import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.mixture import GaussianMixture

from sklearn.preprocessing import StandardScaler

from mpl\_toolkits.mplot3d import Axes3D

# Load the dataset from local machine

file\_path = 'news\_data.tsv' # Update this path to your local file

df = pd.read\_csv(file\_path, sep='\t')

# Print column names and a few rows of the dataframe

print("Column Names:", df.columns)

print(df.head())

# Prepare the data

# Adjust column names based on the actual content

# For example, assuming the actual feature columns and a label column named 'label'

X = df.drop(columns=['label']).values

y = df['label'].values

# If necessary, convert labels to "real" and "fake"

y = np.where(y == 0, "real", "fake")

# Define the objective function

def objective(params):

n\_components = int(params[0])

reg\_covar = params[1]

max\_iter = int(params[2])

tol = params[3]

gmm = GaussianMixture(n\_components=n\_components, covariance\_type='full', reg\_covar=reg\_covar, max\_iter=max\_iter, tol=tol, random\_state=42)

gmm.fit(X\_scaled)

labels = gmm.predict(X\_scaled)

# Map cluster labels to "real" and "fake"

labels = np.where(labels == 0, "real", "fake")

accuracy = np.mean(labels == y)

return -accuracy # Minimize the negative accuracy

# Define the Group Counseling Optimization algorithm

class GroupCounselingOptimization:

def \_\_init\_\_(self, func, bounds, num\_agents, max\_iter):

self.func = func

self.bounds = bounds

self.num\_agents = num\_agents

self.max\_iter = max\_iter

self.dim = len(bounds)

self.population = np.random.rand(num\_agents, self.dim)

for i in range(self.dim):

self.population[:, i] = self.population[:, i] \* (bounds[i][1] - bounds[i][0]) + bounds[i][0]

self.fitness = np.apply\_along\_axis(self.func, 1, self.population)

def optimize(self):

for iteration in range(self.max\_iter):

for i in range(self.num\_agents):

M = np.mean(self.population, axis=0)

r = np.random.uniform(-1, 1, self.dim)

x\_new = self.population[i] + r \* (self.population[i] - M)

for j in range(self.dim):

if x\_new[j] < self.bounds[j][0]:

x\_new[j] = self.bounds[j][0]

elif x\_new[j] > self.bounds[j][1]:

x\_new[j] = self.bounds[j][1]

new\_fitness = self.func(x\_new)

if new\_fitness < self.fitness[i]:

self.population[i] = x\_new

self.fitness[i] = new\_fitness

best\_index = np.argmin(self.fitness)

return self.population[best\_index], self.fitness[best\_index]

# Scale the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Set GCO parameters

bounds = [(2, 10), (1e-6, 1e-2), (100, 500), (1e-4, 1e-2)]

num\_agents = 10

max\_iter = 50

# Run GCO algorithm

gco = GroupCounselingOptimization(objective, bounds, num\_agents, max\_iter)

best\_params, best\_fitness = gco.optimize()

# Train the final GMM with optimized parameters

n\_components = int(best\_params[0])

reg\_covar = best\_params[1]

max\_iter = int(best\_params[2])

tol = best\_params[3]

final\_gmm = GaussianMixture(n\_components=n\_components, covariance\_type='full', reg\_covar=reg\_covar, max\_iter=max\_iter, tol=tol, random\_state=42)

final\_gmm.fit(X\_scaled)

# Print optimized parameters obtained from GCO and utilized in GMM

print("Optimized Parameters from GCO utilized in GMM:")

print(f"Number of components: {n\_components}")

print(f"Regularization on covariance: {reg\_covar}")

print(f"Maximum iterations: {max\_iter}")

print(f"Tolerance: {tol}")

# Print GMM final parameters

print("\nFinal GMM Parameters:")

print(f"Weights: {final\_gmm.weights\_}")

print(f"Means: \n{final\_gmm.means\_}")

print(f"Covariances: \n{final\_gmm.covariances\_}")

# Plotting

def plot\_gmm(gmm, X, y):

plt.figure(figsize=(12, 6))

# 2D projection of the data and GMM results

if X.shape[1] == 2:

plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap='viridis')

plt.title("GMM Clustering Results (2D)")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

elif X.shape[1] == 3:

fig = plt.figure(figsize=(12, 10))

ax = fig.add\_subplot(111, projection='3d')

sc = ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, s=40, cmap='viridis')

plt.title("GMM Clustering Results (3D)")

ax.set\_xlabel("Feature 1")

ax.set\_ylabel("Feature 2")

ax.set\_zlabel("Feature 3")

plt.colorbar(sc)

else:

print("Data is neither 2D nor 3D. Unable to plot.")

plt.show()

# Predict the cluster labels for the data

labels = final\_gmm.predict(X\_scaled)

labels = np.where(labels == 0, "real", "fake")

# Plot the results

plot\_gmm(final\_gmm, X\_scaled, labels)