Assignment # 1: Heart Failure Classification Problem

1 Problem Statement

Heart failure is a critical condition often caused by cardiovascular diseases (CVDs). Early prediction of heart failure can aid in timely medical intervention and improve patient outcomes. In this assignment, you will build classifiers to predict whether a patient is at risk of heart failure based on clinical features. The data set for this task is the Heart Failure Prediction dataset (918 samples, 11 features, 2 classes) available on Kaggle – it includes various patient attributes (e.g. age, blood pressure, cholesterol levels, etc.) and a binary label indicating the occurrence of heart failure. This assignment aims to familiarize you with several classical classification methods and their performance on a real-world medical dataset. You will implement certain algorithms from scratch and utilize library implementations for others. Through this process, you will learn how different classifiers work, how to evaluate them, and how to analyze their results.

2 Dataset and Preparation

To begin, download the Heart Failure Prediction dataset from Kaggle (you may need to create a Kaggle account to access the data). Understand the format of the dataset, including feature meanings and the binary class label. Perform the following steps to prepare the data for modeling:

1. **Fixed Random Seed:** Use a fixed random seed (42) for all random operations to ensure reproducibility across experiments.
2. **Train/Validation/Test Split:** Split the dataset into 70% training, 10% validation, and 20% testing. Use stratified splitting so that each subset maintains the same class distribution (this is important given the binary classification).
3. **Feature Preprocessing:** Most features are numerical. Ensure that they are in a suitable format for the classifiers. No extensive preprocessing is required, but you may consider normalizing or standardizing features especially for the KNN and Neural Network models (to ensure no single feature dominates distance calculations or network training). Categorical features (if any) should be encoded (e.g., one-hot encoding) since most algorithms expect numeric input.
4. **Purpose of Validation Set:** Use the validation set for tuning hyperparameters and model selection. Do not peek at the test set during training or tuning. You will use the test set only for the final evaluation of your models.

3 Implementation Details

In this assignment, you will work with a range of classification algorithms. You are required to implement three methods from scratch (without using high-level machine learning library functions for these algorithms).

For each of the below models, train them using only the training set. You may use the validation set to tune hyperparameters or decide on early stopping, etc. Once you have finalized a model (and its hyperparameters) using the validation data, you will evaluate its performance on the test set (as outlined in the next section).

3.1 Decision Tree (Your implementation):

Implement a decision tree classifier from scratch. Use the Information Gain algorithm to split nodes.

Hint: You can structure your tree as a recursive set of nodes. Each leaf will output a class prediction. Consider implementing a maximum depth or minimum samples split to prevent overfitting. Train your decision tree on the training set.

3.2 Bagging Ensemble (Your implementation):

Implement a Bagging classifier from scratch. Bagging (Bootstrap Aggregating) involves training an ensemble of models (base learners) on different bootstrap samples of the training data and averaging their predictions. In this assignment, use your decision tree implementation as the base learner for bagging. For example, you could train an ensemble of, say, 10 or 20 decision trees, each on a random bootstrap subset of the training data. During prediction, aggregate the results (for classification, this can be a majority vote among the trees). Train the bagging ensemble on the training set.

3.3 AdaBoost Ensemble (Your implementation):

Implement the AdaBoost algorithm from scratch. AdaBoost builds an ensemble of weak learners in sequence, where each subsequent learner focuses on the mistakes of the previous ones. Use a simple decision stump or a shallow decision tree as the weak learner for AdaBoost (you can again leverage your decision tree code, restricting it to a depth-1 tree for a stump, or a few levels for a weak tree). Train the AdaBoost ensemble on the training data. You should decide on a reasonable number of boosting rounds (for instance, 50 rounds) or experiment with the number of weak learners using the validation set.

4 Evaluation

Compare the performance of the learned models by realizing the following. a. Compute the accuracy and F-Score for each model. b. Plot the confusion matrices and find the most confusing classes After training the models, evaluate each of them on the test set and record the performance metrics. We will use two primary metrics to assess classification performance: Accuracy and F1-score. Additionally, you will examine the confusion matrix for each model to get deeper insight into their predictions. Perform the following for each of your trained models:

- **Accuracy**
- **F1-Score**
- **Confusion Matrix**

Summarize the accuracy and F1-score of each model in either a table or a concise list in your report, and include the confusion matrix figure for each model. And finally compare the performance of the learned models

5 Analysis and Comparison

After obtaining the results, analyze and compare the performance of the different classifiers. In your report, discuss the following points:

- Which models performed best and why?
- Compare decision trees to ensemble methods (Bagging, AdaBoost).
- **Insights and Observations:** Based on the confusion matrices, identify if there are specific misclassification patterns.

6 Bonus

6.1 K-Nearest Neighbors (KNN) Classifier:

Train a KNN classifier on the training set. You should experiment with different values of k (e.g., 3, 5, 11, etc.) and possibly distance metrics, using the validation set to select the best performing k. Use that tuned KNN model for final evaluation on the test set.

6.2 Logistic Regression:

Train a logistic regression classifier on the training set. Logistic regression is a linear model that uses the logistic sigmoid function to output a probability for the positive class. You can use default parameters for the logistic regression, but feel free to tune aspects such as regularization strength using the validation set. (For instance, you might try different values of the regularization parameter C.) Ensure that the training converges (you might need to set a maximum number of iterations if using scikit-learn).

6.3 Feedforward Neural Network (FNN):

Create a small feedforward neural network to classify the data using a library of your choice (such as TensorFlow, Keras, or PyTorch). Design a simple architecture, for example, a network with one hidden layer (e.g., 16 or 32 neurons) with a non-linear activation (ReLU is a good choice) and an output layer with a single neuron (for binary classification) using a sigmoid activation. Using binary cross-entropy loss, you can train the network for a fixed number of epochs (or until convergence). Use the validation set to help choose hyperparameters such as the number of hidden neurons, learning rate, or number of training epochs.

6.4 Latex

Write a well-documented report in Latex format.

7 Submission Notes

- Work in groups of 3 students.
- Submit a **Jupyter Notebook** or a well-commented Python script containing your code.
- Submit a **PDF report** detailing your approach, results, and analysis.
- Ensure reproducibility by setting random seeds and structuring code properly.

Note

- While discussions are allowed, all code and reports must be your own work. Plagiarism will result in a penalty.