Project5

1.Team member

- **m5268101 Liu Jiahe**
- **m5251140 Ken Sato**
- **m5271051 Keisuke Utsumi**

2.Team Project V

- Using SOFM(Self-Organization feature map) algorithm to cluster the Iris dataset (http://www.ics.uci.edu/~mlearn/MLRepository.html).

3.Algorithm

The SOFM (Self-Organizing Feature Map) algorithm can reduce the dimensionality of data while preserving the topological structure of the data in space. In this algorithm, the neurons are distributed in a two-dimensional network, each neuron has six neighborhoods, forming a hexagonal grid. The number of weights of each neuron is the same as the dimension of the input data vector. The steps of the algorithm are as follows:

1. Initialization:
    - Initialize the connection weight matrix w, where $w[n][i]$ represents the connection weight between the nth neuron and the ith component of the input vector.
2. Self-organizing mapping cycle：
   a. Select a training sample vector:
    - Randomly select a sample vector $x[p]$ from the training set.

   b. Find the most similar neuron:
    - Calculate the distance between the sample vector $x[p]$ and each neuron.
    - Find the neuron with the smallest distance to the sample vector, i.e., calculate the minimum Euclidean distance:
      $d(m_1, m_2) = \sum_{i=1}^{I}(w[m_1, m_2][i] - x[p][i])^2$, where m_1 and m_2 represent the position of the neuron on the two-dimensional grid.

   c. Update the weights of the neurons in the neighborhood:
    - Define the learning rate r and the neighborhood radius nc, which gradually decrease with the increase of iterations.

- For each neuron $w[m_1, m_2]$, calculate its position (x_1, x_2) in two-dimensional space. Adding 0.5 to the x-coordinate of even rows is to visually form a hexagonal network, but it does not affect the weight of the neuron.
- If the neuron (m_1, m_2) is in the neighborhood of the closest neuron (m_{10}, m_{20}) (distance less than or equal to the neighborhood radius nc):
    - Update the connection weights:
      $w[m_1, m_2][i] = w[m_1, m_2][i] + r \cdot (x[p][i] - w[m_1, m_2][i])$, where i represents the index of the input vector component.

3. Calibration
    - Assign the label of each input pattern to the most similar (nearest) neuron and update the label of the neuron.
4. Print results:
    - Print the final neuron label matrix to show the classification results of each neuron.

Iris dataset

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
....
....
```

4.Code

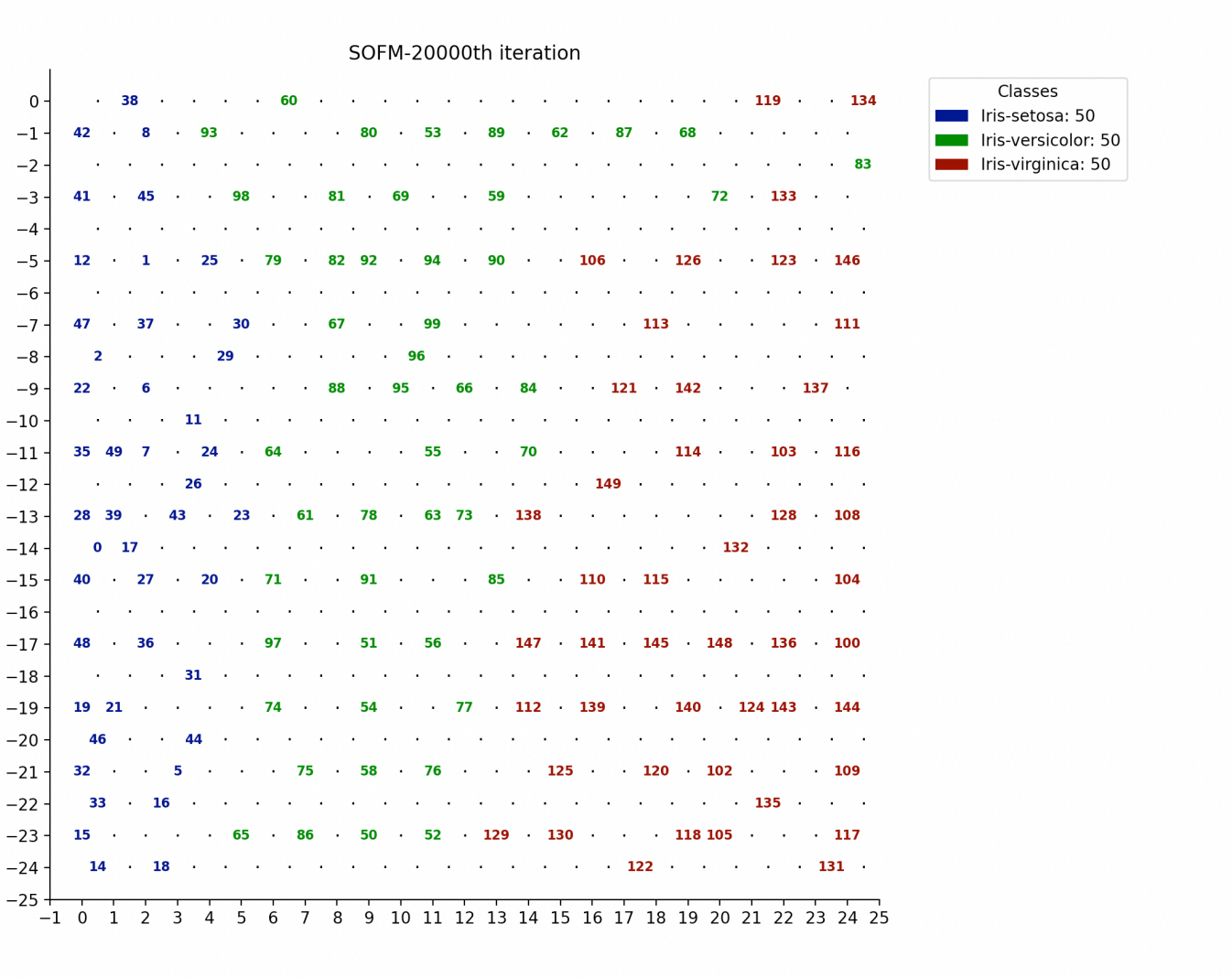
project4.py, use utils.py

```
from utils import *

def main():
    x, label0, label_dict, I, P, color_dict = InputPattern("./iris.data")
    # print(color_dict)
    use_classes_in_iris = True
    use_wta_cluster = True if not use_classes_in_iris else False
    if use_wta_cluster:
        classes = np.loadtxt('./clusters_3.txt', dtype=str)
        label_dict = {label0[i]: classes[i] for i in range(len(classes))}

        unique_classes = np.unique(classes)
        color_list = ['red', 'blue', 'green', 'purple', 'cyan', 'orange',
'magenta', 'lime', 'navy', 'darkred', 'darkgreen', 'gold', 'teal']
        random.shuffle(color_list)  # Randomly shuffle the color list
        color_dict = {unique_classes[i]: color_list[i] for i in
range(len(unique_classes))}
        # print(label_dict)
        print(color_dict)

    # print(label_dict)
    w = np.random.random((N, I))
    w = SOFM(3000, 0.5, 0.04, 10, 1, w, x, I, P)
    label = Calibration(w, x, label0, I, P)
    print("\n\nResult after the first 1,000 iterations:\n")
    # PrintResult(label)
    # PrintResult_figure(label, label_dict, 3000, color_dict)
    w = SOFM(17000, 0.04, 0.0, 1, 1, w, x, I, P)
    label = Calibration(w, x, label0, I, P)
    print("\n\nResult after 10,000 iterations:\n")
    PrintResult(label)
    PrintResult_figure(label, label_dict, 20000, color_dict)

if __name__ == "__main__":
    main()

```

5.Results

1.Using given classes

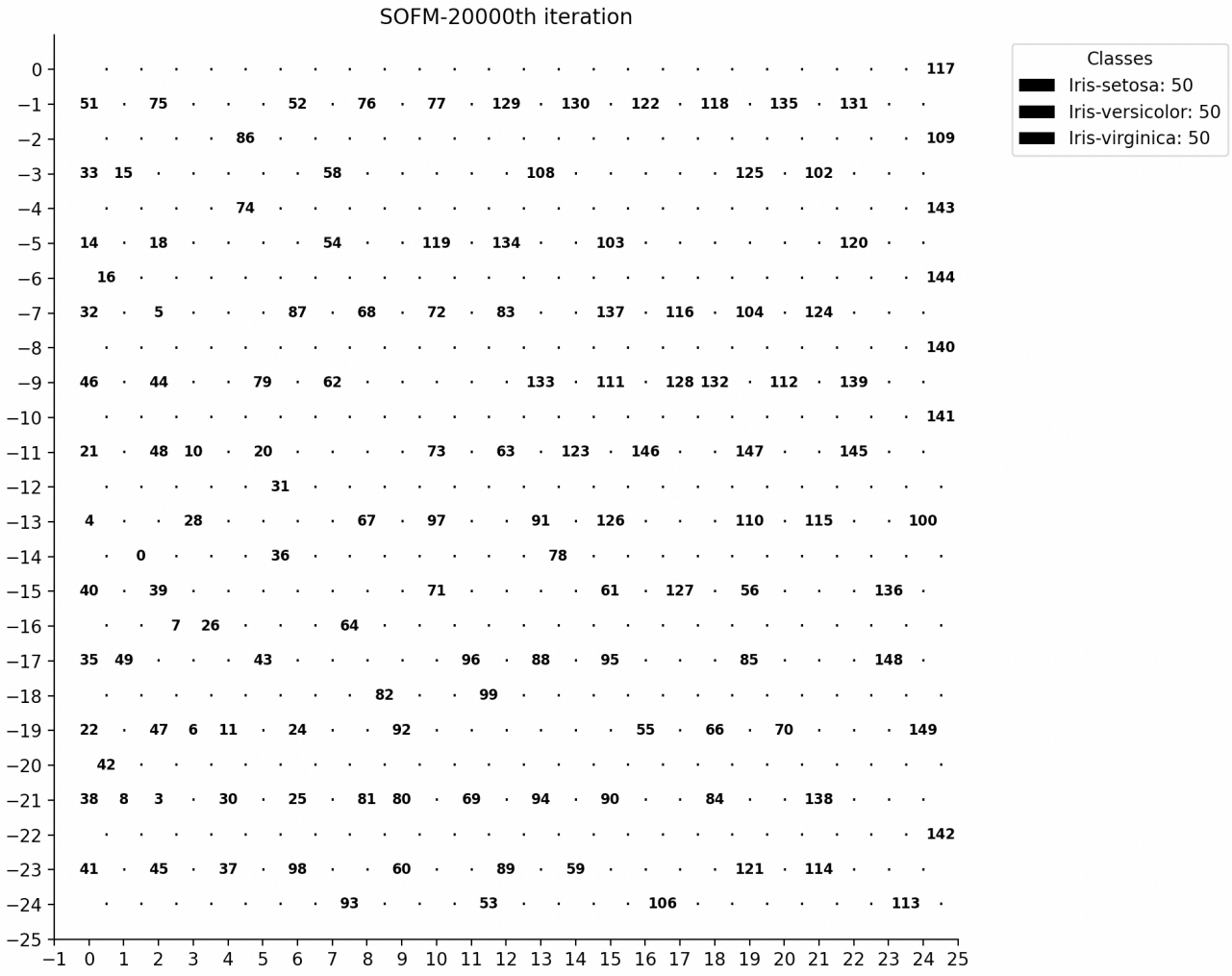
Assign labels ranging from 0 to 149 to the 150 sets of data in the Iris dataset. Classify each label based on the dataset's inherent category information. Then, apply the SOFM algorithm to map the high-dimensional data onto a two-dimensional graph. Color each category separately.



It can be observed that the data retains topological structure after dimensionality reduction. At the same time, the distribution of the data aligns with the given classifications in the dataset. Data belonging to the same class are accurately clustered together.

2.Don’t use given class

In the application scenarios of SOM (Self-Organizing Feature Map), the dataset itself usually does not have inherent classification information. After applying the algorithm, the result is as shown in the figure.



It can be observed that although SOFM compresses the data while preserving the original data's topological structure, it cannot explicitly divide the data into different clusters. This means that we cannot find explicit clusters from the results of the SOFM algorithm.

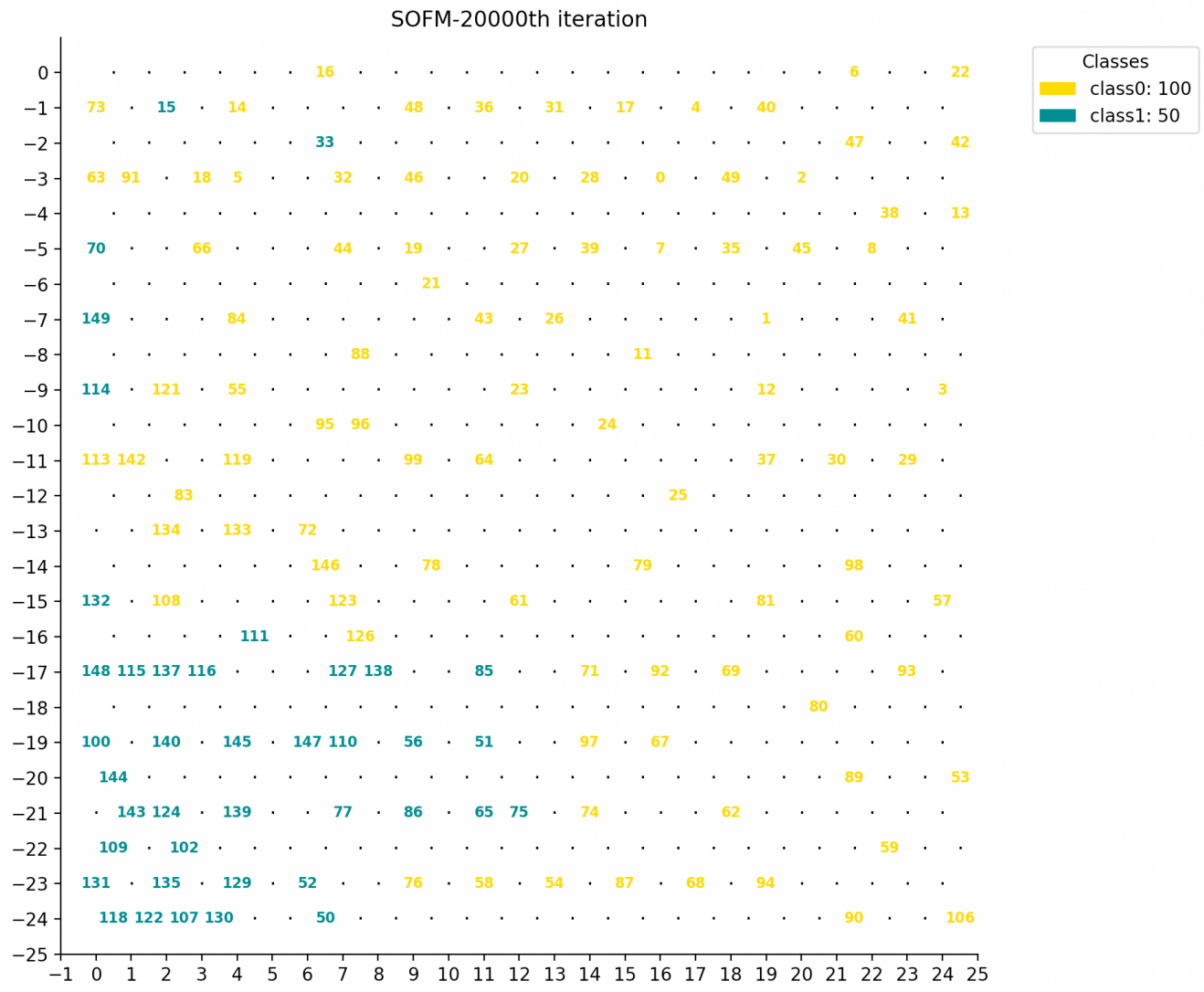
3.Cluster then SOFM

To address the above issue of preserving topological structure while achieving clustering after data compression, we can first use the WTA (Winner-Take-All) algorithm to cluster the dataset. Then, we can apply the SOFM algorithm.

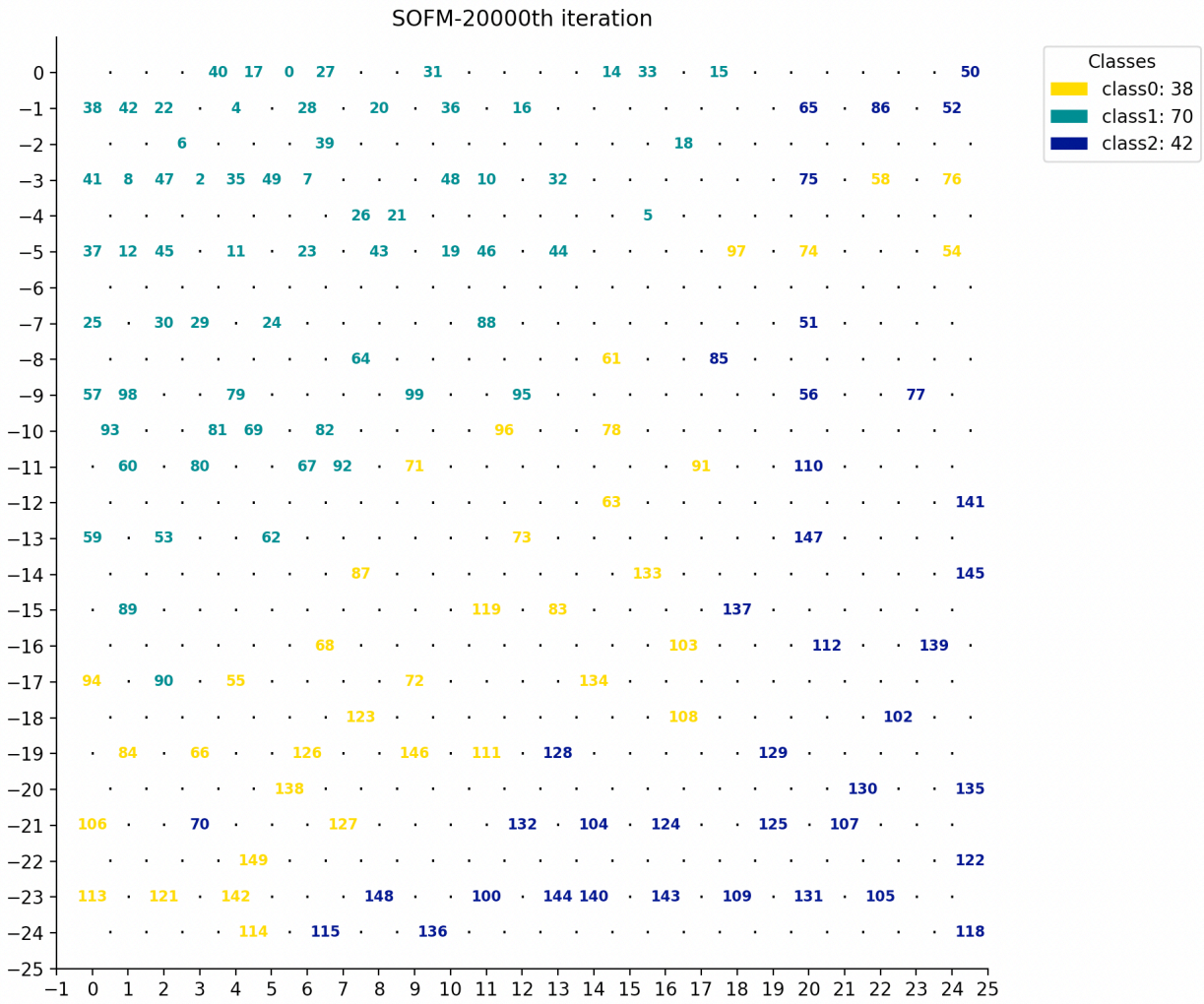
The script output_cluster.py allows customizing the number of clusters and outputs the clustering results to a text file.

The script project4.py reads the text file containing clustering information and implements the SOFM algorithm accordingly.

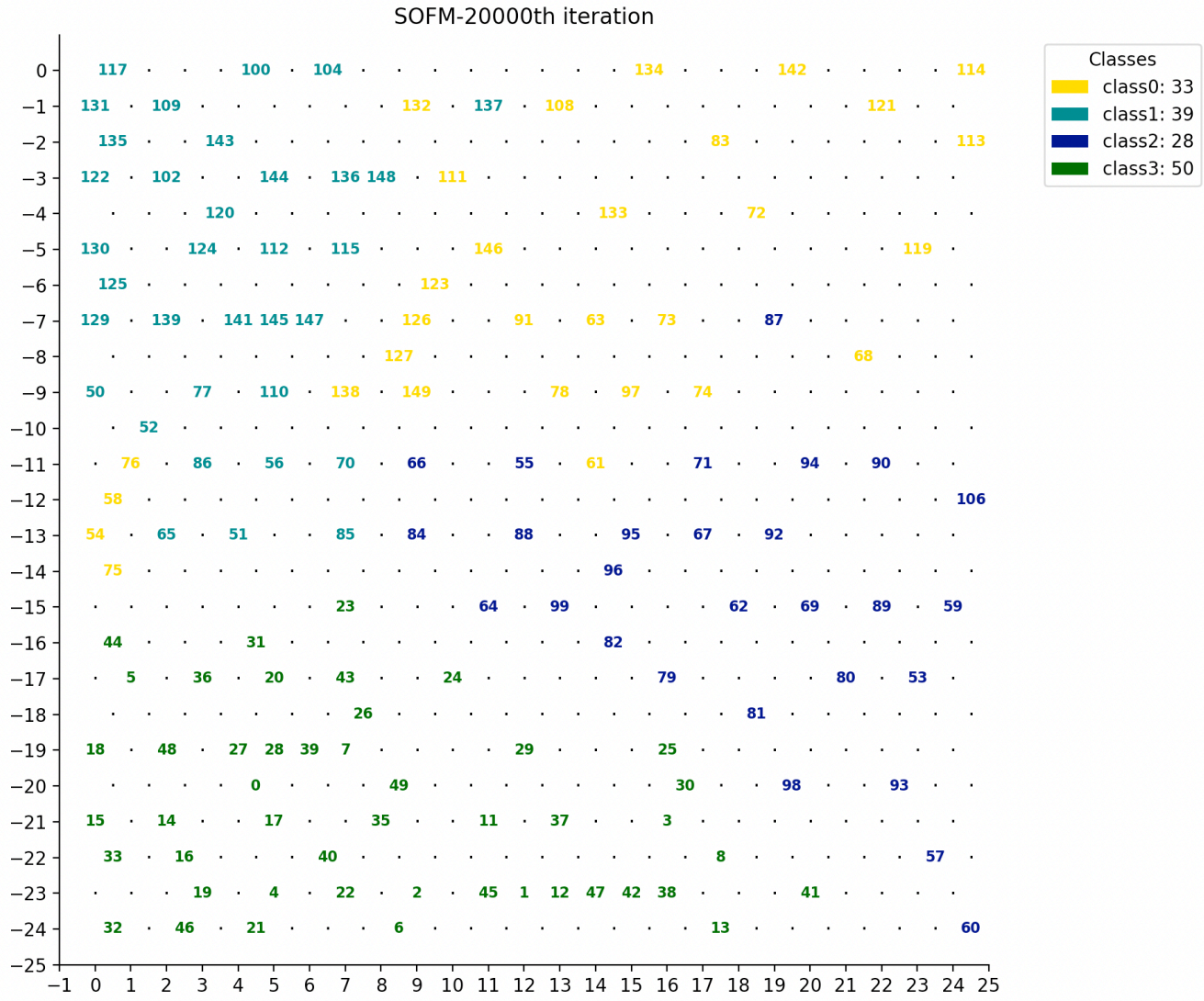
1.n_clusters = 2



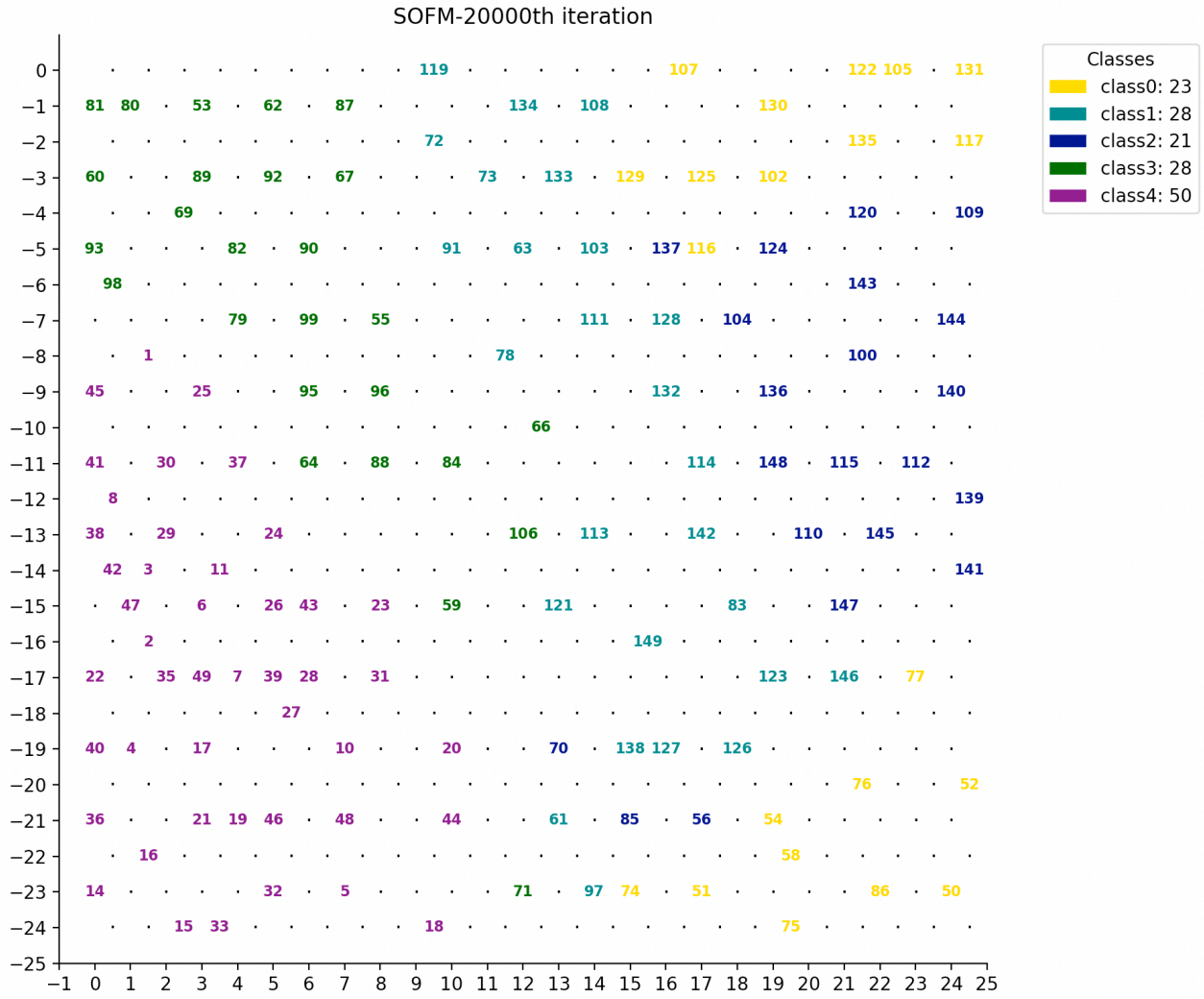
1.n_clusters = 3



3.n_clusters = 4



4.n_clusters = 5



It can be observed that the clustering results are relatively accurate, with data of the same type concentrated together. At the same time, the topological structure is also preserved.

6.Conclusion

The image above demonstrates that the combination of the WTA and SOFM algorithms yields excellent results. It allows for data compression while preserving the original data's topological structure and accurately clustering the data.

Furthermore, by observing the image, it can be noticed that not all 150 labels are fully displayed. This is because:

Each input pattern corresponds to the closest neuron, so it is possible that multiple input patterns correspond to the same neuron.

During the iteration process, if two or more input patterns have the same "closest" neuron, the labels of the later input patterns will overwrite the labels of the previous input patterns in the labeling process, resulting in the omission of some labels in the image display.

Below is the relevant code:

```
def Calibration(w, x, label0, I, P):
    label = ['.'] * N
    for p in range(P):
        d = np.linalg.norm(w - x[p], axis=1)  # calculate the Euclidean
distance
        n0 = np.argmin(d)  # find the index of the smallest distance
        label[n0] = str(label0[p])
    return label
```

It means that different input patterns (p) can correspond to the same neuron (n0).