***Άσκηση 2.2***

*Ερώτημα Α*

A screenshot of a computer program

Description automatically generated

Βοηθητικός κώδικας που χρησιμοποιήθηκε:

import numpy as np

# Data

points = np.array([[2, 3], [3, 2], [1, 2], [4, 5],

                   [5, 4], [3, 4], [6, 4], [6, 5]])

# Initialization of centroids

centroids = np.array([[3, 3], [4, 4]])

def euclidean\_distance(point, centroid):

    """Calculation of the Euclidean distance between a point and a centroid"""

    return np.sqrt(np.sum((point - centroid) \*\* 2))

def assign\_points\_to\_clusters(points, centroids):

    """Assign each point to the nearest centroid"""

    clusters = []

    for point in points:

        distances = [euclidean\_distance(point, centroid) for centroid in centroids]

        cluster = np.argmin(distances)  # Returns the index of the nearest centroid

        clusters.append(cluster)

    return np.array(clusters)

def update\_centroids(points, clusters, k):

    """Calculation of new centroids as the mean of the points in each cluster"""

    new\_centroids = []

    for i in range(k):

        cluster\_points = points[clusters == i]

        if len(cluster\_points) > 0:  # If there are points in the cluster

            new\_centroid = np.mean(cluster\_points, axis=0)

        else:  # If there are no points, retain the old centroid

            new\_centroid = centroids[i]

        new\_centroids.append(new\_centroid)

    return np.array(new\_centroids)

# Execution of the algorithm

iteration = 0

k = len(centroids)

while True:

    print(f"\n=== Iteration {iteration} ===")

    # Calculation of distances and assignment

    clusters = assign\_points\_to\_clusters(points, centroids)

    for i, point in enumerate(points):

        distances = [euclidean\_distance(point, centroid) for centroid in centroids]

        print(f"Point: ({point[0]}, {point[1]}), Distances: {distances}, Assigned to Cluster: {clusters[i] + 1}")

    # Calculation of new centroids

    new\_centroids = update\_centroids(points, clusters, k)

    print(f"New Centroids: {new\_centroids}")

    # Convergence check

    if np.allclose(centroids, new\_centroids):

        print("\nConvergence reached.")

        break

    centroids = new\_centroids

    iteration += 1

# Final output

print("\n=== Final Output ===")

print("Final Centroids:")

for i, centroid in enumerate(centroids, 1):

    print(f"  Cluster {i}: {centroid}")

print("\nFinal Cluster Assignments:")

for i, cluster in enumerate(clusters, 1):

    print(f"  Point {i}: Assigned to Cluster {cluster + 1}")

*Ερώτημα Β, Γ*

Ακολουθεί τα αποτελέσματα του αλγορίθμου k-means με όλα τα χαρακτηριστικά και με μόνο δύο, καθώς και ο αντίστοιχος βοηθητικός κώδικας που χρησιμοποιήθηκε.

A screenshot of a computer

Description automatically generated

import pandas as pd

import numpy as np

from sklearn.metrics import confusion\_matrix, accuracy\_score

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

from collections import Counter

# Load the dataset

data = pd.read\_csv('iris.csv')

data = data.drop(columns=['Id'])

# Encode Species to numerical values for comparison

label = LabelEncoder()

data['SpeciesEncoded'] = label.fit\_transform(data['Species'])

# K-means algorithm implementation

def k\_means(X, k, epsilon=1e-5, max\_iterations=100):

    # Random initialization of centroids

    np.random.seed(42)

    centroids = X[np.random.choice(X.shape[0], k, replace=False)]

    for iteration in range(max\_iterations):

        # Assign points to the nearest centroid

        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)

        clusters = np.argmin(distances, axis=1)

        # Update centroids

        new\_centroids = np.array(

            [X[clusters == i].mean(axis=0) for i in range(k)])

        # Convergence criterion

        if np.linalg.norm(new\_centroids - centroids) < epsilon:

            print("\nConvergence reached!")

            break

        centroids = new\_centroids

    return clusters, centroids

# Function to remap clusters to match true labels

def remap\_clusters(true\_labels, clusters):

    # Create a mapping from clusters to true labels based on majority vote

    mapping = {}

    for cluster\_id in np.unique(clusters):

        # Find true labels corresponding to points in this cluster

        true\_labels\_in\_cluster = true\_labels[clusters == cluster\_id]

        # Get the most common true label for this cluster

        most\_common\_label = Counter(

            true\_labels\_in\_cluster).most\_common(1)[0][0]

        mapping[cluster\_id] = most\_common\_label

    # Remap the clusters

    remapped\_clusters = np.array([mapping[cluster] for cluster in clusters])

    return remapped\_clusters

# (b) Apply k-means with all features

X\_full = data[['SepalLengthCm', 'SepalWidthCm',

               'PetalLengthCm', 'PetalWidthCm']].values

clusters\_full, centroids\_full = k\_means(X\_full, k=3)

# Map cluster labels to the true labels for comparison

true\_labels = data['SpeciesEncoded'].values

clusters\_full = remap\_clusters(true\_labels, clusters\_full)

conf\_matrix\_full = confusion\_matrix(true\_labels, clusters\_full)

accuracy\_full = accuracy\_score(true\_labels, clusters\_full)

print("\n=== (b) Full Feature Set ===")

print("Confusion Matrix:")

print(conf\_matrix\_full)

print(f"\nAccuracy (Full Features): {accuracy\_full \* 100:.2f}%")

# (c) Apply k-means with only two features

X\_two = data[['PetalLengthCm', 'PetalWidthCm']].values

clusters\_two, centroids\_two = k\_means(X\_two, k=3)

clusters\_two = remap\_clusters(true\_labels, clusters\_two)

conf\_matrix\_two = confusion\_matrix(true\_labels, clusters\_two)

accuracy\_two = accuracy\_score(true\_labels, clusters\_two)

print("\n=== (c) Two Features ===")

print("Confusion Matrix:")

print(conf\_matrix\_two)

print(f"Accuracy: {accuracy\_two \* 100:.2f}%")

# (d) Compare results

print("\n=== (d) Comparison ===")

print(f"Accuracy with Full Features: {accuracy\_full \* 100:.2f}%")

print(f"Accuracy with Two Features: {accuracy\_two \* 100:.2f}%")

# Plotting for (c)

plt.figure(figsize=(8, 6))

for i in range(3):

    cluster\_points = X\_two[clusters\_two == i]

    plt.scatter(cluster\_points[:, 0],

                cluster\_points[:, 1], label=f'Cluster {i+1}')

plt.scatter(centroids\_two[:, 0], centroids\_two[:, 1],

            color='black', marker='x', s=100, label='Centroids')

plt.xlabel('Petal Length (cm)')

plt.ylabel('Petal Width (cm)')

plt.title('(c) K-means Clustering on Two Features')

plt.legend()

plt.show()

Επίσης, ακολουθεί γραφική απεικόνιση του ζητούμενου.

A graph of clustering

Description automatically generated

*Ερώτημα Δ*

Χρησιμοποιώντας και τα 4 χαρακτηριστικά, το success rate ήταν 89.33%, ενώ με μόνο το μήκος και το πλάτος των πετάλων αυξήθηκε στο 94.67%. Αυτό δείχνει ότι τα χαρακτηριστικά των πετάλων είναι πιο διακριτικά για τις κλάσεις, ενώ η προσθήκη των χαρακτηριστικών των σεπάλων εισάγει θόρυβο και μειώνει την ακρίβεια.

Συνεπώς, η χρήση όλων των χαρακτηριστικών δεν οδηγεί πάντα σε καλύτερα αποτελέσματα, καθώς μπορεί να περιλαμβάνει περιττή ή παραπλανητική πληροφορία που επηρεάζει αρνητικά την ταξινόμηση.