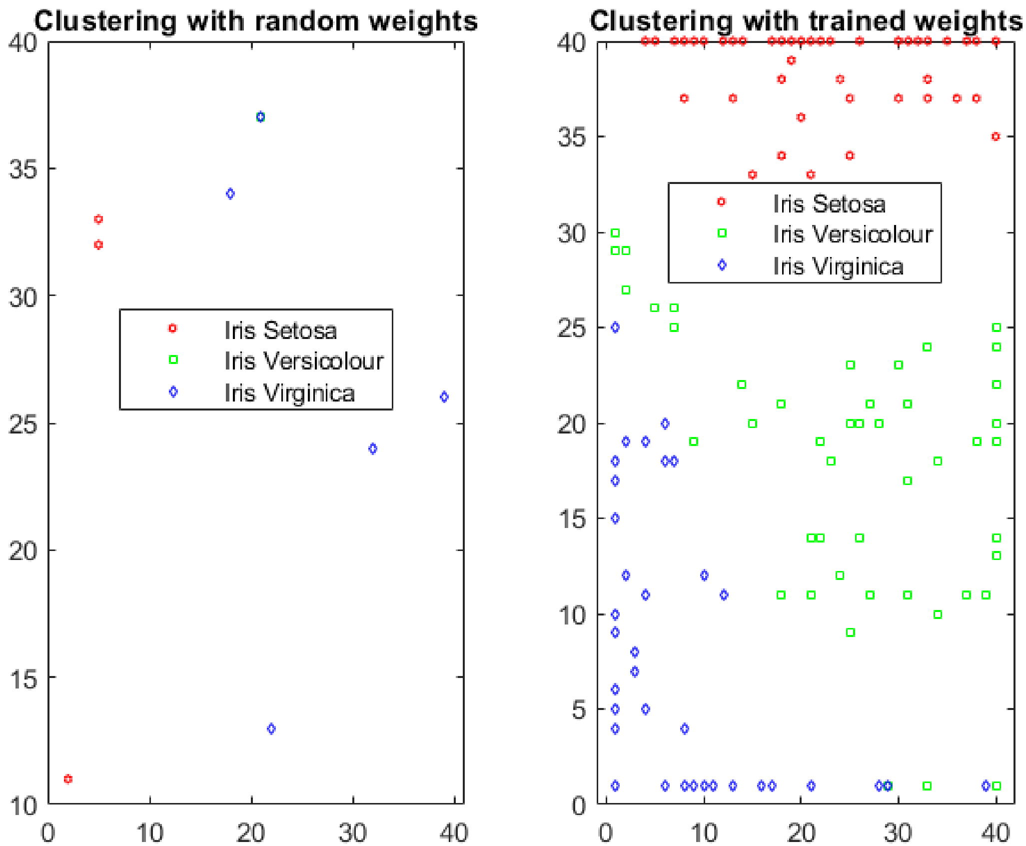
SELF ORGANIZING MAPS

```
clear;

iris_data = readmatrix('iris-data.csv');
iris_labels = readmatrix('iris-labels.csv');
iris_data = iris_data./max(iris_data);

no_of_instances = size(iris_data, 1);
no_of_attributes = size(iris_data, 2);
no_of_epochs = 10;
no_of_patterns = 3;
normalization_factor = max(iris_data);
initial_eta = 0.1;
decay_eta = 0.01;
initial_width = 10;
decay_width = 0.05;
map_size = 40;

weights = randn(map_size,map_size,no_of_attributes);
initial_location = zeros(no_of_instances, no_of_patterns);
final_location = zeros(no_of_instances, no_of_patterns);

for instance = 1:no_of_instances
    input = iris_data(instance, :);
    min_distance = inf;

    for i = 1:map_size
        for j = 1:map_size
            distance = 0;

            for k = 1:no_of_attributes
                distance = distance + (weights(i, j, k) -
input(k))^2;
            end

            if distance <= min_distance
                min_distance = distance;
                i_winning = i;
                j_winning = j;
            end
        end
    end
```

```
    initial_location(instance, :) = [i_winning, j_winning,
iris_labels(instance)];
end


for epoch = 1:no_of_epochs
    eta = initial_eta * exp(-decay_eta * epoch);
    width = initial_width * exp(-decay_width *epoch);
    for instance = 1:no_of_instances
        input_index = randi(no_of_instances);
        input = iris_data(input_index, :);
        min_distance = inf;

        for i = 1:map_size
            for j = 1:map_size
                distance = 0;

                for k = 1:no_of_attributes
                    distance = distance + (weights(i, j, k)
- input(k))^2;
                end


                if distance <= min_distance
                    min_distance = distance;
                    i_min = i;
                    j_min = j;
                    winning_position = [i_min, j_min];
                end
            end
        end

        for i = 1:map_size
            for j = 1:map_size
                neighbourhood_function = exp(-norm([i, j] -
winning_position)^2/(2 * width^2));
                if neighbourhood_function < 3 * width
                    for k = 1:no_of_attributes
                        delta_weights = eta *
neighbourhood_function * (input(k) - weights(i, j, k));
                        weights(i, j, k) = weights(i, j, k)
+ delta_weights;
                    end
                end
```

```
            end
        end
    end
end

for instance = 1:no_of_instances
    input = iris_data(instance, :);
    min_distance = inf;

    for i = 1:map_size
        for j = 1:map_size
            distance = 0;

            for k = 1:no_of_attributes
                distance = distance + (weights(i, j, k) -
input(k))^2;
            end

            if distance <= min_distance
                min_distance = distance;
                i_winning = i;
                j_winning = j;
            end
        end
    end
    final_location(instance, :) = [i_winning, j_winning,
iris_labels(instance)];
end
figure;
colors = 'rgb'; markers = 'hsdp';
ax1 = subplot(1,2,1);
gscatter(initial_location(:, 1), initial_location(:, 2),
initial_location(:, 3), colors, 'hsdp', 3);
title(ax1,'Clustering with random weights')
legend('Iris Setosa', 'Iris Versicolour', 'Iris
Virginica');

ax2 = subplot(1,2,2);
gscatter(final_location(:, 1), final_location(:, 2),
final_location(:, 3), colors, 'hsdp', 3);
title(ax2,'Clustering with trained weights')
legend('Iris Setosa', 'Iris Versicolour', 'Iris
Virginica');
```



A self-organizing map is trained to cluster the different varieties of flowers using the iris data set. The clustering is done on the basis of the attributes. Initially, we find out the winning neuron by calculating the distance between the input attribute and its corresponding weight vector for each flower and mapping is done. The left plot shows the output array off for all the data points with the random initialized weights. We can see here that there is no feature clustering at all.

However, the right plot shows the three different clusters when we train the weights using Kohonen's rule. We calculate the positions of the wining neurons after the training is done and plot the final data set that we get. In the plot we can see that some of the points are at very random locations, the reason for that is the size of the data set that we have.