LAB # 03

K-NEAREST NEIGHBOR (KNN) ALGORITHM

OBJECTIVE

Implementing K-Nearest Neighbor (KNN) algorithm to classify the data set.

Lab Tasks:

Weather	Temperature	Play
Sunny	Hot	No
Sunny	Hot	No
Overcast	Hot	Yes
Rainy	Mild	Yes
Rainy	Cool	Yes
Rainy	Cool	No
Overcast	Cool	Yes
Sunny	Mild	No
Sunny	Cool	Yes
Rainy	Mild	Yes
Sunny	Mild	Yes
Overcast	Mild	Yes
Overcast	Hot	Yes
Rainy	Mild	No

Fig 1

1. Implement K-Nearest Neighbor (KNN) Algorithm on the above dataset in Fig 1 to predict whether the players can play or not when the weather is overcast and the temperature is mild.Also apply confusion Matrix.

OUTPUT

```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import pandas as pd

# Defining the dataset based on the image content
data = {
    "Outlook": ["Sunny", "Sunny", "Overcast", "Rainy", "Rainy", "Rainy", "Overcast", "Sunny", "Sunny", "Rainy",
                "Sunny", "Overcast", "Overcast", "Rainy"],
    "Temperature": ["Hot", "Hot", "Hot", "Mild", "Cool", "Cool", "Cool", "Mild", "Cool", "Mild",
                    "Mild", "Mild", "Hot", "Mild"],
    "Play": ["No", "No", "Yes", "Yes", "Yes", "No", "Yes", "No", "Yes", "Yes",
             "Yes", "Yes", "Yes", "No"]
}

# Convert dataset to DataFrame
df = pd.DataFrame(data)

# Encoding categorical variables
df['Outlook'] = df['Outlook'].map({"Sunny": 0, "Overcast": 1, "Rainy": 2})
df['Temperature'] = df['Temperature'].map({"Hot": 0, "Mild": 1, "Cool": 2})
df['Play'] = df['Play'].map({"No": 0, "Yes": 1})

# Splitting features and target
X = df[['Outlook', 'Temperature']]
y = df['Play']

# Initialize and train KNN model
k = 3  # number of neighbors
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X, y)

# Predicting for the specific test case (Outlook=Overcast, Temperature=Mild) => (1,1)
test_case = pd.DataFrame([[1, 1]], columns=['Outlook', 'Temperature'])  # Add column names
prediction = knn.predict(test_case)
print(f"Prediction for (Overcast, Mild): {'Yes' if prediction[0] == 1 else 'No'}")
```

```python
# Generating confusion matrix on the whole dataset as no separate test set is provided
y_pred = knn.predict(X)
conf_matrix = confusion_matrix(y, y_pred)
class_report = classification_report(y, y_pred, target_names=["No", "Yes"])

print("\nConfusion Matrix:")
print(conf_matrix)

print("\nClassification Report:")
print(class_report)
```

```
Prediction for (Overcast, Mild): Yes

Confusion Matrix:
[[3 2]
 [1 8]]

Classification Report:
              precision    recall  f1-score   support

          No       0.75      0.60      0.67         5
         Yes       0.80      0.89      0.84         9

    accuracy                           0.79        14
   macro avg       0.78      0.74      0.75        14
weighted avg       0.78      0.79      0.78        14
```

2. Here are 4 training samples. The two attributes are acid durability and strength. Now the factory produces a new tissue paper that passes laboratory test with X1=3 and X2=7. Predict the classification of this new tissue.

X1= Acid durability (sec)	X2=Strength (kg/m^2)	Y=Classification
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

- Calculate the Euclidean Distance between the query instance and all the training samples. Coordinate of query instance is (3,7).

In the Euclidean plane, if $\mathbf{p} = (p_1, p_2)$ and $\mathbf{q} = (q_1, q_2)$ then the distance is given by

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}.$$

Suppose K = number of nearest neighbors = 3, sort the distances and determine nearest neighbors. Gather the class (Y) of the nearest neighbors. Use majority of the category of nearest neighbors as the prediction value of the query instance

OUTPUT

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

# Training data (X1=Acid durability, X2=Strength, Y=Classification)
# Provided training samples
training_samples = [
    {"X1": 7, "X2": 7, "Y": "Bad"},
    {"X1": 7, "X2": 4, "Y": "Bad"},
    {"X1": 3, "X2": 4, "Y": "Good"},
    {"X1": 1, "X2": 4, "Y": "Good"}
]

# Query instance (new tissue): X1=3, X2=7
query_instance = (3, 7)

# Calculate Euclidean distance between query instance and each training sample
distances = []
for sample in training_samples:
    # Calculate Euclidean distance
    distance = np.sqrt((sample["X1"] - query_instance[0]) ** 2 + (sample["X2"] - query_instance[1]) ** 2)
    distances.append((distance, sample["Y"]))

# Sort distances and select k nearest neighbors
k = 3
nearest_neighbors = sorted(distances)[:k]

# Output distances and nearest neighbors
print("Distances to query instance (3,7):")
for dist, classification in distances:
    print(f"Distance: {dist:.2f}, Class: {classification}")

print("\nNearest neighbors (k=3):")
for dist, classification in nearest_neighbors:
    print(f"Distance: {dist:.2f}, Class: {classification}")

# Predict class using majority voting
classes = [neighbor[1] for neighbor in nearest_neighbors]
predicted_class = Counter(classes).most_common(1)[0][0]

print(f"\nPredicted class for the query instance (3, 7): {predicted_class}")
```

```
Distances to query instance (3,7):
Distance: 4.00, Class: Bad
Distance: 5.00, Class: Bad
Distance: 3.00, Class: Good
Distance: 3.61, Class: Good

Nearest neighbors (k=3):
Distance: 3.00, Class: Good
Distance: 3.61, Class: Good
Distance: 4.00, Class: Bad

Predicted class for the query instance (3, 7): Good
```

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

# Training data with specific values: (X1, X2, Y)
# Assume we have these four training samples
training_samples = [
    {"X1": 1, "X2": 1, "Y": "Good"},
    {"X1": 2, "X2": 4, "Y": "Good"},
    {"X1": 4, "X2": 6, "Y": "Bad"},
    {"X1": 5, "X2": 7, "Y": "Bad"}
]

# Query instance (the new tissue paper): X1=3, X2=7
query_instance = (3, 7)

# Calculate Euclidean distance between query instance and each training sample
distances = []
for sample in training_samples:
    # Calculate Euclidean distance
    distance = np.sqrt((sample["X1"] - query_instance[0]) ** 2 + (sample["X2"] - query_instance[1]) ** 2)
    distances.append((distance, sample["Y"]))

# Sort distances and select k nearest neighbors
k = 3
nearest_neighbors = sorted(distances)[:k]

# Output distances and nearest neighbors
print("Distances to query instance (3,7):")
for dist, classification in distances:
    print(f"Distance: {dist:.2f}, Class: {classification}")

print("\nNearest neighbors (k=3):")
for dist, classification in nearest_neighbors:
    print(f"Distance: {dist:.2f}, Class: {classification}")

# Predict class using majority voting
classes = [neighbor[1] for neighbor in nearest_neighbors]
predicted_class = Counter(classes).most_common(1)[0][0]

print(f"\nPredicted class for the query instance (3, 7): {predicted_class}")

# Comment on the prediction
if predicted_class == "ClassA":
    print("\nThis prediction is good if the nearest neighbors reliably indicate class membership.")
else:
    print("\nThis prediction may not be reliable if the classes are too close or overlap.")
```

```
Distances to query instance (3,7):
Distance: 6.32, Class: Good
Distance: 3.16, Class: Good
Distance: 1.41, Class: Bad
Distance: 2.00, Class: Bad

Nearest neighbors (k=3):
Distance: 1.41, Class: Bad
Distance: 2.00, Class: Bad
Distance: 3.16, Class: Good

Predicted class for the query instance (3, 7): Bad
```

HOME Tasks:

OUTPUT#01

```python
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from collections import Counter

# Generate a random dataset similar to the one in the image
np.random.seed(0)

# Possible values for each feature
weather_options = ["Sunny", "Overcast", "Rainy"]
temperature_options = ["Hot", "Mild", "Cool"]
play_options = ["Yes", "No"]

# Generate 14 random samples
n_samples = 14
data = {
    "Outlook": np.random.choice(weather_options, n_samples),
    "Temperature": np.random.choice(temperature_options, n_samples),
    "Play": np.random.choice(play_options, n_samples)
}

# Create DataFrame
df = pd.DataFrame(data)

# Encode categorical variables
df['Outlook'] = df['Outlook'].map({"Sunny": 0, "Overcast": 1, "Rainy": 2})
df['Temperature'] = df['Temperature'].map({"Hot": 0, "Mild": 1, "Cool": 2})
df['Play'] = df['Play'].map({"No": 0, "Yes": 1})

# Split data into features and target
X = df[['Outlook', 'Temperature']]
y = df['Play']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Initialize and train KNN model
k = 3  # number of nearest neighbors
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Predict for the specific test case (Outlook=Overcast, Temperature=Mild) => (1,1)
query_instance = pd.DataFrame([[1, 1]], columns=['Outlook', 'Temperature'])
prediction = knn.predict(query_instance)
print(f"Prediction for (Overcast, Mild): {'Yes' if prediction[0] == 1 else 'No'}")

# Evaluate model performance using the test set
y_pred = knn.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred, target_names=["No", "Yes"])

# Print results
print("\nConfusion Matrix:")
print(conf_matrix)

print("\nClassification Report:")
print(class_report)
```

OUTPUT#02

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

# Training data (X1=Acid durability, X2=Strength, Y=Classification)
# Provided training samples
training_samples = [
    {"X1": 7, "X2": 7, "Y": "Bad"},
    {"X1": 7, "X2": 4, "Y": "Bad"},
    {"X1": 3, "X2": 4, "Y": "Good"},
    {"X1": 1, "X2": 4, "Y": "Good"}
]

# Query instance (new tissue): X1=3, X2=7
query_instance = (3, 7)

# Calculate Euclidean distance between query instance and each training sample
distances = []
for sample in training_samples:
    # Calculate Euclidean distance
    distance = np.sqrt((sample["X1"] - query_instance[0]) ** 2 + (sample["X2"] - query_instance[1]) ** 2)
    distances.append((distance, sample["Y"]))

# Sort distances and select k nearest neighbors
k = 3
nearest_neighbors = sorted(distances)[:k]

# Output distances and nearest neighbors
print("Distances to query instance (3,7):")
for dist, classification in distances:
    print(f"Distance: {dist:.2f}, Class: {classification}")

print("\nNearest neighbors (k=3):")
for dist, classification in nearest_neighbors:
    print(f"Distance: {dist:.2f}, Class: {classification}")

# Predict class using majority voting
classes = [neighbor[1] for neighbor in nearest_neighbors]
predicted_class = Counter(classes).most_common(1)[0][0]

print(f"\nPredicted class for the query instance (3, 7): {predicted_class}")
```

```
Distances to query instance (3,7):
Distance: 4.00, Class: Bad
Distance: 5.00, Class: Bad
Distance: 3.00, Class: Good
Distance: 3.61, Class: Good

Nearest neighbors (k=3):
Distance: 3.00, Class: Good
Distance: 3.61, Class: Good
Distance: 4.00, Class: Bad

Predicted class for the query instance (3, 7): Good
```