EXPERIMENT 5

Aim: Program to demonstrate k–Nearest Neighbours flowers classification.

Theory:

The k-Nearest Neighbours (k-NN) algorithm is a supervised learning technique used for both classification and regression problems. It is one of the simplest machine learning algorithms, based on the principle of similarity. The fundamental idea behind k-NN is that data points with similar features tend to belong to the same category.
k-NN is a lazy learning algorithm, meaning it does not create an explicit model during the training phase. Instead, it stores the training data and classifies new instances by finding the k closest training examples.

Key Characteristics of k-NN:

- *Instance-based Learning:* The algorithm memorizes training data rather than learning a general model.
- *Non-parametric:* No assumption is made about the underlying data distribution.
- *Versatile:* Can be used for both classification and regression tasks.
- *Computational Complexity:* Increases with dataset size, as it requires searching for nearest neighbours.

Algorithm:

1. Choose the value of k (the number of nearest neighbours to consider).
2. Calculate the distance between the test sample and all training samples using a distance metric such as:
    - Euclidean Distance (most common):
      $$d = \sqrt{\left(x_2 - x_1\right)^2 + \left(y_2 - y_1\right)^2}$$
    - Manhattan Distance
    - Minkowski Distance

3. Find the k nearest neighbours (smallest distances).
4. Assign the class label based on majority voting from the k-nearest neighbours.
5. Return the predicted class.

Dataset Description:

The Iris dataset is a well-known dataset used for classification tasks. It contains 150 samples of iris flowers from three species:

- Setosa
- Versicolor
- Virginica

Each sample has four features:

1. Sepal Length (cm)
2. Sepal Width (cm)
3. Petal Length (cm)
4. Petal Width (cm)

Target Variable:

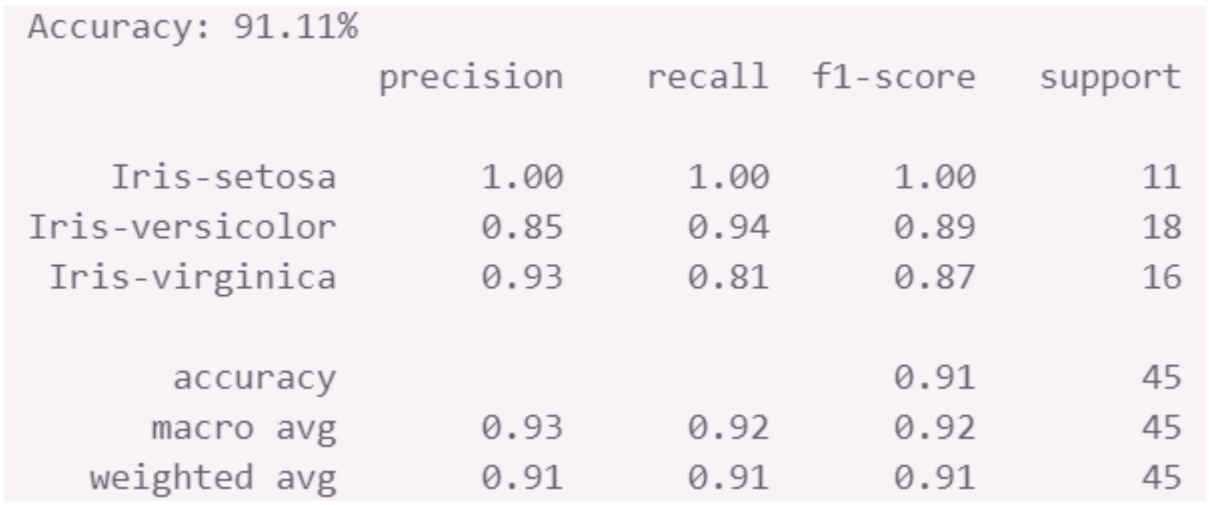
- Species (Categorical: Setosa, Versicolor, Virginica)

Dataset Source:
The dataset is originally from Fisher’s Iris dataset and is widely used for pattern recognition in Machine Learning.

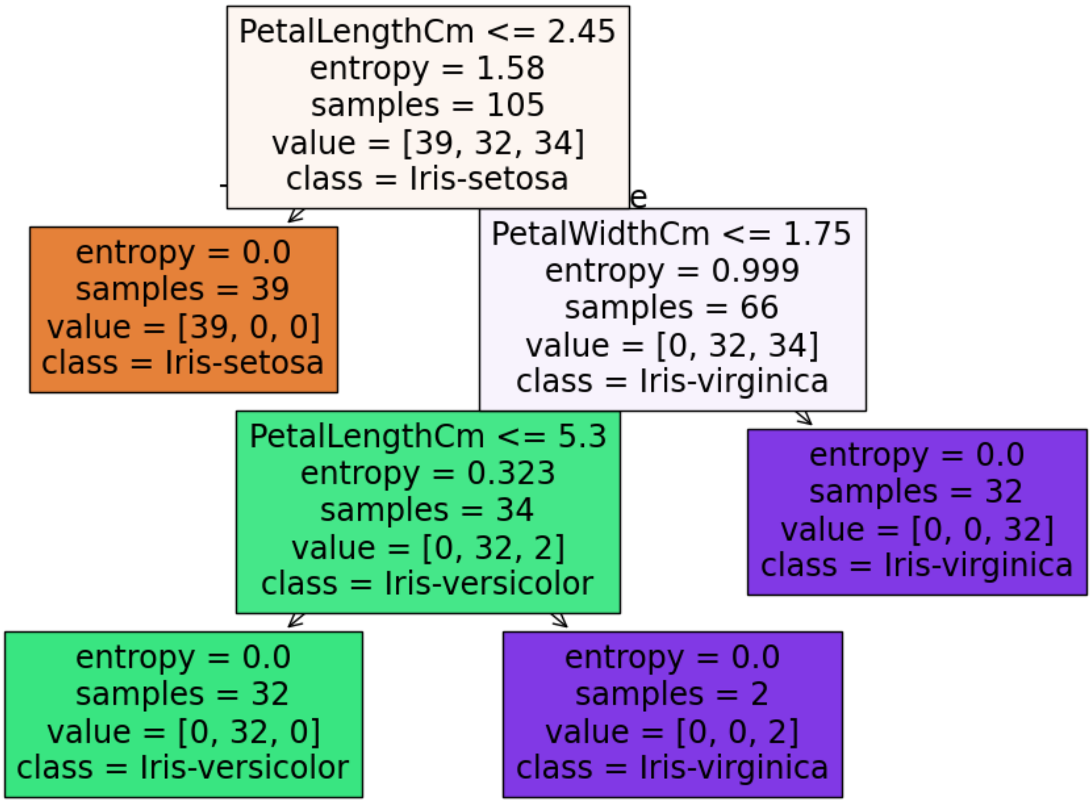
Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, classification_report

dataset = pd.read_csv("Iris.csv")

X = dataset[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"]]
y = dataset["Species"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=41)
model = DecisionTreeClassifier(criterion="entropy")
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print(classification_report(y_test, y_pred))

plt.figure(figsize=(12,8))
plot_tree(model, feature_names=X.columns, class_names=model.classes_, filled=True)
plt.show()
```

Output:



```
Accuracy: 91.11%
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.85      0.94      0.89        18
 Iris-virginica       0.93      0.81      0.87        16

       accuracy                           0.91        45
      macro avg       0.93      0.92      0.92        45
   weighted avg       0.91      0.91      0.91        45
```



Learning Outcomes:

EXPERIMENT 6

Aim: Program to demonstrate Naive–Beyes classifier.

Theory:

The Naïve Bayes classifier is a probabilistic machine learning algorithm based on Bayes' Theorem. It is widely used for classification tasks, especially in text classification, spam filtering, and medical diagnosis.
Despite its simplicity, Naïve Bayes is highly efficient and performs well in many real-world applications. The key assumption of this model is that features are independent given the class label, which simplifies calculations but may not always hold in practice.

Bayes' Theorem
The classifier relies on Bayes' theorem, which is given by:

$$P(C \vee X) = \frac{P(X \vee C)P(C)}{P(X)}$$

Where: $P(C \vee X)$ = Probability of class C given features X \(posterior probability\).
$P(X \vee C)$ = Probability of features X given class C \(likelihood\).
$P(C)$ = Probability of class C occurring \(prior probability\).
$P(X)$ = Probability of features X across all classes \(evidence\).
The classifier selects the class C that maximizes $P(C \vee X)$.

Algorithm:

1. Calculate Prior Probabilities for each class.
2. Compute Likelihoods for each feature given the class.
3. Apply Bayes' Theorem to determine the posterior probability.
4. Classify the instance into the class with the highest probability.

Dataset Description:

The Iris dataset is a well-known dataset used for classification tasks. It contains 150 samples of iris flowers from three species:

- Setosa
- Versicolor
- Virginica

Each sample has four features:

1. Sepal Length (cm)
2. Sepal Width (cm)
3. Petal Length (cm)
4. Petal Width (cm)

Target Variable:

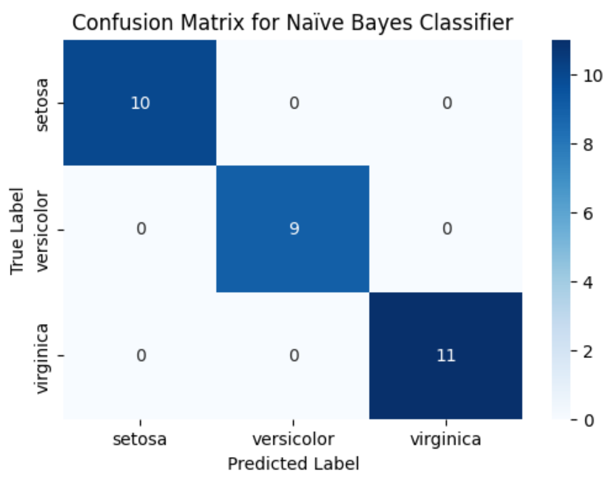
- Species (Categorical: Setosa, Versicolor, Virginica)

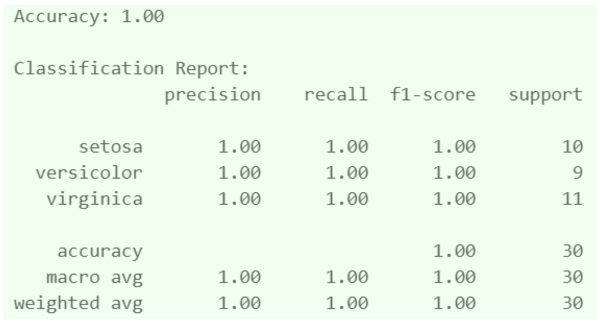
Dataset Source:
The dataset is originally from Fisher's Iris dataset and is widely used for pattern recognition in Machine Learning.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

iris = datasets.load_iris()

X = iris.data  # Features
y = iris.target  # Target labels

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

nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt="d", xticklabels=iris.target_names,
yticklabels=iris.target_names)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix for Naïve Bayes Classifier")
plt.show()
```

Output:





```
Accuracy: 1.00

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Learning Outcomes:

EXPERIMENT 7

Aim: Program to demonstrate PCA and LDA on Iris Dataset.

Theory:

Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) are dimensionality reduction techniques used in machine learning and data analysis.

- PCA is an unsupervised technique that transforms data into a lower-dimensional space while preserving as much variance as possible. It helps in feature extraction and visualization.
- LDA is a supervised technique that maximizes class separability, making it useful for classification tasks.

Both techniques reduce the number of features while retaining important information, making models more efficient and reducing computational costs.

PCA Algorithm:

1. Standardize the dataset.
2. Compute the covariance matrix.
3. Compute the eigenvalues and eigenvectors of the covariance matrix.
4. Select the top k eigenvectors corresponding to the k largest eigenvalues.
5. Transform the original dataset into the new k-dimensional space.

LDA Algorithm:

1. Compute the mean vectors for each class.
2. Compute the scatter matrices (within-class and between-class scatter).
3. Compute the eigenvalues and eigenvectors of the scatter matrix.
4. Select the top k eigenvectors corresponding to the largest eigenvalues.
5. Transform the dataset into the new k-dimensional space.

Dataset Description:

The Iris dataset is a well-known dataset used for classification tasks. It contains 150 samples of iris flowers from three species:

- Setosa
- Versicolor
- Virginica

Each sample has four features:

1. Sepal Length (cm)
2. Sepal Width (cm)
3. Petal Length (cm)
4. Petal Width (cm)

Target Variable:

- Species (Categorical: Setosa, Versicolor, Virginica)

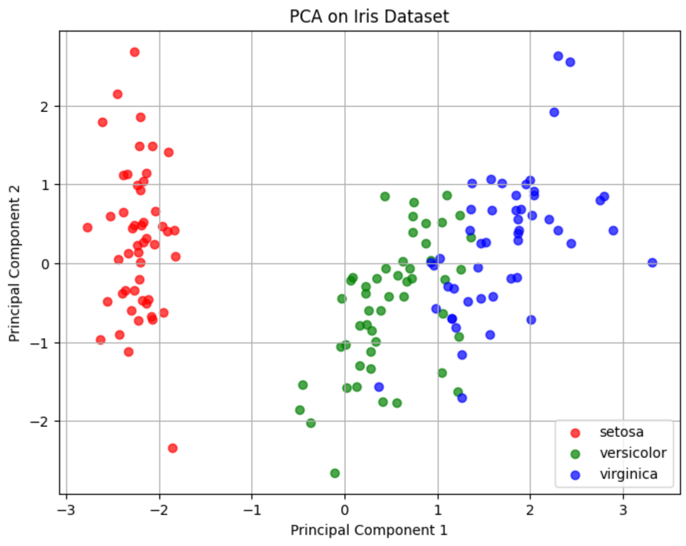
Dataset Source:

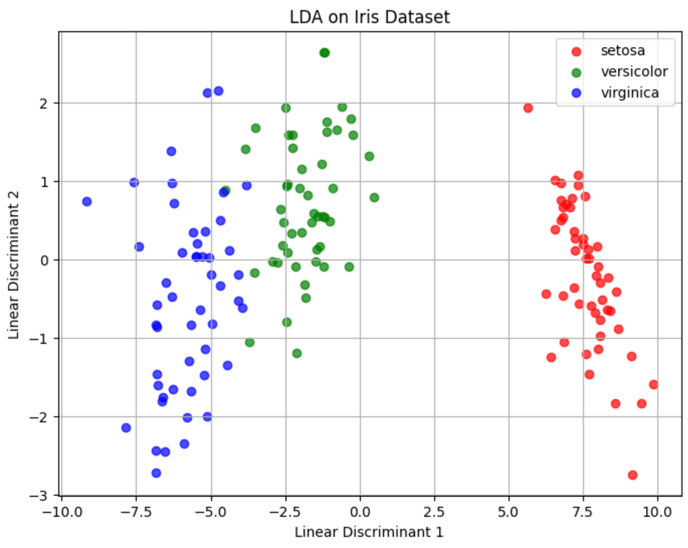
The dataset is originally from Fisher’s Iris dataset and is widely used for pattern recognition in Machine Learning.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
X = iris.data  # Features
y = iris.target  # Target labels
target_names = iris.target_names

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=2)  # Reduce to 2D for visualization
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
colors = ['red', 'green', 'blue']
for i, target_name in enumerate(target_names):
        plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1], label=target_name,   color=colors[i],
alpha=0.7)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA on Iris Dataset")
plt.legend()
plt.grid()
plt.show()

lda = LDA(n_components=2)  # Reduce to 2D for visualization
X_lda = lda.fit_transform(X, y)

plt.figure(figsize=(8, 6))
for i, target_name in enumerate(target_names):
        plt.scatter(X_lda[y == i, 0], X_lda[y == i, 1], label=target_name,    color=colors[i],
alpha=0.7)
plt.xlabel("Linear Discriminant 1")
plt.ylabel("Linear Discriminant 2")
plt.title("LDA on Iris Dataset")
plt.legend()
plt.grid()
plt.show()
```

Output:





Learning Outcomes:

EXPERIMENT 8

Aim: Program to demonstrate DBSCAN algorithm.

Theory:

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised machine learning algorithm used for clustering data points based on density. Unlike k-Means, which requires a predefined number of clusters, DBSCAN groups data based on high-density regions, making it effective for discovering clusters of arbitrary shape.

Key Characteristics of DBSCAN:

- Can identify clusters of arbitrary shape (unlike k-Means, which assumes spherical clusters).
- Can detect outliers (marked as noise).
- Does not require specifying the number of clusters.

Algorithm:

1. Select a point that is not yet classified.
2. Find all neighbors within a distance of ε.
3. If the point has at least MinPts neighbors, mark it as a core point and expand the cluster.
4. If it has fewer than MinPts neighbors, mark it as noise (outlier).
5. Repeat the process for all points until all points are classified.

Dataset Description:

The Iris dataset is a well-known dataset used for classification tasks. It contains 150 samples of iris flowers from three species:

- Setosa
- Versicolor
- Virginica

Each sample has four features:

1. Sepal Length (cm)
2. Sepal Width (cm)
3. Petal Length (cm)
4. Petal Width (cm)

Target Variable:

- Species (Categorical: Setosa, Versicolor, Virginica)

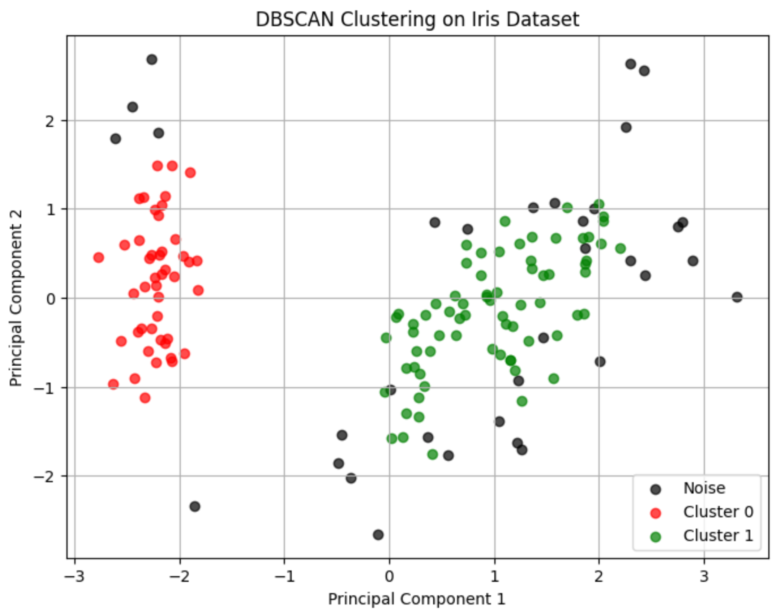
Dataset Source:

The dataset is originally from Fisher's Iris dataset and is widely used for pattern recognition in Machine Learning.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA

iris = datasets.load_iris()
X = iris.data
target_names = iris.target_names

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

dbscan = DBSCAN(eps=0.5, min_samples=5)
clusters = dbscan.fit_predict(X_scaled)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
unique_clusters = np.unique(clusters)
colors = ['red', 'green', 'blue', 'black']
for cluster in unique_clusters:
    plt.scatter(X_pca[clusters == cluster, 0], X_pca[clusters == cluster, 1],
                label=f'Cluster {cluster}' if cluster != -1 else 'Noise',
                color=colors[cluster] if cluster != -1 else 'black', alpha=0.7)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("DBSCAN Clustering on Iris Dataset")
plt.legend()
plt.grid()
plt.show()
```

Output:



Learning Outcomes:

EXPERIMENT 9

Aim: Program to demonstrate K-medoid clustering algorithm.

Theory:

K-Medoid is a clustering algorithm that is similar to K-Means, but instead of using centroids (mean of points), it selects actual data points (medoids) as cluster centers. This makes K-Medoid more robust to outliers compared to K-Means.

Key Characteristics of K-Medoid:

- Uses medoids (actual points) instead of centroids.
- Less sensitive to outliers compared to K-Means.
- Finds clusters by minimizing the sum of distances between points and their medoid.

Algorithm:

1. Initialize k medoids randomly from the dataset.
2. Assign each data point to the nearest medoid.
3. Swap medoids with other points in the cluster to find a better set of medoids (i.e., those that minimize total distance).
4. Repeat steps 2-3 until medoids do not change or convergence is reached.

Dataset Description:

The Iris dataset is a well-known dataset used for classification tasks. It contains 150 samples of iris flowers from three species:

- Setosa
- Versicolor
- Virginica

Each sample has four features:

1. Sepal Length (cm)
2. Sepal Width (cm)
3. Petal Length (cm)
4. Petal Width (cm)

Target Variable:

- Species (Categorical: Setosa, Versicolor, Virginica)

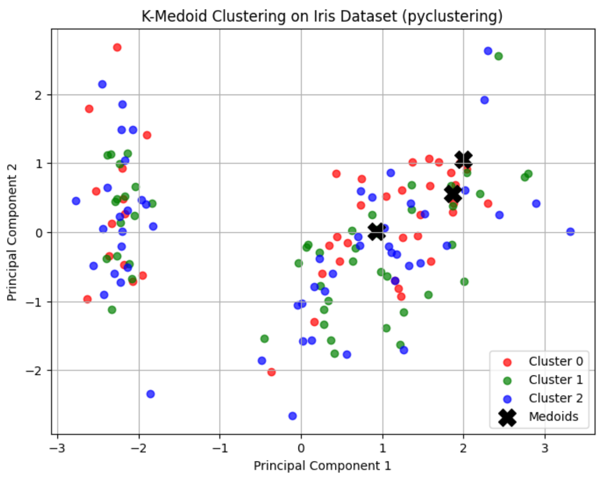
Dataset Source:

The dataset is originally from Fisher's Iris dataset and is widely used for pattern recognition in Machine Learning.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from pyclustering.cluster.kmedoids import kmedoids
from scipy.spatial.distance import cdist

iris = datasets.load_iris()
X = iris.data

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

np.random.seed(42)
initial_medoids = np.random.choice(len(X_scaled), 3, replace=False)

kmedoids_instance = kmedoids(X_scaled, initial_medoids, data_type='distance_matrix')
kmedoids_instance.process()
clusters = kmedoids_instance.get_clusters()
medoid_indices = kmedoids_instance.get_medoids()

cluster_labels = np.zeros(len(X_scaled))
for i, cluster in enumerate(clusters):
    for index in cluster:
        cluster_labels[index] = i

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
colors = ['red', 'green', 'blue']
for i in range(3):
    plt.scatter(X_pca[np.array(clusters[i]), 0], X_pca[np.array(clusters[i]), 1],
                label=f'Cluster {i}', color=colors[i], alpha=0.7)

plt.scatter(X_pca[medoid_indices, 0], X_pca[medoid_indices, 1],
            color='black', marker='X', s=200, label='Medoids')

plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("K-Medoid Clustering on Iris Dataset (pyclustering)")
plt.legend()
plt.grid()
plt.show()
```

Output:



Learning Outcomes:

EXPERIMENT 10

Aim: Program to demonstrate K-Means Clustering Algorithm on handwritten dataset.

Theory:

Clustering is an unsupervised learning technique that groups similar data points together based on certain features. Unlike supervised learning, where data is labeled, clustering algorithms find intrinsic patterns in the data without predefined categories.
One of the most widely used clustering techniques is the K-Means Algorithm.
Advantages of K-Means:

- **Simple & Efficient –** Works well on large datasets.
- **Scalable –** Can handle high-dimensional data.
- **Faster convergence** compared to hierarchical clustering.

Disadvantages of K-Means:

- Requires the number of clusters (K) to be specified beforehand.
- Sensitive to initial centroid selection – Poor initialization can lead to bad clustering.
- Works best with spherical clusters – Struggles with complex cluster shapes.

Algorithm:

1. Select the number of clusters K.
2. Randomly initialize K cluster centroids.
3. Assign each data point to the nearest centroid (based on Euclidean distance).
4. Compute the new centroid for each cluster as the mean of all assigned points.
5. Repeat steps 3 & 4 until centroids no longer change significantly (convergence).

Dataset Description:

We use the Digits Dataset from sklearn.datasets. This dataset contains images of handwritten digits (0-9), each represented as a 64-dimensional vector (8×8 pixel values).

Dataset Details:

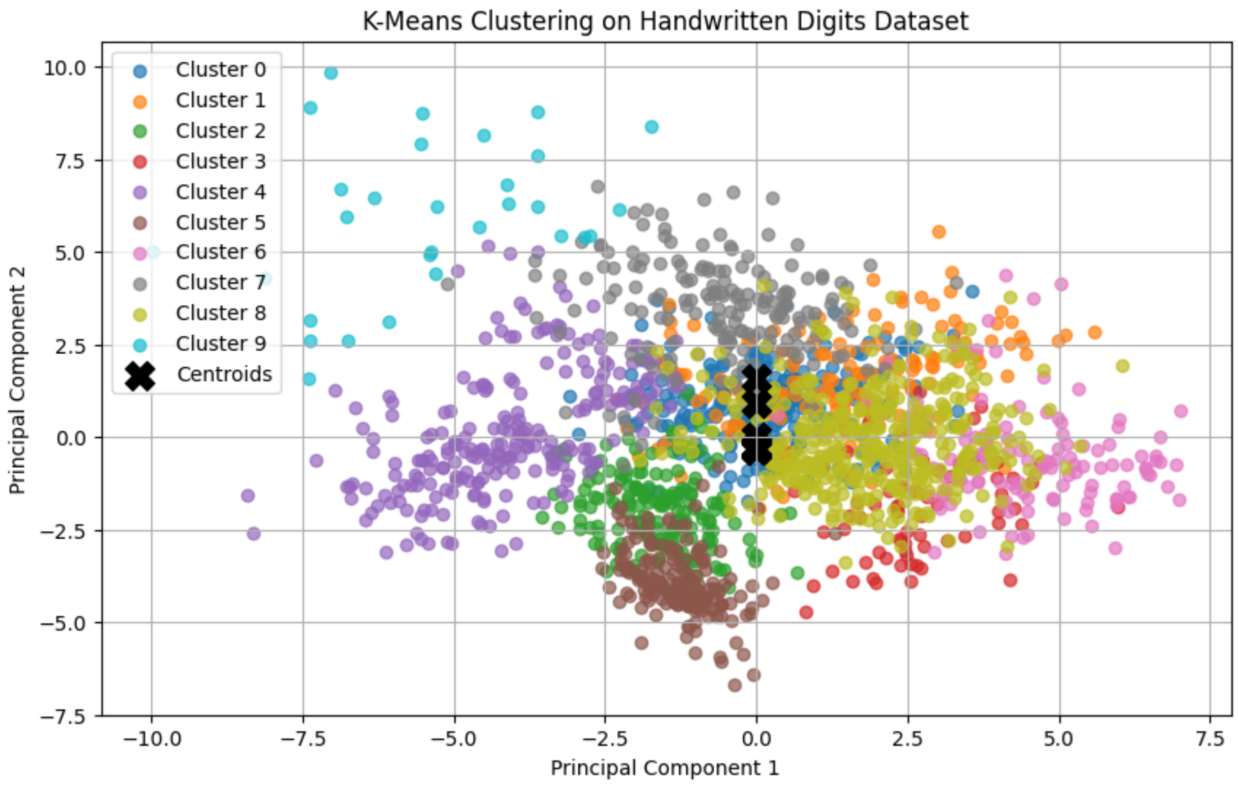
Features: 64 (Each 8×8 image is flattened into a 1D vector).
Samples: 1797 handwritten digit images.
Labels: 10 classes (digits 0-9).
Data Type: NumPy array.

Each image is grayscale, and pixel intensity values range from 0 (black) to 16 (white).

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

digits = datasets.load_digits()
X = digits.data

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

k = 10
kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
labels = kmeans.fit_predict(X_scaled)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(10, 6))
for i in range(k):
    plt.scatter(X_pca[labels == i, 0], X_pca[labels == i, 1], label=f'Cluster {i}', alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            color='black', marker='X', s=200, label='Centroids')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("K-Means Clustering on Handwritten Digits Dataset")
plt.legend()
plt.grid()
plt.show()
```

Output:



Learning Outcomes: