DATA630 9042

Assignment 4: Neural Networks

Sina Alemi

[sina.alemi@gmail.com](mailto:sina.alemi@gmail.com)

Table of Contents

[Introduction](#intro)

[Data Exploration](#data)[[Neural Network Model](#data)](#model)

[Conclusion](#conclude)

[References, Appendix, Script](#ref)

**Introduction**

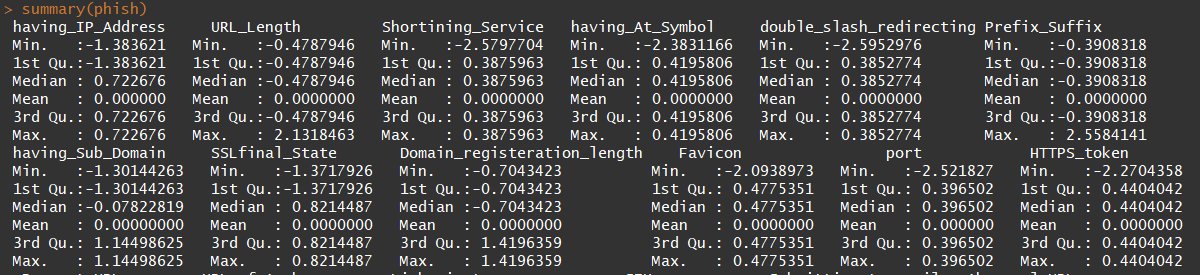
According to the Federal Trade Commission, phishing is “when a scammer uses fraudulent emails or texts, or copycat websites to get you to share valuable personal information” (Phishing, 2017). This information can include account numbers, social security numbers, or login information. The most common phishing methods rely on the victim to click on a link that points to a malicious website or runs dangerous web scripts. Many businesses train their employees to be on the lookout for possible phishing attempts by avoiding/reporting suspicious emails and links. Fortunately, many phishing links contain common characteristics that can be used to identify them. In this analysis, we will use the neuralnet package in R to create an artificial neural network which will be used to classify phishing websites using a dataset consisting of various characteristics of known phishing websites. The ANN models a biological brain consisting of neurons and the synapses that connect them. The ANN method is a good choice for this analysis because ANNs are able to detect complex relationships among variables and represent that information in the nodes and their interconnections.

An ANN consists of input nodes each representing an input variable. The input nodes feed their output towards one or more layers of hidden nodes. Finally, the hidden nodes output into the layer of output nodes. In a feed forward system, each neuron in a layer of neurons feeds its output to each neuron in the next layer. Each input into a neuron is associated with a weight determined, which is multiplied by the input and summed. The output of the node is determined by the activation function and whether the summed inputs (also known as activation value) meet an activation threshold. If the threshold is not met, no information is outputted by the node. A bias is added to each weighted sum, which shifts the activation value and allows flexibility in learning.

**Data and Algorithm Exploration**

The dataset to be used in this analysis contains tuples with various characteristics of phishing websites as well as whether the instance is a phishing website. An ANN will be built to with the goal of classifying tuples as a phishing website or not based on their given characteristics. The dataset contains around 11,000 tuples with 30 input variables and 1 class variable. Each input variable represents a common feature of phishing websites. Examples include address bar features (such as whether the URL has the ‘@’ symbol), script based features (such as whether the site disables right clicking), and domain based features (such as the age of the site’s domain).

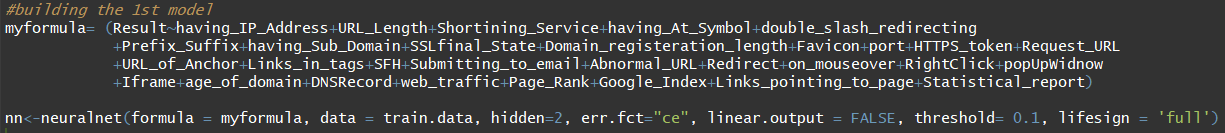
Each variable has 1 or 2 levels: possible values for 2 level variables are -1 or 1, and possible values for 3 level variables are -1, 0, or 1. The input variables will be scaled to have the same mean and standard deviation. This is to ensure that each variable has equal impact upon the output. After scaling, the variables have various ranges but all have a mean of 0, as seen below.



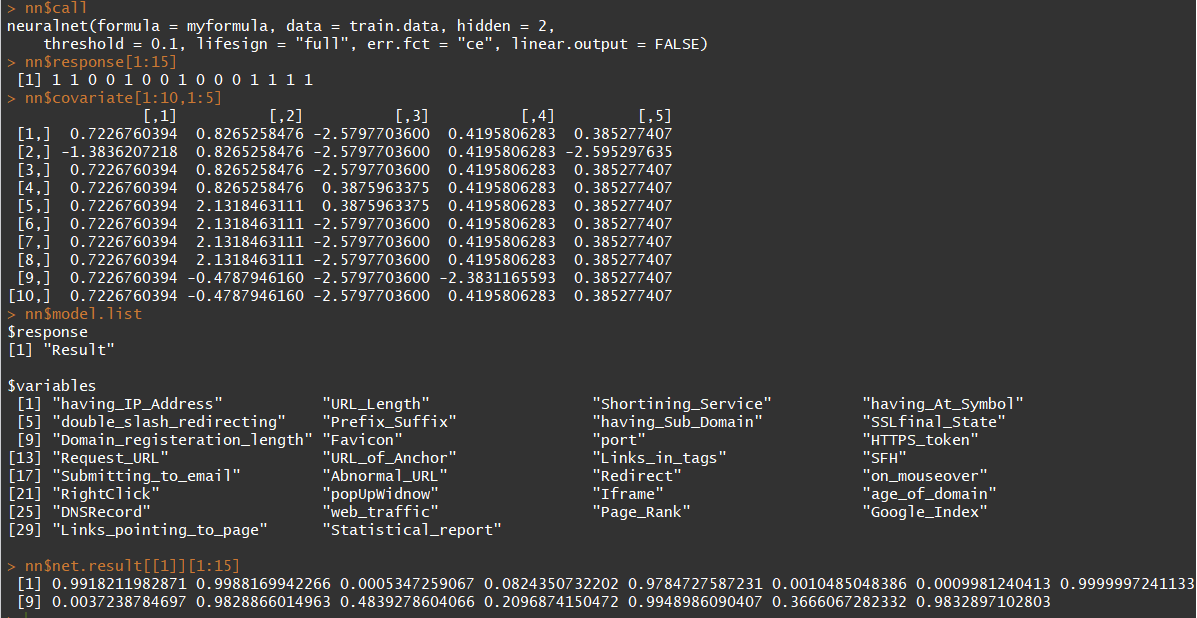
The neural net method employs supervised learning by training a network with a training set with known inputs and outputs. The model can then be applied to a test set to evaluate its accuracy on previously unseen data. The network compares its prediction with the known value and modifies the weights of the neurons so as to minimize the error between the predicted and known class value (Han, p. 11). The weight adjustments are made in the backward direction, from the output layer to the hidden layers and finally back to the initial input layer. This backward directional method of weight modifications is known as backpropagation. As the model improves its predictive ability, the weights converge and the learning process stops. The calculation of the error can be done using different functions such as mean squared error and cross entropy.

Sometimes, the weights appear to converge onto a solution, but the solution is not optimal (Han, p.12). The goal is ultimately to minimize the prediction error. Weights are adjusted in a gradual manner, and sometimes they converge on a local minimum of error as opposed to the global minimum. The learning rate, which can be specified in the parameters of the neural net function, helps to avoid converging on global minima. A lower learning rate reduces learning speed whereas a higher learning rate increases the chances that the results oscillate between solutions. Depending on how the learning rate is set, there is a tradeoff between learning speed (and thus computation time) and learning accuracy.

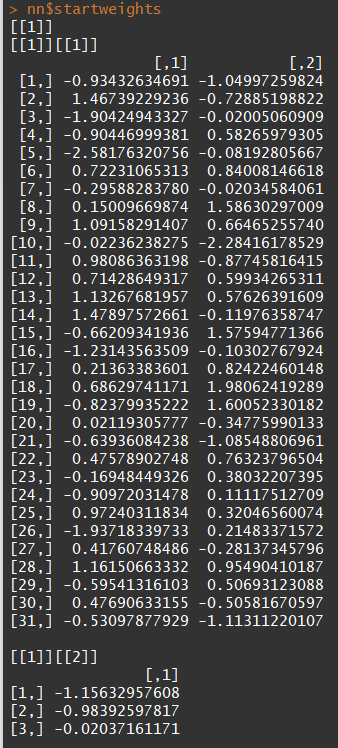
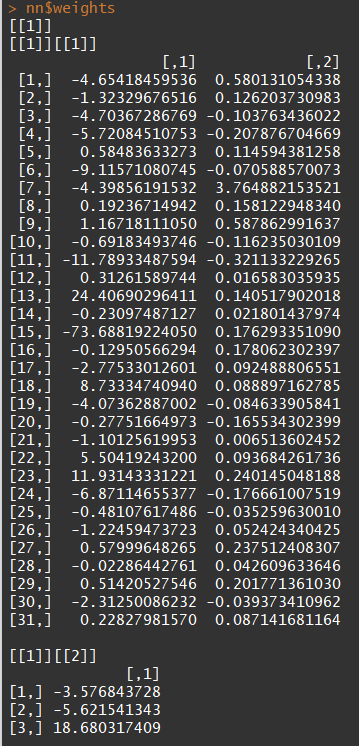
**Neural** **Network Model**

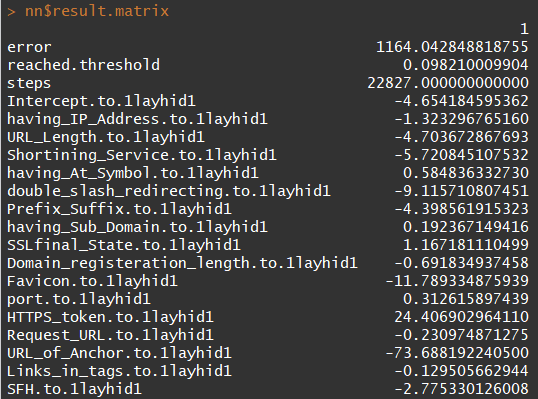
Since the ANN method is a form of supervised learning, the dataset will be split into training and test sets with a 70/30 split ratio. The initial model is built on training data using the class variable ‘Result’ as the designated response variable and all remaining variables as inputs. The command to build the model is shown below:  
 The model will contain 1 layer of 2 hidden nodes. This parameter can be adjusted to experiment with differing network topology. The cross entropy function will be used to calculate error as it is felt to be superior to the mean squared error function for classification problems. The linear output is set to false to specify that we want to conduct classification as opposed to regression. The default maximum cycles are 100,000, which can be adjusted lower if computation takes too long. The start weights are randomized by default to prevent them from converging too soon.

The default activation threshold (not shown) is 0.01, meaning if the activation value of a neuron exceeds this value the neuron will output a signal. Under this default value, the model’s weights did not converge after 100,000 steps. The ‘lifesign’ parameter was used to monitor the threshold values every 1000 cycles, leading to the adjustment of the threshold to 0.1. This value was chosen as it was higher than 0.1 but still under the thresholds reported by the lifesign parameter. Using these parameters, the learning concluded after around 23,000 cycles. The various sections of the output of the neural net are displayed below:



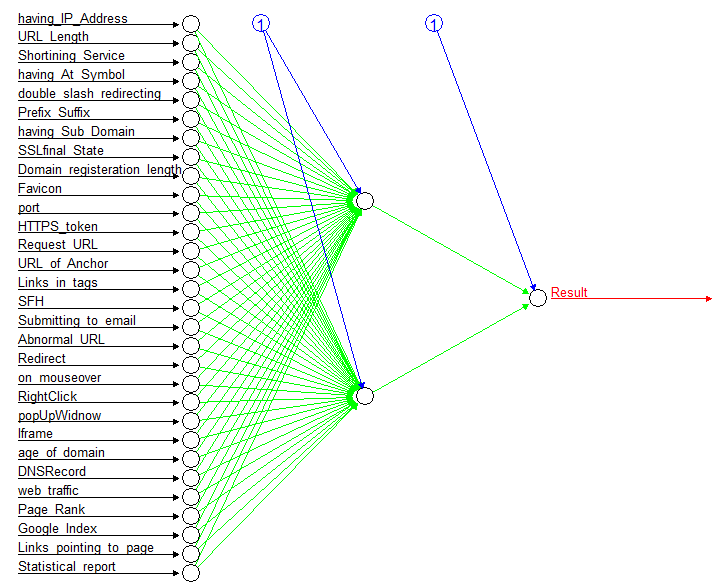
The call is the command used to create the model. The response lists the class value for the first 15 tuples. The covariates listed are the values of the first 5 input variables for the first 10 tuples. The model list displays the output and input variables used in the analysis. The net results are the results of the analysis. The values range from 0 to 1, with a value of [0, 0.5) indicating a negative class value and [0.5, 1] indicating a positive class value. The two figures below show starting weights and ending weights for the inputs into each hidden and output node. Each row corresponds to an input node. The two nodes at the top are the two hidden nodes, and the one node at the bottom is the output node (with two rows, one for each hidden node). The starting weights are all roughly between -1 and 1, whereas the ending nodes sometimes vary greatly, with some weights at 24 or -73.



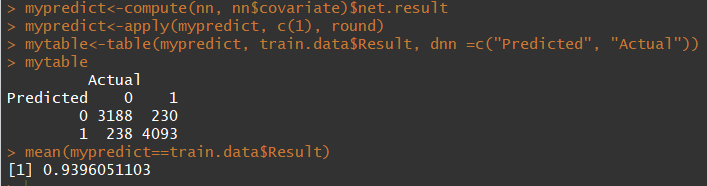


The result matrix shows the error and number of training steps, and lists the various node connections and their weights for each hidden and output node. Only parts are shown to save space. An outlier weight such as the -73 below is not necessarily indicative of the effect of the feature. The various weights in the network are deeply interconnected and the “black box” nature of the network is not easily interpretable from the numbers alone.

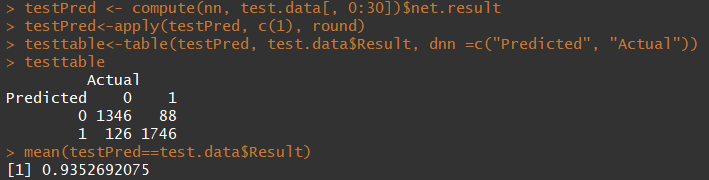
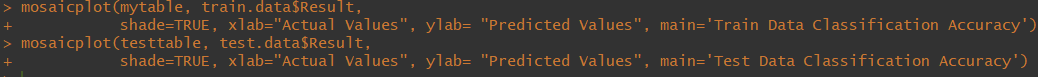
A visual representation of the network and its many connections is displayed below. The nodes on the left comprise the input layer. The green lines are connections from one node to another where output values are weighted and summed. The blue nodes indicate the biases or intercepts being applied. The red line is the final output.

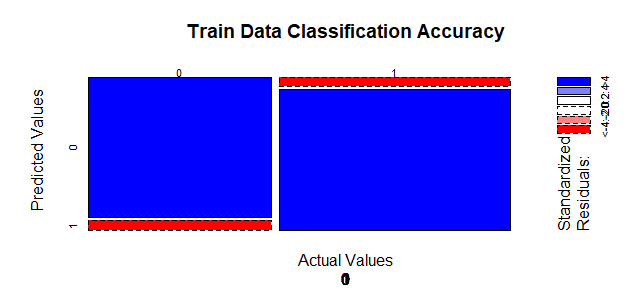
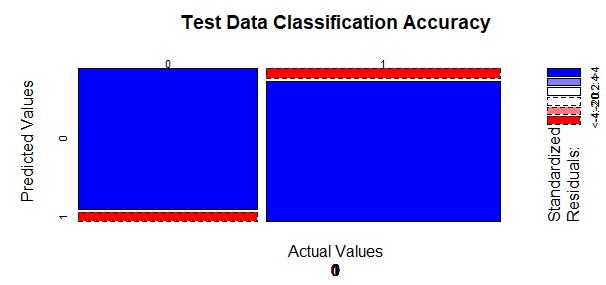


The model is evaluated by creating a confusion matrix. Values are predicted for the training set using the compute function and rounded to 0 or 1 for clarity. The classification accuracy is 93.9%. The sensitivity of the model is TP/(TP+FN), in this case 4093/(4093+230) = 95%. The specificity is TN/(TN+FP), in this case 3188/(3188+238) = 93%. High scores in all three metrics indicate an effective model.



Similar results are seen for the model’s performance on the test data. The classification accuracy is 93.5%. The sensitivity of the model is TP/(TP+FN), in this case 1746/(1746+88) = 95%. The specificity is TN/(TN+FP), in this case 1346/(1346+126) = 91%.

  
The tables are visually represented by the mosaic plots below, and they are nearly identical. 



**Conclusion**

In summation, the created model scored highly on classification accuracy, sensitivity, and specificity. The model performed well on both training and test data sets, meaning that the model was not overfit to fit the training set. Excessive use of hidden layers and nodes was not necessary, as the model only required two hidden nodes to perform well. One possible reason for the model’s good performance is that the data simply did not have much variation between the negative and positive classes. In other words, there was not much overlap in the characteristics of phishing and non-phishing websites. Perhaps the problem of classifying phishing websites it relatively simple and an advanced method such as ANN was not necessary.

It would be interesting to take a dataset that is difficult to classify with more traditional methods, such as logistic regression, and compare the results with those of a neural network. The same performance may also be replicated using fewer input variables. Perhaps only using the URL based features might yield similar results with the benefit of lower computing cost. Another approach would be to adjust the parameters. Instead of increasing the activation threshold, we may simply increase the number of maximum cycles to allow more time for the weights to converge. This of course would negatively impact computing cost, but the option is available. A disadvantage of the ANN method is that it is difficult to tell which variables had the biggest impact on the outcome. Simply looking at the weights does not tell the whole picture. In fact, research into extracting knowledge out of trained neural networks is an emerging topic. For example, clustering can be used to find sets of hidden node activation values and relate them to corresponding output values (Han, p. 16). Neural networks are a powerful yet enigmatic tool that will no doubt be developed further in the coming years.

**Ref****erences**

Han, Kamber, and Pei. *Data Mining: Concepts and Techniques*, Third Edition (2011).  Chapter 9.

Phishing. (2017, July 07). Retrieved November 10, 2017, from <https://www.consumer.ftc.gov/articles/0003-phishing>

**Appendix**

R Script

#DATA630 9042

#Assignment 4

#By: Sina Alemi

library("neuralnet")

#dataframe creation

setwd("D:/school/DATA630/module 5")

phish<-read.csv(file='phishing.csv',head=TRUE, sep=",")

str(phish)

summary(phish)

#scaling the input variables (first 30)

phish[1:30]<-scale(phish[1:30])

#converting Result values of '-1' to '0'

phish$Result[phish$Result=='-1']<-0

#make sure that the result is reproducible

set.seed(12345)

#split the data into a training and test set

ind <- sample(2, nrow(phish), replace = TRUE, prob = c(0.7, 0.3))

train.data <- phish[ind == 1, ]

test.data <- phish[ind == 2, ]

#building the model

myformula= (Result~having\_IP\_Address+URL\_Length+Shortining\_Service+having\_At\_Symbol+double\_slash\_redirecting

+Prefix\_Suffix+having\_Sub\_Domain+SSLfinal\_State+Domain\_registeration\_length+Favicon+port+HTTPS\_token+Request\_URL

+URL\_of\_Anchor+Links\_in\_tags+SFH+Submitting\_to\_email+Abnormal\_URL+Redirect+on\_mouseover+RightClick+popUpWidnow

+Iframe+age\_of\_domain+DNSRecord+web\_traffic+Page\_Rank+Google\_Index+Links\_pointing\_to\_page+Statistical\_report)

nn<-neuralnet(formula = myformula, data = train.data, hidden=2, err.fct="ce", linear.output = FALSE, threshold= 0.1, lifesign = 'full')

nn

#Run the commands to display the network properties

nn$call

nn$response[1:15]

nn$covariate[1:10,1:5]

nn$model.list

nn$net.result[[1]][1:15]

nn$weights

nn$startweights

nn$result.matrix

#visualization

plot(x=nn, col.intercept = 'blue', col.hidden.synapse = 'green', col.out.synapse='red',

fontsize = 10, dimension = 10, show.weights=F)

#Model Evaluation

#obtaining predicted values

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

#rounding results to nearest integer

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

#confusion matrix for the training set

mytable<-table(mypredict, train.data$Result, dnn =c("Predicted", "Actual"))

mytable

#Classification Accuracy %

mean(mypredict==train.data$Result)

#mosaic plot of train data evaluation

mosaicplot(mytable, train.data$Result,

shade=TRUE, xlab="Actual Values", ylab= "Predicted Values", main='Train Data Classification Accuracy')

# confusion matrix for the test set

testPred <- compute(nn, test.data[, 0:30])$net.result

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

testtable<-table(testPred, test.data$Result, dnn =c("Predicted", "Actual"))

testtable

mean(testPred==test.data$Result)

mosaicplot(testtable, test.data$Result,

shade=TRUE, xlab="Actual Values", ylab= "Predicted Values", main='Test Data Classification Accuracy')