Pattern Recognition Assignment 1

Heart Failure Classification Using Machine Learning

Team Members

Name	Student ID
Hager Ashraf	21011505
Noran Ashraf	21011492
Rana Mohamed	21010528

Overview

Heart failure is a critical condition often resulting from cardiovascular diseases. The early prediction of heart failure can help improve patient outcomes through timely medical intervention. This report explores the application of machine learning classifiers to predict heart failure based on clinical attributes.

1 Dataset and Preparation

1.1 Data Description

- **Clinical data:** includes age, cholesterol levels, resting blood pressure and maximum heart rate.
- **Categorical variables:** sex, chest pain type, resting ECG results, exercise-induced angina and ST slope type.
- **Binary targets:** 0=No heart disease, 1=Heart disease

1.2 Preprocessing Steps

1. Load the heart disease dataset from csv file into IDE using read_csv().
2. Split the features the dataset into features and target using drop().
3. Encode the categorical features using one hot encoding (this technique split the categorical features into columns for each distinct value in the corresponding feature and the values for this columns is 1 to indicate this is the value for feature and 0 otherwise). for this, we will use get_dummy() and take 2 basic parameters: the dataset need to encode (which will be features in our case) and the column names need to encode.
4. Split the encoded dataset into train set (70 %) and validation set (10 %) and test set (20 %) (this will be done in 2 steps, one to split the train dataset then split remaining into validation and test set) using `train_test_split()` and it take parameters: features, target column, test_size (in 1st time it will be 0.3 and 2nd time it will be 2/3), stratify = y (ensures that the split maintains the same class distribution in both sets) and random_state (which will be 42 for reproducibility).

   Note: we check the distribution balance by using value_count() with normalization.

5. (When needed) Normalize the encoded dataset using the StandardScaler.fit_transform() which normalize the feature values by subtract the mean and divide by the standard deviation.

2 Implementation Details

2.1 Decision Tree

- **Decision tree node:** stores feature index, threshold, entropy, sample count, class distribution and left/right children.
- **Decision tree structure:** implements a tree-based model where each node represents a decision based on feature values, leading to predictions at the leaves.
- **Entropy and information gain:** utilizes these metrics to evaluate the quality of splits, aiming to maximize the information gain and reduce uncertainty in predictions.
- **Recursive tree building:** constructs the tree recursively, splitting the data until max depth or minimum samples are met.
- **Prediction mechanism:** traverses the tree to make predictions based on input features.
- **Hyperparameter tuning:** we have two primary hyperparameters to tune, which are the max depth which controls the complexity of the tree and min samples split which regulates the minimum number of samples required to create a split. Balancing these parameters helps balance the complexity and generalization of the model.

2.2 Bagging Ensemble

2.2.1 Without caring about decision tree hyperparameters

1. **Bagging training:** iteratively trains decision tree (one model per time) and each time, resample the train dataset(features + target) using resample() (it already link each sample with its target value) it take parameters: features and target, replace = True (to do correct resample) and random_state (In order to enable reproducibility and also generate new dataset each time we do resample, we use m which is number of model \iterator as random_state).

2. **Hyperparameter tuning:** each time we train a new model, we use the validation set to get the accuracy of current trained m model and keep track of the best m value, best accuracy and best m models. we do prediction for each model on validation set then get mean for each col (which is the majority vote for each sample in the validation datset) and convert each value into 0 or 1 (by comparing to 0.5).
3. **Evaluation:** Once we finish the training and validation, we evaluate the model on the test dataset to get the test_accuracy

2.2.2 With caring about decision tree hyperparameters

- Same as above but each time we train new model, we tune decision tree also on max depth for each different resample.

2.3 AdaBoost Ensemble

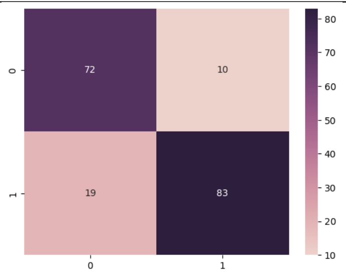
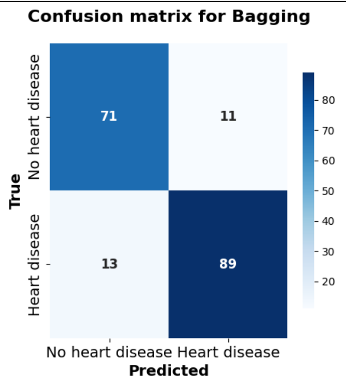
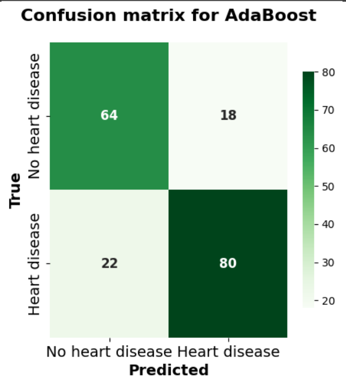
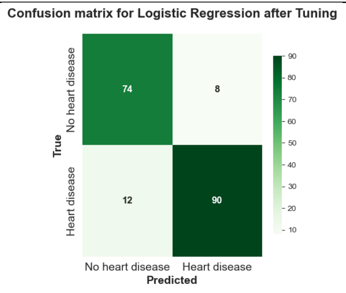
- **Decision Stump:** a weak classifier selecting the best feature and threshold to minimize weighted classification error.
- **Adaboost training:** iteratively trains decision stumps, updates sample weights, and assigns an importance weight alpha to each stump.
- **Prediction:** combines weak learners weighted predictions for final classification.
- **Hyperparameter tuning:** evaluates different number of weak learners to minimize exponential loss.

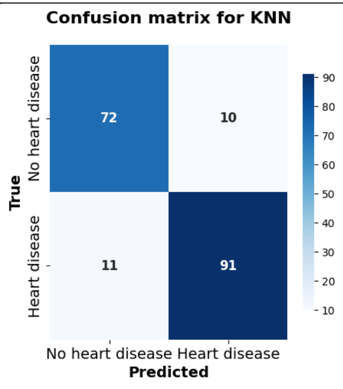
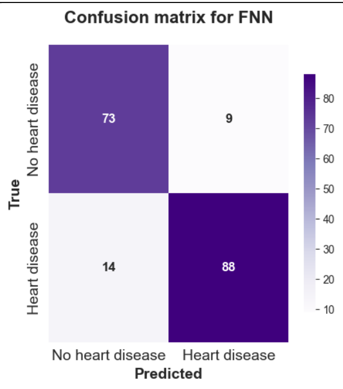
3 Evaluation and Results

3.1 Performance Metrics

- **Accuracy**: The ratio of correctly predicted instances to the total number of instances. It measures how often the model is correct.
- **F1-Score**: The harmonic mean of precision and recall. It balances the trade-off between precision (how many selected items are relevant) and recall (how many relevant items are selected).
- **Confusion Matrix**: A table that summarizes the performance of a classification model by showing the counts of true positives, true negatives, false positives, and false negatives.

3.2 Results and Visualizations

Model	Accuracy	F1-Score	Confusion Matrix
Decision Tree	0.8423	0.85	
Bagging	0.8695	0.88	
AdaBoost	0.7826	0.8	
Logistic Regression	0.89	0.9	

Model	Accuracy	F1-Score	Confusion Matrix
KNN	0.8858	0.89	
FNN	0.86	0.88	

4 Analysis and Discussion

- **Best model KNN and Logistic Regression:** accuracy of nearly **88-89** and F1-score of **89-90** and fewest false negatives **FN:(11-12)** respectively.
- **FNN and Bagging:** both performed well; fnn captures comples patterns, while bagging reduces overfitting and has lower false negatives than a decision tree.
- **Decision tree:** prone to overfitting with higher false negatives making it less reliable.
- **Adaboost:** lowest accuracy 78.26 highest false negatives 22 and too sensitive to noise making it unsafe for medical use.

5 Bonus Implementations

5.1 K-Nearest Neighbors (KNN) Classifier

- **Model training:** iteratively trains built_in KNN model with specifying number of nearest neighbors (one model per time), the model just memorize the training data and it use Minkowski matrix as default and p = 2 which mean Euclidean distance. Also, we standardized the training features before training the model on it.

$$d(A,B) = \left(\sum |x_i - y_i|^p\right)^{1/p}$$

Figure 1: Minkowski Equation

- **Hyperparameter tuning:** each time we train a new model, we use the validation set to get the accuracy of current trained model. the model get the k nearest neighbors and take the majority vote (it counts how many of the k neighbors belong to each class and assigns the most common class label to the test sample) and keep track of best k that achieve the best accuracy. Also, we standardized the training features before training the model on it.
- **Evaluation:** Once we finish the training and validation, we evaluate the model on the test dataset to get the test_accuracy

5.2 Logistic Regression

- **Logistic regression:** a model used to predict the probability of a binary outcome.
- **Training:** first normalize the features to have a mean of 0 and standard deviation of 1 to improve model performance and ensure that all features contribute equally. Then, the model is trained on the scaled data. Finally, predictions are made using the trained model on the scaled test data.
- **Hyperparameter tuning:** exploring different combinations of regularization strength and penalty type (lasso or ridge) for the logistic regression model and keeping track of the best hyperparameters and corresponding accuracy.

5.3 Feedforward Neural Network (FNN)

- **FNN:** a simple neural network with an input layer, one hidden layer(Relu activation), and an output layer(sigmoid activation).
- **Model comilation:** we used Adam optimizer and binary cross-entropy loss for binary classification task.
- **Training process:** trains using batch gradient descent with validation and callbacks.
- **Training callbacks:** includes early stopping and learning rate reduction to enhance model training efficiency.
- **Early stopping:** stops training if validation accuracy doesn't improve for a certain number of epoches and restoring best weights.
- **Reduction of learning rate:** lowers the learning rate when validation accuracy shows no progress to help the model converge better.

6 Conclusion

- Bagging improved over a single decision tree by reducing overfitting and lowering false negatives, while AdaBoost (using decision stump) performed the least because of its sensitivity to noise.