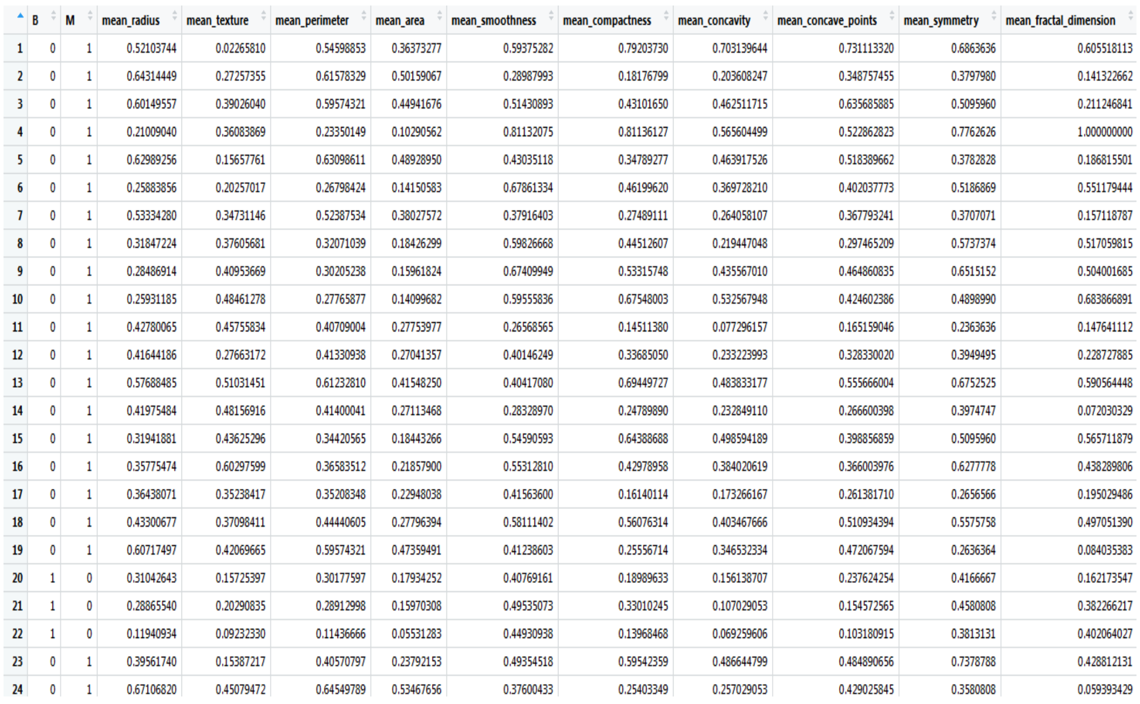
The data is imported directly into R from UCI Machine learning repository followed by pre-processing or cleaning of the data. We use the following piece of R code to do so:

```
data=read.table(file="https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data", header = FALSE, sep = ",")
```

The first column consists of patient I. Ds which we remove. Further we name the columns of this table according to the data descriptions. Then we express the diagnosis column as an expansion of two variables 'B' and 'M', such that a '1' appears where the cell is benign and a '0' appears where the cell is malignant. Further we normalize the features to ensure that the neural networks works well.

```
data=data[,-1]

n=c("diag", "mean_radius", "mean_texture", "mean_perimeter", "mean_area", "mean_smoothness", "mean_compactness", "mean_concavity", "mean_concave_points", "mean_symmetry", "mean_fractal_dimension", "radius_SE", "texture_SE", "perimeter_SE", "area_SE", "smoothness_SE", "compactness_SE", "concavity_SE", "concave_points_SE", "symmetry_SE", "fractal_dimension_SE", "worst_radius", "worst_texture","worst_perimeter","worst_area","worst_smoothness","worst_compactness", "worst_concavity", "worst_concave_points","worst_symmetry","worst_fractal_dimension")

colnames(data) = n

code=class.ind(data$diag)

data1=as.data.frame(cbind(code, sapply(data[,2:31],

function(x){(x-min(x))/(max(x)-min(x))})))
```

```
n1=c("B", "M", "diag", "mean_radius", "mean_texture", "mean_perimeter", "mean_area", "mean_smoothness", "mean_compactness", "mean_concavity", "mean_concave_points", "mean_symmetry", "mean_fractal_dimension", "radius_SE", "texture_SE", "perimeter_SE", "area_SE", "smoothness_SE", "compactness_SE", "concavity_SE", "concave_points_SE", "symmetry_SE", "fractal_dimension_SE", "worst_radius", "worst_texture", "worst_perimeter", "worst_area", "worst_smoothness", "worst_compactness", "worst_concavity", "worst_concave_points", "worst_symmetry",
"worst_fractal_dimension")

head(data1)

colnames(data1)=n1
```



	B	M	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	mean_compactness	mean_concavity	mean_concave_points	mean_symmetry	mean_fractal_dimension
1	0	1	0.52103744	0.02265810	0.54598853	0.36373277	0.59375282	0.79203730	0.703139644	0.731113320	0.6863636	0.605518113
2	0	1	0.64314449	0.27257355	0.61578329	0.50159067	0.28987993	0.18176799	0.203608247	0.348757455	0.3797980	0.141322662
3	0	1	0.60149557	0.39026040	0.59574321	0.44941676	0.51430893	0.43101650	0.462511715	0.635685885	0.5095960	0.211246841
4	0	1	0.21009040	0.36083869	0.23350149	0.10290562	0.81132075	0.81136127	0.565604499	0.522862823	0.7762626	1.000000000
5	0	1	0.62989256	0.15657761	0.63098611	0.48928950	0.43035118	0.34789277	0.463917526	0.518389662	0.3782828	0.186815501
6	0	1	0.25883856	0.20257017	0.26798424	0.14150583	0.67861334	0.46199620	0.369728210	0.402037773	0.5186869	0.551179444
7	0	1	0.53334280	0.34731146	0.52387534	0.38027572	0.37916403	0.27489111	0.264058107	0.367793241	0.3707071	0.157118787
8	0	1	0.31847224	0.37605681	0.32071039	0.18426299	0.59826668	0.44512607	0.219447048	0.297465209	0.5737374	0.517059815
9	0	1	0.28486914	0.40953669	0.30205238	0.15961824	0.67409949	0.53315748	0.435567010	0.464860835	0.6515152	0.504001685
10	0	1	0.25931185	0.48461278	0.27765877	0.14099682	0.59555836	0.67548003	0.532567948	0.424602386	0.4898990	0.683866891
11	0	1	0.42780065	0.45755834	0.40709004	0.27753977	0.26568565	0.14511380	0.077296157	0.165159046	0.2363636	0.147641112
12	0	1	0.41644186	0.27663172	0.41330938	0.27041357	0.40146249	0.33685050	0.233223993	0.328330020	0.3949495	0.228727885
13	0	1	0.57688485	0.51031451	0.61232810	0.41548250	0.40417080	0.69449727	0.483833177	0.555666004	0.6752525	0.590564448
14	0	1	0.41975484	0.48156916	0.41400041	0.27113468	0.28328970	0.24789890	0.232849110	0.266600398	0.3974747	0.072030329
15	0	1	0.31941881	0.43625296	0.34420565	0.18443266	0.54590593	0.64388688	0.498594189	0.398856859	0.5095960	0.565711879
16	0	1	0.35775474	0.60297599	0.36583512	0.21857900	0.55312810	0.42978958	0.384020619	0.366003976	0.6277778	0.438289806
17	0	1	0.36438071	0.35238417	0.35208348	0.22948038	0.41563600	0.16140114	0.173266167	0.261381710	0.2656566	0.195029486
18	0	1	0.43300677	0.37098411	0.44440605	0.27796394	0.58111402	0.56076314	0.403467666	0.510934394	0.5575758	0.497051390
19	0	1	0.60717497	0.42069665	0.59574321	0.47359491	0.41238603	0.25556714	0.346532334	0.472067594	0.2636364	0.084035383
20	1	0	0.31042643	0.15725397	0.30177597	0.17934252	0.40769161	0.18989633	0.156138707	0.237624254	0.4166667	0.162173547
21	1	0	0.28865540	0.20290835	0.28912998	0.15970308	0.49535073	0.33010245	0.107029053	0.154572565	0.4580808	0.382266217
22	1	0	0.11940934	0.09232330	0.11436666	0.05531283	0.44930938	0.13968468	0.069259606	0.103180915	0.3813131	0.402064027
23	0	1	0.39561740	0.15387217	0.40570797	0.23792153	0.49354518	0.59542359	0.486644799	0.484890656	0.7378788	0.428812131
24	0	1	0.67106820	0.45079472	0.64549789	0.53467656	0.37600433	0.25403349	0.257029053	0.429025845	0.3580808	0.059393429

There are a total of 569 observations with 30 features in all. This is just a glimpse of the whole data.

3.2. Splitting of sample into training and testing samples:

Now we split the data into training and testing sets in the ratio of 80% and 20% respectively, such that the original proportion of benign and malignant cells in the parent sample is retained in both the splits. We achieve this using the sample.split() function found in the caTools package.

```
set.seed(123454)

split =sample.split(data[, 2], SplitRatio = 0.80)

data1_train =subset(data1, split == TRUE)
```

data1_train contains the training sample, and data1_test contains testing sample. We set seed to 123454 to achieve reproducibility of results.

3.3. <u>Data Visualisation</u>:

Here we perform PCA of training and testing data respectively, and take the first two scores (principal components) obtained and plot them against each other to visualize the training and testing data. We employ the prcomp function in R's 'Stats' package:

We use the following piece of R code :

```
par(mfrow = c(1,2))


pca_train <- prcomp(data1_train, retx = TRUE)

scores_train <- pca_train$x

train_new <- as.data.frame(cbind(data1_train$M, scores_train[,1:2]))

colnames(train_new)[1]="class"

plot(train_new$PC1, train_new$PC2, type = "p", pch = train_new$class+1,

    col = train_new$class+1,

    main = "Scatterplot of the first two Principal Components for the training set",

    xlab = "PCA1", ylab = "PCA2")


pca_test <- prcomp(data1_test, retx = TRUE)

scores_test <- pca_test$x


test_new <- as.data.frame(cbind(data1_test$M, corrclass, scores_test[,1:2]))

colnames(train_new)[1]="class"

plot(scores_test[,1], scores_test[,2], type = "p", pch = data1_test$M+1,

    col = data1_test$M+1,

    main = "Scatterplot of the first two Principal Components for the testing set",

    xlab = "PCA1", ylab = "PCA2")
```

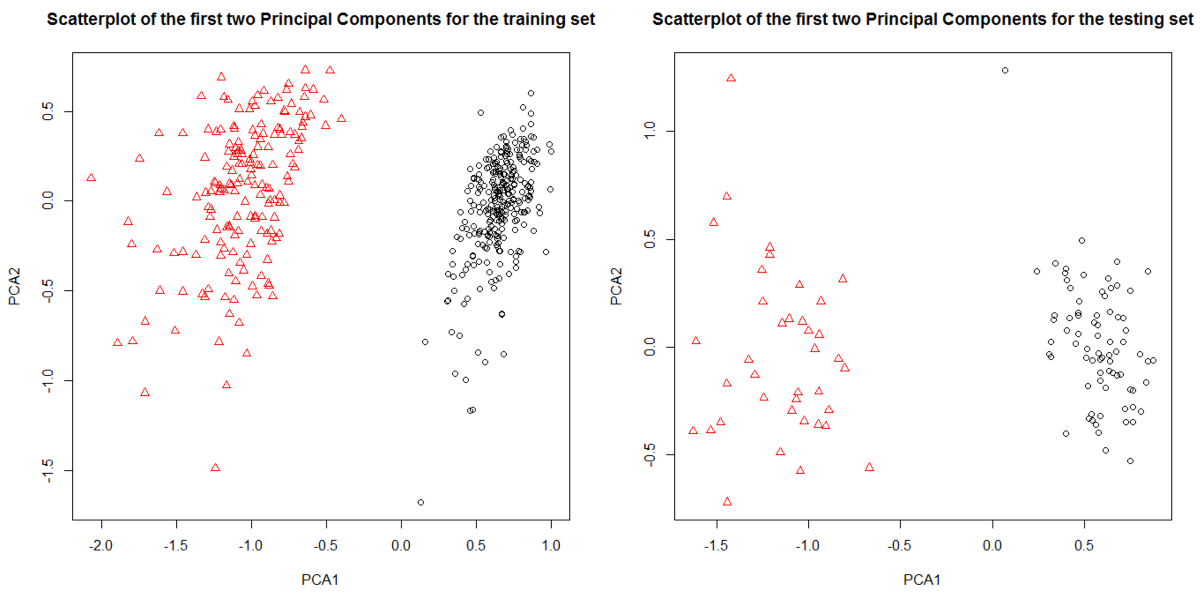


Fig 3.1: A two dimensional representation of the distribution of benign and Malignant cell in training and test samples.

3.4. <u>Initial fitting of the neural-networks to the training set</u>:

Initially, we fit three neural network models using Resilient back-propagation with, without weight back-tracking, and the traditional back-propagation models with two-node hidden layers and we compute the time taken. We use the **neural-net package** in R to do this.

Parameters taken for the initial models are :

Model	**Nodes in hidden layer**	**Activation function**	**Error function**	**Learning rate**
rprop+	2	Logistic	Cross-entropy	-
rprop-	2	Logistic	Cross-entropy	-
Backprop	2	Logistic	Cross-entropy	0.01

```
set.seed(13)

st_rprop = Sys.time()

model_rprop = neuralnet(formula = f, data = data1_train, act.fct="logistic",

                        hidden = 2, linear.output=FALSE, err.fct = "ce",

                        algorithm = "rprop+")

et_rprop = Sys.time()

tt_rprop = et_rprop - st_rprop

tt_rprop



set.seed(13)

st_rpropw = Sys.time()

model_rpropw = neuralnet(formula = f , data = data1_train, act.fct="logistic",

                         hidden = 2, linear.output=FALSE, err.fct = "ce",

                         algorithm = "rprop-")

et_rpropw = Sys.time()

tt_rpropw = et_rpropw - st_rpropw

tt_rpropw


set.seed(12)

st_back = Sys.time()

model_backprop = neuralnet(formula = f, data = data1_train, rep = 1, learningrate = 0.01,

                           stepmax = 1e+05, act.fct="logistic", hidden = 2, linear.output=FALSE,

                           err.fct = "ce", algorithm = "backprop")

et_back = Sys.time()

tt_back = et_back - st_back

tt_back
```

The variables model_rprop, model_rpropw, model_backprop contains the models and the tt_rprop, tt_rpropw and tt_back contains the time taken to fit these models.

3.5. **Investigation of performance of networks from test set:**

This is achieved by computing the predicted classes from the test data using the three models created in the previous section, and then finally calculating the various performance measures like accuracy, sensitivity, specificity, kappa, etc. Finally, we obtain the ROC curve for the three models. We achieve this using the following piece of code:

```
model_rprop.result <- compute(model_rprop, data1_test[,3:ncol(data1_test)])

model_rpropw.result <- compute(model_rpropw, data1_test[, 3 : ncol(data1_test)])

model_backprop.result <-  compute(model_backprop, data1_test[, 3 : ncol(data1_test)])



predicted_prob_rprop <- model_rprop.result$net.result

predicted_prob_rpropw <- model_rpropw.result$net.result

predicted_prob_backprop <- model_backprop.result$net.result



predicted_class_rprop <- max.col(predicted_prob_rprop)

predicted_class_rpropw <- max.col(predicted_prob_rpropw)
predicted_class_backprop <- max.col(predicted_prob_backprop)
observed_class <- max.col(data1_test[, 1 : 2])


conf_rprop <- confusionMatrix(predicted_class_rprop, observed_class)
conf_rpropw <- confusionMatrix(predicted_class_rpropw, observed_class)
conf_backprop <- confusionMatrix(predicted_class_backprop, observed_class)
conf_rprop


time1 <- rbind(tt_rprop, tt_rpropw, tt_back)
```

```
overall <- rbind(conf_rprop$overall, conf_rpropw$overall, conf_backprop$overall)

rownames(overall) <- c("rprop+", "rprop-", "backprop")

overall <- as.data.frame(cbind(overall, time1))

colnames(overall)[8] <- "time
```

The Confusion matrices generated by the three fitted models are :

Table 3.5.1 Confusion Matrix for Resilient Back-Propagation with weight back-tracking

		Observed	
Predicted		**Benign**	**Maglinant**
	Benign	74	3
	Malignant	2	35

Table 3.5.2 Confusion Matrix for Back-Propagation without weight back-tracking

		Observed	
Predicted		**Benign**	**Maglinant**
	Benign	73	2
	Maglinant	3	36

Table 3.5.3 Confusion Matrix for Traditional Back-Propagation

		Observed	
Predicted		**Benign**	**Malignant**
	Benign	72	3
	Malignant	4	35

Table 3.5.4 Performance Measures for the models

	Rprop	**Rpropw**	**Back-prop**
Sensitivity	0.9736842	0.9605263	0.9473684
Specificity	0.9210526	0.9473684	0.9210526
Pos Pred Value	0.961039	0.9733333	0.96
Neg Pred Value	0.9459459	0.9230769	0.8974359
Precision	0.961039	0.9733333	0.96
Recall	0.9736842	0.9605263	0.9473684
F1	0.9673203	0.9668874	0.9536424

Prevalence	0.6666667	0.6666667	0.6666667
Detection Rate	0.6491228	0.6403509	0.6315789
Detection Prevalence	0.6754386	0.6578947	0.6578947

<u>Interpretation of performance for initial models</u>: The sensitivity, i.e, the chances of the models of detecting a true benign cell is highest for Resilient backpropagation algorithms followed by Traditional Back-propagation algorithm. The specificity, i. e, the chances of detecting a Malignant (cancerous) cell correctly by Resilient back propagation algorithm are higher than Traditional Back-propagation algorithm

<u>Table 3.5.5 Accuracy, Kappa, and time taken for training</u>

Algorithm	**rprop+**	**rprop-**	**Backprop**
Measures			
Accuracy	0.9561404	0.9561404	0.9385965
Kappa	0.9006623	0.9019608	0.8627451
Accuracy Lower	0.9006113	0.9006113	0.8775731
Accuracy Upper	0.9856077	0.9856077	0.9749587
Accuracy Null	0.6666667	0.6666667	0.6666667
Accuracy P –value	4.24E-14	4.24E-14	3.10E-12
Mcnemar	1	1	1
Time	3.549203	3.069176	41.989402

Now we draw the ROC curves for the three models using the **pROC package** in R and their Area Under Curves.

```
rec_op_char_rprop = roc(data1_test[,2], predicted_prob_rprop[,2])

rec_op_char_rpropw = roc(data1_test[,2], predicted_prob_rpropw[,2])

rec_op_char_backprop = roc(data1_test[,2], predicted_prob_backprop[,2])


plot(1 - rec_op_char_rprop$specificities,
    rec_op_char_rprop$sensitivities, type = "l", xlab = "1-specificity",
    ylab = "sensitivity", main = "ROC Curve", lwd = 2, col = "black")
lines(1 - rec_op_char_rpropw$specificities,
     rec_op_char_rpropw$sensitivities, lty = 2, lwd = 2, col = "blue")
lines(1 - rec_op_char_backprop$specificities,
     rec_op_char_backprop$sensitivities, lty = 4, lwd = 2, col = "red")
abline(a = 0, b = 1, lwd = 2, lty = 2)
legend(0.7, 0.3, legend = c("rprop+", "rprop-", "Back-prop"),
      lty = c(1, 2, 4), lwd = 2, col = c("black", "blue", "red"))


area = rbind(rec_op_char_backprop$auc,
             rec_op_char_rprop$auc,
             rec_op_char_rpropw$auc)
rownames(area) = c("Back-propagation", "rprop+", "rprop-")
```

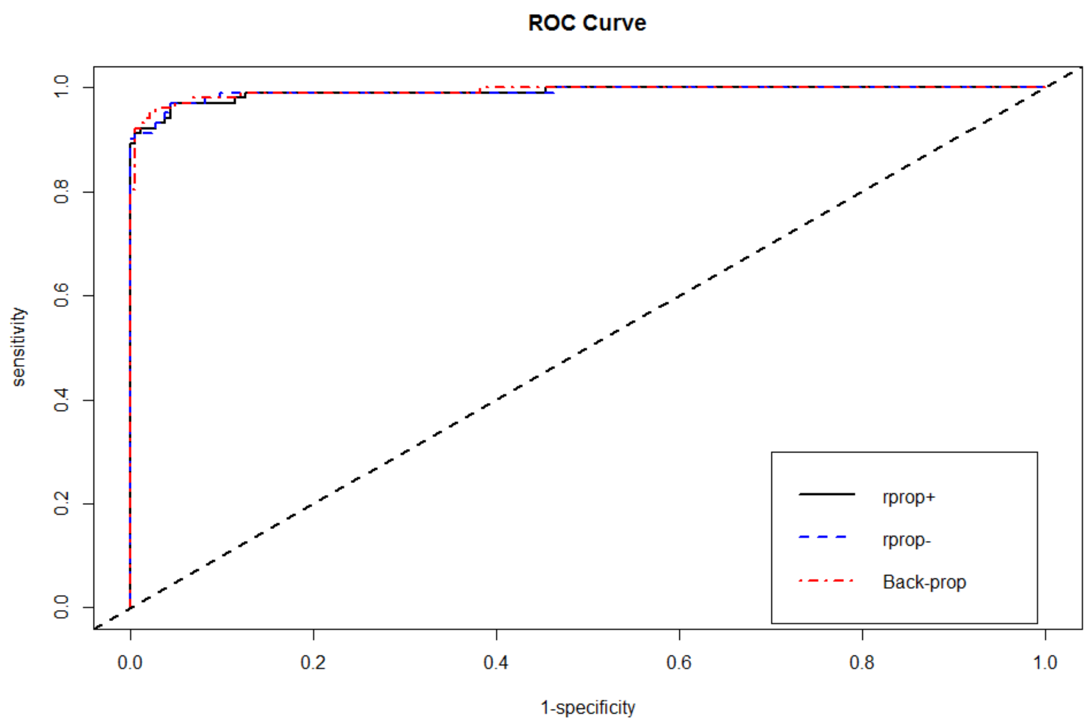


Fig 3.2 ROC curves of initial models

Table 3.5.6 Area Under Curve for the initial classifiers

Algorithm	**AUC**
Back-propagation	0.992553
rprop+	0.991053
rprop-	0.991535

Interpretation of ROC for initial models: It is quite evident that accuracy of resilient back propagation is more than traditional back propagation, by about 2%. Further, the time taken by resilient back-propagation algorithms to converge is also lesser compared to traditional back-propagation. Further, The Areas under the curves obtained are relatively close to one another. This indicates that traditional back-propagation, although slower in convergence as well as lesser accurate, are still decent classifiers.

3.6. Model Improvement

Now, we will attempt at improving the models, by finding the optimal number of nodes in the hidden layer using K- fold Cross validation. We have written our own function of k-fold cross validation using createFolds function of **caret** package to create the initial k folds from the training data.

```
#K-Fold Cross Validation


set.seed(2)

test_index = createFolds(data1_train[,1], k = 10 )

test_index

test_index[[1]]

cross_val_2 = function(struct = 2, algo = "rprop+", lr = NULL,

                       dataframe = data1_train,

                       folds = 10, ind = test_index){

  #Initialise the error prop

  train_prop = NULL

  test_prop  = NULL

  for(i in 1 : folds) {

    data_train = dataframe[-ind[[i]],]

    data_test = dataframe[ind[[i]], ]

    #Training our neural network

f=as.formula(B+M~mean_radius+mean_texture+mean_perimeter+mean_area+mean_smoothness+mean_compactness+

mean_concavity+mean_concave_points+mean_symmetry+mean_fractal_dimension+radius_SE+texture_SE+

perimeter_SE+area_SE+smoothness_SE+compactness_SE+concavity_SE+concave_points_SE+symmetry_SE+

fractal_dimension_SE+worst_radius+worst_texture+worst_perimeter+worst_area+worst_smoothness+

worst_compactness+worst_concavity+worst_concave_points+worst_symmetry+worst_fractal_dimension)


    set.seed(123)
    model = neuralnet(formula = f, data = data_train, hidden=struct,
                       learningrate = lr, algorithm = algo, act.fct="logistic",
                       err.fct = "ce", linear.output=FALSE)
      #Computing The Predictions

    result_train = compute(model, data_train[,3:ncol(data_train)])
    pred_train = result_train$net.result
    result_test = compute(model, data_test[,3:ncol(data_test)])
    pred_test = result_test$net.result

    #Computation of efficiency from training data
    orig_diag_train = max.col(data_train[,1:2])
    pred_diag_train = max.col(pred_train)
    per_eff_train = mean(orig_diag_train==pred_diag_train)*100
    train_prop[i] = per_eff_train

    #Computation of efficiecy from testing data
    orig_diag_test = max.col(data_test[,1:2])
    pred_diag_test = max.col(pred_test)
    per_eff_test = mean(orig_diag_test==pred_diag_test)*100
    test_prop[i] = per_eff_test
  }
  l = list(train_prop, test_prop)
  return(sapply(l, mean))
}
```

Now, we apply this function over 9 values of the ‘struct’ parameter of the function, i.e, number of nodes in the hidden layer, and over three algorithms. We use map() and map2() function from **purrr package** in R to achieve this.

```
x <- rep(c(2:10), 2)

y <- rep(c("rprop+", "rprop-"), each = 9)

cv_res <- map2( .x = x, .y = y, .f = cross_val_2, ind = test_index)

cv_res2 <- map(.x = c(2:10), .f = cross_val_2, algo = "backprop",

            lr=0.01, ind = test_index)

cv_final <- list(cv_res, cv_res2)

v1<- as.data.frame(t(data.frame(cv_res)))

rownames(v1) <- NULL

v2 <- as.data.frame(t(data.frame(cv_res2)))

rownames(v2) <- NULL

cv <- rbind(v1, v2)

a <- rep(seq(2,10,1), 3)

b <- rep(c("rprop+", "rprop-", "backprop"), each = 9)

cv2 <- cbind(a, b, cv)

colnames(cv2) <- c("nodes", "Algorithm", "training error", "testing error")

cv3 <- gather(data = cv2, key = "error", value, -c(nodes, Algorithm))

cv4 <- spread(data = cv3, key = Algorithm, value = value)

cv4_train <- cv4[which(cv4$error=="training error"),]

cv4_test <- cv4[-which(cv4$error=="training error"),]
```

Table 3.6.1 Accuracy for each model at different nodes

Nodes	**Backprop**	**rprop-**	**rprop+**
2	96.47343	96.26087	95.81159
3	96.47826	97.13043	97.36232
4	96.69565	96.91787	96.26087
5	96.47343	95.81643	96.91787
6	97.13043	96.46377	97.1401
7	96.91304	96.03382	96.69565

8	96.69082	97.13043	97.57005
9	96.69565	95.80676	97.13527
10	96.25604	94.93237	97.35266

We plot the accuracies for the three models against the number of nodes using the **matplot** function in R’s **graphics** package:

```
matplot(cv4_test$nodes, cv4_test[,3 : 5], main = "Testing accuracy",
    xlab = "Nodes", ylab = "Accuracy", type = "o", pch = 16,
    col = c("red", "blue", "black"))
legend(2, 95.5, legend = c("backprop", "rprop-", "rprop+"),
    pch = 16, col = c("red", "blue", "black"), lty = c(1, 2, 3))
```

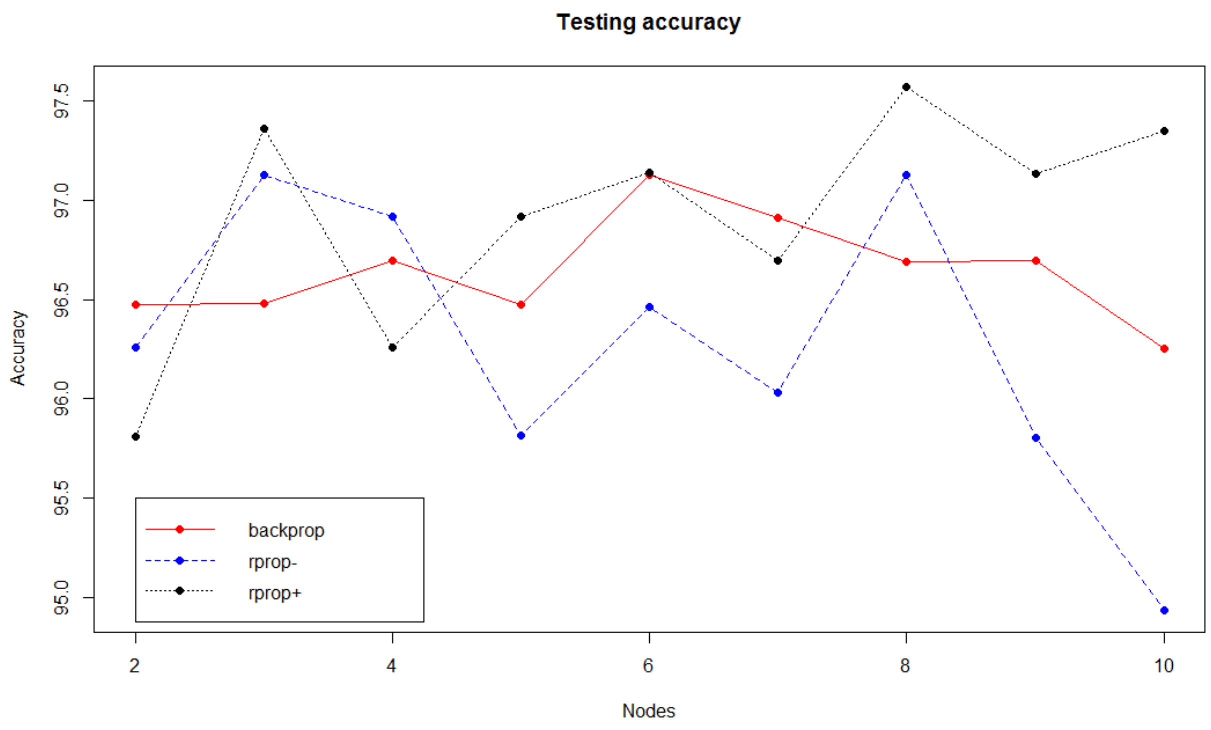


Fig 3.3 Accuracies of the two models plotted against at different nodes.

Hence, we observe that, after cross-validation, back-propagation produces least error at 6 nodes, resilient back-propagation without weight back-tracking produces least error at 3 and 8 nodes, and resilient back-propagation without weight back-tracking produces least error at 8 nodes.

Now we fit new models on the training data with these newly obtained optimal parameters.

3.7. Improved Model fitting and validation

We perform the same set of actions used to fit the initial models, with the exception of the number of nodes in the hidden layer being those obtained by cross-validation. We use the following code.

Parameters taken for the improved models are :

Model	Nodes in hidden layer	Activation function	Error function	Learning rate
rprop+	8	Logistic	Cross-entropy	-
rprop-	3	Logistic	Cross-entropy	-
Backprop	6	Logistic	Cross-entropy	0.01

```
set.seed(13)

st_rprop2 <- Sys.time()

model_rprop2 <- neuralnet(formula = f, data = data1_train, act.fct="logistic",

                    hidden = 8, linear.output=FALSE, err.fct = "ce",

                    algorithm = "rprop+")

et_rprop2 <- Sys.time()

tt_rprop2 <- et_rprop2 - st_rprop2

tt_rprop2


set.seed(13)

st_rpropw2 <- Sys.time()

model_rpropw2 <- neuralnet(formula = f , data = data1_train, act.fct="logistic",

                    hidden = 3, linear.output=FALSE, err.fct = "ce",

                    algorithm = "rprop-")

et_rpropw2 <- Sys.time()

tt_rpropw2 <- et_rpropw2 - st_rpropw2

tt_rpropw2


set.seed(12)

st_back2 <- Sys.time()

model_backprop2 <- neuralnet(formula = f, data = data1_train, rep = 1,
                        learningrate = 0.01, stepmax = 1e+05,
                        act.fct="logistic", hidden = 6, linear.output=FALSE,
                        err.fct = "ce", algorithm = "backprop")
et_back2 <- Sys.time()
tt_back2 <- et_back2 - st_back2
tt_back2

model_rprop.result2 <- compute(model_rprop2, data1_test[,3:ncol(data1_test)])
model_rpropw.result2 <- compute(model_rpropw2, data1_test[, 3 : ncol(data1_test)])
model_backprop.result2 <-  compute(model_backprop2, data1_test[, 3 : ncol(data1_test)])

predicted_prob_rprop2 <- model_rprop.result2$net.result
predicted_prob_rpropw2 <- model_rpropw.result2$net.result
predicted_prob_backprop2 <- model_backprop.result2$net.result

predicted_class_rprop2 <- max.col(predicted_prob_rprop2)
predicted_class_rpropw2 <- max.col(predicted_prob_rpropw2)
predicted_class_backprop2 <- max.col(predicted_prob_backprop2)
observed_class2 <- max.col(data1_test[, 1 : 2])

conf_rprop2 <- confusionMatrix(predicted_class_rprop2, observed_class2)
conf_rpropw2 <- confusionMatrix(predicted_class_rpropw2, observed_class2)
conf_backprop2 <- confusionMatrix(predicted_class_backprop2, observed_class2)

time2 <- rbind(tt_rprop2, tt_rpropw2, tt_back2)
overall2 <- rbind(conf_rprop2$overall, conf_rpropw2$overall, conf_backprop2$overall)
rownames(overall2) <- c("rprop+", "rprop-", "backprop")
overall2 <- as.data.frame(cbind(overall2, time2))
colnames(overall2)[8] <- "time"
```

The final weights of the improved rprop model are :

Table 3.7.1.1 Weight vectors to obtain the hidden layer of improved resilient back-propagation model with weight backtracking

Starting Nodes				**Nodes**				
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	1.173609	1.89561	1.578061	-1.15496	1.331965	-1.82004	2.040915	2.177206
2	0.237271	1.331394	0.528932	2.535887	-0.16804	0.102845	0.564031	0.517292
3	-0.05323	-0.69299	0.996599	2.720545	-0.15812	-5.7846	-6.69529	-1.60189
4	0.228091	0.546351	0.974918	3.315117	-0.01327	-0.75862	0.868645	0.046687
5	0.797764	-0.72906	-0.79079	-0.99177	-0.96054	1.775586	1.399427	0.013016
6	0.731741	-0.15976	1.333925	0.026074	-0.14201	2.72996	1.240707	0.194627
7	4.236112	-0.14664	2.445604	-17.7092	0.9021	-2.94725	2.148553	2.136482
8	-1.06736	-2.4691	-2.22376	4.12083	-3.30202	0.251601	-2.4367	-3.29255
9	-1.28868	-2.88922	-4.33489	-0.0448	-0.79055	2.399307	-0.30401	-1.23629
10	1.267089	0.252041	1.612566	-1.67083	1.415155	0.613595	0.571757	0.006445
11	-1.30464	1.56781	0.090945	19.86185	6.655301	-1.21192	1.883154	-2.70474
12	-1.84079	12.1392	-5.35874	7.944953	-5.32827	6.662364	-1.43225	-1.96316
13	1.536874	4.058434	0.339242	12.12828	0.300491	-1.28029	1.861353	5.120385
14	-2.59481	16.56006	-0.45528	-12.9081	6.962788	-6.57846	-2.08867	-1.36445
15	-13.5195	-16.4514	-17.3178	7.788251	-20.1144	22.5218	-7.36592	-18.8616
16	-0.04614	-6.8378	-3.86215	2.656371	1.051415	1.487976	1.558427	0.107965
17	1.849965	-0.78393	1.853492	-25.9508	1.059274	-1.94114	1.533899	0.869073
18	-4.4843	-2.54591	1.008312	7.578112	-1.03147	3.792153	1.090069	1.656373
19	-0.10134	3.01201	0.207373	1.63311	-0.53979	0.429262	1.131667	1.532448
20	0.908115	1.522466	0.762234	-9.24354	-0.84807	-1.30351	-1.42947	0.479873
21	2.874735	7.106428	10.7772	-0.81139	2.286956	-2.04633	-4.55963	1.103273
22	1.430552	-0.91321	-2.35478	-1.01315	1.633104	1.276146	1.421291	-0.97372
23	-2.28485	-1.21478	0.793655	-4.42722	-1.90363	0.343699	-3.55322	-0.71642
24	-1.44719	-0.33016	-0.41603	-1.88133	-0.42993	1.710207	-1.21155	0.123427
25	-0.72385	-2.03581	-9.25802	1.45631	-11.7243	1.389002	-4.32703	-1.7289
26	0.119384	-6.19104	-1.09677	0.188532	-0.8059	3.577871	1.632786	-6.42988
27	-1.5569	0.189493	1.180778	-34.8515	-2.91968	1.290897	-1.20954	-1.13753
28	-4.37269	0.72535	-0.43282	-3.40565	1.197823	-0.7707	-0.06919	-0.65089
29	-1.4668	-0.27768	-0.31156	3.038184	-1.47974	1.630723	-2.09954	0.389004
30	-0.77623	-3.09131	-3.42963	-13.6615	-1.89706	6.582427	0.467243	-0.82185
31	-0.03475	0.17152	-1.08313	9.760299	-0.1036	-5.28946	-3.79422	2.577165

Table 3.7.1.2 Weight vectors to obtain the output layer of improved resilient back-propagation model with weight backtracking

Hidden Nodes	**Nodes**	
	1	**2**
1	-4.17299	3.492767
2	18.32241	-14.5231

3	23.1086	-23.0825
4	8.837016	-9.31802
5	-56.2415	56.64369
6	41.61336	-41.0121
7	-44.8528	45.01795
8	71.04155	-72.5495
9	26.18163	-26.6846

The final weights of improved Resilient backpropagation model without weight backtracking are:

Table 3.7.2.1 Weight vectors to obtain the hidden layer of improved resilient back-propagation model without weight backtracking

Starting Nodes		**Nodes**	
	1	**2**	**3**
1	3.910442	5.163862	3.070331
2	-0.17032	0.04974	1.38121
3	-4.77239	-1.48106	2.116778
4	0.283765	0.160053	0.735998
5	1.920647	-0.76718	3.290423
6	2.237222	0.644087	-1.8646
7	4.497703	-0.75029	4.578372
8	-2.01754	-3.96009	-2.8335
9	-2.46422	-3.43429	-4.11639
10	1.471293	0.709773	0.01422
11	0.574563	3.201423	1.611399
12	-7.10609	17.65222	-12.5665
13	-1.9747	19.2462	0.325963
14	-5.40813	38.91449	2.852488
15	-25.3172	-6.61652	-19.6929
16	3.553747	-10.0646	-3.86838
17	3.675773	-0.54212	1.423817
18	-3.42784	-12.4884	1.124108
19	0.018156	1.922917	0.651929
20	3.580659	-1.79263	1.303214
21	-4.88776	7.626227	5.215534
22	0.852503	-1.25373	-2.92381
23	-6.38529	-5.85975	-0.51237
24	-1.15168	-0.6694	-1.00936
25	-1.95273	-22.983	-0.82803
26	2.881809	-2.32884	-2.55582
27	-1.99298	-0.19737	-0.068
28	-3.41064	-0.55808	-1.03414
29	-2.50899	-1.23658	-1.85023

30	-2.69021	-6.03311	-3.96364
31	-0.82068	5.725383	-1.28903

Table 3.7.2.2 Weight vectors to obtain the output layer of improved resilient back-propagation model without weight backtracking

Hidden Nodes	**Nodes**	
	1	**2**
1	-47.1448	45.17682
2	72.51975	-70.1117
3	26.52335	-25.2658
4	66.06102	-63.7673

The final weights of the improved traditional back propagation algorithm are given as :

Table 3.7.3.1 Weight vectors to obtain the hidden layer of improved traditional back-propagation model.

Starting Nodes			**Nodes**			
	1	**2**	**3**	**4**	**5**	**6**
1	-0.62502	13.3182	-3.91159	1.135494	5.627459	16.14624
2	1.707558	-0.44276	-2.75613	1.344362	2.569659	-0.09052
3	-0.41537	-1.58881	2.510837	-5.29609	-6.63043	-1.77765
4	-0.76981	-0.3462	-1.86432	0.712569	3.329281	0.74494
5	-1.99319	-1.94983	0.490852	-0.90234	-1.29957	-1.69838
6	-0.81876	1.132564	-0.51025	-0.67328	-1.13054	-2.22597
7	0.08348	0.958586	-10.9792	-0.34745	2.212213	14.26157
8	-0.82433	-5.30167	2.216836	0.186327	-1.19076	-12.1448
9	-0.66594	-8.13838	2.601324	-1.75844	-5.3066	-5.2699
10	0.360524	2.754341	1.099445	1.214876	-0.1507	4.112727
11	-0.86323	0.243117	7.162191	-2.0009	-0.0539	3.774865
12	-2.14323	-7.8541	11.63677	-4.02032	-4.70189	-5.59435
13	-0.95785	-4.72201	2.263264	-2.39664	-5.86649	11.92702
14	-0.44705	-5.99691	6.795456	-3.0804	-1.94147	1.815804
15	-0.63888	-4.55874	5.54401	-1.85468	-4.62616	-4.41869
16	-0.62954	-4.10684	-4.47151	-0.06343	2.929761	-8.34887
17	1.309723	0.303032	-5.23767	1.559759	1.29048	0.466833
18	0.451943	-0.65696	-1.96942	0.003113	-0.33642	-6.99228
19	0.499504	-3.32002	-1.27634	-1.64807	-1.30602	5.234912
20	-0.37249	-1.06113	-10.6719	-0.89106	-0.19697	1.726935
21	0.692567	3.107344	-3.03868	1.59501	2.006466	8.617417
22	1.687857	-2.03455	5.202564	-2.67427	-1.59968	-10.4788
23	1.635879	-0.40827	-2.10541	-7.00257	-8.21759	-9.725
24	-0.37361	-2.56073	1.784802	-1.188	0.836884	-1.29396

25	-1.40306	-2.72714	4.75157	-1.72757	-2.44418	-10.768
26	-0.39309	-2.72625	0.635893	1.367457	5.757556	-6.3404
27	-0.14054	0.032104	-2.57999	0.454652	1.223763	3.824335
28	0.141066	-3.28063	-0.76304	-1.72044	1.679919	-5.32936
29	-0.19086	-4.60897	4.278034	-1.79738	-4.45086	-8.57591
30	-0.11226	-1.92549	3.028396	-2.20118	-3.10494	-14.3249
31	0.833098	1.522884	5.290141	-2.58447	2.771442	2.397736

Table 3.7.3.2 Weight vectors to obtain the output layer of improved traditional back-propagation model

Hidden Nodes	**Nodes**	
	1	**2**
1	-15.3341	15.16458
2	1.753826	-1.17549
3	9.645002	-9.7323
4	-24.3677	24.48728
5	5.579329	-5.3566
6	11.22744	-11.1958
7	22.06509	-22.0442

```
rec_op_char_rprop2 <- roc(data1_test[,2], predicted_prob_rprop2[,2])

rec_op_char_rpropw2 <- roc(data1_test[,2], predicted_prob_rpropw2[,2])

rec_op_char_backprop2 <- roc(data1_test[,2], predicted_prob_backprop2[,2])


plot(1 - rec_op_char_rprop2$specificities, rec_op_char_rprop2$sensitivities,
     type = "l", xlab = "1-specificity", ylab = "sensitivity",
     main = "Revised ROC Curve", lwd = 2, col = "black")
lines(1 - rec_op_char_rpropw2$specificities, rec_op_char_rpropw2$sensitivities,
      lty = 2, lwd = 2, col = "blue")
lines(1 - rec_op_char_backprop2$specificities, rec_op_char_backprop2$sensitivities,
      lty = 4, lwd = 2 , col = "red")
abline(a = 0, b = 1, lwd = 2, lty = 2)
legend(0.7, 0.3, legend = c("rprop+", "rprop-", "backprop"),
       lty=c(1, 2, 4), col =c("black", "blue", "red"))
area2 <- rbind(rec_op_char_backprop2$auc,
            rec_op_char_rprop2$auc,
            rec_op_char_rpropw2$auc)
rownames(area2) <- c("backprop", "rprop+", "rprop-")
```

Table 3.7.4 Confusion matrix for improved rprop model

		Predicted	
Predicted		**Benign**	**Malignant**
	Benign	74	2
	Malignant	2	36

Table 3.7.5 Confusion matrix for improved rpropw model

		Predicted	
Predicted		**Benign**	**Malignant**
	Benign	74	2
	Malignant	2	36

Table 3.7.6 Confusion matrix for improved Back-prop model

		Observed	
Predicted		**Benign**	**Malignant**
	Benign	74	3
	Malignant	2	35

Table 3.7.7 Performance Measures of improved models

	Rprop2	**Rpropw2**	**Back-prop2**
Sensitivity	0.9736842	0.9736842	0.9736842
Specificity	0.9473684	0.9473684	0.9210526
Pos Pred Value	0.9736842	0.9736842	0.961039
Neg Pred Value	0.9473684	0.9473684	0.9459459
Precision	0.9736842	0.9736842	0.961039
Recall	0.9736842	0.9736842	0.9736842
F1	0.9736842	0.9736842	0.9673203
Prevalence	0.6666667	0.6666667	0.6666667
Detection Rate	0.6491228	0.6491228	0.6491228
Detection Prevalence	0.6666667	0.6666667	0.6754386
Balanced Accuracy	0.9605263	0.9605263	0.9473684

Table 3.7.8 Accuracy, Kappa and time taken for training

Algorithm	**rprop+**	**rprop-**	**Back prop**
Measures			
Accuracy	0.9649123	0.9649123	0.9561404
Kappa	0.9210526	0.9210526	0.9006623
Accuracy Lower	0.9125955	0.9125955	0.9006113
Accuracy Upper	0.9903585	0.9903585	0.9856077
Accuracy Null	0.6666667	0.6666667	0.6666667
Accuracy PValue	3.78E-15	3.78E-15	4.24E-14
Mcnemar PValue	1	1	1
time	3.663209	3.308189	1.1824(minutes)

Interpretation of performance of improved models:

From Table 3.7.7., the sensitivity, i.e, the chance of the models detecting a benign cell correctly are equally high. The specificity, i. e, the chance of detecting a Malignant (cancerous) cell correctly by Resilient back propagation algorithm are higher than Traditional Back-propagation algorithm.

From the Table 3.7.8., we observe that Accuracy, i. e, the proportion of times the algorithms correctly classify a cell as benign or malignant, for Resilient Back-propagation algorithms are 96.49123% each followed by Traditional Back-Propagation algorithm of 95.61404%. The kappa statistic shows similar results, with Resilient Back-Propagation heavily outperforming Traditional back-propagation. Judging from the p-values, we can be fairly certain that as far as this dataset is concerned, the accuracies are significantly more than the null-value at a very low level of significance. However, when we look at the time taken by the models to fit, Traditional back-propagation is clearly no match for Resilient Back-propagation algorithms.

The ROC curves for the three models overlaid on the same graph are provided below:

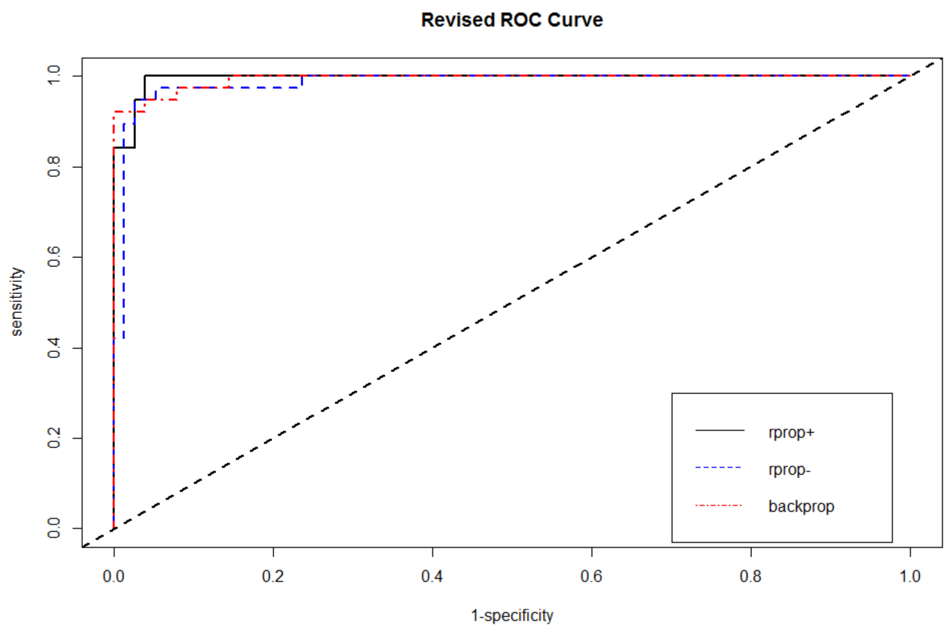


Fig 3.5 ROC curves for improved models

Table 3.7.9 Area Under Curve for the three classifiers

Algorithm	**AUC**
Backprop	0.993075
rprop+	0.995152
rprop-	0.984765

Interpretation of ROC for improved models:

Thus, the AUCs for all three algorithms are really high, with AUC for resilient back-propagation with weight back tracking being the highest of 0.995152, and resilient back-propagation without weight back-tracking being the lowest of 0.984765.

Summary and Conclusion

1) We have managed to build a robust classification model using neural networks that can classify cancerous cells into benign and malignant cells with a high accuracy. The mathematical form of model can be calculated manually by using the weights obtained in tables 3.7. However, this is unnecessary as these weights can be directly fed into the **neuralnet** function in R, or any other function equivalent to this in any software, and classifies a new data point. Hence, we have avoided mentioning the model explicitly.
2) Thus, in this project, we observed that traditional back propagation works almost equally as resilient back propagation with and without weight back-tracking. However, it is not advisable to use traditional back-propagation as it takes significantly more time than resilient back-propagation algorithms. On the other hand, resilient back-propagation algorithms significantly improve the weight convergence time, thereby rendering them more feasible for use in real life situations.

Bibliography

1. Lentz Brett, “Machine Learning with R (2nd edition)”, Packt Publishing, 2015.
2. Hastie Trevor & Tibshirani Robert & Friedman Jerome, “Elements of Statistical Learning (2nd edition)”, Springer,2008.
3. Rajasekaran S. & Vijayalakshmi Pai G.A, “Neural Networks,Fuzzy logic and Genetic Algorithms”, PHI Learning Private Limited,2009.
4. Basheera I.A & Hajmeer M,“Artificial Neural Networks: Fundamentals,Computing, design and Application”, Journal of Microbiological Methods, 43, 3–31, 2000.
5. Bailer-Jones C.A.L & Gupta Ranjan & P. Singh Harindar, “Automated Data Analysis in Astronomy” , Narosa Publishing House,New Delhi,India ,pp 51-58, 2001.
6. Dongare A.D. & Kharde R.R. & Kachare Amit D, “ Introduction to Artificial Neural Network”, International Journal of Engineering and Innovative Technology, 2(1), 2012.
7. Mu Tingting & K. Nandi Asoke, “Breast Cancer Diagnosis from Fine-Needle Aspiration Using Supervised Compact Hyperspheres and Establishment of Confidence of Malignancy”,16th European Signal Processing Conference (EUSIPCO 2008), Lausanne, Switzerland, 2008.
8. Anagnostopoulos Ioannis & Anagnostopoulos Christos & Rouskas Angelos & Kormentzas George & Vergados Dimitrios, “The Wisconsin Breast Cancer Problem: Diagnosis and DFS Time Prognosis Using Probabilistic and Generalised Regression Neural Classifiers” ,The Oncology Reports, special issue Computational Analysis and Decision Support Systems in Oncology, 2005.
9. Fawzi M. Al-Naima & Ali H. Al-Timemy,“Resilient Back Propagation Algorithm for Breast Biopsy Classification Based on Artificial Neural Networks, Computational Intelligence and Modern Heuristics”, Al-Dahoud Ali (Ed.),pp 145-158 ,InTech ,2010.Available from: http://www.intechopen.com/books/computational-intelligence-and-modernheuristics/resilient-backpropagation-algorithm-for-breast-biopsy-classification-based-on-artificial-neural-net,pp 145-158,InTech ,2010.
10. https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wiscosin/wdbc.data.