ASSIGNMENT 1
Machine learning with Python

Objective

The objective of this assignment is to practice implementing a basic machine learning project in Python using a simulated dataset. The task focuses on building and evaluating a K-Nearest Neighbors (KNN) classifier on artificially generated data, analyzing model performance, and comparing the results of the default KNN configuration with a parameterized version. This exercise helps in understanding model training, evaluation metrics, and visualization of decision boundaries in a classification setting.

Methodology

1. Libraries Used

scikit-learn, numpy, matplotlib, seaborn, pandas.

2. Dataset Creation (Artificial)

To replicate the study, we generated an artificial dataset using the make_blobs function from sklearn.datasets. This function allows the creation of synthetic datasets by sampling from Gaussian (normal) distributions centered around specified points.

- Centers: We defined three cluster centers at coordinates [[2, 4], [6, 6], [1, 9]], which represent the mean positions of each class in a 2D feature space.
- Classes: Since we used three centers, the dataset naturally had three classes.
- Samples: A total of 150 data points were generated and distributed across the three clusters.
- Random State: A fixed random_state=1 was applied to ensure reproducibility of results, meaning every run produces the same dataset.

The generated dataset contained two input features (corresponding to the x and y coordinates of the points) and one target label (class assignment based on which cluster a point belonged to). This dataset was then split into training (80%) and testing (20%) subsets for model building and evaluation.

3. Model Training

- Two KNN models were trained:

- **Default KNN:** KNeighborsClassifier() with all default parameters.
- **Custom KNN:** KNeighborsClassifier() with tuned parameters:
    i. n_neighbors=5
    ii. metric='minkowski', p=2 (Euclidean distance)
    iii. weights='uniform'

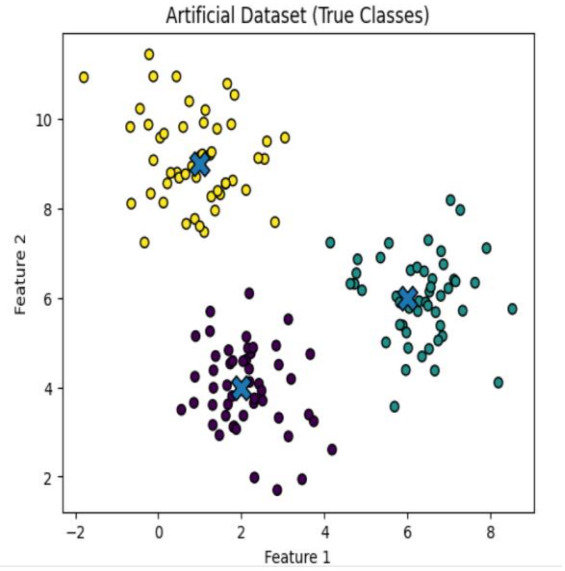
Both models were trained on the artificial dataset.

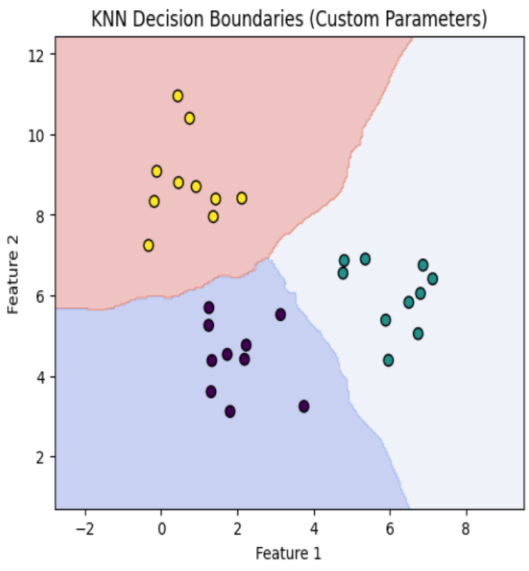
4. Evaluation

- Accuracy scores were calculated for both training and test sets.
- Confusion matrices were generated and visualized using heatmaps.
- Decision boundary plots were created to illustrate how each KNN model separated the classes in 2D space.

Results

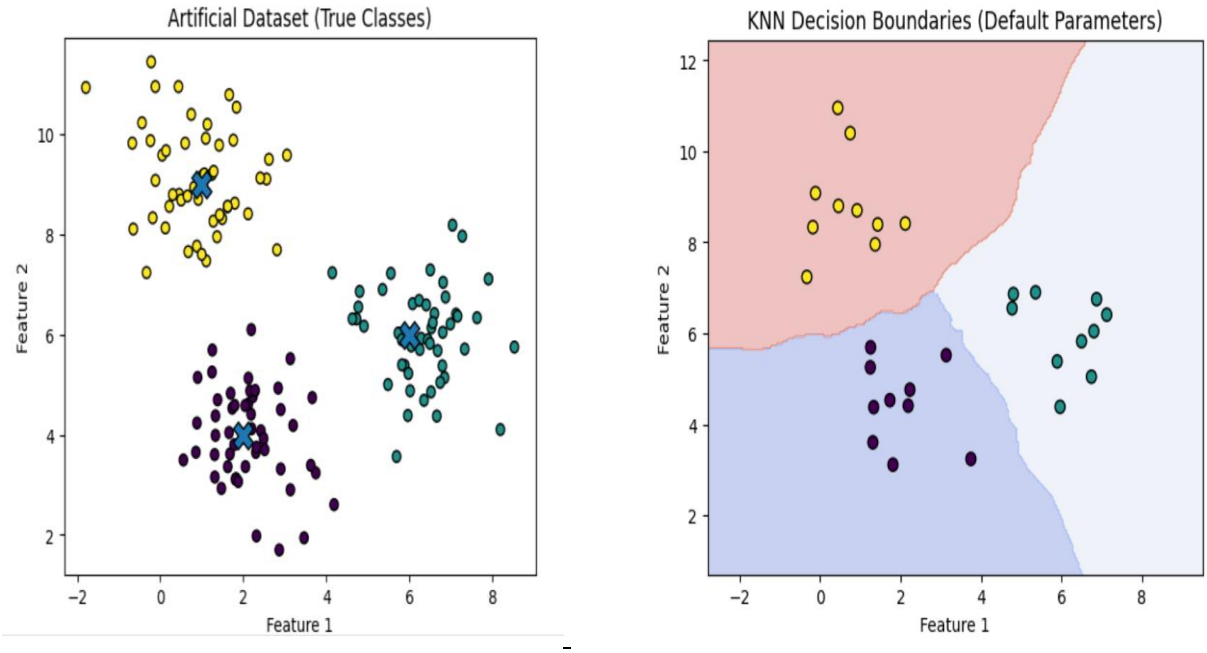
KNN Classifier with parameters





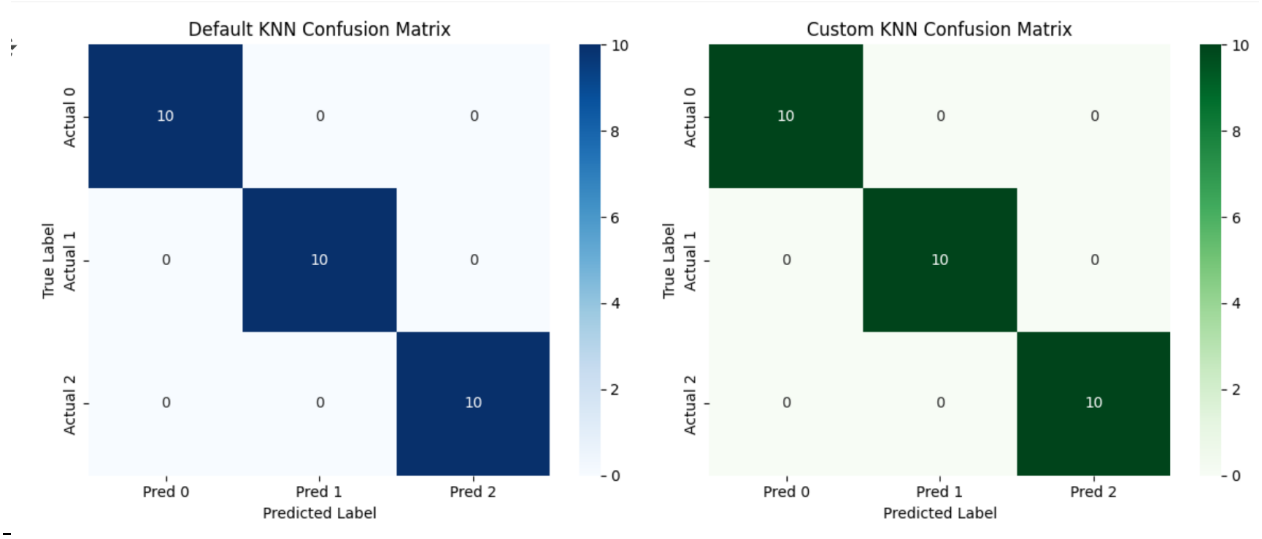
This figure compares the true class distribution of the artificial dataset (left) with the decision boundaries produced by the KNN model with custom parameters (right). On the left, three clearly separated clusters represent the true classes, each centered around a distinct point. On the right, the decision boundary plot shows how the KNN model partitions the feature space into three regions, each corresponding to one class. The shaded areas represent the model's predicted regions, while the dots are the actual test data points. Since the dataset is linearly separable, the model's decision boundaries align very well with the true clusters, and all test samples fall into their correct regions, which explains why the model achieved perfect classification accuracy.

KNN Classifier without any parameters



This figure shows a comparison between the true classes of the artificial dataset (top plot) and the decision boundaries produced by the default KNN model (bottom plot). In the top plot, three well-separated clusters represent the true labels, each centered around a defined point. In the bottom plot, the KNN model partitions the feature space into three colored regions, each corresponding to one class. The test samples (dots) fall neatly into their respective regions, showing that the model was able to perfectly classify them. The smooth and distinct decision boundaries highlight that the dataset is highly separable, which explains why the default KNN achieved 100% training and testing accuracy—every sample was classified correctly without any overlap or misclassification.

Confusion Matrix for Default KNN and KNN with parameters



These confusion matrices compare the performance of the default KNN model (left) and the custom KNN model (right). In both cases, the diagonal cells (Actual = Predicted) contain all the test samples for each class, with values of 10 per class. This indicates that both models correctly classified all instances of all three classes, resulting in no misclassifications (all off-diagonal cells are zero). Consequently, both models achieved 100% accuracy, precision, recall, and F1-score across all classes. The identical results highlight that for this particular artificial dataset—where the clusters are very well separated—the choice between default and custom parameters does not affect classification performance.

Conclusion

In this assignment, we built and evaluated K-Nearest Neighbors (KNN) classifiers on a simulated dataset with three well-separated classes. Both the default KNN and the custom parameterized KNN achieved perfect classification performance, with 100% accuracy on both training and testing sets, as well as flawless precision, recall, and F1-scores across all classes. The decision boundary plots clearly illustrated how each model partitioned the feature space, while the confusion matrices confirmed the absence of misclassifications. These results demonstrate that when datasets are highly separable, KNN can classify perfectly without the need for extensive parameter tuning. However, in more complex or overlapping datasets, hyperparameter choices such as the number of neighbors, distance metric, and weighting scheme would play a more significant role in improving generalization. This assignment provided valuable hands-on experience in dataset generation, model training, evaluation, and visualization, reinforcing core concepts of supervised machine learning.

Future Recommendations

- **Experiment with Different Hyperparameters:**
  Test varying values of k, different distance metrics (e.g., Manhattan, cosine), and weighting schemes to see how they impact performance, especially on less separable datasets.

- **Apply Cross-Validation:**
  Instead of a single 80/20 split, use k-fold cross-validation to get a more reliable estimate of model performance across multiple splits.

- **Introduce Noise or Overlap in Data:**
  Generate artificial datasets with overlapping clusters or added noise to test how robust KNN is under more challenging conditions.

- **Compare with Other Algorithms:**
  Benchmark KNN results against other classifiers such as Logistic Regression, Decision Trees, or SVM to evaluate differences in accuracy, interpretability, and computational efficiency.

- **Scalability Considerations:**
  Since KNN can be computationally expensive on large datasets, explore techniques like dimensionality reduction (PCA) or approximate nearest neighbors to improve efficiency.

- **Real-World Dataset Application:**
  Extend this work by applying KNN to real-world datasets (e.g., UCI repository, Kaggle datasets) to analyze its effectiveness beyond synthetic data.