Heart Disease Classification with Multilayer Perceptron

Benjamin Pittelkau
Virginia Tech
Blacksburg, VA
bpittelkau@vt.edu

Abstract

Cardiovascular disease (CVD) is a fatal and costly issue worldwide. Methods to predict and prevent CVD can save lives and mitigate serious complications. CVD prevention for individuals that live in an area with poor healthcare is critical, especially since CVD may be misdiagnosed or those patients may not receive the proper treatment. Machine learning can be used to analyze an individual's lifestyle and medical data to determine whether they have or do not have CVD with high accuracy. This study aimed to implement the supervised machine learning algorithm multilayer perceptron (MLP) to classify whether an individual has CVD, and the performance is compared to an MLP from a Python package, logistic regression, gaussian naïve bayes, support vector machine, decision tree, and random forest. The MLP developed in this study had 94.44% accuracy, which was the highest accuracy after random forest with an accuracy of 95.32%. The MLP designed here has high accuracy and could be implemented in, for example, a mobile application that anyone can easily use by entering data or downloading their medical data.

1. Introduction

Heart disease is a global issue affecting about 18.2 million adults in the US alone and has cost $363.4 billion between 2016 to 2017 due to medical bills, loss of productivity, and mortality [1]. Heart disease includes cardiovascular diseases (CVDs) such as blood vessel disease, arrythmias, congenital heart defects, disease of the myocardium, and heart valve disease [2]. Many of these CVDs are indicated by certain health metrics that can be confounded with other non-CVD diseases and there are numerous health metrics to consider, so reliable prediction of CVD is challenging. However, reliable prediction of CVD can give foresight to prevent an individual from developing CVD and can provide information for personalized treatment plans. The problem that must be solved is the implementation of a machine learning algorithm (MLA) that can accurately predict CVD. In this paper, the use of a multilayered perceptron (MLP) was investigated as a classification algorithm to predict whether an individual has CVD. Additionally, the performance of the MLP was compared to the performance of various other classification algorithms.

2. Background Study

Many machine learning models have been tested to predict CVD, such as logistic regression (LR), naïve bayes (NB), multilayer perceptron (MLP), support vector machine (SVM), decision tree (DT), random forest (RF), and XGBoost [3]. Accuracy close to 100% is required for medical applications since CVD is a life-threatening disease. The limit of current models is the lack of high accuracy required for CVD classification. Jyoti Soni et al. used a genetic algorithm with a DT that showed a 99.2% accuracy [4]. In 2018, Jan et al. showed that RF tends to have very high accuracy (98.14%) while classification with regression analysis provides poor accuracy [5]. Md Mamun Ali et al., in 2021, showed that K-nearest neighbor (KNN), DT, and RF provide 100% accuracy and 1.0 sensitivity [6].

Table 1. Heart disease dataset attribute details.

#	Attribute name	Description
1	age	Age in years
2	sex	Female = 0, Male = 1
3	cp	Chest pain type (values 0 to 4)
4	trestbps	Resting blood pressure (mmHg on hospital admission)
5	chol	Serum cholestoral (mg/dl)
6	fbs	Fasting blood sugar >120 mg/dl (1 = true, 0 = false)
7	restecg	Resting electrocardiographic (ECG) results (0, 1, or 2)
8	thalach	Maximum heart rate achieved (bpm)
9	exang	Exercise induced angina (1 = yes, 0 = no)
10	oldpeak	ST depression induced by exercise relative to rest (related to ECG signal)
11	slope	Slope of the peak exercise ST segment
12	ca	Number of major vessels colored by fluoroscopy (0-3)
13	thal	1 = normal, 2 = fixed defect, 3 = reversible defect
14	Target (class)	-1 = no CVD, 1 = CVD

A small number of studies analyze the performance of neural network classifiers, so it is of great interest to investigate how well a MLP would perform.

3. Approach

3.1. Data Preprocessing

In this study, a heart disease dataset from Kaggle was used. This dataset contains 14 attributes, which are shown in Table 1. The dataset contains 1025 structured patient records since 1988 from Cleveland, Hungary, Switzerland, and Long Beach V. The performance of machine learning algorithms depends on how the dataset is preprocessed. First, the data entries are randomly shuffled, and half of the dataset is randomly chosen as the test dataset. Next, outliers are removed using an Isolation Forest algorithm from the Scikit-Learn (sklearn) package. The Isolation Forest (IF) algorithm is an anomaly detection technique that identifies anomalies by isolating data points in a dataset, as shown in Supplementary Figure 1. It is based on the premise that anomalies are few and different, making them easier to isolate than normal observations. The result of the IF is a list of outliers and inliers, and data marked as an outlier is removed. After IF, the dataset was reduced to 685 examples. Then, the data is standardized for each feature in the training dataset such that the mean is zero and the variance is one. This is critical for MLP, and other MLAs, since MLP is sensitive to feature scaling [7]. The fitting parameters for the training dataset is applied to the test dataset so that the MLP results are meaningful. The distribution of the training and test data after outlier removal and standardization is shown in Supplementary Figures 2-5.

3.2. Multilayered Perceptron Overview

The MLP is a supervised MLA and a type of artificial neural network that consists of multiple layers of interconnected neurons called nodes. Each node receives input data, applies an activation function on the input data, computes a dot product between learned weights and the result from the activation function, then outputs the dot product result to the next node (Figure 1a). This process is repeated until the output is reached. More hidden layers can be added to provide a deeper network. The basic architecture for the MLP is shown in Figure 1b, where there are i features, j hidden nodes in 1st layer, n hidden layers, m hidden nodes in $(n-1)^{th}$ layer, and p hidden nodes in n^{th} layer. The output of the n^{th} layer is a single node z_y, which is the classification result of the MLP. The classification output is between -1.0 and 1.0, with a decision line (threshold) at zero. An output $z_y < 0$ indicates that the individual does not have CVD, and $z_y > 0$ indicates that an individual does have CVD.

MLP was specifically chosen since (1) MLP is a relatively simple non-linear function approximator that can effectively classify small datasets, and (2) multiple hidden nodes within the architecture allow for classification of data that is linearly non-separable. There is nothing particularly new about the implementation of this MLP.

The MLP was implemented in Python using the NumPy package. The parameters that must be learned in an MLP are the weights for each connection between each node. There are two key steps to learn the weights in an MLP: forward and backward propagation. At first, in forward propagation, all the weights are randomly initialized and feature i enters the input node X_i where the activation functions and dot products are calculated. All nodes use a hyperbolic tangent function as the activation function. Once the inputs are propagated through the MLP to output node z_y, a loss function (squared error) is calculated between the estimated result z_y and the ground-truth labeled data y. Then, backpropagation is performed by stepping back by one layer, then computing the error between each node and the previous layer's node, then updating the weights for that node using gradient descent (GD), as shown in Figure 1c.

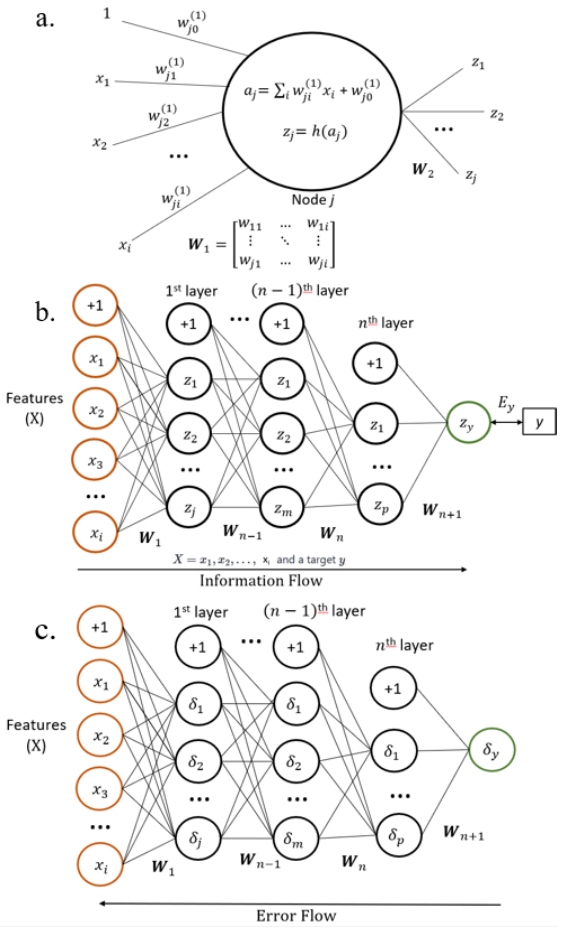

Figure 1: MLP concept overview. a. Unit neuron within the MLP. b. Forward propagation. c. Backward propagation.

3.3. MLP Mathematics

Let the matrix of weights at the 1st layer be,

$$\boldsymbol{W_1} = \begin{bmatrix} w_{11} & \dots & w_{1j} \\ \vdots & \ddots & \vdots \\ w_{i1} & \dots & w_{ij} \end{bmatrix} \quad (1)$$

The activation is calculated as the dot product between the weights and the inputs,

$$a_j = \sum_i w_{ij}^{(1)} x_i \quad (2)$$

Then the output of node z_j is,

$$z_j = h_1(a_j) = \tanh\left(\sum_i w_{ij}^{(1)} x_i\right) \quad (3)$$

Suppose $\boldsymbol{X}$ is a matrix of size $r \times c_0$, where r is the number of examples from the dataset and c_0 is the number of features for each example. Here, r varies depending on how many examples are chosen as training or test data, and c_0 is 13 (13 features in the dataset). Then, (3) can be rewritten in matrix form as,

$$\boldsymbol{Z_1} = \tanh\left(\boldsymbol{W_1^T X^T}\right) \quad (4)$$

where the hyperbolic tangent activation function is applied to each element in the matrix multiply $\boldsymbol{W_1^T X^T}$, and $\boldsymbol{T}$ denotes transpose. The matrix $\boldsymbol{Z_1}$ at layer 1 is then of size $r \times c_1$, where c_1 is the number of nodes in layer 1 (i.e., $c_1 = j$). This matrix multiply can be done for each layer in the MLP until the output node z_y is reached. The last layer contains weights,

$$\boldsymbol{W}_{n+1} = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix} \quad (5)$$

And the output node is calculated as,

$$\vec{z}_y = \tanh(\boldsymbol{W}_{n+1}^T \boldsymbol{Z}_n^T) \quad (6)$$

The output vector $\vec{z}_y$ is of size r, which contains the estimated classification for each example.

Next, back propagation is performed. The partial derivative of the error with respect to the weights is,

$$\delta_{il} = \frac{\partial E}{\partial w_{il}} = \frac{\partial E}{\partial a_l}\frac{\partial a_l}{\partial w_{il}} \quad (7)$$

By using chain rule and using a square error loss function,

$$\delta_l = \frac{\partial E}{\partial a_l} = \begin{cases} h'(a_p)(z_y - y), & \text{at output node} \\ h'(a_l)(\sum_l w_{lk}\delta_k), & l \in \text{hidden nodes} \end{cases} \quad (8)$$

where δ_l is the partial gradient at one of any hidden nodes. Also, using equation (2),

$$\frac{\partial a_l}{\partial w_{il}} = z_i \quad (9)$$

Therefore, the gradient is equivalent to,

$$\delta_{il} = \frac{\partial E}{\partial w_{il}} = z_i \delta_l \quad (10)$$

Since a hyperbolic tangent activation function is used,

$$h'(a_l) = 1 - \tanh(a_l)^2 \quad (11)$$

Finally, the weights are updated by the following update rule,

$$w_{il} \leftarrow w_{il} - \eta \frac{\partial E}{\partial w_{il}} \quad (12)$$

where η is the learning rate. Gradient descent is used since the squared error must be minimized. At the output, equation (8) can be rewritten as,

$$\delta_y = (\vec{z}_y - \vec{y})(1 - \vec{z}_y^2) \quad (13)$$

The gradient for the weights between the n^{th} layer and the output is then,

$$\boldsymbol{\nabla}_{n+1} = \boldsymbol{Z}_n^T \delta_y \quad (14)$$

Thus, the weights are updated by computing,

$$\boldsymbol{W}_{n+1} \leftarrow \boldsymbol{W}_{n+1} - \eta \boldsymbol{\nabla}_{n+1} \quad (15)$$

For any layer other than the output, such as the n^{th} layer, the gradient is computed as follows,

$$\begin{aligned} \boldsymbol{\nabla}_n &= \boldsymbol{Z}_{n-1}^T \boldsymbol{\nabla}_{n+1} \\ \boldsymbol{W}_n &\leftarrow \boldsymbol{W}_n - \eta \boldsymbol{\nabla}_n \end{aligned} \quad (16)$$

This process is repeated until the input nodes are reached, at which point forward propagation with the updated weights occurs. This cycling of forward and backward propagation repeats ε times, where each iteration is called an *epoch*. Thus, ε is a hyperparameter that the user must set. The other hyperparameters that must be set are the learning rate η, hidden layer size, and the number of nodes per layer. In Python, the MLP size is represented as a tuple that has *n* elements, each of which is a number to indicate the number of nodes at each layer. An overview of the implemented MLP code structure is shown in Figure 2.

3.4. MLP Performance Comparison Overview

The performance of the MLP created here was compared to the performance of logistic regression (LR), gaussian naïve bayes (GNB), support vector machine (SVM), decision tree (DT), and random forest (RF) algorithms from the sklearn package. Additionally, the performance of the MLP from the sklearn package was compared.

The MLP implemented in this paper has $n = 3$ hidden layers, each of which contains 10 nodes (represented as (10,10,10) in Python), and the learning rate was set to $\eta = \frac{1}{\#\text{ of training examples}}$ and $\varepsilon = 2000$ iterations, which was found to provide the best MLP performance.

The MLP from sklearn also has $n = 3$ hidden layers with 10 nodes in each layer. The activation function is also hyperbolic tangent and uses gradient descent (using all training examples in optimization, which is also how the MLP in this paper optimizes the weights). The learning rate is also set to $\eta = \frac{1}{\text{\# of training examples}}$ with $\varepsilon = 2000$ iterations.

LR uses a L2 penalty term and the "liblinear" optimizer with $\varepsilon = 2000$ iterations. GNB uses 10^{-9} variance smoothing with no priors. A C-support vector classifier (C-SVC), a type of SVM, is used with unity regularization, 10^{-3} tolerance for the stopping criterion, $\varepsilon = 2000$ iterations, a radial basis kernel, and a kernel coefficient of $\gamma = \frac{1}{N_{features} * var(X)}$, where $N_{features} = 13$ is the number of features and $var(X)$ is the variance of input data. The shrinking heuristic is also applied to the C-SVC. The DT classifier uses the Gini impurity criterion to measure split quality, splits on the best split, nodes expand until all leaves are pure, requires 2 samples to split an internal node, and requires one sample to be at a leaf node. Finally, the RF classifier uses 100 trees, the Gini impurity criterion, requires 2 samples to split an internal node, requires one sample to be at a leaf node, and the maximum number features to consider when looking for the blest split is $\sqrt{N_{features}} \approx 4$ features.

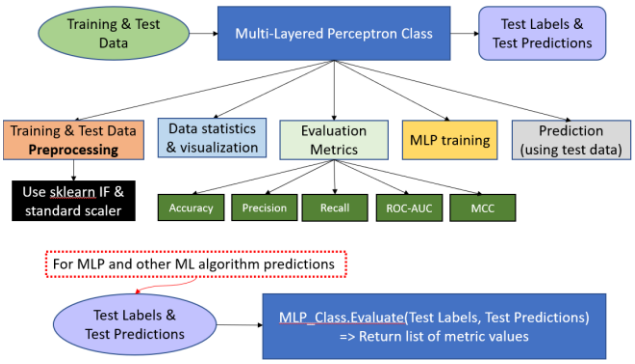

Figure 2: MLP code structure as implemented in Python.

3.5. Anticipated Problems

There were a couple anticipated problems. Of course, the two biggest problems were correctly preprocessing the dataset and implementing the MLP in Python using the preceding process. However, once the code was debugged, no problems occurred. One other anticipated problem was the time and resources needed to train the MLP. A small dataset and MLP structure were chosen to reduce computation time. The implementation of the MLP using mostly linear algebra in NumPy, rather than loops, also decreased computation time. All models were trained on a Windows computer that contains an Intel® Core™ i7-14700HX processor with 20 cores and 32 GB of memory, so resource constraints were not an issue. After verifying the correctness of the MLP program implemented in this paper, the MLP successfully classified the training and test data, and the performance is comparable to the MLP from Scikit-learn when given similar initialization parameters.

4. Experimental and Results

Five evaluation metrics were used to measure the performance of all MLAs: accuracy, precision, recall, F1-Score, receiver operating characteristic (ROC), area under the curve (AUC) of the ROC graph, and Matthew's correlation coefficient (MCC). Accuracy measures whether the model's predicted values are equivalent to the ground truth values in training and test data [9]. The corresponding equation for accuracy is,

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{17}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives. Precision is defined at the proportion of the model's correctly classified positive classifications that are actually positive [6].

$$\text{Precision} = \frac{TP}{TP+FP} \tag{18}$$

Recall, which is also known as the true positive rate (TPR), is the proportion of the model's positive classifications that are actually positive to all actual positives [6].

$$\text{Recall} = \frac{TP}{TP+FN} \tag{19}$$

The F1-Score combines precision and recall into one balanced metric [6].

$$\text{F1-Score} = \frac{2*Precision*Recall}{Precision+Recall} \tag{20}$$

The ROC is a graph of TPR vs. FPR and is a visual representation of model performance over all thresholds [10]. FPR is defined as,

$$\text{FPR} = \frac{FP}{FP+TN} \tag{21}$$

If given a randomly chosen positive and negative example, the AUC represents the probability that the model will rank the positive example higher than the negative example [10]. The AUC is found by numerically integrating the ROC for each model. Here, the numerical integration is done using the trapezoidal integration method. MCC measures the differences between actual values and predicted values, where a MCC near unity

Table 2. Performance metrics of the MLP in this paper with varying parameters.

Parameter	Accuracy	Recall	FPR	Precision	F1-Score	MCC
$n = 2$, 5 nodes, $\varepsilon = 100$	83.63%	84.44%	17.28%	84.44%	0.8444	0.6716
$n = 3$, 5 nodes, $\varepsilon = 100$	87.13%	86.11%	11.73%	89.08%	0.8757	0.7429
$n = 3$, 5 nodes, $\varepsilon = 1000$	84.50%	87.78%	19.14%	83.60%	0.8564	0.6893
$n = 3$, 10 nodes, $\varepsilon = 1000$	94.44%	96.67%	8.02%	93.05%	0.9482	0.8891
$n = 3$, 10 nodes, $\varepsilon = 2000$	94.44%	95.00%	6.17%	94.47%	0.9474	0.8886
$n = 3$, 20 nodes, $\varepsilon = 1000$	91.23%	93.33%	11.11%	90.03%	0.9180	0.8243

Table 3. Performance metrics of various MLAs.

Algorithm	Accuracy	Recall	FPR	Precision	F1-Score	MCC
MLP in this work	94.44%	95.00%	6.17%	94.47%	0.9474	0.8886
sklearn MLP	88.30%	88.88%	12.34%	88.88%	0.8888	0.7654
LR	85.09%	89.44%	19.75%	83.42%	0.8633	0.7018
GNB	82.75%	83.89%	18.52%	83.43%	0.8366	0.6539
SVM	90.06%	91.11%	11.11%	90.11%	0.9061	0.8005
DT	93.56%	91.67%	4.32%	95.93%	0.9375	0.8723
RF	95.32%	95.00%	4.32%	96.07%	0.9553	0.9063

signifies little difference between the actual and predicted values [6].

(22) $$\mathrm{MCC} = \frac{TP*TN-FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

At first, the first half of the dataset was chosen as training data, and the other half was chosen as testing data. For the MLP implemented in this paper, the learning rate was initially set to $\eta = \frac{1}{\#\text{ of training examples}}$ and $\varepsilon = 1000$ iterations. Then the number of layers and nodes were chosen based on how well the MLP performed. Initially, the MLP had $n = 2$ hidden layers with 5 nodes in each layer. Then, n was set to 3 but each layer still had 5 nodes. Next, ε was increased to 1000, which appeared to perform better. Then, the number of nodes in each layer was increased to 10, which further improved the performance. Subsequently, ε was increased to 2000, which showed slight improvement. Finally, the number of nodes per layer was increased to 20 with $\varepsilon = 1000$, which showed worse performance. The summary of these results is shown in Table 2. The corresponding ROC-AUC graphs are shown in Supplementary Figure 6. Thus, $n = 3$ with 10 nodes per layer and $\varepsilon = 2000$ was chosen as the best MLP and was used in the comparison with the other MLAs.

As shown in Table 3, the MLP implemented in this paper performed better compared to sklearn's MLP in all metrics and performed well in comparison to the other MLAs. RF had the best performance over all metrics, but the MLP here performed only about 1-2% worse and is the 2nd best performing MLA. The ROC-AUC graph for each model is shown in Figure 3. The ROC curves for both the MLP and RF models are above all other curves, which indicates higher performance than the other models. The ROC curve for the MLP developed here (orange) nearly matches the RF curve (grey). However, the AUC for the MLP is 0.972, whereas the AUC for RF is 0.832, which indicates that the probability of ranking a positive example higher than a negative example is greater for the MLP than the RF model.

Compared to the other MLAs, it appears that the MLP implemented here successfully classifies whether an individual has CVD based on their lifestyle and medical data, achieving about 94% accuracy. This accuracy is comparable with the SVM, DT, and RF accuracies and – along with the other metrics – is better than the accuracy from sklearn's MLP.

However, the MLP model suffers from overfitting since there is no regularization and the dataset is relatively small. Improvements could be made, such as further optimizing the MLP size or using adaptive optimization like adaptive moment estimation (ADAM). The user interface to the MLP class developed in this paper allows the user to set all hyperparameters and input datasets. This allows the MLP to be generalized for any single-class classification problems. Further development is needed for multi-class classification.

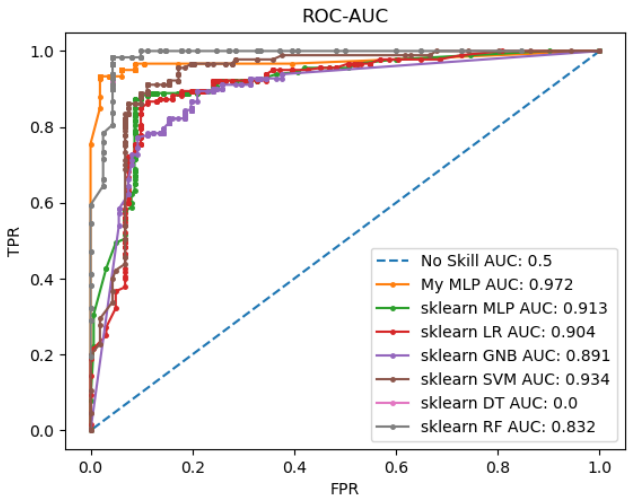

Figure 3: ROC-AUC for all considered models. The orange curve is the ROC for the MLP implemented in this study, with an AUC of 0.972.

5. Conclusion

In this paper, an MLP was implemented and tested in Python along with Scikit-Learn's MLP, LR, GNB, SVM, DT, and RF. The MLP developed here out-performed the MLP from Scikit-Learn and was the most accurate model after RF. This model likely overfits the data since no regularization is implemented and only a small dataset is used. Improvements could be made to the MLP to further generalize the functionality, but the current implementation can support any single-class classification problem. Additionally, future work could optimize the MLP structure to provide an accuracy above RF. Artificial neural networks are a powerful tool that can address a wide range of problems, and perhaps different neural network models could be considered, such as a deep neural network.

6. Availability

The code for the MLP developed in this paper is available on GitHub and is licensed with a GNU General Public License:

www.github.com/AstroCAT10/MLprojHDMLP

7. Reproducibility

There is a program within the GitHub repository that automatically downloads the dataset used in this paper. The dataset is hosted on Heart Disease Dataset (kaggle.com) and is freely available.

References

[1] M. Heid, "Heart Disease Statistics," Verywell Health, 22-Sep-2022. [Online]. Available: https://www.verywellhealth.com/heart-disease-statistics-5198489. [Accessed: 10-Oct-2024].

[2] "Heart Disease," Mayo Clinic, 13-Oct-2021. [Online]. Available: https://www.mayoclinic.org/diseases-conditions/heart-disease/symptoms-causes/syc-20353118. [Accessed: 10-Oct-2024].

[3] N. L. Fitriyani, M. Syafrudin, G. Alfian, and J. Rhee, "HDPM: An Effective Heart Disease Prediction Model for a Clinical Decision Support System," IEEE Access, vol. 8, pp. 133034–133050, 2020, doi: https://doi.org/10.1109/access.2020.3010511.

[4] J. Soni, U. Ansari, D. Sharma, S. Soni, Predictive data mining for medical diagnosis: an overview of heart disease prediction, Int. J. Comput. Appl. 17 (8) (2011) 43–48.

[5] M. Jan, A.A. Awan, M.S. Khalid, S. Nisar, Ensemble approach for developing a smart heart disease prediction system using classification algorithms, Res. Rep. Clin. Cardiol. 9 (2018) 33–45.

[6] M. M. Ali, B. K. Paul, K. Ahmed, F. M. Bui, J. M. W. Quinn, and M. A. Moni, "Heart disease prediction using supervised machine learning algorithms: Performance analysis and comparison," *Computers in Biology and Medicine*, vol. 136, p. 104672, 2021. DOI: 10.1016/j.compbiomed.2021.104672.

[7] "1.17. Neural network models (supervised)," *Scikit Learn*. [Online]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html#classification. [Accessed: Dec. 9, 2024].

[8] M. M. Ali, B. K. Paul, K. Ahmed, F. M. Bui, J. M. W. Quinn, and M. A. Moni, "Heart disease prediction using supervised machine learning algorithms: Performance analysis and comparison," Computers in Biology and Medicine, vol. 136, p. 104672, Sep. 2021, doi: https://doi.org/10.1016/j.compbiomed.2021.104672.

[9] "Classification: Accuracy, Precision, and Recall," Google Developers, [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall. [Accessed: 10-Oct-2024].

[10] "ROC and AUC," Google Developers, [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc. [Accessed: 10-Oct-2024].