|  |
| --- |
| UMUC |
| Neural Network Analysis Using R |
| Week 5 Required Exercise |
|  |
| **DBST 667** |
|  |

|  |
| --- |
| In this exercise, you will build the network model to classify the diabetes test result as positive or negative based on number of pregnancies, age, plasma concentration, diastolic blood pressure, triceps skinfold thickness, and body mass index . An exercise illustrates the steps for building and visualizing the network, and for evaluating the model accuracy. |

Table of Contents

[Diabetes Data 2](#_Toc429330584)

[Launch the Program 2](#_Toc429330585)

[Run the Method 5](#_Toc429330586)

[Network Visualization 9](#_Toc429330587)

[Model Evaluation 11](#_Toc429330588)

[Further Exploration 12](#_Toc429330589)

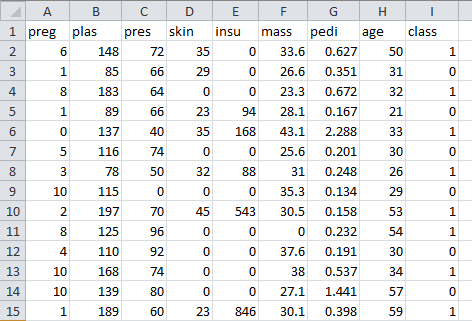
Neural Network Method to Diagnose Diabetes

The purpose of this exercise is to build a Multilayer Perceptron network model for predicting the diabetes test results, when the number of pregnancies, plasma concentration, diastolic blood pressure, triceps skinfold thickness, age, and body mass index are known. The analyses include the result interpretation, accuracy evaluation of a model and model visualization.

Your results might be slightly different depending on R and R Studio version, and depending on your operating system.

# Diabetes Data

Figure 1 shows the partial content of the diabetes.csv file. The column headings in the first row of the file are the diabetes attribute names called variables. The remaining 768 rows are the data, where each row is a single patient record.



**Independent Variables**

**Preg**=number of pregrancies

**Plas** =plasma concentration

**Pres** =diastolic blood pressure

**Skin** =triceps skinfold thickness

I**nsu** = 2 hour serum insulin

**Mass**=body mass index

**Pedi** = pedigree function

**Age** =individual’s age

**Dependent variable**

**Class**= 1- tested positive; 0-tested negative

Figure 1: Partial diabetes Data

# Launch the Program

Launch R studio program to open an interface on Figure 2.

To run the neural network method, you need to install the neuralnet package if you have not installed it before. Enter the following command into an application console and hit enter.

install.packages("neuralnet")

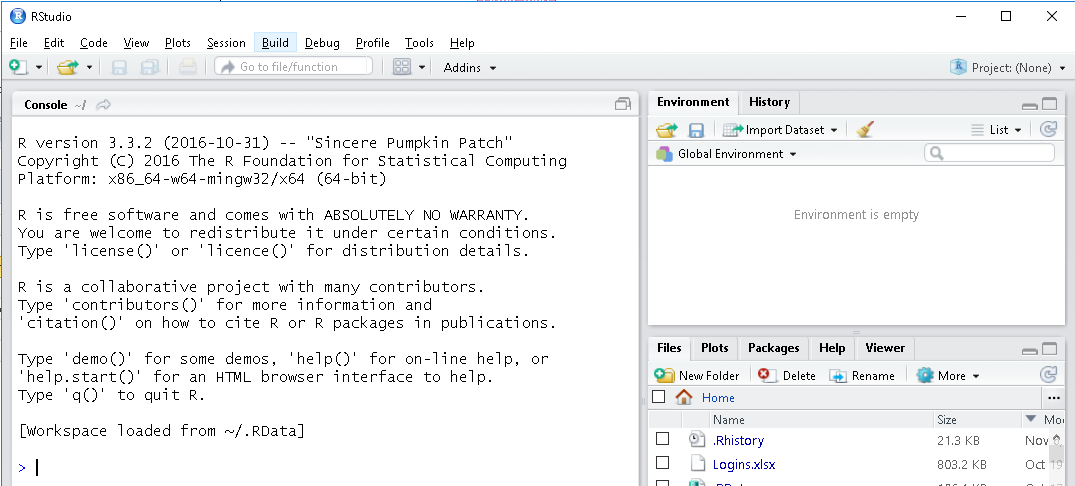
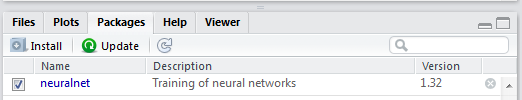


Figure 2: R Studio Interface

Before a package can be used in the current session, it needs to be loaded into memory.



Select the Packages tab at the bottom right window of an interface. Check the checkbox next to neuralnet on Figure 3.

Figure 3: Select the Package to Load

Suppose that the diabetes.csv file we want to load is in the E:/Datasets folder. To set the working directory to E:/Datasets, enter the following setw command in the console window and hit the enter key. The directory path is specified in parentheses enclosed in double quotes.

setwd("E:/Datasets")

Use **read.csv** command to read the diabetes file content into a data frame variable called diabetes. The first input parameter for the read.csv function is the data file name enclosed in double quotes. The second parameter, head=TRUE, specifies that the first row in the file contains the column headers. The sep parameter is the columns delimiter enclosed in double quotes. For example, sep=“,” means that the values in each data row are comma delimited.

The values delimiter

Command to Read from CSV file

diabetes<-read.csv(file="diabetes.csv", head=TRUE, sep=",")

Data frame name – stores data from the first sheet in CSV file

Read the column headings from the first row

File Name

Run the head command to preview the first 6 data rows. The head command takes the dataset name and the number of rows to preview as an input. When the number of rows argument is omitted, the first 6 rows are displayed.

head(diabetes)

**Figure 4** shows the first 6 rows in the diabetes dataset.

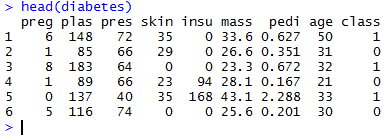


Figure 4: First 6 Rows in diabetes Dataset

Run the summary command to verify the all variables are numeric, and that the data does not have any missing values.

summary(diabetes)

Figure 5 shows the statistics for all variables in the diabetes dataset. All variables are numeric and do not have missing values.

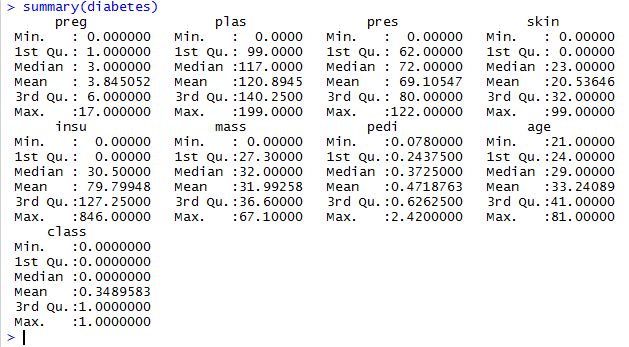


Figure 5: Diabetes Variables Statistics

# Run the Method

The method result in part depends on initial weights assignment to the input variables. The initial weights are assigned randomly, and the assigned initial weights may vary when the method is rerun, even with the same input. Setting the seed value enables reproducing the results when the method is rerun.

Run the following command to set the seed value.

set.seed(12345)

We run the neuralnet function and store the results in a variable called nn. The method input parameters:

**Formula** - specifies the dependent variable on the left hand side and independent variables on the right hand side.

class~preg+plas+pres+skin+insu+mass+pedi+age

Independent variables

Dependent variable

**Dataset name** – in this case, the dataset is diabetes

**Hidden** – the number of hidden layers and the number of nodes in each layer. Value 2 mans the network has one hidden layer with 2 nodes. To specify more than one hidden layer, use a vector. For instance, vector c(4, 2) means two hidden layers with the four nodes in the first hidden layer and 2 nodes in the second hidden layer.

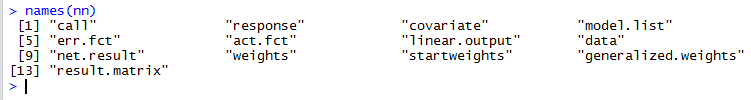
Run the following command to build the network.

nn<-neuralnet(formula = class~preg+plas+pres+skin+insu+mass+pedi+age, data = diabetes, hidden=2)

Run the names command to display the network properties.

names(nn)

Figure 6 shows the available neural network properties.

Figure 6: Neural Network Properties

Enter nn at the prompt to view the output stored in nn variable. The output on Figure 7 includes the call, numberof repetions, error, and number of steps information. Call is the command we ran to create the model.

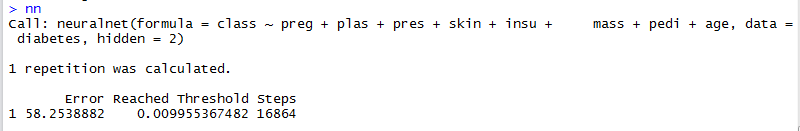


Figure 7: Default Output Sections

To see the remaining output, enter the nn followed by the $ and by the outut property name. For instance, enter the following command to see only the command we ran to generate the model.

nn$call

Figure 8 shows the command we ran to build the model.

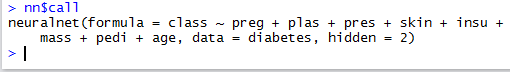


Figure 8: Call

To see the actual values of the dependent variable(s), called response, enter the following command.

nn$response

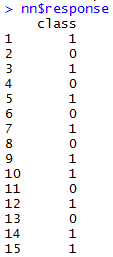


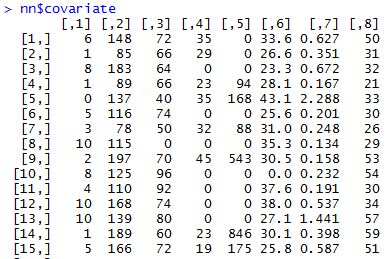
Figure 9 shows the actual diagnosis for the first 15 patients. The first column is the instance number, and the second column is a diagnosis.

Figure 9: Actual Diagnosis for 15 Records

To see the values of the input variables that were used to build the model, run the following command.

nn$covariate

Figure 10 shows the input values for the first 15 instances.



Row number

preg

plas

pres

skin

insu

mass

pedi

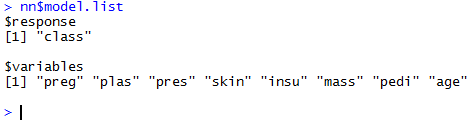
age

Figure 10: Network Input

To get the list of dependent and independent variables in the model, run the following command.

nn$model.list

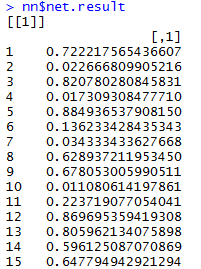
The response section at the top on Figure 11 is the list of dependent variables (left hand side of the model). The variables section is the list of independent variables (right hand side of the formula)



Independent Variable

Dependent Variable

Figure 11: Dependent and Independent Variables



Tested Positive

Tested Negative

To view the predicted output, run the following command. An output is the probability that the diabetes test is positive.

nn$net.result

Figure 12 shows the predicted probability of positive test for the first 15 patients.

The test is positive if the probability is 0.5 or above.

The test is negative if the probability is below 0.5

Figure 12: Predicted Probability that the Test is Positive

To display the predicted probabilities for the first 10 instances on Figure 13, run the following command. Note that nn$net.results is a list. The predicted probabilities are contained in the first element of a list. Hence, we specify the list element in the double squared brackets followed by the rows to display in the single square brackets.

nn$net.result[[1]][1:10]

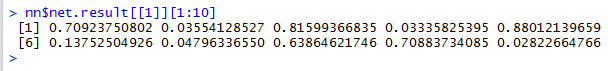


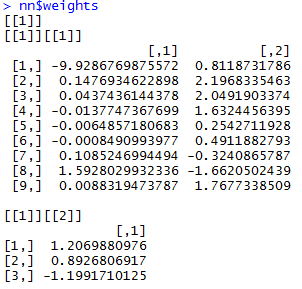
Figure 13: The First 10 Predicted Probabilities

Run the following command to display the network weights after the last method iteration.

nn$weights

The top section on Figure 14 shows the weights from input to the hidden layer nodes, including the weights for the node activation function intercepts. The column heading is the hidden layer node number.

The bottom section shows the weights from the hidden layer into the output layer, including the weights for the output node activation function intercept.



Weights from hidden layer to the output layer nodes.

Weight for the output nodes intercept

Weights from input layer to the hidden layer nodes.

Weights for the hidden nodes intercepts

Hidden layer nodes

Figure 14: Weights after Last Iteration

To see the weights on the first method iteration, run the following command.

nn$startweights

The top section on Figure 14 shows the initial weights from input to the hidden layer nodes, including the weights for the node activation function intercepts. The column heading is the hidden layer node number.

The bottom section shows the weights from the hidden layer into the output layer, including the weight for the output node function intercept.

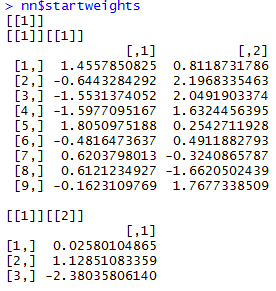
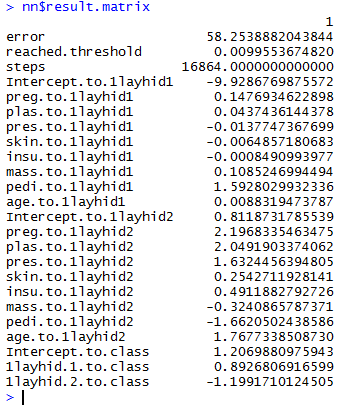


Figure 15: Initial Weights

You may also run the following command to see the number of trainings steps, the error, and the weights on Figure 16.

nn$result.matrix



Weights for the first hidden layer node

Weights for the second hidden layer node

Output node weights

Figure 16: Result Matrix

# Network Visualization

Run the plot function to graph the network. You may ignore the warning message on Figure 17.

plot(nn)



Figure 17: Warning Message

The plot on Figure 18 shows the input nodes, 2 hidden layer nodes, and the output nodes. Each input node is connected to each hidden layer node, and each connection line has a weight. Each hidden layer node is connected to an output layer node, and each connection line has a weight.

The nodes in the same layer are not connected. The nodes represented as number 1 inside the blue circle are the intercepts for the activation functions. Each intercept is connected to all nodes in the corresponding layer, and each connection has a different weight.

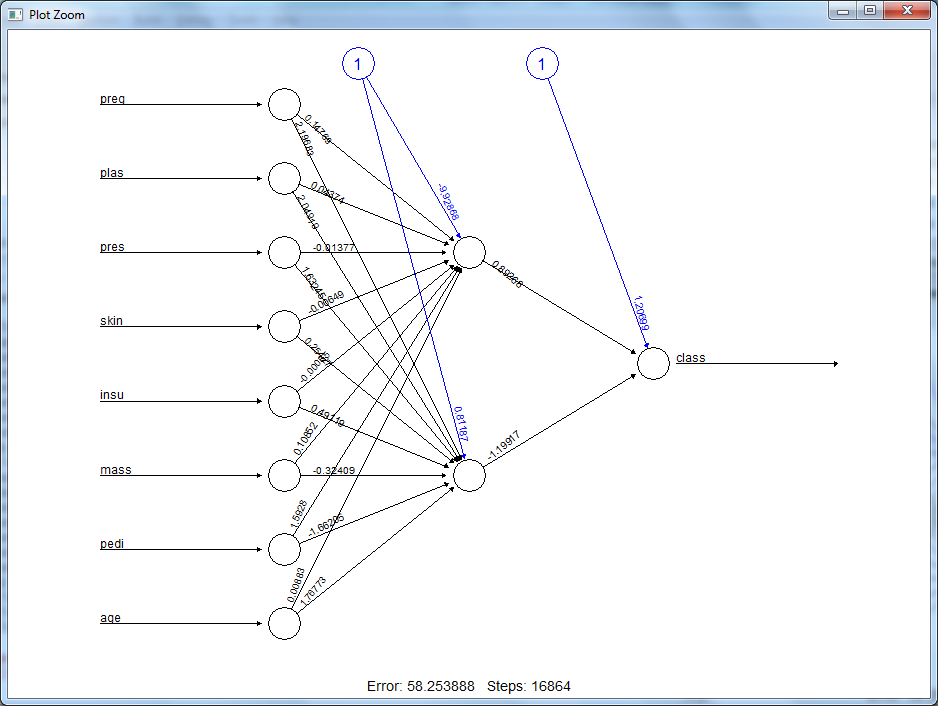


Figure 18: Network Model

# Model Evaluation

We need to build the confusion matrix that shows how many positive test predictions are actually positive and how many negative predictions are actually negative. To build the confusion matrix

Step1. Run compute command to generate the predicted positive test probabilities for all instances and store the predictions in a variable mypredict. The compute command input is the network name and the model input values.

mypredict<-compute(nn, nn$covariate)$net.result

Step 2. Use apply functionn to round the probabilities. If the probability is below 0.5, the predicted value is 0. If the predicted value is 0.5 or above, the predicted value is 1.

mypredict<-apply(mypredict, c(1), round)

Figure 19 shows the predicted diabetes test for all instances/patient records.

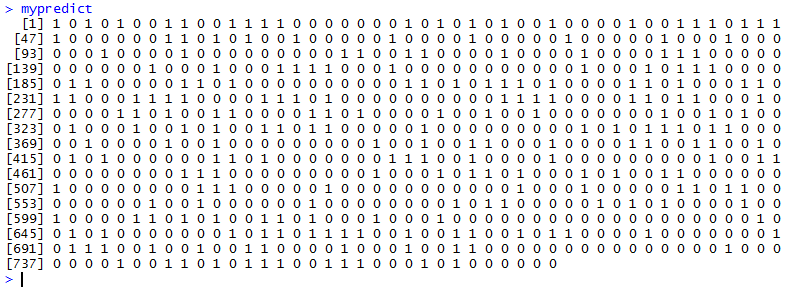


Figure 19: Predicted Values

Step 3 – Run the table command to build the cross-tabulation between predicted test and actual test.

table(mypredict, diabetes$class)

The confusion matrix on Figure 20 shows that 433 individuals have predicted and actual negative test. Hence, 433 predictions are true negatives.

The matrix shows that 165 individuals have predicted and actual positive test. Hene, 165 perdictions are true positive.

The sum of all numbers in a matrix is the number of instances. The number of correct predictions is the sum of numbers on diagonal from top left to bottom right. 422+165=585

The prediction accuracy is the sum of numbers on diagonal divided by the number of instances. 585/768=76%



Predicted Test

Actual Test

Figure 20: Confusion Matrix

# Further Exploration

What method parameters can be adjusted to improve the accuracy of a model?

How does running the method on a subset of independent variables affect the classification accuracy?