Predicting Heart Disease with Supervised Machine Learning

Evan Chang, Nicholas Seidl

Abstract

Determining if a person is at risk of having heart disease is difficult, as it often depends on multiple variables such as age, gender, weight, etc. Even though certain doctors may be very good at predicting heart disease, patients would still have to physically go in for a visit. This restricts the number of people able to get evaluated to those who can afford a doctor's visit. If there were a way to predict whether someone was at risk of getting (or has) heart disease with a machine, then many more people would be able to be evaluated. Early evaluation is very helpful when diagnosing heart disease because treatment can start sooner.

Introduction

The difficulty in predicting the risk of heart disease lies in the fact that the previous data any single doctor or organization has access to is sparse. This is due to both physical variations in humans as well as doctor-patient confidentiality agreements. This makes it difficult for a doctor to predict whether or not a patient is at risk of having heart disease. Predicting heart disease early is important because medication and treatment can be administered before it is too late.

The problem we addressed is that it is difficult for doctors to predict heart disease provided traditional non-algorithmic methods (such as past hands-on experience and collaboration with peers). Through using a few mathematical approaches to modelling heart disease patterns, hopefully there is another source of input and data for doctors to use when diagnosing patients.

Technical Approach

Three different machine learning algorithms could be used to predict whether a subject has heart disease or not, given various pieces of medical information about that patient. These various pieces of medical information were called features. Example features of patients would be blood pressure, heart rate, sex, etc. Each approach's output was a binary classification of the patient: whether they were predicted to have heart disease, or predicted to not have heart disease (also called healthy). Our output classes were 0 = Healthy and 1 = Heart Disease.

The first classification technique used was Logistic Regression. Logistic Regression allowed for an arbitrary number of features (*d* features) to be involved in the final classification of the input sample. It used a linear combination of these features with trained weights as an input to the Sigmoid function. Logistic Regression used a *d*-dimensional hyperplane to separate data into two output classes. As per *Figure 1*, the output of Logistic Regression was a score (between 0 and 1, exclusively). If the

output was >= 0.50, classify the input sample as class 1 = Heart Disease; otherwise, classify the input sample as class 0 = Healthy.

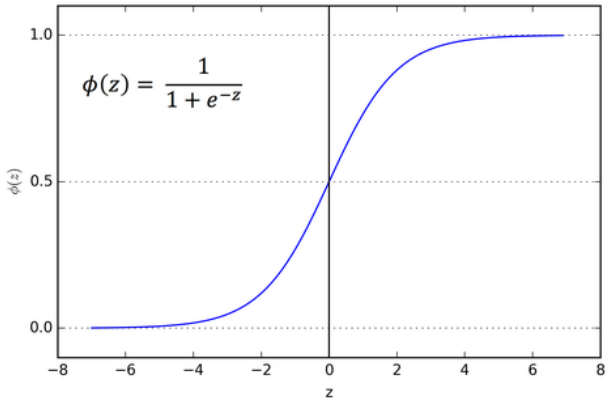

Figure 1: The Sigmoid function.

The second classification technique used was Gaussian Naive Bayes. Traditionally, Naive Bayes classifiers are used to classify input samples whose features are binary (take values of either 0 or 1). However, due to the nature of our features (such as blood pressure), some couldn't be represented in a binary nature, so these features were modeled as continuous values. Therefore, using a Gaussian distribution allowed for an estimation of the distribution of continuous values. The naive assumption that all of the input features were independent of each other was made, and therefore, Bayes' Rule could be used to calculate the probability of the output class, given conditional probabilities of each of the input features.

$$P(y|x_1,...,x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

$$P(y|x_1,...,x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}$$

$$P(y|x_1,...,x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

Figure 2: Application of Bayes' Rule for derivation of classification of input sample. y took values of 0 or 1 (output classes), and each x_n took values from its respective (potentially continuous) range.

As seen in *Figure 2*, the combination of conditional features $P(x_n \mid y)$ could be combined into a product notation, due to the assumption that all of the features were independent. A Gaussian distribution was used to estimate each $P(x_n \mid y)$ due to the fact that some features took continuous values. Finally, the output class chosen is the one that maximizes the right hand side of the final equation.

The third classification technique used was Feed Forward Neural Network (FFNN). Support Vector Machine (SVM) was originally planned to be used, but due to both personal interests and potential to investigate nuances in both techniques (changing kernel vs. changing number of layers, size of layers, activation functions, etc.), we

opted to use a FFNN instead. A FFNN was represented by a collection of nodes and the connections between those nodes. The network was comprised of a series of layers, where each layer fed forward its output to the next layer. Each layer was comprised of multiple nodes. *Figure 3* displays the architecture of a fully connected Feed Forward Neural Network (FCFFNN), where the output of each node in a layer was forwarded to each node (besides the bias node) in the next layer.

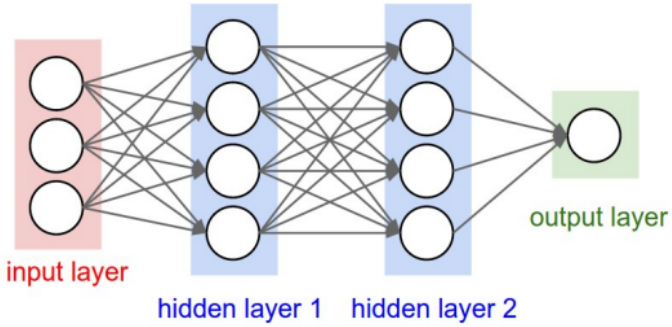

Figure 3: Fully Connected Feed Forward Neural Network architecture. Note that there was an additional node in each hidden layer that represents 'bias': it took no input, and always outputs +1 to each node (besides the bias) in the next layer.

Each node calculates the weighted sum of inputs, and then feeds result into an activation function, as diagrammed in *Figure 4*.

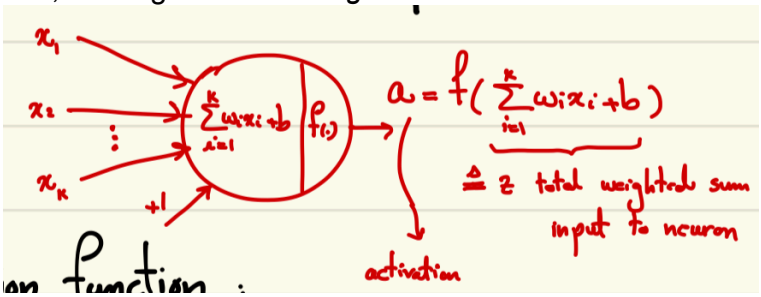

Figure 4: Diagram of a single neuron/node. k feature inputs were provided in addition to a +1 bias. The weighted sum was used as input to an activation function a.

Neural Networks could be used for both regression techniques (where the output value was continuous) or for classification techniques (where the output value was discrete). FCFFNN was used for classification in this case, so the activation function used in the output layer was the Sigmoid function (*Figure 1*) (more about this in the Experimental Results section) in the same fashion as in Logistic Regression.

The k-fold cross-validation technique was used for both the Logistic Regression model and the Gaussian Naive Bayes model. K-fold cross-validation was a technique used to resample data, when the size of the dataset was small. The entire dataset was shuffled, and then split into *k* groups. For each group k_i, all other groups were used to train the model, and the data in group k_i was used to evaluate the model. The resulting weights and error for each holdout set was maintained. Finally, the final set of weights used for predictions were the weights that resulted in the lowest error. Intuitively, as *k* increased to a large number, the size of the evaluation holdout set k_i decreased to an extremely small proportion of the whole dataset, so error calculated on that holdout set should not be trusted. Therefore, the highest *k* used is 10 (similar to a 90% training and 10% testing split).

The z-score normalization was used in order to normalize our data. Z-score normalization normalized features according to the mean value of that feature as well as standard deviation of the feature. As seen in *Figure 5*, if the value of a feature for a given sample was exactly equal to the mean of all the samples for that feature, the z-score of that value would be 0. If the value of that feature was larger than the mean, the z-score is positive, and if the value of that feature was less than the mean, the z-score was negative. The z-score was then scaled by the standard deviation of all the samples for that feature - if there was a high standard deviation for the features, the z-score values would be larger and more separated.

$$\frac{value - \mu}{\sigma}$$

Figure 5: Calculated the z-score of a value, given μ (the mean of the values) and σ (the standard deviation of the values).

Experimental Results

The Cleveland Heart Disease Dataset had 303 data points. Each of these samples was made of 14 attributes (one of which is whether they have heart disease or not). These attributes contained both binary features and continuous values. The attributes measured for: age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar levels, resting electrocardiographic results, maximum heart rate, exercise induced angina, ST depression induced by exercise relative to rest, the slope of the peak exercise ST segment, number of major vessels (0-3) colored by fluoroscopy, and Thalassemia.

To ensure that there were indeed relationships between the features and the presence of heart disease before training any model, a generalized linear model was used in R to find p values. This showed there were relationships of varying degrees; the strongest ones being values like sex and type of chest pain to heart disease.

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.441483   2.557158   1.346 0.178359
age         -0.004908   0.023175  -0.212 0.832266
sex         -1.758181   0.468774  -3.751 0.000176 ***
cp           0.859851   0.185397   4.638 3.52e-06 ***
trestbps    -0.019477   0.010339  -1.884 0.059582 .
chol        -0.004630   0.003782  -1.224 0.220873
fbs          0.034888   0.529465   0.066 0.947464
restecg      0.466282   0.348269   1.339 0.180618
thalach      0.023211   0.010460   2.219 0.026485 *
exang       -0.979981   0.409784  -2.391 0.016782 *
oldpeak     -0.540274   0.213849  -2.526 0.011523 *
slope        0.579288   0.349807   1.656 0.097717 .
ca          -0.773349   0.190885  -4.051 5.09e-05 ***
thal        -0.900432   0.290098  -3.104 0.001910 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 6: Statistical analysis on the features. A smaller value represented a stronger correlation..

The data was sourced from a .csv file and then normalized using z-score normalization.

Logistic Regression

A training-testing split of 90%-10% of the data was first used to learn a model. This yielded a training accuracy of 83.46%, a testing accuracy of 83.87%, and an overall accuracy of 83.50%. New models were then calculated in a similar manner but using k-fold cross validation. K-fold cross-validation was performed with k values of 2, 5, and 10.

K Value	Error
2	.2132
5	. 1481
10	. 0370

Therefore k = 10 achieved the optimal weights and error. It achieved an accuracy on the entire dataset of 85.14% which was slightly better than without k-fold cross validation (83.50%).

True positives (predicted heart disease correctly)	146
True negatives (predicted healthy correctly)	107
False positives (predicted heart disease when healthy)	31
False negatives (predicted healthy when heart disease)	19

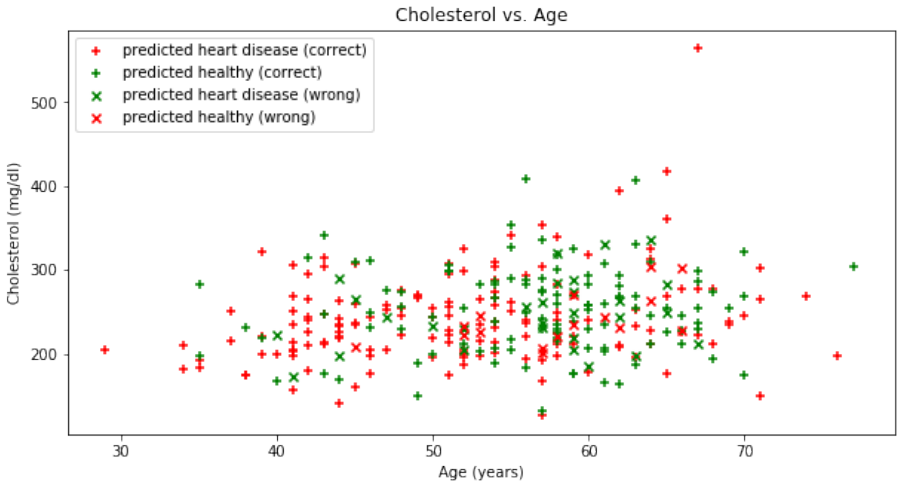

Figure 7: Shows the predicted labels for data points as well as the correlation between two variables (during logistic regression). Similar graphs are included across all jupyter notebooks.

Naïve Bayes

Before Naïve Bayes was performed on the dataset, the same preprocessing was done to normalize the features as done with logistic regression. The same split was done on the dataset, as well as performing k-fold cross validation. This achieved a training accuracy of 84.19%, test accuracy of 80.61%, and an overall accuracy of 84.15%.

K Value	Error
2	.1765
5	.0926
10	.0370

Therefore k = 10 again achieved the optimal error. It achieved accuracy on the total dataset of 84.49% which marginally beat not using k-fold cross validation (84.15%).

True positives (predicted heart disease correctly)	148
True negatives (predicted healthy correctly)	108
False positives (predicted heart disease when healthy)	30
False negatives (predicted healthy when heart disease)	17

Feed Forward Neural Network

The same steps were performed for importing the data and preprocessing as the previous two mentioned approaches. A neural network framework was implemented using PyTorch to allow for networks of varying depth (number of layers), width (number of nodes in each layer), intermediate activation functions, and final activation function. After these were chosen, gradient descent was performed to learn optimal weights from the dataset. Once again, a 90%-10% split of the 303 samples was used. Various combinations were tried and compared to find the best performing weights. Several settings were held constant: intermediate activation being ReLu, final activation being sigmoid, an epoch number of 150,000 during training, and a learning rate of .001.

Number of layers	Layer size	Final loss	Training error	Testing error
3	5	.4220	.0956	.1290
3	10	.4142	.0882	.1613
5	10	.3965	.0809	.1290
3	15	.3995	.0735	.1290
5	15	.3823	.0662	.1613
6	15	.3877	.0735	.1613
7	15	.3882	.0735	.1290
5	20	.3747	.0588	.1290

5 layers and layer sizes of 20 had the best accuracy.

True positives (predicted heart disease correctly)	155
True negatives (predicted healthy correctly)	128
False positives (predicted heart disease when healthy)	10
False negatives (predicted healthy when heart disease)	10

However other settings were only off by 3 to 10 more misclassifications.

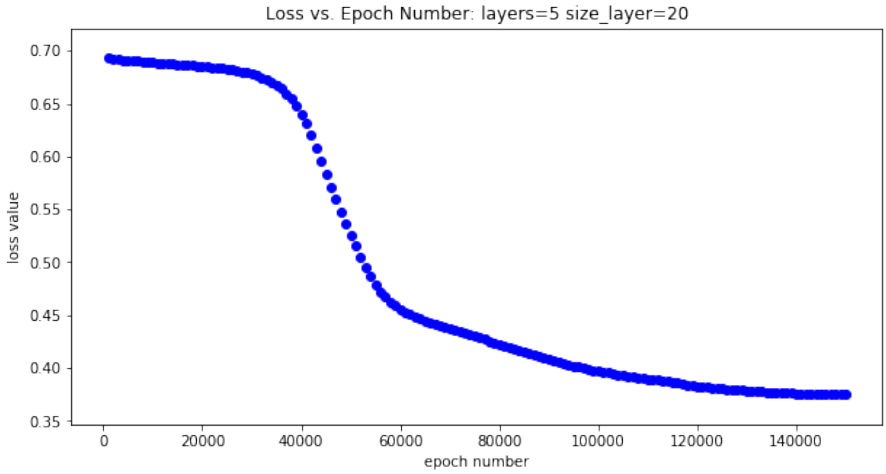

Figure 8: Shows the loss over time due to gradient descent due to the set learning rate. Other graphs are included for other settings in attached jupyter notebook

As shown by the previous tables, the FFNN generally performed better with larger depth and width values. Doing hyper parameter tuning such as these values as well as k-values in the previous approached did reap slight increases in accuracy of the model. It is possible these benefits would be more apparent if done when working with larger sample sizes. The FFNN outperformed for several possible reasons. The dataset wasn't totally separable by a hyper plane, leading to misclassifications. However, a FFNN is able to model deeper relationships than a separating plane by using multiple connected layers.

Participants Contribution

Evan Chang and Nick Seidl both researched which dataset to use and agreed on the UCI Heart Disease one. Code for Logistic Regression and FCFFNN came from the implementation in both of their past homeworks. Both developed the code for Gaussian Naive Bayes, together. Both worked together on introducing the project.

Evan wrote the code for loading and processing the data and the statistical R code, ran experiments with Logistic Regression and Gaussian Naive Bayes, and wrote the experimental results. Nick wrote the code for visualizing and analyzing the results, ran experiments with the FCFFNN, and wrote up the technical approach.

References:

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
https://www.geeksforgeeks.org/naive-bayes-classifiers/
https://piazza.com/class_profile/get_resource/k040nmury17vx/k39cr7zuu1z2ez
https://brilliant.org/wiki/feedforward-neural-networks/
https://machinelearningmastery.com/k-fold-cross-validation/
https://www.codecademy.com/articles/normalization
https://www.kaggle.com/ronitf/heart-disease-uci

Code repository - https://github.com/nseidl/ml_heart_disease