Assignment 1: K-Nearest Neighbors Classification using Python

1. Objective

The purpose of this assignment is to practice applying Python for a basic machine learning task. In this project, we focused on the K-Nearest Neighbors (KNN) algorithm to classify a synthetic dataset with three distinct classes. **The main goal is to generate the dataset, train the KNN model, and evaluate its performance using accuracy scores and visualizations.** Both default KNN parameters and a specifically tuned KNN classifier are applied so that we can observe any differences in performance and clearly report the outcomes.

2. Dataset Creation

We generated an artificial dataset containing **150 samples**, each assigned to one of three classes. Each sample has **two numerical features**. The centers for the classes were defined as follows:

- **Class 0:** Centered at (2, 4)
- **Class 1:** Centered at (6, 6)
- **Class 2:** Centered at (1, 9)

The dataset was created using **Scikit-Learn's make_blobs** function. This ensured that the classes are clearly separated, making it suitable for classification with the KNN algorithm.

3. Data Splitting

After creating the dataset, the dataset was split into training and testing datasets using Scikit-Learn's train_test_split function.

- Training Data: 80% of the total dataset (120 samples)
- Test Data: 20% of the total dataset (30 samples)

The split would allow the model to be trained on most of the data and also have a separate segment to evaluate its performance on new data. A random state was set to allow for reproducibility of results.

4. Model Technique – KNN Classifier

For this project, we applied the K-Nearest Neighbors (KNN) classifier to categorize our synthetic dataset. KNN is an instance-based learning algorithm that assigns a class to a data point based on the majority class of its nearest neighbors.

We trained two versions of the KNN classifier to study the effect of parameter tuning:

1. **<u>Default KNN Classifier:</u>**
    - Used the default parameters of the KNeighborsClassifier() function.
    - Defaults include n_neighbors=5, Euclidean distance (p=2), and uniform weighting.

Reason for choosing: This provides a baseline performance of the algorithm without any modifications, helping us understand how KNN behaves "out-of-the-box" on our dataset.

2. **Specific KNN Classifier:**
   - Set parameters explicitly to explore their effect:
   - n_neighbors=7 – considers 7 nearest neighbors instead of the default 5.
   - metric='minkowski' with p=2 – Euclidean distance.
   - weights='uniform' – all neighbors contribute equally.

   Reason for choosing: Adjusting the number of neighbors and confirming the distance metric allows us to test whether tuning these hyperparameters improves performance or affects stability. It also demonstrates how KNN can be optimized for different datasets.

Both classifiers were trained on 80% of the data and tested on the remaining 20%. This approach allows us to compare baseline performance with a tuned model and justify the choice of parameters based on observed results.

5. Model Performance and Evaluation

Training Accuracy: 1.0
Test Accuracy: 1.0

We used two KNN classifiers: one with default settings and one with specific parameters (n_neighbors=7, Euclidean distance, uniform weights). Both classifiers correctly predicted all training and test samples, giving 100% accuracy.

Reason for Two Approaches:

- The default classifier helps understand how KNN performs without tuning.
- The specific classifier shows the effect of adjusting parameters. In this case, even with 7 neighbors, the classifier performed perfectly, indicating that the dataset is clearly separated and easy to classify.

Predictions:
Predicted values for both training and test data matched the actual labels exactly. This confirms that the model is effectively classifying all points correctly.

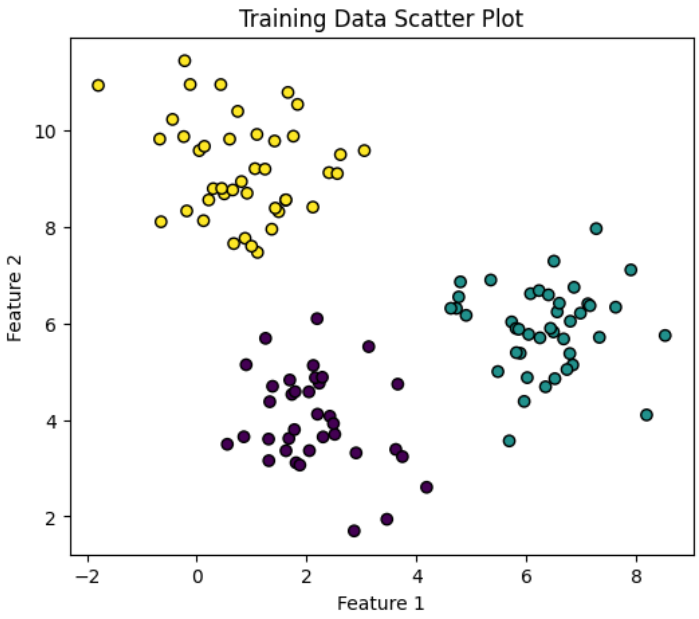
6. Data Visualization

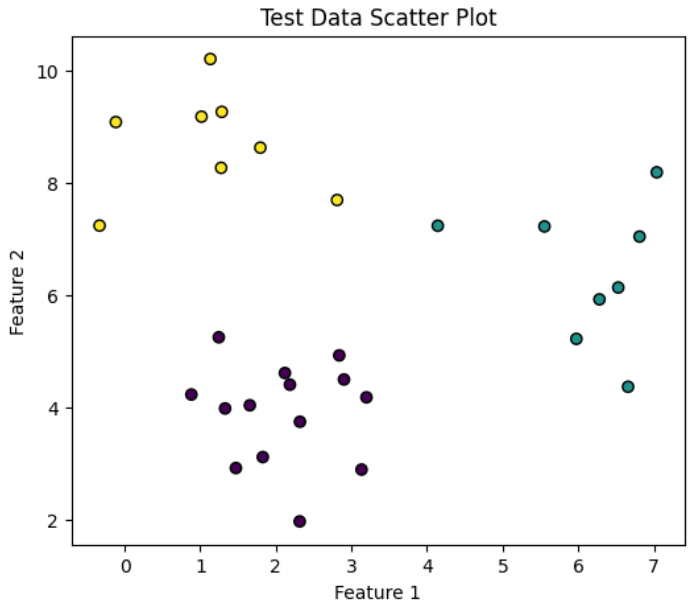
To better understand the performance of our KNN classifiers, we visualized the data and results using different plots.

A. Scatterplots

- **Training Data:** The points are clearly grouped into three regions, each representing a different class. This shows that the classes are well separated, making KNN an effective choice.

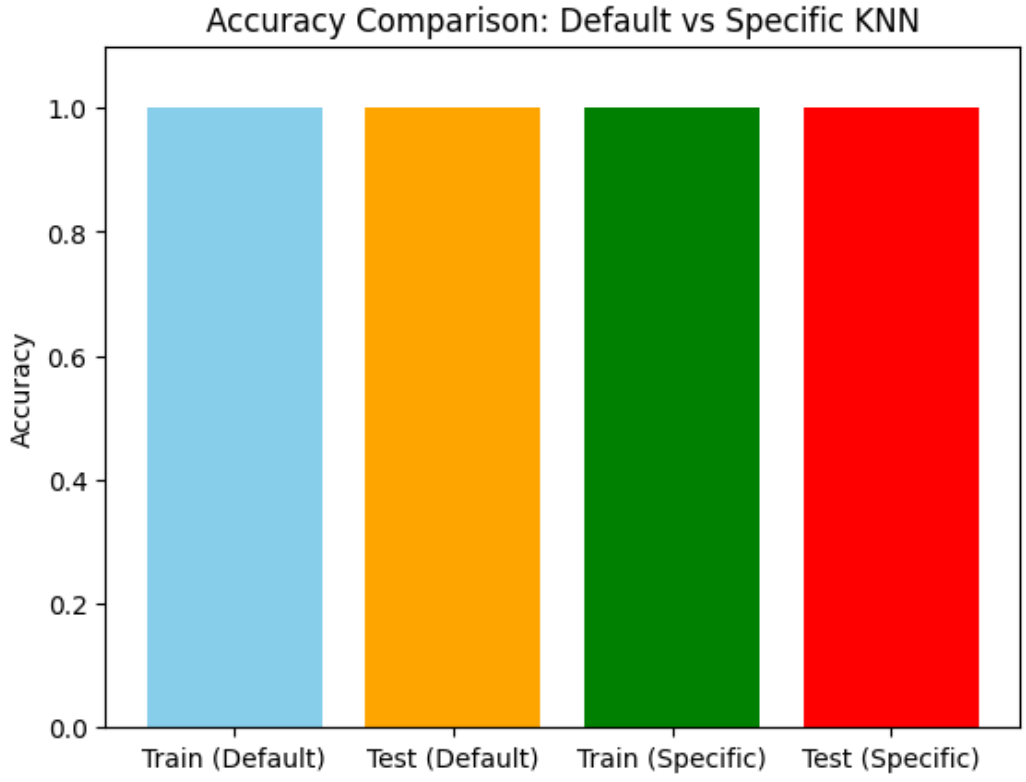
- **Test Data:** The test data points maintain the same separation as training data, indicating that the model can classify unseen data accurately.





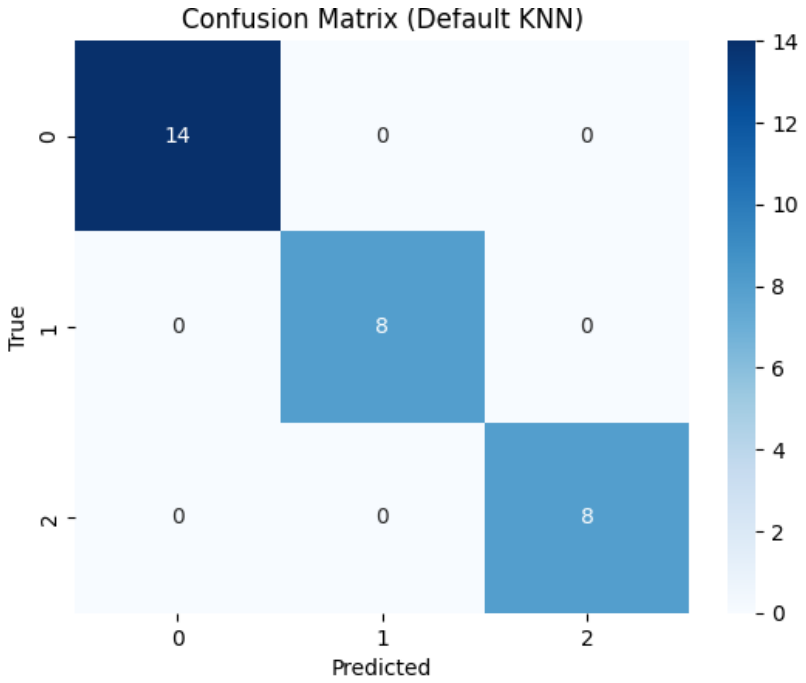
B. Accuracy Comparison Bar Chart

A bar chart was plotted to compare the training and test accuracies of the default and specific KNN classifiers. Both classifiers achieved perfect accuracy (1.0), showing that the model generalizes well on the test data.



Confusion Matrix

A confusion matrix was generated for the default KNN classifier on test data. All values appear on the diagonal, which confirms that all test samples were correctly classified without any misclassification.



7. Conclusion

- The K-Nearest Neighbors (KNN) classifier performed exceptionally well on this synthetic dataset due to the clear separation of classes.
- Both the default KNN and the specific KNN classifier with customized parameters achieved **100% accuracy** on training and test datasets, indicating that the model generalizes perfectly to unseen data.
- The scatter plots and confusion matrix further confirmed that the model correctly classified all data points with no errors.
- This simple dataset demonstrates the effectiveness of KNN for linearly separable data.
- For more complex or noisy datasets, hyperparameter tuning, cross-validation, or other classifiers like SVM or Decision Trees could be explored.