B351 Final Report

Ronit Jha, Chinmay Deshpande, Yanin Charoenpornsawat, Kaushik Arulkumar

May 2025

1 Introduction

Heart disease remains one of the leading causes of morbidity and mortality worldwide, emphasizing the need for effective prediction and early detection methods. In recent years, machine learning algorithms have gained prominence as valuable tools for diagnosing and predicting various health conditions, including heart disease. The ability to accurately predict the likelihood of heart disease based on patient data can significantly aid in early intervention and personalized treatment plans, potentially saving lives.

This research paper aims to analyze the performance of four prominent machine learning algorithms: Decision Trees (DT), K-Nearest Neighbors (KNN), Artificial Neural Networks (ANN), and Hopfield Networks. These algorithms represent a diverse range of approaches in machine learning, each with its unique strengths and limitations. By comparing the effectiveness of these algorithms on a well-established heart disease dataset, this paper seeks to determine which method offers the most accurate and reliable prediction.

By evaluating the performance of these algorithms on the heart disease dataset, we aim to provide a comprehensive analysis of their strengths, weaknesses, and overall suitability for medical applications. Our results will highlight not only the prediction accuracy but also the computational efficiency and interpretability of each algorithm, contributing valuable insights to the field of healthcare informatics.

2 Historical Attempts

There have been historical attempts to solve this problem. Predicting Heart Disease Using Machine Learning, a study from 2023, is a good example of a past attempt, where they compared Logistic Regression, Decision Tree, Naive Bayes, K-Nearest Neighbors, and Support Vector Machine models together in various metrics. Their results showed that Support Vector Machine model was the superior model. Another attempt, from 2021, *Predicting Heart Disease Using Machine Learning*, also compared many different models. Their results showed that using an ANN with DT as the activation function for its layers achieved the best results.

3 Dataset Used

The dataset that we used is called Heart Disease Dataset on Kaggle. It contains 13 attributes, age, sex, chest pain type, resting blood pressure, serum cholestoral in mg/dl, fasting blood sugar > 120 mg/dl, resting electrocardiographic results (values 0,1,2), maximum heart rate achieved, exercise induced angina, oldpeak = ST depression induced by exercise relative to rest, the slope of the peak exercise ST segment, number of major vessels (0-3) colored by flourosopy, and thal. The dataset had 1025 data points. As the dataset was in good condition, we did not have to perform any preprocessing.

4 Imports

In the implementation of machine learning models for heart disease prediction, several Python libraries were used to streamline the workflow from data preprocessing to evaluation. Pandas was used to load and manipulate structured data from a CSV file, while NumPy supported numerical operations. Scikit-learn played a pivotal role in writing the code. The methods it includes were used to train and scale the data for each algorithm. Evaluation methods were used in order to evaluate the performance of each of the models. Finally, matplotlib and seaborn were used to make specific visualizations for each of the algorithms, ranging from heatmaps to decision trees.

5 Decision Trees

DT is a supervised training algorithm that is capable of classification and regression. DTs are highly interpretable as their structure can mirror the human decision making process. It operates by recursively splitting into subsets based on the data's attributes, aiming to create groups that are pure, all the samples belong to the same class, with respect to the target attribute. Each node on a DT represents a feature or test and each branch represents the outcome of that feature. The leaf nodes represent the final prediction.

When making a DT, the most important factor is selecting the correct attribute and threshold for the data split. The objective is to make each child node pure. There are many methods and criteria for splitting the data, however, the most common and highly effective method is the Gini Impurity. This method measures the likelihood of incorrectly classifying a randomly chosen element from the data set. It is mathematically defined as:

$$Gini = 1 - \sum p_i^2$$

P_i is the proportion of samples in class i. 0 represents pure or perfect purity and higher values indicate a more mixed class distribution. The aim is to minimize

the weighted Gini impurity after a split. Another method to split a DT is Entropy:

$$Entropy = \sum p_i log_2(p_i)$$

Both Gini and Entropy quantify the purity of a node, however, the Gini method can be computationally faster as it avoids logarithms. In practical scenarios, both yield similar results and DTs although slight differences in split choice can occur. For this reason, we decided to use Gini impurity for our method of data splitting. The time complexity of this DT model is $O(d \cdot m \cdot n^2)$ where n is the number of data samples, m is the number of attributes, and d is the depth of the DT.

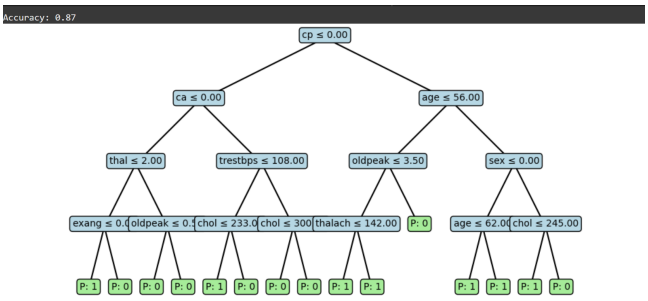

Figure 1: DT graph

To implement DT, a custom model was developed using only these three fundamental Python libraries: pandas, numPy, and matplotlib. The feature in each node of the tree was selected by minimizing the weighted Gini Impurity. This process was repeated recursively until a node was pure or until the max depth of 4 was reached. A max depth was chosen at 4 in order to prevent overfitting the tree to the data. Using these methods, the resulting tree was produced using matplotlib (see Figure 1). Overall, after being trained on the heart disease dataset, the achieved training accuracy was 87 percent, shown at the top of Figure 1. The tree shows that the most influential attributes are cp or chest pain, ca or number of major vessels covered in fluoroscopy, and age.

6 K-Nearest Neighbors

The KNN algorithm is a learning method primarily used for the classification of data. It is an instance-based algorithm, meaning that it makes no prior assumptions about the data until the query is made. The core idea of KNN is to classify a data point based on a majority vote of its neighbors, the data points that are

closest to it. The new data point is classified based on its nearest neighbors.

In order to run the algorithm, KNN first stores all the data, simply memorizing all the data points. Given a new point, KNN then computes the distance between this point and all the other points in the dataset. Given a value of k, it then computes the k nearest neighbors to the new point, retrieving the class of each neighbor. It then assigns the most frequent class among the k neighbors to the new point, thereby classifying the new data point.

Our algorithm works on this same basis and classifies the likelihood of heart disease given a new patient's symptoms and features. First, the algorithm was fed a dataset of over a thousand patients, both with and without heart disease. In this algorithm, we used the Python libraries NumPy, pandas, and scikit-learn to help with data manipulation and measuring accuracy. Then, the distance between two points was calculated using the Euclidean distance formula:

$$d = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

We can then predict whether the new patient has heart disease or not by comparing it to the other patients' data. Initially, we had set our algorithm to have $k = 3$, which reduced the accuracy. To optimize the KNN method, we found that setting $k = 5$ produced the best results (see Figure 2). This model yields an accuracy of approximately 83.4%.

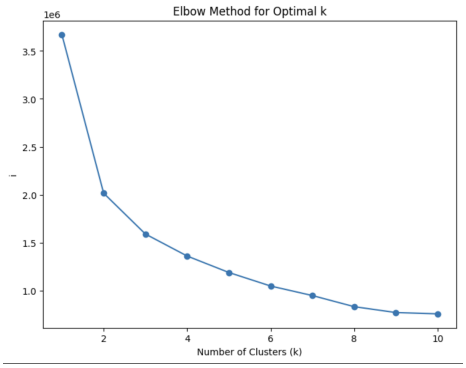

Figure 2: Elbow Graph of k values

Using Python libraries such as Matplotlib and Seaborn, we were able to produce a correlation heatmap, showcasing the correlation between all the attributes in the dataset and the target, whether or not a patient has heart disease (see Figure 3). A dark blue color suggests a strong negative correlation, while a dark red color suggests a strong positive correlation. Looking at the bottom row of the heatmap, we can see that higher chest pain (cp) and maximum heart rate

(thalach) indicate higher rates of heart disease. On the other hand, higher ST depression (oldpeak) and exercise angina (exang) have an inverse relationship with the target.

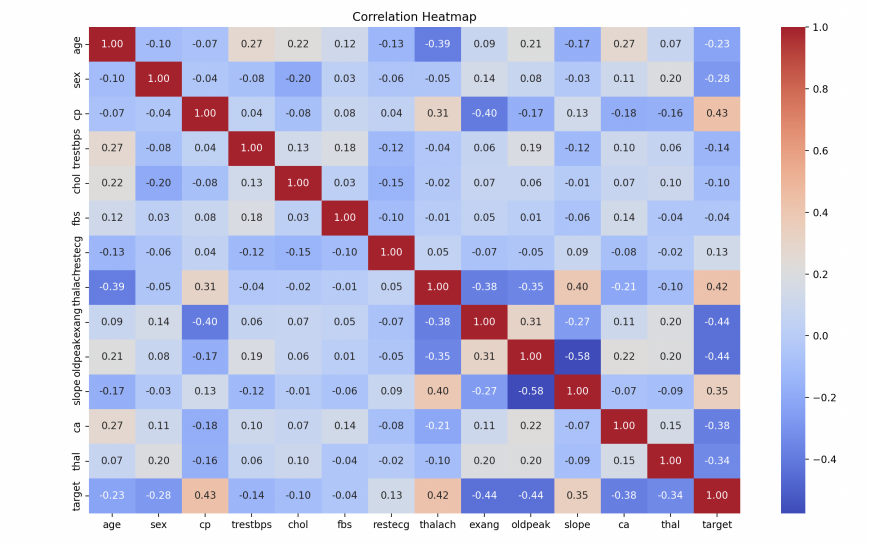

Figure 3: Correlation Heatmap

The KNN algorithm has a time and space complexity of $O(n \cdot d)$, where n is the number of training points and d is the number of features (dimensions). KNN models human thought by simply relying on the most common answer. For example, suppose you are in a group of 10 people and you ask them a question with a definite answer. You would agree with the most common answer among the 10. KNN is limited due to its high time complexity for large datasets. Since the time complexity is determined by the size of the dataset, it takes a while for datasets with many points. KNN stores the entire dataset in memory, which requires significant space. It is also very vulnerable to outliers, as the model predicts based on the closest data points, and an outlier can easily skew the predictions. Overall, KNN is a fairly accurate algorithm for prediction, but it falls short due to its time and space limitations.

7 Artificial Neural Network

The algorithm that we chose to use is the Multilayer Perceptron classifier implemented through scikit-learn's MLPClassifier. This is a type of feedforward neural network consisting of an input layer, a hidden layer, and an output layer. For the input layer, each neuron corresponds to an input feature. For the hid-

den layer, the MLP can have as many hidden layers as it needs, with each layer containing any number of nodes. The output layer works similarly to the input layer, where the number of neurons will correspond to the number of outputs.

The 4 key mechanisms found in an MLPClassifier are forward propagation, loss calculation, backpropagation, and optimization. During forward propagation, the data moves from the input layer, passing through any of the hidden layers, and ending in the output layer. Each neuron in the hidden layer processes the input as a weighted sum, then the weighted sum is passed through an activation function. The loss is then calculated using a loss function, which compares the predicted output to the actual label. The calculated loss is propagated backwards through the network during backpropagation, where gradients are computed for each weight and bias. Finally, the optimization step updated these parameters to reduce the loss, allowing the model to improve its predictions over many iterations, as seen in figure 4. The time complexity is O(i * n*(m * h + (k - 1)* h * h + h *o) where n is training samples, m is features, k is hidden layers, h is neurons, o is output neurons and i is the number of iterations.

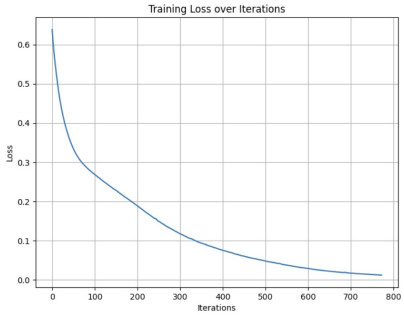

Figure 4: Training Loss Curve

Some limitations of this model are being sensitive to hyperparameters, can overfit easily, and require a large amount of data. To combat this, we use tenfold validation. This is a cross-validation technique where the data is split randomly into 10 folds. Nine folds are the training set, and the other fold is the testing set. The model gets trained and evaluated 10 times, with each fold serving as the test set once. With this technique, we would know the average accuracy of the model, giving us a more reliable estimate of the model's performance. We can also use GridSearchCV from scikit-learn to find the optimal values for the parameters. The optimal parameters for this model are ReLU as the activation function, two hidden layers consisting of 30 neurons each, a learning rate of 0.001 (which controls how fast the model learns), and the Adam optimizer. The optimized model achieved an accuracy of 98% and an average accuracy of 99%. As seen in figure 5, only 3 patients were misclassified.

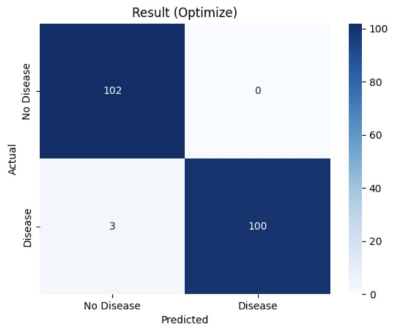

Figure 5: Result for MLPclassifier

8 Hopfield Network

The Hopfield Network is an auto-associative memory model that stores and recalls patterns, making it particularly useful for reconstructing noisy or incomplete data. Unlike traditional classifiers, it operates as a content-addressable system, converging to stable states (stored patterns) through energy minimization. The heart disease dataset's features were preprocessed into bipolar values (scaled to [-1, 1]) to align with the network's activation scheme. The network was trained using Hebb's rule, which constructs a symmetric weight matrix by strengthening connections between co-active neurons. During evaluation, the network achieved 76 percent accuracy in reconstructing both training and test patterns corrupted by 20 percent noise, demonstrating its robustness for tasks like recovering missing medical records

The weight matrix (see below Figure 6) revealed key feature correlations, such as strong ties between chest pain and heart disease outcomes, consistent with findings from the Decision Tree analysis. While less accurate than the Decision Tree (87) or ANN, the Hopfield Network's strength lies in its pattern completion capability rather than classification. Its limitations include a constrained storage capacity (0.14 * number of neurons) and susceptibility to unstable states. However, its simplicity and noise tolerance make it a viable tool for pre-processing or hybrid systems, such as denoising inputs before classification.

The time complexities for the Hopfield network are $O(p \cdot n^2)$ for the training method, where p is the number of training patterns stored inside the network, and $O(n^2)$ for the prediction and energy calculation methods. The imports used for this implementation are Pandas(used for data load and manipulation), Numpy(calculations), MinMaxScaler, Binarizer, and train.test.split from scikit-learn, which helped normalize data to 0 and 1, binarized values, and help partition the data itself, and finally matplotlib.pyplot to help visualize our findings.

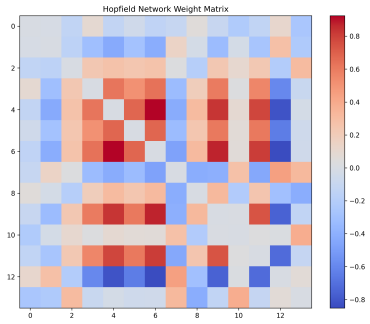

Figure 6: Hopfield Network Weight Matrix

9 Comparing the Algorithms

After writing and running each of these algorithms, we decided to compare each of them and record their strengths and weaknesses in regards to this dataset. Out of the four methods, the ANN model produced the highest accuracy at 98%. While the Decision Tree was easy to understand and visualize, it risked overfitting the data and lacked flexibility for capturing more complex patterns. KNN doesn't need training, but it struggles with larger datasets due to computational cost and memory requirements. It is also prone to noisy features. In general, the Hopfield Network was not an ideal model for classification tasks, as it was originally designed for pattern recognition and recall rather than prediction. Overall, the ANN has a great balance between accuracy, flexibility, and generalization, making it the most effective model for heart disease prediction.

10 Conclusion

In our project, we successfully implemented four machine learning algorithms: Decision Trees, K-nearest neighbors, Artificial Neural Networks, and Hopfield, to predict heart disease on a given data set. When comparing the accuracy of each algorithm, we found that ANN is the most accurate, with an accuracy of 98%. In the future, we can achieve better/ more accurate results by testing on a larger dataset, which will make our models more applicable to real-world patient scenarios. Another way we can improve our models is to incorporate data augmentation into our dataset, which can help train our models to perform more accurately in scenarios such as edge cases.