FYS-STK 4155 Project 3 H22
Prediction of Heart Disease Diagnosis with Neural Network and Random Forest machine learning models

Lasse Totland, Domantas Sakalys, Synne Mo Sandnes Semya A. Tønnessen

(Dated: December 18, 2022)

This report covers the development of two binary classification machine learning models for prediction of heart disease diagnosis using the 1988 Cleveland Clinic Foundation survey data on heart disease among hospital patients. The data set contains 14 attributes relevant to our analysis and we determined that none of them should be extracted during preprocessing as all of them contributed positively on the prediction test accuracy of our models. Using a classification Feed Forward Neural Network with back-propagation (and activation functions RELU for one hidden layer and softmax for output layer) we achieved an accuracy of up to 91% with learning rate $\eta = 0.01$ and regularization factor $\lambda = 0.01$ using hyperparameter analysis. Using a Random Forest Classifier with Adaptive Boosting (500 estimators) we achieved a test accuracy of 88% with $\eta = 0.1$ and max tree depth of 1 using hyperparameter analysis, as well as a 89% true positive prediction accuracy. These results are comparable with established literature analysing the same data set.

I. INTRODUCTION

Data science and machine learning tools can be useful in the medical field. They support complex clinical decision-making and could be helpful in the prediction of various ailments, such as heart decease where risk factors such as high blood pressure and cholesterol can be considered. While there are still barriers to adopting machine learning in the medical field, such as lack of resources and the importance of finding suitable mathematical models, in addition to privacy concerns, the method has considerable potential to transform healthcare.

Heart disease refers to a group of conditions that affect the heart. According to World Health Organization data, cardiovascular diseases are now the leading cause of mortality globally, accounting for 17.9 million deaths per year [1]. Various harmful activities, such as an unhealthy diet, use of tobacco, lack of physical activity and alcohol abuse, can increase the risk of heart disease. Other factors that may rise the risk are excessive cholesterol, obesity, increased triglyceride levels, hypertension, high bloodsugar and so on [1]. The American Heart Association cites some symptoms such as sleep problems, an increase and drop in heart rate (irregular heartbeat), swollen legs, and in certain cases rapid weight gain [2]. All of these symptoms mirror various diseases, such as those seen in the elderly, making it difficult to establish a precise diagnosis, which might lead to death in the near future.

In this study, we aim to predict the presence of heart disease in patients using machine learning techniques. We used the 1988 Heart Disease data set from the Cleveland Clinic Foundation, which consists of 297 patients and 14 features. Our approach involved training a feedforward neural network and a decision tree on this data set to classify patients as either having heart disease or not. Through our analysis, we sought to determine which of these two models would be more effective at predicting heart disease in this population.

When evaluating the final models we will compare their performance with established literature studying the same data set. The research article we compare with achieved test accuracies of 88% using Random Forest classification and 86.8% using Deep Learning classification [3].

In section II we introduce the relevant theory in detail. The data set and preparation are presented in section III, as well as the models that have been used. Our results and a discussion of these are presented in section IV, before we go over the conclusion and final outlook in section V.

All python code referred to in this report can be found in the GitHub repository located at https://github.com/lassetotl/fys-stk/tree/main/project%203.

II. THEORY

A. Bivariate correlation

In statistics, bivariate correlation is a measure of linear correlation between two sets of data. If the values of one set increases as the other one also increases, then they have significant positive linear correlation. If one *decreases* as the other increases they have negative linear correlation. The formula for bivariate correlation will thus score positive or negative values in the range $r_{xy} \in [-1,\ 1]$ (values close to 0 will have no significant correlation), and is defined by the Pearson method for data sets x and y as:

$$\mathrm{r}_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}, \qquad (1)$$

where n is sample size, x_i and y_i are sample points of index i and $\bar{x}$ and $\bar{y}$ are the sample mean values of each set [4].

B. Feed Forward Neural Network

A Feed Forward Neural Network (FFNN) is a neural network that processes data in one direction. We feed it with an input, and the network processes the input to the output. This happens without any loops. The weights and biases of the connections between neurons and layers are adjusted during the training that minimizes error between the predicted and desired output. One example of a task where this type of network is used, is a classification task. The algorithm was previously discussed in further detail in project 2 [5].

C. Hyper parameters

The regularization factor λ is used to control the complicity of the model. It adds a penalty term to the loss function when it is being optimized during the training process. One common form of regularization is called L2 regularization, which adds a penalty term to the loss function which is proportional to the sum of the square of the weights, and this is the type of regularization used in our method. This regularization is important in neural network training, as it can help to prevent overfitting and improve the performance of the model. This means that by adding a regularization term to the loss function, the model is "encouraged" to find balance between fitting the training data well and keeping the complexity of the model under control. We have written about the theory on mini-batch sizing in detail in project 2 [5].

D. ADAM optimizer

Adaptive Moment Estimator (ADAM) is an optimization algorithm that is used in training neural networks. In our case, this optimizer is used to train our feed forward neural network. The principle of ADAM is to combine the ideas of stochasctic gradient descent (SGD) and momentum optimization by using moving averages of the parameters to provide a running estimate of the second raw moments of the gradients. This means that the algorithm "adapts" the learning rates of each parameter based on the historical gradient information.

E. Decision trees

Decision trees are supervised learning algorithms used in classification and regression tasks. The method is typically divided into a root node, the interior nodes, and the final leaf nodes, which are all connected by what we refer to as branches. The goal is to create a model which can predict the value of a target variable by learning simple decision rules which are inferred from the data features. The most informative descriptive features are those which reproduce the best output features. The process of finding this feature is continued until we find a stopping criteria where we end up in the leaf nodes.

There are several advantages to using decision trees. The visualization of a tree makes the method easy to understand and interpret. In addition, tree models can handle both continuous and categorical data, can model nonlinear relationships, and can model interactions between the different descriptive features [6]. Unfortunately, trees often have a lower level of predictive accuracy than some other approaches. The method is also prone to overfit data. However, by aggregating decision trees with methods such as bagging, random forests, and boosting, the predictive performance of trees can be improved.

F. The CART algorithm

There are several different algorithms that generate trees differently. In this report we will discuss the CART algorithm used in the `scikit-learn` decision tree python modules [6].

The Classification and Regression Tree (CART) algorithm grows decision trees with structures like we can see in Figure 1, where the root and internal nodes split to a maximum of two new nodes. Proceeding downwards leads to a leaf node and a prediction of class 1 or 2.

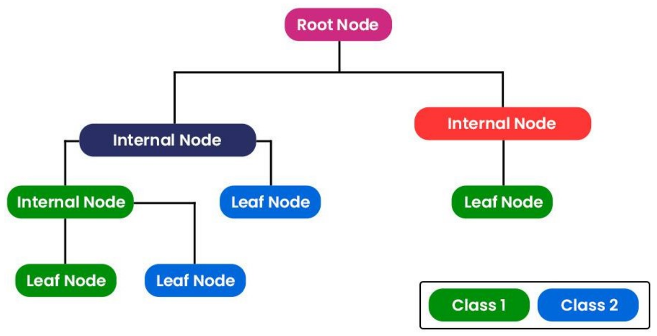

Figure 1. Hypothetical CART binary classification decision tree structure with color coded outcomes in the leaf nodes [7].

The core of the algorithm is deciding how the branches should be split depending on the attributes of the input data. For a single tree, the algorithm will choose some attributes and attempt to divide the distribution of classes in the data into different regions. To get 'pure nodes', these regions should ideally exclusively contain data relating to either class 1 or 2 but a more realistic case is that it manages to define regions with a clear majority.

Thus, when the tree classifies a specific data set, each node in the tree basically evaluates which region in the same distribution space (focusing on the same attributes) the instance should fall under, and then conclude the classification following the majority vote in the region.

The algorithm also seeks to minimize error in the leaf nodes by minimizing variance in the defined region, and may define a new region (and thus a new split in the tree) if a certain criteria is met. For this project the 'entropy' criterion $s \in [0,\ 1]$ is relevant, which evaluates the disorder within a specific region and is defined as:

$$s = -\sum_{k=1}^{K} p_{mk} \log p_{mk}, \tag{2}$$

where K is the amount of classes and p_{mk} is the amount of outcomes of the class k in a given region.

For a binary case we use $K = 2$ and the range becomes $s \in [0,\ 1]$ where the minimum means we can split into a pure node and the maximum means that the probability of class 1 and class 2 is the exact same (maximum disorder). The growing of the tree will persist until the entropy is sufficiently low.

G. Random forest algorithm

Plain decision trees suffer from high variance, where the data is prone to overfitting. Bootstrap aggregation (bagging), is a procedure for reducing the variance of a statistical learning method, and typically results in improved accuracy over prediction using a single tree. However, it can be challenging to interpret the resulting model. Random forests improve bagging by implementing a small tweak that decorrelates the trees.

We build a number of decision trees on bootstrapped training samples. Each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. This split only uses one of the m predictors and the algorithm will not consider a majority of the available predictors [6]. At each split, a fresh sample of m predictors is taken, typically

$$m \approx \sqrt{p}. \tag{3}$$

This prevents the bagged trees from all being similar to one another by not favoring one strong predictor which is always used in the top split, overshadowing the other moderately strong predictors.

In order to build the algorithm, we assume that we grow a forest of B trees. For $b = 1 : B$ we then draw a bootstrap sample from the training data organized in the $\mathbf{X}$ matrix and grow a random forest tree T_b based on the bootstrapped data by repeating three steps:

1. Select $m \leq p$ variables at random from the p predictors
2. Pick the optimal split among the m features using an algorithm (for example the CART algorithm), and create a new node.
3. Split the node into daughter nodes.

The steps are repeated until the maximum node size is obtained. Finally, we can output the ensemble of trees ${T_b}_1^B$ and make predictions for the problem.

H. Boosting method

The boosting method combines weak classifiers by applying the weak classifier to modify the data, in order to modify the data and create a better classifier. This is done for several iterations, for which we emphasize the observations which are misclassified by weighting them with a factor.

The principle is very similar to standard random forests except that after iterating, the trees that showed the fewest misclassifications will weigh more significantly on the classification voting. The process of updating these tree-specific weights over a number of iterations corresponds to a Gradient Descent algorithm which is a topic we have discussed in detail in report 2 [5].

III. METHOD

A. Data Set Description

For our analysis we are using the 1988 Heart disease data set from the Cleveland Clinic Foundation containing data on 297 patients including their diagnosis (1 = disease, 0 = no disease). The set includes 137 positive diagnoses, 160 negative [8].

The initial study contained 76 measured attributes, but all published experiments referring to the data only make use of 14 attributes, including the diagnosis. The remaining are documented as:

1. sex - (1 = male, 0 = female)
2. age - patient age in years
3. cp - chest pain type (1-4)
4. trestbps - resting blood pressure (in mm Hg on admission to hospital)
5. fbs - fasting blood sugar < 120 mg/dl (1 = true, 0 = false)
6. chol - cholesterol measure in mg/dl
7. restecg - ecg observation at resting condition (0-2)
8. thalach - maximum heart rate achieved
9. exang - exercise induced angina (1 = True, 0 = False)
10. oldpeak - ST depression induced by exercise relative to rest
11. slope - slope of the peak exercise ST segment
12. ca - number of major vessels (0-3) colored by flouroscopy
13. thal - 3 = normal, 6 = fixed defect, 7 = reversable defect

In our analysis we use this selection. The UCI archive contains further attribute documentation [8]. Some features (like blood pressure levels) are numerical, and some are categorical. However, we are going to use a modified version of the Cleveland data set (’*heart_cleveland_upload.csv*’ [9]) that has already relocated each feature category to their numerical values (0 and 1 instead of False and True and so on).

The scatter plots for some of the numerical features in figure 2 gives an impression of the data variety and distribution between positive and negative diagnoses.

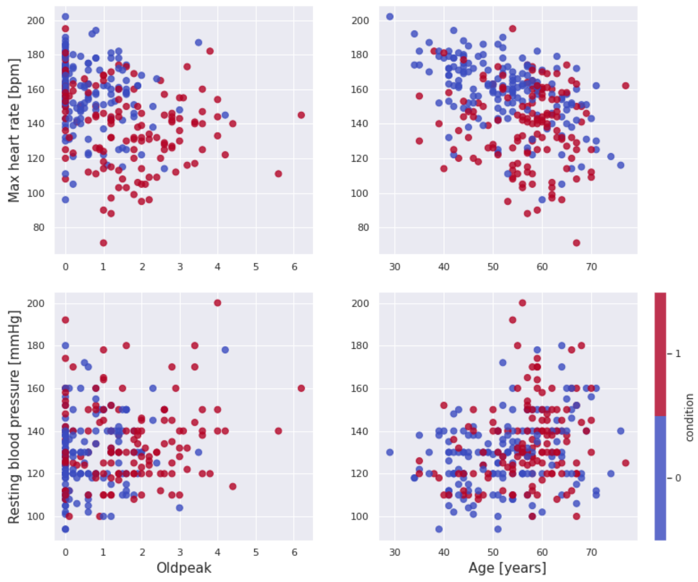

Figure 2. Scatter plots of the attributes Age, Oldpeak, Trestbps (Resting blood pressure) and Thalach (Max heart rate), with condition (1 = positive or 0 = negative diagnosis) color-coded for the corresponding patient.

B. Data preparation

Before proceeding with the analysis we need to have a proper look at our data, and potentially smooth out any errors if needed.

We will calculate a correlation matrix using the Pearson method, as defined in theory section II A and implemented with the pandas python module `dataframe.corr`, and consider extracting features with little correlation to the diagnosis from the data set. Other correlations will also be discussed briefly to get further insight into the data set, as well as verify that the correlations seem qualitatively correct taking into account external literature on cardiology as a way to prove that the data has been imported and handled correctly.

The prepared data set will be split into a training set (80%) that is used to train our models and a test set (20%) which will be a target for the models to evaluate their accuracy with.

In this data set we have a matrix that represent features and its values, and a target array which tells us if the patient has a heart disease or not. For the neural network analysis we use one-hot encoding to convert these target values from single integers into binary representation. This means that statement 1 would become (0,1), and 0 would be (1,0). This one-hot encoding ensures that machine learning does not assume that higher numbers are more important.

C. Neural network analysis

Feed Forward Neural Network (FFNN) will be the first machine learning method that we will apply to the data-set. We will start off by writing our own FFNN algorithm, and then compare the results by using TensorFlow python module.

When defining the neural network architecture, we define a network that consist of an input layer, one or more hidden layers, and an output layer. The input layer contains a number of neurons equal to the number of the features. In our case, we have 13 features. The hidden layer contains a number of neurons determined based on the complexity of the data. However, we will be using 90 neurons and one hidden layer as a starting point. We will then try different values and see how our neural network performs. Since this is a classification problem, meaning that we either have a true or false statement for a patient having a heart disease, our output layer contains a single neuron. This neuron will have value of 0 indicating the absence of heart disease and a value of 1 indicating the presence of heart disease.

When writing our own script for FFNN, we also have to define some hyper-parameters. These hyper-parameters are

- Regularization parameter λ
- The size of mini-batch
- Learning rate.

See theory section II C for a description of each of these hyper-parameters. As a starting point, we will be using batches with size of 50 random data-points. As for the regularization parameter λ and learning rate, we will perform a grid search. To do this, we define two lists containing values from -5 to 7 with a step of 1 on a logarithmic scale. We then make two for loops that runs over all the values from these lists and performs our FFNN and calculates the accuracy score for each of these values. This way, we can map out the values of learning rate and λ that fit the best with our data.

The activation function of the neurons in the hidden layer is determined based on the type of data and desired behavior of the network. Since this is a binary classification task, we are using the Sigmoid [5] activation function. These activation's are then used in the output layer, where we perform Softmax function [5] to determine the probability for the patient to have a heart disease. Since this is a classification problem, we use this probability and return a statement 1 if the probability is higher than 0.5, and 0 if it is lower than 0.5.

However, in order to train FFNN, we need an optimizer to find the best values for weights for each neuron and bias for the hidden layer. In order to do this, we use Stochastic gradient descent with momentum (See [5] theory section B) in backpropagation. This means that we calccu-late the error of output, and accordingly to the error, we backpropagate and change the values of weights and biases.

We wish to compare the results with TensorFlow built in library. Therefore, we first import the library, and define input and target data and feed it to the library. Once again we use one-hot decoding for the targets.

We also wish to experiment with different architectures by trying out different number of hidden layers, and different number of neurons for each layer. Additionally, we wish to test out different activation functions such as Sigmoid, ReLu, Tanh and softmax function. We will also be testing different optimizers, such as Stochastic gradient descent (SGD), Adam, RMSprop and Adagrad.

D. Random forest analysis

For the second model we will develop a random forest classifier using adaptive boosting with scikit-learn python modules, and using our prepared data set with no exctracted features. The models will use the 'entropy' criterion when splitting trees as established section II F.

The optimal learning rate η and max depth of the aggregated decision trees will be determined with hyperparameter analysis by calculating the accuracy from models of various configurations and presenting these accuracies in a heatmap.

The highest scoring configuration will be used to generate a confusion matrix to evaluate the final model's true and false positive/negative test prediction accuracy after being trained by the training data that we have prepared. The confusion matrix simply compares how many positive and negative diagnoses that our model gives correctly, and normalizes to give accuracy values between 0 and 1.

The total accuracy score will be calculated, and we will use the same model with the data set where we extracted the 'fbs' attribute (the one with 0.00 correlation with the patient condition) to see if there is a significant change in accuracy.

IV. RESULTS & DISCUSSION

A. Results of initial data set analysis

Figure 3 shows the correlation matrix for the features in the heart disease data set. The diagonal elements were removed because they would score a perfect 1 and de-saturate the color scale for the other positive correlations. The upper triangle was removed because it is symmetric with the lower and therefore redundant.

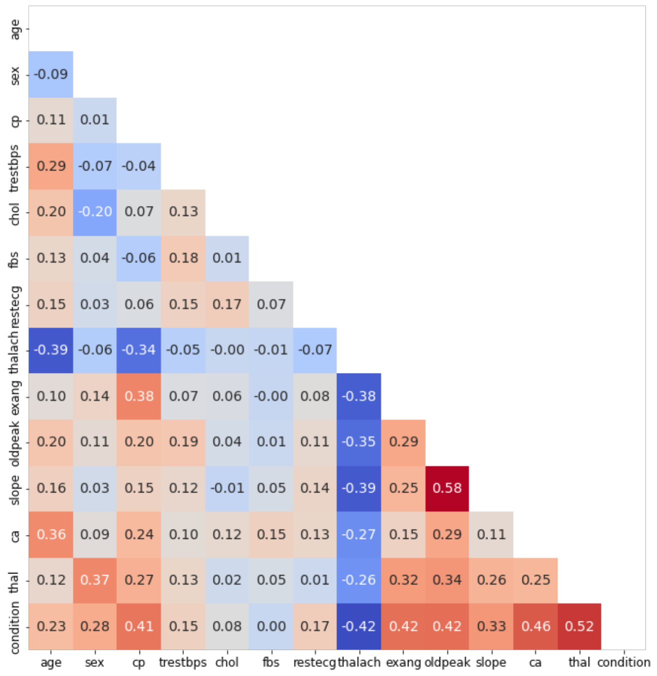

Figure 3. Correlation matrix for the relevant features in the UCI heart disease data set. The blue elements display negative correlation and the red positive correlation.

Some correlations of note are that age has a positive correlation with chol (0.20) and condition (0.23), implying that the older patients in the survey had higher cholesterol levels and were more likely to be diagnosed with heart disease. Age also has a significant negative correlation with thalach (-0.39), implying that maximum heart rate decreases with patient age. These observations agree qualitatively with literature on cardiology and aging [10] so we assume that we have imported and handled the data set correctly.

When constructing our machine learning models we are most interested in the bottom row of the heatmap which shows us the correlation between the various features and the condition, and we do see a fair amount of significant positive/negative correlations over 0.4, meaning they are important for predicting the diagnosis.

The attribute for fasting blood sugar 'fbs' scores 0.00 correlation even though it has a stronger correlation to other features. The data set isn't huge so deciding whether or not to extract this feature should not affect the model performance a lot, but we will make separate processed data sets that include all features and one that excludes fbs to see if it affects the prediction accuracy.

B. FFNN results

By writing our own Feed Forward Neural network, and doing a grid search for the most optimal learning rate and regularization factor λ value, we find that the most optimal learning rate has value of 0.00001, and λ has value 0.01, which gives an accuracy of 56%. Table I shows the accuracy rate for other values that we have picked.

Table I. Grid search for optimal λ and learning rate with the corresponding accuracy rate for FFNN analysis

Learning rate	λ	Acc. rate
0.00001	0.01	0.56
0.0001	0.01	0.53
0.01	0.01	0.53
1	1	0.53
0.00001	0.001	0.38

By having 56% as accuracy rate from our FFNN, we can say that our neural network performs poorly. We have included only some of the values in table I, and we can see that most of them have accuracy rate of 53%. It seems that our model guesses the output, and gets it wrong and right half of the time. This is not a good model. Additionally, since we are using stochastic gradient descent with momentum as our optimizer, we expect the learning to be a larger number than 0.00001. This is not the case here.

By running the grid-search multiple times, we see that the accuracy rates for the same values of hyper-parameters differ. This might be because our model is poor and it simply guesses the output randomly every time. Meaning that in the training part where we use backpropagation to find the optimal weights and biases, we find different values for weights and biases every time.

We now wish to compare the results by using the Tensorflow library. We have used the same architecture, and we choose to find the learning rate and the regularization factor which gave us the best results in table I.

By using tensorflow, we get **the accuracy rate up to 81%**. However, by increasing the number of epochs, we were able to achieve higher accuracy rates. For example, when we ran the FFNN with tensorflow using 2000 epochs, we achieved an **accuracy rate of 85%**. However, this result was not consistent across all runs. In some cases, the accuracy rate dropped back to 81%, even with 2000 epochs. This inconsistency may be due to the random mini-batch selection used in the optimization of weights and biases. However, we observed that in most cases, we achieved accuracy to be around 85% with higher number of epochs.

We wanted to optimize the architecture of our tensorflow (FFNN) by creating three lists of varying numbers of hidden layers, the number of neurons in each layer, and different sizes of mini-batches. We ran these values and found the optimal architecture. The results can be found in the file 'analyse.txt' in our github repository.

However, we found that the high number of hidden layers and neurons does not improve FFNN. From one run, we saw that by having **1 hidden layer**, **10 neurons** on the hidden layer, and **batch size of 70** gave us the highest accuracy rate of up to **91%**. This is higher than the 86.8% achieved with deep learning in the article we referenced in the introduction.

Additionally, we have tested different activation functions and optimizers here. We concluded that the most effective optimizer is Adam-optimizer. See theory section II D. As for the hidden layer activation function, we found that RELU-function is most effective. And for the output, softmax function.

However, please note that the accuracy rate by using the same architecture and the same hyper-parameters is not the same every time we run the network. Most often we achieve an accuracy rate of 88% up to 90%.

C. Boosted random forest classification results

The result from the hyperparameter analysis using the scikit-learn random forest classifier with adaptive boosting and 500 estimators, with varying learning rates η and max depths is presented in a heatmap in Figure 4.

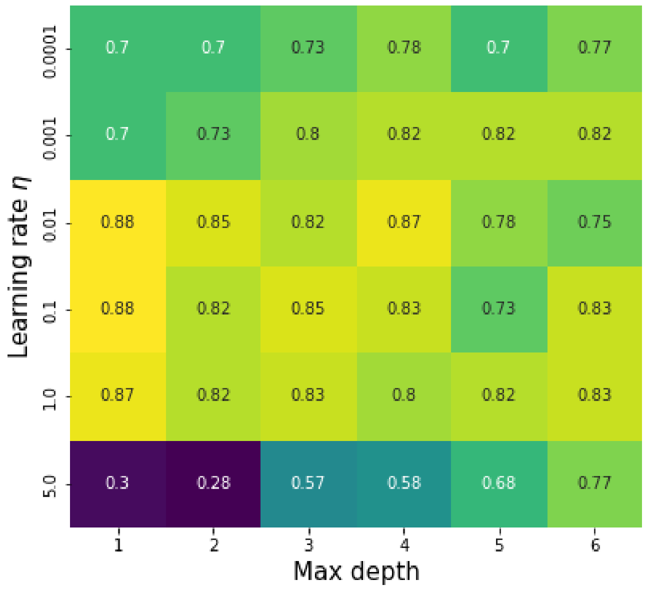

Figure 4. Prediction test accuracies from adaptive boosted random forest classification models using various hyperparameter configurations presented in a heatmap.

We observe from the grid search that the best performing models appear to have a max depth of 1, and learning rates 0.1 and 0.01 and test accuracies of 0.88. This is the same result that was achieved with Random Forest classification in the article noted in the introduction [3]. When we run the model with the fbs attribute extracted, the accuracy decreases to 0.77.

This implies that *all* significant correlations are important in prediction models even though the attribute doesn't directly correlate with the condition. The fbs attribute (0.00 correlation with condition) has positive correlation with the attribute 'ca' which has a strong correlation with condition, and their connected relation appears to be important when constructing a prediction model of this data set as the accuracy drops when fbs is removed.

Going back to our model trained with all the features, we generate a confusion matrix as seen in Figure 5. We observe that our model has a true positive accuracy of 0.89 and a true negative accuracy of 0.88. This means that a given positive prediction from our model has an 89% chance of being correct, and a negative prediction 88%.

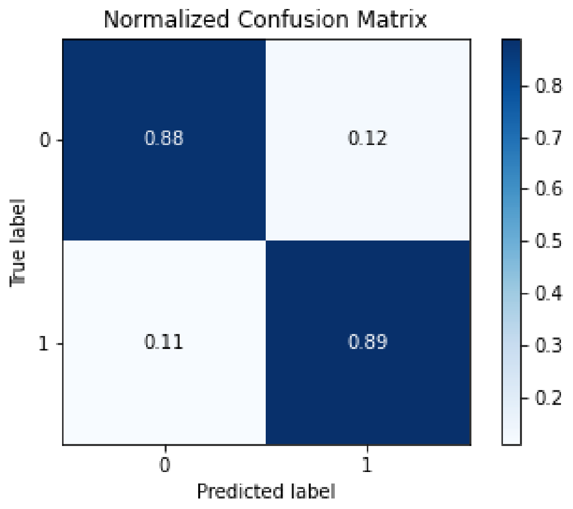

Figure 5. Confusion matrix presenting distribution of true positive/negative and false positive/negative predictions of our final Random Forest classifier model.

When performing any practical medical diagnosis, several tests may need to be performed to be confident in the result. However, it is more important that the confidence in a positive test is higher than vice versa. Take for example a covid test; a positive test has a big consequence on a person's daily routine so you want the test method to have a high true positive accuracy to avoid unnecessary diagnoses.

This is why the CDC (Centers for Disease Control and Prevention) in the US suggests use of covid antibody tests with specificity (true positive rate) of $\geq$ 99.5% because of the large societal consequences of false positive diagnoses [11].

The difference between the true positive/negative accuracy in our case is not large, but the clinical perspective is interesting either way.

V. CONCLUSION

Two machine learning models were developed in the study: a classification feedforward neural network with back-propagation, and a random forest classifier with adaptive boosting. The neural network model achieved an accuracy up to 91% on the test set, while the random forest model achieved an accuracy of 88% on the test set. The random forest model also had a true positive prediction accuracy of 89%.

Even though we weren't sure what all of the attributions in the feature list represented, such as oldpeak and slope, we were able to make them useful for the model. By including them, the model appeared to improve. That is, we do not need a deep understanding of all of the features to build a decent model.

Another point of potential improvement is that we could work with the original, unprocessed data. The set used in our analysis seems to have been properly pre-processed, and little work was done on it ourselves. Looking at the entire data set we could evaluate for ourselves which attributes are interesting or not, 'smoking' (True/-False) is an example of a missing attribute that could prove useful. More knowledge of cardiology could help us determine relevant attributes more efficiently.

We have demonstrated that the true positive accuracy in our random forest model is higher than the true negative accuracy, which is more favourable in medical diagnosis than the opposite case.

Overall, the results of the study suggest that both the neural network and random forest models were able to achieve high levels of accuracy in predicting heart disease diagnosis using the 1988 Cleveland Clinic Foundation survey data. These results are similar to those reported in other studies that have analyzed this data set.

[1] W. H. Organization, "Cardiovascular diseases," https://www.who.int/health-topics/cardiovascular-diseases/#tab=tab_1 (2020), accessed 02.12.22.

[2] A. H. Association, "Classes of heart failure," https://www.heart.org/en/health-topics/heart-failure/what-is-heart-failure/classes-of-heart-failure, accessed 02.12.22.

[3] R. Bharti *et al.*, "Prediction of heart disease using a combination of machine learning and deep learning," (2021), accessed 23.11.22.

[4] "Pearson correlation coefficient," https://en.wikipedia.org/wiki/Pearson_correlation_coefficient, accessed 08.12.22.

[5] S. M. S. Lasse Totland, Domantas Sakalys, "Developing neural networks for regression and classification problems," (2022).

[6] M. Hjorth-Jensen, "Applied data analysis and machine learning," https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html, accessed 02.12.22.

[7] T. Chandraveni, "Cart (classification and regression tree) in machine learning," https://www.geeksforgeeks.org/cart-classification-and-regression-tree-in-machine-learnin (2022), accessed 18.12.22.

[8] D. R. Detrano, "Cleveland heart disease data set," https://archive.ics.uci.edu/ml/datasets/Heart+Disease (1988), accessed 20.11.22.

[9] Cherngs, "Heart disease cleveland uci," https://www.kaggle.com/datasets/cherngs/heart-disease-cleveland-uci (2019), accessed 21.11.22.

[10] "Heart health and aging," https://www.nia.nih.gov/health/heart-health-and-aging (2018), accessed 08.12.22.

[11] "Why specificity matters in covid-19 antibody testing," https://www.siemens-healthineers.com/laboratory-diagnostics/assays-by-diseases-conditions/infectious-disease-assays/specificity-matters (2020), accessed 18.12.22.