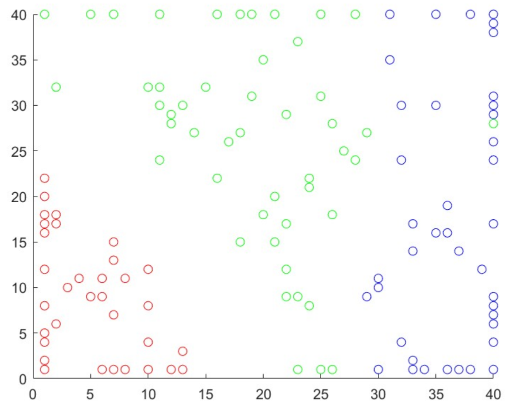

Figure 1: The classification results

1.Build the self-orangising map. It has 4 inputs and 40 * 40 outputs, and each input is connected with each output, so there are 4 * 40 * 40 weights.

2.Initialization. Initial learning rate is set to 0.1, with a decay rate of 0.01, and and the initial width of the neighbourhood function is set to 10, with a decay rate of 0.05. The learning rate and width follow the rule

$$\eta = \eta_0(-d_\eta \times epoch) \tag{1}$$

and

$$\sigma = \sigma_0(-d_\sigma \times epoch) \tag{2}$$

3.Standardisation and training. Standardise the dataset by dividing all the data with the maximal value of them, and feed all 150 data one at a time in random order. every time after feeding the data, find the winning neuron, i.e. the smallest angle between the input and the weight, and update all the 40 * 40 weights using the learning rule

$$\delta\omega_i = \eta h(i, i_0)(\mathbf{x} - \omega_i) \tag{3}$$

with the neighbourhood function

$$h(i, i_0) = exp(-\frac{1}{2\sigma^2}|r_i - r_{i0}|^2) \tag{4}$$

and repeat the step for 10 epochs.

4.Test. Feed all 150 data into the well-trained self-orangising map, and draw a scatter plot with data 1-50 red, 51-100 green, and 101-150 blue, as shown in Figure 1. In several runs, data 84 is always labeled incorrectly. It is because the third component (petal length) of data 84 is 0.6456, which is the biggest between label 1, close to the average petal length of label 2. Therefore, it is understandable that data 84 is marked with label 2.

```
irisData = readmatrix('iris-data.csv');
irisLabels = readmatrix('iris-labels.csv');

irisData = irisData / max(max(irisData));

dataInput = zeros(1,4);
dataOutput = zeros(40,40);
weightMatrix = zeros(40,40,4);

for i = 1:40
    for j = 1:40
        for k = 1:4
            weightMatrix(i,j,k) = rand();
        end
    end
end

initialLearningRate = 0.1;
learningRateDecayRate = 0.01;
initialwidth = 10;
widthDecayRate = 0.05;

for epoch = 1:10
    trainingOrder = randperm(150);
    learningRate = initialLearningRate * exp(-learningRateDecayRate * epoch);
    width = initialwidth * exp(-widthDecayRate * epoch);
    for p = 1:150
        dataInput = irisData(trainingOrder(p),:);
        winningNeuron = FindWinningNeuron(weightMatrix, dataInput);
        for i = 1:40
            for j = 1:40
                for k = 1:4
                    deltaWeight = FindDeltaWeight(learningRate, width,
winningNeuron,  [i j], dataInput, weightMatrix(i, j, k));
                    weightMatrix(i, j, k) = weightMatrix(i, j, k) + deltaWeight(k);
                end
            end
        end
    end
end

finalWinningNeuron = zeros(150,2);
for p = 1:150
    dataInput = irisData(p,:);
    finalWinningNeuron(p,:) = FindWinningNeuron(weightMatrix, dataInput);
end

scatter(finalWinningNeuron(1:50,1), finalWinningNeuron(1:50,2),"red");
hold on;
scatter(finalWinningNeuron(51:100,1), finalWinningNeuron(51:100,2),"green");
```

```
hold on;
scatter(finalWinningNeuron(101:150,1), finalWinningNeuron(101:150,2),"blue");
hold on;
```

```
function vectorAngle = CalculateVectorAngle(weight, dataInput)

vectorAngle = acos(dot(weight, dataInput)/(norm(weight) * norm(dataInput)));

end

function winningNeuron = FindWinningNeuron(weightMatrix,dataInput)

winningNeuronValue = transpose(squeeze(weightMatrix(1,1,:)));
winningNeuron = [1 1];

for i = 1:40
    for j = 1:40
        if CalculateVectorAngle(winningNeuronValue,dataInput) >
CalculateVectorAngle(transpose(squeeze(weightMatrix(i,j,:))),dataInput)
            winningNeuronValue = transpose(squeeze(weightMatrix(i,j,:)));
            winningNeuron = [i j];
        end
    end
end

end

function deltaWeight = FindDeltaWeight(learningRate, width, winningNeuron,
targetOutput, dataInput, weight)

neighbourhoodFunction =  exp ( (- norm(winningNeuron - targetOutput) ^ 2) / (2 *
width ^ 2));
deltaWeight = learningRate * neighbourhoodFunction * (dataInput - weight);

end
```