Handwritten Digit Recognition with KNN: A Machine Learning approach using MNIST Dataset

[Github Link]

1. Introduction

1.1 Motivation & Overview

Handwritten digit and text recognition is a fundamental problem in machine learning and computer vision, with applications in postal mail sorting, bank check processing, and many more. In this report, we explore the use of the **k-Nearest Neighbours (KNN)** algorithm to classify handwritten digits using the Digits Dataset from SKLearn, which is similar to MNIST dataset but smaller in size. The K-Nearest Neighbors (KNN) algorithm is a simple and effective supervised learning method used for both classification and regression tasks.

1.2 Why KNN?

KNN is a simple yet powerful algorithm for image classification. It classifies a new data point based on the majority class among its k-nearest neighbours. By tuning k, the number of neighbours, [1, 2] we can balance bias and variance for optimal performance.
KNN is a non-parametric, lazy learning algorithm that does not make assumptions about the data points. Instead, it stores the entire dataset and makes predictions by comparing the similarity of a test instance to the training samples. This makes KNN highly interpretable and simple to implement, but computationally expensive for large datasets.

2. Dataset and Problem Setup

2.1 Digits Dataset (Scikit-learn)

The Digits dataset from Scikit-learn is a well-known benchmark dataset for handwritten digit recognition. It consists of images of digits (0-9) that have been resized to 8x8 pixels, making it computationally efficient while still retaining enough detail for classification tasks. [1]
Dataset Details:

- Total Samples: 1,797
- Image Size: 8 × 8 pixels
- Classes: 10 (digits 0 to 9)
- Data Type: Grayscale images, pixel values ranging from 0 to 16

Unlike the MNIST dataset, which has 28x28 images, this dataset is much smaller, making it ideal for quick experimentation and prototyping. [2]

2.2 Preprocessing Steps

Since pixel values range from 0 to 16, we scale them to 0-1 for better performance in distance-based models like KNN. Each 8x8 image is converted into a 1D vector of size 64 (since KNN operates on vectorized data).

2.3 Splitting the Data

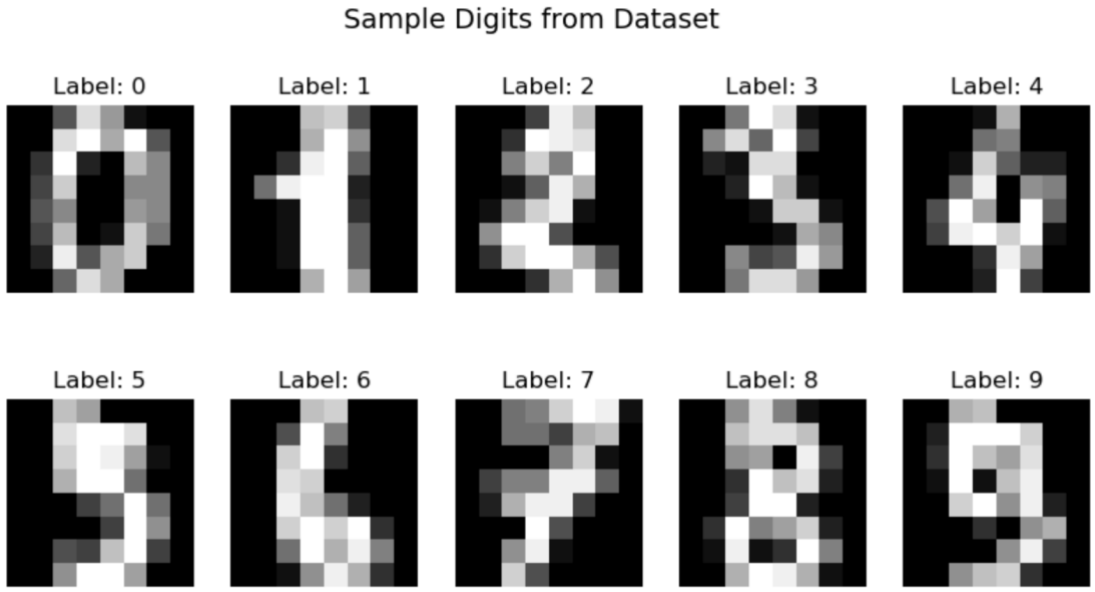
We split the dataset into 80% training and 20% testing to evaluate model performance.

3. Exploratory Data Analysis (EDA)

This dataset needs to be carefully investigated, to [3] [4] [5]understand the images that are present in the dataset.

3.1 Data Visualization

To understand the dataset better, we visualize some sample images. The figure below shows a selection of handwritten digits, each labelled with its corresponding numeric value.



4. Methodology

The following steps were followed to implement KNN classifier:

4.1 Data Preprocessing

Before training, the dataset was split into training data and test data to ensure model performance. This ensures that we can access how well the model generalizes to new & unseen data. Feature scaling was also applied to normalize the pixel intensity values, ensuring that distance calculations in the KNN algorithm are not biased by larger numerical values.

4.2 Model Training

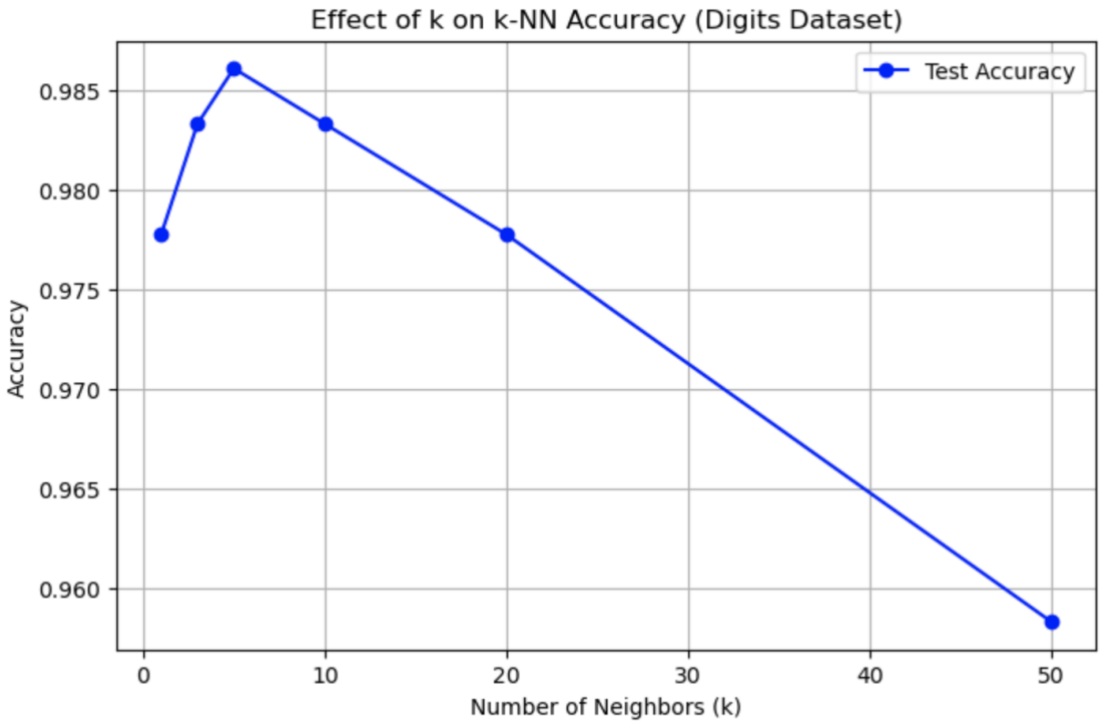
A KNN classifier was trained using different k neighbours. The choice of k significantly affects the model's performance, as a smaller k can lead to overfitting, while a larger k can smooth out classification boundaries. The Euclidean distance metric was used as it provides a straightforward way to measure similarity in high-dimensional spaces.

4.3 Hyperparameter Tuning

A grid search was conducted to find the optimal k value, evaluating different choices such as k=1, 3, 5, 10, 20 and 50. The performance metrics were recorded to determine the best k for the dataset.

4.4 Model Evaluation and Results

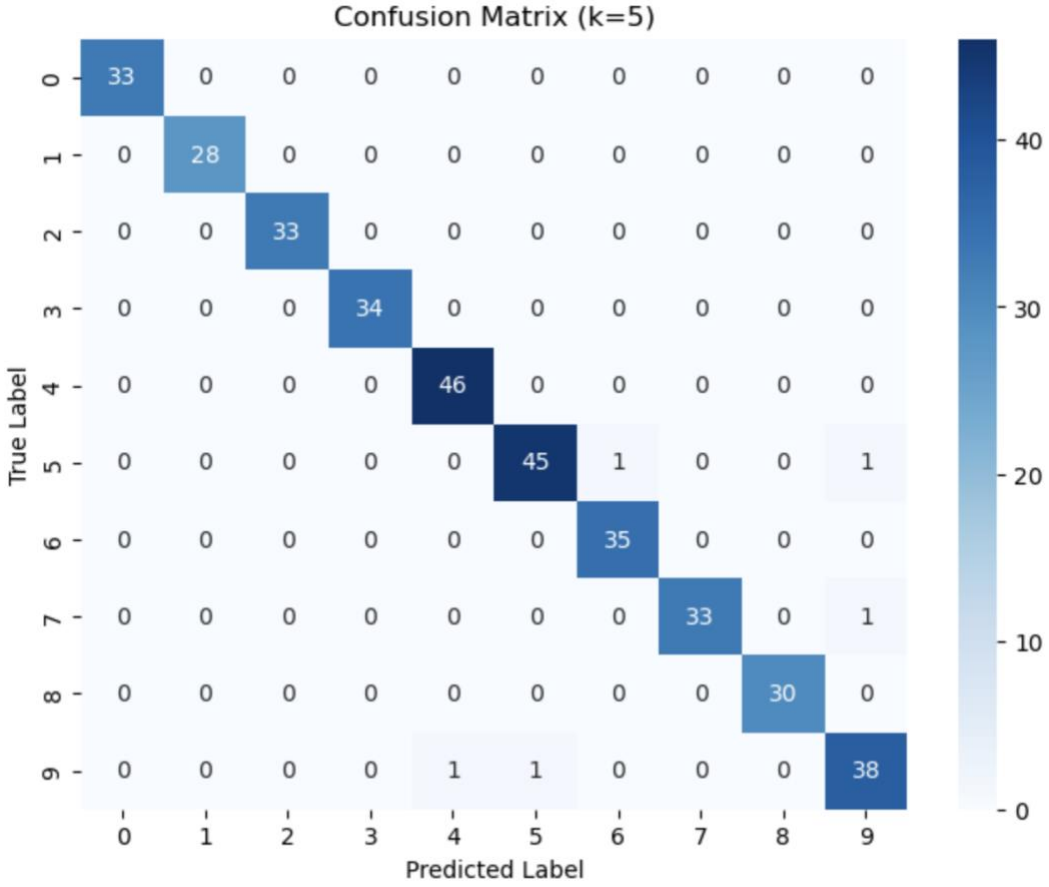
The model's performance was evaluated using accuracy, a confusion matrix, and a classification report. Accuracy provides an overall measure of correctness, while the confusion matrix highlights misclassifications across digit classes.



5. Results and Discussion

5.1 Confusion Matrix

The confusion matrix highlights how well the model distinguishes different digits. Misclassifications were observed primarily between digits with similar structures.



From the confusion matrix, we observe that most predictions are correct, but some misclassifications occur. The misclassification is particularly high between digits that have visually similar structures, such as 3 and 8 or 5 and 9.

5.2 Classification Report

The classification report provides insights into the precision, recall, and F1-score of each class. High precision indicates a low false positive rate, while high recall ensures that most instances of a class are correctly identified.

```
Classification Report for k=5:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       1.00      1.00      1.00        28
           2       1.00      1.00      1.00        33
           3       1.00      1.00      1.00        34
           4       0.98      1.00      0.99        46
           5       0.98      0.96      0.97        47
           6       0.97      1.00      0.99        35
           7       1.00      0.97      0.99        34
           8       1.00      1.00      1.00        30
           9       0.95      0.95      0.95        40

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```

The classification report confirms strong model performance, with high precision and recall for most digits. However, misclassifications occur in digits with similar stroke patterns, suggesting that feature extraction techniques like PCA (Principal Component Analysis) or image augmentation could further improve results.

5.3 Performance Analysis

The model achieved high accuracy on the test dataset. Although, some misclassifications occurred, particularly between similar-looking digits. Increasing **k** or using distance weighting can further improve model's accuracy. Feature scaling played an important role in improving the model's performance.

5.4 Accessibility

- **Color-blind-Friendly Plots**: We have used cool-warm color maps to ensure the visual distinction is accessible.

Descriptive Captions and Titles: Each plot has clear labelling to assist screen readers and color-impaired users.

6. Conclusion

The KNN classifier effectively classifies handwritten digits with high accuracy. However, it is computationally expensive for large datasets. Optimizing **k** and preprocessing data can enhance performance.

6.1 Key Findings

KNN achieves reasonable accuracy but struggles with overlapping digits. Increasing k improves stability but reduces sensitivity to fine details.

6.2 Future Improvements

- Using a weighted KNN approach where closer neighbours have more influence on classification.

- Applying dimensionality reduction techniques such as PCA to improve computation efficiency.
- Exploring other distance metrics like Manhattan or Minkowski distance, instead of Euclidean.

References

[1] F. V. G. G. A. M. V. T. B. G. O. B. M. P. P. W. R. D. V. V. J. P. A. C. D. B. M. P. M. &. D. Pedregosa, “Scikit-learn: Machine Learning in Python,” 2011. [Online]. Available: https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf.

[2] M. Lichman, “UCI Machine Learning Repository,” 2013. [Online]. Available: https://archive.ics.uci.edu/.

[3] T. T. R. &. F. J. Hastie, “The Elements of Statistical Learning.,” 2009. [Online]. Available: https://web.stanford.edu/~hastie/ElemStatLearn/.

[4] T. &. H. P. Cover, “Nearest neighbor pattern classification.,” 1967. [Online]. Available: https://ieeexplore.ieee.org/document/1053964.

[5] Altman, “Kernel and Nearest-Neighbor Nonparametric Regression,” 1992. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879.