Experiment 7

Apply K-nearest neighbors algorithm on handwritten digit classification.

Aim: Apply K-nearest neighbors algorithm on handwritten digit classification.

Theory:

K-Nearest Neighbors (KNN) Algorithm:

KNN is a supervised machine learning algorithm used mainly for classification and also for regression. It works by finding the **k closest data points** (neighbors) to a new input based on a distance metric like Euclidean distance.

- **Classification:** The new point is assigned the class that is most common among its k nearest neighbors (majority voting).
- **Regression:** The predicted value is the average of the neighbors' values.

KNN is a **lazy learner** because it doesn't build a model in advance—it stores the training data and performs computation only when making predictions.

Working of KNN Algorithm:

Step 1: Selecting the optimal value of K

Step 2: Calculating distance

Step 3: Finding Nearest Neighbors

Step 4: Voting for Classification or Taking Average for Regression

The choice of k is important:

- A small k may lead to noisy predictions (overfitting).
- A large k may smooth out patterns too much (underfitting).
- Techniques like cross-validation and the elbow method help find the optimal k.

Common distance metrics used:

- **Euclidean distance:** Straight-line distance between points.
- **Manhattan distance:** Distance measured along grid-like paths.
- **Minkowski distance:** Generalized distance including Euclidean and Manhattan.

Advantages:

- Simple and easy to implement
- No training phase required
- Can handle both classification and regression tasks

Disadvantages:

- Slow for large datasets (calculates distance to all points)
- Performance drops with many features (high dimensionality)
- Can overfit if data is noisy or not cleaned

Applications:

- Recommendation systems
- Spam detection
- Customer segmentation
- Speech recognition

CONCLUSION:

K-Nearest Neighbors (KNN) is a straightforward yet powerful algorithm for both classification and regression tasks. Its simplicity, combined with its ability to make predictions based on local patterns in data, makes it a popular choice for many practical applications. However, its efficiency and accuracy depend heavily on the choice of the parameter k and the quality of the data. While KNN can struggle with large or high-dimensional datasets, its intuitive approach and ease of implementation continue to make it a valuable tool in the machine learning toolbox.

VIVA QUESTIONS:

1. What does the parameter k represent in the K-Nearest Neighbors algorithm?
2. How does KNN classify a new data point?

3. Why is KNN called a lazy learner?
4. Name three common distance metrics used in KNN.
5. What are the main disadvantages of using KNN on large datasets?

PROGRAM:

Open Command Prompt and run:

C:/Users/Admin/AppData/Local/Programs/Python/Python313/python.exe -m pip install numpy

Main Code:

```
# K-Nearest Neighbors (KNN) Classifier

# Make sure you install numpy using:

# C:/Users/Admin/AppData/Local/Programs/Python/Python313/python.exe -m pip install numpy

import numpy as np

from collections import Counter

# Step 1: Function to calculate Euclidean distance

def euclidean_distance(point1, point2):

    return np.sqrt(np.sum((np.array(point1) - np.array(point2)) ** 2))

# Step 2: KNN Prediction function

def knn_predict(training_data, training_labels, test_point, k):

    distances = []
```

```
        for i in range(len(training_data)):
            dist = euclidean_distance(test_point, training_data[i])
            distances.append((dist, training_labels[i]))

    # Sort by distance
    distances.sort(key=lambda x: x[0])

    # Get the k nearest labels
    k_nearest_labels = [label for _, label in distances[:k]]

    # Return the most common label among the nearest neighbors
    return Counter(k_nearest_labels).most_common(1)[0][0]

# Step 3: Example dataset
training_data = [[1, 2], [2, 3], [3, 4], [6, 7], [7, 8]]
training_labels = ['A', 'A', 'A', 'B', 'B']
test_point = [4, 5]
k = 3

# Step 4: Make prediction
```

```
prediction = knn_predict(training_data, training_labels, test_point, k)
print("Predicted label for test point", test_point, "is:", prediction)
```