**Handwritten Digit Classification using K-Nearest Neighbors (KNN)**

**Introduction**

Handwritten digit recognition is one of the classic problems in machine learning and computer vision. In this project, we use the Digits dataset from scikit-learn, which consists of 1,797 grayscale images of handwritten digits (0–9), each represented as an 8×8 image.

The objective is to build a classification model that can accurately predict the digit shown in each image. For this purpose, the K-Nearest Neighbors (KNN) algorithm is applied and evaluated using different distance metrics and values of *k*.

**Key Insights**

1. **Data Visualization**
   * The dataset consists of 8×8 grayscale digit images.
   * Initial visualization shows digits are distinguishable with noticeable patterns.
2. **Model Training**
   * Data standardized using StandardScaler.
   * Split into 80% training and 20% testing (stratified).
3. **Results**
   * KNN (k=3, Euclidean distance) → Accuracy = 96.6%
   * KNN (k=3, Manhattan distance) → Accuracy = 98.3%
   * Cross-validation across k=1 to 21 confirmed optimal accuracy around k=3–5.
4. **Confusion Matrix**
   * Visual heatmaps showed minimal misclassifications.
   * Strong performance across all digit classes.

**Conclusion**

This study demonstrates that KNN is highly effective for handwritten digit classification, achieving nearly 98% accuracy with Manhattan distance.

Although KNN is simple, it performs remarkably well on this moderate-sized dataset. However, it can become computationally expensive for larger datasets due to distance calculations

**Complete Python Code**

# Import libraries

from sklearn.datasets import load\_digits

import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

import seaborn as sns

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix, classification\_report, accuracy\_score

from sklearn.model\_selection import cross\_val\_score

# Load dataset

digits = load\_digits()

x = digits.data

y = digits.target

# Visualize first 5 digits

plt.figure(figsize=(10, 15))

for i in range(5):

plt.subplot(5, 1, i + 1)

plt.imshow(x[i].reshape(8,8))

plt.title(f"value:{y[i]}")

plt.show()

# Train-test split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

x, y, test\_size=0.2, stratify=y, random\_state=42

)

# Standardize the data

scaler = StandardScaler()

x\_train = scaler.fit\_transform(x\_train)

x\_test = scaler.transform(x\_test)

# KNN with Euclidean distance

knn = KNeighborsClassifier(n\_neighbors=3, metric='euclidean')

knn.fit(x\_train, y\_train)

y\_pred = knn.predict(x\_test)

print("Accuracy (Euclidean):", accuracy\_score(y\_test, y\_pred))

# Confusion matrix (Euclidean)

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True)

plt.ylabel('Truth')

plt.xlabel('Predicted')

plt.show()

# KNN with Manhattan distance

KNN\_M = KNeighborsClassifier(n\_neighbors=3, metric='manhattan')

KNN\_M.fit(x\_train, y\_train)

y\_pred = KNN\_M.predict(x\_test)

print("Accuracy (Manhattan):", accuracy\_score(y\_test, y\_pred))

# Confusion matrix (Manhattan)

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True)

plt.ylabel('Truth')

plt.xlabel('Predicted')

plt.show()

# Cross-validation for k values

k\_values = range(1, 21)

cv\_scores = []

for k in k\_values:

knn = KNeighborsClassifier(n\_neighbors=k)

scores = cross\_val\_score(knn, x\_train, y\_train, cv=5, scoring='accuracy')

cv\_scores.append(scores.mean())

# Plot accuracy vs. k

plt.plot(k\_values, cv\_scores, marker='o', linestyle='-')

plt.xlabel("Number of Neighbors (k)")

plt.ylabel("Cross-Validation Accuracy")

plt.title("KNN Accuracy vs k")

plt.show()