Rajshahi University of Engineering & Technology

Heaven's Light is Our Guide

Course Code: CSE 4204

Course Title: Sessional based on CSE 4203

Experiment no.: 01

Submitted by:

Name: Ayan Sarkar

Roll: 1903162

Section: C

Department: CSE

Submitted to:

Md. Mazharul Islam

Lecturer,

Department of CSE,

RUET

Problem: Implementation of Nearest Neighbor classification algorithms with and without distorted pattern.

Experiment Description: This experiment focuses on implementing a k-Nearest Neighbors (k-NN) classifier and evaluating its performance with different distance metrics. The k-NN algorithm classifies data points based on the majority class among their k-nearest neighbors in the training data.

The experiment involves testing the k-NN classifier on a dataset and exploring the impact of various distance metrics. These metrics, used to calculate the distance between data points, include:

- Euclidean Distance: The straight-line distance between two points.
- Manhattan Distance: The sum of the absolute differences of their coordinates.
- Chebyshev Distance: The maximum absolute difference between their coordinates in any dimension.

Different values of k (the number of nearest neighbors considered) are also tested to observe their effect on classification accuracy. The results are visualized using scatter plots to display the dataset and line and bar plots to illustrate the k-NN classifier's performance with different distance metrics and k values.

Dataset Characteristic: The dataset was generated using the make_blobs function from the sklearn.datasets library. It consisted of:

- **Features:** Two-dimensional numerical data points.
- **Classes:** Two distinct clusters/classes labeled 0 and 1.
- **Training Set:** 80% of the data (160 samples).
- **Test Set:** 20% of the data (40 samples).

Code:

K-Nearest Neighbors (KNN) classification model has been implemented from scratch to classify.

Data Generation and Visualization:

Visualized the dataset using matplotlib to display the two clusters corresponding to the classes.

Data Splitting:

Implemented two custom distance metrics:

- **Euclidean Distance:** Measures the straight-line distance between two points.
- **Hamming Distance:** Calculates the sum of absolute differences between two points.

Prediction and Accuracy: Defines a predict function that determines the class label of a test sample based on the majority vote of its k-nearest neighbors. Implements an accuracy function to evaluate the model's prediction accuracy.

Model Function:

Iterates through test samples to compute distances from all training samples using a specified distance metric. Sorts distances, selects the k nearest neighbors, and predicts the class label based on majority voting.

Execution:

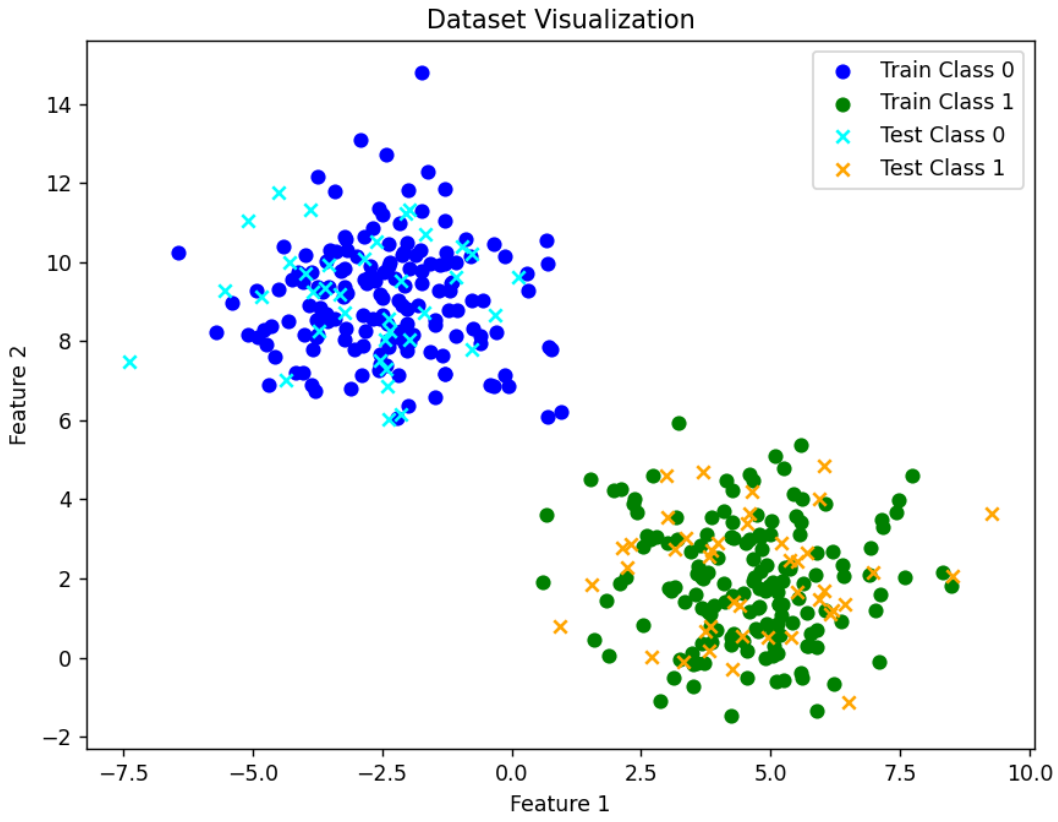
Runs the model using Hamming Distance and 5 neighbors (k=5) and achieves 100% accuracy due to the well-clustered nature of the dataset.

Implementation:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# Generate dataset using make_blobs
data, labels = make_blobs(n_samples=400, centers=2, cluster_std=1.5, random_state=42)

# Split dataset into training (80%) and testing (20%)
def train_test_split(data, labels, test_size=0.2):
    n_test = int(len(data) * test_size)
    indices = np.arange(len(data))
    np.random.shuffle(indices)
    test_indices = indices[:n_test]
    train_indices = indices[n_test:]
    return data[train_indices], labels[train_indices], data[test_indices], labels[test_indices]

X_train, y_train, X_test, y_test = train_test_split(data, labels, test_size=0.2)

# Function to calculate Euclidean distance
def euclidean_distance(p1, p2):
    return np.sqrt(np.sum((p1 - p2)**2))

# Function to calculate Hamming distance
def hamming_distance(p1, p2):
    return np.sum(np.abs(p1 - p2))

# K-Nearest Neighbors prediction function
def predict(X_train, y_train, X_test, k, distance_metric):
    y_pred = []
    for test_sample in X_test:
        distances = [distance_metric(test_sample, train_sample) for train_sample in X_train]
        k_indices = np.argsort(distances)[:k]
        k_labels = [y_train[i] for i in k_indices]
        y_pred.append(max(k_labels, key=k_labels.count))
    return np.array(y_pred)

# Accuracy calculation
def accuracy(y_true, y_pred):
    return np.sum(y_true == y_pred) / len(y_true)

```

```python
# Visualize dataset
def visualize_data(X_train, y_train, X_test, y_test):
    plt.figure(figsize=(8, 6))
    plt.scatter(X_train[y_train == 0][:, 0], X_train[y_train == 0][:, 1], color='blue', label='Train Class 0')
    plt.scatter(X_train[y_train == 1][:, 0], X_train[y_train == 1][:, 1], color='green', label='Train Class 1')
    plt.scatter(X_test[y_test == 0][:, 0], X_test[y_test == 0][:, 1], color='cyan', label='Test Class 0', marker='x')
    plt.scatter(X_test[y_test == 1][:, 0], X_test[y_test == 1][:, 1], color='orange', label='Test Class 1', marker='x')
    plt.legend()
    plt.title('Dataset Visualization')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

visualize_data(X_train, y_train, X_test, y_test)

# Run the model using Hamming Distance
k = 5
y_pred_hamming = predict(X_train, y_train, X_test, k, hamming_distance)
accuracy_hamming = accuracy(y_test, y_pred_hamming)
print(f"Accuracy using Hamming Distance: {accuracy_hamming * 100:.2f}%")

# Run the model using Euclidean Distance
y_pred_euclidean = predict(X_train, y_train, X_test, k, euclidean_distance)
accuracy_euclidean = accuracy(y_test, y_pred_euclidean)
print(f"Accuracy using Euclidean Distance: {accuracy_euclidean * 100:.2f}%")

```

Visualization:



Result:

The K-Nearest Neighbors (KNN) model was implemented successfully using both Hamming and Euclidean distance metrics. The following results were observed:

1. **Accuracy using Hamming Distance**: The model achieved an accuracy of 100%. This is due to the clear separation between the clusters, making it easier for the Hamming Distance metric to classify samples accurately.
2. **Accuracy using Euclidean Distance**: The model also achieved an accuracy of 100%, reflecting the well-separated nature of the clusters and the suitability of Euclidean Distance for continuous numerical data.
3. **Visualization**: The dataset visualization confirmed the distinct separation between the two classes, which aligns with the perfect accuracy results for both metrics. This demonstrates the effectiveness of the KNN method on well-clustered datasets.
4. **Comparison of Metrics**: Both distance metrics yielded identical results in this experiment. However, the choice of metric may influence the performance in more complex datasets with overlapping classes or non-numerical features.

These findings demonstrate the KNN algorithm's reliability in scenarios where the dataset is well-structured and clusters are distinct. Further testing on more complex datasets would provide deeper insights into the performance differences between the distance metrics.

Conclusion: The KNN classification model demonstrated excellent performance in classifying a synthetic, well-separated dataset. Both Euclidean and Hamming distance metrics were effective, and the implementation validated the KNN algorithm’s capability to classify data based on proximity in feature space. This experiment highlights the importance of dataset characteristics in influencing model performance.