Machine Learning Lab (PMCA507P)

> **Reg No:** 23MCA1030

> **Name :** Vinayak Kumar Singh

Exercise 6 KNN Classification (Split the data set)

Collab url : https://colab.research.google.com/drive/1zC8u7kGecK7kBOBhMzNzfhr7GEc3oUnz?usp=sharing

Dataset url : https://www.kaggle.com/datasets/uciml/iris?resource=download

Import necessary libraries

```
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
```

Load the dataset (iris.csv)

```
df=pd.read_csv("/content/Iris.csv")
```

```
df
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

Next steps: Generate code with df View recommended plots

Split dataset into test and train (20:80).

```
from sklearn.model_selection import train_test_split

X = df.iloc[:, 0:4]
y = df.iloc[:, 4]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (120, 4)
y_train shape: (120,)
X_test shape: (30, 4)
y_test shape: (30,)
```

KNN classifiers with k values of 2 and 4 & Evaluate using the appropriate metrics

```
from sklearn.preprocessing import LabelEncoder

# Encoding the target variable
le = LabelEncoder()
y = le.fit_transform(df['Species'])

# Split the data into training and testing sets
X = df.iloc[:, 0:4]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train KNN models with k=2 and k=4
knn2 = KNeighborsClassifier(n_neighbors=2)
knn2.fit(X_train, y_train)

knn4 = KNeighborsClassifier(n_neighbors=4)
knn4.fit(X_train, y_train)

# Print accuracy for both models
print("Accuracy of KNN with k=2:", knn2.score(X_test, y_test))
print("Accuracy of KNN with k=4:", knn4.score(X_test, y_test))
```

```
Accuracy of KNN with k=2: 1.0
Accuracy of KNN with k=4: 1.0
```

Distance Calculation and Nearest Neighbors using Euclidean distance

```
import numpy as np

def euclidean_distance(x, y):

  # Calculating the squared difference between each dimension
  diff = np.square(x - y)

  # Sum the squared differences
  sum_of_squared_diff = np.sum(diff)

  # Taking the square root of the sum
  distance = np.sqrt(sum_of_squared_diff)

  return distance

# Example
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

distance = euclidean_distance(x, y)

print("Euclidean distance between x and y:", distance)
```

```
Euclidean distance between x and y: 5.196152422706632
```

Method for Finding Nearest Neighbors

```
def find_nearest_neighbors(data, target_point, k):

  # Calculating the distances between the target point and all other points
  distances = np.linalg.norm(data - target_point, axis=1)

  # Sort the distances in ascending order
  sorted_indices = np.argsort(distances)

  # Select the k nearest neighbors
  nearest_neighbors = data[sorted_indices[:k]]

  return nearest_neighbors

# Example
data = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])
target_point = np.array([4, 5])
k = 2
```

```
nearest_neighbors = find_nearest_neighbors(data, target_point, k)

print("Nearest neighbors of", target_point, ":")
for neighbor in nearest_neighbors:
  print(neighbor)
```

```
Nearest neighbors of [4 5] :
[3 4]
[5 6]
```

Method predicting the data point using the above two methods

```
def predict_data_point(data, query_point, k):

  # Calculating Euclidean distances between the query point and all training points.
  distances = np.sqrt(np.sum((data - query_point) ** 2, axis=1))

  # Find the k nearest neighbors.
  nearest_neighbors = data[np.argsort(distances)[:k]]

  # Calculating the average of the k nearest neighbors.
  prediction = np.mean(nearest_neighbors, axis=0)

  return prediction
```