Machine Learning Models for Heart Disease Prediction*

Xin Xiang, Michael Pacocha, Matthew Barroso†
(Dated: December 11, 2021)

Over 17.9 million people every year die of heart disease making it the worlds number one killer. Many preventative measures can be administered when properly diagnosed. In this paper, we take a look at a using machine learning methods to determine if a particular patient is likely to have heart disease based on a few variables including age, cholesterol, and blood pressure.

I. INTRODUCTION

Machine learning is used to make predictions on particular outcomes based on previous data.

We chose to try and classify whether or not a patient would have heart disease based on various parameters. We use the data from [1]. The data has various features, all of which are shown in fig. 1. This data also includes whether or not the patient had heart disease.

We use different machine learning methods to try and classify our data in order to see which method is best. We look at decision trees, naive Bayes, and neural networks. For decision trees, We build two base tree classifiers using Information gain and Gini Index as splitting criterion and two boosted classifiers using Bootstrap aggregating algorithm and adaptive boosting algorithm. For naive Bayes, we calculate prior probabilities and likelihoods from our data, and use Bayes theorem to determine the most likely outcome.

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat	1
914	68	M	ASY	144	193	1	Normal	141	N	3.4	Flat	1
915	57	M	ASY	130	131	0	Normal	115	Y	1.2	Flat	1
916	57	F	ATA	130	236	0	LVH	174	N	0.0	Flat	1
917	38	M	NAP	138	175	0	Normal	173	N	0.0	Up	0

918 rows × 12 columns

Attribute	Description
Age	age of the patient [years]
Sex	Gender of the patient [M: Male, F: Female]
ChestPain	chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
RestingBP	resting blood pressure [mm Hg]
Cholesterol	serum cholesterol **[mg/dl]**
FastingBS	fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
RestingECG	resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
MaxHR	maximum heart rate achieved [Numeric value between 60 and 202]
ExerciseAngina	exercise-induced angina [Y: Yes, N: No]
Oldpeak	oldpeak = ST [Numeric value measured in depression]
ST_Slope	the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
HeartDisease	output class [1: heart disease, 0: Normal]

FIG. 1. The overview of our data and the explanations for each feature

* CS 4641 - Machine Learning - Final Paper Project. Code and all the files can be access here: The link to our code

† Georgia Institute of Technology.

II. THEORETICAL CONSIDERATIONS

A. Decision Trees, Bagging and Boosting

In this section, we show the algorithms of three different decision trees. The first one is the basic decision tree classifier using either Gini Impurity or Information Gain as the splitting criterion. The other two are more advanced classifiers using Bootstrap Aggregation Algorithm and Adaptive Boosting Algorithm. The latter two are forms of ensemble learning.

1. *Gini Impurity and Information Gain*

Gini impurity, or Gini index, measures the probability of misclassifying a variable when it is randomly chosen. If all the elements belong to a single class, then it can be called pure. The degree of Gini index varies between 0 and 1, where, 0 denotes that all elements belong to a certain class (the node is pure), and 1 denotes that the elements are randomly distributed across various classes (the node is impure). The Gini index can be calculated by [2]:

$$G_i = \sum_{c=1}^{C} \hat{\pi}_{ic}(1 - \hat{\pi}_{ic}) = 1 - \sum_{c} \hat{\pi}_{ic}^2 \tag{1}$$

, where $\hat{\pi}_i$ is the probability a random entry in the leaf belongs to class c. While building the decision tree, we would prefer choosing the feature with the least Gini index to spilt on. Alternatively, the entropy can also measure the cost or deviance of the node:

$$H(Y) = -\sum_{i=1}^{k} \hat{\pi}(y_i) log \hat{\pi}(y_i) \tag{2}$$

Information gain computes the difference between entropy before split and average entropy after split of the data set based on given attribute values. Information gain is the decrease in entropy:

$$IG(X) = H(Y) - H(Y \mid X) \tag{3}$$

, where $H(Y \mid X)$ is the conditional entropy given by:

$$H(Y \mid X) = -\sum_{j=1}^{v} \hat{\pi}(x_j) \sum_{i=1}^{k} \hat{\pi}(y_i \mid x_i) log \hat{\pi}(y_i \mid x_i) \tag{4}$$

The attribute X with the highest information gain, IG(X), is chosen as the splitting attribute at the node. The tree models are easy to interpret and can easily handle mixed discrete and continuous inputs. However, trees are unstable. Even small changes to the input data can have large effects on the structure of the tree. Also, they are very likely to over-fit with the training data due to the greedy nature of the tree construction algorithm. In other words, decision trees are a high variance estimator. A way to reduce variance is to average multiple tree models. This is called ensemble learning[2]. We will introduce two ensemble learning algorithm called Bagging and AdaBoost in the following two subsection.

2. *Bootstrap Aggregation Algorithm (Bagging)*

Bootstrap aggregating is a simple form of ensemble learning. A number of base classifiers (e.g., the base decision trees introduced in the previous section) are trained on random subsets of the original data set and their individual predictions are aggregated to form a final prediction. This final bootstrap aggregating classifier is an ensemble meta-estimator that can be used as a efficient way to reduce the variance of the base estimator. It can prevents the model from relying too much on individual training example. Bagging algorithms draw the subdata set with replacement [3], so a given example may appear multiple times, until we have a total of N examples per model, where N is the size of the original training set. Then we apply the decision tree algorithm to the sample and store the resulting classifier. The final estimator will return the class for each instance that is predicted most often for all the base estimators.

3. *Adaptive Boosting Algorithm (AdaBoost)*

The Adaptive Boosting is another form of ensemble learning. A classifier (e.g., a base decision tree) is built first for the unweighted original training sample. If there is a misclassified data point, the weight of that training data point is increased (boosted). Then a second classifier is built using the data set with new weights. Many classifiers are built sequentially using this repeatedly procedure. A final classifier is build by the linear combination of all the classifiers learned previously. The algorithm known as AdaBoost-SAMME [4] is used in this project.

B. Naive Bayes Classifier

The idea for Naive Bayes is to take a probabilistic approach to determine a decision. We start from Bayes theorem, given by

$$P(Y = y_i|X_1, ..., X_n) = \frac{P(Y = y_i)P(X_1, ..., X_n)}{P(X_1, ...X_n)} \quad (5)$$

Where y_i are the possible outcomes, X_n are the features, each of which can take on different values. In qualitative terms, this equation is saying that the probability of a particular outcome happening based on values of some features that occurred is equal to the probability of that outcome happening (called the prior) multiplied by the probability of those features happening when that outcome happened (likelihood) divided by the probability of those features happening (evidence). Given some previous data, the prior and evidence are easy to calculate. For each outcome y_i, count each time that outcome happened and divide by the size of your data. The same goes for some feature X_i. It gets more complicated for likelihood, because certain features may be conditionally dependant.

We wish to find an expression for $P(Y = y_i|X_1, ..., X_n)$. The Naive part of Naive Bayes comes from the assumption that all variables X_n are conditionally independent, i.e.

$$P(X_1, ..., X_n|Y) = \prod_i P(X_i) \quad (6)$$

Therefore, our expression for our classifier becomes

$$P(Y = y_i|X_1, ..., X_n) = \frac{P(Y = y_i)\prod_i P(X_i)}{P(X_1, ...X_n)}$$

Notice, when trying to classify some data, the denominator is the same for each guess at an outcome, thus can be ignored. Therefore we have

$$P(Y = y_i|X_1, ..., X_n) \propto P(Y = y_i)\prod_i P(X_i) \quad (7)$$

Now with this assumption, to calculate the likelihood of some event, we take previous data and count each time a feature takes a value for a particular y_i and divide that by the number of times that y_i occurred, i.e.

$$P(X_i = x|Y = y_i) = \frac{Count(X_i = x, Y = y_i)}{\sum Count(X_i = x', Y = y_i)} \quad (8)$$

Finally, to make a prediction on the outcome, we take the argmax of $P(Y = y_i|X)$, so if a particular set of likelihoods for a given y_i multiplied by the prior probability of that y_i has the highest value, we choose that particular y_i to be the decision.

This assumption that the variables are conditionally independent is generally not true , especially in the case of predicting heart disease. And yet, Naive Bayes is often the most used classifier. It sometimes still performs well on data that is still conditionally dependent. Therefore, we will analyze it as a method of classifying our data.

1. *Continuous Data and Zero Frequency Problem*

Calculating likelihoods is easy for data that takes discrete values, like gender or whether or not coin lands on

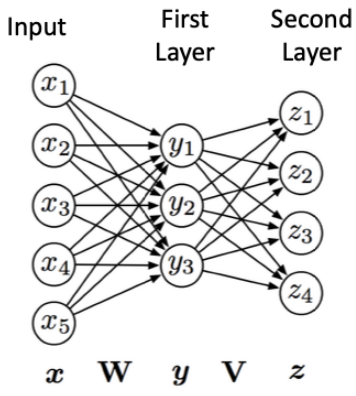

FIG. 2. Example of a multi layer neural network with at least two hidden layers

heads. But for variables that take on continuous values, like heart rate, it is very possible to not have examples for particular values, resulting in a likelihood of zero.

Instead of calculating likelihoods directly from the data, we can assume a probability distribution based on the data [5]. The most common distribution is a normal/Gaussian distribution (hence the name), which takes on the form

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{9}$$

The parameters of this Gaussian can be chosen from a given set of training data. μ is given by average of the data and σ is the standard deviation.

This is a probability distribution, the probability of an event occurring within a range $x + \delta x$ is given by

$$\int_x^{\delta x} p(z)dz \tag{10}$$

We can use this probability for $P(X_i|Y)$ for our result.

C. Neural Networks

The purpose of using a neural network is to be able to extend linear models to represent nonlinear models. This is achieved by using nonlinear functions to transform the decision space. The general architecture for a neural network can be seen in Fig. 2. In this example, there is the input vector $\vec{x}$, first layer vector vector $\vec{y}$, second layer vector $\vec{z}$, and weight vectors vector $\vec{W}$ and $\vec{V}$.

From the input vector, we can calculate the vectors for the other two layers by applying a linear function. The first layer becomes $\vec{y} = \vec{W}\vec{x} + b$ and the second layer become $\vec{z} = \vec{V}\vec{y} + c = \vec{V}(\vec{W}\vec{x} + b) + c$. However, this current model does not allow for the model to represent nonlinear functions as it is comprised solely of linear operations. To mend this, a nonlinear function, g, is applied to each layer resulting in $\vec{y} = g(\vec{W}\vec{x} + b)$ and $\vec{z} = g(\vec{V}(g(\vec{W}\vec{x} + b)) + c)$. The process of calculating the values each layer is called the forward pass.

The choice of the nonlinear function is entirely dependent on the implementation of the algorithm and the geometry of the true decision boundary. In theory, if a function that is similar to the true decision boundary is used for the nonlinear function, then the results of the neural network will be better or converge quicker. In most cases, we wish to learn this decision boundary, so it is common to use some standard ones. A few of these are the: Sigmoid function - $\sigma = \frac{1}{1+e^{-x}}$, Rectified Linear Unit (ReLU) - $max(0, x)$, and the hyperbolic tangent function - $\tanh x$.

While the nonlinear functions allow for nonlinear decision boundaries, the weight vectors are what are used to learn from a model. To update these weights, we apply the back propagation algorithm which is essentially just gradient descent. A forward pass is done to calculate each node value N_k. At the end of the pass, we calculate $\delta_\mathrm{o} = N_\mathrm{o}(1 - N_\mathrm{o})(p - N_\mathrm{o})$ each output unit o with output p. For each hidden unit h, we calculate $\delta_\mathrm{h} = N_\mathrm{h}(1 - N_\mathrm{h})\Sigma_\mathrm{k} w_{\mathrm{h,k}}\delta_\mathrm{k}$. Finally, we can update these weights using $w_{\mathrm{i,j}} = w_{\mathrm{i,j}} + \eta\delta_\mathrm{j} x_{\mathrm{i,j}}$.

III. METHODS

A. Data Processing

The data with categorical features in the original data set (Sex, ChestPainType, RestingECG, ExerciseAngina, ST_Slope) are converted to numerical feature using OrdinalEncoder() from sk-Learn. By reading the data file, we noticed that there is a large amount of zero inputs for Cholesterol and one zero input for RestingBP. Since a person's Cholesterol level and resting blood pressure are impossible to be zero, we assume that these zero inputs are collected incorrectly. For one set of data, we include these points, and for another we exclude the data with zero inputs for Cholesterol and RestingBP. For the third data set, a linear regression from sk-Learn was used to predict the missing cholesterol values using the other data given by the users. This method was found to be within 20.9% of the true value. This method also only minimally changed the distribution of the data, changing the average by .484%. The cholesterol values' mean, median, and mode can be see in Fig. I for the three data sets. We use 80% of our data as training data and 20% as testing data for decision trees and Naive Bayes Classifier. For Neural Network, the data was split 70% train, 20% test, and 10% dev set.

B. Decision Trees

All the decision trees classifiers are built using sk-learn.

	Mean	Median	Mode
Dataset #1	199.02	223.00	0
Dataset #2	244.63	237.00	254
Dataset #3	243.45	236.00	225

TABLE I. This table shows the cholesterol values' mean, median, and mode for each of the three data sets. Data set #1 is with all data, Data set #2 is with the zero cholesterol values removed and Data set #3 is with the predicted values for cholesterol.

1. *The Baseline Decision Tree*

We built two trees using Gini Index and Information Gains as splitting criterion. The GridSearchCV method in sk-learn is used to do the hyperparameters tuning for the minimum number of leaves and the maximum depths of the tree. We search for the best combination of the two hyperparameters in the list of values of [2,4,8,16,24,32,40,48,56,64,72] and [4,5,6,7,8,9,10,11,12] for the minimum number of leaves and the maximum depths of the tree respectively.

2. *Bagging Classifier*

We use the Gini Index decision trees as base estimators for the Bagging. We choose the maximum samples that are drawn from the original data set to be equaled to 250. The same method is used to do the hyperparameters turning for the maximum depth and minimum leaves of base estimators, the number of estimators. The values lists for these four hyperparameters are [4,5,6,7,8,9,10], [1,2,4,8,16,24], [500, 1000, 1500] respectively.

3. *AdaBoost Classifier*

Similarly, we use the Gini Index decision trees as base estimators for the AdaBoost Classifier. We turn the hyperparameters for the the maximum depth and minimum leaves of base estimators, the number of estimators, and the learning rate. The values lists for these four hyperparameters are [4,5,6,7,8,9,10,11,12], [2,4,8,16,24,32,40,48,56,64,72], [1,2,3,4,5,6,7,8,9], and [0.01, 0.04, 0.08, 0.12, 0.16, 0.20] respectively.

All the hyperparameters may be manually turned. More details are discussed in the Results section.

C. Naive Bayes Classifier

To analyze our data, we will use a few different methods since each feature has a different distribution. First, calculating a prior probability is easy, we count each instance someone did or did not have heart disease and divide it by the length of the data. Since there are only two outcomes, we simply make 2 variables, priorPClass and priorNClass for the prior probabilities.

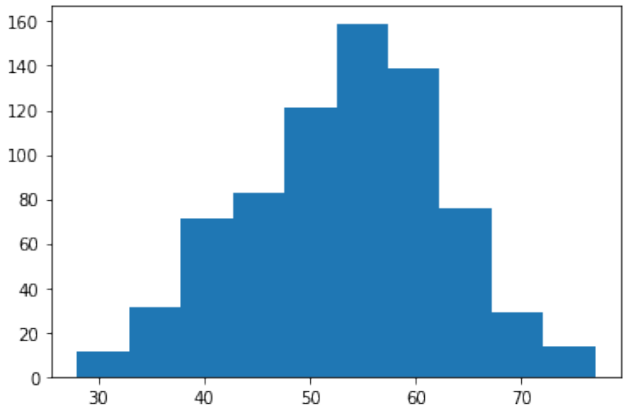

FIG. 3. Distribution of age

The more complicated part if calculating the likelihood. For each feature class, we need to count each possibility for that class when the outcome was a particular value. For instance, if we had 5 patients, and their sexes were [M,M,F,M,F], and the outcome for each was [1,0,0,1,1], we would have 4 likelihoods $P(X = M|Y = 1) = 2/3$,$P(X = F|Y = 1) = 1/3$, $P(X = M|Y = 0) = 1/2$, $P(X = F|Y = 0) = 1/2$. This is simple for features that take on discrete values, we simply count and divide by the times that an outcome happened. For our data there were 6 features that were simple, namely "Sex", "ChestPainType", "FastingBS", "RestingECG", "ExerciseAngina", "ST_Slope". The rest of the data we have to do something different.

For our harder, continuous data, we have to decide what sort of distribution to fit to our data. Most of the continuous data looks like a normal distribution, as shown in fig. 3.

As discussed in sec. II B 1, continuous data poses a problem that we might not see an example for every possible value, and even if we do it might not be very representative of its true probability. In our data, heart rate could take on values of 60-180, but it is not very likely we see every single value in our training set. So instead, we fit a Gaussian to the data and use that to find probabilities. This safely avoids the zero frequency problem. Our data has 4 features that take on continuous, Gaussian distributions, namely "Age", "MaxHR", "RestingBP", "Cholesterol".

Cholesterol in particular has a weird feature that there is a very large peak at zero and then looks Gaussian after, we take two strategies to analyze this, the first is to simply remove the zeros and just use the Gaussian, the other is to include the zeros in a separate, discrete likelihood, and include the Gaussian as well. A slight caveat to this strategy is that the sum of likelihoods for a particular $P(X_i = x|Y = y_i)$ has to sum to one for each x. So we divide the Gaussian for cholesterol by

$1 - P(Cholesterol = 0|Y = y_i)$. We do this because our normal Gaussians are defined by

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{11}$$

So in order for the total likelihood to add up to one, we must divide the Gaussian by the contribution of the 0 likelihood.

There is one last feature that does not look nicely Gaussian, namely "Oldpeak". So instead of fitting a nice function, we simply separate the data into even bins.

Finally, after calculating all the likelihoods, we put them all into a dictionary to use for later. We then loop over the test data, referencing the dictionary for the likelihood of each feature, multiply for each likelihood and then multiply the probability of the outcome. We put the result for each example in a vector and dot this with the actual outcome to find our accuracy.

D. Neural Network

The neural network used for this project was built using PyTorch and optimized with Optuna. In this part, the train set was used to train the neural network, the dev set was used to report accuracy of the epochs and to tune with Optuna, and the test set was used to report the accuracy of the trained neural network.

For the neural net, PyTorch was used due to our familiarity with the package from the course. Two main combinations of layers were used. One uses a dropout layer due to its use in preventing over fitting.

For the first combination: 1. Input layer has the linear transformation $y = xA^{\mathrm{T}} + b$ applied resulting in a hidden layer 2. The ReLU nonlinearity is applied 3. If there is another layer, repeat from 1. If not, another linear transformation is applied resulting in an output layer 4. The log softmax is applied to turn the output layer into a probability distribution.

For the second combination: 1. Input layer has the linear transformation $y = xA^{\mathrm{T}} + b$ applied resulting in a hidden layer 2. The ReLU nonlinearity is applied 3. A dropout layer is applied 4. If there is another layer, repeat from 1. If not, another linear transformation is applied resulting in an output layer 5. The log softmax is applied to turn the output layer into a probability distribution.

The tuning was done with Optuna over RayTune, HyperOpt, Keras Tuner, and SciKit due to ease of use and built in trial pruning. Since Optuna is known for its efficiency and ability to prune runs that are not promising, we were able to tune many parameters. For each of the three data sets, we let Optuna run 400 trials. Of these 400 trials, 200 included the dropout layer and the other 200 excluded it. For each trial, Optuna was allowed to optimize the number of layer sets, the number of nodes in each hidden layer, (if there was one) the probability for the dropout layer, the learning rate of the Adam optimizer, and the number of epochs.

Once the best parameters were calculated by Optuna, the values were used in the neural network. The networks were then tested 50 times each with the data randomized each time.

IV. RESULTS

A. Decision Trees

1. *The Baseline Decision Tree*

The best tree we found for the baseline decision tree for Dataset #1 has the maximum depths of 5 and minimum leaves of 16. The structure of the tree using information gains as splitting criterion is showed in figure 4. The first three important features are ST_Slope, ChestPainType, and Cholesterol. The split on Cholesterol is 42.5. According to the article [6], the healthy level for Cholesterol should below 200mg/dL. Apparently the Dataset #1 's average inputs for Cholesterol is lower than normal. This can be due to the wrong zero inputs. For the Dataset #2 and #3, the first three important features become 'ST_Slope', 'ChestPainType', and 'Sex'. The splitting on Cholesterol is increased to 266 and 232.5 for set #2 and #3 respectively. Thus, we can confirm that the data of zero values for Cholesterol are collected incorrectly. And the prediction for the Cholesterol values by the linear regression is performing very well. The accuracy for all the test sets are 0.83 ± 0.3, and 0.86 ± 0.1 for all the training set.

2. *Bagging Classifier and AdaBoost Classifier*

For the dataset #1, The best hyperparameters sets of the bagging classifier we found for the maximum depth and minimum leaves of base estimators, and the number of estimators are 6, 1, 1500 respectively. For dataset #2 and #3, the best hyperparameters sets are 4, 4, 1500; and 7, 2, 500. The accuracy are around 0.90. However, for the AdaBoost, we get an accuracy of 1.0 for the training set of the data #2 and 0.77 for testing set. This indicates extremely bad over-fitting happens. The table 5 summarizes the accuracy for all the decision tree algorithm we have done for the original hyperparameters setup. To avoid the over-fitting, we change the base estimator for the AdaBoost to has smaller depths and leaves (weaker base trees). We found that when the maximum depth equals to 3 and minimum leaves equals to 10, the AdaBoost Classifier performs better.

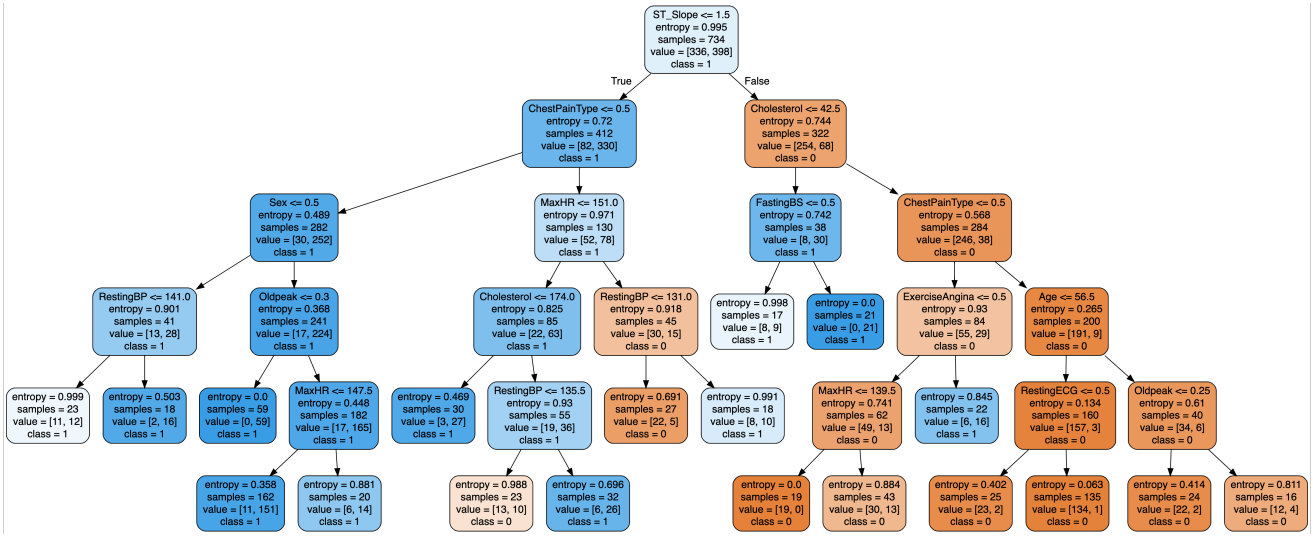

FIG. 4. The Baseline Decision Tree

Dataset #1

Decision Trees	Accuracy_Train	Accuracy_Test
Entropy	0.86649	0.83696
Gini	0.86921	0.85870
Bagging	0.91417	0.89130
AdaBoost	0.88420	0.90761

Dataset #2

Decision Trees	Accuracy_Train	Accuracy_Test
Entropy	0.86745	0.82000
Gini	0.87248	0.80667
Bagging	0.89597	0.87333
AdaBoost	1.00000	0.76667
AdaBoost (weaker base)	0.90101	0.88667

Dataset #3

Decision Trees	Accuracy_Train	Accuracy_Test
Entropy	0.86630	0.82065
Gini	0.86767	0.86957
Bagging	0.92360	0.88587
AdaBoost	0.89359	0.85326

FIG. 5. The accuracy for decision trees

B. Naive Bayes Classifier

Our accuracy on the test set was 78.3%. Considering the assumption of conditional independence, this is a decent result. Despite the fact that these features are not at all independent from one another, our result was still decently accurate. Using the different data sets #2 and #3 yielded little to no difference in accuracy.

C. Neural Networks

Out of all of the tests ran, the best accuracy was 89.6% given by the neural network without a dropout layer and with the predicted cholesterol levels. The best for the neural network with the dropout layer was with the unmodified data at 87.4%.

Since we are predicting heart disease, we really want to avoid giving false negatives. If implemented in real life, a false positives would not harm anyone. It would likely result in someone being examined further by a doctor. A false negative, however, would mean someone who has a problem would not get the attention they need. Because of this, we decided to do further analysis on how accurate the positive and negative readings were.

Yet again the better decider was the neural net without the dropout layer and on the predicted cholesterol at 89.1% overall accuracy, 91.7% accuracy on predicting correctly those who do have heart disease, and 85.1% on those who don't. The best for the neural net with the dropout layer was on the no cholesterol at 81.1% overall accuracy, 91.8% accuracy on predicting correctly those who do have heart disease, and 70.7% on those who don't.

A broader analysis of the many tests done on each neural network revealed that the inclusion of the dropout layer did not seem to improve accuracy. Surprisingly, the neural networks with the dropout layers seemed to be more inclined to over fitting on all data sets. There were some interesting cases in the dropout layer tests however. There were some that had +95% accuracy in correctly identifying those with heart disease, but 50-60% in correctly identifying those without heart disease. While it seems to do the important job of identifying those with a problem, it is also likely to increase doctor work load through false positives. On the other hand, the neural networks without the drop out layer, they all seemed to be fairly accurate and had pretty good accuracy on both the positive and negative cases, 85%.

V. CONCLUSION

All of our methods did a satisfactory job at properly classifying those who might have heart disease. The best of which being the AdaBoost Classifier for dataset #1 with the accuracy of 91% for the testing set.

From the accuracy table of decision trees, we found that the Bagging algorithm has the most stable results. Although the accuracy of training is always larger than that of the testing, the improvement could have been done. The Bagging's base model only select a portion of the original data. A large amount of the training data (around 30%) are not used by the given base model. The cross validation can be done for these out-of-bag instances to improve the performance of the Bagging.

In the analysis of the neural network data, we found that there were networks that had very accurate positive predictions and many false negatives. An interesting continuation of this project might involve a follow up study that sees if any of the negative patients later developed heart disease. That could then be compared to the cases the were previously false positives. This would mean that the particular neural network might also be good at predicting at risk individuals.

[1] Fedesoriano, Heart Failure Prediction Dataset (2021).
[2] K. P. Murphy, *Probabilistic machine learning: an introduction* (The MIT Press, 2022).
[3] L. Breiman, Bagging predictors, Machine Learning **24**, 123–140 (1996).
[4] T. Hastie, S. Rosset, J. Zhu, and H. Zou, Multi-class adaboost, Statistics and Its Interface **2**, 349–360 (2009).
[5] T. Gupta, Continuous Data and Zero Frequency Problem in Naive Bayes Classifier (2020).
[6] Cholesterol: The good and the bad (2021).
[7] C. Moffitt, Guide to Encoding Categorical Values in Python (2017).