K-Nearest Neighbors (KNN) – Summary Report

Shankeshi Sonali

1) Context and Goal

For this classification exercise, we will use K-Nearest Neighbors (KNN) classifier to perform this task. We will be working with a synthetic dataset generated using make_blobs, so we might not be using a real dataset to train the KNN model, but we will be training a KNN model to separate the points into three distinct classes (classes will be clearly labeled) and we will determine how well those classes predict the class of new (test) points.

2) Dataset Creation

We will create 150 points clustered around three centers. Each point will have two features (X and Y). The points will be created centered around three centers located at [[2, 4], [6, 6], [1, 9]]. Therefore, the clusters will be relatively spatially obvious, and understanding how KNN learns will be more intuitive

3) Train/Test Split

Next, we will split the dataset into an 80% training data and a 20% test data. The train_test_split function will specify random_state=42, (the random_state is in both make_blobs and train_test_split, so that every time we read in the function the data points, and data split data will be the same). The training data is simply for fitting the model, and the test date is to check how well the model generalizes to new, unseen data.

4) Model and Method

KNN is based on the assumption that points that are near each other have similar labels. KNN then classifies a new point by considering the k-nearest training points (neighbors) and selecting the most frequent label. A small value of k is sensitive to noise. A larger value of k will give a smoother decision boundary.

We will use the KNeighborsClassifier default settings (k=5). KNN predicts a label for a new point by examining the k closest training points and selecting the most common label of those k points. It is a simple non-parametric method.

5) Evaluation Result

After training, we predict on the test set and compute accuracy. Accuracy = 1.000. Accuracy is the fraction of test points that the model classified correctly. Because our clusters are well separated, accuracy is high.

6) Visual Outputs

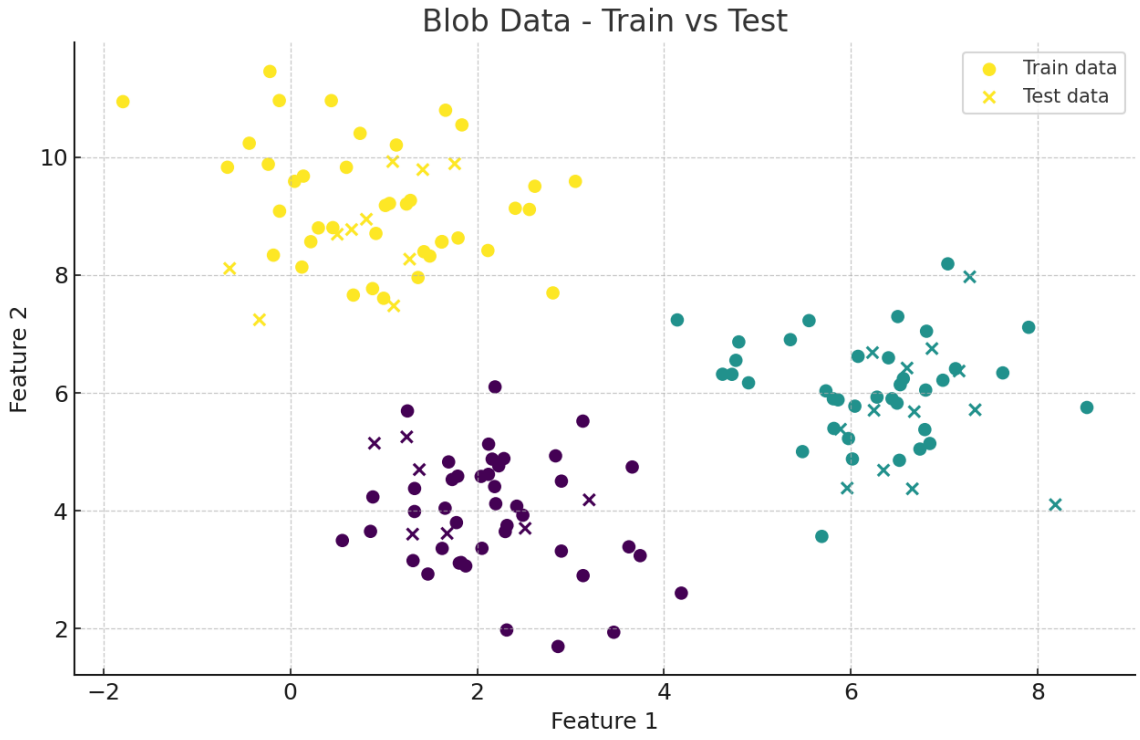

Figure 1 – Train vs Test Split: Circles show training points and X markers show test points.

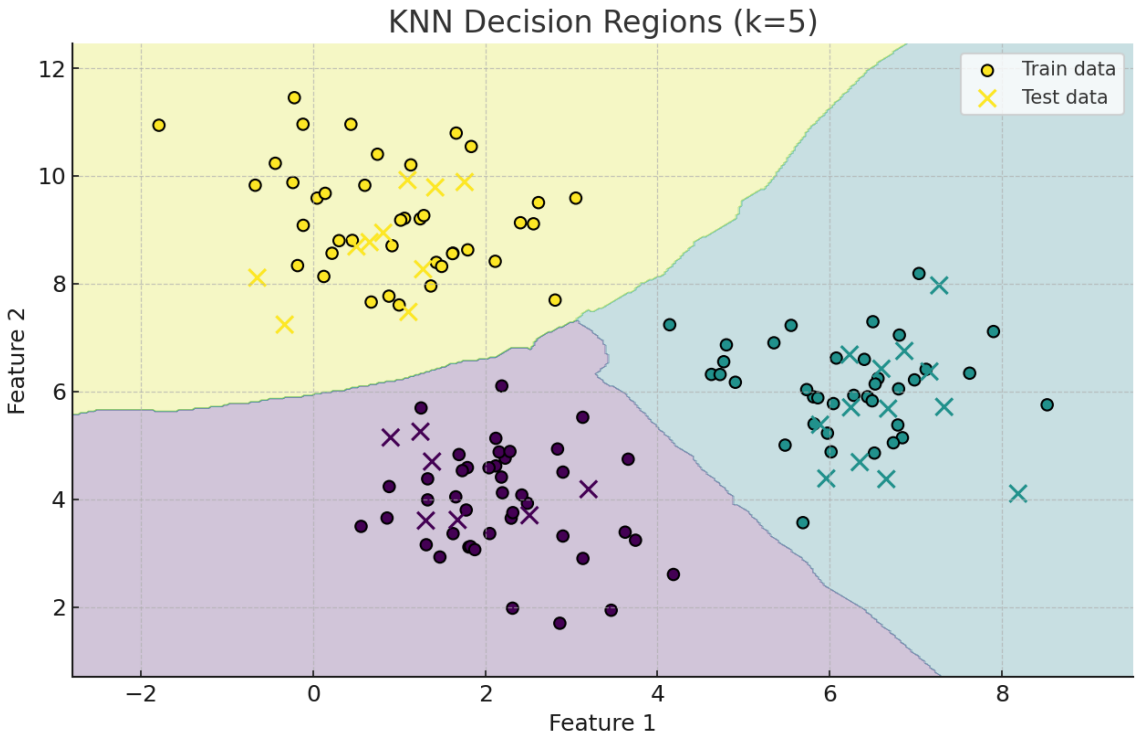

Figure 2 – Decision Regions: The shaded areas show predicted classes, with training and test points overlayed.

7) Confusion Matrix – Explanation and Interpretation

In addition to accuracy, one other way to assess the overall effectiveness of classifier models is to utilize a Confusion Matrix. A Confusion matrix reveals how many predictions are correct, and where the model made errors. The rows show actual (true) classes and the columns show predicted classes.

• The numbers located on the diagonal, top-left to bottom-right, showing the amount of correct predictions (True Positives).

• The numbers that are outside the diagonal are errors (misclassifications).

• Each row shows us, for a true class, how the model classified the test points.

In our testing results, the confusion matrix looked like this:

True 0 → Pred 0: 7, Pred 1: 0, Pred 2: 0

True 1 → Pred 0: 0, Pred 1: 13, Pred 2: 0

True 2 → Pred 0: 0, Pred 1: 0, Pred 2: 10

This means the model made no errors on the test samples at all - so it has done its predictive task without error. This is also why it has an accuracy of 1.000 (100%). Why is it important: The confusion matrix gives you a lot more detail than accuracy alone. For instance, in real datasets, one class might be more difficult to predict than the others. The overall accuracy might looks fine, but the confusion matrix may show you whether your model is confusing classes. In our case, wouldn't you know it, because the blobs are well separated, the matrix shows a perfect diagonal (correct predictions).

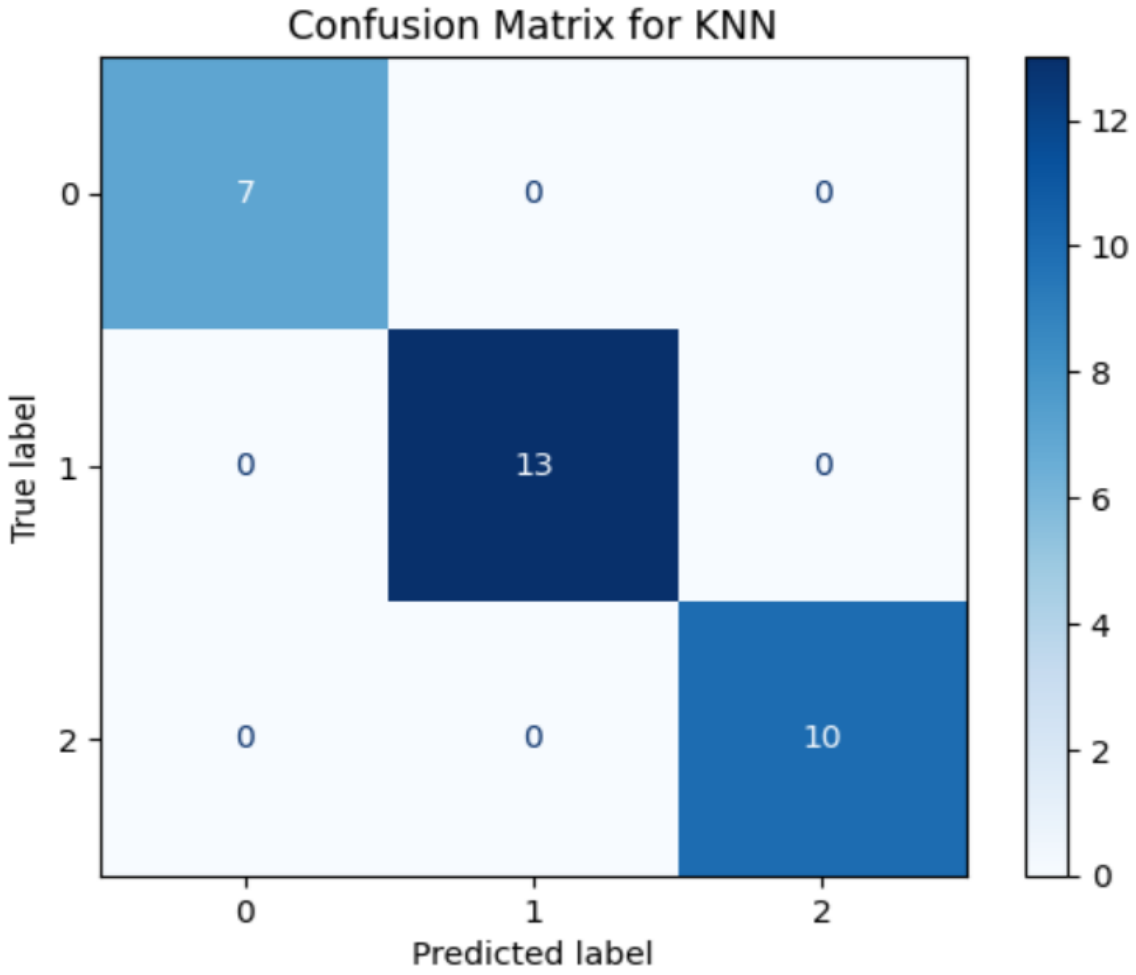

Figure 3 – Confusion Matrix: All values are on the diagonal, showing perfect predictions

8) Code Structure (Step-by-Step)

1. Import libraries (sklearn, matplotlib, numpy).
2. Create synthetic data with make_blobs and choose cluster centers.
3. Split into training (80%) and testing (20%).
4. Initialize KNN (k=5) and fit the model.

5. Predict on test data and compute accuracy.
6. Plot the train/test points.
7. Build a grid, predict each cell, and draw decision regions.

9) Conclusion

The KNN model achieved the maximum accuracy of 100% using the simple synthetic dataset above. The confusion matrix confirmed that we classified every test point correctly, with all values appearing on the diagonal and no other off-diagonal cases. This was to be expected because the synthetic dataset was designed with clear and useful clusters, and the separation of the clusters made the classification task easier.

Typically, in the real world we rarely see data this clean or with perfect separable classes. We've also seen how accuracy itself can be somewhat misleading, which is one of the reasons a confusion matrix can be useful: it tells us about overall performance, but also which of the classes are predicted correctly and where the misclassifications occurred. The value in that allows us to diagnose the model's weaknesses more easily.

If the model didn't perform perfectly then we could think about a few alternatives or improvements including:

- Tuning k (using a smaller k version can be sensitive to local noise and larger k will smooth the decision boundary).
- Using distance weighed neighbours and giving respective importance to closer points.
- Scaling all features to ensure distances are compared on an even playing field.
- Comparing KNN with other classifiers, for example Logistic Regression, Decision Trees, Naïve Bayes etc because it allows us to differentiate KNN within generalization.