ROHAN AHMED 2022F-BSE-201 SECTION: E

# 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 |

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.

```
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split

weather = ['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']
```

```
le_weather = preprocessing.LabelEncoder()
le_temperature = preprocessing.LabelEncoder()
le_play = preprocessing.LabelEncoder()

weather_encoded = le_weather.fit_transform(weather)
temperature_encoded = le_temperature.fit_transform(temperature)
play_encoded = le_play.fit_transform(play)

features = list(zip(weather_encoded, temperature_encoded))

features_train, features_test, label_train, label_test = train_test_split(
    features, play_encoded, test_size=0.2, random_state=42
)


model = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
model.fit(features_train, label_train)


predicted = model.predict(features_test)


new_weather = 'Overcast'
new_temperature = 'Mild'
new_weather_encoded = le_weather.transform([new_weather])[0]
new_temperature_encoded = le_temperature.transform([new_temperature])[0]
new_data = [[new_weather_encoded, new_temperature_encoded]]

single_prediction = model.predict(new_data)
print("Prediction for weather '{}' and temperature '{}': {}".format(
    new_weather, new_temperature, le_play.inverse_transform(single_prediction)[0]
))

conf_mat = confusion_matrix(label_test, predicted)
print("Confusion Matrix:")
print(conf_mat)

accuracy = accuracy_score(label_test, predicted)
print("Accuracy:", accuracy)
```

**OUTPUT:**

```
Prediction for weather 'Overcast' and temperature 'Mild': Yes
Confusion Matrix:
[[1 0]
 [1 1]]
Accuracy: 0.6666666666666666
```

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/m²) | 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

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

training_samples = np.array([
    [7, 7, 'Bad'],
    [7, 4, 'Bad'],
    [3, 4, 'Good'],
    [1, 4, 'Good']
], dtype=object)

query_instance = np.array([3, 7])

def euclidean_distance(sample, query):
    return float(np.sqrt(np.sum((sample[:2].astype(float) - query) ** 2)))

distances = []
for sample in training_samples:
    distance = euclidean_distance(sample, query_instance)
    distances.append((distance, sample[2]))
    print(f"Distance to {sample[:2]}: {distance:.2f}")

distances.sort(key=lambda x: x[0])
print("\nSorted distances:", distances)

K = 3
print(f"\nK = {K}")
nearest_neighbors = distances[:K]
print(f"\nThe {K} nearest neighbors:")

for i, (dist, label) in enumerate(nearest_neighbors, 1):
    print(f"{i}. Distance: {dist:.2f}, {label}")

classes = [neighbor[1] for neighbor in nearest_neighbors]
class_counts = Counter(classes)
for cls, count in class_counts.items():
    print(f"{count} ({cls})", end=". ")
```

```
prediction = class_counts.most_common(1)[0][0]

print(f"\n\nPredicted classification for query instance (3, 7): {prediction}")
```

**OUTPUT**:

```
Distance to [7 7]: 4.00
Distance to [7 4]: 5.00
Distance to [3 4]: 3.00
Distance to [1 4]: 3.61

Sorted distances: [(3.0, 'Good'), (3.605551275463989, 'Good'), (4.0, 'Bad'), (5.0, 'Bad')]

K = 3

The 3 nearest neighbors:
1. Distance: 3.00, Good
2. Distance: 3.61, Good
3. Distance: 4.00, Bad
2 (Good). 1 (Bad).

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

## HOME TASK:

```
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split

age = ['Young', 'Young', 'Middle-Aged', 'Middle-Aged', 'Old', 'Old',
       'Young', 'Middle-Aged', 'Old', 'Middle-Aged', 'Young', 'Old',
       'Middle-Aged', 'Young']
income = ['Low', 'Low', 'Medium', 'High', 'Medium', 'High',
          'Low', 'Medium', 'High', 'Medium', 'Low', 'Medium',
          'High', 'Low']
buy = ['No', 'No', 'Yes', 'Yes', 'Yes', 'No',
       'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes',
       'Yes', 'No']

le_age = preprocessing.LabelEncoder()
le_income = preprocessing.LabelEncoder()
le_buy = preprocessing.LabelEncoder()

age_encoded = le_age.fit_transform(age)
income_encoded = le_income.fit_transform(income)
buy_encoded = le_buy.fit_transform(buy)

features = list(zip(age_encoded, income_encoded))

features_train, features_test, label_train, label_test = train_test_split(
    features, buy_encoded, test_size=0.2, random_state=42
)

model = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
model.fit(features_train, label_train)

predicted = model.predict(features_test)

new_age = 'Middle-Aged'
new_income = 'High'
new_age_encoded = le_age.transform([new_age])[0]
new_income_encoded = le_income.transform([new_income])[0]
new_data = [[new_age_encoded, new_income_encoded]]
```

```
single_prediction = model.predict(new_data)
print("Prediction for age '{}' and income '{}': {}".format(
    new_age, new_income, le_buy.inverse_transform(single_prediction)[0]
))

conf_mat = confusion_matrix(label_test, predicted)
print("Confusion Matrix:")
print(conf_mat)

accuracy = accuracy_score(label_test, predicted)
print("Accuracy:", accuracy)
```

**OUTPUT:**

```
Prediction for age 'Middle-Aged' and income 'High': Yes
Confusion Matrix:
[[1 0]
 [0 2]]
Accuracy: 1.0
```