Heart Failure Classification Problem

Pattern Recognition Lab

March 14, 2025

Name	ID
Amin Mohamed Amin	21010310
Rafy Hany Said	21010504
Omar Aldawy	21010864

Contents

1 Introduction

Heart failure is a severe condition caused primarily by cardiovascular disease (CVD), affecting millions worldwide. It occurs when the heart struggles to pump blood effectively, leading to inadequate oxygen and nutrient circulation. Without early detection, heart failure can reduce quality of life and increase mortality. However, timely intervention improves outcomes through better management and treatment.

Machine learning has revolutionized medical diagnostics by identifying patterns in large datasets and aiding in early disease prediction. This study evaluates multiple machine learning models for heart failure prediction using the Kaggle Heart Failure Prediction dataset, which contains **918 patient records** and **11 key clinical features** such as age, blood pressure, and cholesterol levels. The models will be assessed based on accuracy and effectiveness in aiding early diagnosis.

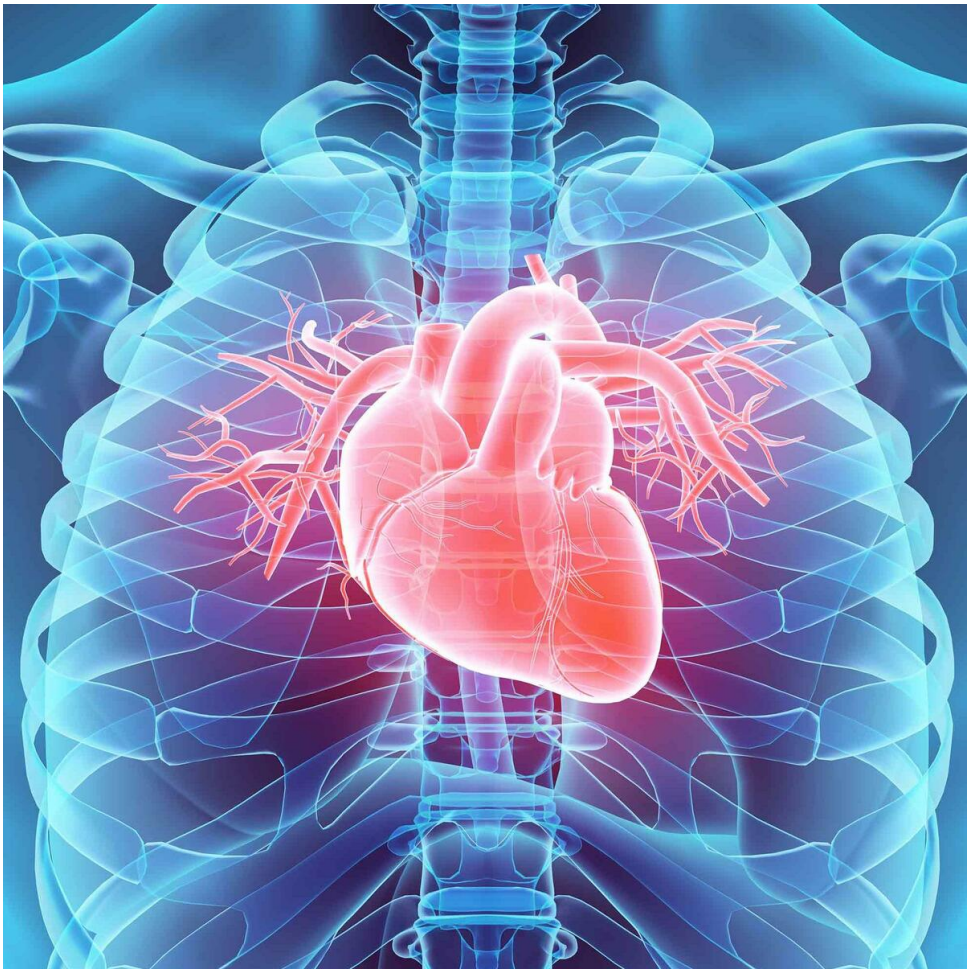

Figure 1: Illustration of heart failure.

To comprehensively analyze and compare the effectiveness of various models in medical diagnostics, we will take the following approach:

- **Data Preprocessing:** We will perform necessary preprocessing steps, normalizing numerical features, and encoding categorical variables to ensure the dataset is suitable for machine learning models.

- **Model Implementation:** Some models will be implemented from scratch to gain a deeper understanding of their inner workings, while others will be built using established machine learning libraries such as *scikit-learn* and *PyTorch*.
- **Performance Evaluation:** The models will be evaluated using key metrics such as accuracy, precision, recall, and F1-score to determine their effectiveness in predicting heart failure.
- **Comparison and Analysis:** We will compare the performance of different classifiers to identify the most reliable and accurate models for heart failure prediction.

2 Background

Cardiovascular diseases (CVDs) are the leading cause of death worldwide, with heart failure being a major concern. Traditional diagnostics can be time consuming and may miss early signs. Machine learning (ML) offers a way to improve accuracy by identifying hidden patterns in patient data.

Using the Heart Failure Prediction dataset, which includes clinical features such as age, blood pressure, and cholesterol, we will train ML models to assess the risk of heart failure. Techniques such as logistic regression, decision trees, and neural networks will be explored to evaluate their predictive effectiveness.

This project improves our understanding of ML classifiers and their potential in predictive healthcare, aiding early diagnosis and treatment planning.

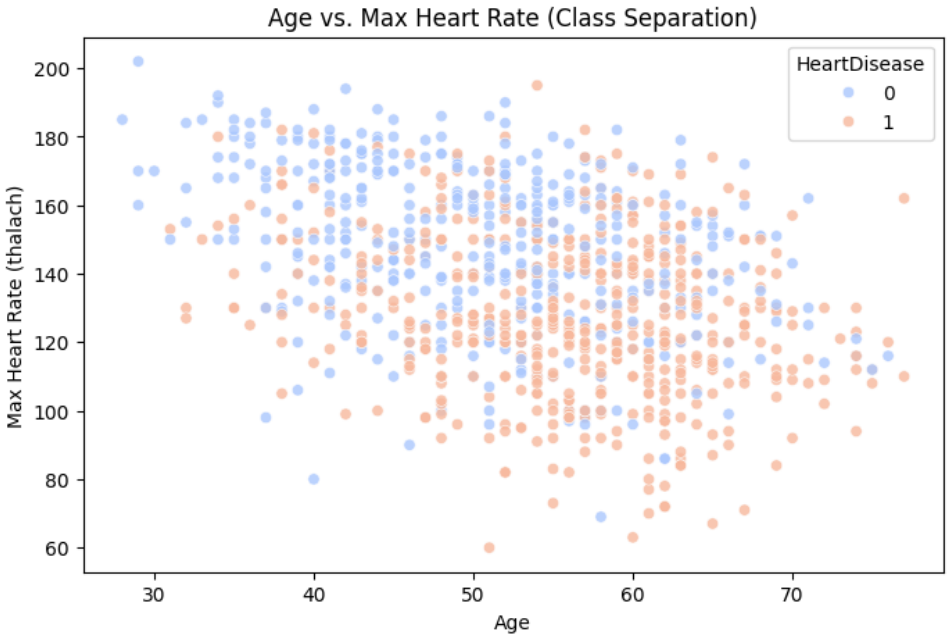

Figure 2: Illustration of Age vs. Heart Failure

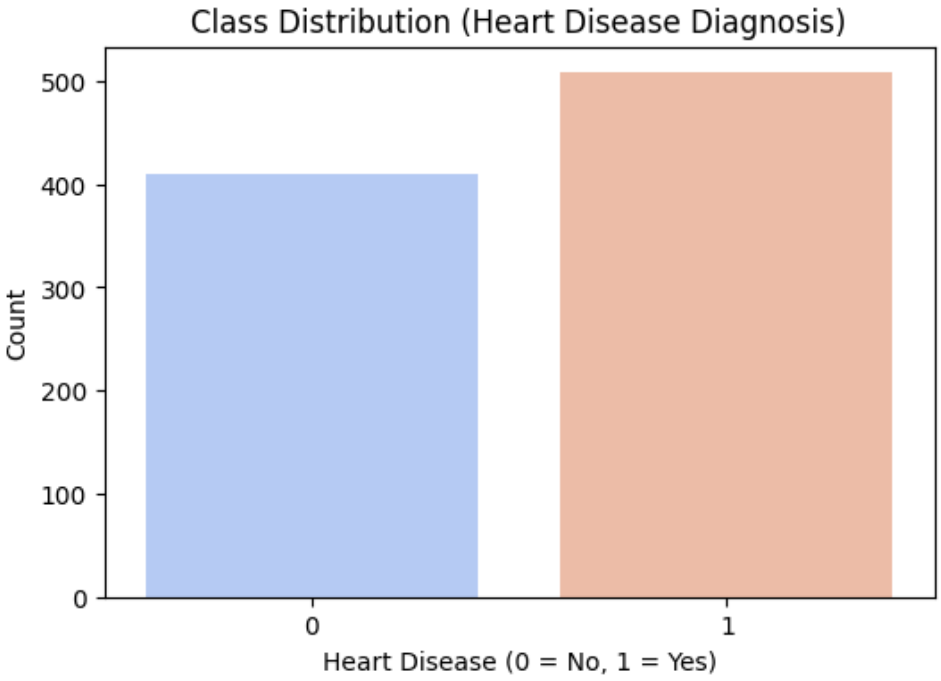

Figure 3: Illustration for probability distribution.

3 Data Preparation

3.1 Fixed Random Seed

To ensure reproducibility, a fixed random seed of 42 was used for all random operations. This guarantees that dataset splitting and model training produce the same results across different runs.

3.2 Train/Validation/Test Split

The dataset was split into three subsets:

- **70% Training Set**: Used to train the models.
- **10% Validation Set**: Used for hyperparameter tuning and model selection.
- **20% Test Set**: Used for the final model evaluation.

A stratified splitting technique was used to maintain the class distribution in all subsets. This is crucial for binary classification problems where an imbalance in class representation may impact model performance.

3.3 Feature Preprocessing

- **Numerical Features**: Since most features are numerical, they were checked for missing values and outliers. Normalization or standardization was applied where necessary. Standardization was particularly useful for models such as K-Nearest Neighbors (KNN) and Neural Networks, which rely on distance-based calculations.

- **Categorical Features**: If any categorical features were present, they were encoded using one-hot encoding to ensure compatibility with machine learning algorithms.

3.4 Purpose of the Validation Set

The validation set was used exclusively for hyperparameter tuning and model selection. The test set remained untouched during training and validation to ensure an unbiased evaluation of model performance. This approach prevents data leakage and provides a reliable estimate of how well the final model generalizes to unseen data.

3.5 Conclusion

By following the above preprocessing steps, the dataset was prepared for machine learning models while maintaining reproducibility, fairness in class distribution, and effective feature representation.

4 Algorithms and Anaylsis

we have used 6 different algorithms for classification.

4.1 Decision Tree

A decision tree is a tree-like model used for decision-making and classification tasks. It consists of nodes representing features, branches representing decisions, and leaf nodes representing outcomes or class labels.

1. **Tree Structure**
   - A decision tree consists of a root node, internal nodes (decision nodes), and leaf nodes (terminal nodes).
   - Each internal node splits based on a feature, guiding data points to child nodes.
2. **Splitting Criterion**
   - Decision trees use different criteria for splitting:
     - **Gini Impurity**: Measures node impurity.

$$G = 1 - \sum_{i=1}^{c} p_i^2$$

       where p_i is the probability of class i.
     - **Entropy**: Measures dataset uncertainty.

$$H = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

     - **Information Gain (used in our model)**: Reduction in entropy after splitting.

$$IG(Y|X) = H(Y) - H(Y|X)$$

3. **Pruning**

   - Pruning prevents overfitting by removing less significant branches.
   - Types of pruning:
     - **Pre-pruning**: Stops growth early based on criteria like depth or impurity threshold.
     - **Post-pruning (Not covered in our model)**: Grows the tree fully and then removes branches that do not improve performance.

4. **Prediction**

   - Traverse the tree from the root node to a leaf node based on feature values and thresholds in internal nodes.
   - The leaf node determines the predicted class.

5. **Complexity**

   - The depth of the tree affects performance:
     - Shallow trees may underfit.
     - Deep trees may overfit.
   - Time Complexity:
     - Training: $O(n^2)$ in the worst case, where n is the number of samples in the dataset.
     - Prediction: $O(d)$, where d is the depth of the tree.

6. **Training a Decision Tree**

   The training process for a decision tree involves recursively splitting the dataset based on feature values to maximize information gain (or another splitting criterion like Gini impurity). The steps are as follows:

   (a) **Define the Problem** Given a dataset D with n samples and d features:

   $$D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$$

   where:

   - x_i is a feature vector $(x_{i1}, x_{i2}, ..., x_{id})$.
   - y_i is the target label (classification or regression).

   (b) **Choose a Splitting Criterion** The goal is to select the best feature to split on using an impurity measure.

   - **For Classification**:
     - **Entropy and Information Gain**:

     $$IG(Y|X) = H(Y) - H(Y|X)$$

     where $H(Y)$ is the entropy of the parent node and $H(Y|X)$ is the weighted entropy after the split.

– **Gini Impurity** (used in CART):

$$Gini(D) = 1 - \sum_c p_c^2$$

where p_c is the probability of class c.

- **For Regression**:
  – **Mean Squared Error (MSE)**:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \bar{y})^2$$

(c) **Find the Best Split** For each feature:

i. Sort the dataset by that feature (for numerical features).
ii. Evaluate all possible split points (midpoints between sorted values).
iii. Compute the impurity measure (Entropy, Gini, or MSE) before and after the split.
iv. Select the split that maximizes information gain (or minimizes impurity).

(d) **Split the Dataset** Once the best split is found, partition the dataset into two or more subsets. Assign each subset to a child node.

(e) **Repeat Recursively** Apply Steps 2–4 on each child node until a stopping condition is met.

(f) **Stopping Conditions** The recursion stops when:

- **Pure Nodes**: All samples in a node belong to the same class.
- **Maximum Depth Reached**: A predefined depth limit is met.
- **Minimum Samples per Leaf**: If a node has fewer than a set number of samples.
- **No Information Gain**: If splitting does not improve the prediction.

7. **Hyperparameter Tuning for Decision Trees**

To optimize decision tree performance, we tune hyperparameters like max depth and minimum sample splits. The steps are:

(a) **Define the Search Space**:

- Choose a range of values for:
  – **Max Depth (`max_depth`)**: Limits the depth of the tree to prevent overfitting.
  – **Min Samples Split (`min_samples_split`)**: The minimum number of samples required to split an internal node.

(b) **Split the Data into Training, Validation, and Test Sets**:

- Divide the dataset into three sets:
  – **Training Set**: Used to train the model.
  – **Validation Set**: Used to tune hyperparameters and prevent overfitting.

– **Test Set**: Used for final model evaluation.

(c) **Train the Model on the Training Set**:

- Fit the decision tree using the training data.
- Use an initial set of hyperparameters.

(d) **Evaluate on the Validation Set**:

- Compute performance metrics (e.g., accuracy, F1-score, or MSE).
- Adjust hyperparameters to balance bias and variance.

(e) **Select the Best Hyperparameters**:

- Choose the hyperparameters that achieve the best validation performance while avoiding overfitting.

(f) **Test on the Final Test Set**:

- Evaluate the final model on the test dataset to check its generalization performance.
- Ensure that no hyperparameter tuning is performed on the test set.

8. **Advantages and Disadvantages of Decision Trees**

(a) **Advantages**

i. Easy to Understand & Interpret
ii. Requires Minimal Data Preprocessing
iii. Handles Non-linearity Well
iv. Feature Selection is Built-in
v. Fast Training & Prediction
vi. Can Handle Missing Values
vii. Works for Both Classification and Regression

(b) **Disadvantages**

i. Prone to Overfitting
ii. Unstable (Sensitive to Small Variations in Data)
iii. Greedy Splitting May Not Find the Best Tree
iv. High Computational Cost for Large Datasets
v. Biased with Imbalanced Data
vi. Not Always the Best for Continuous Variables

9. **Evaluation**

To evaluate the performance of Decision-Tree, we compute accuracy and the F1-score using a test dataset.

Metric	**Value**
Accuracy	80.98%
F1-score	0.8223

Table 1: Performance Metrics for Decision-Tree

The confusion matrix provides insight into the classification performance.

	Predicted Positive	Predicted Negative
Actual Positive	81	21
Actual Negative	14	68

Table 2: Confusion Matrix for Decision-Tree

4.2 Bagging

Bagging, short for **Bootstrap Aggregating**, is an ensemble learning technique used to improve the stability and accuracy of machine learning algorithms. It works by combining the predictions of multiple models trained on different subsets of the training data.

1. **Bootstrap Sampling**:
   - From the original training dataset D with N samples, create M new datasets $D_1, D_2, \ldots, D_M$ by randomly sampling N samples *with replacement*.
   - Each dataset D_i may contain duplicate samples and may omit some samples from the original dataset.
2. **Model Training**:
   - Train a separate base model h_i on each bootstrap sample D_i.
   - The base models are typically weak learners (e.g., decision trees).
3. **Aggregation**:
   - Combine the predictions of all M models to produce the final prediction.
   - For **classification**, use majority voting(in binary classification):

$$H(x) = \frac{1}{M} \sum_{i=1}^{M} h_i(x)$$

   - For **regression**, use averaging:

$$\hat{y} = \frac{1}{M} \sum_{i=1}^{M} h_i(x)$$

Advantages of Bagging:

- **Reduces Variance**: By averaging multiple models, bagging reduces the variance of the predictions, making the model more robust to overfitting.

$$\text{Var}\left(\frac{1}{m} \sum_{i=1}^{m} h_i\right) = \frac{1}{m}(1-\rho)\sigma^2 + \rho\sigma^2.$$

- **Improves Stability**: Small changes in the training data have less impact on the final model.
- **Parallelizable**: Each model can be trained independently, making bagging suitable for parallel processing.

Disadvantages of Bagging:

- **Computationally Expensive**: Training multiple models increases computational cost.
- **Less Interpretable**: The final ensemble model is harder to interpret compared to a single model.
- **Affect on bias**: Bagging doesn't decrease the bias of the overall model.

Hyper Parameters tuning

- number of classifiers = 20

To evaluate the performance of Bagging, we compute accuracy and the F1-score using a test dataset.

Metric	**Value**
Accuracy	84.24%
F1-score	.8612

Table 3: Performance Metrics for Bagging

The confusion matrix provides insight into the classification performance.

	Predicted Positive	**Predicted Negative**
Actual Positive	90	12
Actual Negative	17	65

Table 4: Confusion Matrix for Bagging

4.3 Adaboost

Adaptive Boosting (AdaBoost) is a popular ensemble learning technique that combines multiple weak classifiers to create a strong classifier. The algorithm iteratively adjusts the weights of misclassified samples to improve subsequent classifiers.

Given a training dataset:

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$$

where $x_i \in R^d$ represents feature vectors, and $y_i \in \{-1, 1\}$ represents class labels, AdaBoost works as follows:

1. **Initialize weights:** Assign equal weights to all samples:

$$w_i^{(1)} = \frac{1}{n}, \quad \forall i = 1, \ldots, n.$$

2. **For each iteration** $t = 1, 2, ..., T$**:**

(a) Train a weak classifier $h_t(x)$ using the weighted dataset.

(b) Compute the weighted error:

$$\epsilon_t = \frac{\sum_{i=1}^{n} w_i^{(t)} I(h_t(x_i) \neq y_i)}{\sum_{i=1}^{n} w_i^{(t)}}$$

where $I(\cdot)$ is the indicator function.

(c) Compute the classifier weight:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right).$$

(d) Update the sample weights:

$$w_i^{(t+1)} = w_i^{(t)} \exp(-\alpha_t y_i h_t(x_i)).$$

(e) Normalize the weights:

$$w_i^{(t+1)} = \frac{w_i^{(t+1)}}{\sum_{j=1}^{n} w_j^{(t+1)}}.$$

3. **Final classifier:** The final strong classifier is given by:

$$H(x) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right).$$

Advantages of Boosting:

- **Reduces Bias**: Boosting corrects the mistakes of weak learners, reducing bias, leading to a more accurate model.
- **Improves Predictive Performance**: Boosting often results in higher accuracy than bagging due to its sequential learning approach.
- **Feature Importance**: Many boosting algorithms provide feature importance scores, aiding in feature selection and interpretability.

Disadvantages of Boosting:

- **Prone to Overfitting**: If not carefully tuned, boosting can overfit the training data, especially on noisy datasets.
- **Computationally Intensive**: Training is sequential, making boosting more time-consuming and less parallelization than bagging.
- **Sensitive to Noisy Data**: Boosting gives higher weights to misclassified points, which can amplify the effect of outliers.

To evaluate the performance of AdaBoost, we compute accuracy and the F1-score using a test dataset.

The confusion matrix provides insight into the classification performance.

AdaBoost is an effective boost algorithm that improves classification performance by iteratively combining weak learners. Despite its sensitivity to noise, it remains widely used in machine learning applications.

Metric	Value
Accuracy	88.04%
F1-score	0.8911

Table 5: Performance Metrics for AdaBoost

	Predicted Positive	Predicted Negative
Actual Positive	90	12
Actual Negative	10	72

Table 6: Confusion Matrix for AdaBoost

5 Bouns Algorithms

5.1 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, non-parametric, and instance-based machine learning algorithm used for both **classification** and **regression**. It works by finding the k closest data points (neighbors) in the feature space and making predictions based on their labels or values.

1. **Choose the Number of Neighbors (k)**:
   - Select an odd integer k (e.g., 3, 5, 7) to determine the number of neighbors to consider.
2. **Compute Distances**:
   - For a given test point x, compute the distance to all training points x_i in the dataset.
   - Common distance metrics include:
     - **Euclidean Distance**:
     $$d(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2}$$
     - **Manhattan Distance**:
     $$d(x, x_i) = \sum_{j=1}^{n} |x_j - x_{ij}|$$
     - **minkowski Distance**:
     $$d(x, x_i) = \sqrt[3]{\sum_{j=1}^{n} (x_j - x_{ij})^3}$$
3. **Select the k Nearest Neighbors**:
   - Identify the k training points with the smallest distances to the test point x.

4. **Make a Prediction**:
    - For **classification**:
        - Use majority voting among the k neighbors to assign the class

Advantages of KNN:

- **Simple and Intuitive**: Easy to understand and implement.
- **No Training Phase**: KNN is a lazy learner; it does not require a training phase.
- **Non-Parametric**: Makes no assumptions about the underlying data distribution.

Disadvantages of KNN:

- **Computationally Expensive**: Requires computing distances to all training points for each prediction.
- **Sensitive to Noise and Outliers**: Noisy or irrelevant features can degrade performance.
- **Choice of k**: The performance of KNN depends heavily on the choice of k.

Hyper Parameters tuning

- k = 3
- metric = manhattan

To evaluate the performance of K-nn, we compute accuracy and the F1-score using a test dataset.

Metric	**Value**
Accuracy	89.67%
F1-score	0.9082

Table 7: Performance Metrics for Knn

The confusion matrix provides insight into the classification performance.

	Predicted Positive	**Predicted Negative**
Actual Positive	94	8
Actual Negative	11	71

Table 8: Confusion Matrix for K-nn

5.2 Logistic Regression

Logistic regression is a statistical model used for binary and multi-class classification. It estimates the probability that a given input belongs to a specific class using the logistic (sigmoid) function.

1. **Mathematical Formulation**

   - Logistic regression models the probability of a class label y given input features X as:

     $$P(y = 1|X) = \frac{1}{1 + e^{-(wX+b)}}$$

     where:

     - w represents the weight coefficients.
     - b is the bias term.
     - The function outputs values between 0 and 1, which are interpreted as probabilities.

   - The decision boundary is defined as:

     $$y = \begin{cases} 1, & P(y = 1|X) \geq 0.5 \\ 0, & P(y = 1|X) < 0.5 \end{cases}$$

2. **Regularization**

   - To prevent overfitting, logistic regression applies regularization:
     - **L1 Regularization (Lasso)**: Encourages sparsity by shrinking some coefficients to zero.

       $$J(w) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) + \lambda \sum |w|$$

     - **L2 Regularization (Ridge)**: Penalizes large coefficients to improve generalization.

       $$J(w) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) + \lambda \sum w^2$$

3. **Solvers**

   - Logistic regression can be optimized using different solvers:
     - **Liblinear**: Good for small datasets, supports L1 and L2 regularization.
     - **SAGA**: Stochastic solver, works well with large datasets and supports L1 and L2.
     - **Newton-CG, SAG, LBFGS**: Support only L2 regularization and are efficient for large datasets.

4. **Hyperparameter Tuning**

   - We performed hyperparameter tuning to find the best combination of regularization, solver, and maximum iterations.

- The search space included:
  - **Regularization**: L1, L2
  - **Solvers**: `liblinear`, `saga`, `newton-cg`, `sag`, `lbfgs`
  - **Maximum Iterations**: {100, 300, 500, 700, 900, 1100}
- The best combination was selected based on validation accuracy.

5. **Training and Prediction**
   - The best model was trained using L1 regularization with the `liblinear` solver and 100 iterations.
   - Predictions were made on the test dataset.
6. **Evaluation Metrics**
   - We evaluated the model using accuracy, precision, recall, and F1-score.

Metric	**Value**
Accuracy	89.673%
F1-score	0.90909

Table 9: Performance Metrics for Logistic Regression

7. **Confusion Matrix**

	Predicted Positive	**Predicted Negative**
Actual Positive	95	7
Actual Negative	12	70

Table 10: Confusion Matrix for Logistic Regression

5.3 Neural Networks

A Neural Network (NN) is a machine learning model inspired by the human brain. It consists of layers of interconnected neurons that transform input data into meaningful outputs through learning.

A typical neural network consists of three types of layers:

- **Input Layer:** Accepts feature vectors x.
- **Hidden Layers:** Apply transformations through activation functions.
- **Output Layer:** Produces final predictions.

Given an input $x \in R^d$, a neural network with one hidden layer computes:

$$z^{(1)} = W^{(1)}x + b^{(1)}$$

$$a^{(1)} = f(z^{(1)})$$

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)}$$

$$\hat{y} = g(z^{(2)})$$

where:

- $W^{(1)}, W^{(2)}$ are weight matrices.
- $b^{(1)}, b^{(2)}$ are bias vectors.
- $f(\cdot)$ is the activation function (e.g., ReLU, sigmoid).
- $g(\cdot)$ is the output function (e.g., softmax for classification).
- $\hat{y}$ is the predicted output.

Neural networks are trained using backpropagation and optimization techniques like gradient descent:

1. **Initialize:** Set weights W and biases b randomly.
2. **Forward Propagation:** Compute activations through the network.
3. **Loss Computation:** Calculate error using a loss function:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \hat{y}_i).$$

4. **Backward Propagation:** Compute gradients:

$$\frac{\partial \mathcal{L}}{\partial W} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b}$$

   and update parameters using an optimizer like Adam or SGD.

5. **Repeat** until convergence.

Advantages of Neural Networks:

- **Ability to Learn Complex Patterns**: Neural networks can model highly non-linear and intricate relationships in data.
- **Generalization Power**: With sufficient training data and regularization techniques, neural networks can generalize well to unseen data.
- **Automated Feature Extraction**: Deep neural networks can automatically learn hierarchical features, reducing the need for manual feature engineering.
- **Adaptability**: Neural networks can be applied to various domains, such as image processing, NLP, and time-series forecasting.

Disadvantages of Neural Networks:

- **Computationally Expensive**: Training deep neural networks requires significant computational power, often needing GPUs or TPUs.
- **Black Box Nature**: Neural networks lack interpretability, making it difficult to understand how predictions are made.
- **Requires Large Datasets**: Neural networks generally need a large amount of data to achieve good performance and avoid overfitting.
- **Difficult to Tune**: Hyperparameter tuning, architecture selection, and training optimization require significant expertise and experimentation.

The performance of the neural network is evaluated using accuracy and the F1 score.

Metric	Value
Accuracy	91.3%
F1-score	0.9223

Table 11: Performance Metrics for Neural Network

The confusion matrix provides insight into the performance of the classification.

	Predicted Positive	**Predicted Negative**
Actual Positive	95	7
Actual Negative	9	73

Table 12: Confusion Matrix for Neural Network

Neural networks are powerful models capable of learning complex patterns from data. Their performance is influenced by architecture, optimization techniques, and data quality.

6 Results

6.1 Comparison of Algorithms

The following table compares six machine learning algorithms based on accuracy and F1-score:

6.2 Analysis of Results

Key Observations:

- **Neural Networks achieve the best performance** with the highest accuracy (91.3%) and F1-score (0.92), indicating strong generalization.
- **Bagging improves over Decision Trees** by reducing variance, leading to a higher accuracy (84.2%) and F1-score (0.86).
- **AdaBoost outperforms Bagging**, achieving 88.0% accuracy and 0.89 F1-score, demonstrating its effectiveness in refining weak classifiers.

Algorithm	Accuracy (%)	F1-score
Decision Tree	80.9	0.82
Bagging	84.2	0.86
Logistic Regression	89.6	0.91
K-Nearest Neighbors (KNN)	89.6	0.91
AdaBoost	88.0	0.89
Neural Network	91.3	0.92

Table 13: Performance Comparison of Different Algorithms

- **AdaBoost**: over the the main three algorithms (bagging - decision tree) Adaboost is the best in performance of them due to use multiple weak learner to correct its wrong classified samples by giving them more weight.

7 Discussion

After obtaining the results, we analyze and compare the performance of different classifiers to determine their effectiveness in heart failure prediction.

7.1 Best Performing Models

Among all the classifiers, the **Neural Network** achieved the highest accuracy (**91.3%**) and F1-score (**0.92**), followed closely by **Logistic Regression** and **K-Nearest Neighbors (KNN)** with similar performance metrics. The success of the Neural Network is attributed to its ability to learn complex patterns and relationships in the dataset, making it well-suited for medical diagnosis.

7.2 Decision Trees vs. Ensemble Methods

A standalone **Decision Tree** classifier performed relatively poorly, with an accuracy of **80.9%**, highlighting its tendency to overfit the training data. However, applying ensemble methods significantly improved performance:

- **Bagging (Bootstrap Aggregating)** reduced variance and improved stability, achieving **84.2%** accuracy.
- **AdaBoost** further refined the model's performance by focusing on misclassified instances, reaching **88.0%** accuracy.

This demonstrates that ensemble techniques enhance the predictive power of weak learners by combining multiple models.

7.3 Insights and Observations

An analysis of the confusion matrices reveals key misclassification patterns:

- The **Decision Tree** model misclassified a higher number of negative cases, suggesting it struggles with boundary decision-making.

- **KNN and Logistic Regression** provided a good balance between sensitivity and specificity, making them reliable choices.
- **Neural Networks** minimized false negatives, which is crucial in medical diagnosis, as missing a heart failure case can have severe consequences.

8 Conclusion

Overall, ensemble methods significantly enhanced model performance, and Neural Networks emerged as the most effective approach due to their advanced pattern recognition capabilities.