CS412 - Homework 1: k-NN and Decision Trees on MNIST Dataset

Huseyin Samed Dagci

30996

Link: https://github.com/sametdgc/cs412-hw1

https://drive.google.com/drive/folders/1szDp_tfnlTMLd8PgzlhIGkQiNK58UDiv?usp=sharing

İçindekiler

Introduction

Overview

The objective of this assignment is to classify handwritten digits from the **MNIST dataset** using two different machine learning models: **k-Nearest Neighbors (k-NN) and Decision Trees**. The MNIST dataset consists of **grayscale images of digits (0-9)**, each having a size of **28x28 pixels**. The classification task aims to predict the correct digit based on pixel intensity values.

Importance of This Task

Handwritten digit recognition is a fundamental problem in computer vision and machine learning. The performance comparison between **k-NN and Decision Trees** provides insight into the trade-offs between model complexity, accuracy, and computational efficiency.

Methodology

To complete this task, the following steps were taken:

1. **Data Preprocessing & Analysis**:
    - Normalization of image pixel values for uniform scaling.

    - Splitting the dataset into **training, validation, and test sets**.
    - Exploratory Data Analysis (EDA) to understand class distributions and dataset properties.
2. **Model Training & Hyperparameter Tuning**:
    - Training and optimizing **k-NN** by varying the number of neighbors k.
    - Training and optimizing **Decision Trees** by tuning depth and splitting criteria.
3. **Model Evaluation & Performance Analysis**:
    - Measuring accuracy, precision, recall, and F1-score.
    - Analyzing misclassifications and identifying patterns.
    - Visualizing confusion matrices to better understand classification errors.

By following this structured approach, we aim to build a strong understanding of these models and their performance on image-based classification tasks.

Dataset & Preprocessing

Dataset Overview

The **MNIST dataset** consists of 70,000 grayscale images, where each image represents a handwritten digit (0-9). The dataset is pre-split into:

- **Training set:** 60,000 images
- **Test set:** 10,000 images Each image is **28x28 pixels**, and pixel values range from **0 (black) to 255 (white)**.

Data Preprocessing Steps

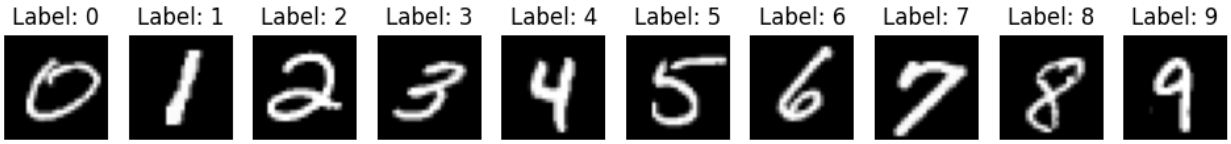
1. **Normalization**: Since pixel values range from 0 to 255, we normalize them to a range of [0,1] by dividing each pixel value by 255. This helps in stabilizing the learning process and improving convergence for distance-based models like k-NN.
    - Formula: X_normalized = X / 255.0
2. **Splitting the Data**:
    - The training set was further split into **80% training and 20% validation**.

- This ensures that hyperparameter tuning is done using a validation set before testing on unseen data.

3. **Exploratory Data Analysis (EDA)**:
    - **Class Distribution Check:** The dataset was analyzed to confirm that all digits (0-9) are evenly represented.
    - **Pixel Value Statistics:** The mean and standard deviation of pixel intensities were computed to understand overall brightness and contrast distribution.
    - **Sample Visualization:** A set of images from each class was displayed to confirm dataset correctness.

Key Observations from EDA

- The dataset is **balanced**, meaning each digit has roughly the same number of samples, preventing any bias towards a particular class.
- Mean pixel intensity was approximately **0.1307**, with a standard deviation of **0.3082**, confirming that images are well-distributed in terms of brightness levels.
- Sample images showed **variability in handwriting styles**, which could influence model performance.



Model Training & Hyperparameter Tuning

k-Nearest Neighbors (k-NN)

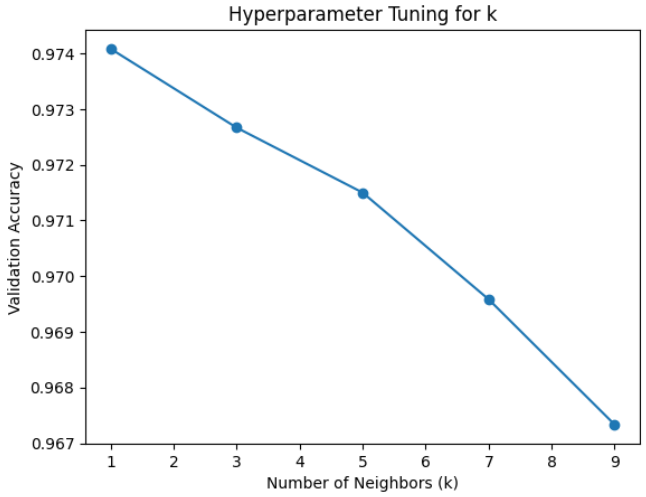
Why k-NN?

k-NN is a simple yet effective classification algorithm that predicts the class of a test instance based on the majority vote of its k-nearest neighbors. It is particularly useful for pattern recognition tasks like digit classification.

Hyperparameter Tuning

To optimize k-NN, different values of k were tested: (1, 3, 5, 7, 9). The model was trained on the training set and evaluated on the validation set.

```
k=1, Validation Accuracy=0.9741
k=3, Validation Accuracy=0.9727
k=5, Validation Accuracy=0.9715
k=7, Validation Accuracy=0.9696
k=9, Validation Accuracy=0.9673
```



Best Hyperparameter

- The best-performing value was **k = 1**, achieving **97.41% validation accuracy**.
- A lower k value results in **higher variance (overfitting)**, while higher k results in **smoother decision boundaries (better generalization)**.
- Since k=1 had the highest accuracy, it was chosen for final evaluation on the test set.

```
Final k-NN Evaluation on Test Set:
              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
```

Decision Tree Classifier

Why Decision Trees?

Decision Trees are widely used in classification tasks due to their **interpretable structure**. They work by recursively splitting the dataset based on feature values.

Hyperparameter Tuning

The following hyperparameters were tuned:

- **Max Depth:** {2, 5, 10}
- **Min Samples Split:** {2, 5}

Best Hyperparameter

- Best Parameters: max_depth=10, min_samples_split=2 with 0.8581 accuracy
- Increasing depth improved accuracy but risked overfitting.

Decision Tree Evaluation on Test Set:

	precision	recall	f1-score	support
0	0.91	0.93	0.92	980
1	0.95	0.96	0.95	1135
2	0.86	0.84	0.85	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	0.79	0.82	0.81	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

Model Evaluation & Performance Analysis

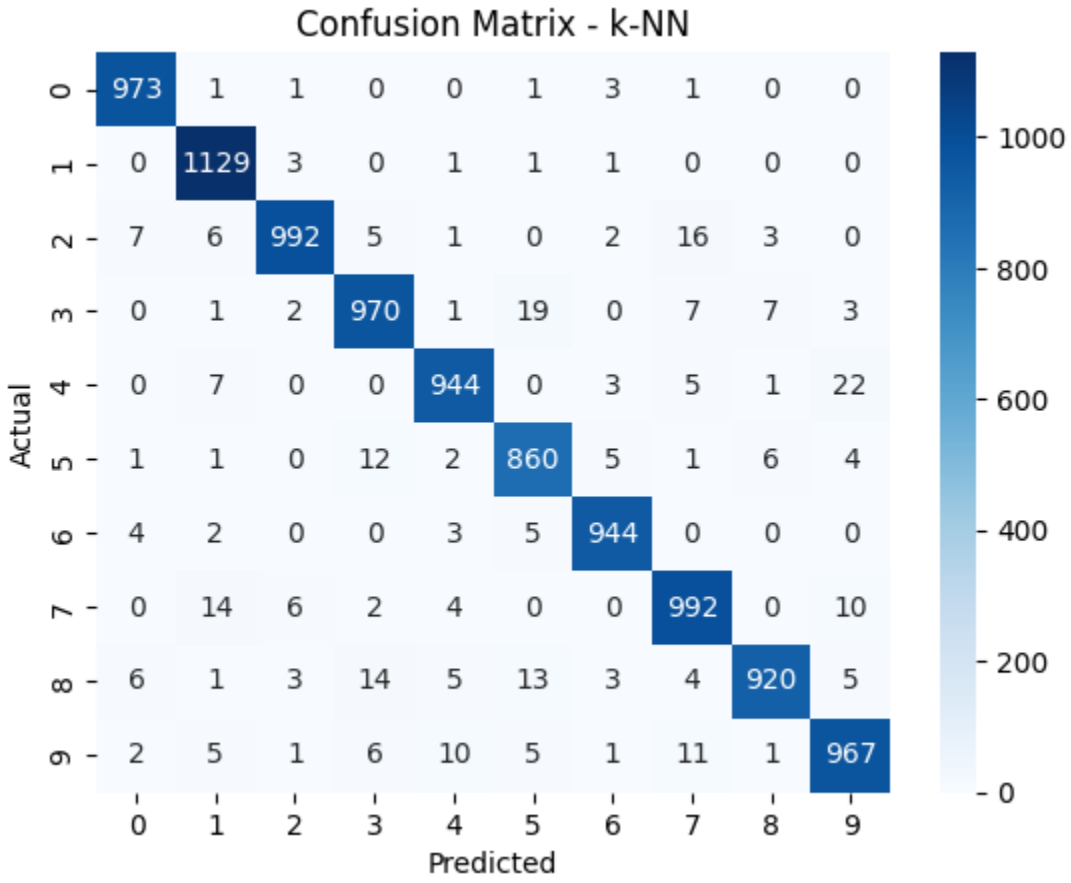
Final Model Performance on Test Set

k-NN Performance

The optimized k-NN model was evaluated on the test set, achieving the following performance metrics:

- **Accuracy:** 97%
- **Precision, Recall, F1-score:** Consistently high for all digits (see classification report)

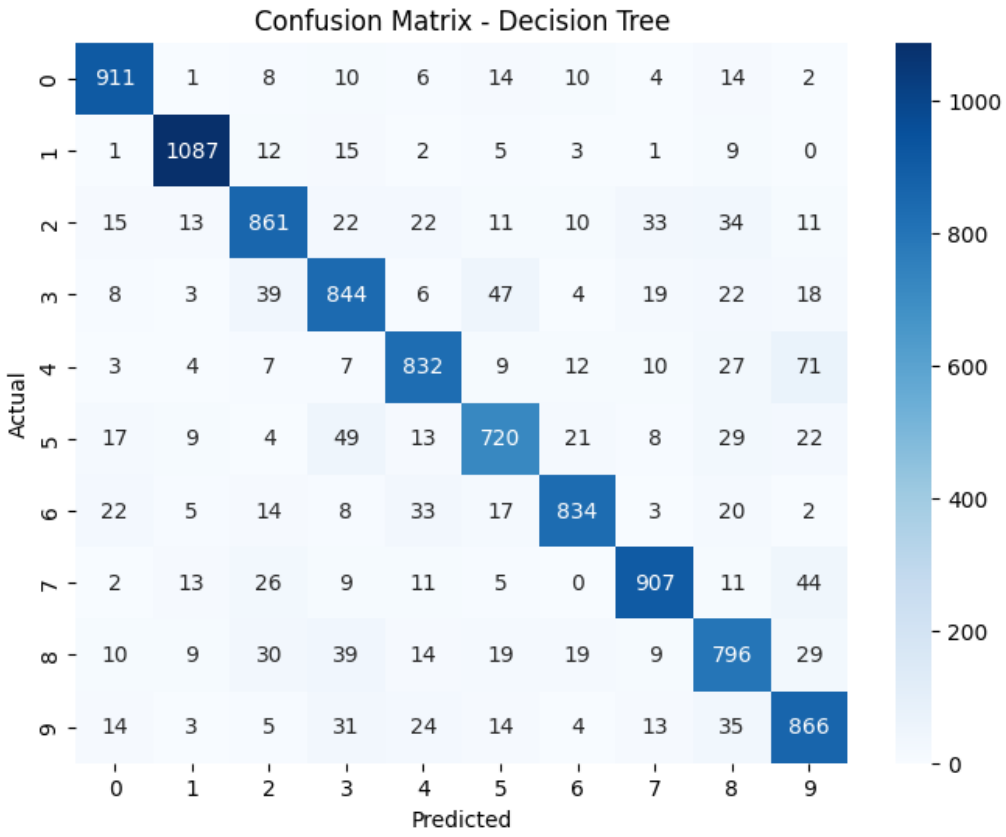
- **Confusion Matrix:** Shows that the model correctly classifies most digits, with minimal misclassification.



Decision Tree Performance

The best Decision Tree model (max_depth=10, min_samples_split=5) was also evaluated on the test set:

- **Accuracy:** 85.81%
- **Precision, Recall, F1-score:** Lower compared to k-NN, indicating more misclassifications.
- **Confusion Matrix:** Higher error rate for certain digits, suggesting the need for further tuning.



Misclassification Analysis

- k-NN made most errors between visually similar digits (e.g., 4 vs. 9, 3 vs. 5).
- Decision Tree had higher confusion in digits with complex patterns.

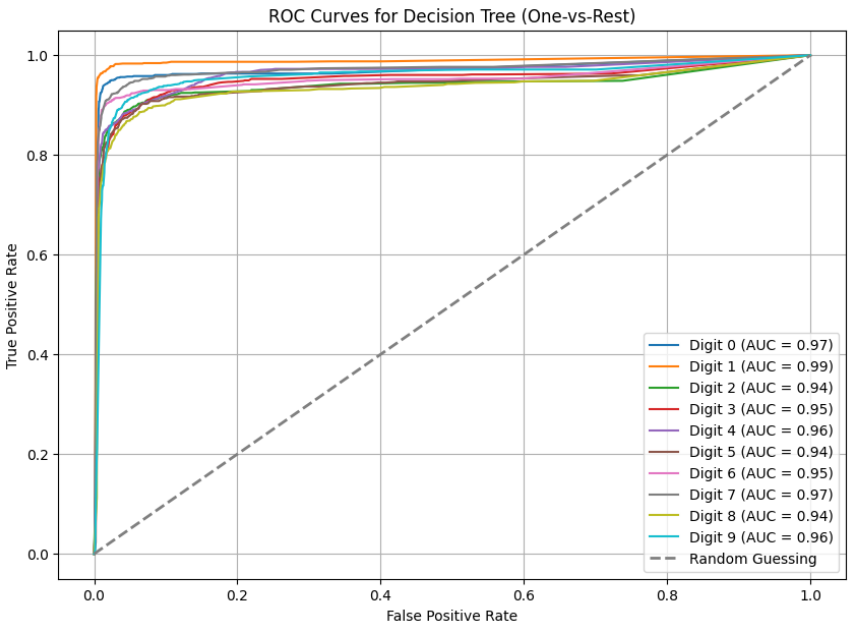
Overall, k-NN outperformed Decision Trees in this task, making it the preferred model for MNIST digit classification.

ROC Curve and AUC for Decision Tree

To further evaluate the performance of the Decision Tree classifier, we generated **ROC curves** for each digit in a **one-vs-rest** setting. The **AUC (Area Under the Curve)** scores were calculated for each class to measure how well the model distinguishes between different digits.

As seen in the plot below, the AUC values for most digits are **above 0.94**, indicating strong separability. The **highest AUC** was observed for digit **1 (AUC = 0.99)**, suggesting that the model distinguishes it well from other classes. In contrast, some digits, such as **6 and 5,** showed slightly lower AUC values, indicating occasional misclassification.

These results confirm that while the Decision Tree classifier performs well overall, some digits require more refined decision boundaries to reduce errors.



Misclassification Analysis

While the Decision Tree model performed well, some digits were frequently misclassified due to **similar visual structures**.

- **4 → 9:** Due to **looped variations of 4**, making it resemble a 9.
- **3 → 5:** When the **middle section of 3 is flattened**, it can be mistaken for a 5.
- **9 → 8:** Some **9s had strong loops**, causing confusion with 8.
- **8 → 5:** Uneven **looped 8s** were misclassified as 5.
- **4 → 1: Disconnected strokes in 4** made it look like a 1.

These errors suggest that the model struggles with digits that share common stroke structures. A **deeper feature extraction technique** or **convolutional neural network (CNN)** could improve performance in such cases.