Nearest Neighbor, k-Nearest Neighbors and Reference Template Model: Detailed Implementation and Evaluation

Niti Priya ,*Research Intern , Indian Institute of Technology, Dharwad*

***Abstract*— This report describes the application of the Nearest Neighbor, k-Nearest Neighbors algorithm and the Reference Template Model in classification tasks. Their performance on linear, non-linear, and real-world datasets is discussed, demonstrating the ability of these models to adapt to a wide range of data patterns. Key visualizations, such as scatter plots, decision boundaries, and confusion matrices, support an extensive analysis. Metrics such as precision, recall, and F-score assess the effectiveness of the classification. This work provides a structured template for applying k-NN in the varied classification scenarios while highlighting the simplicity and adaptability of this method.**

I. Introduction

Classification is an important task in machine learning, which involves assigning labels to data points based on their features. Among the foundational algorithms for this purpose is the **k-Nearest Neighbors (k-NN)** method, a non-parametric algorithm that classifies a data point based on the majority label among its *k* nearest neighbors. When k = 1, the algorithm reduces to the method of **Nearest Neighbor**, where the classification relies on the closest labeled data. This intuitive approach allows solving complex classification problems while remaining computationally efficient.

The **Reference Template Model** uses a different approach, where each class is represented by a template, often the mean or centroid of the feature values in that class. Classification is based on the distance from the data points to these templates. This model offers an elegant and interpretable approach to class separation, especially when observations are well-separated into distinct classes.

To assess the merits of these models, numerous performance metrics were used, including precision, recall, F-score, and accuracy. These metrics provide an integrated measure of the model's ability to predict class labels and handle incorrect classifications. Additionally, visualizations such as scatter plots, decision boundary plots, and classification plots help clarify the behavior and performance of these models.

II. Dataset Description

A. Linearly Separable Dataset

This dataset consists of points that can be perfectly separated by a straight line or hyperplane in the feature space. It serves as an ideal starting point for testing the performance of models on well-defined, simple boundaries. The dataset's simplicity allows for a clear understanding of how effectively the models classify data with linear decision boundaries.

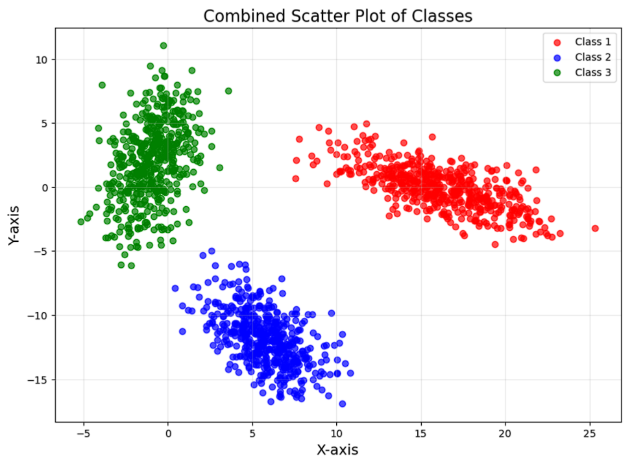


Fig. 1. Linearly Separable Dataset

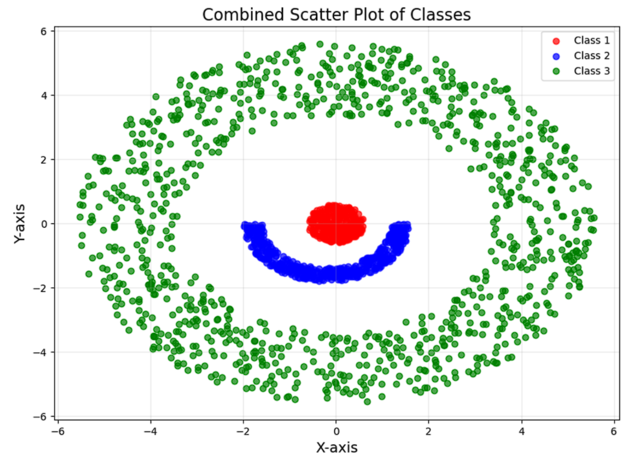


Fig. 2. Non-Linearly Separable Dataset

B. Non Linearly Separable Dataset

The second dataset is more complex, where no single straight line can separate the points. The decision boundaries are curved or intricate, and models must adapt to capture the data's non-linear relationships. This dataset tests how well models generalize beyond basic linear separability.

C. Real World Dataset

[The final dataset is for real-world scenarios, incorporating noisy and outlier data. Real-world data often contains irregularities, overlaps between classes, and inconsistent patterns, unlike artificially generated data. This dataset was used to evaluate how well the models handle imperfections and

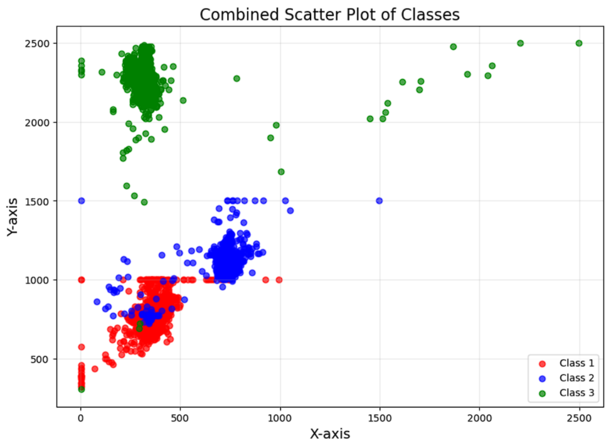


Fig. 3. Real-World Dataset

the complexity typical of real-world applications.

III. Nearest Neighbor Classification

A. Methodology

The basic NN algorithm is a very fundamental and intuitive machine learning technique and has applications in both classification and regression tasks. The essence of the principle of similarity it uses is to make predictions for an input data point by finding its closest neighbor(s) in the training dataset. The simplicity and non-parametric nature of the NN algorithm result in it being so general, so applicable to a lot of other kinds of problems, because it does not pose any assumptions concerning specific underlying distributions.

Algorithm Steps

1) Training Phase: Unlike most other machine learning models, there isn't an explicitly stated training phase for the NN algorithm. Instead:
   - The training data is simply a repository of feature values with their class labels or output values.
   - The stored data now serves as a reference against which the algorithm will evaluate new input points when predicting. This phase makes the NN algorithm light in weight and easy to implement but requires much more computational effort at prediction time.
2) Prediction Phase: The prediction process is as follows:
   - Distance Calculation: For a given input data point, the algorithm calculates its distance to every data point in the training dataset. In this implementation, the Euclidean distance is used as the distance metric. This metric is mathematically expressed as:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

     where:
     - p and q are two data points in n-dimensional space.

     The Euclidean distance measures the straight-line distance between two points in space, making it ideal for continuous-valued data.
   - Identifying Nearest Neighbors: Once the distances are computed, the algorithm identifies the closest data point(s) from the training set to the input point. For the basic NN algorithm, only one nearest neighbor is considered, while in its extension, k-NN, multiple nearest neighbors can be used to make a more robust prediction.
3) Making Predictions:
   - For classification tasks:
     - The class label of the nearest neighbor(s) determines the predicted class of the input point.
     - If k>1, majority voting among the k-nearest neighbors is used to decide the class.
   - For regression tasks:
     - The algorithm predicts the output as the average value of the nearest neighbor(s).

In summary, the $k = 1$ NN algorithm works well with linearly separable data, but its performance declines with non-linearly separable or noisy datasets. The 70-30 split allowed the model to be tested on a representative portion of the data, highlighting its strengths and limitations.

B. Predicted outputs and Decision boundary

The performance of the Nearest Neighbor (NN) algorithm was evaluated using three distinct types of datasets, each designed to assess the model's accuracy, adaptability, and efficiency in handling various data structures. These datasets were carefully selected to test the algorithm's robustness under different conditions, ranging from simple, well-structured data to complex, real-world scenarios. The evaluation covered:

1) **Linearly Separable Dataset**
   In this scenario, the dataset consisted of three classes that were linearly separable, ensuring minimal overlap between the classes. This simplicity in the dataset structure significantly enhanced the performance of the k-Nearest Neighbors (k-NN) algorithm. With k=1, the model classified each data point based on its closest neighbor, making the decision process straightforward and efficient.
   The well-separated nature of the classes enabled the k-NN algorithm to demonstrate optimal performance with high precision and accuracy. Each data point was assigned to the correct class with minimal computational effort. The decision boundary formed in this case was sharp and distinctly delineated, reflecting the separability of the data. These boundaries effectively partitioned the feature space into regions corresponding to each class, leaving no ambiguity in the classification results.
   The Euclidean distance metric was employed to calculate the straight-line distance between data points.

This metric proved to be effective in this context due to the simplicity and uniform distribution of the dataset. By considering the shortest path between points in a multidimensional space, the metric ensured that the nearest neighbor was correctly identified, which directly influenced the classification accuracy.

The model's computational efficiency and precision were particularly evident in this scenario. With only *k*=1, the algorithm avoided the potential complexities of considering multiple neighbors, thereby reducing the computational overhead. This simplicity allowed the algorithm to classify data points quickly, making it well-suited for real-time or resource-constrained applications in similar scenarios. Overall, the k-NN algorithm excelled in this scenario, leveraging the inherent simplicity of the dataset and the effectiveness of the Euclidean distance metric to achieve high-speed and accurate classification.

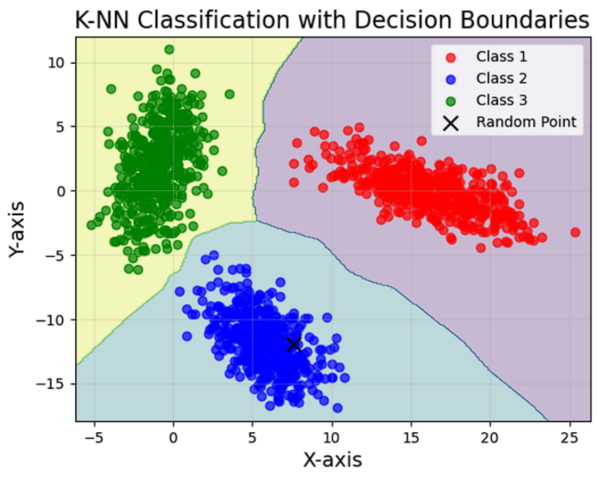


Fig. 4. Linearly Separable Dataset with Decision Boundary when *k*=1

2) **Non-linearly Separable Dataset**

The dataset posed significant complexity, with intertwined classes and non-linear relationships, making it impossible to separate them with a single hyperplane. For k=1, the k-NN algorithm classified points solely based on their nearest neighbor, resulting in highly irregular and fragmented decision boundaries that closely followed local data patterns.

While this approach allowed the model to capture fine-grained details, it also made it highly sensitive to noise, often misclassifying outliers or points near class boundaries. The use of the Euclidean distance metric, while effective in simple datasets, was less reliable in this scenario, especially in high-dimensional spaces, where the "curse of dimensionality" reduced its discriminative power.

The analysis revealed that with k=1, the algorithm prioritized local precision at the expense of overall stability and accuracy, making it less effective for non-linear and overlapping datasets.

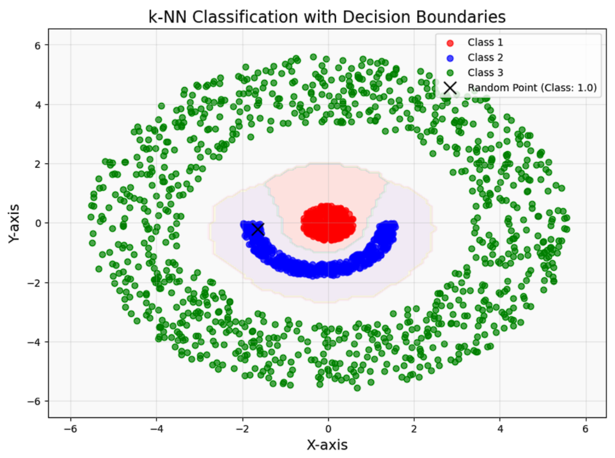


Fig. 5. Non-Linearly Separable Dataset with Decision Boundary when *k*=1

3) **Real-World Dataset**

The third dataset was sourced from a real-world scenario, containing noise, outliers, and other irregularities that are typically present in practical applications. This dataset posed the most significant challenge to the NN algorithm. The presence of noisy data and outliers affected the accuracy of the model, as the algorithm could be misled by incorrect neighbors, especially when the data points were far removed from the true decision boundary. However, the model was still able to classify the majority of points correctly. The decision boundary analysis for this dataset revealed how the noise and outliers influenced the boundary formation, causing slight distortions. Performance metrics like accuracy, precision, recall, and F-score were computed to quantitatively assess the model's performance in this real-world context. Despite the challenges, the algorithm's ability to adapt to noisy data and outliers was a testament to its robustness.

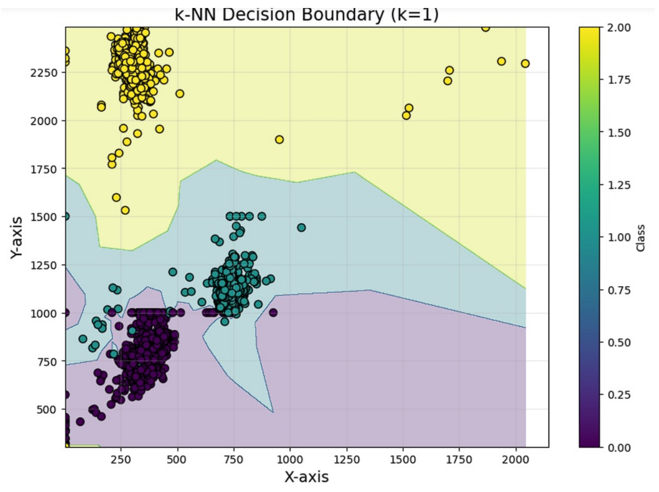


Fig. 6. Real World Dataset with Decision Boundary when *k*=1

C. Confusion Metrics

A confusion matrix for the k-Nearest Neighbor (k-NN) algorithm summarizes the performance of the model by comparing predicted and true class labels. It is a square matrix where the rows represent actual classes and the columns represent predicted classes. The entries show how many samples were correctly or incorrectly classified for each class.

Key metrics derived from the confusion matrix include:

- Accuracy: The proportion of correctly classified instances.
- Precision: The proportion of true positive predictions relative to all predicted positives.
- Recall: The proportion of true positive predictions relative to all actual positives.
- F-Score: The harmonic mean of precision and recall, balancing both metrics.
- Specificity: The proportion of true negatives relative to all actual negatives.

For k-NN with k=1, the algorithm directly assigns a class based on the nearest neighbor, making it sensitive to outliers. Its performance varies with dataset structure, excelling in clean, well-separated data and maintaining flexibility in more complex scenarios.

```
Class 1: Precision = 1.00, Recall = 1.00, F-Score = 1.00
Class 2: Precision = 1.00, Recall = 1.00, F-Score = 1.00
Class 3: Precision = 1.00, Recall = 1.00, F-Score = 1.00
```

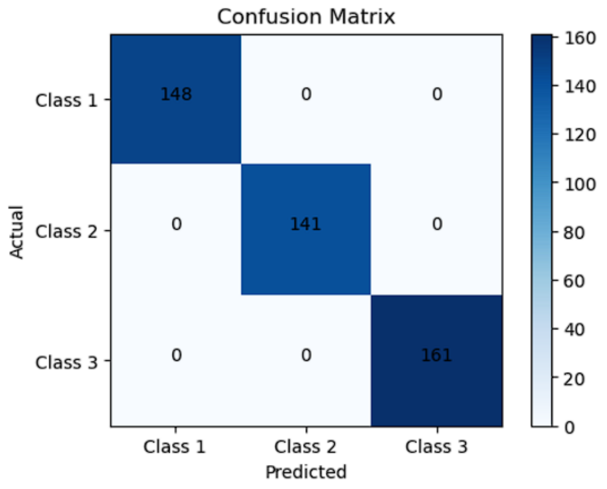


Fig. 7. Linearly Separable Dataset Metrics

IV. *k*-Nearest Neighbors Classification

A. Methodology

The K-Nearest Neighbors (KNN) algorithm is a fundamental and intuitive machine learning method used for both classification and regression tasks. It operates based on the principle of similarity, where the output for a given data point is predicted by considering the k nearest data points from the training set. Being a non-parametric algorithm, KNN does

```
Confusion Matrix:
[[ 80   0   0]
 [  0 157   0]
 [  0   0 303]]
Class 1: Precision = 1.00, Recall = 1.00, F-Score = 1.00
Class 2: Precision = 1.00, Recall = 1.00, F-Score = 1.00
Class 3: Precision = 1.00, Recall = 1.00, F-Score = 1.00
```

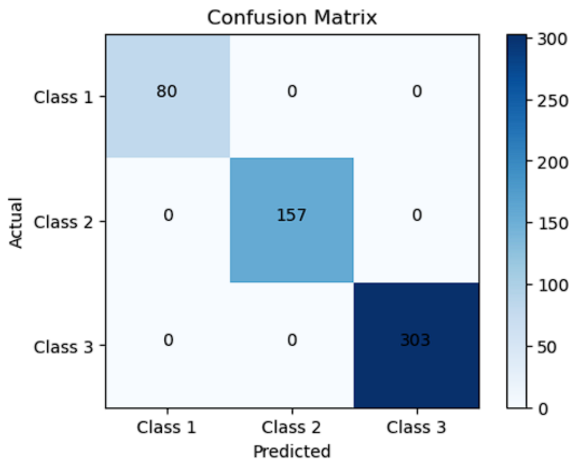


Fig. 8. Non-Linearly Separable Dataset Metrics

```
Class 1: Precision = 0.98, Recall = 0.98, F-Score = 0.98
Class 2: Precision = 0.98, Recall = 0.98, F-Score = 0.98
Class 3: Precision = 1.00, Recall = 1.00, F-Score = 1.00
```

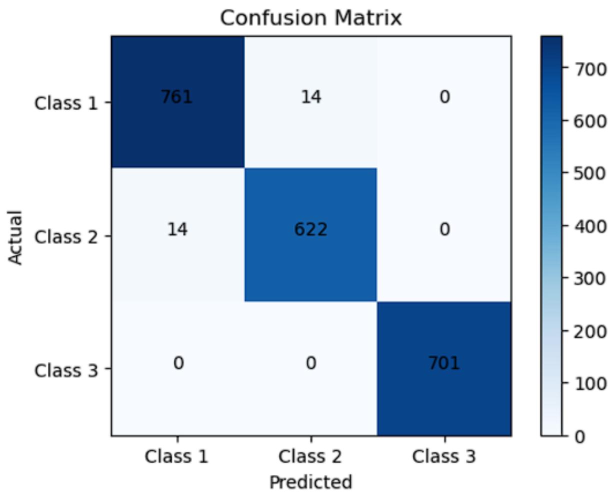


Fig. 9. Real World Dataset Metrics

not assume any specific data distribution, making it versatile and applicable across various domains.

Algorithm Steps

1) Training Phase: The KNN algorithm does not involve an explicit training process. Instead, the entire training dataset is stored and utilized during prediction. This characteristic makes KNN a lazy learner, as computation is deferred to the prediction phase.
   - **NOTE**: In this implementation, the value of k was set to varying values, adhering to the recommendation of using a odd and prime number for k.
2) Prediction Phase: The prediction process is as follows:

- Distance Calculation: For a given input data point, the algorithm calculates its distance to every data point in the training dataset. In this implementation, the Euclidean distance is used as the distance metric. This metric is mathematically expressed as:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

where:

  - p and q are two data points in n-dimensional space.

  The Euclidean distance measures the straight-line distance between two points in space, making it ideal for continuous-valued data.
- Identifying Nearest Neighbors: After computing distances, the algorithm sorts the data points in non-decreasing order of distance and identifies the k-nearest neighbors to the input point.

3) Making Predictions:
   - For classification tasks: The input point is assigned to the class most frequently represented among the k-nearest neighbors (majority vote).
   - For regression tasks: The predicted value is calculated as the average (or weighted average) of the values of the k-nearest neighbors.

B. *Predicted outputs and Decision Boundary*

The performance of the K-Nearest Neighbors (KNN) algorithm varies significantly depending on the structure and characteristics of the dataset. By analyzing its behavior on different types of datasets, we can better understand its strengths, limitations, and adaptability.

1) **Linearly Separable Dataset**
The value of k in the K-Nearest Neighbors (KNN) algorithm significantly influences the nature of the decision boundary, which directly affects the model's classification behavior. For smaller values of k (e.g., $k = 11$), the decision boundary closely follows the local structure of the dataset, resulting in sharper and more detailed separations between classes. However, this can make the model sensitive to noise and outliers, as individual data points heavily influence predictions. As k increases (e.g., $k = 37$), the decision boundary becomes smoother and more generalized, as predictions are based on a larger neighborhood of points. This reduces the impact of noise but can obscure finer patterns, especially near class overlaps. In plots, smaller k produces intricate boundaries that adapt to specific data points, while larger k creates smoother, more stable divisions. The choice of k thus involves a trade-off between precision and robustness, requiring careful tuning to balance overfitting and underfitting based on the dataset's characteristics.
We observe from Fig. 10. and Fig. 11. that as the value of k changes from 11 to 37, the decision boundary linear separation becomes more smooth.

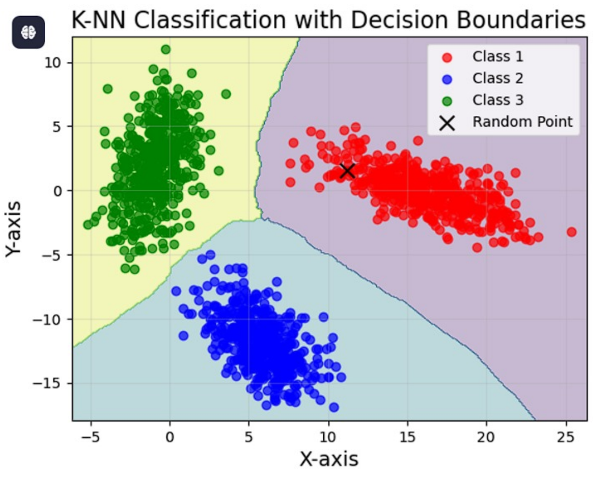


Fig. 10. Linearly Separable Dataset with Decision Boundary when k=11

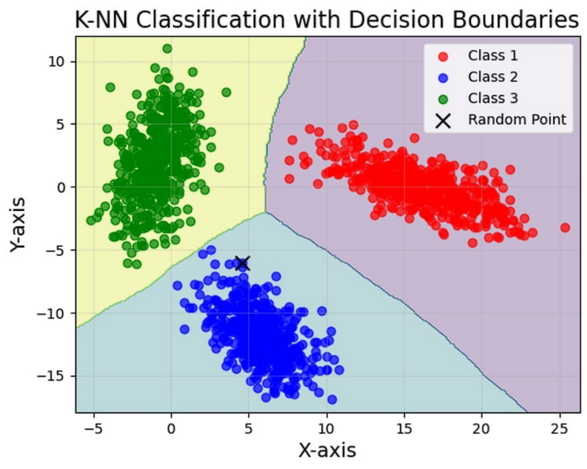


Fig. 11. Linearly Separable Dataset with Decision Boundary when k=37

2) **Non-Linearly Separable Dataset**
In non-linearly separable datasets, where classes are intertwined or exhibit complex and overlapping patterns, the KNN algorithm adapts by forming irregular decision boundaries to approximate the local data distributions. This flexibility allows KNN to capture non-linear relationships, making it capable of handling intricate datasets. However, the increased complexity poses challenges. With $k = 11$, the algorithm smooths the decision boundary by considering a broader neighborhood of points for classification. This approach helps mitigate the effects of noise and outliers, as predictions rely on a consensus of multiple neighbors rather than individual points. Nevertheless, the smoothing effect can result in misclassifications, particularly near class boundaries, where local patterns may conflict with global trends. For example, in regions with high overlap between classes, the inclusion of distant neighbors from a different class can skew predictions, reduc-

ing local precision. While KNN remains adaptable in these scenarios, the overall classification accuracy often declines compared to simpler, linearly separable datasets, highlighting the trade-off between robustness and sensitivity in complex data environments.

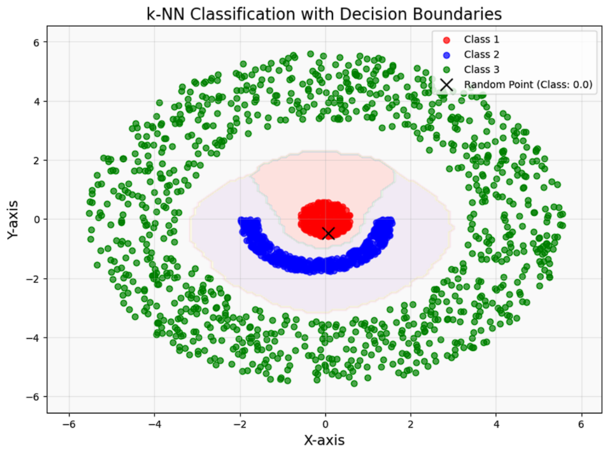


Fig. 12. Non-Linearly Separable Dataset when k=11

3) **Real-World Dataset**
In real-world datasets, the K-Nearest Neighbors (KNN) algorithm encounters significant challenges due to the inherent complexity and imperfections of the data. These datasets often contain overlapping class boundaries, missing values, irrelevant features, and imbalanced class distributions, all of which can impact the algorithm's performance. KNN relies on distance metrics like Euclidean distance, which makes it sensitive to the scaling of features. If features are not normalized, those with larger magnitudes may dominate the distance calculations, leading to biased predictions. For $k = 11$, the algorithm averages over a broader set of neighbors, reducing the influence of individual noisy data points and outliers. While this smoothing effect enhances robustness, it does not completely eliminate the impact of outliers or irrelevant features, which can still lead to misclassifications. Additionally, imbalanced data can skew predictions toward the majority class, reducing the algorithm's effectiveness for minority classes. Despite these challenges, KNN remains a viable choice for real-world scenarios, especially when paired with preprocessing techniques such as feature selection to eliminate irrelevant variables, normalization to ensure uniform scaling, and imputation methods to handle missing data. These steps help optimize KNN's performance, making it a flexible tool for various real-world applications.

Across datasets, KNN's performance is heavily influenced by the choice of k and the distance metric. Smaller values of k may result in overfitting, while larger values provide robustness but risk oversmoothing the decision boundary. For structured datasets, KNN achieves high accuracy and efficiency, while for real-world or complex data, preprocessing and careful parameter tuning are essential to optimize its performance. Despite its limitations, KNN's simplicity and versatility make it a popular algorithm across a wide range of applications.

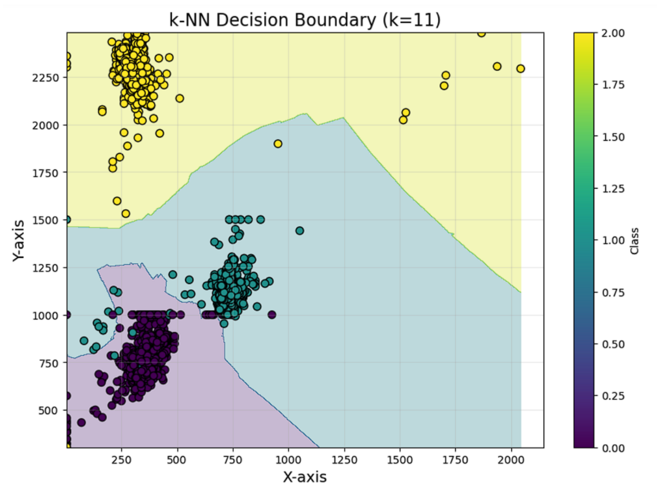


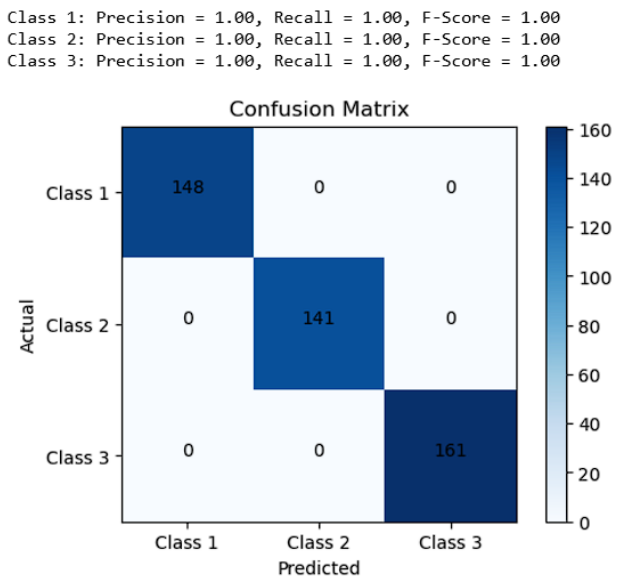
Fig. 13. Real World Dataset when k=11

C. Confusion Metrics

A confusion matrix for the k-Nearest Neighbor (k-NN) algorithm provides a comprehensive summary of the model's classification performance by comparing the predicted class labels with the true class labels. It is a square matrix, with each row corresponding to the actual classes in the dataset and each column representing the predicted classes. The matrix entries indicate the number of instances for each combination of actual and predicted labels, making it easy to identify how many samples were correctly classified (true positives and true negatives) and where the model made errors (false positives and false negatives). For multi-class datasets, the confusion matrix expands to include all classes, allowing detailed analysis of performance across each class. This visualization helps pinpoint strengths and weaknesses of the model, such as which classes are often confused with one another. It also serves as the basis for calculating key metrics like accuracy, precision, recall, F-score, and specificity, offering deeper insights into the model's effectiveness and reliability.

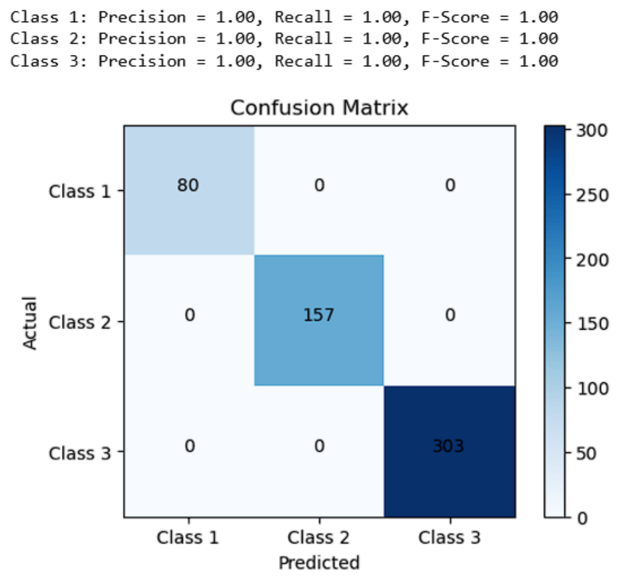
Key metrics derived from the confusion matrix include:

- Accuracy: The proportion of correctly classified instances.
- Precision: The proportion of true positive predictions relative to all predicted positives.
- Recall: The proportion of true positive predictions relative to all actual positives.
- F-Score: The harmonic mean of precision and recall, balancing both metrics.
- Specificity: The proportion of true negatives relative to all actual negatives.



```
Class 1: Precision = 1.00, Recall = 1.00, F-Score = 1.00
Class 2: Precision = 1.00, Recall = 1.00, F-Score = 1.00
Class 3: Precision = 1.00, Recall = 1.00, F-Score = 1.00
```

Fig. 14. Linearly Separable Dataset Metrics for k-NN



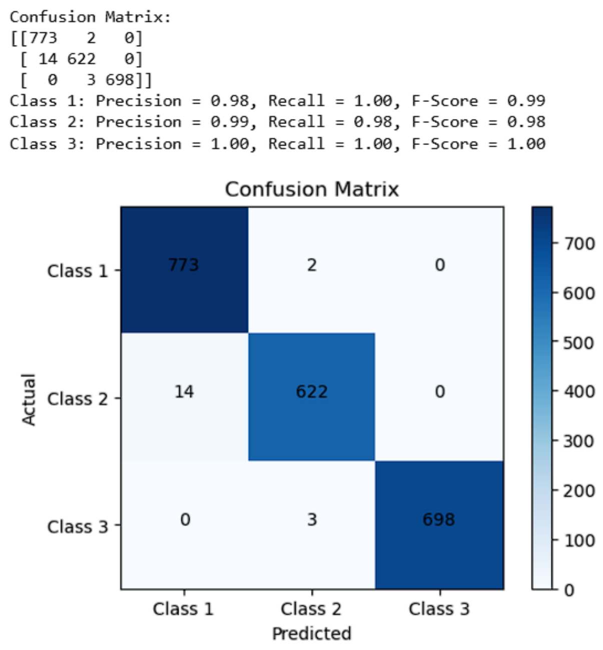
```
Class 1: Precision = 1.00, Recall = 1.00, F-Score = 1.00
Class 2: Precision = 1.00, Recall = 1.00, F-Score = 1.00
Class 3: Precision = 1.00, Recall = 1.00, F-Score = 1.00
```

Fig. 15. Non-Linearly Separable Dataset Metrics for k-NN

D. Summary

The k-Nearest Neighbors (k-NN) algorithm is a versatile and non-parametric method that assigns class labels by considering the k closest data points in the feature space. Its performance is highly dependent on the choice of k and the structure of the dataset. For smaller k values, the algorithm is sensitive to noise and outliers, as predictions are driven by very local patterns. Larger k values help smooth the decision boundary by incorporating a broader neighborhood, which can enhance robustness but may also lead to a loss of precision in regions with overlapping classes.

KNN excels in clean, well-separated datasets where the relationships between classes are straightforward, producing sharp and accurate decision boundaries. However, in more



```
Confusion Matrix:
[[773   2   0]
 [ 14 622   0]
 [  0   3 698]]
Class 1: Precision = 0.98, Recall = 1.00, F-Score = 0.99
Class 2: Precision = 0.99, Recall = 0.98, F-Score = 0.98
Class 3: Precision = 1.00, Recall = 1.00, F-Score = 1.00
```

Fig. 16. Real World Dataset Metrics for k-NN

complex or real-world scenarios—characterized by noise, overlapping classes, or imbalanced distributions—its performance may decline without proper preprocessing. Techniques like feature scaling, normalization, and imputation for missing values are crucial for improving its accuracy. Despite its simplicity, KNN's flexibility makes it suitable for both linear and non-linear classification tasks, providing a reliable benchmark in machine learning while demonstrating adaptability across varied data conditions.

V. REFERENCE TEMPLATE METHOD

A. Methodology

Calculate the Distance Using Mahalanobis Distance

To implement the Reference Template algorithm, we first calculate the distance between the input data point and each reference template using the Mahalanobis distance metric. The steps involved in this calculation are as follows:

1. **Compute the Mean Vector** μ: The mean vector of the dataset, which represents the center of the data distribution, is calculated as:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

where x_i is the i-th data point in the dataset and n is the total number of data points. This vector captures the central tendency of the dataset.

2. **Compute the Covariance Matrix** S: The covariance matrix is a measure of how much each feature in the dataset varies with respect to each other. It is computed as:

$$S = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T$$

where $(x_i - \mu)$ is the deviation of each data point from the mean vector. The covariance matrix S is symmetric and encapsulates the variance and correlation between each pair of features.

3. **Calculate the Inverse of the Covariance Matrix** S^{-1}: The inverse of the covariance matrix is necessary for computing the Mahalanobis distance. It is calculated as:

$$S^{-1}$$

which is used to normalize the distances and account for the correlations between the features of the dataset.

4. **Compute the Mahalanobis Distance**: The Mahalanobis distance between two points x_1 and x_2 is calculated as:

$$d(x_1, x_2) = \sqrt{(x_1 - x_2)^T S^{-1} (x_1 - x_2)}$$

This distance measure accounts for the shape of the data distribution by incorporating the covariance of the dataset, making it more robust to feature correlations compared to the Euclidean distance.

5. **Sort Reference Templates and Identify Nearest Templates**

Once the distances between the input data point and each reference template are computed, we sort these reference templates in non-decreasing order of their calculated distances. This allows us to identify the closest reference templates. The nearest template(s) are selected based on the smallest Mahalanobis distances. This sorting step is crucial for accurately identifying the reference templates that are most similar to the input point.

6. **Classification Based on Nearest Reference Template(s)**

For classification, the input point is predicted to belong to the class of the nearest reference template(s). In cases where multiple templates are equidistant from the input point, a strategy is employed to make a final decision. Common strategies include:

- Randomly selecting one of the closest templates.
- Selecting the template based on predefined criteria, such as choosing the template that is most frequent among the nearest neighbors.

This step involves assigning the class label of the closest template(s) to the input point, completing the classification process. If multiple templates belong to different classes, a majority voting system or weighted decision approach can be employed to determine the final class of the input data point.

B. Predicted Outputs and Decision Boundary

The performance of the Reference Template method varies based on the dataset's complexity and the number of reference templates used. By analyzing its behavior on different types of datasets, we can gain a deeper understanding of its strengths and limitations.

1) **Linearly Separable Dataset**

The Reference Template algorithm relies on calculating the distance between the input point and each reference template to determine its class. In the case of linearly separable datasets, the decision boundary created by the Reference Template method can be sharp and well-defined, closely matching the natural divisions between classes. When using a small number of reference templates, the decision boundary can be more sensitive to local variations in the data, resulting in highly specific classifications that closely follow the structure of the dataset. However, this sensitivity can lead to overfitting, especially in regions with sparse data or noise. Increasing the number of reference templates results in a smoother decision boundary, as more templates are considered for classification. This smoother boundary helps reduce the impact of noise or outliers but may sacrifice the precision needed to distinguish between closely spaced classes. The choice of the number of reference templates influences the balance between local precision and global generalization, requiring careful consideration based on the dataset's complexity and the desired level of specificity.

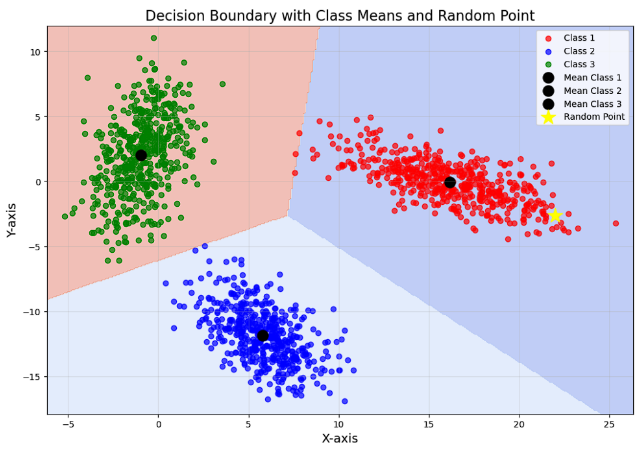


Fig. 17. Linearly Separable Dataset with Reference Template Method

2) **Non-Linearly Separable Dataset**

In non-linearly separable datasets, where classes are intertwined or exhibit complex relationships, the Reference Template method adapts by forming irregular decision boundaries that attempt to capture the local data structure. However, this flexibility also presents challenges, as the algorithm can struggle to capture global patterns due to the intricacy of class distributions. When fewer reference templates are used, the decision boundary may become overly sensitive to local variations, leading to jagged and complex boundaries that fit the specific training data but fail to generalize well. On the other hand, using a larger number of reference templates smooths out the decision boundary, helping to reduce the model's sensitivity to noise and outliers. However, this increased smoothing may result in mis-classifications, especially near regions where classes are closely overlapping or where class boundaries are not well-defined. The ability to balance between capturing local patterns and ensuring robustness in such complex datasets requires careful tuning of the number of reference templates, as well as pre-processing steps to reduce noise and better define class boundaries.

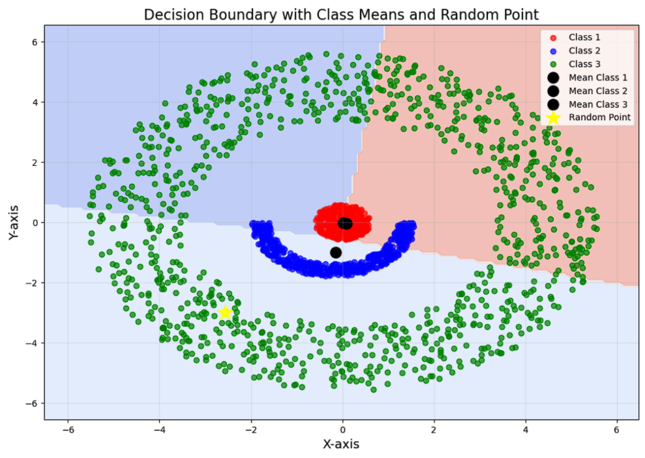


Fig. 18. Non-Linearly Separable Dataset with Reference Template Method

3) **Real-World Dataset**
In real-world datasets, which often contain noise, imbalanced class distributions, missing values, and irrelevant features, the Reference Template method faces several challenges that can impact its performance. These datasets typically exhibit non-linear relationships and overlapping class distributions, which complicate the task of defining well-separated decision boundaries. The presence of noise and outliers can significantly influence the calculation of distances, especially when fewer reference templates are used. This may lead to erratic decision boundaries that become overly sensitive to local variations in the data, making the model more prone to overfitting.
When fewer reference templates are considered (i.e., smaller template sets), the algorithm may adapt too closely to local fluctuations, creating highly detailed decision boundaries that are not generalizable. On the other hand, increasing the number of reference templates (e.g., considering a larger neighborhood) can reduce the influence of noise and outliers by averaging over a broader set of templates. This results in a smoother, more stable decision boundary, but it also causes the model to potentially miss finer nuances in the data structure, particularly when class boundaries are complex or contain subtle distinctions.
In addition, the calculation of distances in the presence of imbalanced class distributions can exacerbate the model's bias toward majority classes. This is because the model tends to favor the nearest reference templates, which may belong predominantly to the majority class, leading to skewed predictions. To mitigate this, techniques such as distance weighting, where closer reference templates are given more importance, or applying synthetic data generation methods like SMOTE (Synthetic Minority Over-sampling Technique), can be used to address class imbalance.
Furthermore, missing values and irrelevant features can distort the distance metrics, as they lead to inaccurate distance calculations, especially when the dataset contains categorical, sparse, or incomplete data. Feature scaling (e.g., normalization or standardization) is crucial to prevent certain features from dominating the distance calculations due to their larger magnitudes. Missing values can be handled through imputation strategies, such as using the mean, median, or a more sophisticated method like k-nearest neighbors imputation.
Despite these challenges, with proper data preprocessing—such as feature selection to remove irrelevant features, imputation for missing values, and normalization to standardize feature scales—the performance of the Reference Template method in real-world scenarios can be significantly improved. The key to effective application lies in balancing robustness (resilience to noise and outliers) and specificity (ability to capture meaningful, detailed data patterns). Thus, careful tuning of the number of reference templates and the pre-processing steps are essential for achieving high performance on complex, real-world datasets.

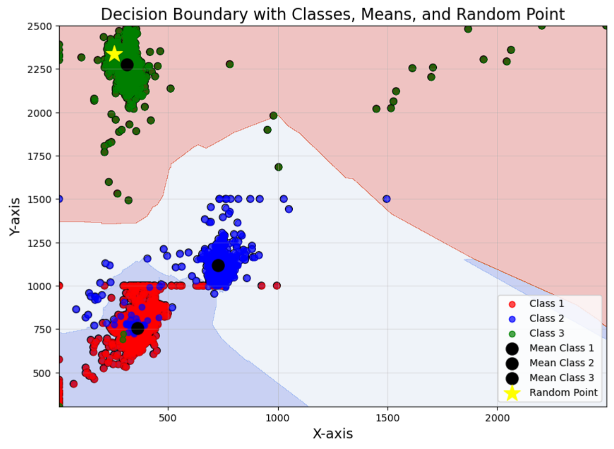


Fig. 19. Real World Dataset with Reference Template Method

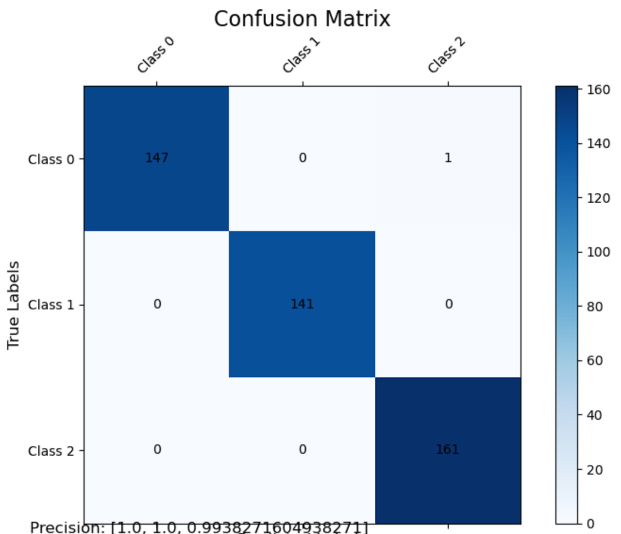
C. Confusion Metrics

A confusion matrix for the Reference Template method summarizes the model's classification performance by comparing predicted and actual class labels. It is a square matrix where rows represent the true classes and columns represent the predicted classes. The matrix entries indicate the number of instances for each combination of actual and predicted labels, showing how many instances were correctly classified (true positives and true negatives) and where errors occurred (false positives and false negatives).

In multi-class classification, the matrix expands to include all classes, highlighting which classes are most often confused. This analysis helps identify the model's strengths and weaknesses. The confusion matrix also forms the basis for calculating important metrics such as accuracy, precision, recall, F-score, and specificity, providing valuable insights into the model's overall effectiveness and areas for improvement.

Key metrics derived from the confusion matrix include:

- Accuracy: The proportion of correctly classified instances.
- Precision: The proportion of true positive predictions relative to all predicted positives.
- Recall: The proportion of true positive predictions relative to all actual positives.
- F-Score: The harmonic mean of precision and recall, balancing both metrics.
- Specificity: The proportion of true negatives relative to all actual negatives.



```
Precision: [1.0, 1.0, 0.9938271604938271]
Recall: [0.9932432432432432, 1.0, 1.0]
F1 Score: [0.9966101694915254, 1.0, 0.9969040247678018]
```

Fig. 20. Linearly Separable Dataset Metrics for Reference Template

VI. CONCLUSION

Based on the analysis of accuracy scores and decision boundary behavior for three algorithms—Nearest Neighbor (NN), K-Nearest Neighbor (KNN), and Reference Template—applied to three distinct datasets, several key insights emerged:

1. *Performance on Linearly and Non-Linearly Separable Datasets*: All three algorithms demonstrated strong performance on both linearly and non-linearly separable datasets. Their ability to adapt to the structure of the data allowed them to correctly classify instances, although the complexity of the decision boundaries varied based on the nature of the dataset and the chosen algorithm.

2. *Performance on Real-World Dataset*: The real-world dataset posed challenges due to the presence of significant noise and outliers. Despite these challenges, all three algorithms showed good accuracy. As expected, the decision boundary became progressively clearer and more stable as we moved from the Nearest Neighbor algorithm to the Reference Template. The Reference Template's ability to smooth over noise by averaging across multiple reference points made it particularly effective in dealing with real-world data complexities.

```
Precision: [0.5862068965517241, 0.4227941176470588, 0.6906077348066298]
Recall: [0.6375, 0.732484076433121, 0.41254125412541254]
F1-Score: [0.6107784431137724, 0.5361305361305361, 0.5165289256198347]
```

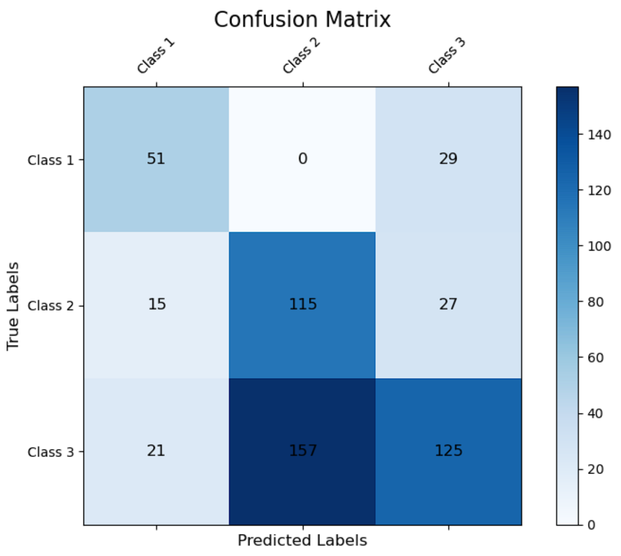
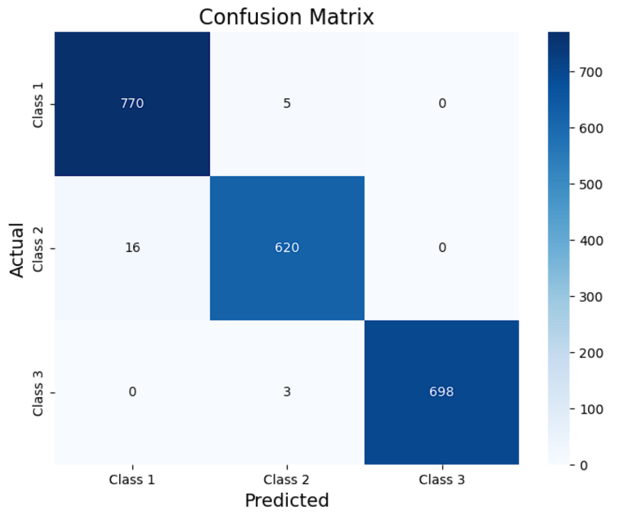


Fig. 21. Non-Linearly Separable Dataset Metrics for Reference Template



```
Precision for each class: [0.9796437659033079, 0.9872611464968153, 1.0]
Recall for each class: [0.9935483870967742, 0.9748427672955975, 0.9957203994293866]
F1 Score for each class: [0.9865470852017938, 0.9810126582278481, 0.997855611150822]

Overall Weighted Precision: 0.99
Overall Weighted Recall: 0.99
Overall Weighted F1 Score: 0.99
```

Fig. 22. Real World Dataset Metrics for Reference Template

3. *Impact of Noise on Precision and Recall*: As accuracy increased on the real-world dataset, a slight decrease in precision and recall was observed for certain classes. This suggests that while the algorithms were successful in identifying the majority patterns, they were also effectively

ignoring outliers. This trade-off between accuracy and the performance metrics highlights the challenges of achieving high classification accuracy while maintaining balance across all classes in the presence of noisy data.

4. *Efficiency Analysis*: The Reference Template algorithm, while robust against noise, is computationally more expensive than KNN or Nearest Neighbor. It calculates the mean of every row in the dataset and computes distances based on these means rather than individual data points, which contributes to its higher computational cost. The time complexity of the Reference Template is approximately $O(n^3)$, while KNN operates with a time complexity of $O(n)$, making KNN much more efficient for large datasets.

5. *Recommendations*:

- For simple, well-structured datasets with minimal noise, the *Nearest Neighbor* or *K-Nearest Neighbor* algorithms are recommended due to their computational efficiency and simplicity. These algorithms are quick to train and test, making them ideal for less complex problems.
- For more complex datasets that contain significant noise and outliers, the *Reference Template* algorithm is preferable. Despite its higher computational cost, it excels in handling noisy data patterns and improving the overall classification accuracy by averaging over a larger set of templates.

In conclusion, the choice of algorithm depends heavily on the nature of the dataset and the specific trade-offs between accuracy, computational efficiency, and noise robustness. For simpler, less noisy datasets, KNN or NN are ideal, while for challenging real-world datasets with noise and outliers, the Reference Template method provides a more stable and accurate solution.