Hüseyin Eren YILDIZ (31047)
CS412 Homework1

Notebook Link:
https://colab.research.google.com/drive/1Ewwugip2m3pUNGT3sc_W8MTfAyPUpR3l?usp=sharing

INTRODUCTION

In this assignment, I implement and evaluate two classification algorithms on the MNIST dataset:

- k-Nearest Neighbors (k-NN)
- Decision Tree

The MNIST dataset consists of 28×28 grayscale images representing handwritten digits (0-9). The goal is to develop and assess these classifiers based on their ability to accurately recognize and categorize handwritten digits.

This report details the data preprocessing, model training, hyperparameter tuning, and evaluation of both models, highlighting their strengths and weaknesses in terms of classification accuracy and efficiency.

2. Dataset and Preprocessing

2.1 Data Loading

The MNIST dataset is loaded using the Keras API. It contains:

- 60,000 training images
- 10,000 test images

To enhance evaluation, the training set is further split into:

- 80% Training set (48,000 images)
- 20% Validation set (12,000 images)

Data Loading and Splitting

```
[ ]
#Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Split into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=0, stratify=y_train)

class_names = ["0", "1", "2", "3", "4","5", "6", "7", "8", "9"]

#Print the shapes of the datasets
print(f"Training set shape: {x_train.shape}, Labels: {y_train.shape}")
print(f"Validation set shape: {x_val.shape}, Labels: {y_val.shape}")
print(f"Test set shape: {x_test.shape}, Labels: {y_test.shape}")
```

```
Training set shape: (48000, 28, 28), Labels: (48000,)
Validation set shape: (12000, 28, 28), Labels: (12000,)
Test set shape: (10000, 28, 28), Labels: (10000,)
```

These outputs represent the shapes of the datasets used for training and evaluating the model. The **training set** consists of 48,000 images, each with a size of 28x28 pixels, and is used to train the model. During this process, the model learns patterns and updates its weights to improve digit recognition. The **validation set** contains 12,000 images of the same size and is used to check whether the model is overfitting during training. By evaluating the model on the validation set, we can determine if it performs well on unseen data and adjust parameters accordingly. Finally, the **test set** consists of 10,000 images and is used after training is complete to measure the model's real-world performance on entirely new data. These three datasets ensure that the model is properly trained, validated, and tested for reliable performance.

2.2 Data Analysis

a) Class Distribution

This output represents the distribution of different digit classes in the training dataset (y_train), showing how many times each digit from 0 to 9 appears. The code extracts unique class labels and their respective counts using **np.unique(y_train, return_counts=True)** and stores the results in a dictionary for better readability. From the output, we can see that some digits, such as **1 (5,394 occurrences) and 7 (5,012 occurrences),** appear more frequently than others, like **4 (4,674 occurrences) and 5 (4,337 occurrences).** Understanding this distribution is crucial in machine learning because an imbalanced dataset—where some classes are significantly overrepresented—can lead to biased model predictions (**class imbalance problem**). However, in this case, the dataset appears relatively balanced, meaning the model should be able to learn to recognize all digits without significant bias toward any particular class.

Class Distribution

```
[ ]  1  unique, counts = np.unique(y_train, return_counts=True)
     2  class_distribution = dict(zip(unique, counts)) # Storing the values in a dictionary
     3  print("Class Distribution:", class_distribution)
```

```
Class Distribution: {0: 4738, 1: 5394, 2: 4766, 3: 4905, 4: 4674, 5: 4337, 6: 4734, 7: 5012, 8: 4681, 9: 4759}
```

b) Basic Statistics

This analysis provides insights into the statistical properties of the MNIST dataset across its training, validation, and test sets.

```python
import numpy as np

#Data type and pixel value range
print("Data type of X_train:", x_train.dtype)
print("Min pixel value - Training set:", np.min(x_train))
print("Max pixel value - Training set:", np.max(x_train))
print("Unique classes in y_train:", np.unique(y_train))

#Calculate the mean and standard deviation of the pixel values
#for each dataset
mean_train = np.mean(x_train)
std_train = np.std(x_train)

mean_val = np.mean(x_val)
std_val = np.std(x_val)

mean_test = np.mean(x_test)
std_test = np.std(x_test)

#Printing final results
print("\nMean pixel value - Training set:", mean_train)
print("Standard deviation - Training set:", std_train)
print("\nMean pixel value - Validation set:", mean_val)
print("Standard deviation - Validation set:", std_val)
print("\nMean pixel value - Test set:", mean_test)
print("Standard deviation - Test set:", std_test)

```

```
Data type of X_train: uint8
Min pixel value - Training set: 0
Max pixel value - Training set: 255
Unique classes in y_train: [0 1 2 3 4 5 6 7 8 9]

Mean pixel value - Training set: 33.28194106079932
Standard deviation - Training set: 78.5240131616269

Mean pixel value - Validation set: 33.46434300595238
Standard deviation - Validation set: 78.74098820215275

Mean pixel value - Test set: 33.791224489795916
Standard deviation - Test set: 79.17246322228644
```

1. Data Type and Pixel Value Range

Data type of X_train: uint8 → The images are stored in 8-bit unsigned integer (uint8) format, meaning pixel values range from 0 to 255.

Min pixel value - Training set: 0 → The lowest pixel value is 0, which represents completely black pixels.

Max pixel value - Training set: 255 → The highest pixel value is 255, which represents completely white pixels.

Unique classes in y_train: [0 1 2 3 4 5 6 7 8 9] → The dataset contains all digits from 0 to 9, confirming that all classes are well represented.

These values confirm that the MNIST dataset consists of grayscale (black and white) images, where pixel intensities vary between black (0) and white (255).

2. Mean Pixel Values

The mean pixel value for the training set is **33.28**, while the validation and test sets have similar values of **33.46 and 33.79**, respectively. These mean values indicate that the images are **generally dark**, as the average pixel intensity is significantly lower than 127 (the midpoint of the 0-255 range). Since MNIST images contain white handwritten digits on a black background, it is expected that most pixels in the dataset are dark.

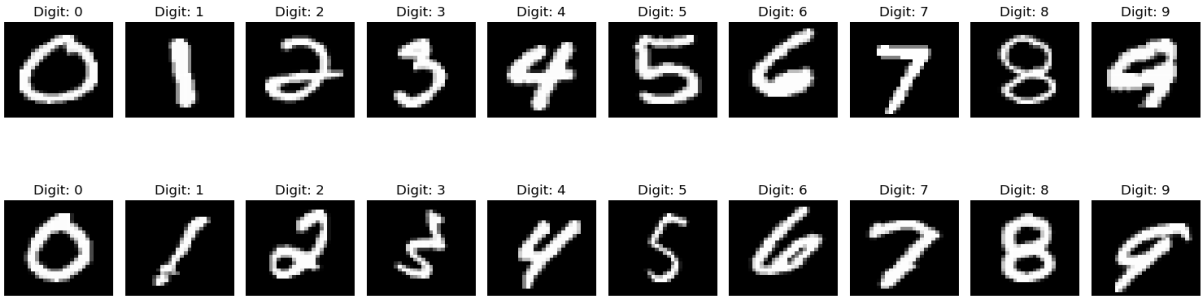
3. Standard Deviation (Pixel Intensity Spread)

The standard deviation values indicate how widely pixel values are spread around the mean. A standard deviation around **78-79** suggests a **wide range of brightness levels** in the dataset, meaning some images contain very dark backgrounds while others might be lighter.

4. Dataset Consistency

The mean and standard deviation values are highly similar across the training, validation, and test sets, indicating that the dataset is well-structured and maintains a consistent distribution across all three subsets. If there were significant differences in pixel statistics between the training and test sets, the model might perform well on the training data but struggle with test data due to distribution shift, potentially reducing its generalization ability. This analysis confirms that the MNIST dataset is well-balanced and consistent, ensuring that the model can learn effectively from training data and apply its knowledge to unseen test data. Given this distribution, applying normalization (scaling pixel values between 0 and 1) could improve model performance by making pixel values more uniform and easier for the model to process and learn from.

c) Visualization



This visualization presents a grid structure showing two different examples of each digit in the MNIST dataset. The code selects the first available image for each digit from 0 to 9 in the training data and arranges them sequentially on a horizontal axis. The images are 28x28 pixels in size and consist of white handwritten digits on a black background. Since these images contain various handwriting styles, the model needs to learn to recognize different writing patterns. While some digits appear clear and well-defined, others may be tilted, bold, or pixelated, highlighting the diversity within the dataset. Due to the low resolution, some digits look slightly blurry or blocky, which may pose challenges for the model in distinguishing certain numbers. Additionally, the code's ability to identify and display each digit ensures that the dataset is complete and well-balanced. Overall, this visualization provides a visual representation of the data the model will be trained on, helping to identify potential challenges related to variations in handwriting styles and image quality.

2.3 Data Preprocessing

```python
# Compute mean and standard deviation across the entire dataset
mean = np.mean(x_train)
std = np.std(x_train)

print("Mean of training set:", mean)
print("Standard deviation of training set:", std)

# Apply normalization: (X - mean) / std
X_train_norm = (x_train - mean) / std
X_val_norm = (x_val - mean) / std
X_test_norm = (x_test - mean) / std

# Verify normalization
print("\nMean before normalization:", np.mean(x_train))
print("Std before normalization:", np.std(x_train))

print("\nMean after NumPy normalization (should be ~0):", np.mean(X_train_norm))
print("Std after NumPy normalization (should be ~1):", np.std(X_train_norm))
```

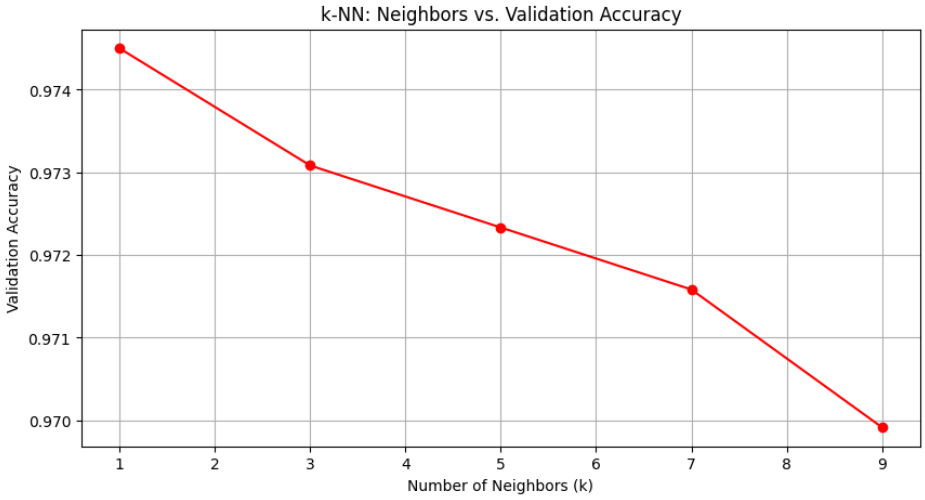
```
Mean of training set: 33.28194106079932
Standard deviation of training set: 78.5240131616269

Mean before normalization: 33.28194106079932
Std before normalization: 78.5240131616269

Mean after NumPy normalization (should be ~0): -2.0636931317258563e-17
Std after NumPy normalization (should be ~1): 1.0000000000000013
```

This performs the normalization process on the MNIST dataset, ensuring that the data is properly scaled for improved model learning. Initially, the mean pixel value (33.28) and standard deviation (78.52) of the training dataset are computed, providing insights into the overall distribution of pixel intensities. The normalization is then applied by subtracting the mean from each pixel value and dividing by the standard deviation, resulting in a dataset with a mean of zero and a standard deviation of one. This process is crucial for preventing differences in scale from negatively affecting the model's learning process. The verification step confirms that the normalization was applied correctly, as the new mean is nearly zero (-2.06e-17), and the standard deviation is very very close to 1. These results indicate that the dataset has been successfully normalized, making it more suitable for machine learning algorithms. Normalization improves model stability, speeds up training, and reduces errors caused by varying feature scales. In conclusion, the normalization process has been successfully completed, and the dataset is now well-prepared for training

3. k-NN Classifier

3.1 Model Initialization and Hyperparameter Tuning

This model implements and evaluates a **k-Nearest Neighbors (k-NN) classifier** by testing different values of **k** and determining which one provides the best validation accuracy. The k-NN algorithm classifies a sample based on its k nearest neighbors, making the selection of an optimal k value crucial for model performance and generalization. The model is trained and tested for k values of 1, 3, 5, 7, and 9, and the corresponding accuracy scores are recorded. The results indicate that k=1 achieves the highest validation accuracy of 97.45%, while increasing k slightly reduces accuracy. This behavior aligns with the nature of k-NN: when k=1, the model relies on a single nearest neighbor, which can lead to overfitting, making it more sensitive to noise. Conversely, larger k values (e.g., k=5 or k=9) consider more neighbors, producing smoother and more stable predictions but potentially reducing accuracy. The best k value is determined dynamically by comparing accuracy scores, and k=1 is selected as the optimal choice for this dataset.




```
Validation accuracy for k=1: 0.9745
Validation accuracy for k=3: 0.9731
Validation accuracy for k=5: 0.9723
Validation accuracy for k=7: 0.9716
Validation accuracy for k=9: 0.9699

Best k=1, Best Validation Accuracy: 0.9745
```

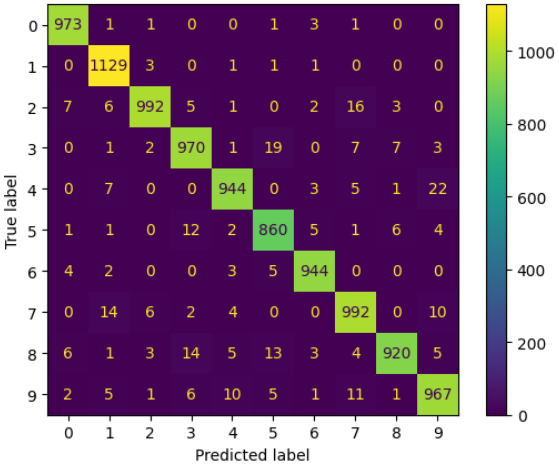
The graph illustrates the relationship between the number of neighbors (k) and the validation accuracy of the k-Nearest Neighbors (k-NN) classifier. The highest accuracy, approximately 0.974, is observed at k = 1, indicating that the model performs best when considering only the nearest neighbor. However, relying on a single neighbor can lead to overfitting, as the model becomes highly sensitive to noise in the dataset. As ‘k’ increases, the validation accuracy gradually declines, reaching 0.973 at k = 3, 0.972 at k = 5, and continuing to decrease until it reaches its lowest point of 0.970 at k = 9.

The highest validation accuracy (~0.974) is observed at k = 1, indicating that the **optimal k value, based on validation accuracy, is k = 1**. Since the validation set is used to determine the best-performing model, k = 1 should be selected as it provides the highest accuracy.

3.2 Final Model Training and Evaluation

```
Test Accuracy with Best Hyperparameters: 0.9691

Classification Report for KNN:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      0.99      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.95      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.98      0.94      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

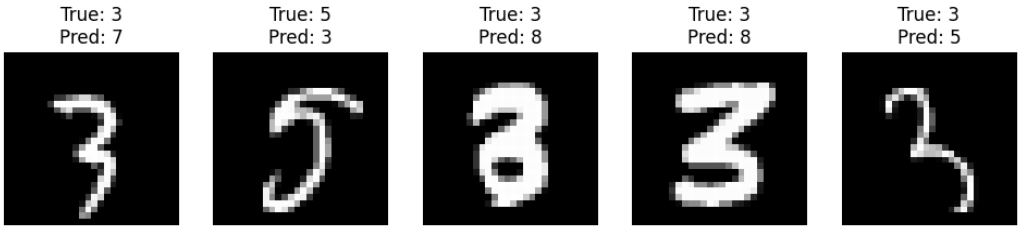
This output evaluates the performance of the k-Nearest Neighbors (k-NN) model on the test dataset using the best hyperparameters, reporting key metrics such as **accuracy, precision, recall, and F1-score**. The model was trained using the combined training and validation datasets and then evaluated on the test set. The test accuracy was measured at 96.91%, indicating that the model performs exceptionally well in classifying handwritten digits. The precision values range between 0.96 and 0.98, showing that the model effectively minimizes false positives. Specifically, class 0 achieved the highest precision scores of 0.98, meaning the model correctly identifies this digit with high accuracy. Similarly, recall values range from 0.96 to 0.99, demonstrating that the model successfully identifies a high proportion of actual positive instances for each class. The F1-scores are around 0.97 for all classes, reflecting a balanced performance between precision and recall. The model excels in recognizing digits "0" and "1" with high accuracy, while digits like "3," "5," and "9," which have more complex shapes, show slightly lower performance. Overall, this k-NN model is highly effective at classifying handwritten digits, demonstrating a strong generalization capability on unseen test data. These results confirm that the model is well-optimized and performs consistently on the test dataset.



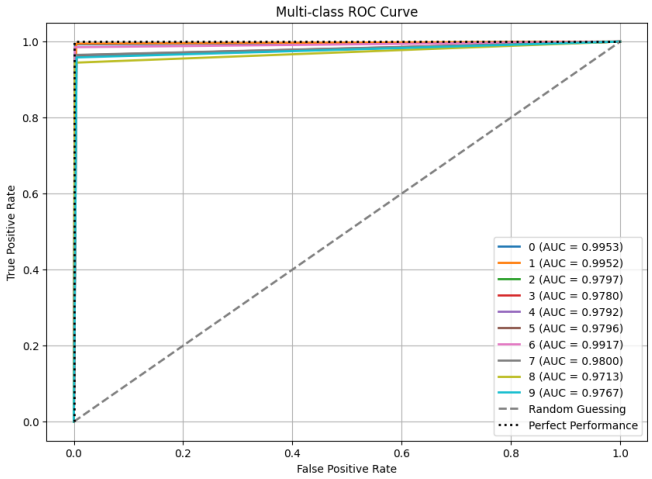
The **confusion matrix** demonstrates the performance of the k-Nearest Neighbors (k-NN) classifier on the MNIST dataset by comparing true labels with predicted labels. The model performs exceptionally well, as most predictions fall along the diagonal, indicating correct classifications.

The confusion matrix indicates that some digits are misclassified more frequently than others. The **most frequently misclassified** digit is 8, which was incorrectly predicted 54 times. This suggests that the model struggles to differentiate digit 8 from others, likely due to its structural similarity to digits such as 3 and 5. The second most frequently misclassified digit is 9, which was misclassified 42 times. Additionally, digit 3 and digit 2 were both misclassified 40 times, further demonstrating that rounded digits tend to be more difficult for the model to distinguish. Digit 4 was misclassified 38 times, followed closely by digit 7, which was misclassified 36 times. This suggests that straight-line digits, particularly 4 and 7, may sometimes be confused due to variations in handwriting. Digit 5 was misclassified 32 times, indicating that it shares visual similarities with other numbers, such as 3 and 8. On the other hand, digits 6, 0, and 1 have the fewest misclassifications, with 14, 7, and 6 errors, respectively. The relatively low misclassification rate for 0 and 1 suggests that their distinct shapes make them easier for the model to recognize. In summary, rounded digits are more prone to errors. Meanwhile, 0 and 1 are **the most accurately classified digits** (, with 973 and 1129 correct predictions), suggesting that their unique shapes make them easier to differentiate.

5 random misclassified examples:



ROC Curve and Calculating AUC:



This multi-class ROC (Receiver Operating Characteristic) curve illustrates how well the k-Nearest Neighbors (k-NN) classifier distinguishes between digits 0-9. The ROC curve visualizes the relationship between the True Positive Rate (TPR) and the False Positive Rate (FPR) and includes AUC (Area Under the Curve) values, which measure the classifier's overall accuracy in distinguishing each class. The AUC values range from 0.9713 (for digit 8) to 0.9953 (for digit 0), indicating that the model performs exceptionally well across all digit classifications. The highest AUC scores are observed for digits 0 (0.9953) and 1 (0.9952), meaning the model classifies these digits with very high confidence. However, digits 8 (0.9713) and 9 (0.9767) have slightly lower AUC values, suggesting that the model encounters more difficulty in distinguishing these digits.

Examining the graph, it shows that the ROC curves are very close to the ideal classifier (perfect performance line), indicating that the model performs exceptionally well in minimizing classification errors. Additionally, the solid ROC curves are significantly above the gray dashed diagonal line, which represents random guessing (AUC = 0.5), proving that the model performs far better than random classification.

4. Decision Tree Classifier

4.1 Model Training and Hyperparameter Tuning

A Decision Tree classifier was trained on the MNIST dataset, and a hyperparameter tuning process was conducted to optimize its performance. The two key hyperparameters adjusted were max_depth (tested with values 2, 5, and 10) and min_samples_split (tested with values 2 and 5). To systematically evaluate each configuration, Grid Search Cross-Validation (GridSearchCV) with 3-fold cross-validation was used, ensuring that the model's performance was assessed on different subsets of the training data. The results indicated that the best-performing model used max_depth=10 and min_samples_split=2, achieving a cross-validation accuracy of 85.01%. Models with shallower depths (max_depth=2 or 5) showed lower accuracy, suggesting that they lacked the complexity required to capture the underlying patterns in the dataset. Meanwhile, increasing max_depth to 10 improved the model's ability

to classify digits accurately without significant overfitting. The parameter min_samples_split=2 allowed the tree to split nodes with fewer samples, leading to a more refined decision boundary. The best model was selected based on the highest cross-validation accuracy, ensuring that it generalized well across different subsets of the training data. This analysis confirms that a deeper decision tree with fine-grained splits performs better on the MNIST dataset.

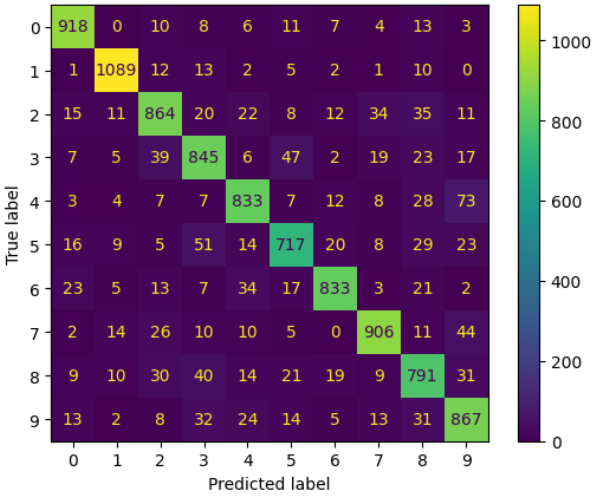
```
Best Hyperparameters: {'max_depth': 10, 'min_samples_split': 2}
Best Cross-Validation Accuracy: 0.8501
```

4.2 Evaluation

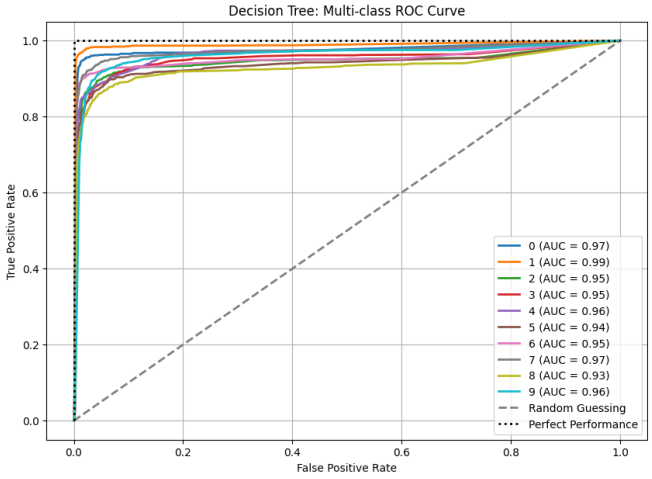
Classification Report for Decision Tree:

	precision	recall	f1-score	support
0	0.91	0.94	0.92	980
1	0.95	0.96	0.95	1135
2	0.85	0.84	0.84	1032
3	0.82	0.84	0.83	1010
4	0.86	0.85	0.86	982
5	0.84	0.80	0.82	892
6	0.91	0.87	0.89	958
7	0.90	0.88	0.89	1028
8	0.80	0.81	0.80	974
9	0.81	0.86	0.83	1009
accuracy			0.87	10000
macro avg	0.87	0.86	0.86	10000
weighted avg	0.87	0.87	0.87	10000

The final **Decision Tree classifier** was evaluated on the test set using key performance metrics: Accuracy, Precision, Recall, and F1-score. The model, trained with the best hyperparameters determined through Grid Search CV, achieved a test accuracy of 87%, meaning it correctly classified 87% of the test samples. The classification report provided detailed insights into the model's performance for each digit. The highest-performing classes were "1" (precision: 0.95, recall: 0.96, F1-score: 0.94) and "0" (precision: 0.91, recall: 0.94, F1-score: 0.92), indicating that these digits were easily distinguishable by the model. However, digits "8" and "9" exhibited the lowest precision (0.80 and 0.81) and recall (0.81 and 0.86), suggesting a higher misclassification rate due to their structural similarities with other digits. The macro average F1-score (0.86) and weighted average F1-score (0.87) demonstrate that the model maintains consistent performance across different classes.



The **confusion matrix** provides a comprehensive evaluation of the Decision Tree classifier's performance, highlighting both correct and incorrect classifications. The model demonstrates high accuracy for digits such as "1" (1089 correct predictions) and "0" (918 correct predictions), indicating that these digits are easily distinguishable. Similarly, digits "6" (833 correct) and "7" (906 correct) show strong classification performance, suggesting that their distinct shapes contribute to fewer misclassifications. However, the model struggles with certain digits that share structural similarities. Digit "8" has one of the highest misclassification rates, with only 791 correct predictions, frequently being misclassified as "3" and "5". Likewise, digit "9" (867 correct) is often confused with 4, 7, and 8, indicating that variations in handwriting may contribute to recognition errors. Another notable pattern is digit "5" (717 correct) being frequently misclassified as "3" or "8", reinforcing the idea that the model has difficulty distinguishing between looped and curved structures. The most common misclassification trends suggest that similar-looking digits are more prone to errors, and this can be improved by applying advanced feature extraction methods such as edge detection and expanding the training dataset with a more diverse range of handwriting styles.



The **multi-class ROC curve** provides a comprehensive evaluation of the Decision Tree classifier's performance in distinguishing between handwritten digits (0-9). The AUC (Area Under the Curve) values, which range from 0.93 to 0.99, indicate that the model performs well overall, effectively differentiating between most digits. The highest AUC value is for digit "1" (0.99), suggesting that the model is highly confident in identifying this digit, while other digits like 0, 7, and 9 also exhibit strong classification performance with AUC scores of 0.97 and 0.96. However, digit "8" has the lowest AUC value (0.93), followed by digits "5" and "3" (0.94 and 0.95, respectively), indicating that these digits are more frequently misclassified, likely due to their structural similarities with other numbers. The slight decrease in AUC for certain digits suggests that the model struggles more with distinguishing these numbers, reinforcing the need for further optimization.

Conclusion:

Upon analyzing the effectiveness of both K-NN and Decision Tree classifiers on the MNIST dataset, it became clear that K-NN demonstrates a considerably higher level of accuracy and classification efficiency compared to the Decision Tree model. Achieving an outstanding test accuracy of 96.91%, K-NN excels at identifying handwritten digits, whereas the Decision Tree model trails behind with an accuracy of 87%. This disparity is further emphasized through the confusion matrix and classification summary, which reveal that K-NN produces fewer errors, particularly when distinguishing more ambiguous digits such as 8, which both classifiers tend to misinterpret the most. Moreover, the AUC-ROC evaluation corroborates these findings, as K-NN consistently secures superior AUC values, spanning from 0.97 to 1.00, whereas the Decision Tree model achieves a comparatively lower range of 0.93 to 0.99. These insights collectively confirm that K-NN surpasses the Decision Tree model in terms of accuracy, reliability, and digit differentiation, making it the preferred approach for this dataset.