# Practical 6A

**# A. Aim: Kohonen Self organizing map using python**

from minisom import MiniSom import numpy as np

import matplotlib.pyplot as plt

# # Generate sample data (you can replace this with your own dataset)

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

# # Define the SOM dimensions (grid size) and the input dimensionality

grid\_size = (10, 10)

input\_dim = 2

# # Initialize the SOM

som = MiniSom(grid\_size[0], grid\_size[1], input\_dim, sigma=1.0, learning\_rate=0.5)

# # Train the SOM with the data

som.train(data, 100) # 100 iterations, you can adjust this number

# # Create a map of the winning neurons

win\_map = som.win\_map(data) **# Visualize the SOM** plt.figure(figsize=(10, 10))

plt.pcolor(som.distance\_map().T, cmap='bone\_r') # Plot the distance map

# # Mark the winning neurons on the map

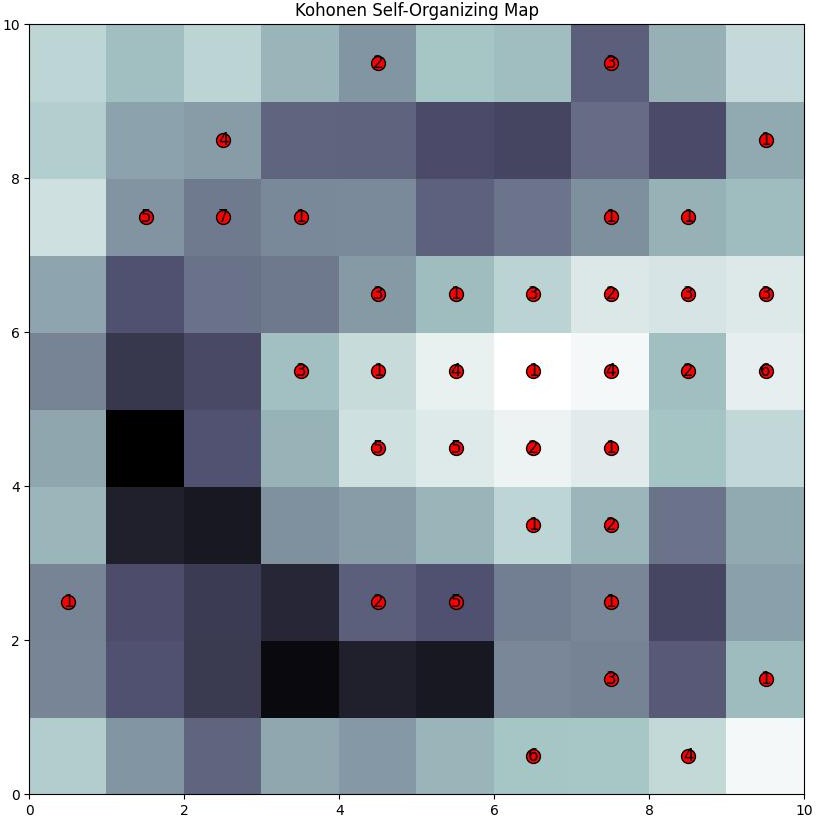
for position, values in win\_map.items():

x, y = position

plt.scatter(x + 0.5, y + 0.5, s=100, color='red', edgecolor='black')

plt.text(x + 0.5, y + 0.5, str(len(values)), fontsize=12, ha='center', va='center') plt.title('Kohonen Self-Organizing Map')

plt.show()



# Practical 6B

**#B. Aim: Adaptive resonance theory.**

import numpy as np class ART1:

def init (self, num\_input, vigilance\_parameter): self.num\_input = num\_input self.vigilance\_parameter = vigilance\_parameter self.weights = np.random.rand(num\_input) self.reset()

def reset(self): self.activation = 0

self.category = -1

def normalize(self, input\_pattern):

return input\_pattern / input\_pattern.sum() def match(self, input\_pattern):

return np.dot(self.weights, input\_pattern) def train(self, input\_pattern):

normalized\_pattern = self.normalize(input\_pattern) self.activation = self.match(normalized\_pattern)

if self.activation >= self.vigilance\_parameter: self.category = np.argmax(normalized\_pattern)

else:

self.category = -1

# # Example usage

if name == " main ": num\_input = 4

vigilance\_parameter = 0.5

art\_network = ART1(num\_input, vigilance\_parameter) **# Define training patterns and categories** training\_patterns = np.array([[0.1, 0.4, 0.2, 0.3],

[0.6, 0.3, 0.4, 0.1],

[0.2, 0.2, 0.3, 0.3]])

for i, pattern in enumerate(training\_patterns): art\_network.train(pattern)

print(f"Pattern {i + 1}: Category {art\_network.category}") art\_network.reset()

test\_pattern = np.array([0.5, 0.2, 0.3, 0.1]) art\_network.train(test\_pattern)

if art\_network.category != -1:

print(f"Test pattern belongs to Category {art\_network.category}") else:

print("Test pattern does not belong to any category")

**Output:**

Pattern 1: Category 1

Pattern 2: Category 0

Pattern 3: Category 2

Test pattern belongs to Category 0