# Machine Learning in Heart Disease Prediction: A Comparative Study of Algorithms

Mehmet Cem Yedekci [*]

March 25, 2024

**Abstract**

Cardiovascular diseases (CVDs), simply heart diseases, are the leading cause of death globally, needing effective early detection tools. Traditional methods for diagnosis are time-consuming and they cost too much. The goal of this project is to conduct a comprehensive comparison analysis on the applicability of machine learning algorithms for predicting heart diseases. We used a publicly available dataset. We conducted Exploratory Data Analysis (EDA) and data preprocessing, including feature scaling and encoding, to make the data suitable for machine learning algorithms. The study included utilizing fundamental machine learning methods like Logistic Regression and Support Vector Machines, as well as more advanced techniques such as Artificial Neural Networks. The performance of our classifiers was evaluated using the metrics recall, precision, F1 Score, and accuracy. Based on our study, Logistic Regression was the most accurate model, with the accuracy rate of 90%. Furthermore, we found out that hyperparameter tuning and data standardization increased classifier performance. Our findings provide a thorough guidance for healthcare practitioners and data scientists interested in using machine learning for heart disease prediction, including all steps from data preparation through model evaluation.

## 1 Introduction

The term cardiovascular diseases (CVDs), simply heart diseases, refer to a group of disorders affecting the heart and blood vessels. These conditions include coronary heart disease, cerebrovascular disease, rheumatic heart disease, and various other heart related health complications. CVDs are leading cause of mortality

[*]Advised by: Guillermo Goldzstein from Georgia Institute of Technology

worldwide, accounting for almost 18 million of deaths annually according to the World Health Organization [Wor22]. Most of these deaths, over 80%, are caused by heart attacks and strokes [MPN+11]. Factors like poor diet, lack of exercise, alcohol consumption, and smoking increases the risk of having these heart issues. However, early diagnosis of CVDs is essential for starting appropriate medication, counseling, and therapy as soon as possible. The early identification of those at a higher risk and taking appropriate actions timely can reduce unexpected and premature mortality.

Traditional methods for diagnosing CVDs involve performing many tests such as electrocardiograms (ECGs), blood tests, and sometimes more complicated procedures are required. These tests are evaluated detailly by medical professionals [GAK+21]. The traditional way is not only time-consuming but also expensive. In an era where healthcare data is being produced in previously unreachable amounts, there is a rising need to develop quicker, more efficient approaches for diagnosing this widespread condition.

This is where machine learning's transformative potential comes into play. Machine learning is a specialized area within computer science that focuses on the development of algorithms that enable computers to make predictions or decisions based on input data [ENM15]. In other words, machine learning enables computers to learn from data, use that knowledge to handle certain situations, and make predictions that apply to various scenarios. In contrast to traditional methods, machine learning algorithms can rapidly analyze large healthcare data and uncover some patterns that may be missed by traditional diagnostic process. Machine learning algorithms have been increasingly popular in the medical industry in recent years, including the diagnosis of heart diseases. Various methods have been applied, ranging from simpler ones like logistic regression to more complex neural networks. However, there are still unanswered concerns regarding which provides the greatest performance in terms of accuracy and effectiveness.

While there has been considerable research utilizing machine learning algorithms for predicting heart diseases, most studies focus on specific algorithms without extensive comparative analyses. This research aims to bridge this gap by conducting a comprehensive comparative analysis on prediction of heart diseases using a variety of machine learning algorithms, ranging from basic models like Logistic Regression and Support Vector Machines to more advanced techniques such as Artificial Neural Networks (ANN). The research not only focuses on model implementation and evaluation, but it also dives into hyperparameter tuning and data preprocessing.

There are a lot of different goals to be accomplished with this research:

1. Examining effective techniques, such as feature scaling and encoding, for preprocessing the data for optimum model training

2. Conducting exploratory data analysis in order to provide a better understanding of the data characteristics and distributions.

3. Testing and evaluating the performance of multiple machine learning algorithms in predicting the risk of heart diseases.

4. Applying hyperparameter tuning and data standardization, analyzing their influence on model performance to optimize the performances of our models

5. Presenting a custom-designed Artificial Neural Network model, explaining its architecture and efficiency in comparison to standard models.

The idea of this work is not limited with using algorithms to achieve better results. Different algorithms have their strengths and weaknesses and may perform differently based on the specific set of data they are given. Therefore, our primary objective is to evaluate and compare these different machine learning methods on the same dataset, aiming to identify which algorithm is the most accurate in predicting heart disease.

Through this comprehensive approach, this research aims to provide valuable insights into the efficacy of various machine learning models for heart disease prediction and aims to be a resource for both healthcare practitioners and data scientists in the field.

## 2 Dataset

The dataset used in this study is obtained from Kaggle [Rah21]. The name of the dataset in Kaggle platform is “Heart Attack Analysis & Prediction Dataset”. The dataset in Kaggle was taken from UCI Machine Learning Repository [JD88], originally named “Heart Disease”, formed by a combination of four different databases:

- Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
- University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
- University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
- V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

However, the dataset in Kaggle which was extracted from UCI, only consists of the Cleveland database out of these four databases.

The dataset contains 13 medical attributes of 304 patients, helping us to identify those at risk of developing heart disease by categorizing patients into groups: those at risk and those not at risk of heart disease. The dataset is composed of 303 samples (303 rows) and 13 input features as well as 1 output feature (14 columns). Attributes are listed and described below in Table 1 .

| Attribute Name | Type | Description | Range |
|---|---|---|---|
| age | Numerical | Age of the patient in years | Values between 29 and 77 |
| sex | Binary | Gender of the patient | 1 = male<br>0 = female |
| cp | Categorical | Level of chest pain the patient suffering from | 3 = non anginal pain<br>2 = atypical angina<br>1 = typical angina<br>0 = asymptomatic |
| trtbps | Numerical | Resting blood pressure in mmHg (at entry to the health center) | Values between 94 and 200 |
| chol | Numerical | Serum cholesterol level in mg/dl (at entry to the health center) | Values between 126 and 564 |
| fbs | Binary | Fasting blood sugar according to mg/dL | Fasting blood sugar level ¿ 120 mg/dL;<br>1 = true<br>0 = false |
| restecg | Categorical | Resting ECG results | 2 = having ST-T wave abnormality<br>1 = normal<br>0 = hypertrophy |
| thalachh | Numerical | Maximum heart rate achieved | Values between 71 and 202 |
| exng | Binary | Exercise induced angina | 1 = yes<br>0 = no |
| oldpeak | Numerical | ST depression induced by exercise relative to rest | Real number values between 0 and 6.2 |
| slp | Categorical | The slope of the peak exercise ST segment | 2 = upsloping<br>1 = flat<br>0 = downsloping |
| caa | Numerical | Number of major vessels colored | 0, 1, 2, 3 |
| thall | Categorical | Thallium test result | 3 = reversable defect<br>2 = normal<br>1 = fixed defect |
| num | Binary | The predicted attribute - diagnosis of heart disease | 1: $>$ 50% diameter narrowing, more chance of heart disease<br>0: $<$ 50% diameter narrowing, less chance of heart disease |

Table 1: Description of Dataset Attributes

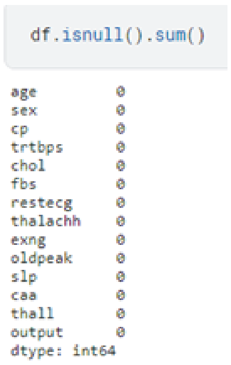
## 2.1 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) refers to analyzing data sets to identify their main characteristics by using graphical representations and statistical techniques [Gel04]. The primary goal of performing EDA is to gain insights into the data, understand its underlying structure, and identify patterns, trends, and potential outliers. EDA usually involves displaying data using various graphs and charts, calculating statistical summaries, and examining data distributions. It is an important phase in the data analysis process because it allows researchers to make better decisions in further steps.

Exploratory Data Analysis (EDA) is a critical stage in our study, diving deeply into the dataset in order to uncover insights and understand its underlying structure. Graphs and charts were plotted with the library Matplotlib present in Python [Bis19].

### 2.1.1 Missing Value Check

In order to ensure the reliability of the dataset, we checked dataset for missing values, shown in Figure 1. The positive outcome was that no missing values were detected, indicating that our dataset is complete and ready for further steps without the need for imputation or any other missing data handling procedures.



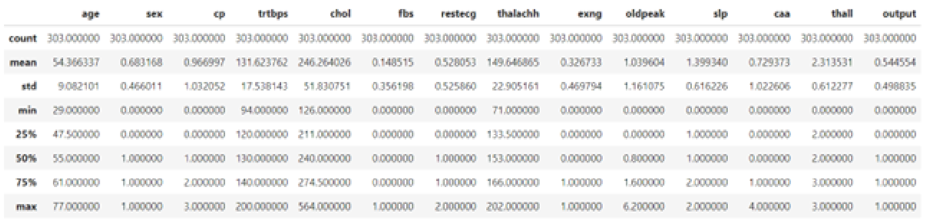
```
df.isnull().sum()
```

```
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
output      0
dtype: int64
```

Figure 1: Missing Value Count

### 2.1.2 Statistical Outline

The statistical summary presented in Figure 2 provides an overview of the dataset's statistical measures such as mean, standard deviation, minimum, maximum, and quartiles for each feature. This numerical summary provides us an initial understanding of the data's distribution.



| | age | sex | cp | trtbps | chol | fbs | restecg | thalachh | exng | oldpeak | slp | caa | thall | output |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 |
| **mean** | 54.366337 | 0.683168 | 0.966997 | 131.623762 | 246.264026 | 0.148515 | 0.528053 | 149.646865 | 0.326733 | 1.039604 | 1.399340 | 0.729373 | 2.313531 | 0.544554 |
| **std** | 9.082101 | 0.466011 | 1.032052 | 17.538143 | 51.830751 | 0.356198 | 0.525860 | 22.905161 | 0.469794 | 1.161075 | 0.616226 | 1.022606 | 0.612277 | 0.498835 |
| **min** | 29.000000 | 0.000000 | 0.000000 | 94.000000 | 126.000000 | 0.000000 | 0.000000 | 71.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 47.500000 | 0.000000 | 0.000000 | 120.000000 | 211.000000 | 0.000000 | 0.000000 | 133.500000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 2.000000 | 0.000000 |
| **50%** | 55.000000 | 1.000000 | 1.000000 | 130.000000 | 240.000000 | 0.000000 | 1.000000 | 153.000000 | 0.000000 | 0.800000 | 1.000000 | 0.000000 | 2.000000 | 1.000000 |
| **75%** | 61.000000 | 1.000000 | 2.000000 | 140.000000 | 274.500000 | 0.000000 | 1.000000 | 166.000000 | 1.000000 | 1.600000 | 2.000000 | 1.000000 | 3.000000 | 1.000000 |
| **max** | 77.000000 | 1.000000 | 3.000000 | 200.000000 | 564.000000 | 1.000000 | 2.000000 | 202.000000 | 1.000000 | 6.200000 | 2.000000 | 4.000000 | 3.000000 | 1.000000 |

Figure 2: Statistical Outline of The Dataset

### 2.1.3 Visualizing Categorical Feature Distribution

Figure 3 displays graphs showing the distributions of categories within each individual categorical feature. These graphs provide a thorough look at how data is distributed across different categories and reveal information about the frequency and balance of different categorical variables in the dataset.

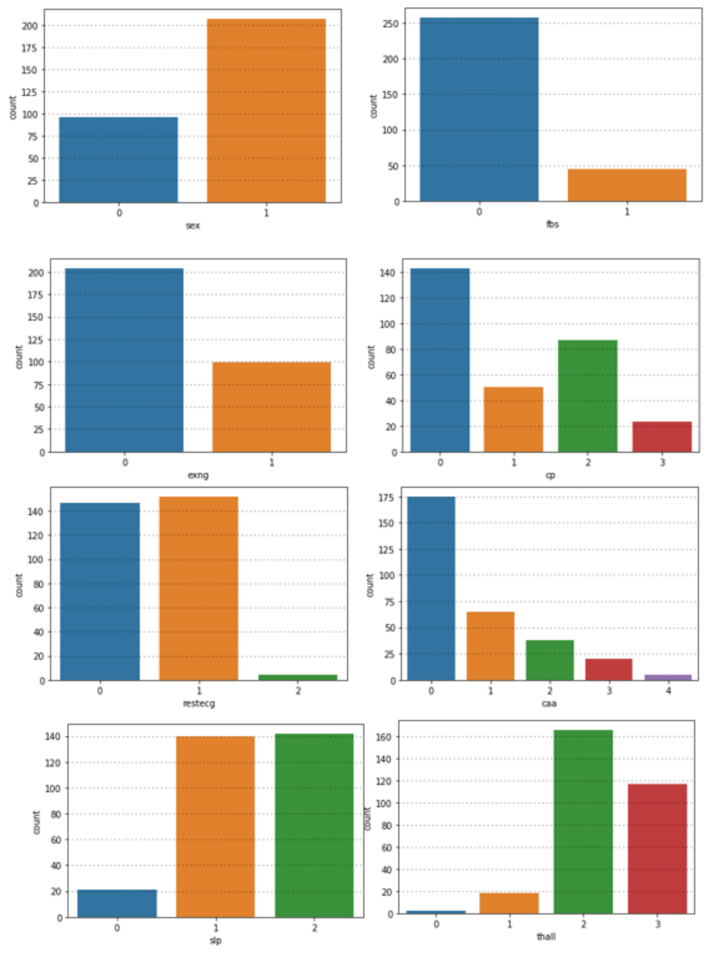


Figure 3: Categorical Feature Distributions

### 2.1.4 Visualizing Continuous Feature Distributions

Figure 4 displays box plots that give an overview of the distributions and statistical traits of our continuous features. These visualizations highlight the central tendencies, variations, and potential outliers within each feature, shedding light on their properties and their relevance in predicting heart disease risk.

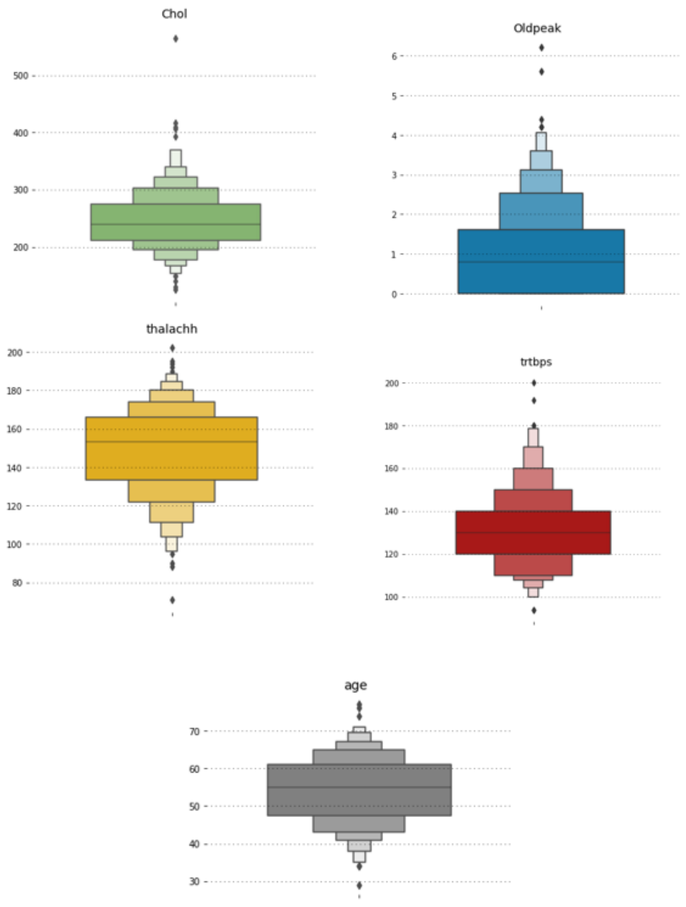


Figure 4: Continuous Feature Distributions

#### 2.1.5 Class Distributions

Figure 5 shows an overview of the label distribution, providing information on the number of instances with heart disease and those without it. While developing machine learning models for classification tasks, it is important to take the class balance into account and handle any potential class imbalance problems. Our chart confirms that dataset exhibits a balanced distribution of the cases '0' and '1'.

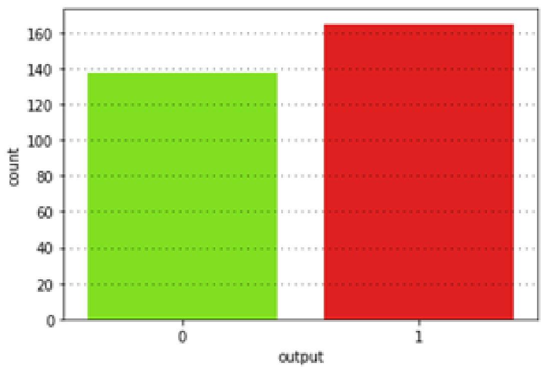


Figure 5: Class Distribution

### 2.1.6 Correlation Matrix

Understanding the interdependencies among variables has significant importance in machine learning for several reasons [Hal99], such as finding relevant features, spotting multicollinearity, and understanding our dataset's structure. Because of that, a correlation matrix assumes a pivotal role. A brief summary of the pairwise correlations between several attributes in our dataset may be seen in the matrix below.

This heatmap of our correlation matrix, shown in Figure 6, provides a quick view of the relationships. A color spectrum ranging from yellow to deep red is used to denote the strength and direction of correlations in this heatmap. Yellow indicates strong negative correlations, deep red indicates strong positive correlations, and colors in the middle indicate varying degrees of correlation.

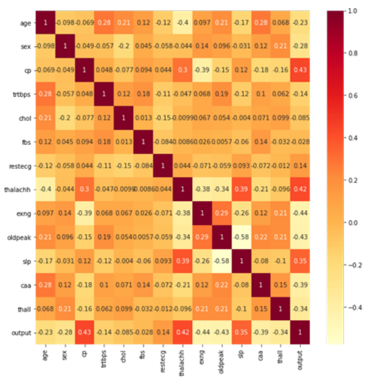


Figure 6: Correlation Matrix Heatmap

### 2.1.7 Pairplot Based on Target Variable

The pairplot, shown in Figure 7, is the next analytical tool in our EDA after the correlation matrix. A pairplot displays scatter plots for each pair of features to demonstrate their relationships, as well as graphs to show the distribution of each variable. This is a very useful method for having a quick idea of relationships and distributions [SGN20]. It allows us to see how different pieces of data relate to one another and how they look like overall. In this pairplot, each point is colored based on the target variable (output) value, making it easier to identify potential patterns or clusters that could be significant for our machine learning models.

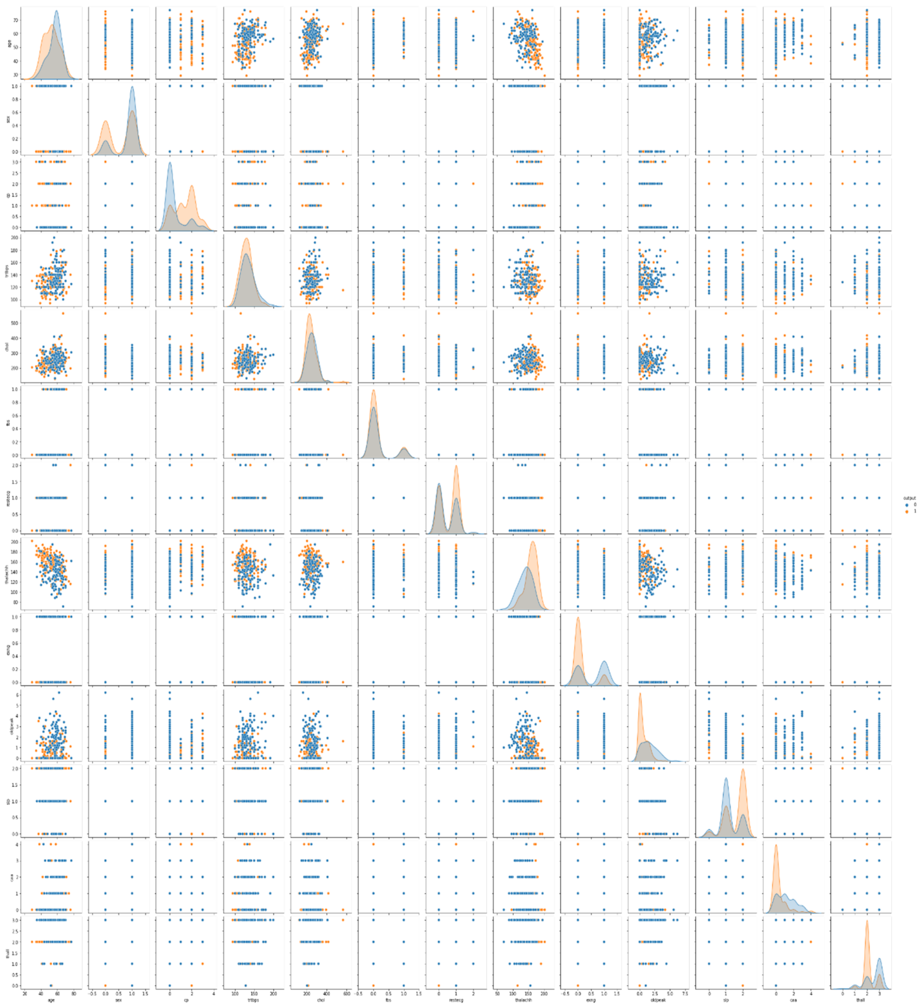


Figure 7: Pairplot

## 2.2 Data Preprocessing

### 2.2.1 Feature Categorization in the Dataset

In our dataset, we have purposefully divided the features into two distinct categories: categorical and continuous columns. Categorical columns, which stored in the list categorical_cols, include features such as 'sex', 'exng', 'caa', 'cp', 'fbs', 'restecg', 'slp', and 'thall'. These columns contain values that are limited and fixed, often representing different categories or classes. For example, the 'sex' column only contains values '1' or '0' representing male and female respectively, and 'cp' column contain '3', '2', '1' or '0' representing 4 different types of chest pains.

On the other hand, the continuous columns, which stored in the list continuous_cols, include features such as 'age', 'trtbps', 'chol', 'thalachh', and 'oldpeak'. These features can take any numerical value within a specific range, representing quantitative measurements. For example, 'age' could range from young adults to elders, and 'chol' represents varying levels of cholesterol in the blood.

Lastly, our aim is to predict the variable 'output' which tells us if someone has heart disease or not.

This categorization into categorical_cols and continuous_cols is vital because each type of variable requires specific preprocessing techniques to make the data suitable for machine learning algorithms.

### 2.2.2 One-Hot Encoding of Categorical Features

One-hot encoding [Seg18] is a technique used for converting categorical variables into a format that can be provided to machine learning algorithms to improve predictions by making the data more understandable to the machine.

For instance, consider the variable 'cp' (Chest Pain Type), which is categorical and contains the following values:

- 0: asymptomatic
- 2: atypical angina
- 1: typical angina
- 3: non-anginal pain

The machine learning algorithm may produce misleading results if it directly evaluates these values numerically. This is because the algorithm can incorrectly consider the ordinal values as continuous inputs with a significant quantitative relationship, resulting in inaccurate analytical results.

To give an example, Table 2 represents a part of the dataset that contains the 'cp' values for five different patients:

| **Patients** | **'cp' Value** |
|---|---|
| 0 | 2 |
| 1 | 0 |
| 2 | 0 |
| 3 | 3 |
| 4 | 1 |

Table 2: 'cp' Values

Upon application of one-hot encoding, this single 'cp' variable will be decomposed into four binary variables, corresponding to each unique category in the original 'cp' column, shown below in Table 3:

| **Patients** | **cp_0** | **cp_1** | **cp_2** | **cp_3** |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 |

Table 3: Encoded Version of 'cp

#### 2.2.3 Scaling of Continuous Features Using Standard Scaler

Feature scaling is the second crucial stage in the preprocessing pipeline for our machine learning models. This step is particularly important because many machine learning methods do not perform well when the input features have different scales or when they are on different ranges.

Many machine learning algorithms are sensitive to the scale of input features. Algorithms like k-NN, SVM, and Logistic Regression calculate the distance between data points to make predictions. When the features are not on a similar scale, the algorithm might give higher importance to variables with a higher value, which can result in an inaccurate model.

In order to overcome this obstacle, we implemented Standard Scaler for scaling features by standardization. We aimed to mitigate issues related to the range and distribution of data points thus improving the models' efficiency and accuracy.

Mathematically, the transformation by Standard Scaler [AHRD23] can be understood as follows:

For each feature $X_i$ ; the scaler calculates its mean $\mu$, and standard deviation $\sigma$. Then, each data point x of the feature $X_i$ is transformed using the following formula:

$$Z = \frac{x - \mu}{\sigma}$$

Where;

- $Z$ is the standardized feature value (scaled datapoint)
- $x$ is the original feature value (original datapoint).
- $\mu$ is the mean of the feature.
- $\sigma$ is the standard deviation of the feature.

When a feature is standardized, this formula is applied to every single value in the feature, and the original values are replaced with the resulting standardized values. The outcome is a feature that has been scaled such that its values (datapoints) now hover around zero (we get values ranging from -1 to +1), which makes it easier for machine learning algorithms to find patterns or make decisions based on this data.

### 2.2.4 Train-Test-Validation Split

It is critical in machine learning studies to ensure that models are trained and evaluated on distinct data sets. This not only ensures that the model is learning patterns from the data, but it also validates its performance on previously unseen data. To achieve this goal, we separated the available dataset into three different subsets: training, testing, and validation [TYW$^+$21].

Training set is the largest subset of the data, generally ranging from 60% to 80% of the entire dataset. The training set is used to 'teach' the machine learning algorithm. All the weights and transformations are learned on this dataset.

After the model has been trained and fine-tuned, its performance is evaluated using the testing set, which typically comprises 10% to 20% of the dataset. This is data that the model has never seen before, thus it provides an accurate assessment of how the model would perform in a real-world scenario.

Validation set sits in between the training and testing phases. During the training phase, it is used to fine-tune model parameters and offer an unbiased evaluation of model fit. The model sees this data but does not learn from it, therefore the validation set is an important component for hyperparameter tuning [TYW+21].

We initially split the dataset into two parts: 80% for training and 20% for testing. Then, we took 20% of the data from the training set to construct a validation set. As a result, the final data distribution was 60% training, 20% validation, and 20% testing.

# 3 Method

## 3.1 Machine Learning Algorithms

A machine learning algorithm is a computing process that uses input data to complete a prediction or a classification task without being explicitly programmed to do so [Sar21]. These algorithms automatically adjust or adapt their architecture as a result of repetition and experience.

The adaptation process is known as training. Training involves providing samples of input data along with desired outputs. The algorithm is then set up in the most optimal manner so that it can both generate the expected outcome when given the training data and generalize to produce the desired result when given new, previously unseen data.

The "learning" component of machine learning is this training. The training does not have to be restricted to a primary adaption over a set period of time. An effective algorithm may engage in "lifelong" learning as it analyses current data and learns from its errors, much like humans can [ENM15]

Supervised learning [CNM06] is the machine learning method to develop a function that translates an input to an output using example input-output pairs. It uses labeled training data made up of a collection of training samples to form a function.

Train and test datasets are created from the input dataset. The output variable in the train dataset has to be predicted or categorized. All algorithms extract some sort of patterns from the training dataset and use them to predict or classify the test dataset. [Man20]

In this work, we introduce variety of supervised machine learning algorithms. This involves both non-tree-based and tree-based algorithms, as well as a custom-built Artificial Neural Network (ANN) model. Further in the results section, we will analyze the effectiveness of them in predicting heart disease.

### 3.1.1 Non-Tree Based Algorithms

Non-tree-based algorithms are a type of machine learning algorithms that doesn't use decision trees or ensembles of trees as their underlying structure to make predictions or decisions. To learn from data, these algorithms depend on various mathematical approaches. The study provides an explanation of the algorithms used in the given order, namely Support Vector Machines [Sut16], Logistic Regression [HJLS13], K-Nearest Neighbors [CD21], and Naive Bayes [R$^+$01] [M$^+$06].

#### 3.1.1.1 Support Vector Machines (SVM)

Support Vector Machine is a supervised learning algorithm. It is frequently used for regression and classification problems. However, this method works greatest for categorization tasks. Support vector machines construct optimal separating boundaries in the data set by solving a constrained quadratic optimization problem. Various levels of nonlinearity and adaptability can be implemented in the model by distinct kernel functions [DOM02].

The SVM method's goal is to find the optimal line or decision boundary, known as a hyperplane, which can divide n-dimensional space into classes, allowing us to quickly classify new data points in the future. SVM selects the extreme vectors and points that will be used to generate the hyperplane. The algorithm's name comes from support vectors, which are the term for these significant conditions. A separating hyperplane for two-dimensional data points is shown in Figure 8. the circles and the squares represent data points in classes -1 and +1, respectively. On the dotted lines, in black, lie the support vectors.

In this algorithm, each data attribute is shown as a point in a space with n dimensions, where n is the number of features. After drawing the graph, classification is performed by selecting the hyper-plane that best separates the two groups.

Shortly, the SVM model's goal is to identify the space in the data matrix where multiple data groups may be formed with a widely separation on the hyperplane.

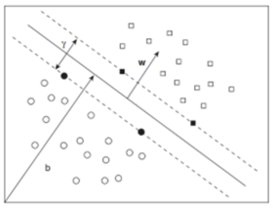


Figure 8: A separating hyperplane for two-dimensional data [MTC09]

#### 3.1.1.2 Logistic Regression (LR)

Logistic regression is a supervised machine learning algorithm that can be used to solve both regression and classification problems. It is mostly used for binary classification problems. In logistic regression, probability is used to predict how the data will be categorized [SV21].

Logistic Regression uses the sigmoid function. Sigmoid determines the probability of a new observation belonging to a certain class by transforming the model's linear output into a value between 0 and 1, making it very useful for binary classification problems [Sza21].

The mathematical expression for the sigmoid function $\sigma(z)$:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

In this equation, $z$ is a linear function of the input features $x$ and their corresponding weights $w$ given by:

$$z = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + x_n \cdot w_n + b$$

Here, $x_1$, $x_2$, $\cdots$, $x_n$ are input features $x_1$, $x_2$, $\cdots$, $x_n$ are learned weights and $b$ is the bias term.

The graph of the sigmoid function is displayed in Figure 9. The sigmoid function has an S-shaped curve. As $z$ approaches positive infinity, $\sigma(z)$ approaches 1; and as $z$ approaches negative infinity, $\sigma(z)$ approaches 0. This squeezing effect ensures that the output is bounded between 0 and 1, making it useful for estimating probabilities.

A threshold value, usually 0.5, is then applied to this probability to deliver a binary outcome: if the computed probability is greater than or equal to the threshold, the model predicts that the instance belongs to the positive class; otherwise, it belongs to the negative class.

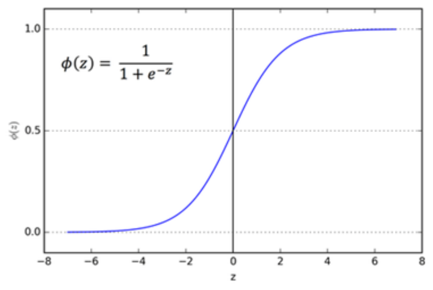


Figure 9: Graph of Sigmoid Function [Far19]

#### 3.1.1.3 K-Nearest Neighbors (k-NN)

K-Nearest Neighbor is one of the most straightforward but efficient classification algorithms [UHL+22]. It is an instance-based learning method that does not come with a complete theoretical model.

The algorithm classifies unknown instances based on their similarity to known instances in the training set. The k-NN method does an extensive search to identify the 'k' training samples that are closest to the new unclassified instance when it is added to the pattern space. In other words, it makes decisions for classification of the data by calculating the distances between the new unknown data point and all the known data points in the training set. Numerous techniques, including the Euclidean distance, Manhattan distance, and other metrics, might be used to determine the distance.

The calculation of Euclidean distance can be made using following formula:

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

Where;

- $n$ =number of dimensions
- $x_i, y_i$ =data points

And also, calculation of Manhattan distance can be made using following formula:

$$D_M = \sum_{i=1}^{n} |x_i - y_i|$$

Once the 'k' closest neighbors are identified, the algorithm examines the labels or categories of them. The new, unclassified data point is then classified using the category that most frequently occurs among its closest 'k' neighbors. In other words, the class of the unclassified data point is predicted by majority vote among its closest neighbors.

For example, if 'k' is 3, and two out of the three closest neighbors to the unknown point are labeled as "Class A" and one is labeled as "Class B," then the new data point would be classified as "Class A" due to the majority rule.

In some variations of k-NN, weighted voting can be applied, where closer neighbors have more influence on the classification of the new data point. This is often done by assigning weights to the votes of the neighbors based on their distance to the unknown point.

#### 3.1.1.4 Naive Bayes (NB)

The Naive Bayes classifier is a probabilistic machine learning model based on Bayes Theorem and predicated on the idea of predictor independence. The classifier is termed "Naive" because it makes an important assumption that the features in the dataset are mutually independent of each other, given the class label [IWASA07]. In other words, the presence of one feature does not affect the presence of another.

The fundamental principle behind the Naive Bayes classifier Bayes' Theorem [Ber19], can be mathematically expressed as:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Where,

- $P(A|B)$ is the posterior probability: The probability of the event A occurring given that B is true.
- $P(B|A)$ is the likelihood: The probability of the event B occurring given that A is true.

- $P(A)$ is prior probability.
- $P(B)$ is marginal probability.

In the context of Naive Bayes classification, the prediction $\hat{y}$ for a given feature vector $x$, is determined by identifying the class $c$ that maximizes the posterior probability $P(c|x)$. Mathematically, this is expressed as:

$$\hat{y} = \arg\max_{c} P(c|x)$$

According to Bayes' theorem [Ber19], after simplifications the expression can be written as:

$$\hat{y} = \arg\max_{c} \prod_{i=1}^{n} P(x_i|c) \times P(c)$$

Here, $n$ is the number of features in $x$. The class $c$ that maximizes this expression is selected as the predicted label $\hat{y}$ for the given input $x$. This mathematical framework allows the Naive Bayes classifier to make efficient and effective classifications.

### 3.1.2 Tree-Based Algorithms

Tree-based classification models are effective tools within supervised learning algorithms in the field of machine learning [PP18]. They give a logical, step-by-step approach to decision-making. These models use a hierarchy of conditional statements to divide the training data into subsets, each adding a layer of complexity to the final model. These conditional statements, also known as splits, are designed to optimize information gain or other similar criteria, allowing for more precise predictions.

The complete model may be represented graphically as a tree structure, effectively functioning as a roadmap of logical tests. Beginning at the root, each branch provides a decision rule that further splits the data, eventually leading to the leaf nodes, which contain the final prediction or classification label. This research investigates the sequence in which the algorithms Decision Tree [CA21], Random Forest [BD16], Gradient Boosting [BCMM21] and XGBoost [CHB$^+$15] are utilized.

#### 3.1.2.1 Decision Tree (DT)

Decision Tree (DT) is regarded as one of the earliest and widely used machine learning algorithms. Decision trees are prediction models that analyze data in the form of a tree. A DT, which is a flowchart-like structure, typically consists of multiple layers of nodes. The highest level is known as the root or parent node, while the lower levels are known as child nodes. Each internal node represents an attribute test, each branch represents the outcome of the attribute test, and each leaf node represents the class label [SK16].

The root node has zero entering degree, which indicates it has no incoming edges. Initially, all tuples are at the root node. The tree achieves classification by splitting its branches, with each split representing a test on a data attribute.

A decision tree classifies data items by asking a sequence of questions regarding the objects' attributes. Each question is contained in a node, and each internal node refers to one child node for each potential response to its query. As a result, the questions create a hierarchy, which is stored as a tree [SK16].

In its most basic form, yes-or-no questions are asked, and each internal node has a child that answers "yes" or "no." Following the path from the topmost node, the root, to a node without children, a leaf, an item is sorted into a class based on the responses that apply to the object under consideration. An item is assigned to the class that corresponds to the leaf it reaches. An example of a simple decision tree is shown in Figure 10.

The primary principle behind any multistage procedure is to break down a complex decision into a union of numerous simpler decisions, hoping that the final answer received this way will closely match the desired solution.

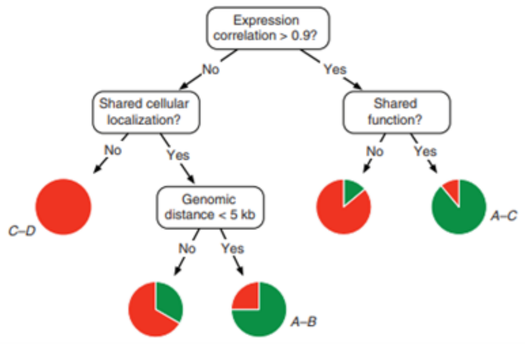


Figure 10: An Example of a Decision Tree [KS08]

#### 3.1.2.2 Random Forest (RF)

Random Forest (RF) is one of the most widely used supervised machine learning techniques for classification and regression. It works by building a forest out of several random and disconnected Decision Trees (DTs), which perform as an ensemble, throughout the training phase [Pal05].

Each Decision Tree makes a random selection of the sample features and the portion of the sample data set that is going to be used as the training set. This is done with the help of a random vector that is utilized as a parameter.

During the testing period, each decision tree, which is an element of the forest, predicts the class label for each instance. The results of all the decision trees are combined before any predictions are made. After each tree has made its prediction regarding the class label, the final decision for each set of test data is made by a process of majority voting. A simple scheme for the structure of Random Forest is presented in Figure 11. The class label with the most votes is considered the best appropriate label for the test data. This cycle is repeated for every single piece of data contained in the collection.

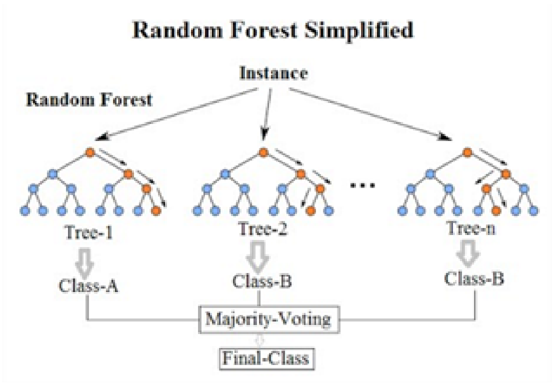


Figure 11: Simplified Structure of a Random Forest [SK19]

#### 3.1.2.3 Gradient Boosting (GB)

Gradient Boosting Machine is a very effective and extensively used machine learning technique that is mainly used for regression and classification problems. It is a member of the boosting algorithm family, which are ensemble techniques intended to improve the overall performance of weak learners by combining the results they produce [NK13]. Gradient boosting obtains predictions in a sequential manner. Each decision tree in gradient boosting predicts the error of the previous decision tree, consequently boosting the error. The error, or residual, acquired after developing a model is known as the gradient. Boosting means to improve. Gradient boosting is a technique for gradually reducing error. Rather of focusing just on bias reduction, gradient boosting employs the gradient descent technique to reduce residuals. The approach computes the negative gradient of the loss function with respect to the prediction at each iteration. The model is then trained on these pseudo-residuals and added to the ensemble of models explained in Figure 12.

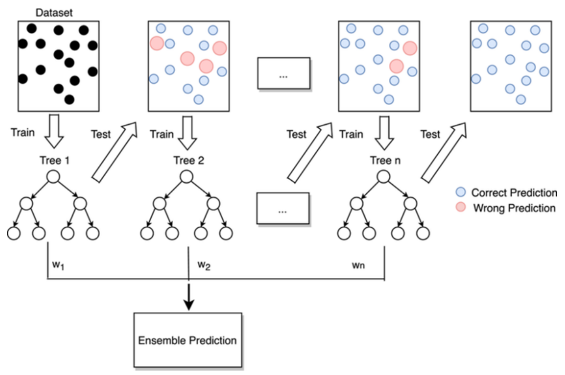


Figure 12: Flow Diagram of Gradient Boosting [ZLV$^{+}$21]

### 3.1.2.4 XGBoost(XGB)

XGBoost, which stands for "eXtreme Gradient Boosting," is an improved and optimized version of the gradient boosting technique, designed to be very efficient, scalable, flexible, and portable.

XGBoost, like gradient boosting, creates an ensemble of weak learners, often decision trees, in a sequential manner to generate a strong learner that accurately predicts the target variable [CG16].

It is essentially an enhanced implementation of the gradient boosting method with additional advanced features and improvements explained in Table 4, focused on performance and speed:

| **Features** | **Description** |
|---|---|
| Regularization | XGBoost includes an additional regularization term in the loss function, which controls the complexity of the individual trees. This helps in reducing overfitting. |
| Handling Missing Data | XGBoost has an in-built routine to handle missing values, thereby providing robustness to the model. |
| Parallel and Distributed Computing | XGBoost is designed to be highly efficient. It can utilize the power of parallel processing (on a single machine) and distributed computing (across multiple machines) to train models, which is one reason why it's faster than many other implementations of gradient boosting. |
| Flexibility | XGBoost allows users to define custom optimization objectives and evaluation criteria, which makes it versatile for a wide array of problems. |
| Tree Pruning | Unlike other gradient boosting methods that grow a tree by level, XGBoost grows a tree depth-wise and then prunes it using the "max depth" parameter, making it more computationally efficient. |
| Cross-Validation | Inbuilt k-fold cross-validation is another feature that sets XGBoost apart from the traditional GB, thereby providing a more robust measure of error during the model training phase. |
| Column Block | The algorithm utilizes a compressed memory-efficient format for storing the dataset to optimize cache performance. This leads to faster computation. |

Table 4: XGBoost Improved Features

Designed for speed and performance, XGBoost has gained popularity for its effectiveness in competitions and real-world machine learning tasks.

### 3.1.3 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANNs) are computational models inspired by the biological neural structures found in the human brain [ZHS09]. Designed to complete tasks faster than standard computing methods, ANNs are a critical technology in the machine learning realm. Similar to the human brain, which consists of interconnected neurons transmitting signals, ANNs are composed of a network of artificial neurons, or nodes, designed to work collaboratively to perform complex tasks [Gra13].

Structural Overview: The architecture of an ANN comprises three fundamental layers: the input layer, one or more hidden layers, and the output layer. Each of these layers is made up of nodes that simulate the function of biological neurons, shown in Figure 13. The nodes are interconnected by "edges", drawn in Figure 14, which can be thought of as virtual synapses [BH00]. These edges are weighted, and the values of these weights are iteratively updated during the learning phase. Moreover, each node is associated with a bias term, which allows the model to fit the data better.

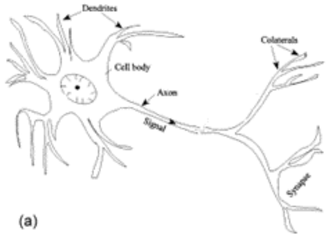


Figure 13: Scheme of A Biological Neuron [BH00]

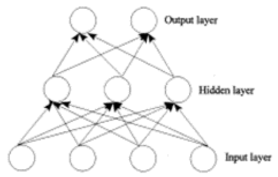


Figure 14: Scheme of An Artificial Neural Network [BH00]

Functional Dynamics: In terms of its operation, an ANN is highly similar to the neural processes of the human brain. The input layer serves as the initial point of contact for external signals. It captures data from the external environment and channels it to the hidden layer. The hidden layer, lying between the input and output layers, is the computational engine of the network [WF18]. It identifies hidden patterns and characteristics in the ingested data by performing complex, often non-linear, computations. The extracted patterns are then passed on to the output layer, which provides the final prediction or classification result.

Data Flow: The flow of information starts from the input layer and proceeds in a feedforward manner through the hidden layers to reach the output layer. Unlike biological neurons that communicate through electrical impulses, nodes in the input layer of an ANN receive data inputs, which they relay to the hidden layer after applying specific weights [Abr05]. The hidden layer, in turn, refines these inputs into meaningful patterns or features through mathematical transformations. These refined features are then transferred to the output layer, which yields the final output based on the type of the task—classification, prediction, or any other machine learning objective [ALPM$^+$13]. Overview of the ANN is explained in Table 5:

| **Concepts** | **Description** |
|---|---|
| Sequence of Layers | A network is essentially a sequence of interconnected layers. |
| Input Layer | This is the first layer in the network, and it receives the initial data for the machine to process. In our case, the input $X$ would be the features of an example. |
| Output Layer | The last layer of the network produces the output, denoted as $\hat{y}$ which is the model's prediction for the given input. |
| Hidden Layers | These are the layers that reside between the input and the output layers. They perform most of the computation required by the network. |
| Layer as a Function | Each layer can be seen as a mathematical function that transforms its input data into a specific output. |
| Layer Types | Different types of layers can be used, such as convolutional layers, max-pooling layers, and dense layers, depending on the application. |
| Data Flow | The output of one layer serves as the input for the next layer, establishing a chain of computational steps. |
| Learning Process | During the training phase, the network learns by adjusting the weights and biases based on the error of its predictions. |
| Backpropagation | This is the most common training algorithm used for ANNs. It minimizes the error by adjusting the weights in the reverse order, starting from the output layer, and moving toward the input layer. |
| Cost Function | ANNs utilize a cost function (such as Mean Squared Error for regression tasks or Cross-Entropy Loss for classification tasks) to measure the difference between the predicted output and actual target. |
| Epochs and Mini batches | ANNs are often trained using the entire dataset for several iterations (epochs). Mini-batch gradient descent can be used to update the weights using a subset of the data, which is computationally more efficient. |
| Regularization | Techniques like dropout, weight decay, and early stopping are used to prevent overfitting, a common problem in ANNs due to their high capacity for learning complex patterns. |

Table 5: Concepts in ANN

The architecture is influenced by a range of factors [DSHSAF+17]. The factors determining network architecture is explained in Table 6:

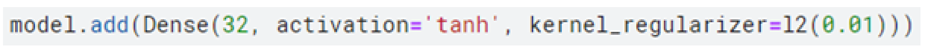
| **Factors** | **Description** |
|---|---|
| Number of Layers | This refers to the total count of layers in the network, including input, hidden, and output layers. |
| Type of Each Layer | Different tasks may require different types of layers. |
| Hyperparameters | These are settings or configurations that are external to the model but influence its behavior—like learning rate or the number of neurons in a layer. |
| Activation Functions | These are mathematical functions associated with each layer, like ReLU or Sigmoid, that introduce non-linear properties into the system. |

Table 6: Factors Determining the Architecture of ANN

#### 3.1.3.1 Architecture of Our Custom ANN Model

The architecture of our custom Artificial Neural Network (ANN) is designed to optimize performance for the specific problem at hand.

Input layer: In our custom model, the input layer (Figure 15) consists of 32 neurons, utilizing the hyperbolic tangent (tanh) activation function [DSC22].



```
model.add(Dense(32, activation='tanh', kernel_regularizer=l2(0.01)))
```

Figure 15: Code for The Input Layer Formation

The mathematical representation of the tanh [SSA17] function $tanh(x)$ is

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Graph of tanh function is shown in Figure 16:

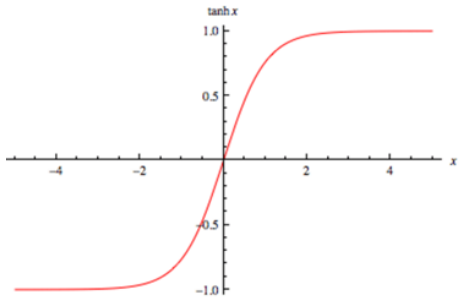


Figure 16: Graph of tanh Function

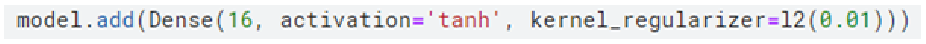
Additionally, the layer incorporates L2 regularization [TZ22] with a regularization factor of 0.01 to mitigate the effects of overfitting. L2 regularization is used to constrain the weights and thus minimize overfitting. The formula for L2-regularized loss is:

$$\text{Loss}_{L2} = \text{Loss}_{\text{Original}} + \lambda \sum_{i} \omega_i^2$$

Where:

- $\lambda$ is the regularization parameter.
- $\omega_i$ represents each weight in the neural network.

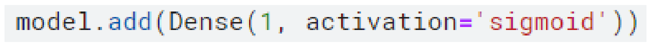
Hidden Layers: Our model includes a single hidden layer (Figure 17) to capture the complexities and intricacies in the data. This hidden layer has 16 neurons and also employs the tanh activation function. L2 regularization is applied here as well with a factor of 0.01, consistent with the input layer.



```
model.add(Dense(16, activation='tanh', kernel_regularizer=l2(0.01)))
```

Figure 17: Code for Hidden Layer Formation

Output Layer: The output layer is the final layer in the network, responsible for producing predictions. In the case of our binary classification problem, the output layer (Figure 18) consists of a single neuron employing the sigmoid [PWS+20] activation function.



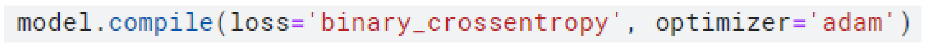
```
model.add(Dense(1, activation='sigmoid'))
```

Figure 18: Code for Output Layer Formation

Graph of sigmoid function was previously shown in Figure 9. The mathematical expression for the sigmoid function $\sigma(z)$ is:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Model Compilation and Training: After defining the layers, the model is compiled (Figure 19) using the Adam [Zha18] optimization algorithm with a binary cross-entropy loss function [ZS18], suitable for binary classification problems.



```
model.compile(loss='binary_crossentropy', optimizer='adam')
```

Figure 19: Code for Compiling the Model

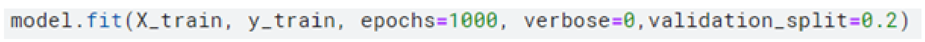
For $N$ data points, the binary cross-entropy loss is often calculated as the average loss over all data points:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$$

Where:

- $y$ is the true label for a given data point, which is either 0 or 1 in binary classification.
- $\hat{y}$ is the predicted label, a number between 0 and 1 outputted by the model.

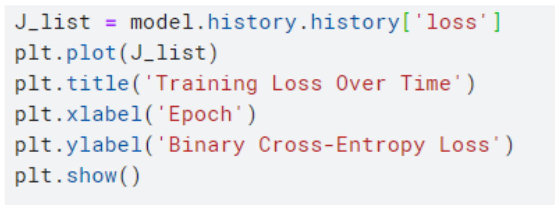
The model is trained (Figure 20) on the dataset for 1000 epochs with a validation split of 20% to monitor performance on unseen data.



```
model.fit(X_train, y_train, epochs=1000, verbose=0,validation_split=0.2)
```

Figure 20: Code for Model Training

To evaluate the effectiveness of the training process, the training loss is plotted over the epochs (Figure 21). This plot serves as a valuable tool for diagnosing the model's learning behavior.



```
J_list = model.history.history['loss']
plt.plot(J_list)
plt.title('Training Loss Over Time')
plt.xlabel('Epoch')
plt.ylabel('Binary Cross-Entropy Loss')
plt.show()
```

Figure 21: Code for Plotting the Training Loss

This architecture was chosen based on preliminary experiments and was found to deliver robust and reliable performance on the task at hand. Further details on model evaluation and comparisons with other architectures will be discussed in subsequent sections of this paper.

## 3.2 Hyperparameter Tuning

Through training, several parameters of machine learning algorithms are determined. In addition to that, the majority of machine learning algorithms contain parameters that must be adjusted before being utilized.

Hyperparameters are the name given to such parameters. In many circumstances, an algorithm's performance on a given task is highly affected by its hyperparameter settings. For achieving best performance, the hyperparameters must be tuned. [WMV20] Hyperparameter tuning refers to the process of systematically searching for the optimal set of hyperparameters that regulate the behavior of a machine learning model. Unlike model parameters, which are learned during training, hyperparameters are not learned from the data but must be set prior to the training process [RKVR19]. The primary objective is to fine-tune the machine learning model to achieve the best performance.

In our research, hyperparameter tuning was performed to optimize each machine learning model for the most accurate predictive performance. We utilized the GridSearchCV [RKVR19] method from the Scikit-learn library to systematically work through multiple combinations of the different type of hyperparameters, cross-validating as it goes to determine which tune gives the best performance.

### 3.2.1 Tuning of Support Vector Machines (SVM)

The hyperparameters tuned in SVM are shown below in Table 7:

| **Hyperparameter** | **Description** | **Interval** |
|---|---|---|
| C | Regularization parameter | {0.01, 0.1, 0.0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 50, 100} |
| gamma | Parameter that influences the shape of the decision boundary | {0.000001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10} |
| kernel | Specifies the kernel type to be used | {'linear', 'poly', 'rbf', 'sigmoid'} |

Table 7: SVM Hyperparameters

### 3.2.2 Tuning of Logistic Regression (LR)

The hyperparameters tuned in LR are shown below in Table 8:

| **Hyperparameter** | **Description** | **Interval** |
|---|---|---|
| C | Inverse of regularization strength | {0.01, 0.1, 0.0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 50, 100} |
| penalty | Used to specify the norm used in the penalization | {'l1', 'l2', 'elasticnet'} |
| solver | Algorithm to use in the optimization problem | {'newton-cg', 'lbfgs', 'liblinear', 'sag'} |

Table 8: LR Hyperparameters

### 3.2.3 Tuning of K-Nearest Neighbors (k-NN)

The hyperparameters tuned in k-NN are shown below in Table 9.

| **Hyperparameter** | **Description** | **Interval** |
|---|---|---|
| n_neighbors | Number of neighbors to consider inverse of regularization strength | list(range(1,30)) |
| leaf_size | Inverse of regularization strength | list(range(1,50)) |
| weights | Weight function used in prediction | {'uniform','distance'} |
| metric | Distance metric used | {'minkowski','euclidean','manhattan'} |

Table 9: k-NN Hyperparameters

### 3.2.4 Tuning of Gaussian Naive Bayes (GNB)

The hyperparameter tuned in GNB are shown below in Table 10

| **Hyperparameter** | **Description** | **Interval** |
|---|---|---|
| var_smoothing | Used for smoothing the variance of the features | np.logspace(0,-9, num=100) |

Table 10: GNB Hyperparameters

### 3.2.5 Tuning of Bernoulli Naive Bayes (BNB)

The hyperparameter tuned in BNB is shown below in Table 11:

| **Hyperparameter** | **Description** | **Interval** |
|---|---|---|
| alpha | Used for additive smoothing | {0.0, 0.01, 0.1, 0.5, 1.0, 1.5, 5, 10} |

Table 11: BNB Hyperparameters

### 3.2.6 Tuning of Random Forest (RF)

The hyperparameters tuned in RF are shown below in Table 12:

| **Hyperparameter** | **Description** | **Interval** |
|---|---|---|
| n_estimators | The number of trees in the forest | {25, 50, 100, 150, 200, 300, 1000} |
| max_depth | The maximum depth of each decision tree | {80, 90, 100, 110} |
| max_features | The number of features to consider for each split | {2, 3} |
| min_samples_leaf | The minimum number of samples required to be at a leaf node. | {3, 4, 5} |
| min_samples_split | The minimum number of samples required to split an internal node. | {8, 10, 12} |

Table 12: RF Hyperparameters

### 3.2.7 Tuning of XGBoost (XG)

The hyperparameters tuned in XGBoost are shown below in Table 13

| **Hyperparameter** | **Description** | **Interval** |
|---|---|---|
| n_estimators | The number of boosting rounds or trees to build | {60, 220, 40} |
| learning_rate | Controls the contribution of each tree added to the model | {0.1, 0.01, 0.001, 0.05} |
| max_depth | Maximum depth per tree | {2, 10, 1} |

Table 13: XGBoost Hyperparameters

## 3.3 K-Fold Cross-Validation

We used K-Fold Cross-Validation as part of the model evaluation procedure to verify the reliability and generalizability of our machine learning models. This technique provides a comprehensive way to evaluate the performance of a predictive model and reduces the risk [WY20] of the results being affected by the initial train/test split.

In K-Fold Cross-Validation, the original dataset is randomly divided into 'K' equally sized folds (subsets) [AGG+12]. Following that, the model is trained 'K' times, with 'K-1' of the folds used for training and the remaining fold used for testing. This method is continued until each fold has exactly served as the test set once. The performance metrics are then averaged over the 'K' folds to create a single, composite metric that is less sensitive to the dataset's initial random splitting [RPL10].

For our study, we chose a 10-fold Cross-Validation, widely considered a standard and effective choice for balancing both bias and variance in model evaluation.

Figure 22 represents a 10-fold cross-validation approach. The figure depicts that the dataset is split into 10 folds or groups, where 9 folds participate in model training and remaining 1 fold participates in evaluation of the training in each iteration. In our study, 10-fold cross-validation is employed. Cross-validation is performed to protect against overfitting in a predictive model.

## 3.4 Metrics for Evaluation

When evaluating the performance of classification models, the concepts of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) [ZGCH21] form the basis for many important metrics such as precision, recall, and accuracy [V+21]. Here's a brief explanation of each:

- True Positives (TP) are the cases in which the model predicted the positive class, and the true label was also positive.

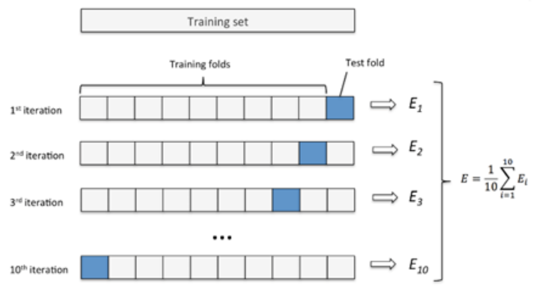


Figure 22: 10-k Cross-Validation [SLDS19]

- True Negatives (TN) are the cases in which the model predicted the negative class, and the true label was also negative.
- False Positives (FP) are the cases in which the model predicted the positive class, but the true label was negative.
- False Negatives (FN) are the cases in which the model predicted the negative class, but the true label was positive. This is often referred to as a "Type II error."

### 3.4.1 Accuracy

Accuracy is the ratio of correctly predicted instances to the total number of instances. It's a general indicator of how well the model performs across all classes.

$$\text{Accuracy} = \frac{\text{True Positives (TP)} + \text{True Negatives (TN)}}{\text{Total Number of Samples}}$$

While easy to interpret, accuracy can be misleading in the case of imbalanced datasets.

### 3.4.2 Precision

Precision (also known as the positive predictive value) is the fraction of True Positives over the sum of True Positives and False Positives. It shows how many of the items identified as positive are actually positive.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

High precision means that false positive errors are low. It is a good measure to use when the cost of a false positive is high.

### 3.4.3 Recall

Recall (or Sensitivity or the True Positive Rate) measures the fraction of True Positives over the sum of True Positives and False Negatives. It shows how many of the actual positive cases were identified correctly.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

High recall indicates that false negative errors are low. It is a good measure to use when the cost of a false negative is high.

### 3.4.4 F1 Score

The F1 Score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is a good measure to use if you need to seek a balance between these two metrics.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1 score is particularly useful in contexts where either false positives or false negatives carry a significant cost.

## 4 Results and Discussion

We collected the dataset from Kaggle. We applied EDA to the data, in order to understand its underlying structure and identify patterns, and also potential anomalies or outliers. We presented a variety of graphs to show the distributions of the data, and the relations between the features. After EDA, we preprocessed the data by encoding the categorical features using one-hot encoding and standardizing the continuous features using standard scaler. Then, we split our dataset into train, validation, and test subsets.

We tested a variety of non-tree based and tree-based machine learning classifiers, and a custom ANN on the dataset. We performed hyperparameter tuning on most of our classifiers. The optimal set of hyperparameters for each of the model is shown in Table 14. We used K-Fold Cross-Validation as part of the models' evaluation procedure to verify the reliability and generalizability. All the classifiers are trained and tested using 10-fold cross-validation.

| **Models** | **Optimal Hyperparameter Values** |
|---|---|
| SVM | 'C' value of 50, a 'gamma' value of 0.01 and kernel type of 'rbf' |
| LR | *C* value of 1, a 'penalty' type of *l*2, and solver type of 'lbfgs' |
| k-NN | 'n_neighbors' value of 3, a 'leaf_size' value 01, 'weights' type of uniform, and 'metric' type of manhattan |
| GNB | 'var_smoothing' value of 0.04328761281083057 |
| BNB | 'alpha' value of 0.0 |
| RF | 'n_estimators' value of 100, a 'max_depth' value of 110, a 'max_features' values of 2, a 'min_samples_leaf' value of 3, and a 'min_samples_split' value of 10. |
| XG | 'n_estimators' value of 220, a 'learning_rate' value of 0.05, and 'max_depth' values of 2. |

Table 14: Optimal Hyperparameters for Models

Table 15 presents recall, precision, F1 Score, and accuracy for the classifiers. The maximum accuracy is achieved by the Logistic Regression classifier with 90%. Logistic Regression classifier also has highest F1 score 90% both for the classes '0' and '1'. The highest recall for the class '1' is 88%, which both SVM and Logistic Regression classifiers have. Both the Logistic Regression and the ANN holds the maximum precision in class '1' and maximum recall in class '0' which are both 93%. The maximum precision in class '0' which is 87% is shared by SVM and LR classifiers. From the comparison of different classifiers, we conclude that SVM and custom ANN classifiers showed overall good accuracy. However, Logistic Regression achieved the best accuracy which is 90%.

| **Class** | **Metrics** | **SVM** | **LR** | **k-NN** | **GNB** | **BNB** | **DT** | **RF** | **GB** | **XG** | **ANN** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Precision | 0.87 | 0.87 | 0.83 | 0.81 | 0.79 | 0.74 | 0.81 | 0.84 | 0.84 | 0.84 |
| | Recall | 0.90 | 0.93 | 0.83 | 0.90 | 0.90 | 0.86 | 0.90 | 0.90 | 0.90 | 0.93 |
| | F1 Score | 0.88 | 0.90 | 0.83 | 0.85 | 0.84 | 0.79 | 0.85 | 0.87 | 0.87 | 0.89 |
| 1 | Precision | 0.90 | 0.93 | 0.84 | 0.89 | 0.85 | 0.89 | 0.90 | 0.90 | 0.90 | 0.93 |
| | Recall | 0.88 | 0.88 | 0.84 | 0.81 | 0.78 | 0.81 | 0.84 | 0.84 | 0.84 | 0.84 |
| | F1 Score | 0.89 | 0.90 | 0.84 | 0.85 | 0.82 | 0.85 | 0.87 | 0.87 | 0.87 | 0.89 |
| Accuracy | | 0.89 | 0.90 | 0.83 | 0.85 | 0.83 | 0.78 | 0.85 | 0.87 | 0.87 | 0.89 |

Table 15: Performance Evaluation of The Classifiers

The accuracy of the classifiers before and after hyperparameter tuning is presented in Table 16 From the results, it is clear that most of the classifiers (SVM, k-NN, GNG, RF, XG) improved their accuracy with hyperparameter

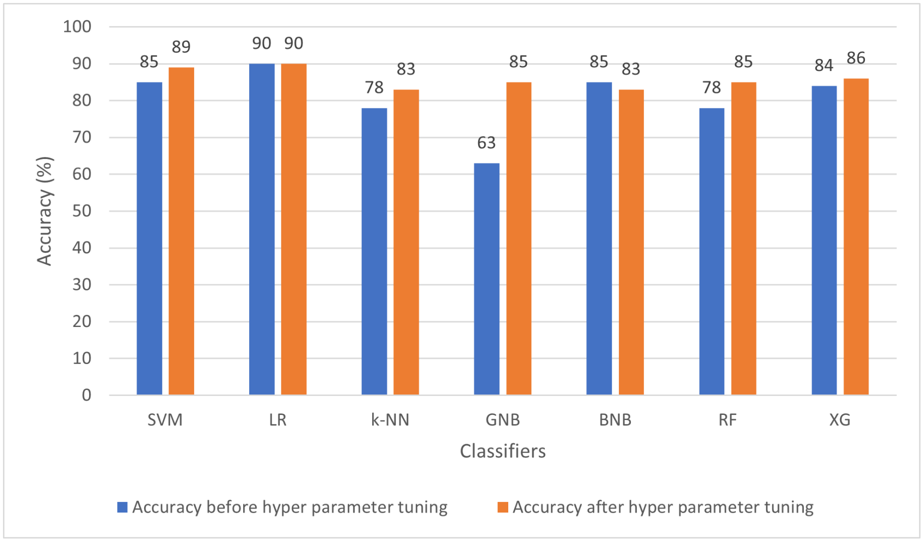


Table 16: Accuracy Change After Hyperparameter Tuning

tuning, while the accuracy of Logistic Regression was not changed, and the accuracy of BNB was a bit decreased.

The accuracy of the classifiers before and after the standardization of the data is presented in Table 17. It can be seen that most of the classifiers' (SVM, BNB, RF, ANN) accuracy increased after the data was standardized, while the accuracy of GNB was decreased, and the accuracy of DT, GB, XG classifiers nearly remained unchanged. The SVM classifier and ANN model showed a significant increase in terms of accuracy after the standardization. By comparing results obtained before and after standardization, it is clear that the standardization mostly has a positive impact on the accuracy, and some classifiers show an accuracy improvement of up to 29%, which is a huge performance improvement.

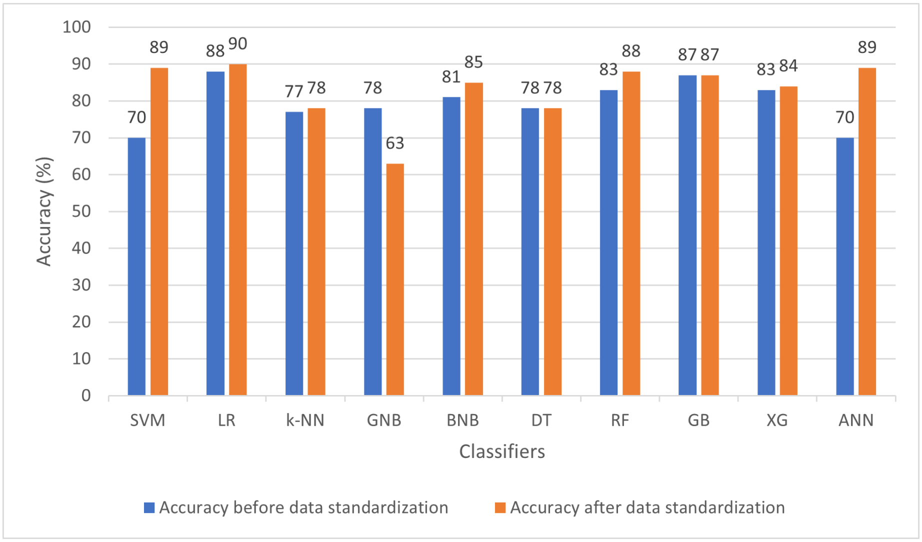


Table 17: Accuracy Change After Standardization

## 5 Conclusion and Future Work

Our study's initial objective was to provide a comprehensive analysis regarding prediction of heart disease using a wide range of machine learning algorithms. Unlike previous research, which frequently focused on individual algorithms, our work aimed to compare a variety of models, ranging from fundamental approaches such as Logistic Regression and Support Vector Machines to advanced techniques such as Artificial Neural Networks (ANN).

Initially, we focused on fully understanding the features in our dataset. This resulted in the development of efficient data preprocessing methods, such as feature scaling and encoding, which optimized the data for model training. We also performed extensive exploratory data analysis to gain a better understanding of data characteristics, using graphs and other visual tools to highlight patterns and distributions.

The performance of various machine learning algorithms on the same dataset was critically compared in accordance with our main goal. To improve the performance of these models, hyperparameter tuning and data standardization were implemented. We also built and evaluated a custom-designed ANN model, comparing its efficiency to more typical algorithms.

Our findings provide useful information and may serve as a practical guide for both healthcare practitioners and data scientists interested in using machine learning to predict heart disease. Our research not only highlights the predictive capabilities of various machine learning models, but it also demonstrates the significance of each step in the analytical pipeline, from initial data understanding to final model evaluation.

Our primary objective for the future is to explore the capabilities of Artificial Neural Networks (ANNs) with more complex architectures. The current model used in our research is simple, but ANN architectures can be much more complex, with multiple layers, different activation functions, and different types of neurons. Incorporating these elements can improve the model's ability to capture non-linear relationships and complex patterns in the data. Developing an effective evaluation strategy becomes critical as we implement these more advanced techniques. In addition to standard metrics such as accuracy and F1 score, we may investigate using the area under the ROC curve [DG06], which is known as AUC-ROC [Nar18]. In addition, the created models may be uploaded to a server to provide doctors with support and assistance in diagnosing cardiovascular diseases via an application.

## References

[Abr05] Ajith Abraham. Artificial neural networks. *Handbook of measuring system design*, 2005.

[AGG$^+$12] Davide Anguita, Luca Ghelardoni, Alessandro Ghio, Luca Oneto, Sandro Ridella, et al. The'k'in k-fold cross validation. In *ESANN*, volume 102, pages 441–446, 2012.

[AHRD23] Febri Aldi, Febri Hadi, Nadya Alinda Rahmi, and Sarjon Defit. Standardscaler's potential in enhancing breast cancer accuracy using machine learning. *Journal of Applied Engineering and Technological Science (JAETS)*, 5(1):401–413, Dec. 2023.

[Ala22] R. Alanazi. Identification and prediction of chronic diseases using machine learning approach. *Journal of Healthcare Engineering*, 2022:1–9, 2022.

[ALPM$^+$13] Filippo Amato, Alberto López, Eladia María Peña-Méndez, Petr Vaňhara, Aleš Hampl, and Josef Havel. Artificial neural networks in medical diagnosis, 2013.

[APA$^+$21] Md Mamun Ali, Bikash Kumar Paul, Kawsar Ahmed, Francis M. Bui, Julian M.W. Quinn, and Mohammad Ali Moni. Heart disease prediction using supervised machine learning algorithms: Performance analysis and comparison. *Computers in Biology and Medicine*, 136:104672, 2021.

[BCMM21] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54(3):1937–1967, 2021.

[BD16] Mariana Belgiu and Lucian Drăguţ. Random forest in remote sensing: A review of applications and future directions. *ISPRS Journal of Photogrammetry and Remote Sensing*, 114:24–31, 2016.

[Ber19] Daniel Berrar. Bayes' theorem and naive bayes classifier., 2019.

[BH00] I.A Basheer and M Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3–31, 2000. Neural Computting in Micrbiology.

[Bis19] Ekaba Bisong. *Matplotlib and Seaborn*, pages 151–165. Apress, Berkeley, CA, 2019.

[CA21] Bahzad Charbuty and Adnan Abdulazeez. Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, 2(01):20 – 28, Mar. 2021.

[CD21] Pádraig Cunningham and Sarah Jane Delany. k-nearest neighbour classifiers - a tutorial. *ACM Comput. Surv.*, 54(6), jul 2021.

[CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[CHB$^+$15] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4, 2015.

[CNM06] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 161–168, New York, NY, USA, 2006. Association for Computing Machinery.

[DG06] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 233–240, New York, NY, USA, 2006. Association for Computing Machinery.

[DOM02] Stephan Dreiseitl and Lucila Ohno-Machado. Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics*, 35(5):352–359, 2002.

[DSC22] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022.

[DSHSAF$^+$17] Ivan Nunes Da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, Silas Franco dos Reis Alves, Ivan Nunes da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, and Silas Franco dos Reis Alves. *Artificial neural network architectures and training processes*. Springer, 2017.

[Dwi18] Ashok Kumar Dwivedi. Performance evaluation of different machine learning techniques for prediction of heart disease. *Neural Computing and Applications*, 29(10):685–693, 2018.

[ELBK$^+$21] A. A. Abd El-Latif, R. Bharti, A. Khamparia, M. Shabaz, G. Dhiman, S. Pande, and P. Singh. Prediction of heart disease using a combination of machine learning and deep learning. *Computational Intelligence and Neuroscience*, 2021:8387680, 2021.

[ENM15] Issam El Naqa and Martin J. Murphy. *What Is Machine Learning?*, pages 3–11. Springer International Publishing, Cham, 2015.

[Far19] Muhammad Farrukh. Modeling on feature vectors in compressed spaces by the use of neural network techniques, 03 2019.

[GAK$^+$21] Pronab Ghosh, Sami Azam, Asif Karim, Mehedi Hassan, Kuber Roy, and Mirjam Jonkman. A comparative study of different machine learning tools in detecting diabetes. *Procedia Computer Science*, 192:467–477, 2021. Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 25th International Conference KES2021.

[Gel04] Andrew Gelman. Exploratory data analysis for complex models. *Journal of Computational and Graphical Statistics*, 13(4):755–779, 2004.

[Gra13] Daniel Graupe. *Principles of artificial neural networks*, volume 7. World Scientific, 2013.

[Hal99] Mark A Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[HJLS13] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression.* John Wiley & Sons, 2013.

[IWASA07] Mohammed J. Islam, Q. M. Jonathan Wu, Majid Ahmadi, and Maher A. Sid-Ahmed. Investigating the performance of naive-bayes classifiers and k- nearest neighbor classifiers. In *2007 International Conference on Convergence Information Technology (ICCIT 2007)*, pages 1541–1546, 2007.

[JD88] Steinbrunn William-Pfisterer Matthias Janosi, Andras and Robert Detrano. Heart Disease. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C52P4X.

[JJKS20] Siva Kumar Jonnavithula, Abhilash Kumar Jha, Modepalli Kavitha, and Singaraju Srinivasulu. Role of machine learning algorithms over heart diseases prediction. *AIP Conference Proceedings*, 2292(1):040013, 10 2020.

[KKA$^+$22] Deepika Koundal, K. Karthick, S. K. Aruna, Ravi Samikannu, Ramya Kuppusamy, Yuvaraja Teekaraman, and Amruth Ramesh Thelkar. [retracted] implementation of a heart disease risk prediction model using machine learning. *Computational and Mathematical Methods in Medicine*, 2022:6517716, 2022. This article has been retracted.

[KS08] Carl Kingsford and Steven L Salzberg. What are decision trees? *Nature Biotechnology*, 26(9):1011–1013, 2008.

[KV19] R. Kannan and V. Vasanthi. *Machine Learning Algorithms with ROC Curve for Predicting and Diagnosing the Heart Disease*, pages 63–72. Springer Singapore, Singapore, 2019.

[M$^+$06] Kevin P Murphy et al. Naive bayes classifiers. *University of British Columbia*, 18(60):1–8, 2006.

[Man20] B. Manesh. Machine learning algorithms - a review. *International Journal of Science and Research (IJSR)*, 9(1):381–386, 2020.

[Mat] Matplotlib Development Team. Matplotlib — visualization with python. Matplotlib Official Website.

[MPN$^+$11] Shanthi Mendis, Pekka Puska, B editors Norrving, World Health Organization, et al. *Global atlas on cardiovascular disease prevention and control.* World Health Organization, 2011.

[MTC09] Alessia Mammone, Marco Turchi, and Nello Cristianini. Support vector machines. *WIREs Computational Statistics*, 1(3):283–289, 2009.

[Nar18] Sarang Narkhede. Understanding auc-roc curve. *Towards data science*, 26(1):220–227, 2018.

[NK13] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013.

[Pal05] M. Pal. Random forest classifier for remote sensing classification. *International Journal of Remote Sensing*, 26(1):217–222, 2005.

[PP18] Harsh H Patel and Purvi Prajapati. Study and analysis of decision tree based classification algorithms. *International Journal of Computer Sciences and Engineering*, 6(10):74–78, 2018.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PWS+20] Heny Pratiwi, Agus Perdana Windarto, S. Susliansyah, Ririn Restu Aria, Susi Susilowati, Luci Kanti Rahayu, Yuni Fitriani, Agustiena Merdekawati, and Indra Riyana Rahadjeng. Sigmoid activation function in selecting the best model of artificial neural networks. *Journal of Physics: Conference Series*, 1471(1):012010, feb 2020.

[R+01] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. Citeseer, 2001.

[Rah21] R. Rahman. Heart attack analysis & prediction dataset. Kaggle, March 2021.

[RKVR19] G S K Ranjan, Amar Kumar Verma, and Sudha Radhika. K-nearest neighbors and grid search cv based real time fault monitoring system for industries. In *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, pages 1–5, 2019.

[RPL10] Juan D. Rodriguez, Aritz Perez, and Jose A. Lozano. Sensitivity analysis of k-fold cross validation in prediction error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(3):569–575, 2010.

[RSU+22] H. T. Rauf, A. Saboor, M. Usman, S. Ali, A. Samad, M. F. Abrar, and N. Ullah. A method for improving prediction of human heart disease using machine learning algorithms. *Mobile Information Systems*, 2022:1410169, 2022.

[Sar21] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN computer science*, 2(3):160, 2021.

[Seg18] Cedric Seger. An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing, 2018.

[SGN20] Prateek P. Sengar, Mihir J. Gaikwad, and Ashlesha S. Nagdive. Comparative study of machine learning algorithms for breast cancer prediction. In *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 796–801, 2020.

[SK16] Himani Sharma and Sunil Kumar. A survey on decision tree algorithms of classification in data mining. *International Journal of Science and Research (IJSR)*, 5(4):2094–2097, 2016.

[SK19] Prashanth Subramaniam and Maninder Jeet Kaur. Review of security in mobile edge computing with deep learning. In *2019 Advances in Science and Engineering Technology International Conferences (ASET)*, pages 1–5, 2019.

[SLDS19] Sumedh Anand Sontakke, Jay Lohokare, Reshul Dani, and Pranav Shivagaje. Classification of cardiotocography signals using machine learning. In Kohei Arai, Supriya Kapoor, and Rahul Bhatia, editors, *Intelligent Systems and Applications*, pages 439–450, Cham, 2019. Springer International Publishing.

[SSA17] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[Sut16] Shan Suthaharan. *Support Vector Machine*, pages 207–235. Springer US, Boston, MA, 2016.

[SV21] Patrick Schober and Thomas R Vetter. Logistic regression in medical research. *Anesthesia & Analgesia*, 132(2):365–366, 2021.

[Sza21] Tomasz Szandała. *Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks*, pages 203–224. Springer Singapore, Singapore, 2021.

[TLM$^+$22] Ramesh TR, Umesh Kumar Lilhore, Poongodi M, Sarita Simaiya, Amandeep Kaur, and Mounir Hamdi. Predictive analysis of heart diseases with machine learning approaches. *Malaysian Journal of Computer Science*, page 132–148, Mar. 2022.

[TYW$^+$21] Jimin Tan, Jianan Yang, Sai Wu, Gang Chen, and Jake Zhao. A critical look at the current train/test split in machine learning, 2021.

[TZ22] Yingjie Tian and Yuqi Zhang. A comprehensive survey on regularization strategies in machine learning. *Information Fusion*, 80:146–166, 2022.

[UHL$^+$22] Shahadat Uddin, Ibtisham Haque, Haohui Lu, Mohammad Ali Moni, and Ergun Gide. Comparative performance analysis of k-nearest neighbour (knn) algorithm and its different variants for disease prediction. *Scientific Reports*, 12(1):6256, 2022.

[V$^+$21] Ž Vujović et al. Classification model evaluation metrics. *International Journal of Advanced Computer Science and Applications*, 12(6):599–606, 2021.

[WF18] Yu-chen Wu and Jun-wen Feng. Development and application of artificial neural network. *Wireless Personal Communications*, 102:1645–1656, 2018.

[WMV20] Hilde J. P. Weerts, Andreas C. Mueller, and Joaquin Vanschoren. Importance of tuning hyperparameters of machine learning algorithms, 2020.

[Wor22] World Health Organization. Cardiovascular diseases. World Health Organization, 2022.

[WY20] Tzu-Tsung Wong and Po-Yang Yeh. Reliable accuracy estimates from k-fold cross validation. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1586–1594, 2020.

[YS20] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[ZGCH21] Jianlong Zhou, Amir H Gandomi, Fang Chen, and Andreas Holzinger. Evaluating the quality of machine learning explanations: A survey on methods and metrics. *Electronics*, 10(5):593, 2021.

[Zha18] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–2, 2018.

[ZHS09] Jinming Zou, Yi Han, and Sung-Sau So. *Overview of Artificial Neural Networks*, pages 14–22. Humana Press, Totowa, NJ, 2009.

[ZLV+21] Tao Zhang, Wuyin Lin, Andrew M. Vogelmann, Minghua Zhang, Shaocheng Xie, Yi Qin, and Jean-Christophe Golaz. Improving convection trigger functions in deep convective parameterization schemes using machine learning. *Journal of Advances in Modeling Earth Systems*, 13(5):e2020MS002365, 2021. e2020MS002365 2020MS002365.

[ZS18] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.