Quinn Huettner
Brandon Roehl
Dave Schmitz

Heart Disease Detection

The page that we got our data from had both processed and pre processed data. When we first used the processed data it seemed that the column that had the diagnosis, which according to the document was a binary option, had options 0 through 4. We decided that it was possible that the processed data was incorrect and had errors so we decided to process the data ourselves. The first step was to look at the data and determine what we needed and what could be dropped. The document listed the 13 key attributes that are used to make the diagnosis. We first converted the data from a space separated list to a comma separated list. After looking at it some more we realized that each line was not its own entry, and that would make it much easier to work with. Since the last thing in each person's data is “name” we used regex to split at name and insert a line break, so we now have a space separated list where each line is its own data point. After we simply used regex to replace the spaces with commas. Then using pandas we dropped the columns we didn't need and saved it to a processed data file. At this point we were basically back to where we started, and when looking at the diagnosis attribute it was still 0-4. After some research online we found an article of someone who used R to determine the most important attributes for determining heart disease, and they stated that the 0 and 1 were the ones that actually mattered, where 0 is a less likely chance to have heart disease, and 1 was a greater chance. So we learned our data was not saying if someone had it or not, rather their chances of having it based on the data. So we used regex again to remove any rows that had a 2, 3, or 4 in the last spot of the row, which left us with only the rows which ended in 0 or 1, the values we

care about. At this point our data was normalized to a point that we could begin to work with it.

Our model is making use of a MultiLayered Perceptron to determine the chances of a person having heart disease. Since our output is binary our model can fit into the category of classification, either below 50% chance or above 50% chance of developing heart disease. In our case, data is very limited. The only data set that had some decent data was the cleveland dataset. Other places had too many holes in the data that we cannot fill in. Even with the decent data we only had about 250 entries after cleaning it up. This allows us 150 entries to train and 100 to test. We ended up with results usually in the low 80% for accuracy and around .35 in loss. In a future trial it would be beneficial to collect more data to try and get better accuracy, but with how confidential health records are obtaining the information would be difficult.

Another problem that we don't really have a way of knowing is the background of the people in the data. It is a good idea to think about possible bias in your data and depending on the collection method there could be some issues. One is if the people are selected randomly or not. If this data is collected off people already in the hospital, it is likely that these attributes are skewed due to the fact that healthy people are not usually in a hospital. Additionally, the person would have to be asked to release the data for the study and then you have the bias of a certain type of people who are willing to release their data. The best way to get data for this would to be randomly selecting people to ask for their data and then measure the needed attributes.

We are using about 75% of the data for training and the rest for testing. This is the default for SciKit Learn. Since we have a small amount of data even getting a few wrong in testing can drastically alter the percentage of accuracy.

Our project aims to take in a bunch of data about a person's cardiovascular health and give you a prediction of how likely that person is to get heart disease. The model returns either 0 or 1, which is either a less than 50% chance to get heart disease or a greater than 50% chance to get heart disease respectively. Since heart diseases

has become a much greater problem recently it is useful to see the chances someone has of developing it so that they can change their habits earlier to help fight the chance of getting it. Heart Disease is the number one killer of both men and women in the United States. Having a tool able to predict a person's chances of developing heart disease can be life saving because it would allow doctors to recommend a plan of action for that patient to help lower their chances. The reason we can predict this is because of the attributes that the data has gathered. Doctors know what kind of things are early warning signs of heart disease, and with a lot of data the machine can find the patterns in the data and use that knowledge to determine a person's chances.

After searching on the internet we found a paper from a group of people who did more or less the same machine learning project on the same data as us. The interesting part was when we finally confirmed that they had used the same cleveland dataset from UCI that we did. Then when we looked at their accuracies with different algorithms it seemed that our model did a better job. Our model usually had around 80% accuracy where the paper had a reported accuracy rate from 40% to 55%. After looking through their paper it was hard to determine what exactly caused this major difference in accuracy, but the paper was using naive bayes and some other different algorithms that are slightly more complex, so it is possible that they were over engineering when compared to our simpler algorithm.

When it came to normalizing our data we had first tried to use min max normalization. We were building our normalizer from scratch when we found out we didn't know how to normalize the data with one hot encoding when there was a mix of categorical and non categorical data. We started looking all over for how to encode categorical data with linear data and range data together when we stumbled upon this medium article. (Vickery) This is where we learned about SciKit-Learn and the `ColumnTransformer`. This was perfect for what we were doing because we were able to use Tensorflow's `to_categorical` method for our Y so all we had to do was break down the input by whether the field was categorical or needed to be normalized. This

helped us out a lot because we were able to take advantage of other normalizing algorithms and play around with finding the best one. In the end then we ended up using the processed cleveland data from the datasource. This was because we were able to convert what we were messing around with in RegEx to python that we could use on the dataset that was provided by the datasource.

Something that we noticed when playing with the data is that there were 2 people we could delete from the data that would improve our accuracy by 10-20%. We determined that something about them was causing problems with the model actually learning so when testing there would be problems with accuracy. After removing them we got consistently higher accuracies, around 85%. This sort of problem is likely an issue with the data collection and amount of data. If we had more of these outliers our model would be able to better handle the outliers. Another issue is the data collection itself. A better study would have randomly selected people instead of getting people who are potentially already having problems if they are already in the hospital. Once again data makes a huge difference in how efficient your model is. This makes sense since all we have to make a good model is our data. This might explain why we had such a higher accuracy compared to the people online, they may not have filtered the data as well for outliers and junk data.

Our data was really small for this study though with with only 214 valid records that could be used even in the largest dataset. This lead to there being huge variance with training just from how it randomly split the data. We were left with sometimes having really good train and test others worse train than test and others that appeared to overfit. If we split it so only 10 percent of our data was in the test data than any one wrong answer would be 5% of our accuracy and removing more leaves us with so little training data that our model has a really hard time learning from it.

We originally tried to do this in keras but we found out we had better results with the MNIST MLP code so we did a deep dive into why this was. The entire difference

came down to the type of reductions for our minimizer. Even though we could change the optimizer and the algorithm we couldn't change what the optimizer minimized. The thing that pushed the MNIST MLP code over in being better was minimizing the cost instead with `softmax_cross_entropy_with_logits_v2` instead of just categorical optimizations. This helped us gain an additional 5% accuracy in train and test. We were never able to produce a greater consistent accuracy than about 84-86%.

In conclusion we think this data does do well in a standard classification example. The issue we encountered was in the lack of available data. Even if we were able to mix the sources we still wouldn't have near enough data, minus the fact that mixing data sources like that is a really bad idea. And when mixing data sources dropped our accuracy a lot because the differences in the populations. We need more data to get better analysis and higher accuracy.

Citations

Hazra, Animesh & Kumar Mandal, Subrata & Gupta, Amit & Mukherjee, Arkomita & Mukherjee, Asmita. (2017). Heart Disease Diagnosis and Prediction Using Machine Learning and Data Mining Techniques: A Review. Advances in Computational Sciences and Technology. 10. 2137-2159.

Dheeru, Dua and Karra Taniskidou, Efi. (2017). {UCI} Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences. https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/

Vickery, Rebecca. “Easier Machine Learning with the New Column Transformer from Scikit-Learn.” *Medium.com*, Medium, 1 Oct. 2018, medium.com/vickdata/easier-machine-learning-with-the-new-column-transformer-from-scikit-learn-c2268ea9564c.