**Practical Number 8**

**Aim:** K-Means Clustering, Decision Tree and Multilayer Perceptron using scikit-learn's built-in dataset, Iris Flowers.

**Software Used :** Pycharm Community Edition 2023.1, Python(3.9.12)

**Theory:**

K-Means Clustering: K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

Decision Tree: Decision Tree is a Supervised learning techniquethat can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, whereinternal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome**.** In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node**.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

Multilayer Perceptron: Multi-Layer perceptron defines the most complex architecture of artificial neural networks. It is substantially formed from multiple layers of the perceptron. TensorFlow is a very popular deep learning framework released by, and this notebook will guide to build a neural network with this library. If we want to understand what is a Multi-layer perceptron, we have to develop a multi-layer perceptron from scratch using Numpy.

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.tree import DecisionTreeClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Load the Iris Flowers dataset

iris = load\_iris()

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.2)

# K-Means Clustering

kmeans = KMeans(n\_clusters=3, random\_state=0)

kmeans.fit(X\_train)

kmeans\_pred = kmeans.predict(X\_test)

kmeans\_accuracy = accuracy\_score(y\_test, kmeans\_pred)

# Decision Tree

dt = DecisionTreeClassifier()

dt.fit(X\_train, y\_train)

dt\_pred = dt.predict(X\_test)

dt\_accuracy = accuracy\_score(y\_test, dt\_pred)

# Multilayer Perceptron

mlp = MLPClassifier(hidden\_layer\_sizes=(10, 10, 10), max\_iter=1000)

mlp.fit(X\_train, y\_train)

mlp\_pred = mlp.predict(X\_test)

mlp\_accuracy = accuracy\_score(y\_test, mlp\_pred)

# Plotting the actual output, predicted output by K-Means, Decision Tree, and Multilayer Perceptron

fig, axs = plt.subplots(2, 2)

fig.suptitle('Actual vs Predicted Output\nAccuracy: K-Means {0:.2f}, Decision Tree {1:.2f}, MLP {2:.2f}'.format(kmeans\_accuracy, dt\_accuracy, mlp\_accuracy))

axs[0, 0].set\_title('Actual Output')

axs[0, 0].scatter(X\_test[:, 0], X\_test[:, 1], c=y\_test)

axs[0, 1].set\_title('K-Means')

axs[0, 1].scatter(X\_test[:, 0], X\_test[:, 1], c=kmeans\_pred)

axs[1, 0].set\_title('Decision Tree')

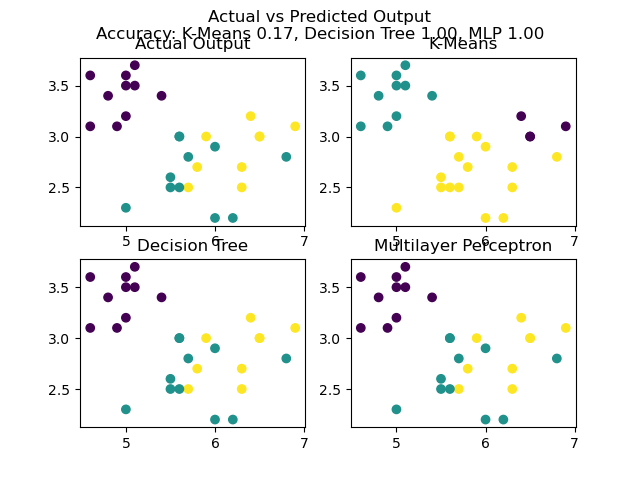
axs[1, 0].scatter(X\_test[:, 0], X\_test[:, 1], c=dt\_pred)

axs[1, 1].set\_title('Multilayer Perceptron')

axs[1, 1].scatter(X\_test[:, 0], X\_test[:, 1], c=mlp\_pred)

plt.show()

**Results:**



**Conclusion** : In this practical, we have successfully studied and implemented K-Means Clustering, Decision Tree and Multilayer Perceptron using scikit-learn's built-in dataset, Iris Flowers.