CS613 Project -Francis Kurian

Self Organizing Map (SOM)

The project impliments Self-Organizing Map (SOM) algorithm to cluster random 2D data points and evaluate the performance under different configurations.

Data Generation:

Generates a dataset of 100 random 2D data points using NumPy.Normalizes the generated data to the range [0, 1].

SOM Configuration:

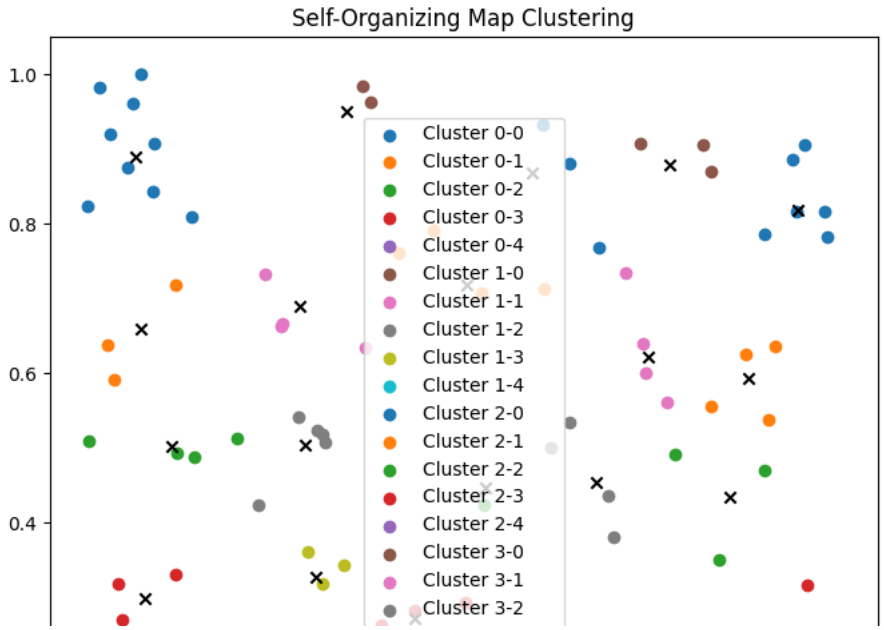
Sets up a SOM grid with a size of 5x5. Defines three different configurations with varying learning rates, neighborhood sizes (sigma), and numbers of training iterations.

SOM Training and Evaluation:

Iterates over each configuration and trains a SOM using the MiniSom library. Calculates the quantization error, which measures the average distance between data points and their corresponding winning neurons in the SOM. Stores the results, including configuration details and quantization error, in a results table.

```
pip install minisom
```

```
import numpy as np
from minisom import MiniSom
import matplotlib.pyplot as plt

np.random.seed(42)
data = np.random.rand(100, 2)

data = (data - data.min(axis=0)) / (data.max(axis=0) - data.min(axis=0))

som_size = (5, 5)
input_size = 2
learning_rate = 0.5
sigma = 1.0

som = MiniSom(som_size[0], som_size[1], input_size, sigma=sigma, learning_rate=learning_rate)

som.train_random(data, 1000)

mapped_data = som.win_map(data)

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

for i in range(som_size[0]):
    for j in range(som_size[1]):
        cluster_points = np.array(mapped_data[(i, j)])  # Convert the list to a NumPy array
        if cluster_points.shape[0] > 0:  # Check if there are any points in the cluster
            plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster {i}-{j}')

plt.scatter(som.get_weights()[:, :, 0].flatten(), som.get_weights()[:, :, 1].flatten(), marker='x', color='k', label='SOM ce

plt.title('Self-Organizing Map Clustering')
plt.legend()
plt.show()

```




```python
import numpy as np
from minisom import MiniSom
from scipy.spatial.distance import euclidean

np.random.seed(42)
data = np.random.rand(100, 2)

data = (data - data.min(axis=0)) / (data.max(axis=0) - data.min(axis=0))

som_size = (5, 5)

configurations = [
    {'learning_rate': 0.5, 'sigma': 1.0, 'num_iterations': 1000},
    {'learning_rate': 0.3, 'sigma': 0.8, 'num_iterations': 1500},
    {'learning_rate': 0.7, 'sigma': 1.5, 'num_iterations': 800},
]

results_table = []

for config in configurations:

    som = MiniSom(som_size[0], som_size[1], data.shape[1], sigma=config['sigma'], learning_rate=config['learning_rate'])


    som.train_random(data, config['num_iterations'])


    quantization_error = 0
    for i in range(data.shape[0]):
        x = data[i]
        winner = som.winner(x)
        quantization_error += euclidean(x, som.get_weights()[winner])

    quantization_error /= data.shape[0]


    results_table.append({'config': config, 'quantization_error': quantization_error})


print("{:<25} {:<15}".format("Configuration", "Quantization Error"))
print("="*45)
for result in results_table:
    config_str = f"LR={result['config']['learning_rate']}, Sigma={result['config']['sigma']}, Iter={result['config']['num_ite
    print("{:<25} {:<15.4f}".format(config_str, result['quantization_error']))

```

```
Configuration             Quantization Error
=============================================
LR=0.5, Sigma=1.0, Iter=1000 0.0569
LR=0.3, Sigma=0.8, Iter=1500 0.0654
LR=0.7, Sigma=1.5, Iter=800 0.0673
```

Summary

The output summarises the quantization error for each combination of learning rate, sigma, and number of iterations, allowing for a comparison of different configurations. Lower the quantization error, the better the SOM model is.

The quantization error serves as a metric to assess how well the SOM is able to represent the input data for each configuration. The project provides insights into the impact of different parameters on the clustering performance of the SOM algorithm.

```
Configuration             Quantization Error
=============================================
LR=0.5, Sigma=1.0, Iter=1000 0.0569
LR=0.3, Sigma=0.8, Iter=1500 0.0654
LR=0.7, Sigma=1.5, Iter=800 0.0673
```