**Introduction**

The Pima diabetes dataset is about the diabetes status of the indigenous Native Pima tribe. This resourceful tribe “…live along the Gila and Salt rivers in Arizona, U.S…” where they were known for their farming skills. Since colonialism and even to this day, tribes across the U.S have been stripped of their (rightful) natural land and resources. According to the American Diabetes Association, in 2018 14.7 % American Indians/Alaskan Natives were diagnosed with diabetes. That is twice as much as non-Hispanic whites with 7.5% (*Statistics About Diabetes,* 2018). Because of the strict overlord control the federal government has over Native reservations across the U.S, it is extremely difficult for people living on the reservations to gain any economical prosperity. Sadly,”… Native American reservations are among the [poorest communities in the United States](http://perc.org/articles/why-are-indian-reservations-so-poor-0) (Flows, 2014). Being in a poor community means that people don’t have ease of access to food, transportation and economic opportunity. Having ease of access and opportunity can negative effect many aspects of one’s life and community, one of those things being poor health. In this paper we are going to explore and build a neural network around this Pima diabetes dataset. Our neural network model that we build will hopefully recognize patterns in the data that can help us identify what are common observations will result in Pima people having diabetes.

**Analysis and Results:**

*Origin of Data*

This Pima diabetes dataset is found on Kaggle (Kumar, 2018), a large open-source community of data scientist and machine learning scientist, where it was added in 2018. The dataset has nine variables and 768 observations (Figure #1). The nine variables are: pregnancies - number of times pregnant (*preg*), plasma glucose - Plasma glucose concentration a 2 hours in an oral glucose tolerance test (*plas*), blood pressure (*pres*), skin thickness (*skin*), insulin (*insu*), BMI (*mass*), diabetes pedigree function – the likelihood of diabetes based on family history (*pedi*), age (*age*) and *class* (1=has diabetes, 0= doesn’t have diabetes) (Kaggle). All the variables except for class are the independent variables in this dataset. Class is the dependent variable in this dataset, and we will be building a neural network model based on whether we can try to predict diabetes likelihood of the Pima people. It should be considered that I don’t know how old this dataset is and some of these metrics are bias based on race and ethnicity.

*Data Exploration*

In the data exploration I used the *str* function to view the data structure of all the variables. All the variables are numeric values. Using the *summary* function, I’m able to view the minimum, maximum, 1&3 quartile values, median and mean of each variable (Figure #2). Because class is a binary variable (0,1) it doesn’t generate much diversity in results in the *summary* view like the independent variables do. I get an idea on how positive diabetes and negative diabetes behave in the dataset; I created a histogram of *class* (Figure #3). As we can see in the output, 0 or negative diabetes are more present than 1 or positive diabetes. From a public health standpoint, I am happy to see that non-diabetes outperformed diabetes. Especially to a population of people that have had their resources stripped and forced to live a lifestyle that is not natural to them. Lastly, in my data exploration, I’m very curious about familial impact. Basically, how does a family history of a health attribute affect the lineage? To do this I made a plot chart of *pedi* (diabetes pedigree function) and class. As we can see from the output, people who have diabetes have a wider spread of *pedi* rate and have cases where it peaks to a high rate (Figure #4).

*Preprocessing*

I’ve evaluated that all the variables in this dataset are impactful when creating a neural network, because of this I will be using all the variables and not omitting any. But I do want to ensure that all the variables don’t have any missing values. To do this, I ran *colSums(is.na(diabetes))* command. This command will total the amount of missing values in each variable (Figure #5). There aren’t any missing values in the dataset, we don’t have to remove or calculate a value that would go in its place. Next in the preprocessing stage, I want to normalize all of the independent variables by using the *scale* function since all the variables are numeric.

*Analysis and Results*

In the creation of the neural network for the Pima dataset, the commands I used to split the model are *set.seed* command followed by *(12345)*. I then created the variable that will define the split population by 70/30. The parameters for this command were the random sample function calling the two buckets from the dataset rows and splitting them up where 70% of the population will go into one bucket and the other 30% will go into the other bucket. Once we have the two buckets, now we have to define the new variables that they will live in. For this, I created a train variable that will house the 70% population where the model will be built from and the test variable where the untreated population will be stored. These two variables are called *train.data* and *test.data* (Figure #6).

Next I created the neural network with all variables. In the creation of the neural network (nn) model, I used all the independent variables from the train.set, set the hidden layers to 3 nodes. I then set *err.fct* to “ce” (cross entrophy), and *linear.output* set to FALSE (Figure #7). With 3 hidden nodes I want to begin to tune the hidden node parameters with the most optimal. When I print the ten predicted probabilities using *nn$net.result[[1]][1:10]net. results* command, 8 out of the 10 predicted probabilities performed above the 0.5 threshold (Figure #8). For the *nn$result.matrix* command, we are reviewing the trainings steps, threshold, errors and the weights. In the output we can see the hidden layer of the neural network model. In this view we can see all the nodes weights and output weight of all 3 nodes from the hidden layer. We can see that in each hidden layer and variables weights-to-nodes calculated. From the calculation we can examine which variable-to-node has the most effect to the dependent variable, class. If the calculated weight is positive then that variable has an effect on class equaling to 1, or positive diabetes. If the weight is negative it is less likely that the node has an effect or doesn’t have diabetes. Lastly, at the end of the matrix, we can see the output node. The output node summarizes the weight effect to class from each hidden layer. We can see that 2 out of the 3 hidden layers have a positive number (Figure #9).

When the nn model is graphed in a *plot*, this will allow us to view the neutral network model we created with the columns data set. In the plot, we see the independent variables, input layer, the hidden layer with 3 nodes, the output layer with the dependent node, the weights associated with each node and the intercept or bias. We also see the error value of 192.2 and steps values of 8712 or it took the model 8712 times to run over the data (Figure #10).

Before I settle on a model, I want to tune the parameters and compare what results to find the best fitting model to the data. To do this, I will go back to me nn2 model and adjust the hidden layer parameter to 2 (Figure #11). The logic behind this is that it will take less steps to for the model to run the data and hopefully have lower error rate because the model doesn’t have to comply to the different intercepts. I next run *nn$net.result[[1]][1:10]net. results* command to the new nn model (Figure #12) to predict the first ten probabilities from this model. Comparatively, the new model had 5 out of 10 probabilities above 0.5 where the first model had 8 out of 10. For the *nn$result.matrix* command I applied to the new model, nn2, to it to see how it differed from the original model (Figure #13). We can see from the output layer that both hidden layers are positive values, which means that both hidden nodes have an effect on the dependent variable being 1.

When plotting the nn2 model (Figure #14), we see that the error rate increased to 208.7 and the steps also increased to 12,132. This isn’t a large increase and comparatively is could be as good as model as using 3 hidden nodes.

Next, I constructed the confusion matrix for the original model that has 3 hidden nodes (nn) against the training data. The confusion matrix shows predicted and actual negative test. To build the confusion matrix, I created a cross tabulation table where we call on the model and the original train. set *class* variable and set the model and the train.set class variable to be labeled predicted and Actual. From the cross tabulation, 0 value has 294 observation and 1 has 135. The accuracy rate of this model is 84% (Figure #15). When I run the confusion matrix on nn2 model the accuracy was 81% with, 0 value has 292 observation and 1 has 126 (Figure #16).

Lastly, I run the confusion matrix on the test untreated population of 30% of the raw data. The first input parameter is a model. The second input parameter is the values of the values of the input variables from the test data. Then running the command to round the probabilities. The accuracy rate of the untreated test population is 66% (Figure #17). I apply the same method to the nn2 model to see how the model with 2 hidden nodes performs against the untreated population is 68% (Figure #18).

Conclusion

Although both models basically resulted in the same model accuracy, I do think that using the original model with 3 hidden nodes is the best fit for this dataset. In the original model (nn), when we ran the confusion matrix, we saw that 127 observations in the test population (30%) tested negative for diabetes while 43 observations tested positive. Interestingly, 43 observation in the test data set is inaccurate. To me this reads that the data maybe inaccurate. The limitation I came across is not knowing that date of collection of datasets. I think it could of provided more context.

**Bibilography**

The Editors of Encyclopedia Britannica. (2019, May 27). Pima. Retrieved April 02, 2020, from https://www.britannica.com/topic/Pima-people

Kumar. (2018, February 27). Pima-indians-diabetes.csv. Retrieved April 04, 2020, from https://www.kaggle.com/kumargh/pimaindiansdiabetescsv

Statistics About Diabetes. (n.d.). Retrieved April 05, 2020, from https://www.diabetes.org/resources/statistics/statistics-about-diabetes

Flows, C. (2014, March 13). 5 Ways The Government Keeps Native Americans In Poverty. Retrieved April 05, 2020, from https://www.forbes.com/sites/realspin/2014/03/13/5-ways-the-government-keeps-native-americans-in-poverty/

**Appendix**

Figure #1

A screenshot of a cell phone

Description automatically generated

Figure #2

Input:

A picture containing drawing

Description automatically generated

Output:

A close up of a black background

Description automatically generated

Figure #3

Input:



Output:

A screenshot of a cell phone

Description automatically generated

Figure #4

Input:

A picture containing flower, drawing

Description automatically generated

Output:

A screenshot of a social media post

Description automatically generated

Figure #5

Input:

A picture containing drawing

Description automatically generated

Output:

A picture containing meter

Description automatically generated

Figure #6

Input

A picture containing drawing

Description automatically generated

Output:



Figure #7

Input

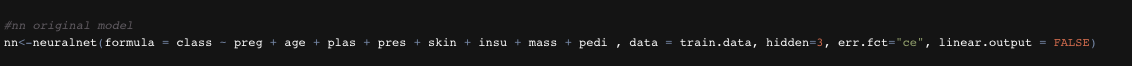


Figure #8

Input:



Output:



Figure #9

Input:



Output:

A picture containing man

Description automatically generated

**Hidden layer 2**

**Hidden layer 3**

**Output layer**

**Hidden layer 1**

Figure #10

Input:



Output:

A close up of a map

Description automatically generated

Figure #11

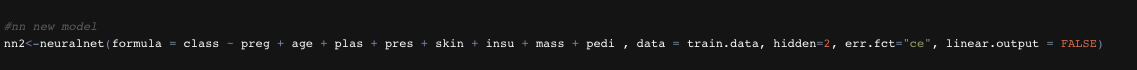
Input:

Figure #12

Input:



Output:



Figure #13

Input:

A picture containing food, drawing

Description automatically generated

Output:

A close up of text on a black background

Description automatically generated

Figure #14

Input:



Output:

A close up of a map

Description automatically generated

Figure #15

Input:

A picture containing drawing

Description automatically generated

Output:

A close up of a logo

Description automatically generated

Figure #16

Input:

A picture containing drawing

Description automatically generated

Output:

A screenshot of a cell phone

Description automatically generated

Figure #17

Input:

A picture containing drawing

Description automatically generated

Output:

A screenshot of a cell phone

Description automatically generated

Figure #18

Input:

A screenshot of a cell phone

Description automatically generated

Output:

Screen of a cell phone

Description automatically generated