WINE CLASSIFICATION USING NEURAL NETWORKS

Project On Classification of Three Types of Wines Using Multi-layered Neural Network

Submitted By:
Rajat Singh Panwar(BT13CSE052)
Aditya Tigga(BT13CSE049)
Hitanshu Seepal(BT14CSE019)

Submitted To:
Maroti Deshmukh
Assistant Professor
National Institute of Technology, Uttarakhand

INDEX

I. ABOUT THE PROBLEM

The problem posed here is to classify different types of wines Using chemical analysis determine the origin of wines.
We have been given different attributes of different types of wines obtained from chemical analysis. There are a total of 13 attributes to determine the type of class the wine belongs to. We can model this problem as a classification problem. Although there are various approaches we can take to solve this problem, but to demonstrate the versatility of Neural Networks we are going to solve this problem using Multi-layered Neural Network.

II. DATASET DESCRIPTION

1. Title of Database: Wine recognition data
Updated Sept 21, 1998 by C.Blake : Added attribute information

2. Sources:
(a) Forina, M. et al, PARVUS - An Extendible Package for Data Exploration, Classification and Correlation. Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno, 16147 Genoa, Italy.

(b) Stefan Aeberhard, email: stefan@coral.cs.jcu.edu.au
(c) July 1991
3. Past Usage:

(1)
S. Aeberhard, D. Coomans and O. de Vel,
Comparison of Classifiers in High Dimensional Settings,
Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Technometrics).

The data was used with many others for comparing various classifiers. The classes are separable, though only RDA has achieved 100% correct classification.
(RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
(All results using the leave-one-out technique)

In a classification context, this is a well posed problem with "well behaved" class structures. A good data set

for first testing of a new classifier, but not very
challenging.

(2)
S. Aeberhard, D. Coomans and O. de Vel,
"THE CLASSIFICATION PERFORMANCE OF RDA"
Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Journal of Chemometrics).

Here, the data was used to illustrate the superior performance of
the use of a new appreciation function with RDA.

4. Relevant Information:

-- These data are the results of a chemical analysis of
wines grown in the same region in Italy but derived from three
different cultivars.
The analysis determined the quantities of 13 constituents
found in each of the three types of wines.

-- I think that the initial data set had around 30 variables, but
for some reason I only have the 13 dimensional version.
I had a list of what the 30 or so variables were, but a.)
I lost it, and b.), I would not know which 13 variables
are included in the set.

-- The attributes are (dontated by Riccardo Leardi,
riclea@anchem.unige.it)
1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10)Color intensity
11)Hue
12)OD280/OD315 of diluted wines
13)Proline

5. Number of Instances

class 1 59
class 2 71
class 3 48

6. Number of Attributes

13

7. For Each Attribute:

All attributes are continuous

No statistics available, but suggest to standardise variables for certain uses (e.g. for us with classifiers which are NOT scale invariant)

NOTE: 1st attribute is class identifier (1-3)

8. Missing Attribute Values:

None

9. Class Distribution: number of instances per class

class 1 59
class 2 71
class 3 48

III. DATASET ANALYSIS

1. Dataset Summary

V1	V2	V3	V4
Min. :1.000000	Min. :11.03000	Min. :0.740000	Min. :1.360000
1st Qu.:1.000000	1st Qu.:12.36250	1st Qu.:1.602500	1st Qu.:2.210000
Median :2.000000	Median :13.05000	Median :1.865000	Median :2.360000
Mean :1.938202	Mean :13.00062	Mean :2.336348	Mean :2.366517
3rd Qu.:3.000000	3rd Qu.:13.67750	3rd Qu.:3.082500	3rd Qu.:2.557500
Max. :3.000000	Max. :14.83000	Max. :5.800000	Max. :3.230000

V5	V6	V7	V8
Min. :10.60000	Min. : 70.00000	Min. :0.980000	Min. :0.34000
1st Qu.:17.20000	1st Qu.: 88.00000	1st Qu.:1.742500	1st Qu.:1.20500
Median :19.50000	Median : 98.00000	Median :2.355000	Median :2.13500
Mean :19.49494	Mean : 99.74157	Mean :2.295112	Mean :2.02927
3rd Qu.:21.50000	3rd Qu.:107.00000	3rd Qu.:2.800000	3rd Qu.:2.87500
Max. :30.00000	Max. :162.00000	Max. :3.880000	Max. :5.08000

V9	V10	V11	V12
Min. :0.1300000	Min. :0.410000	Min. : 1.28000	Min. :0.4800000
1st Qu.:0.2700000	1st Qu.:1.250000	1st Qu.: 3.22000	1st Qu.:0.7825000
Median :0.3400000	Median :1.555000	Median : 4.69000	Median :0.9650000
Mean :0.3618539	Mean :1.590899	Mean : 5.05809	Mean :0.9574494
3rd Qu.:0.4375000	3rd Qu.:1.950000	3rd Qu.: 6.20000	3rd Qu.:1.1200000
Max. :0.6600000	Max. :3.580000	Max. :13.00000	Max. :1.7100000

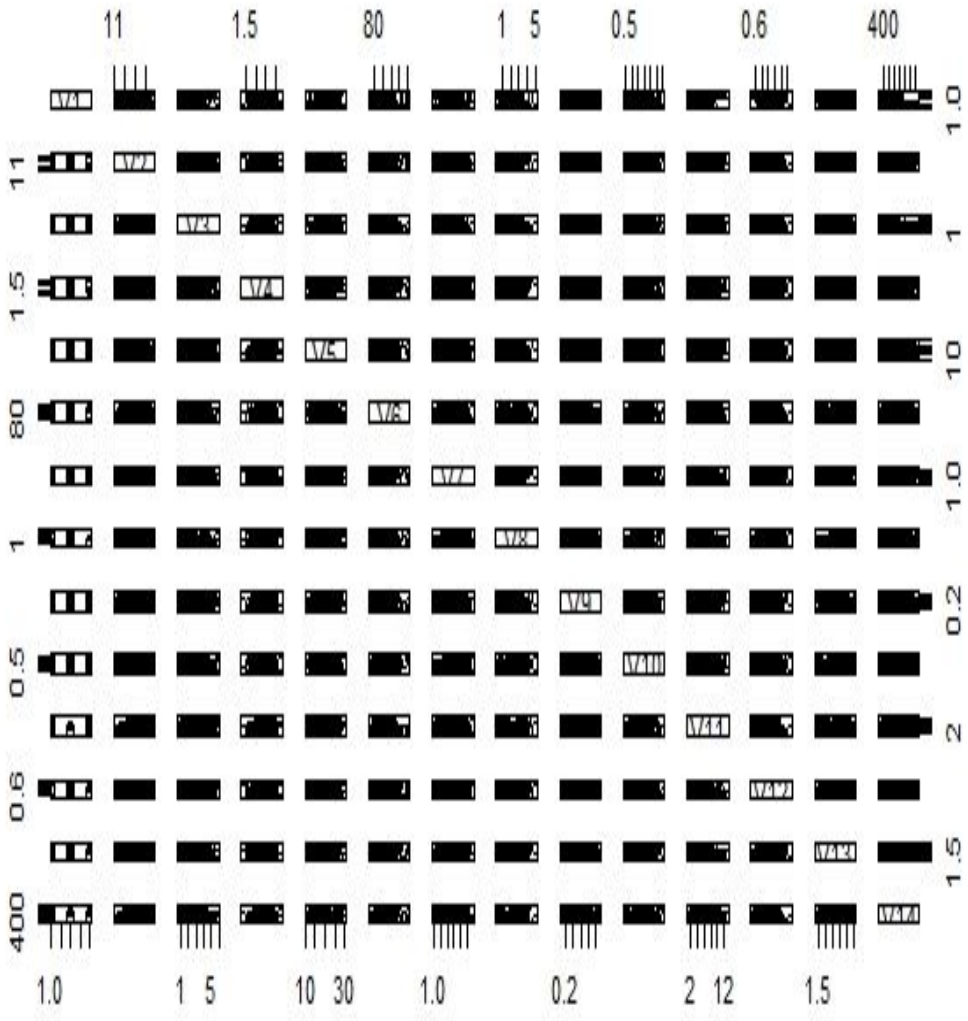
V13	V14
Min. :1.270000	Min. : 278.0000
1st Qu.:1.937500	1st Qu.: 500.5000
Median :2.780000	Median : 673.5000
Mean :2.611685	Mean : 746.8933
3rd Qu.:3.170000	3rd Qu.: 985.0000
Max. :4.000000	Max. :1680.0000

Where,

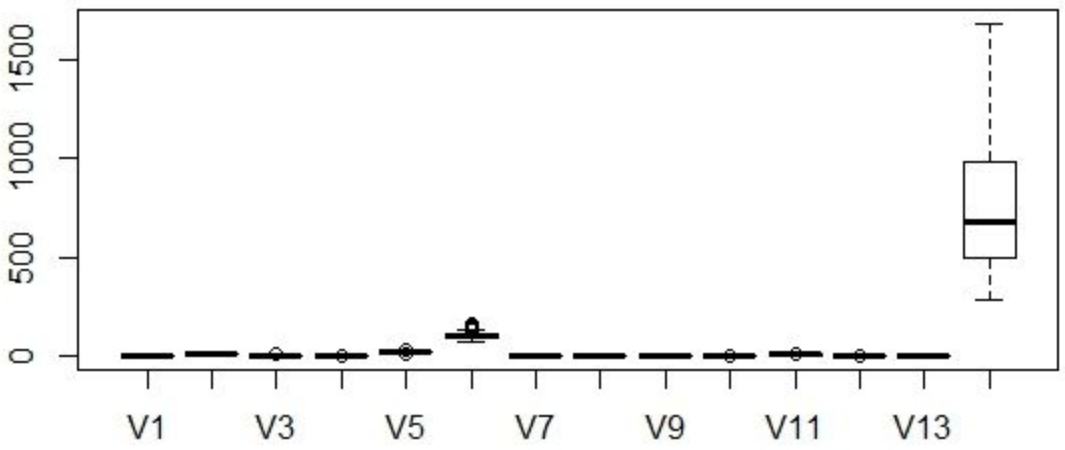
V1 = Alcohol
V2 = Malic acid
V3 = Ash
V4 = Alcalinity of ash
V5 = Magnesium
V6 = Total phenols
V7 = Flavanoids
V8 = Nonflavanoid phenols
V9 = Proanthocyanins

V10 = Color intensity
V11 = Hue
V12 = OD280/OD315 of diluted wines
V13 = Proline

2. Scatterplot Matrix



3. Boxplot:

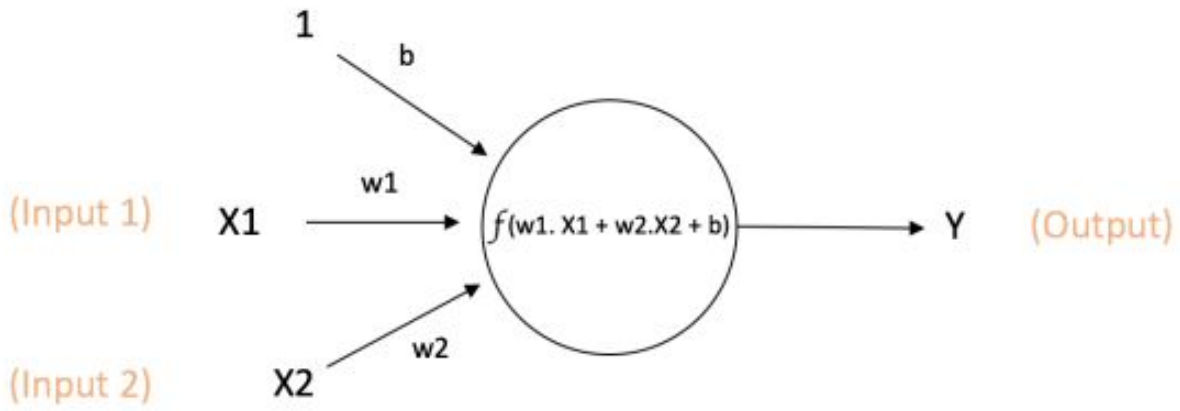


IV. NEURAL NETWORKS - A Quick Introduction

An Artificial Neural Network (ANN) is a computational model that is inspired by the way biological neural networks in the human brain process information. Artificial Neural Networks have generated a lot of excitement in Machine Learning research and industry, thanks to many breakthrough results in speech recognition, computer vision and text processing. In this blog post we will try to develop an understanding of a particular type of Artificial Neural Network called the Multi Layer Perceptron.

A Single Neuron

The basic unit of computation in a neural network is the **neuron**, often called a **node** or **unit**. It receives input from some other nodes, or from an external source and computes an output. Each input has an associated **weight** (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function ***f*** (defined below) to the weighted sum of its inputs as shown in Figure 1 below:



$$\text{Output of neuron} = Y = f(w1.X1 + w2.X2 + b)$$

Figure 1: a single neuron

The above network takes numerical inputs **X1** and **X2** and has weights **w1** and **w2** associated with those inputs. Additionally, there is another input **1** with weight **b** (called the **Bias**) associated with it. We will learn more details about role of the bias later.

The output **Y** from the neuron is computed as shown in the Figure 1. The function ***f*** is non-linear and is called the **Activation Function**. The purpose of the activation function is to introduce non-linearity into the output of a neuron. This is important because most real world data is non linear and we want neurons to *learn* these non linear representations.

Every activation function (or *non-linearity*) takes a single number and performs a certain fixed mathematical operation on it [2]. There are several activation functions you may encounter in practice:

- **Sigmoid:** takes a real-valued input and squashes it to range between 0 and 1
$$\sigma(x) = 1 / (1 + \exp(-x))$$
- **tanh:** takes a real-valued input and squashes it to the range [-1, 1]
$$\tanh(x) = 2\sigma(2x) - 1$$
- **ReLU**: ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero (replaces negative values with zero)
$$f(x) = \max(0, x)$$

The below figures [2] show each of the above activation functions.

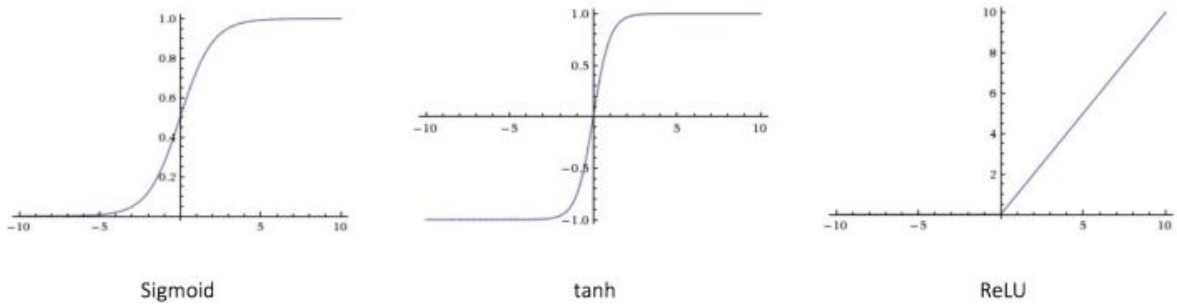


Figure 2: different activation functions

Importance of Bias: The main function of Bias is to provide every node with a trainable constant value (in addition to the normal inputs that the node receives). See this link to learn more about the role of bias in a neuron.

Feedforward Neural Network

The feedforward neural network was the first and simplest type of artificial neural network devised [3]. It contains multiple neurons (nodes) arranged in **layers**. Nodes from adjacent layers have **connections** or **edges** between them. All these connections have **weights** associated with them.

An example of a feedforward neural network is shown in Figure 3.

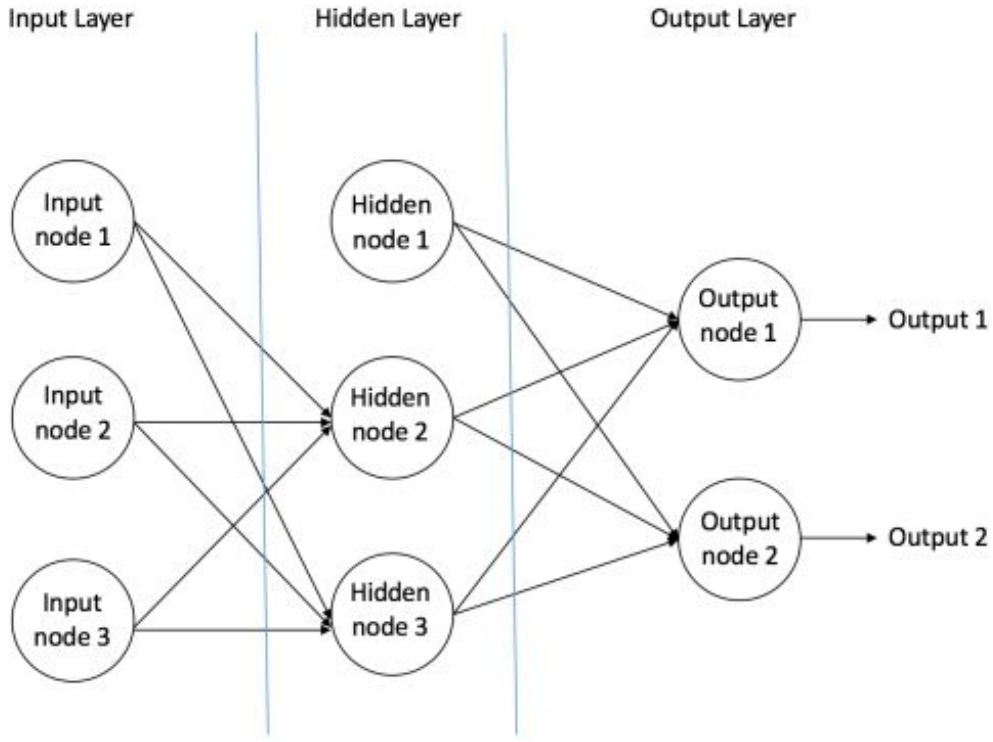


Figure 3: an example of feedforward neural network

A feedforward neural network can consist of three types of nodes:

1. **Input Nodes –** The Input nodes provide information from the outside world to the network and are together referred to as the "Input Layer". No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes.
2. **Hidden Nodes –** The Hidden nodes have no direct connection with the outside world (hence the name "hidden"). They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a "Hidden Layer". While a feedforward network will only have a single input layer and a single output layer, it can have zero or multiple Hidden Layers.
3. **Output Nodes –** The Output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network to the outside world.

In a feedforward network, the information moves in only one direction – forward – from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network [3] (this property of feed forward networks is different from Recurrent Neural Networks in which the connections between the nodes form a cycle).

Two examples of feedforward networks are given below:

1. **Single Layer Perceptron** – This is the simplest feedforward neural network [4] and does not contain any hidden layer. You can learn more about Single Layer Perceptrons in [4], [5], [6], [7].
2. **Multi Layer Perceptron** – A Multi Layer Perceptron has one or more hidden layers. We will only discuss Multi Layer Perceptrons below since they are more useful than Single Layer Perceptons for practical applications today.

Multi Layer Perceptron

A Multi Layer Perceptron (MLP) contains one or more hidden layers (apart from one input and one output layer). While a single layer perceptron can only learn linear functions, a multi layer perceptron can also learn non – linear functions.

Figure 4 shows a multi layer perceptron with a single hidden layer. Note that all connections have weights associated with them, but only three weights (w0, w1, w2) are shown in the figure.

Input Layer: The Input layer has three nodes. The Bias node has a value of 1. The other two nodes take X1 and X2 as external inputs (which are numerical values depending upon the input dataset). As discussed above, no computation is performed in the Input layer, so the outputs from nodes in the Input layer are 1, X1 and X2 respectively, which are fed into the Hidden Layer.

Hidden Layer: The Hidden layer also has three nodes with the Bias node having an output of 1. The output of the other two nodes in the Hidden layer depends on the outputs from the Input layer (1, X1, X2) as well as the weights associated with the connections (edges). Figure 4 shows the output calculation for one of the hidden nodes (highlighted). Similarly, the output from other hidden node can be calculated. Remember that ***f*** refers to the activation function. These outputs are then fed to the nodes in the Output layer.

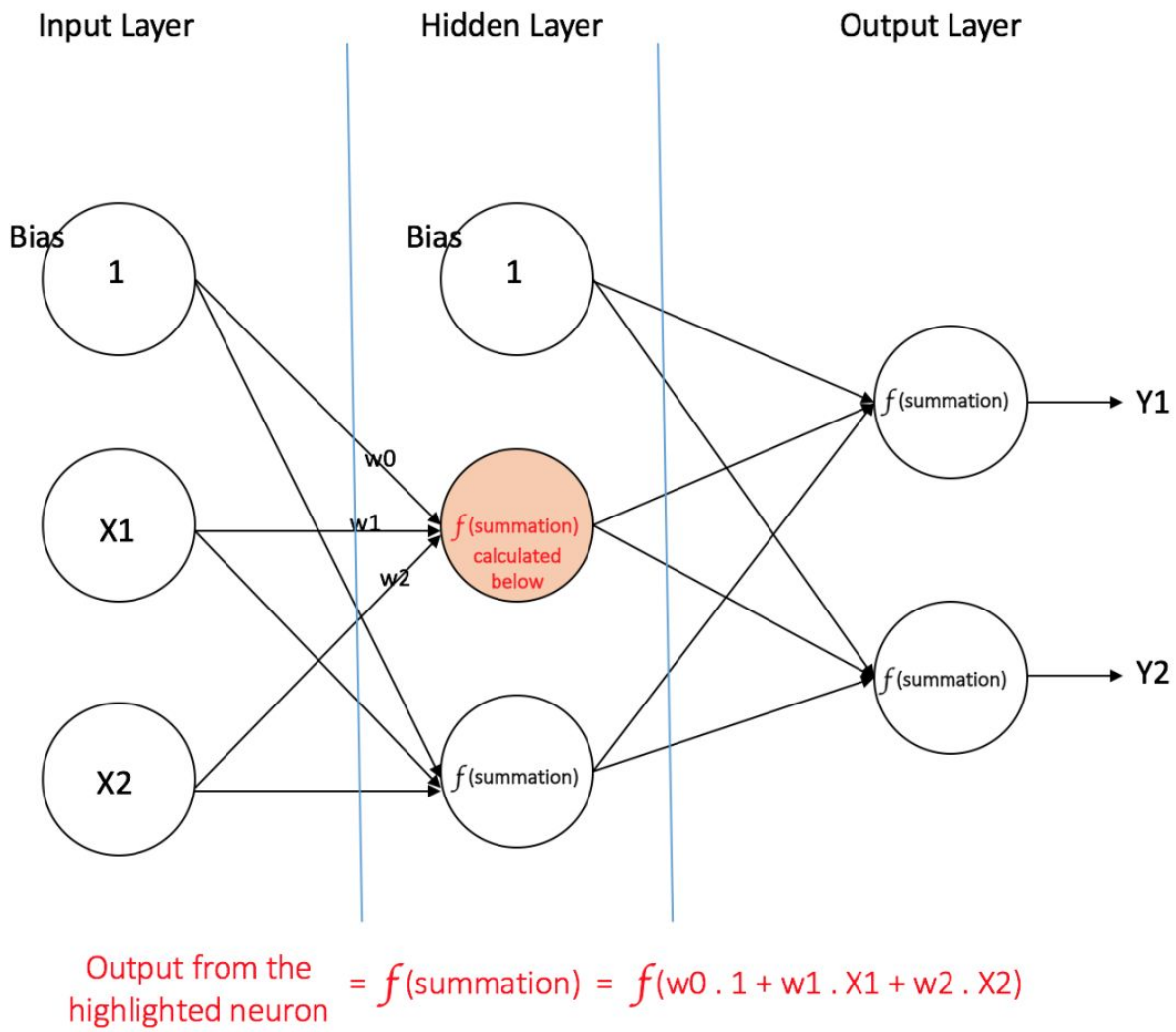


Figure 4: a multi layer perceptron having one hidden layer

Output Layer: The Output layer has two nodes which take inputs from the Hidden layer and perform similar computations as shown for the highlighted hidden node. The values calculated (Y1 and Y2) as a result of these computations act as outputs of the Multi Layer Perceptron.

Given a set of features **X = (x1, x2, …)** and a target **y**, a Multi Layer Perceptron can learn the relationship between the features and the target, for either classification or regression.

Lets take an example to understand Multi Layer Perceptrons better. Suppose we have the following student-marks dataset:

Hours Studied	Mid Term Marks	Final Term Result
35	67	1 (Pass)
12	75	0 (Fail)
16	89	1 (Pass)
45	56	1 (Pass)
10	90	0 (Fail)

The two input columns show the number of hours the student has studied and the mid term marks obtained by the student. The Final Result column can have two values 1 or 0 indicating whether the student passed in the final term. For example, we can see that if the student studied 35 hours and had obtained 67 marks in the mid term, he / she ended up passing the final term.
Now, suppose, we want to predict whether a student studying 25 hours and having 70 marks in the mid term will pass the final term.

Hours Studied	Mid Term Marks	Final Term Result
25	70	?

This is a binary classification problem where a multi layer perceptron can learn from the given examples (training data) and make an informed prediction given a new data point. We will see below how a multi layer perceptron learns such relationships.

V. OUR MODEL

1. Normalization

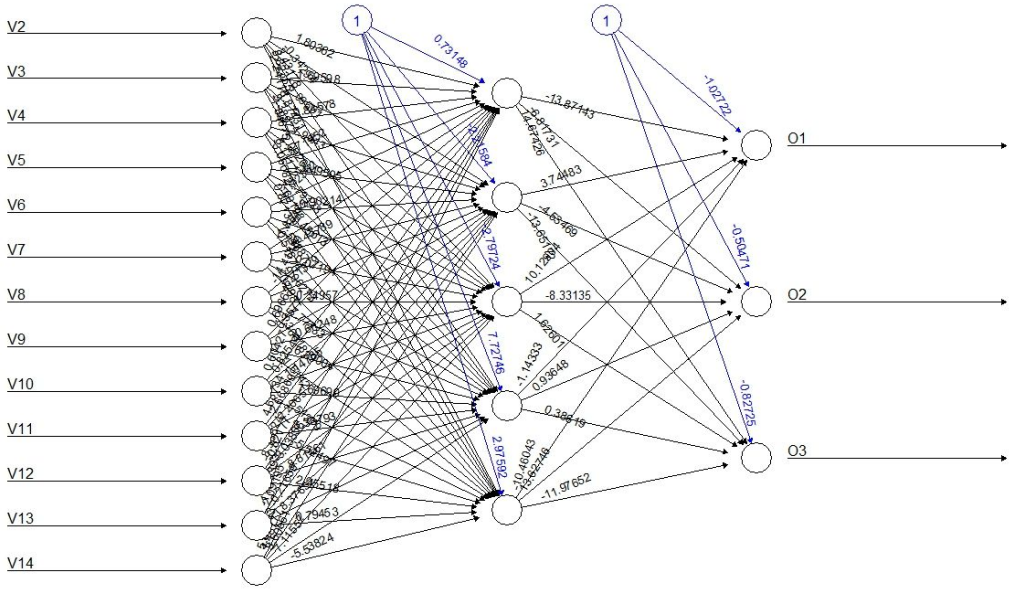
It is important to normalize data before training a neural network on it. The neural network may have difficulty converging before the maximum number of iterations allowed if the data is not normalized. There are a lot of different methods for normalization of data. We will use the built-in scale() function in R to easily accomplish this task.
Usually it is better to scale the data from 0 to 1, or -1 to 1. We can specify the center and scale as additional arguments in the scale() function.

2. Splitting Dataset into Train and Test Dataset

We split the dataset into Training and Testing dataset. The Training dataset is used to train the neural network and the test dataset is used to test the accuracy of our predictions. We set a definite seed for randomly splitting the dataset so that we can regenerate the split.

3. Training the Model

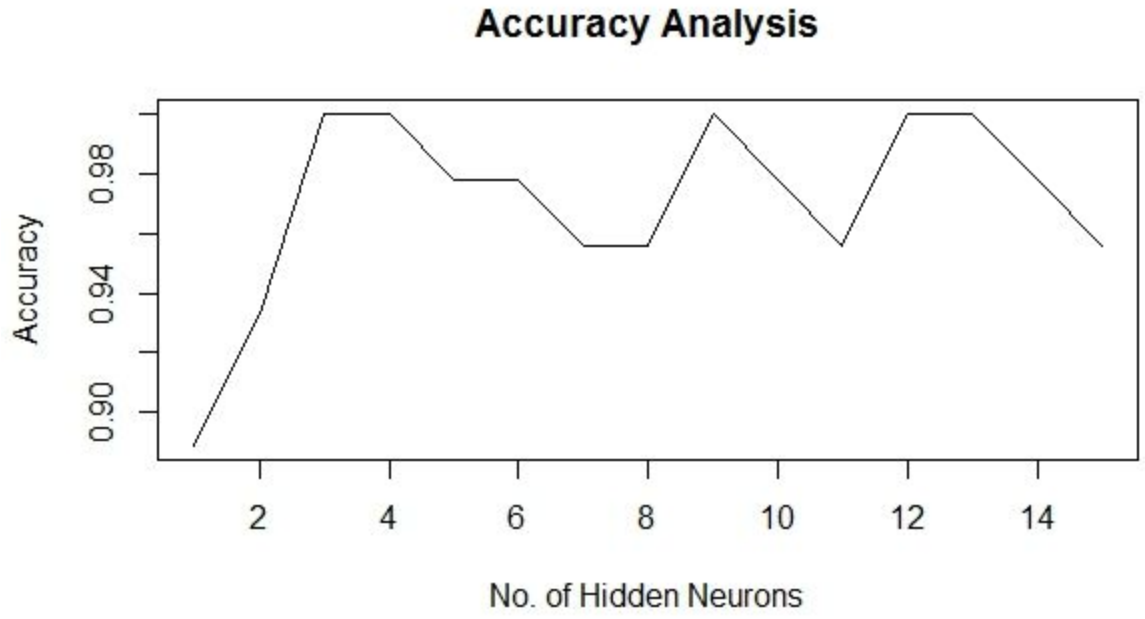
Our Neural Network models makes use of only one hidden layer. We are going to analyze different number of hidden neurons in the neural networks. Right now, we have 13 input neurons the 13 attributes, 5 hidden neurons and 3 output neurons which gives us class probabilities for each wine class.



4. Predictions and Accuracy

Out of 45 test data samples, we were able to correctly predict 44 correct. That gives us an accuracy of 0.9777777778%.

VI. ACCURACY



VIII. RELEVANT PAPERS

1)
S. Aeberhard, D. Coomans and O. de Vel,
Comparison of Classifiers in High Dimensional Settings,
Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Technometrics).

The data was used with many others for comparing various
classifiers. The classes are separable, though only RDA
has achieved 100% correct classification.
(RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))

(All results using the leave-one-out technique)

(2)
S. Aeberhard, D. Coomans and O. de Vel,
"THE CLASSIFICATION PERFORMANCE OF RDA"
Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Journal of Chemometrics).

Here, the data was used to illustrate the superior performance of
the use of a new appreciation function with RDA.