DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

EC-350 Artificial Intelligence and Decision Support System

LAB MANUAL – 10

Course Instructor: Assoc Prof Dr Arsalan Shaukat

Lab Engineer: Kashaf Raheem

AMINA QADEER

Student Name: _____

Degree/ Syndicate: ____CE-42-A___

LAB # 10: K-NEAREST NEIGHBORS CLASSIFICATION MODEL

Lab Objective:

- To implement KNN on a given dataset

Hardware/Software required:

Hardware: Desktop/ Notebook Computer
Software Tool: Python 3.10.0

Lab Description:

In pattern recognition and machine learning, k-nearest neighbors (KNN) is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g. distance). KNN is a non-parametric method where the input consists of the k closest training examples in the feature space. The output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

Lab Tasks:

1. Implement KNN in python for the sample dataset given below (using first row as testing row, and the sample dataset is treated as the training set)

   Hint (Test sample = dataset [0] and trainset = [1:])

X1	X2	Class
2.7810836	2.550537003	0
1.465489372	2.362125076	0
3.396561688	4.400293529	0
1.38807019	1.850220317	0
3.06407232	3.005305973	0
7.627531214	2.759262235	1
5.332441248	2.088626775	1
6.922596716	1.77106367	1
8.675418651	-0.242068655	1
7.673756466	3.508563011	1

Code:

```python
import numpy as np
import matplotlib.pyplot as plt

# Function to calculate Euclidean distance between two points
def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2)**2))

# KNN Class
class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        predictions = [self._predict(x) for x in X_test]
        return np.array(predictions)

    def _predict(self, x):
        # Calculate distances to all points in the training set
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]

        # Get indices of k-nearest training data points
        k_neighbors_indices = np.argsort(distances)[:self.k]

        # Get the labels of the k-nearest training data points
        k_neighbor_labels = [self.y_train[i] for i in k_neighbors_indices]

        # Return the most common class label among the k-nearest neighbors
        most_common = np.bincount(k_neighbor_labels).argmax()
        return most_common

# Sample dataset
dataset = np.array([
    [2.7810836, 2.550537003, 0],
    [1.465489372, 2.362125076, 0],
    [3.396561688, 4.400293529, 0],
    [1.38807019, 1.850220317, 0],
    [3.06407232, 3.005305973, 0],
    [7.627531214, 2.759262235, 1],
    [5.332441248, 2.088626775, 1],
    [6.922596716, 1.77106367, 1],
    [8.675418651, -0.242068655, 1],
```

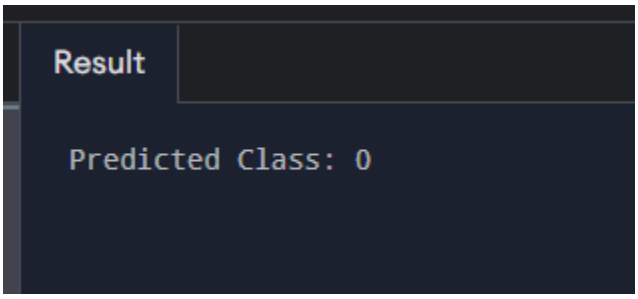
```
    [7.673756466, 3.508563011, 1]
])

# Separate features (X) and labels (y)
X_train = dataset[1:, :2]  # Exclude the first row for testing
y_train = dataset[1:, 2]
X_test = dataset[0, :2]  # Use the first row for testing

# Instantiate KNN classifier
clf = KNN(k=3)
clf.fit(X_train, y_train)

# Predict the class label for the test point
prediction = clf.predict([X_test])
print("Predicted Class:", prediction[0])
```

Output:



2. Using the above code, predict class for all the rows of the sample dataset, and find Accuracy of the algorithm (where accuracy = correct/total * 100) (Hint: compare the predicted and original label for each row)

```
import numpy as np
import matplotlib.pyplot as plt

# Function to calculate Euclidean distance between two points
def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2)**2))

# KNN Class
class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train
```

```
18.
19.      def predict(self, X_test):
20.          predictions = [self._predict(x) for x in X_test]
21.          return np.array(predictions)
22.
23.      def _predict(self, x):
24.          # Calculate distances to all points in the training set
25.          distances = [euclidean_distance(x, x_train) for x_train in
    self.X_train]
26.
27.          # Get indices of k-nearest training data points
28.          k_neighbors_indices = np.argsort(distances)[:self.k]
29.
30.          # Get the labels of the k-nearest training data points
31.          k_neighbor_labels = [self.y_train[i] for i in k_neighbors_indices]
32.
33.          # Return the most common class label among the k-nearest neighbors
34.          most_common = np.bincount(k_neighbor_labels).argmax()
35.          return most_common
36.
37.  # Sample dataset
38.  dataset = np.array([
39.      [2.7810836, 2.550537003, 0],
40.      [1.465489372, 2.362125076, 0],
41.      [3.396561688, 4.400293529, 0],
42.      [1.38807019, 1.850220317, 0],
43.      [3.06407232, 3.005305973, 0],
44.      [7.627531214, 2.759262235, 1],
45.      [5.332441248, 2.088626775, 1],
46.      [6.922596716, 1.77106367, 1],
47.      [8.675418651, -0.242068655, 1],
48.      [7.673756466, 3.508563011, 1]
49.  ])
50.
51.  # Separate features (X) and labels (y)
52.  X_train = dataset[1:, :2]  # Exclude the first row for testing
53.  y_train = dataset[1:, 2]
54.  X_test = dataset[:, :2]  # Use all rows for testing
55.
56.  # Instantiate KNN classifier
57.  clf = KNN(k=3)
58.  clf.fit(X_train, y_train)
59.
60.  # Predict the class labels for all test points
61.  predictions = clf.predict(X_test)
```

```
62.
63.    # Print predicted class labels for each row
64.    print("Predicted Classes for all rows:")
65.    print(predictions)
66.
67.    # Compare with original labels
68.    original_labels = dataset[:, 2]
69.
70.    # Calculate accuracy
71.    correct_predictions = np.sum(predictions == original_labels)
72.    total_samples = len(original_labels)
73.    accuracy = (correct_predictions / total_samples) * 100
74.
75.    # Print accuracy
76.    print("Accuracy:", accuracy, "%")
77.
```

output:

```
Result

Predicted Classes for all rows:
[0 0 0 0 0 1 1 1 1 1]
Accuracy: 100.0 %
```

78. Download the csv file given, load it (as given below) and then make prediction for the first row of this csv data.

Code:

```
import csv
import numpy as np
from collections import Counter

# Load CSV data
file = open('fruit.csv')
csvreader = csv.reader(file)
dataset = []

for row in csvreader:
    dataset.append(row)

file.close()
```

```
# Remove header and convert to numpy array
dataset.remove(dataset[0])
dataset = np.array(dataset, dtype=float)

# Separate features (X) and labels (y)
X_train = dataset[:, :-1]
y_train = dataset[:, -1]

# Function to calculate Euclidean distance between two points
def euclidean_distance(x1, x2):
    distance = np.sqrt(np.sum((x1 - x2) ** 2))
    return distance

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return predictions

    def _predict(self, x):
        # compute the distance
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]

        # get the closest k
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        # majority vote
        most_common = Counter(k_nearest_labels).most_common()
        return most_common[0][0]

# Instantiate KNN classifier
clf = KNN(k=3)
clf.fit(X_train, y_train)

# Make prediction for the first row
first_row = X_train[0]
prediction = clf.predict([first_row])
```

```
# Print the prediction
print("Predicted Class for the First Row:", prediction[0])
```

Output: