# LAB # 03

# K-NEAREST NEIGHBOR (KNN) ALGORITHM

## OBJECTIVE

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

## Lab Tasks:

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
#Assigning Features ad Label variables
data = {
    "Weather" : ["sunny","suuny","ovecast","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"]
}
df = pd.DataFrame(data)
```

```
#PErforming Label Encoding
le = preprocessing.LabelEncoder()
weather_Encoded = le.fit_transform(df["Weather"])
temperature_Encoded = le.fit_transform(df["Temperature"])
play_Encoded = le.fit_transform(df["Play"])
```

```
#Combining all features
features = list(zip(weather_Encoded, temperature_Encoded, play_Encoded))
```

```
#Spliting Dataset into training and testing
from sklearn. model_selection import train_test_split
features_train, features_test,label_train,label_test = train_test_split(features, play_Encoded, test_size = 0.5, random_state = 50)
```

```python
#Generate a model using K-Neighbors classifier
model = KNeighborsClassifier(n_neighbors = 3, metric = 'euclidean')
#Fit the dataset on classifier
model.fit(features_train, label_train)
#Perform prediction
predicted = model.predict(features_test)
#Print prediction
print("Prediction:", predicted)
```

```
Prediction: [1 0 0 0 1 0 0]
```

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

```
Confusion Matrix:
[[1 0]
 [4 2]]
```

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

```
Accuracy: 0.8333333333333334
```

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

```python
#22F-BSE-138
#Saud Hassan
import math
from collections import Counter

# Step 1: Define training samples with (X1, X2) and their classification (Y)
training_samples = [
    {"X1": 7, "X2": 7, "Y": "A"},
    {"X1": 7, "X2": 4, "Y": "A"},
    {"X1": 3, "X2": 4, "Y": "B"},
    {"X1": 1, "X2": 4, "Y": "B"},
]

# Query instance (coordinates)
query_instance = {"X1": 3, "X2": 7}

# Step 2: Define a function to calculate Euclidean distance
def euclidean_distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

# Step 3: Calculate distances from query instance to each training sample
distances = []
for sample in training_samples:
    dist = euclidean_distance(
        query_instance["X1"], query_instance["X2"],
        sample["X1"], sample["X2"]
    )
    distances.append((dist, sample["Y"]))

# Step 4: Sort distances and select k-nearest neighbors (k=3)
k = 3
nearest_neighbors = sorted(distances)[:k]

# Step 5: Gather the classes of the nearest neighbors
neighbor_classes = [neighbor[1] for neighbor in nearest_neighbors]

# Step 6: Use majority voting to predict the class
predicted_class = Counter(neighbor_classes).most_common(1)[0][0]

# Display the result
print("Predicted Class:", predicted_class)
print("Nearest Neighbors:", nearest_neighbors)
```

```
Predicted Class: B
Nearest Neighbors: [(3.0, 'B'), (3.605551275463989, 'B'), (4.0, 'A')]
```