About MNIST Dataset

The MNISTdataset is a benchmark dataset widely used for evaluating image classification algorithms. It contains a total of 70,000 grayscale images of handwritten digits ranging from 0 to 9, with 60,000 images allocated for training and 10,000 for testing.

Each image in the dataset is a 28×28 pixel square, resulting in 784 features per image when flattened into a one-dimensional array. The pixel values range from 0 (black) to 255 (white), representing varying intensities of gray.

Objectives

The primary objectives of this project are as follows:

1. **To build a complete machine learning pipeline from scratch** for classifying handwritten digits using the KNN algorithm, without relying on prebuilt classifiers or libraries

2. **To load and preprocess the MNIST dataset**, which includes:
    - Visualizing sample digits to understand the data.
    - Flattening 28x28 grayscale images into 784-dimensional feature vectors.
    - Normalizing pixel values to a [0, 1] range to standardize input features.

3. **To optionally reduce dimensionality using PCA**

    for performance optimization and better visualization of high-dimensional data.

4. **To implement the KNN:**
    - Efficient computation of Euclidean distances.
    - Vectorized prediction of labels based on nearest neighbors.
    - Scoring model performance using accuracy as a metric.

5. **To split the dataset** into training (60%), validation (20%), and testing (20%) subsets for model development and unbiased evaluation.

6. **To evaluate the performance of the KNN model** by:

    - Trying different values of *k* and analyzing their effect on validation accuracy.

    - Generating and interpreting a confusion matrix and classification report.

7. **To visualize final test predictions** alongside their actual labels to qualitatively assess the classifier's performance.

8. **To gain a deeper understanding of distance-based classifiers**, data preprocessing techniques, and evaluation methods in a practical computer vision context.

Data Source

The dataset used in this project is the **MNIST handwritten digits dataset**, obtained using:

```
from sklearn.datasets import fetch_openml
```

Preprocessing

we Applied following preprocessing steps:

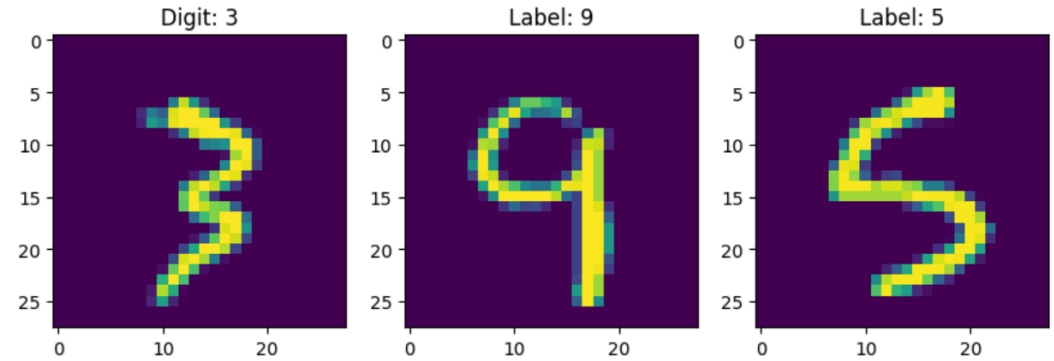
- **Flattening**: Each image of size **28x28 pixels** was reshaped into a **1D array of 784 features**, converting the 2D image into a vector suitable.

- **Normalization**: All pixel values, originally in the range [0, 255], were scaled to the range **[0, 1]** by dividing each value by 255.0. This step ensures that all features contribute equally when computing distances.

These preprocessing steps help improve the performance and accuracy of the KNN classifier by standardizing the input format and reducing numerical bias.

Digit Visualization

Visualising MNIST images using Matplotlib by reshaping the flattened arrays back into **28×28**. This helped verify the data and observe the variations in handwriting styles before training the model.




```
<Figure size 640x480 with 0 Axes>
```

Dimensionality Reduction

To optimize performance and reduce computational cost, **PCA** was applied to the dataset:

- Manual implementation of PCA using NumPy.
- **Objective**: Reduce the dimensionality of the feature vectors while preserving most of the original information.
- **Variance Retention**: The number of principal components was chosen **95% of the total variance** in the data was retained.
- **Result**: The original 784-dimensional feature space was projected onto a lower-dimensional subspace, significantly improving the efficiency of distance computations in the KNN algorithm.

Dimensionality reduction also helped iin visualizing the dataset and eliminating redundant or noisy features.

Data Split

The MNIST dataset was divided into three subsets to support effective training, tuning, and evaluation of the KNN model. Sixty percent of the data was allocated for training, allowing the model to learn patterns from a diverse set of examples. Twenty percent was set aside as a validation set, which was used to tune the hyperparameter *k* and assess generalization during development. The remaining twenty percent was reserved for testing, ensuring an unbiased evaluation of the final model's performance on unseen data.

However due to such larger dataset the time taken was long so we had to convert dataset to subset to compute results properly

---

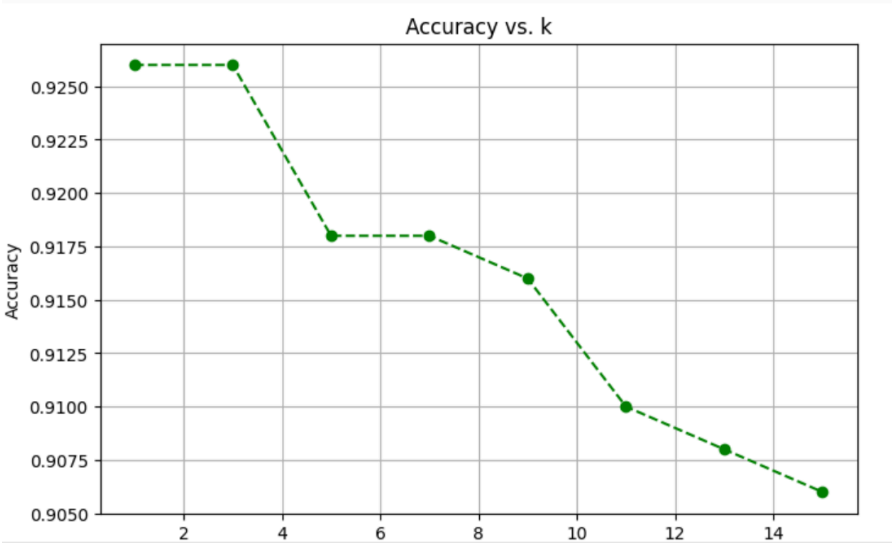
Model

The core of this project is a custom implementation of the K-Nearest Neighbors classifier using only NumPy. The model computes the Euclidean distance between a test sample and all training samples, identifies the *k* nearest neighbors, and assigns the most common label among them as the prediction. To maintain efficiency, the entire implementation was vectorized..

Performance :

To find the optimal number of neighbors for the KNN classifier, a range of odd values from 1 to 15 was tested. Using odd values helps avoid ties during the majority vote among neighbors. The validation set was used to evaluate model accuracy for each value of *k*, and the results were plotted to visualize how performance varied with different *k* values. This experimentation enabled the selection of a *k* that balances model complexity with generalization capability.

Visualization



Interpretation:

To find the best value for **k**, we plotted a validation accuracy graph across several odd values from 1 to 15. The plot shows that accuracy was highest at **k = 1 and 3**, reaching around **92.6%**, and gradually decreased as k increased. This trend reflects how smaller k values capture more local detail, while larger values lead to underfitting. Based on the plot, **k = 3** was chosen as the optimal value for the final model.

Evaluation

Model performance was measured using validation and test accuracy. A **confusion matrix** and **classification report** were generated to analyze prediction accuracy, precision, recall, and F1-score for each digit. These metrics provided a clear view of the classifier's effectiveness across all classes.

Results Summary

The KNN classifier achieved an overall **accuracy of 91%** on the test set. The classification report shows strong performance across most digit classes, with **precision and recall scores typically above 0.90**. Notably, the model performed best on digits like **0, 1, and 6**, while digits such as **2 and 5** had slightly lower recall, indicating some misclassifications. The confusion matrix highlights these cases, with minor confusion observed between similar-looking digits.

Overall, the results demonstrate that the KNN model, despite its simplicity, can effectively classify handwritten digits when properly tuned and preprocessed.