Katherine Reilly
DS210 Final Project Report
12/15/2024

Final Project Report

For my final project, I created something similar to the NeuralNetworks from homework 9.My dataset contains demographic information along with medical history, employment, income, etc features of around 400,000 individuals. I wanted to create an algorithm that could determine if any of those attributes could be used to predict whether or not a person was likely to have mental illness, which was also a feature of the dataset. Although NeuralNetworks are typically used for identifying images, I wanted to modify it to give certain weights to different columns in my data.

To begin, I actually needed to make my dataset smaller, as GitHub has an upload limit of 25KB, and my file was about double that. To fix this problem, I opened my dataset in a software called JMP, typically used for statistical analysis and data modification/cleaning. I removed half the rows of the dataset by randomly selecting them; this way, I should still have a random sample that includes over 200,000 individuals. Once I did this, I was able to upload the smaller file to GitHub and my environment.

I first read and cleaned my dataset. I decided to create a module for this because it ended up being quite bulky and irrelevant to the rest of my code. I created a struct for all of a row's data as Strings (Person), then another struct for all of that row's data as f32 values(Individual). I imported a csv reader module and iterated through the rows of my dataset, making an instance of my Person struct for each and collecting them into a vector. I then created a function to convert a Person into an Individual (aka transforming all Strings -> f32). I assigned numeric values between 0 and 1 for every String value that corresponds to an attribute of my data: For example, for the feature "Employment", I assigned the String value "Employed" to a value of 1.0 and the String value "Unemployed" to a value of 0.0. Most of the dataset's attributes were well-suited for this, except a few which had 3 or 4 Strings as possible values. For these, I did my best to scale them numerically and assign them numbers between 0 and 1 (For education, I assigned "High School" to 0.0, "Associate's Degree" to 0.25, "Bachelor's Degree" to 0.5, "Master's Degree" to 0.75, and "PhD" to 1.0). For the already numerical attributes, like age, I just the minimum and maximum values for each column, scaling all the values in the column so the max was equal to 1 and the minimum was equal to 0. I created an Individual object for each row containing these f32 values.

Then, I split my data into training and testing sets. I made a function for this which is also within my cleaning module (called "read_and_split.rs"). This function took all of my Individual structs, and returned two vectors: One contained 70% of the Individuals–this was my training set–, and the other contained the remaining 30% of my data, which I would use to test.

Next, I determined input, hidden, and output sizes for my data. My rows each had 14 attributes that I wanted to test for a relationship with mental illness, so my input size was 14. I decided on 5 hidden layer nodes. And, because I wanted to predict either Yes/No for history of mental illness, I made my output size 1, which would be a number either 0 or 1. I randomized my weights for my network by generating numbers between 0.0 and 0.1.

Then, I made a struct for my 'NeuralNetwork' called Network containing my weights. I used this for my training.

I needed to make functions to both train and test my data and also to perform forward and backward propagation. I implemented my forward and backward propagation as methods for my Network struct. Starting with forward propagation, I went back to my code from homework 9 and followed the same steps as I did there. I noticed that, with all of my data already being numbers between 0 and 1, the sigmoid function changed little in the function of my propagation, but I decided to keep it anyway. For my backward propagation, I also followed pretty closely with what I'd done previously in the homework. I messed with the attenuation factor a little to see what value would give me the best accuracy, and landed on 0.1.

For my training function, I iterated through every row of my training data, performing forward and backward propagation on each. When the backward propagation function would return an output value, I would collect it in a vector containing all of my outputs for my training data. After I'd iterated through each row, I found the mean and standard deviation of the outputs (by creating functions using the formulas for each) so I could see the distribution of them. The outputs had a mean of 0.30 and typically varied by 0.08. I saved these together in a vector for future use, and my training function finished by returning the trained network and this distribution.

For my testing function, I iterated through each row of my testing data and performed forward propagation with my trained network, collecting all of the outputs into one vector and all of their respective target values into another. Once this was done, I rescaled my data based on the distribution of the outputs (mean and standard deviation) from the testing data so that all of the outputs would have a mean of 0.5 and a standard deviation that reflected the same scaling/transformation. I decided to do this because I knew I was going to be gauging the accuracy of my network by rounding the output values to either 0 or 1, and comparing that to their label, which was either 0 or 1. So, I wanted any outputs that fell below the average to be rounded to a 0, and any above to round to a 1. I scaled the mean and standard deviation by dividing by 0.3 and multiplying by 0.5. This left me with a new mean of 0.5 and a new standard deviation of 0.13. I then standardized each of my collected outputs according to this distribution, then rounded to the nearest integer. Then, I calculated the error by subtracting the target values from each standardized output value. I let accuracy be the amount of error values that were equal to 0 divided by the count of all test rows, which I printed as a percentage.

The choice to scale the data to fit a distribution with a mean around 0.5 could very well be a misrepresentation of the data, as the proportion of the dataset with a history of mental illness is actually 0.3. But, I wanted to see how this would impact my program regardless, and I was only getting an average of 0.3 because all of the output values were being rounded to 0, and the only time my program would mark them as "accurate" would be for the 69% of the sample who had a value of 0 as their target label (meaning: Rather than *actually* being accurate 69% of the time, the program was 100% accurate when an individual did *not* have mental illness, but *0%* accurate if they *did,* because the program output that nobody had a history of mental illness). This is why I chose to print both the accuracy of the original (which is 69%) *and* the accuracy of the scaled (which ends up being about 62%). I was hoping to get my program away from labeling everyone as having no history of mental illness, which I was able to.

I also noted that a tendency of the neural network to create outputs that are similar to the sample proportion might indicate that the attributes I'm investigating have no correlation. This

led to my decision to perform a statistical hypothesis test in my project, as well, to determine if that was the case for my data.

Next, I was interested in seeing which of the 14 attributes impacted the weights the most after all of my forward and backward propagation on my training data. I did this by forward propagating 14 1x14 arrays, each with a value of 1 for an attribute of interest, and the rest of the array filled with zeroes. I found which of those returned the highest output. I found that the feature “Chronic Medical Conditions,” (Yes/No indicating if an individual has chronic medical conditions) had the highest importance.

I decided to conduct a 2-sample independent z-test to determine if there was a significant relationship between having a history of mental illness and having a chronic medical condition. I created a function for this hypothesis test and printed out some of my steps in forming hypotheses, calculating the test statistic, and forming my conclusion. The test revealed that there was evidence that chronic medical conditions are related to mental illness! I was surprised by this initially, but as I reflected on the rest of my project, I figured some of the other, less important/related, attributes were confounding my network and messing with the accuracy of my neural network.