Wesley Clark

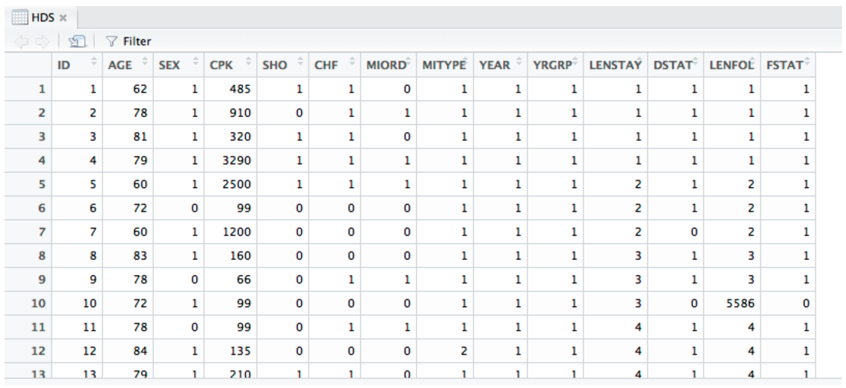
Neural Networking with Heart Disease Data

1) Introduction

In this assignment I will be exploring Neural Networks. Neural networks have become an essential concept intertwined with Data Mining techniques. There has been a paradigm shift favoring Neural Networks, as Neural Networks have outperformed old types of learning processes. This type of learning, is named after the human brain. It is designed to loosely mimic the performance of the brain, using neurons to connect with nodes acting like an elaborate tree. This is different method that has been used previously, and is gaining in popularity of recent years. One example is IBM's famous Deep Blue, the chess computer that famously beat the reigning world champion, Garry Kasparov. Deep Blue was programmed exhaustively. This is a painstaking form of programming, in which people coded every possible move into Deep Blue. Neural networks are much more efficient than this method, and require so much less programming making machine learning more applicable fora wide variety of situations.Each node may be weighted differently, such that the probability that a neuron travels to a node may change, depending on the weight. In order for the neuron to travel, just as in the brain, the activation energy required must be met. Although there have been several types of Artificial Neural Networks(ANN) that have been developed since its inception in the 1960's, multilayer neural networking has become one of the most successful applications specific to data mining. Single layer networks work with one layer of neurons. In many scenarios we examine, multilayer neural networks, which require more than one layer are more accurate. Although there are situations in which more than one hidden layer(two layers total) is optimal for many situations, there are some perspectives in which more than one hidden layer is necessary. Sometimes four or even five layers are more effective. One alternative to this, is to scale the output to a more linear representation of the logistic function. As this is a somewhat new field, one are that could be improved going forward, is providing specific scenarios that deem the number of layers more accurate. This is something that has yet to be concretely solved, and many times people use trial and error to figure out how many layers will be in the most accurate model for the circumstances at hand. In this assignment, I will be developing an ANN as a computational model. It will be used to analyze whas1.csv, a Heart Disease Survival database. Specifially, it will predict the outcome of FSTAT. To do this, an algorithm which uses the remaining 13 columns as independent variables will be developed, with FSTAT as the dependent variable. After this, the model will be tested, to assess its accuracy.

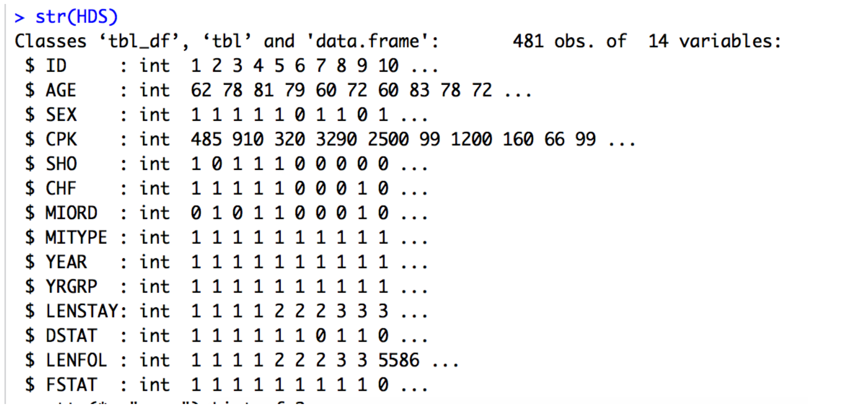
2) Analysis

We will begin our analysis by loading the neural net library, and setting the working directory to the appropriate location. After that, we load the CSV of interest, whas1.csv. As the name was not intuitive, I elected to rename whas1, HDS, short for Heart Disease Survival. Now that the data and packages have been loaded, I will begin to by Viewing the Data. Our CSV consists of 14 variables. Of these I noticed that FSTAT is one variable that takes a binary value, either 0 or 1. I elected to chose this as my dependent variable, and left each other column in - to be used as independent variables that will be used in our algorithm. Further, I explore the database by using the str(HDS), summary(HDS), head(HDS) and names(HDS) commands to

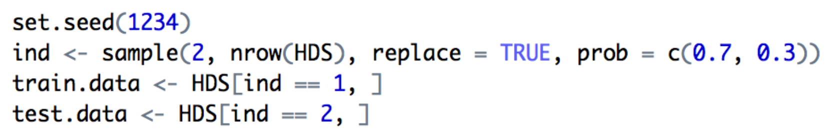

HDS

Filter

	ID	AGE	SEX	CPK	SHO	CHF	MIORD	MITYPE	YEAR	YRGRP	LENSTAY	DSTAT	LENFOL	FSTAT
1	1	62	1	485	1	1	0	1	1	1	1	1	1	1
2	2	78	1	910	0	1	1	1	1	1	1	1	1	1
3	3	81	1	320	1	1	0	1	1	1	1	1	1	1
4	4	79	1	3290	1	1	1	1	1	1	1	1	1	1
5	5	60	1	2500	1	1	1	1	1	1	2	1	2	1
6	6	72	0	99	0	0	0	1	1	1	2	1	2	1
7	7	60	1	1200	0	0	0	1	1	1	2	0	2	1
8	8	83	1	160	0	0	0	1	1	1	3	1	3	1
9	9	78	0	66	0	1	1	1	1	1	3	1	3	1
10	10	72	1	99	0	0	0	1	1	1	3	0	5586	0
11	11	78	0	99	0	1	1	1	1	1	4	1	4	1
12	12	84	1	135	0	0	0	2	1	1	4	1	4	1
13	13	79	1	210	1	1	0	1	1	1	4	1	4	1

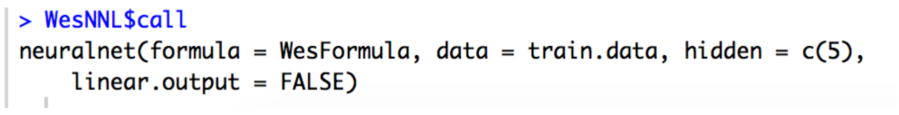

```
> str(HDS)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':	481 obs. of  14 variables:
 $ ID     : int  1 2 3 4 5 6 7 8 9 10 ...
 $ AGE    : int  62 78 81 79 60 72 60 83 78 72 ...
 $ SEX    : int  1 1 1 1 1 0 1 1 0 1 ...
 $ CPK    : int  485 910 320 3290 2500 99 1200 160 66 99 ...
 $ SHO    : int  1 0 1 1 1 0 0 0 0 0 ...
 $ CHF    : int  1 1 1 1 1 0 0 0 1 0 ...
 $ MIORD  : int  0 1 0 1 1 0 0 0 1 0 ...
 $ MITYPE : int  1 1 1 1 1 1 1 1 1 1 ...
 $ YEAR   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ YRGRP  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ LENSTAY: int  1 1 1 1 2 2 2 3 3 3 ...
 $ DSTAT  : int  1 1 1 1 1 1 0 1 1 0 ...
 $ LENFOL : int  1 1 1 1 2 2 2 3 3 5586 ...
 $ FSTAT  : int  1 1 1 1 1 1 1 1 1 0 ...
```

get a better idea about how our dataset is constructed. I see that our columns consist of the following: ID, AGE, SEX, CPK, SHO, CHF, MIORD, MITYPE, YEAR, YRGRP, LENSTAY, DSTAT, LENFOL, FSTAT. There are 481 instances of these 14 variables. Fortunately, our variables are already binary, and we do not need to encode them. If we had a variable that had a text output, we may chose to encode it in order for it to be useful for our ANN purposes. An artificial network does not know how to interpret text by itself, and under that scenario, the onus would be on us to interpret the text and encode it such that the variable is still able to be used in the analysis. Before we begin to develop the model, we must start with setting a random seed. I chose to set.seed(1234), but we could have used any number so long that it was the same for each trial. This allows the data to be split, and reproducible. We split the data into two



```
set.seed(1234)
ind <- sample(2, nrow(HDS), replace = TRUE, prob = c(0.7, 0.3))
train.data <- HDS[ind == 1, ]
test.data <- HDS[ind == 2, ]
```

categories with replace set to TRUE in order to eliminate any repeat values. I chose to use 70% and 30%, as we are going to set the 70% category to our training data, and the 30% to our test data. Our newly created variable, train.data will be used to develop the model, and test.data used to determine how accurate our model is by testing it. We continue the development of our model by calling the neuralnet() function. I create a new variable, WesNNL, setting the data = train.data, 5 nodes, as well as linear output set to false. Next I print the



```
> WesNNL$call
neuralnet(formula = WesFormula, data = train.data, hidden = c(5),
    linear.output = FALSE)
```

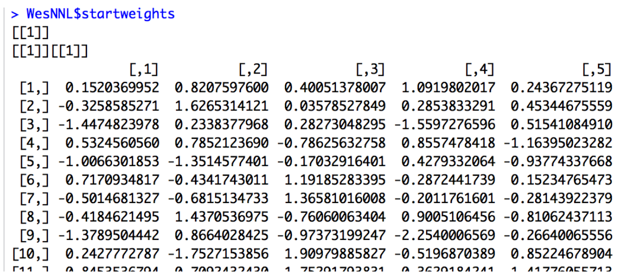
model using WesNNL$call. Note that I can tweak this model as I go, especially if I discover it to be more accurate. Continuing I print the values of the variables using WesNNL$response, display the values input into the model using WesNNL$covariate, display the names of the input variables and target variables using WesNNL$model.list, show the predicted weights for the first ten items using WesNNL$net.result[[1]][1 : 10]. I show the start weights by coding

WesNNL$startweights, the connection weights as well as the names of beginning and terminal nodes - WesNNL$result.matrix. Finally, I am ready to plot the function, plot(WesNNL)

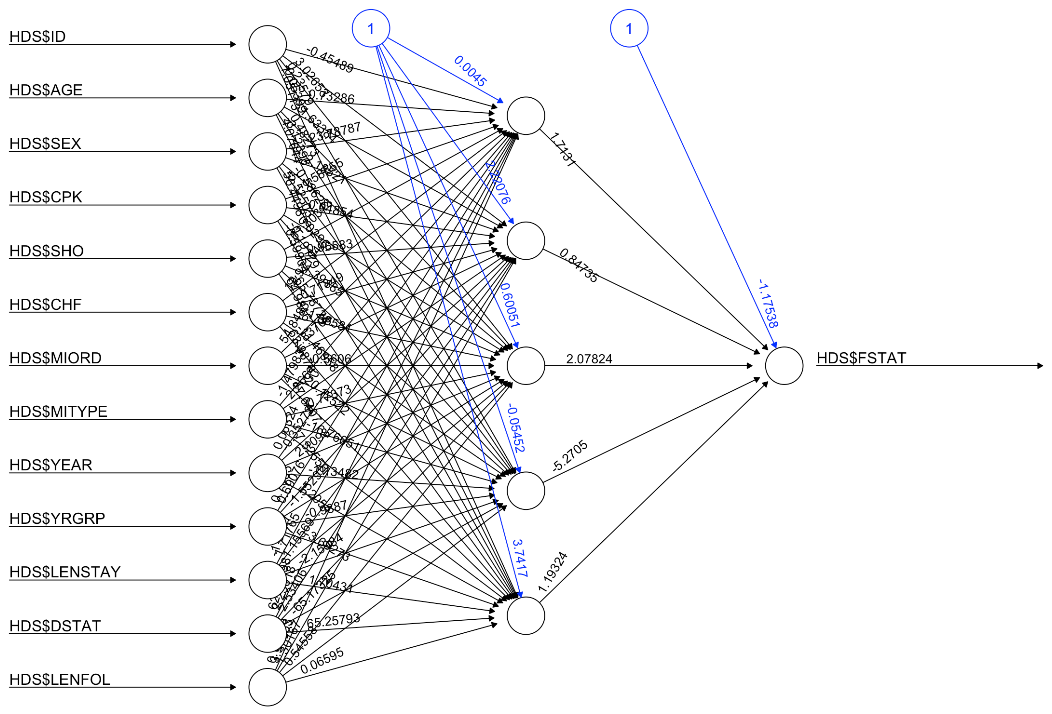
Our start weights are displayed. As they are too long to list, I display the results

The model is complete, but I must move forward with prediction. To do this, I create a new variable,

We are able to see each of the nodes, as well as their corresponding weights displayed on this plot. Because we have quite a few variables listed, the weights can be difficult to read. This is why it is nice to call the weights using different functions, as we did prior. It appears our model works properly, as HDS$FSTAT is our end node, and each of the other variables is a beginning node. We notice The hidden layer displayed in this plot as well.

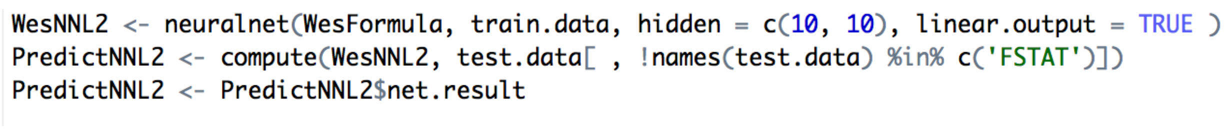


```
> WesNNL$startweights
[[1]]
[[1]][[1]]
              [,1]          [,2]          [,3]          [,4]          [,5]
 [1,]  0.1520369952  0.8207597600  0.40051378007  1.0919802017  0.24367275119
 [2,] -0.3258585271  1.6265314121  0.03578527849  0.2853833291  0.45344675559
 [3,] -1.4474823978  0.2338377968  0.28273048295 -1.5597276596  0.51541084910
 [4,]  0.5324560560  0.7852123690 -0.78625632758  0.8557478418 -1.16395023282
 [5,] -1.0066301853 -1.3514577401 -0.17032916401  0.4279332064 -0.93774337668
 [6,]  0.7170934817 -0.4341743011  1.19185283395 -0.2872441739  0.15234765473
 [7,] -0.5014681327 -0.6815134733  1.36581016008 -0.2011761601 -0.28143922379
 [8,] -0.4184621495  1.4370536975 -0.76060063404  0.9005106456 -0.81062437113
 [9,] -1.3789504442  0.8664028425 -0.97373199247 -2.2540006569 -0.26640065556
[10,]  0.2427772787 -1.7527153856  1.90979885827 -0.5196870389  0.85224678904
```

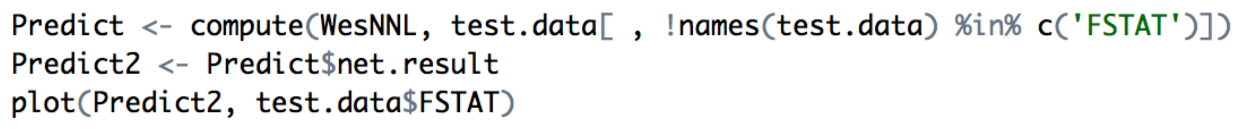


3) Results and Predictive Evaluation

Now that the model has been developed, we are able to move forward with developing a predictive variable. I create the variable Predict, using the compute function with WesNNL, our test data and the FSTAT variable. I decided to name a second variable, Predict2, as I was


```
WesNNL2 <- neuralnet(WesFormula, train.data, hidden = c(10, 10), linear.output = TRUE )
PredictNNL2 <- compute(WesNNL2, test.data[ , !names(test.data) %in% c('FSTAT')])
PredictNNL2 <- PredictNNL2$net.result
```

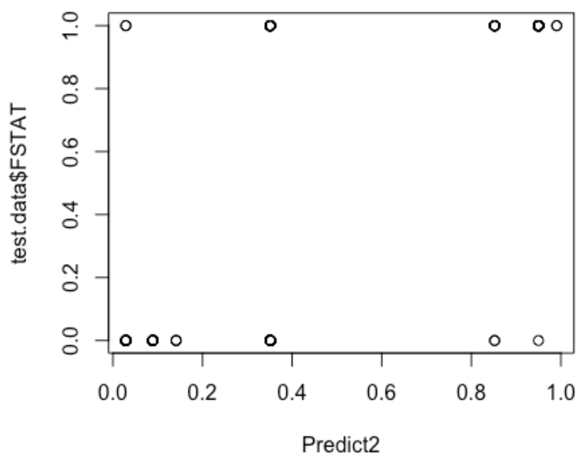
hesitant to replace it the first go around. In retrospect, I could have just renamed it Predict, instead of Predict 2, and set that to net.result. Next, I go ahead and plot our new model on the test data. This displays the predictive values that in a scatter plot, with test.data$FSTAT as a Y-

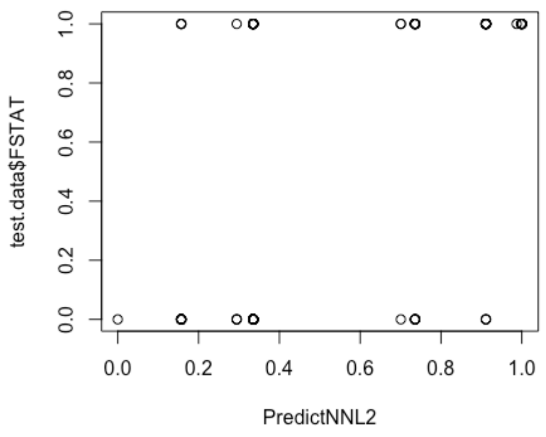

```
Predict <- compute(WesNNL, test.data[ , !names(test.data) %in% c('FSTAT')])
Predict2 <- Predict$net.result
plot(Predict2, test.data$FSTAT)
```

axis variable spanning from 0-1, and Predict2 as the x-axis Variable also spanning from 0-1.

We can be content with our model at this point, but I decided to go further and develop a second predictive variable, PredictNNL2

In PredictNNL2, I decided to use two hidden layers, with 10 nodes each, instead of the previously used 1 hidden layer with 5 nodes.





The corresponding graphs are displayed. They compare the two predictive models. It appears our first model is more accurate, as there is less overlap than present in PredictNNL2. Interestingly, this area of Neural Networks is still being improved. There is no correct way to decide exactly how many layers one is to use on a neural network to improve the maximum predictive value. Instead, it is acceptable to use a trial and error process of several different tweaks of the parameters and subsequent analysis of each to decide which Neural Network is the most effective.

4) Conclusion

This was a wildly fun assignment for me to do! I very much enjoyed the usage of Neural Networks, and found them particularly fascinating, as I could continuously imagine new scenarios in which Neural Networks would be useful in additional analysis. Some future analyses that I may chose to use ANN's for include: stock market prediction, game theory optimal play for a variety of games, such as poker or chess as well as facial and voice recognition. All of these topics intrigue me, and this subject struck a chord that was especially fun to work with. On the whole I found this assignment very rewarding. To improve the analysis, I would like to do continued tweaking of parameters. One cannot help but wonder how tweaking these would improve the results even further. It is interesting to me that this category has not been solved, per se. Perhaps this is an area of future research that can be done. I would be surprised if there was not a stricter guideline for developing ground rules on how many nodes and layers are used for each neural network in the future, instead of the trial and error approach that is used today.

Code

```
Ne
# Wesley Clark

# This is an example of a Neural Network using R
# Table of contents
#
#
#
##

# Load the neuralnet library
library("neuralnet", lib.loc="~/Library/R/3.3/library")

# Set the working directory to the appropriate location
setwd("~/Documents/630_3_Dr.H/Assignment5")

# We must load the CSV
library(readr)
whas1 <- read_csv("~/Documents/630_3_Dr.H/Assignment5/whas1.csv")

# Rename the data HDS.  Short for Heart Disease Survival.
# Check to make sure it has properly changed

HDS <- whas1
View(HDS)

# Look at the data using summary, structure, head and names commands
str(HDS)
summary(HDS)
head(HDS)
names(HDS)

#  Our variables are already binary, we do not need to encode them.



# We have to split the data, and set the random seed
set.seed(1234)
ind <- sample(2, nrow(HDS), replace = TRUE, prob = c(0.7, 0.3))
train.data <- HDS[ind == 1, ]
test.data <- HDS[ind == 2, ]
```

```
# We make a formula, setting HDS$FSTAT as the Dependent Variable, and using all our other
columns as independent variables.
# A tilde, implies every other column, however I decided to list each column individually as a
variable.


WesFormula <- HDS$FSTAT ~ HDS$ID + HDS$AGE + HDS$SEX + HDS$CPK + HDS$SHO +
HDS$CHF + HDS$MIORD + HDS$MITYPE + HDS$YEAR + HDS$YRGRP + HDS$LENSTAY +
HDS$DSTAT + HDS$LENFOL

# The formula that is returned is:
# Fstat ~ ID + AGE + SEX + CPK + SHO + CHF + MIORD + MITYPE + YEAR...
# + YRGRP + LENSTAY + DSTAT + LENFOL

# We will train a neural network with a hidden layer and 5 nodes next
# hidden = 5, creating a neural net with a singel hidden layer and 10 nodes
# linear.output = FALSE creates a network with a regression output

WesNNL <- neuralnet(WesFormula, data = train.data, hidden = c(5), linear.output = FALSE)

# Print the mode
WesNNL

# Print the fomula used to call the model WesNNL
WesNNL$call

# This will print the vaalues of the variables that are used to train the model
WesNNL$response

# Covariate will display the values that were input into the model
WesNNL$covariate

# Model list wil show the name of the input variables and the target variables
WesNNL$model.list

# Net result [1:10] will show the predicted weights for the first 10 items
WesNNL$weights

# The weights after the final iteration are dislayed
WesNNL$net.result[[1]][1 : 10 ]

# The start weights are displayed
WesNNL$startweights

# The connection weights as well as the names of beginning and terminal nodes
WesNNL$result.matrix

# Use the plot function to plot the NNL
plot(WesNNL)
```

```
# We can create a new variable, Predict and Predict2.
# Predict will be used to analyze the test set, and Predict 2 will be used to graph the net.result
# Then we will plot Predict2 on our test data for FSTAT

Predict <- compute(WesNNL, test.data[ , !names(test.data) %in% c('FSTAT')])
Predict2 <- Predict$net.result
plot(Predict2, test.data$FSTAT)

# I will tinker with the neural network, creating a second one with two hidden layers of 10 nodes
each
WesNNL2 <- neuralnet(WesFormula, train.data, hidden = c(10, 10), linear.output = TRUE )
PredictNNL2 <- compute(WesNNL2, test.data[ , !names(test.data) %in% c('FSTAT')])
PredictNNL2 <- PredictNNL2$net.result

# Plot the second Neural Network

plot(PredictNNL2, test.data$FSTAT)

# Display the weights!

WesNNL2$weights
PredictNNL2 <- PredictNNL2$net.result
plot(PredictNNL2, test.data$FSTAT)
```

References

- Bishop, Christopher: Neural Networks for Pattern Recognition, Oxford,1995.
- Herranz, E., Dr. (2017, March 07). DATA 630 Neural Networks - Herranz. Retrieved July 20, 2017, from https://www.youtube.com/watch?v=df2NJufihCM&feature=youtu.be
- Hasheminia, H. (2015, April 27). R-Session 11 - Statistical Learning - Neural Networks. Retrieved July 21, 2017, from https://www.youtube.com/watch?v=lTMqXSSjCvk
- Han. (n.d.). Data Mining: Concepts and Techniques. Retrieved July 23, 2017, from https://learn.umuc.edu/d2l/le/content/223093/viewContent/9220271/View
- ROGERS, S. (2016). FIRST COURSE IN MACHINE LEARNING EBOOK. S.I.: CHAPMAN & HALL CRC.