Predicting Heart Disease in Patients

A supervised machine learning approach

Group: Daphne Chen, Andrew Lo, Daniel Medina

Introduction:

According to the World Health Organization (WHO) heart disease is the 1st leading cause of death globally, responsible for approximately 17.9 million deaths annually and 31% of total deaths globally. Over three quarters of heart related deaths take place in low- and middle-income countries. The total cost of heart disease in the United States is over $210 billion according to the CDC. In the United States, someone has a heart attack every 40 seconds.

Problem Statement:

Most cardiovascular diseases can be prevented or stopped from deteriorating by addressing and minimizing behavioural risk factors such as tobacco use, unhealthy diets and obesity, physical inactivity and harmful use of alcohol. People with cardiovascular disease or who are at high cardiovascular risk due to certain risk factors such as hypertension, diabetes, hyperlipidaemia need early detection and management using counselling and medicines. The earlier heart disease is detected in patients, the faster a medical professional can address such patients in order to reduce the risk of a heart attack.

Given some medical information of about 300 patients, we want to know if we can build a model using supervised learning that can effectively predict if a certain patient is at risk of having heart disease. We would like to be able to use the limited amount of health attributes that we have to effectively classify such patients as ‘0’ *(not prone to heart disease)*, and ‘1’ *(prone to heart disease)*. We have a dataset that is approximately balanced, with 165 patients being healthy and 135 patients being suffering from heart disease.

Methods:

Before any of the methods can be used, the first step was to pre-process the dataset in a form that could be fed into the algorithms.To ensure no overflow error, we divided the columns for age, trtbps, chol, and thalachh by 100. This process uses the function **list2onehot** to convert a list of class labels of length *n* into an *n x k* array where the i-th row is the one-hot encoding of *y[i]*. All the categorical variables (which include cp, slp, thall, restecg, and caa) within the dataset were encoded using this described method, thus creating a data frame of features that can be used by each of the machine learning methods.

Logistic Regression

One of the machine learning methods used for heart attack analysis and prediction is logistic regression. After pre-processing, the dataset was split into train and test sets. Train data consisted of approximately 80% of the dataset (denoted **X_train** and **y_train**), while the rest of the 20% were placed into the test set (denoted **X_test** and **y_test**).

One of the core components of logistic regression is the sigmoid (or logistic) function. In python, we defined a sigmoid function that was then used in the creation of a convex optimization algorithm for logistic regression using gradient descent. This function takes in probabilities and a design matrix and uses gradient descent to obtain an optimal parameter **w** such that the loss function is minimized. Once **w** has been optimized, we use it to obtain the predicted probabilities **Q** for **y_test**. These predicted values are then placed in a binary classification accuracy function to assess the accuracy metrics of this logistic regression machine learning algorithm.

Naive Bayes

Another machine learning algorithm we used to classify and predict heart attack risk is naive bayes. Before we can use the naive bayes algorithm on our dataset, we first pre-processed our data in the manner described above. After preprocessing the data, in order to train on the dataset, we need to compute the prior distribution and class conditional probability mass function (PMF). The prior distribution can be easily computed by looking at the percentage of the dataset that falls under each type of output. For the probability mass function, we apply the following to each set of data that corresponds to a specific output. For example, for all the data with an output of 0, we take the sum of each column with a pseudocount of 1 (in case the sum of the entire column is 0). Then, we take the sums of each column and add them altogether to get a complete total. Lastly, we divide each sum of a column and divide it by the complete total. The corresponding array will be the class conditional probability mass function for the output of 0. We then repeat this process for outputs of 1. After having successfully computed the prior and class conditional PMF, we need to test our data. To test the data, we need to be able to make a prediction based on data given. This is done by first taking the class conditional PMF and dividing it by its minimum value to ensure that taking its log will not result in a value too small. We will assign this value to the variable P. Then, we exponentiate the product of the feature vector of X_test and the log of P. Next, we multiply this value by a matrix that consists of the prior distribution repeated in each row with . Finally, we take the sum of the newly computed matrix along its rows and divide the newly computed matrix by this row to normalize the probabilities. The resulting matrix has each row representing the probabilities of each output for its corresponding test data. For example, if the [0,0] entry is .30, this means that there is a 30% chance that the first row of data will have an output of 0. With the predicted probabilities computed we can then compute its accuracy metrics.

Feed Forward Neural Network

The Feed Forward Neural Network model consists of various portions. We first start off with the input layer, where we multiply our initial vectors by some parameter W_0 that is randomly initialized. Then the data is sent through a hidden layer, where it is passed through some sort of activation function. The data is then multiplied by another randomly generated parameter W_1, converting it to our output prediction y. We then attempt to optimize such parameters (W_0,W_1), aiming to minimize our loss function when training on our test data. Minimizing such loss function is done using the X_train data, as we have supervised learning since we have our y_train labels corresponding to the training data. We can then test the efficiency of our model and it's optimized parameters (W_0, W_1) using the X_test data, which will give us some sort of prediction of y.

Note though that our model may have up to some N number of hidden layers between the input layer and the output layer. The optimal number of hidden layers should give us the best metrics, but it is best to choose the lowest number of hidden layers such that our metrics are still mostly optimized in order to conserve computational power and time. In our example, we decided to choose 5, as increasing the hidden layers after this did not result in a significant increase of our metrics, and lowering it showed a small decrease in accuracy. This means that we will have [W0, …., W_5] parameters to optimize, one in between each layer in our model. Note as well that our model uses a combination of three activation functions, the first is simply a dummy activation used for the input layer to create our feature vectors. The last one is the sigmoid function that we also used in Logistic Regression earlier, and the ones in between are tanh(x).

Theory:

Logistic Regression

Logistic regression is a popular classification algorithm that seeks to map an object into one of **k** possible classes. This particular case is of binary classification, where 0 indicates less of a chance of heart attack and 1 indicates more chance of a heart attack. This algorithm aims to model the output as a Bernoulli random variable ***Y ~ Bernoulli(p)*** with success probability *p* that depends on observed features ***φ(x)***. The probability can be more specifically modeled as $p = \sigma(\phi(x)^T w) = \frac{exp(\phi(x)^T w)}{1 + exp(\phi(x)^T w)}$ where σ is the sigmoid/logistic function. The joint probability of observing the output values from independent Bernoulli variables with certain success probabilities is given as the following:

$$L(y_1,\ldots,y_N; w) = P(Y_1 = y_1,\ldots,Y_N = y_n; w) = \prod_{i=1}^{N} p_i^{y_i}(1 - p_i^{y_i})^{1-y_i}$$

To obtain the optimal parameter, solve for the minimum argument of the loss function:

$$\hat{w} = argmin_{w \in R}[l_{LR}(w) := (\sum_{i=1}^{N} log(1 + exp(\phi(x_i)^T w))) - Y^T \Phi^T w]$$

Since the above is a convex optimization problem, it can be solved effectively using an iterative algorithm such as gradient descent. Recall that the gradient is the direction of change in **w** that increases the loss function most rapidly. As a result, in order to minimize the loss function, it would make sense to incrementally change the current **w** in the opposite direction of the gradient. Once the optimal parameter is defined, the predictive probability $p = \sigma(\phi(x)^T \hat{w})$can be computed. This p is a probability, the prediction the logistic regression algorithm makes that the output be either 0 or 1. If the probability is greater than or equal to 0.5, we can classify it as 1, and if it is less than 0.5, we classify it as 0.
For logistic regression to be fairly accurate, there are some assumptions that need to be made. The first is that all the features fed into the model are useful information. Furthermore, observations must be independent of each other and the data used to train the model must be representative of what is being predicted. We assume that the model is not overfitting, meaning it is not too closely aligned to the training dataset. Regularizers can be used to ameliorate such issues; however, logistic regression will fail to classify accurately if the model is overfitting or underfitting.

Naive Bayes

Naive Bayes is another commonly used classification algorithm that instead uses class conditional probabilities. Using Bayes' Theorem, the Naive Bayes algorithm is based on the idea that the predictive probability is proportional to the product of the class conditional distribution and the prior distribution. Using Y as the random variable for the class label and $\phi(x)$as the feature vector, we get the following equation:
$P(Y = i \mid \phi(x) = [\phi_1, \phi_2, \ldots \phi_p]) \propto P(\phi(x) = [\phi_1, \phi_2, \ldots \phi_p] \mid Y = i) P(Y = i)$
The goal is model the discrete random variable $Y|\phi(x)$that has the probability mass function of $[p_1, \ldots, p_k]$where k is the number of class labels. We will use $q_i$to denote the prior probability of a given data being classified as class i and $q_{i,j}$to denote the probability that given a class label Y =i, that feature value will be found. Assuming that all features are independent, the joint likelihood of being the class labels $y_1, \ldots y_n$given the observed features $\phi(x_1), \ldots, \phi(x_N)$ for the parameter W = $(q_i, q_{i,j})$is therefore written as

$$L(y_1, \ldots, y_n; W) = \prod_{s=1}^{N} P(Y_s = y_s | \phi(X) = \phi(x_s) \propto \prod_{s=1}^{N} P(\phi(X) = \phi(x_s) \mid Y_s = y_s) P(Y_s = y_s).$$

We can then express the joint likelihood as

$$L(y_1, \ldots, y_n; W) = \prod_{s=1}^{N} \prod_{i=1}^{k} \left(\prod_{j=1}^{p} P(\phi_j(X) = \phi_j(x_s) | Y_s = i) P(Y_s = i) \right)^{1(y_s = i)}$$

Taking the log of the equation, the optimal parameter $\widehat{W} = (\widehat{q_i}, \widehat{q}_{i,j})$can then be computed from the maximum likelihood method. So,

$$\widehat{W} = arg\, max_w \left[\sum_{s=1}^{N} \sum_{i=1}^{k} p1(y_s = i) log P(Y_s = i) + \sum_{s=1}^{N} \sum_{j=1}^{p} \sum_{i=1}^{k} 1(y_s = i) log\, P(\phi_j(X) = \phi_j(x_s) \mid Y_s = i \right]$$

Here, note that W is subject to the constraints $0 \leq q_1,\ldots, q_k \leq 1,\ q_1 +\ldots + q_k = 1$ and $0 \leq q_{i,1},\ldots, q_{i,M} \leq 1,\ q_{i,1} +\ldots + q_{i,M} = 1\ for\ j \in \{1,\ldots, M\}$. When we solve for this optimization problem, we get that $\widehat{q_i} = \frac{1}{N}\sum_{s=1}^{N} 1(y_s = 1)$ and that $\widehat{q}_{i,j} \propto \sum_{s=1}^{N} 1(y_s = i)\phi_j(x_s)\ for\ j \in \{1,\ldots, M\}$. Once we have successfully trained the values for q_i and $q_{i,j}$, we make a prediction using the maximum a posteriori decision rule. So,

$$x \to A(x) = arg\,max_{i\in\{1,\ldots,k\}} P(Y = i \mid \phi(X) = \phi(x), \widehat{W})$$

$$= arg\,max_{i \in \{1,\ldots,k\}}[\sum_{i=1}^{N} log\,P(Y = i) + \sum_{i=1}^{N}\sum_{j=1}^{p} logP(\phi_j(X) = \phi_j(x_i) \mid Y = y_i)\,]$$

With an easy implementation and fast run time, Naive Bayes can be used for binary and multinomial classifications and handles both continuous and discrete data. In general, since the Naive Bayes algorithm assumes that all features are independent, the algorithm works well when all features in our feature vector are independent. However, this means that if certain features are dependent and we include them in the feature vector, other algorithms may perform better. Furthermore, Naive Bayes only works with nonnegative values for the data. Since our data is all nonnegative though, we do not encounter this issue. In addition, since the algorithm relies on making predictions from observed values in a training dataset, if a variable is not observed in the training dataset, it will predict a probability of zero. This issue can be resolved by using a pseudocount when training the data. Also, for p features in our feature vector, there is a computational complexity of $O(p^2)$. So, if we have large values of p, the algorithm may run extremely slowly.

Feed Forward Neural Network

The Feed Forward Neural Network lays the foundation for many state-of-the-art neural networks used today. We have already discussed the general overview of our model earlier, which states that we want to train our model using X_train data so that our parameters **W_n** are optimized to minimize our loss function given that we know our y_train labels. The general loss function looks something like this, where n is the number of training examples:

$$\hat{w} = argmin_w[l_{NN}(w) := [\sum_{s=1}^{N} l(y_s, \widehat{y}(x_s;\ w)]$$

Where $\hat{w}$ is our array of **W_n** randomly initialized parameters in our model. Now note this is the simple and general formula of our loss function, as $l(y_s, \widehat{y}(x_s;\ w)$ depends on what type of prediction we will be making in our model. Since we want to know if a certain patient has heart disease or not, and our y_train label is binary, we will be using the cross-entropy loss function:

$$l(y_s, \widehat{y}(x_s;\ w) = \sum_{s=1}^{k} 1(y_s = class\ i) log\widehat{y_s}(x_s; w)$$

Given that in classification we normally model the predictive probabilities given for such features so that $\widehat{y_s}(x_s; w) \in R^k$ is a predictive probability mass function of the class of x_s, we can

similarly use ‘cross-entropy’ between two probability distributions. We can see that the class indicator vector $[1(y_s = class\ i),..., 1(y_s = class\ k)]$ is a PMF on the κ classes. Our equation above is cross-entropy between the class-indicator vector with the predictive PMF $\widehat{y_s}(x_s; w) \in R^k$, so that the *ith* coordinate $\widehat{y_i}(x_s; w)$ is the probability that the class of x_s *is i* as claimed by our model.

Now to derive the loss function formula, we use our joint likelihood formula:

$$L(y_1,..., y_n; w) = P(Y_1 = y_1,..., Y_N = y_n; w) = \prod_{s=1}^{n} \prod_{i=1}^{k} (\widehat{y_i}(x_s; w))^{1(y_s=i)}$$

And thus we take the log to get summations and applying a negative sign gives us our final form:

$$-\ logL(y_1,..., y_n; w) = \sum_{s=1}^{N} \sum_{s=1}^{k} 1(y_s = class\ i) log \widehat{y_s}(x_s; w)$$

Note that this optimization problem is not convex, but given the fact that our model has symmetry, as we can swap our nodes and weight parameters respectively, leads to us having at least (#_of_hidden_layers)! ‘equivalent local minima’. From this we can conclude that any such local minimum will suffice the goal of finding a global one. Not only does this simplify our optimization problem greatly, but it allows us to use standard non-convex algorithms to find such minimums.

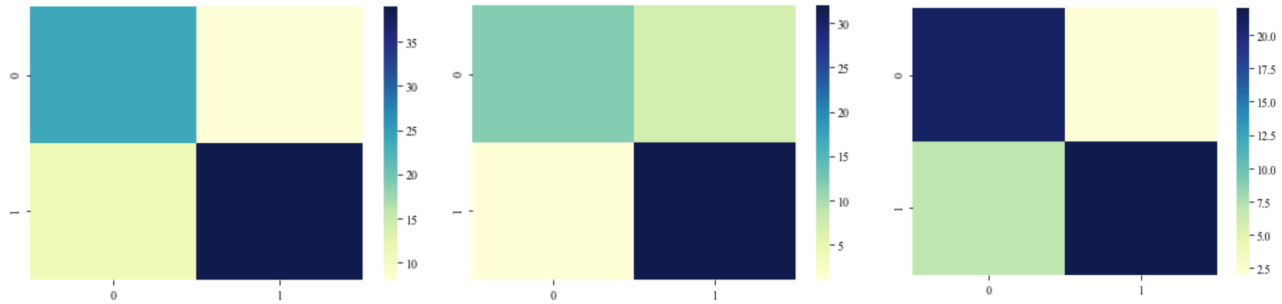
Dataset Description:

The dataset of interest in this paper is “Heart Attack Analysis & Prediction Dataset” found on kaggle. This data comes in two csv files, *heart.csv* and *o2Saturation.csv*. For our machine learning purposes, we focus only on *heart.csv*. The heart data consists of 14 columns; 13 independent variables and 1 output/target variable. The target variable, which is column 14 or the last column of the csv, takes binary values where 0 indicates less chance of a heart attack and 1 indicates more chance of a heart attack. The first column is “age”; it is continuous ranging from 29 to 77. The next column, “sex”, is encoded with binary labels 0 for female and 1 for male. “Cp”, which indicates chest pain type, is categorical with factors 0 (typical angina), 1 (atypical angina), 2 (non-anginal pain), 3 (asymptomatic). The column “trtbps” is the continuous variable resting blood pressure measured in mmHg. “chol” (continuous) measures cholesterol in mg/dl fetched using a BMI sensor. “fbs” is a binary variable with 0 representing that fasting blood pressure is not greater than 120 and 1 being that fasting blood sugar is greater than 120 mg/dl. The next categorical column “restecg” shows resting electrocardiographic results where value 0 is normal, 1 is having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of greater than 0.05 mV), and value 2 is showing probable or definite left ventricular hypertrophy by Estes’ criteria. Column 8, “thalachh”, measures the maximum heart rate achieved. Column 9, “exng”, is a binary column where 1 is “yes” for exercise induced angina dn 0 is “no”. “oldpeak” is a continuous variable for depression induced by exercise relative to rest. The next column “slp” represents slope. The column “caa” is the number of major vessels (0-4). Finally, the last variable is “thall”, indicating thal rate (0-3).

Results:

Logistic Regression

We tested both a binary logistic regression model as well as a multiclass logistic regression algorithm using gradient descent on the heart attack dataset. The codes for both are slightly different but both achieve the same goal of predicting the chance of heart attack. These models were run multiple times to assess accuracy metrics. Some examples of confusion matrices are shown below:



Overall, the model seems to predict chances of heart attack fairly well, with the confusion matrices nearly diagonal.
Furthermore, the accuracies for logistic regression using gradient descent are decent as well, ranging from mid 70% to low 90% with a majority of test cases in the 80% range. Sensitivity and precision also have higher proportions, usually above 0.8. Below are three test cases for reference.

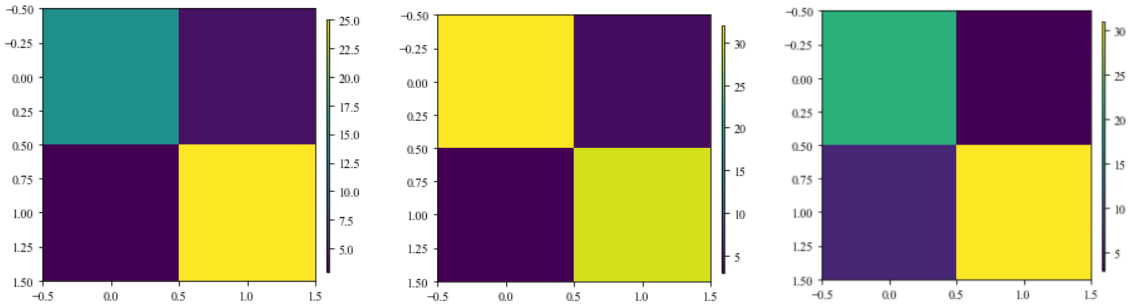
```
AUC = 0.912169
Opt_threshold = 0.000000
Accuracy = 0.806452
Sensitivity = 0.885714
Specificity = 0.703704
Precision = 0.826087
Fall_out = 0.114286
Miss_rate = 0.296296
```

```
AUC = 0.929752
Opt_threshold = 1.000000
Accuracy = 0.909091
Sensitivity = 0.863636
Specificity = 0.931818
Precision = 0.931818
Fall_out = 0.136364
Miss_rate = 0.068182
```

```
AUC = 0.927609
Opt_threshold = 0.000000
Accuracy = 0.775510
Sensitivity = 0.909091
Specificity = 0.666667
Precision = 0.900000
Fall_out = 0.090909
Miss_rate = 0.333333
```

Naive Bayes

Based on the confusion matrices and accuracy data gathered from running the naive Bayes algorithm we can see that it does a good job at predicting the chance of a heart attack. The confusion matrix is always diagonal and most of the cases maintain around an 80% accuracy with 10% or less miss rates. In addition, the precision stays around the mid to high 80%.



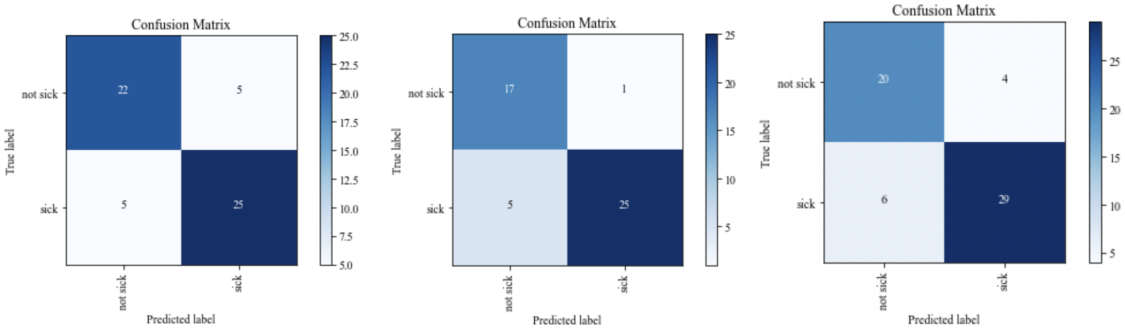
```
AUC = 0.936508
Opt_threshold = 0.649467
Accuracy = 0.847826
Sensitivity = 0.777778
Specificity = 0.892857
Precision = 0.862069
Fall_out = 0.222222
Miss_rate = 0.107143
```

```
AUC = 0.946128
Opt_threshold = 0.458247
Accuracy = 0.898551
Sensitivity = 0.888889
Specificity = 0.909091
Precision = 0.882353
Fall_out = 0.111111
Miss_rate = 0.090909
```

```
AUC = 0.915541
Opt_threshold = 0.456724
Accuracy = 0.852459
Sensitivity = 0.875000
Specificity = 0.837838
Precision = 0.911765
Fall_out = 0.125000
Miss_rate = 0.162162
```

Feed Forward Neural Network

To find the optimal number of hidden layers, I decided to test on the following hidden layers: [1,2,3,4,5,6,7,10,15,20,25]. For each hidden layer #l computed it ten times and then calculated the average accuracy among the ten runs. I found that accuracy stopped increasing significantly after 5 hidden layers, giving us an average accuracy of about 84.4. Below are the metrics of some of our results:




```
AUC ===> 0.922
Opt_threshold ===> 0.545
Accuracy ===> 0.825
Sensitivity ===> 0.815
Specificity ===> 0.833
Precision ===> 0.833
Fall_out ===> 0.185
Miss_rate ===> 0.167
```

```
AUC ===> 0.978
Opt_threshold ===> 0.426
Accuracy ===> 0.875
Sensitivity ===> 0.944
Specificity ===> 0.833
Precision ===> 0.962
Fall_out ===> 0.056
Miss_rate ===> 0.167
```

```
AUC ===> 0.907
Opt_threshold ===> 0.358
Accuracy ===> 0.831
Sensitivity ===> 0.833
Specificity ===> 0.829
Precision ===> 0.879
Fall_out ===> 0.167
Miss_rate ===> 0.171
```

Overall, since all 3 algorithms were able to generate around 80+% in accuracy, we were able to learn and predict heart attack risk. These results make sense as our dataset conformed to the requirements of each algorithm such as non negative values in the data. Furthermore, we had tested three different machine learning algorithms, all yielding similar results and accuracies. The fact that it works presents the opportunity for future use in the medical field, though more improvements may be necessary for real world application. Especially with a larger dataset to train on, our algorithm may do even better. Based on results found in https://www.kaggle.com/nimajehan/heart-attack-analysis-with-diff-models, while different algorithms were used, we can see that their results ranging around 80-85% is very similar to ours.

Conclusion:

In the end, we were able to successfully build a model that predicts heart attack risk based on specific patient information. In each of the methods we used, we were able to get around 80-90% accuracy. With logistic regression, it seems that we get more variability in the accuracy results with the accuracy being able to go up to 90%, but other times going to 77%. In comparison, naive Bayes generated more consistent results staying around 85%. Similar to naive Bayes, the feed forward neural network also maintained consistent accuracy around 85%. Logistic regression seems to have higher precision levels at around 90%. On the other hand, naive bayes seem to get better specificity at close to 90%. Lastly, the feed forward neural network seems to have the least amount of variability after running the algorithm multiple times compared to the previous two. However, in general, all 3 algorithms result in fairly similar results in accuracy with only slight variations.

From our data, we also found that some of the biggest factors that related to heart attack risk are cp, thall, and old peak play some of the biggest roles in determining heart attack risk. This makes sense because the type of chest pain one has, the effect exercise has on a person, and whether or not a person has a defect should be major indicators of heart attack risk.

While around 80% accuracy is good, we could further improve the accuracy results with a larger dataset so that we could train our models more. In addition, the specific patient information that we chose for our feature vectors could be improved upon. There may be key factors that play into heart attack risk that are not in our feature vector. In addition, certain features may be dependent on one another, but we assumed the features to be independent which may affect the accuracy of our results. Thus, while our models do a good job of predicting heart attack risk, more can be done to further improve our model and get more accurate predictions. Hopefully, this will allow us to detect heart disease earlier on and give us the opportunity to set up preventative measures for those that are at risk.

- **age:** Age in years
- **sex:** (1 = male; 0 = female)
- **cp:** Chest pain type (0 = asymptomatic; 1 = atypical angina; 2 = non-anginal pain; 3 = typical angina)
- **trestbps:** Resting blood pressure (in mm Hg on admission to the hospital)
- **cholserum:** Cholestoral in mg/dl
- **fbs** Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)
- **restecg:** Resting electrocardiographic results (0= showing probable or definite left ventricular hypertrophy by Estes' criteria; 1 = normal; 2 = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV))
- **thalach:** Maximum heart rate achieved
- **exang:** Exercise induced angina (1 = yes; 0 = no)
- **oldpeakST:** Depression induced by exercise relative to rest
- **slope:** The slope of the peak exercise ST segment (0 = downsloping; 1 = flat; 2 = upsloping)
- **ca:** Number of major vessels (0-4) colored by flourosopy
- **thal:** 1 = normal; 2 = fixed defect; 3 = reversable defect
- **Sick:** Indicates the presence of Heart disease (True = Disease; False = No disease)