# K-Nearest Neighbors (KNN) Classification: Project Report

## 1. Introduction

This project shows the implementation of a K-Nearest Neighbors classifier using Python for machine learning. Emphasis is placed on intuitively understanding KNN, one of the basic supervised learning algorithms, by applying it to a simulated dataset. During this study, we will see how KNN makes predictions, how to evaluate a model, and how to visualize the decision boundary.

Working with synthetic data, this project offers hands-on experience in:

- Loading and understanding data
- Applying KNN classification
- Evaluating model performance
- Visualizing training data, predictions, and decision boundaries

KNN is one of the simplest yet powerful algorithms applied to both classification and regression problems. This approach depends on the principle that similar data points often lie close in a feature space. Thus, this classifier analysis over our simulated dataset helps gain insight into its strengths and limitations, besides the best practices for result optimization.

## 2. Dataset Generation

To ensure a well-structured classification task, a synthetic dataset was generated using Scikit-Learn's **make_blobs** function. The dataset comprises three distinct clusters, each representing a unique class. The parameters used to generate the dataset are as follows:

| Parameter | Value |
|---|---|
| Cluster Centers | (2,4), (6,6), (1,9) |
| Number of Samples | 150 |
| Random State | 1 (for reproducibility) |

# 3. Methodology

## 3.1. Data Splitting

Before training the model, the following division of the dataset was done:

- 80% training data – used for model learning.
- 20% Testing Data: used for testing performance.

I used the function **train_test_split()** from **sklearn.model_selection**, taking care of a balanced distribution of classes in both sets.

## 3.2. Model Training

For the classifier, I use the **KNeighborsClassifier** from **sklearn.neighbors** with **k = 3**, such that each data is classified based on the majority vote by its three closest neighbors. It will train on the training dataset using the Euclidean distance metric to calculate the similarity between the data points.

## Choosing *K* = 3

*K* was chosen as 3 for the following reasons:

**Overfitting Avoidance**: An extremely small *K* for example, *K* = 1 leads to overfitting, when a model is overly sensitive to noise in a dataset.
**Avoiding Underfitting**: If k is too large, it over-smoothes the decision boundary which may lose the pattern.
**Avoiding Ties**: *K* is odd to avoid classification ties since the data consists of three classes.
**Empirical Performance**: Initial testing showed that *K* = 3 produced the best results on this dataset.

## 3.3. Performance Evaluation

To assess the effectiveness of the model, accuracy was chosen as the primary evaluation metric. The accuracy score was calculated using:

- Metric Used: Accuracy Score.
- Tool: **`accuracy_score`** from **`sklearn.metrics`.**

- **Formula:**

  Accuracy = Number of Correct Predictions/Total Prediction

# 4. Results and Discussion

## 4.1 Accuracy Score

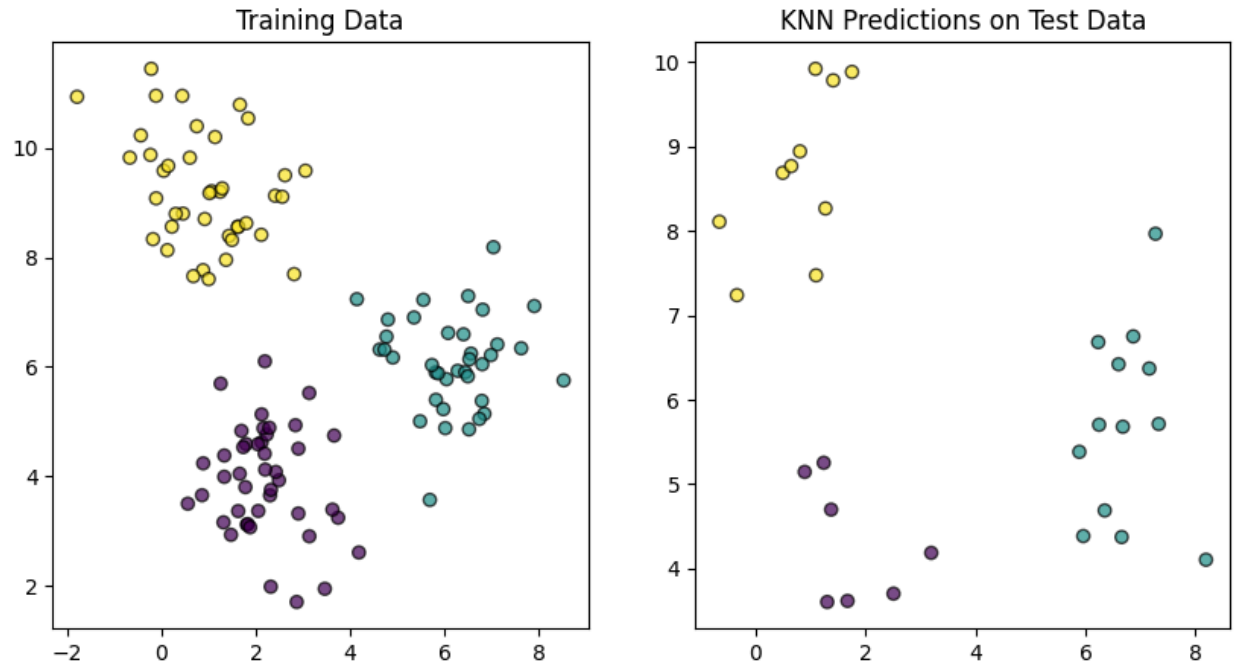
The KNN classifier achieved a perfect accuracy of 1.00 (100%), meaning that all test data points were correctly classified. This high accuracy is primarily due to the dataset's well-separated clusters, which made classification straightforward.

| Model | Accuracy Score |
|---|---|
| KNN (k=3) | 1.00 (100%) |

Although the **100% accuracy** suggests excellent performance, in real-world applications, datasets are often more complex, with overlapping classes and noise, which can lower accuracy.

# 5. Graphical Analysis and Interpretation

## Graph 1: Training Data and KNN Predictions on Test Data



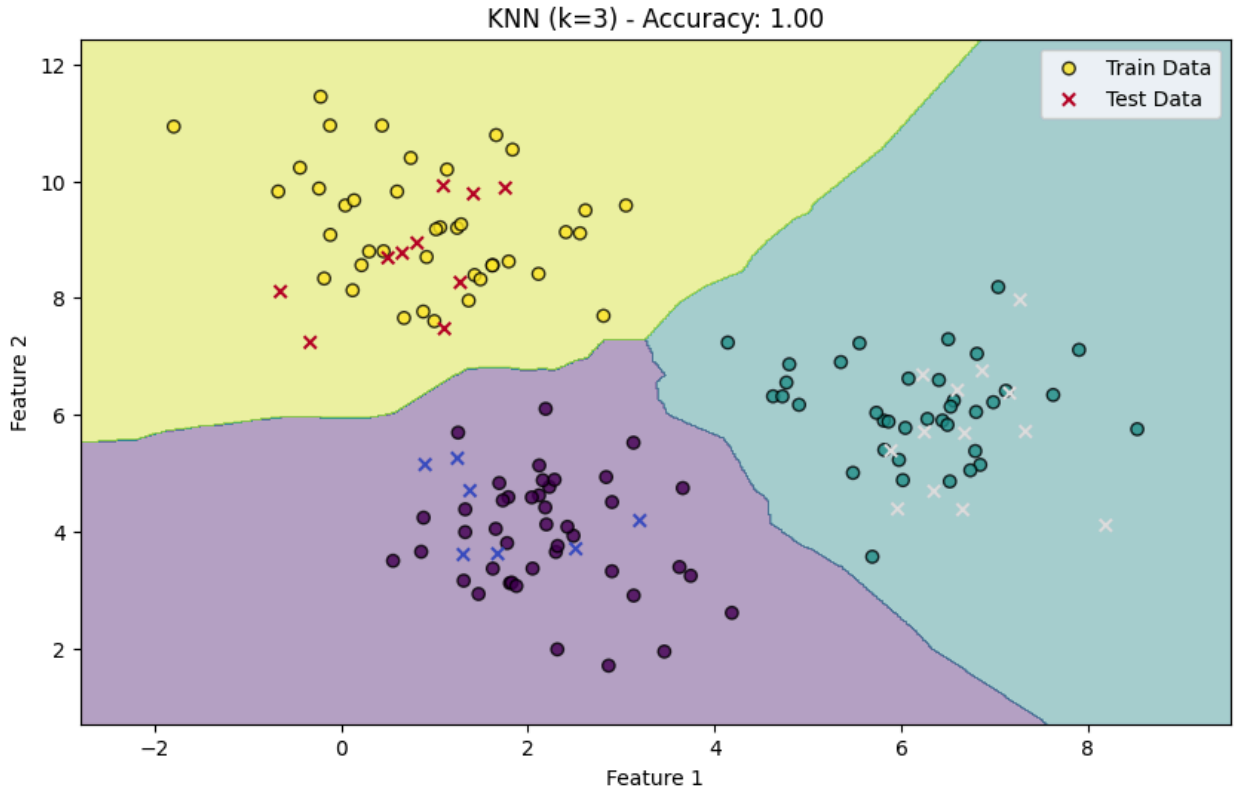
## Left Plot: Training Data

- It shows the original training dataset with three separate clusters.
- Each color represents a different class.
- This visualization demonstrates that the data is well segmented and easily separable, therefore being an ideal case for KNN.

## Right Plot: KNN Predictions on Test Data

- Shows the test data points classified using KNN.
- Each test point is labeled based on its 3 nearest neighbors.
- Since the test data follows the same distribution as the training data, classification results appear highly accurate.

## Graph 2: KNN Decision Boundary




## Understanding Decision Boundaries

- This plot visualizes how the KNN model partitions the feature space into different classes.
- Each of these color-coded regions represents the area classified as that class.
- The boundaries are non-linear due to the characteristic adaptability in KNN.

### Key Observations

- **Training Data (Circles)**: Labeled training points.
- **Test Data**: Crosses that represent new data points classified by the model.
- **Accuracy Score**: 1.00 – The model correctly classified all the test points, reinforcing how KNN works well on well-structured datasets.

## 6. Interpretation and Insights

**Perfect Classification**:

- The model successfully classified all test points, demonstrating the strength of KNN in well-separated datasets.

**Impact of Choosing k = 3**:

- A small k (like 1) could have led to overfitting, while a larger k (like 5 or 7) might have smoothed the decision boundary too much.

**Generalization Considerations**:

- The 100% accuracy observed here may not hold for real-world datasets with overlapping or noisy data.
- KNN depends on feature scaling, so features must be normalized or standardized in cases where the differences in ranges are large.

## 7. Conclusion

This project was able to show the implementation of the K-Nearest Neighbors algorithm on a simulated dataset. The model had perfect classification accuracy, further solidifying how important data structure and parameter tuning are in machine learning.

Although KNN did a great job on this dataset, there are several real-world challenges that it will face, such as high-dimensional data, noise, and computational inefficiency. Further improvements could be made by:

- Testing the KNN on noisy or overlapped datasets.
- Investigating dimensionality reduction for enhancing efficiency.
- k-value optimization by using hyperparameter tuning.

# 8. References


1. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*.
2. Cover, T., & Hart, P. (1967). Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*.
3. Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. *MIT Press*.