**Practical No: 10**

**Distance methods with Prediction | K – Means Clustering.**

**10 A) Aim: Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix.**

**Code and Output:**

import numpy as np

import pandas as pd

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

#Load the dataset

df = pd.read\_csv("../dataset/Iris.csv")

#quick look into the data

print(df.head(5))

print("\n")

#Separate data and label

x = df.drop(['Species'], axis=1)

y = df['Species']

#Prepare data for classification process

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=0)

#Create a model , p = 2 => Euclidean Distance:

knn = KNeighborsClassifier(n\_neighbors = 6, p = 2, metric='minkowski')

#Train the model

knn.fit(x\_train, y\_train)

# Calculate the accuracy of the model

print("Accuracy of Euclidean Distance model:-")

print(knn.score(x\_test, y\_test))

y\_pred = knn.predict(x\_test)

#confusion matrix

from sklearn.metrics import confusion\_matrix

cm=np.array(confusion\_matrix(y\_test,y\_pred))

print("-"\*50)

print("Confusion matrix:-")

print(cm)

print("\n")

#Create a model , p = 1 => Manhattan Distance

knn = KNeighborsClassifier(n\_neighbors = 6, p = 1, metric='minkowski')

#Train the model

knn.fit(x\_train, y\_train)

# Calculate the accuracy of the model

print("-"\*50)

print("Accuracy of Manhattan Distance model:-")

print(knn.score(x\_test, y\_test))

y\_pred = knn.predict(x\_test)

#confusion matrix

from sklearn.metrics import confusion\_matrix

cm=np.array(confusion\_matrix(y\_test,y\_pred))

print("-"\*50)

print("Confusion matrix:-")

print(cm)

print("\n")

#Create a model ,p = ∞, Chebychev Distance

#let ∞ = 10000

knn = KNeighborsClassifier(n\_neighbors = 6, p = 10000, metric='minkowski')

#Train the model

knn.fit(x\_train, y\_train)

# Calculate the accuracy of the model

print("-"\*50)

print("Accuracy of Chebychev Distance model:-")

print(knn.score(x\_test, y\_test))

y\_pred = knn.predict(x\_test)

#confusion matrix

from sklearn.metrics import confusion\_matrix

cm=np.array(confusion\_matrix(y\_test,y\_pred))

print("-"\*50)

print("Confusion matrix:-")

print(cm)

print("\n")

**A black screen with white text

Description automatically generated**

**A computer screen with white text

Description automatically generated**

**A computer screen with white text

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**10B: AIM: Implement the classification model using K-means clustering with Prediction, Test score and Confusion Matrix.**

**Description:**

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.

**Code and output:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import sklearn

#Import the dataset and slice the important features

dataset = pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:, [3,4]].values

#Find the optimal k value for clustering the data.

from sklearn.cluster import KMeans

wcss = []

for i in range(1,11):

kmeans = KMeans(n\_clusters=i, init='k-means++',random\_state=42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1,11),wcss)

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

A graph with a blue line

Description automatically generated

#The point at which the elbow shape is created is 5.

kmeans = KMeans(n\_clusters=5,init="k-means++",random\_state=42)

y\_kmeans = kmeans.fit\_predict(X)

plt.scatter(X[y\_kmeans == 0,0], X[y\_kmeans == 0,1], s = 60, c = 'red', label = 'Cluster1')

plt.scatter(X[y\_kmeans == 1,0], X[y\_kmeans == 1,1], s = 60, c = 'blue', label = 'Cluster2')

plt.scatter(X[y\_kmeans == 2,0], X[y\_kmeans == 2,1], s = 60, c = 'green', label = 'Cluster3')

plt.scatter(X[y\_kmeans == 3,0], X[y\_kmeans == 3,1], s = 60, c = 'violet', label = 'Cluster4')

plt.scatter(X[y\_kmeans == 4,0], X[y\_kmeans == 4,1], s = 60, c = 'yellow', label = 'Cluster5')

plt.scatter(kmeans.cluster\_centers\_[:,0], kmeans.cluster\_centers\_[:,1],s=100,c='black',label='Centroids')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100')

plt.legend()

plt.show()

A chart of colored dots

Description automatically generated

Learning:

This code snippet demonstrates the implementation of K-Means clustering on a Mall Customers dataset using Python's scikit-learn library. It first imports necessary modules and reads the dataset, selecting two key features – Annual Income and Spending Score. The optimal number of clusters (k) is determined by plotting the Within-Cluster-Sum-of-Squares (WCSS) against different k values. In this case, the elbow method suggests k=5. The K-Means algorithm is then applied, and the clusters are visualized with a scatter plot, showcasing distinct clusters based on customers' Annual Income and Spending Score. The black points represent cluster centroids, providing insights into customer segmentation for targeted business strategies.