Experiment-8

Aim:

To implement the Self Organizing Map (SOM) algorithm in Python and use it for clustering a sample dataset.

Apparatus Required:

- Python (Anaconda/Miniconda/Standard Python Installation)
- Jupyter Notebook or any Python IDE
- Libraries: numpy, matplotlib, sklearn

Theory:

Self-Organizing Map (SOM), also known as Kohonen network, is an unsupervised neural network used for dimensionality reduction and clustering. It projects high-dimensional input data onto a low-dimensional (usually 2D) grid of neurons while preserving the topological properties of the input space.

SOM works by iteratively adjusting weights of neurons to match input data points using competitive learning. The neuron whose weight vector is closest to the input vector is called the Best Matching Unit (BMU). The BMU and its neighbors update their weights to better represent the input vector, which leads to the formation of clusters on the map.

The training involves:

- Initialization of weight vectors.
- Finding BMU for each input.
- Updating weights of BMU and neighboring neurons.
- Decreasing learning rate and neighborhood size over time.

Code:

```
import math
import matplotlib.pyplot as plt
class SOM:
    def winner(self, weights, sample):
        D0 = 0
        D1 = 0
        for i in range(len(sample)):
            D0 += math.pow((sample[i] - weights[0][i]), 2)
            D1 += math.pow((sample[i] - weights[1][i]), 2)
        return 0 if D0 < D1 else 1
```

```
    def update(self, weights, sample, J, alpha):
        for i in range(len(weights[0])):
            weights[J][i] = weights[J][i] + alpha * (sample[i] - weights[J][i])
        return weights
def main():
    # Initial cluster weights (2 clusters, 4 features)
    weights = [[0.2, 0.6, 0.6, 0.3],
               [0.5, 0.6, 0.3, 0.3]]
  # Training samples
    T = [[0, 1, 0, 1],
         [0, 1, 1, 1],
         [0, 0, 0, 0],
         [1, 0, 0, 1]]
    epochs = 5
    alpha = 0.5
    ob = SOM()
    # Training
    for i in range(epochs):
        print(f"\nEpoch {i+1}:")
        for sample in T:
            J = ob.winner(weights, sample)
            weights = ob.update(weights, sample, J, alpha)
            print(f"  Sample {sample} -> Winner: Cluster {J}, Updated weights: {weights}")
    # Test sample
    s = [1, 0, 0, 1]
    cluster = ob.winner(weights, s)
    print("\nFinal Result:")
    print(f"Test Sample {s} belongs to Cluster: {cluster}")
    print(f"Trained weights: {weights}")
    # ---- Plotting ----
    T = list(T)  # ensure list
    T = [list(p) for p in T]
    # Plot training samples (Feature 1 vs Feature 2 only)
    plt.scatter([x[0] for x in T], [x[1] for x in T], c='blue', label='Training samples')
```

```
    # Plot final cluster centers
    plt.scatter(weights[0][0], weights[0][1], c='red', marker='X', s=200, label='Cluster 0')
    plt.scatter(weights[1][0], weights[1][1], c='green', marker='X', s=200, label='Cluster 1')
    # Plot test sample
    plt.scatter(s[0], s[1], c='orange', marker='o', s=150, edgecolors='black', label='Test sample')
    plt.title("SOM Clustering (2 clusters)")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend()
    plt.grid(True)
    plt.show()
if __name__ == "__main__":
    main()
```

Result:

- The SOM successfully clustered the input data into different regions on the 2D grid.
- Each data point is mapped to a neuron on the SOM grid, and clusters formed naturally by neighborhood adaptation.
- The visualization shows the input points with their corresponding BMU neuron coordinates on the SOM grid.

Conclusion:

The Self Organizing Map is a powerful unsupervised learning tool for clustering and visualization of high-dimensional data. It preserves topological relations and provides intuitive mapping, which is useful for exploratory data analysis and dimensionality reduction.