ASSIGNMENT 1: A Machine Learning Project with Python

Objective:

In this project, we're working with the K-Nearest Neighbors (KNN) algorithm to classify a synthetic dataset with three different classes. The main goal is to accurately categorize the data and see how well the model performs. We'll start by creating the dataset using Scikit-Learn's make_blobs, making sure the classes are clearly separated. After splitting the data into training and test sets, we'll apply the KNN algorithm using Euclidean distance. To evaluate how well the model is doing, we'll check the accuracy and look at the confusion matrix. Finally, we'll visualize both the distribution of the data and how the classifier performs.

Dataset Creation:

The artificial dataset contains 150 samples, each assigned to one of three classes, with two numerical features. The centers for each class are defined as follows:

- Class 1: Centered at (2, 4)
- Class 2: Centered at (6, 6)
- Class 3: Centered at (1, 9)

Data Preparation:

After generating the dataset, we split the data into training (80%) and testing (20%) dataset using train_test_split.

Model Technique- KNN Classifier:

For this project, we use the K-Nearest Neighbors (KNN) classifier with the following parameters:

- Number of neighbors (n_neighbors): 3
- Distance metric (p=2): Euclidean Distance
- Weighting method (weights='uniform'): All neighbors contribute equally
- Leaf size=30: Number of points to search at a leaf node for efficient search
- Number of parallel jobs (n_jobs=1): Use 1 parallel jobs to speed up the computation

KNN works by classifying a new data point to the most frequently occurring class among its nearest neighbors. Given that our dataset has clearly separated classes, we expect KNN to perform well.

Model Performance and Evaluation:

- **Training Accuracy:** 1.0 | **Test Accuracy:** 1.0
  Similar to default classifier where no parameters were used to train the data, the second classifier with specific parameters also achieved perfect accuracy (100%) on both training as well as test dataset, indicating that it generalizes well on the unseen test dataset.

Data Visualization:

To better understand the dataset, we visualize the training and testing data using a scatter plot. The data points are distinctly separated, indicating that it is a suitable candidate for classification with KNN. Looking at the confusion matrix and its heatmap, we can see that the model is effectively distinguished between the three classes.

Scatter Plot:

- **Training Data Scatter Plot** – The data points are grouped into separate regions, each corresponding to a different class, showcasing clear class separation.
- **Test Data Scatter Plot** – The test points maintain distinct groupings, highlighting the effectiveness of KNN in correctly classifying the data.

Confusion Matrix:

A confusion matrix was also calculated for the test data of the KNN classifier to visualize the classification performance. Given the perfect accuracy, the confusion matrix contained only diagonal values, indicating that all test samples were correctly classified. The matrix provides a clear illustration of the model's ability to correctly classify each class without errors.

Conclusion:

- The dataset is linearly separable, making KNN an ideal choice for classification.
- The model achieved high accuracy, confirming its effectiveness.
- Due to the small dataset, KNN is computationally efficient.
- For larger datasets, models like SVM or Decision Trees may offer better scalability.
- The chosen number of neighbors (k=3) provides a balance between bias and variance.
- Future experiments could explore different k values to further optimize performance.

Future Recommendations:

- The dataset was very simple and artificial with only 150 samples, and the simplicity of data leads to high accuracy in KNN. Real-world datasets usually involve more noise, unbalanced classes, and missing values.
- Cross-validation could be explored to assess the model's performance more robustly and avoid overfitting.
- Experiment with datasets that contain noise or more complex relationships to better understand the KNN algorithm's behavior in real-world scenarios.
- Further hyperparameter tuning could be done to understand its impact on classifier performance, such as adjusting the number of neighbors (n_neighbors) or changing the distance metric.