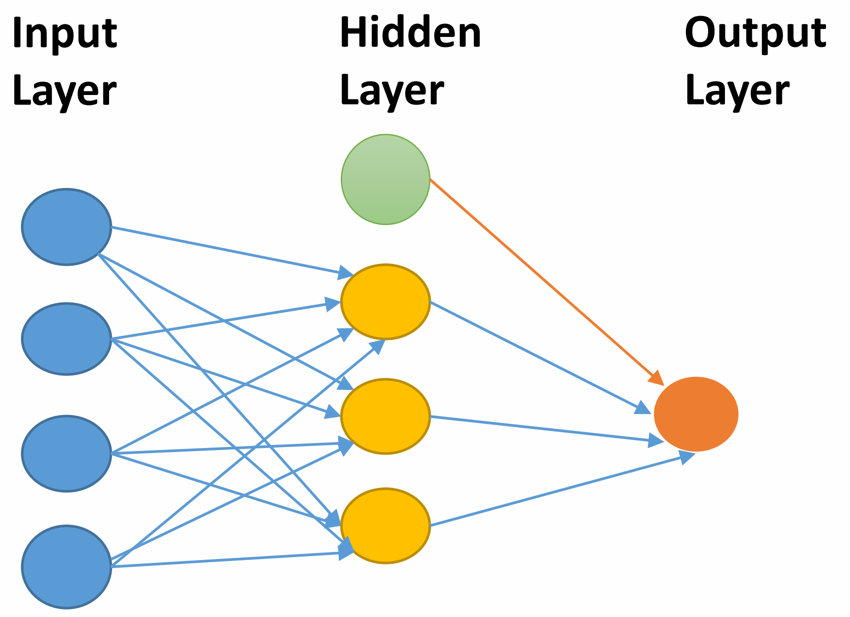
Sane's STATS Academy of Statistics

Neural Networks

Feed Forward

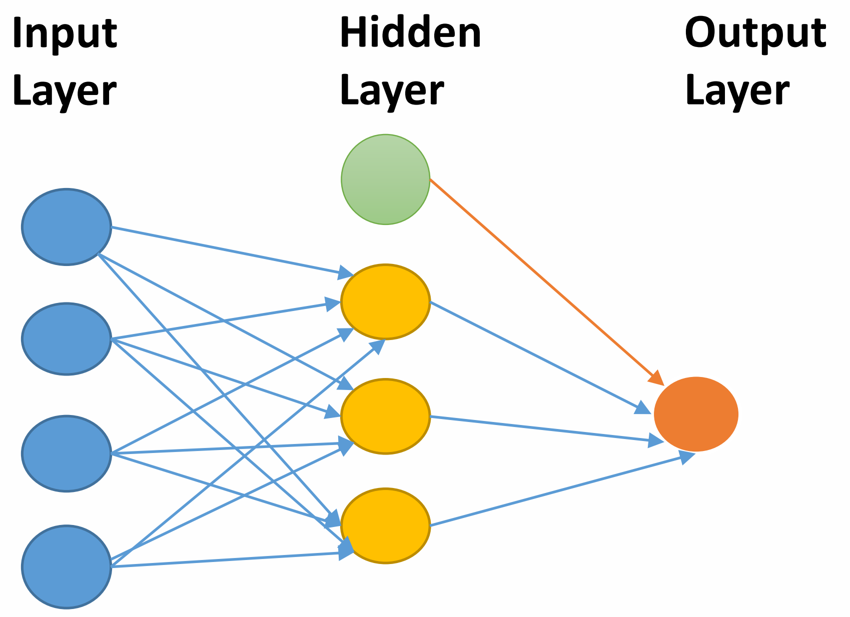
Neural Network

- A neural network is constructed from a number of interconnected nodes called neurons.
- Neurons are arranged into input layer, hidden layer and output layer.
- Input layer corresponds to our predictors / features and Output layer to our response variable(s).



Multi-Layer Perceptron

- The neural network with input layer, one or more hidden layers and one output layer is called multi-layer perceptron (MLP).
- Invented by Frank Rosenblatt in 1957
- MLP given below has 4 input nodes, 3 hidden nodes and one output node



Functioning of MLP

- Input layer neurons receive incoming information from the data which they process and distribute it to the hidden layers.
- That information, in turn is processed by hidden layers and is passed to the output neurons.
- The information in this artificial neural network is processed in terms of one activation function. This function actually imitates the brain neurons.
- Each neuron contains a value of activation functions and a threshold value.
- The threshold value is the minimum value that must be possessed by the input so that it can be activated.
- The task of the neuron is perform a weighted sum of all the input signals and apply activation function on the sum before passing it to the next(hidden or output) layer.

Weighted Sum

- Say that, we have values a_1, a_2, a_3, a_4 for input and weights as w_1, w_2, w_3, w_4 as the input to one of the hidden layer neuron say n_j, then the weighted sum is represented as

$$S_j = \sum_{i=1}^{4} w_i a_i + b_j$$

where b_j: bias due to node

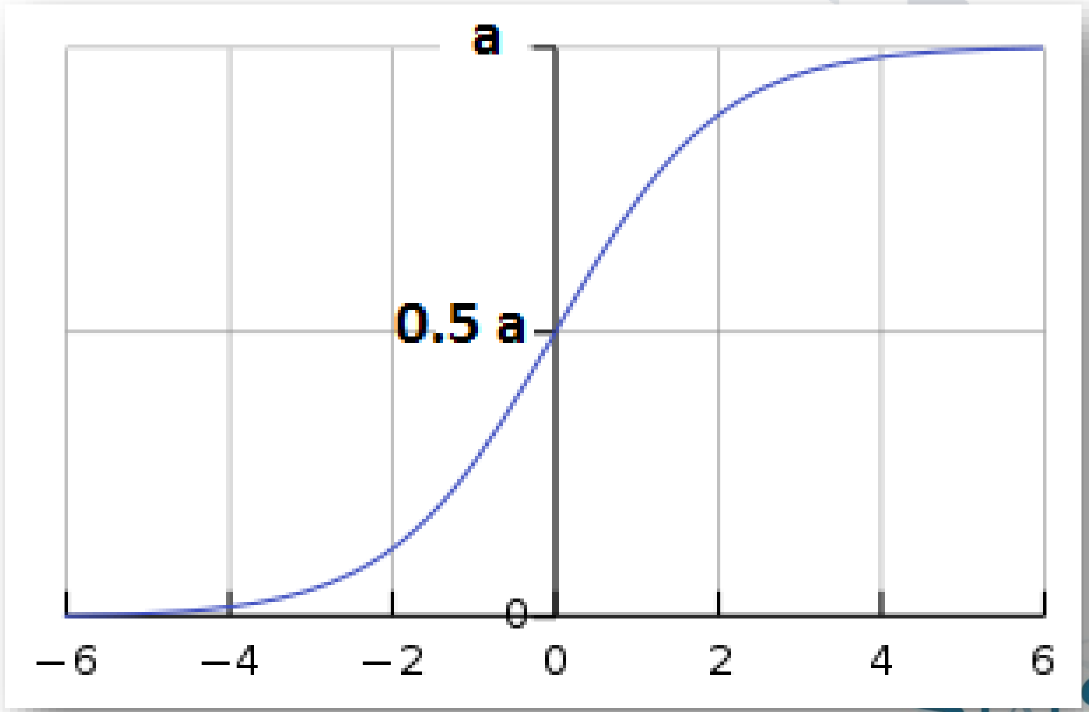
Activation Functions

- These functions are needed to introduce a non-linearity into the network
- Activation function is applied and that output is passed to the next layer
- It is usually chosen such that its output limits between a certain range line (-1,1) or (0,1)
- Possible Functions
  - Sigmoid
  - Hyperbolic Tangent

Sigmoid Function

- It is differentiable
- It produces output between 0 and 1
- Function form is

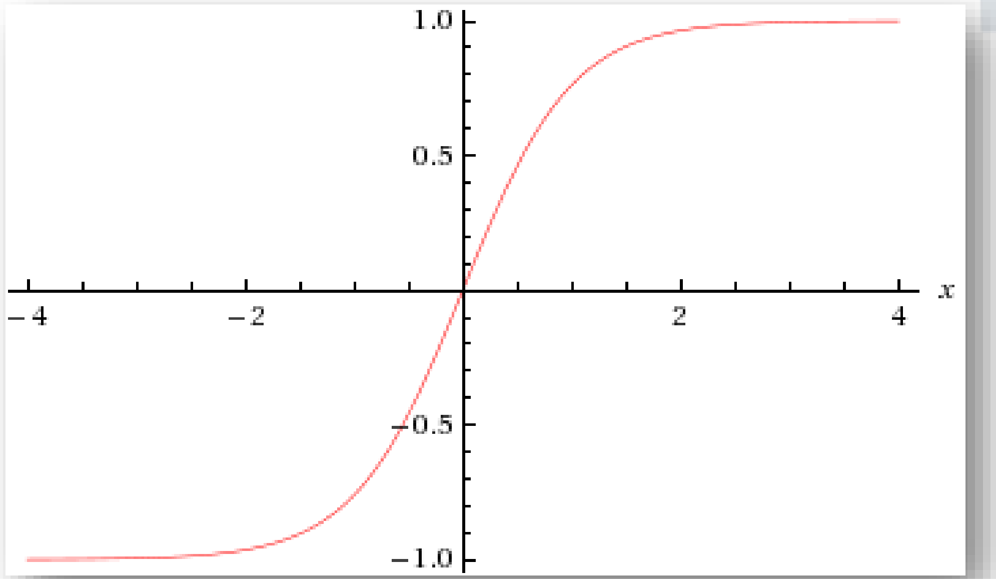
$$S(x) = \frac{1}{1 + e^{-ax}}$$



Hyperbolic Tangent Function

- This is also differentiable
- Produces output between -1 and 1
- Function Form is

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



Learning of Neural Network

- All the learning algorithms of neural network train the network by iteratively modifying the connection weights until the error desired output and the calculated output does not drop to a desired level
- Popularly known algorithm we are going to learn is backpropagation algorithm which uses gradient descent method

Mathematical Significance of Backpropagation

- According to Calculus, chain of functions can be differentiated using the following identity, called the chain rule : f(g(x)) = f'(g(x)) * g'(x).
- If we apply the same chain rule to the computation of the gradient values of the neural network, then it is called backpropagation (reverse-mode differentiation)
- Backpropagation starts with the error in the last layer and traverses back up to the first layer

Backpropagation Algorithm

1. **Initialization of weights**: Weights are randomly assigned usually between 0 and 1
2. **Feed Forward**: Information(calculated data) is passed on to the next layer starting from input layers to hidden layers to output layers with the help of activation function
3. **Error Assessment**: The output of the network is assessed relative to the existing output. If error is below a permissible value then algorithm is terminated, otherwise proceed to step 4

Backpropagation Algorithm

4. **Propagation**: The error of the output layer is used to modify the weights. The errors are propagated backwards through the network thereby calculating the gradient of change in error
5. **Adjust**: Adjustments are made to the weights using the gradients of change with the goal of reducing the error. The weights and biases are adjusted by a factor based on the derivative of the activation function and the errors. Now Step 2 is executed.

Neural Networks in R

- There are several packages and functions implementing different types of neural networks in R. We will see the examples of the package nnet
- There are many other packages like neuralnet, neural, AMORE etc.

Package nnet

- Package nnet comes already installed with the default installation of R
- Function nnet implements single hidden layer algorithm and predict can be used to predict the outcomes on the test set

Syntax : nnet(formula , data, size, maxit, …)

Where

formula : formula expressed in formula syntax

data : matrix or data frame of train set

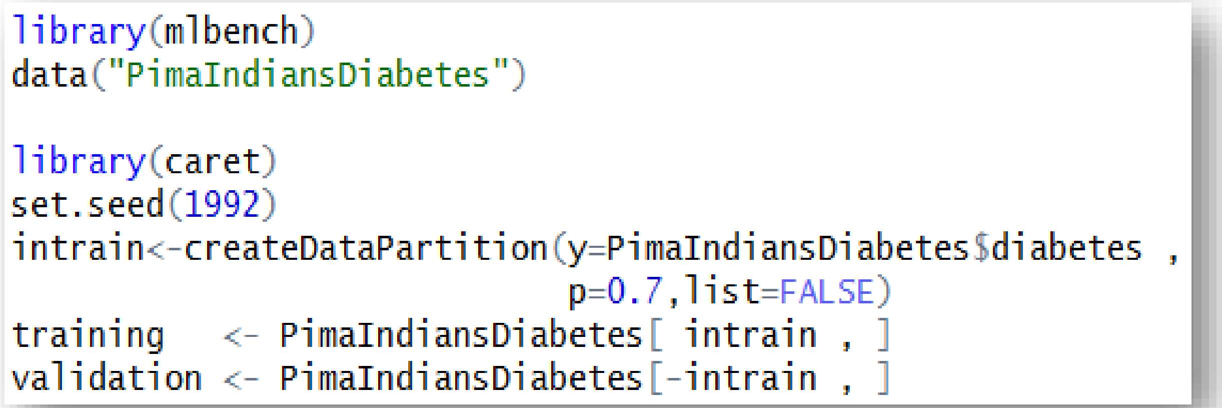
size : number of units in the hidden layer

maximum number of iterations. Default 100

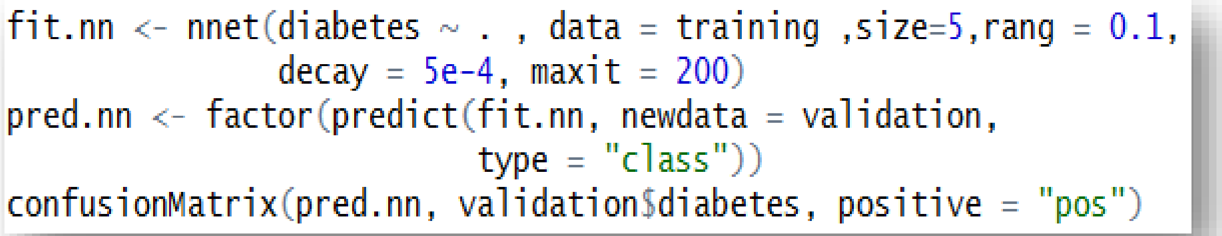
Example: Pima Indian Diabetes

- Data contains demographic profile and also contains one target variable(diabetes) which show whether patient shows signs of diabetes.
- Data Contains following variables:
  1. Number of times pregnant
  2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
  3. Diastolic blood pressure (mm Hg)
  4. Triceps skin fold thickness (mm)
  5. 2-Hour serum insulin (mu U/ml)
  6. Body mass index (weight in kg/(height in m)^2)
  7. Diabetes pedigree function
  8. Age (years)
  9. Class variable (neg or pos)

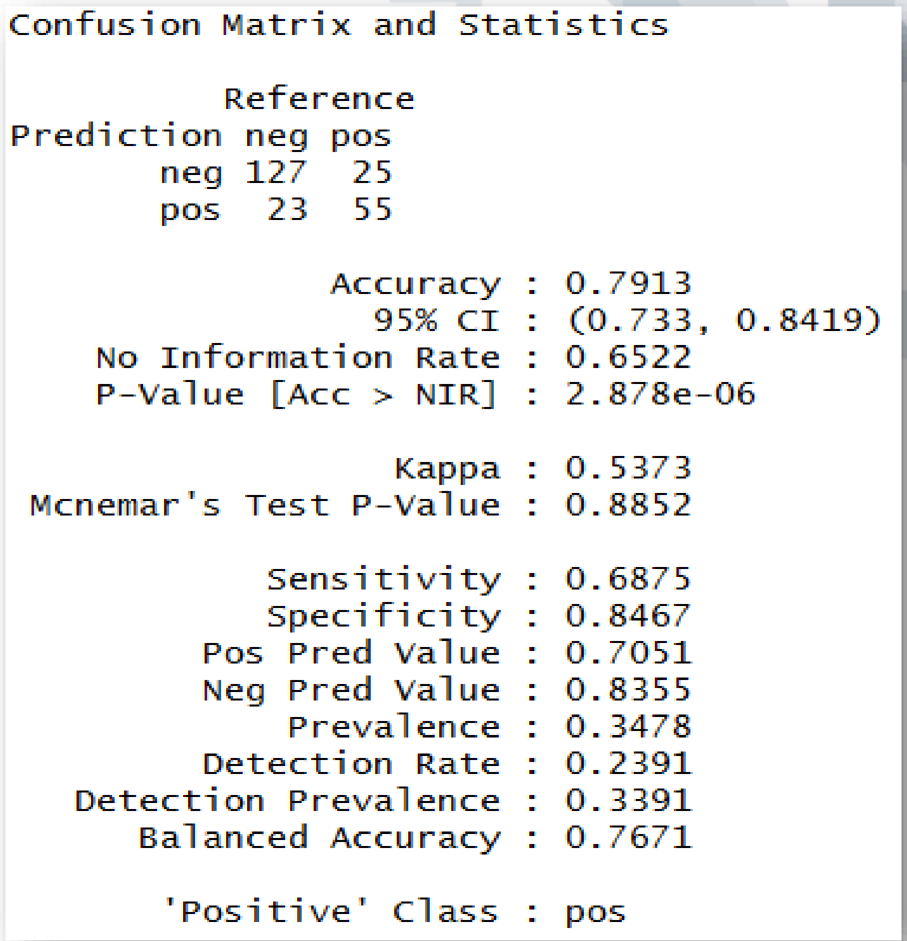
R Program & Output - nnet



```r
library(mlbench)
data("PimaIndiansDiabetes")

library(caret)
set.seed(1992)
intrain<-createDataPartition(y=PimaIndiansDiabetes$diabetes ,
                             p=0.7,list=FALSE)
training   <- PimaIndiansDiabetes[ intrain , ]
validation <- PimaIndiansDiabetes[-intrain , ]
```



```r
fit.nn <- nnet(diabetes ~ . , data = training ,size=5,rang = 0.1,
               decay = 5e-4, maxit = 200)
pred.nn <- factor(predict(fit.nn, newdata = validation,
                          type = "class"))
confusionMatrix(pred.nn, validation$diabetes, positive = "pos")
```

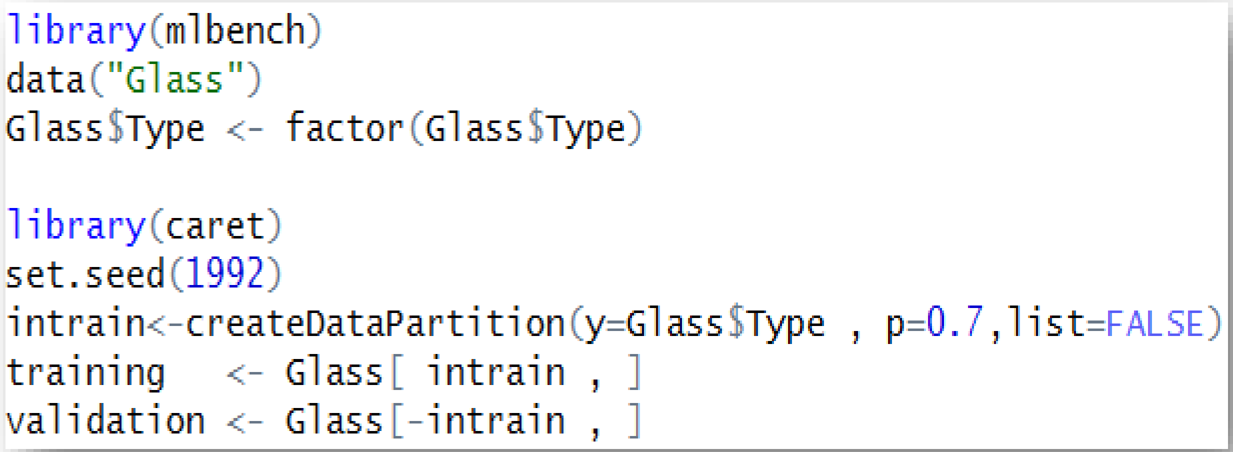


```
Confusion Matrix and Statistics

          Reference
Prediction neg pos
       neg 127  25
       pos  23  55

               Accuracy : 0.7913
                 95% CI : (0.733, 0.8419)
    No Information Rate : 0.6522
    P-Value [Acc > NIR] : 2.878e-06

                  Kappa : 0.5373
 Mcnemar's Test P-Value : 0.8852

            Sensitivity : 0.6875
            Specificity : 0.8467
         Pos Pred Value : 0.7051
         Neg Pred Value : 0.8355
             Prevalence : 0.3478
         Detection Rate : 0.2391
   Detection Prevalence : 0.3391
      Balanced Accuracy : 0.7671

       'Positive' Class : pos
```

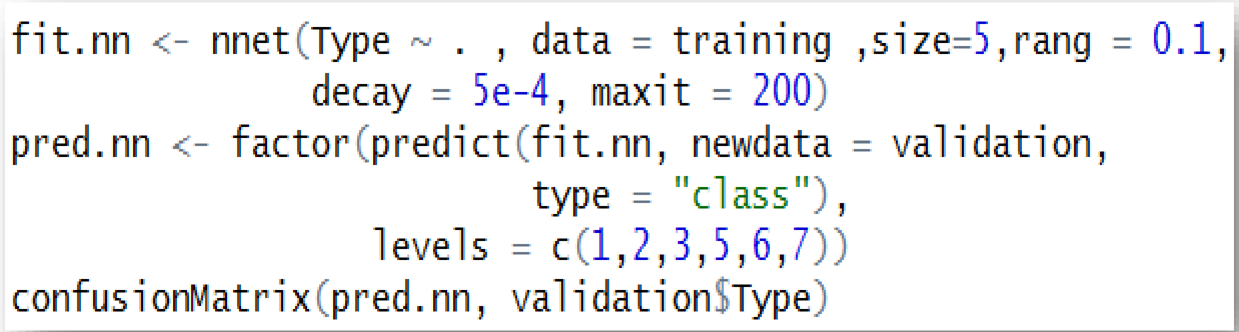
Example

- Consider the dataset Glass in the package **mlbench**
- A data frame with 214 observation containing examples of the chemical analysis of 6 different types of glass.
- The problem is to forecast the type of class on basis of the chemical analysis.
- The study of classification of types of glass was motivated by criminological investigation.
- At the scene of the crime, the glass left can be used as evidence (if it is correctly identified!)

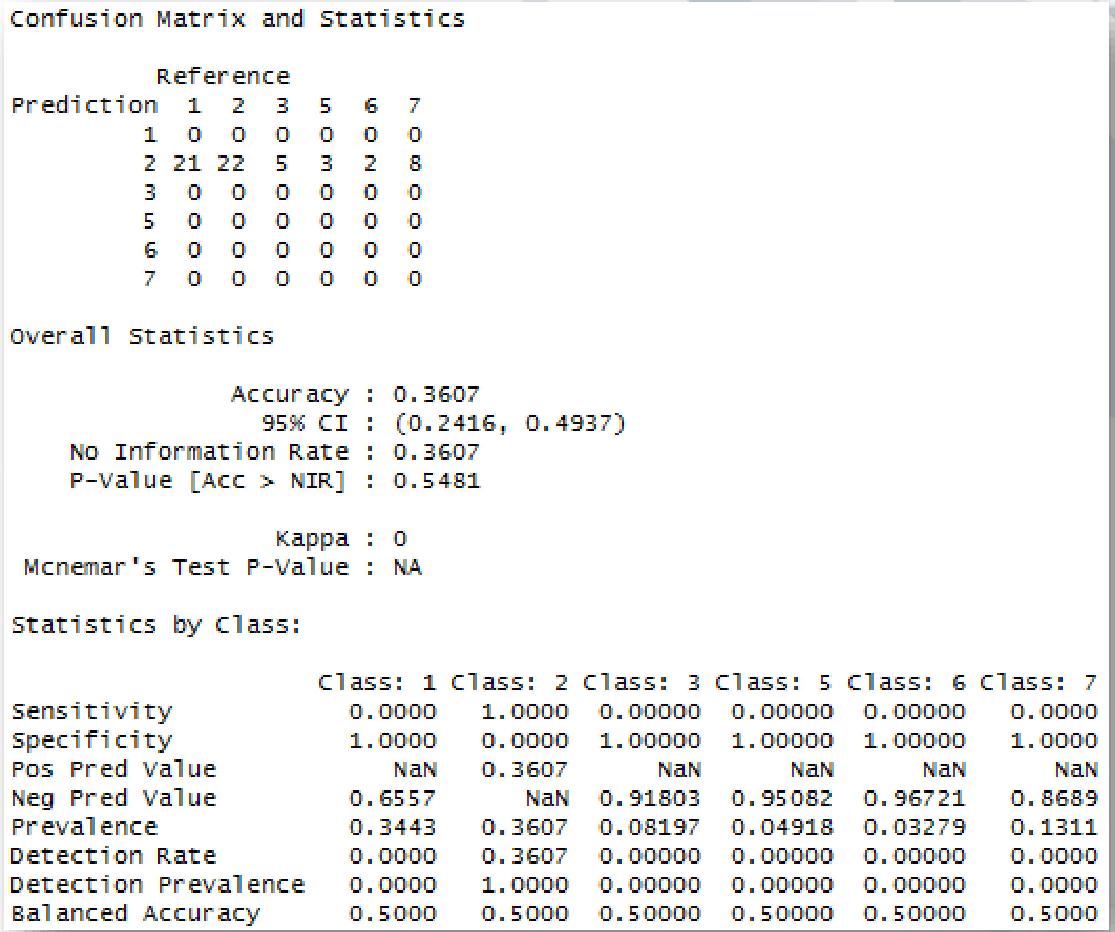
R Program & Output



```
library(mlbench)
data("Glass")
Glass$Type <- factor(Glass$Type)

library(caret)
set.seed(1992)
intrain<-createDataPartition(y=Glass$Type , p=0.7,list=FALSE)
training   <- Glass[ intrain , ]
validation <- Glass[-intrain , ]
```



```
fit.nn <- nnet(Type ~ . , data = training ,size=5,rang = 0.1,
               decay = 5e-4, maxit = 200)
pred.nn <- factor(predict(fit.nn, newdata = validation,
                          type = "class"),
                  levels = c(1,2,3,5,6,7))
confusionMatrix(pred.nn, validation$Type)
```



```
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3  5  6  7
         1  0  0  0  0  0  0
         2 21 22  5  3  2  8
         3  0  0  0  0  0  0
         5  0  0  0  0  0  0
         6  0  0  0  0  0  0
         7  0  0  0  0  0  0

Overall Statistics

               Accuracy : 0.3607
                 95% CI : (0.2416, 0.4937)
    No Information Rate : 0.3607
    P-Value [Acc > NIR] : 0.5481

                  Kappa : 0
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
Sensitivity            0.0000   1.0000  0.00000  0.00000  0.00000   0.0000
Specificity            1.0000   0.0000  1.00000  1.00000  1.00000   1.0000
Pos Pred Value            NaN   0.3607      NaN      NaN      NaN      NaN
Neg Pred Value         0.6557      NaN  0.91803  0.95082  0.96721   0.8689
Prevalence             0.3443   0.3607  0.08197  0.04918  0.03279   0.1311
Detection Rate         0.0000   0.3607  0.00000  0.00000  0.00000   0.0000
Detection Prevalence   0.0000   1.0000  0.00000  0.00000  0.00000   0.0000
Balanced Accuracy      0.5000   0.5000  0.50000  0.50000  0.50000   0.5000
```