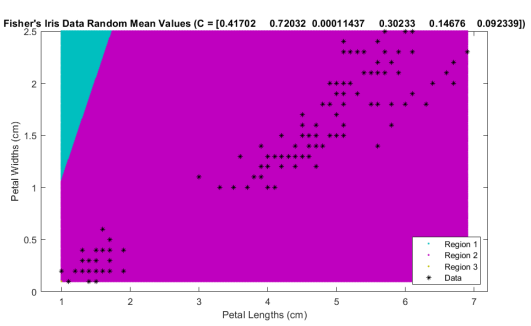
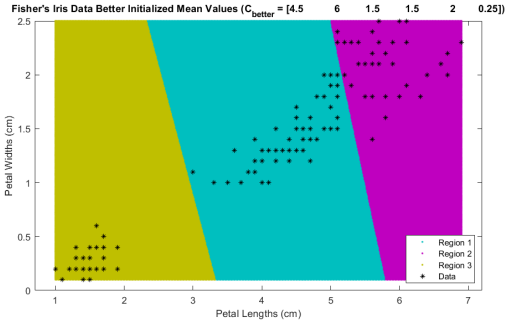
EE 583 Pattern Recognition HW4

Eda Özkaynar 2375582

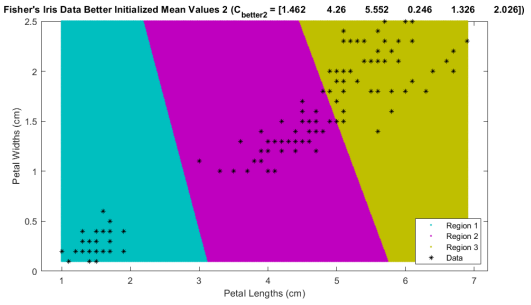
Question 1 Train a k-Means Clustering Algorithm



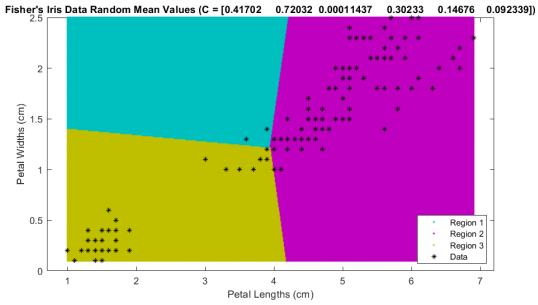
(a) Clustering with Randomly Initialized Mean Values (Results After 1 Iteration)



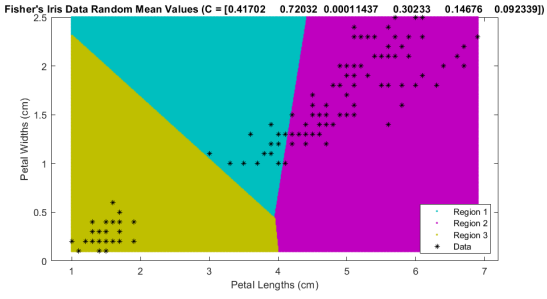
(b) Clustering with Randomly Initialized Mean Values (Results After 1 Iteration)



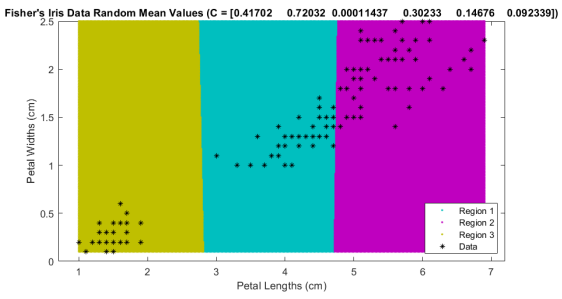
(c) Clustering with Randomly Initialized Mean Values (Results After 1 Iteration)



(d) Clustering with Randomly Initialized Mean Values (Results After 10 Iteration)



(e) Clustering with Randomly Initialized Mean Values (Results After 20 Iteration)



(f) Clustering with Randomly Initialized Mean Values (Results After 25 Iteration)

Fig. 1: K-Mean Clustering with different mean values and number of iterations

To measure quality of clustering I used Within-Cluster Sum of Squares (WCSS), also known as the inertia or distortion measure in clustering. Lower values indicate that data points are closer to their respective cluster centroids, meaning better clustering. Higher values indicate more spread-out clusters or poor clustering. The function computes the total squared Euclidean distance between each data point and the centroid of the cluster it belongs to. This measures how tightly the data points are grouped within their assigned clusters.

The formula for WCSS (Within-Cluster Sum of Squares) is:

$$\text{WCSS} = \sum_{i=1}^{N} \|X_i - C_{k(i)}\|^2$$

where:

- N: Total number of data points.
- X_i: Data point i.
- $C_{k(i)}$: Centroid of the cluster to which X_i is assigned.
- $\|\cdot\|$: Euclidean norm.

TABLE I: Clustering Quality Comparison

Initialization Method	**Quality (WCSS)**
Random Initialization	2.5294×10^3
Better Initialized Centroids	31.4129
Class-Based Centroids	31.3714

Random initialization can lead to poor clustering results, especially with datasets that have clear clusters. Choosing centroids carefully, either by hand or with knowledge about the data, greatly improves these results. In this case, using a class-based approach gives the best clustering outcome, showing how using prior knowledge or thoughtful choices can help k-means clustering. Moreover, As can be seen in Figure 1, although random initialization starts with poor clusters, the algorithm successfully refines clusters over multiple iterations, eventually converging to a stable solution that aligns well with the true class structure.

Question 2 Select the Number of Gaussian Mixture Model Components Using PCA

Gaussian Mixture Model (GMM) is a probabilistic model that represents a dataset as a mixture of multiple Gaussian distributions. It is widely used for clustering, density estimation, and unsupervised learning.

Mixture Model

- A mixture model assumes that data points are generated from a combination of several distributions.
- In GMM, each distribution is a Gaussian (normal) distribution.
- The overall data distribution is the weighted sum of these Gaussian components:

$$p(x) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x \mid \mu_k, \Sigma_k)$$

  where:
  - K: Number of Gaussian components.
  - π_k: Weight (or prior probability) of the k-th Gaussian component, satisfying $\sum_{k=1}^{K} \pi_k = 1$.
  - μ_k: Mean vector of the k-th Gaussian.
  - Σ_k: Covariance matrix of the k-th Gaussian.
  - $\mathcal{N}(x \mid \mu_k, \Sigma_k)$: Probability density function of the k-th Gaussian.
- The data points are assumed to come from a combination of Gaussian distributions.
- Each data point is assigned a probability of belonging to each Gaussian component.

In a Gaussian Mixture Model (GMM), the components consist of individual Gaussian distributions that collectively form the mixture model. Each component signifies a cluster or subset of data points sharing similar characteristics, which are modeled by a Gaussian distribution.

- **1 Component:**
  - The contour encloses all the data points in a single cluster.
- **2 Components:**
  - Two distinct clusters are formed, which partially separate the species.

- **3 Components:**
  - Three distinct clusters are formed, and the GMM aligns well with the true species labels.

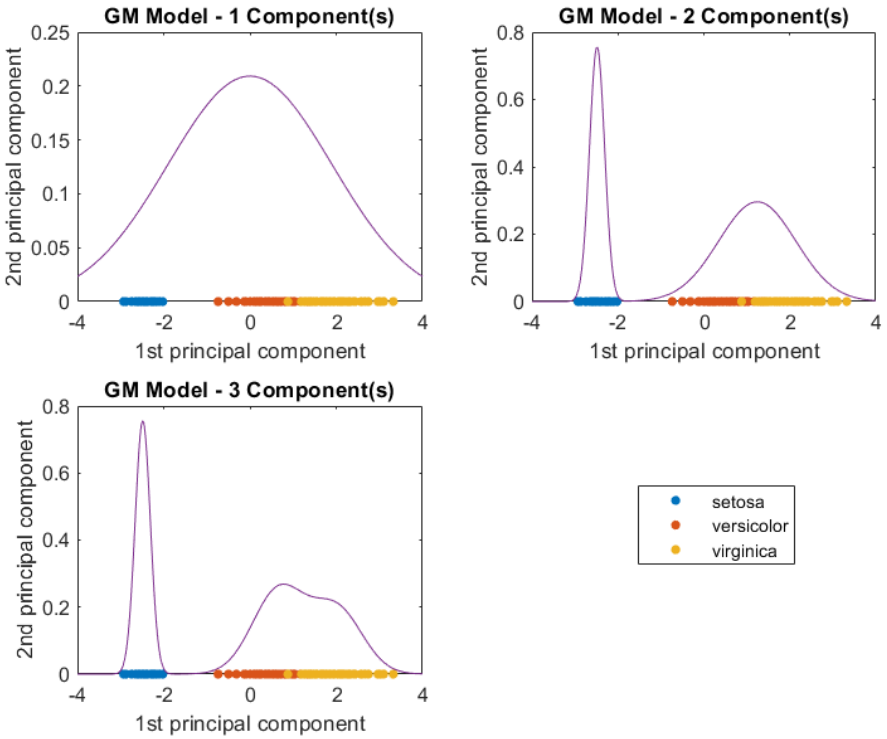


Fig. 2: Select the Number of Gaussian Mixture Model Components Using PCA

1. Gaussian Mixture Model (GMM) - 1 Component: - A single Gaussian fit treats all data as one group, which does not allow for distinguishing between the three classes: Setosa, Versicolor, and Virginica.

2. Gaussian Mixture Model (GMM) - 2 Components: - With two Gaussian components, there is some separation, particularly with Setosa being distinct. However, Versicolor and Virginica still overlap.

3. Gaussian Mixture Model (GMM) - 3 Components: - The three Gaussian components fit the three classes well, effectively modeling Setosa and providing partial separation between Versicolor and Virginica, with peaks appearing for two different mean values, but some overlap remains.

Question 3 Cluster Data Using Dissimilarity Matrix

Definition

A dissimilarity matrix D is an $n \times n$ square matrix, where n is the number of data points. Each element $D(i, j)$ represents the **dissimilarity** (or distance) between data points i and j.

- **Diagonal elements:** $D(i, i) = 0$, because a data point is identical to itself (no dissimilarity).
- **Off-diagonal elements:** $D(i, j)$ measures how "different" two data points are.

How is a Dissimilarity Matrix Computed?

A dissimilarity matrix is typically derived from a **distance metric**, such as:

1) **Euclidean Distance:**

$$D(i, j) = \sqrt{\sum_{k=1}^{d} (x_{ik} - x_{jk})^2}$$

Measures the straight-line distance between points in d-dimensional space.

2) **Manhattan Distance:**

$$D(i, j) = \sum_{k=1}^{d} |x_{ik} - x_{jk}|$$

Measures the "taxicab" distance by summing absolute differences.

3) **Cosine Dissimilarity:**

$$D(i,j) = 1 - \frac{x_i \cdot x_j}{\|x_i\|\|x_j\|}$$

Measures the angular difference between two vectors.

4) **Mahalanobis Distance:**

$$D(i,j) = \sqrt{(x_i - x_j)^T \Sigma^{-1} (x_i - x_j)}$$

Accounts for correlations between features.

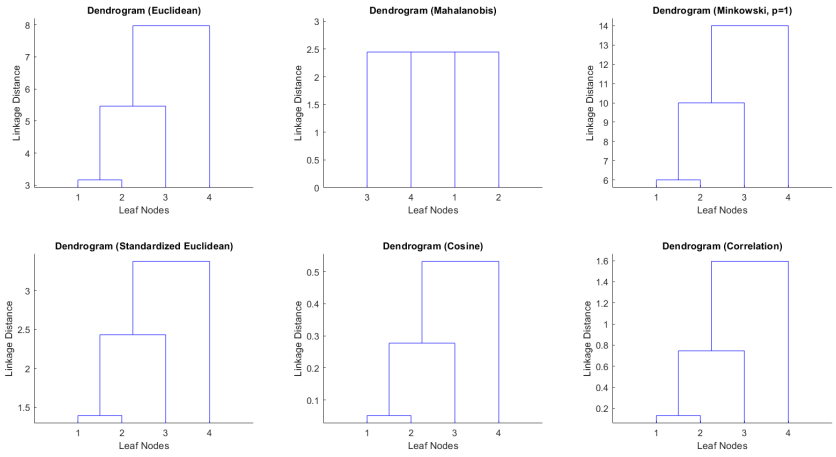


Fig. 3: Dendrograms for different distance metrics using weighted average distance between clusters

Metrics other than Mahalanobis distance showed similar clustering performance and dendrogram for this data. However, Mahalanobis distance shows abrupt merges. Sudden mergers occur because Mahalanobis overcompensates for feature correlations and changes the distance between certain clusters.

QUESTION 4

In example, it constructs the similarity matrix S using a Gaussian kernel.

$$S(i,j) = \exp\left(-dist(i,j)^2\right) \tag{1}$$

Euclidean distance is used to calculate the distance between nodes. The similarity matrix S captures how similar data points are based on their distances and uses kernels like Gaussian to map distances into similarity scores.

- A Laplacian matrix is constructed from S:

$$L = D - S$$

Here, D is the degree matrix, where

$$D(i,i) = \sum_j S(i,j).$$

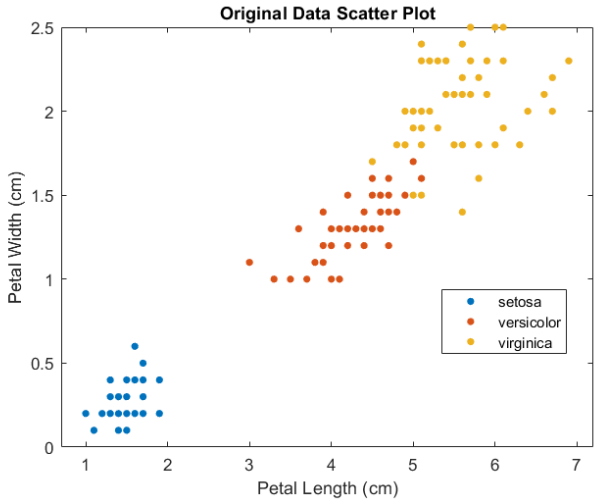


Fig. 4: Original Data scatter plots

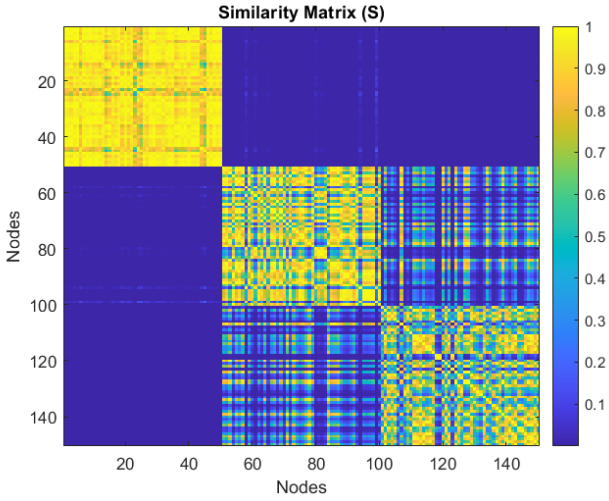


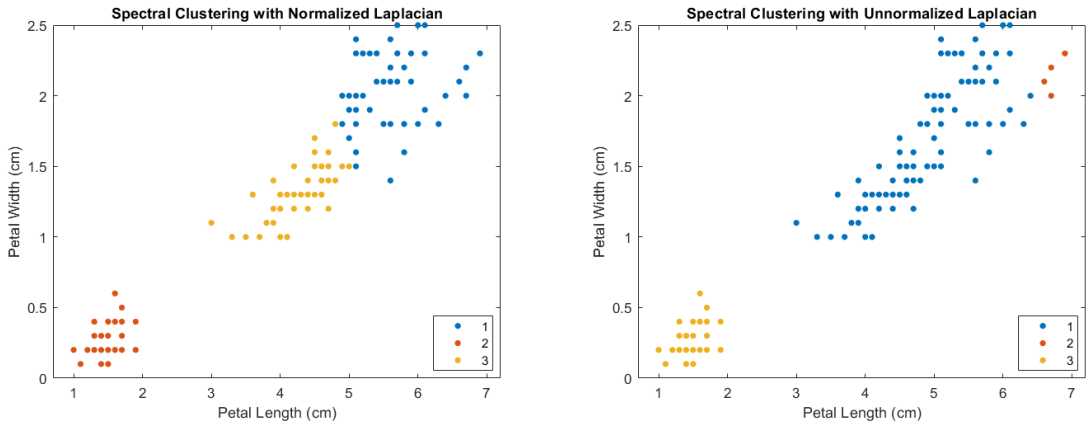
Fig. 5: Similarity Matrix

1) Normalized vs. Unnormalized Laplacian: Using the unnormalized Laplacian matrix causes nodes with higher degrees (more connected) to dominate the graph structure and affect the clustering results. It also works well for balanced graphs where all nodes have approximately the same degree.

Symmetric Normalized Laplacian

$$L_{\text{sym}} = I - D^{-1/2} S D^{-1/2}$$

- $D^{-1/2}$ normalizes the adjacency matrix by the degrees of the nodes.
- Ensures symmetry of the matrix, which is useful for spectral clustering and eigenvalue computation.



(a) (b)

Fig. 6: Normalized vs Unnormalized Laplacian Matrices

The normalized Laplacian (a) clearly outperforms the unnormalized Laplacian (b) in this dataset, as it produces clusters that better match the expected grouping. As a result, the normalized Laplacian is better suited for spectral clustering when the dataset has unbalanced clusters, varying densities, or irregular connectivity.

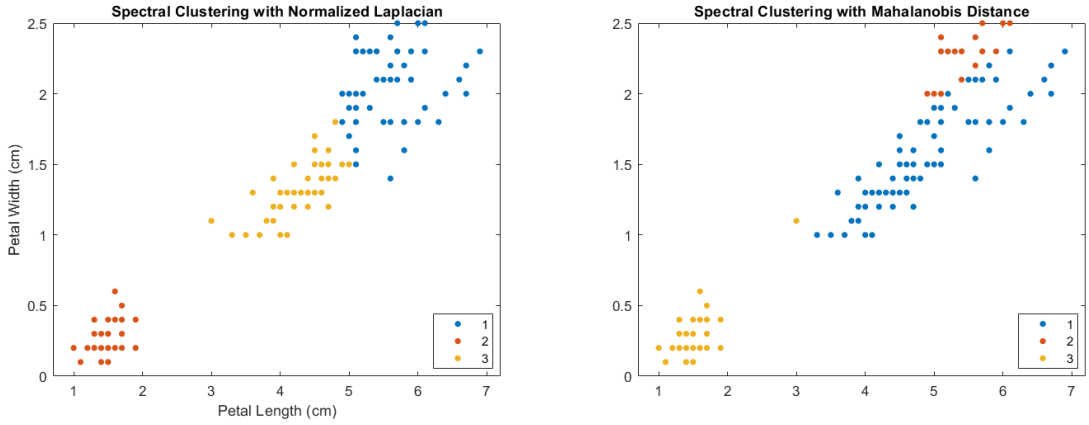
2) Euclidean vs. Mahalanobis Distance:

Euclidean Distance

It works well when features have similar scales and are uncorrelated. Sensitive to feature scaling, for example; if one feature has larger values, it dominates the distance calculation. It cannot handle datasets where features are correlated or have varying distributions.

Mahalanobis Distance

It measures the distance between two points while accounting for the correlations between features and their variances. Therefore, it handles correlated features effectively and is better suited for data with anisotropic clusters.



(a) (b)

Fig. 7: Euclidean vs Mahalanobis Distance

Euclidean distance provided better clustering performance. Mahalanobis distance, although powerful for handling correlated or anisotropic clusters, overcomplicated the clustering process by adjusting for correlations that were not strongly present in the data and provided poor performance.

3) Kernel Scale: KernelScale plays a major role in creating the similarity matrix used in the Spectral Clustering algorithm. It controls the width or scale of the Gaussian Kernel function. The Spectral Clustering algorithm usually uses a Gaussian (RBF) kernel to create the similarity matrix S as follows:

$$S(i,j) = \exp\left(-\frac{d(i,j)^2}{2 \cdot \text{KernelScale}^2}\right)$$

KernelScale determines how significant this distance is:
Large Kernel Scale Values: Creates similarity between points over a larger distance. Focuses on capturing the overall structure of the data.
Small Kernel Scale Values: Strong similarities only occur between close neighbors. Similarity values between distant points are almost zero.

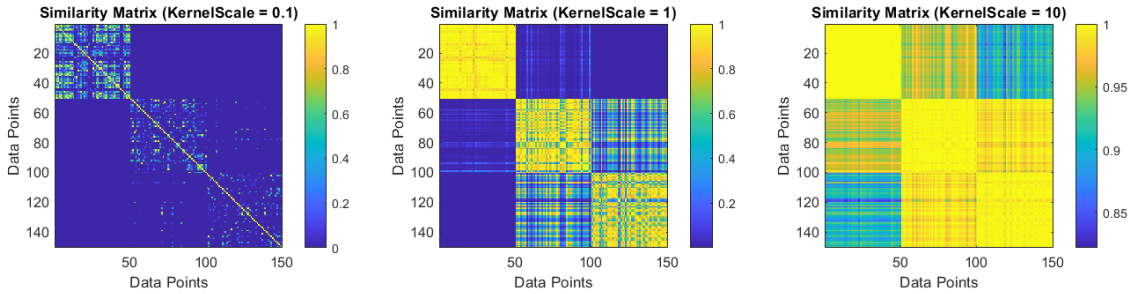


Fig. 8: Similarity Matrices for Different Kernel Scales

- Smaller KernelScale (0.1): Captures fine-grained, local relationships.
- Larger KernelScale (10): Captures broad, global relationships but loses fine details.
- For datasets with clear clusters, a moderate KernelScale (1) typically yields the best results, as it balances local and global similarities.

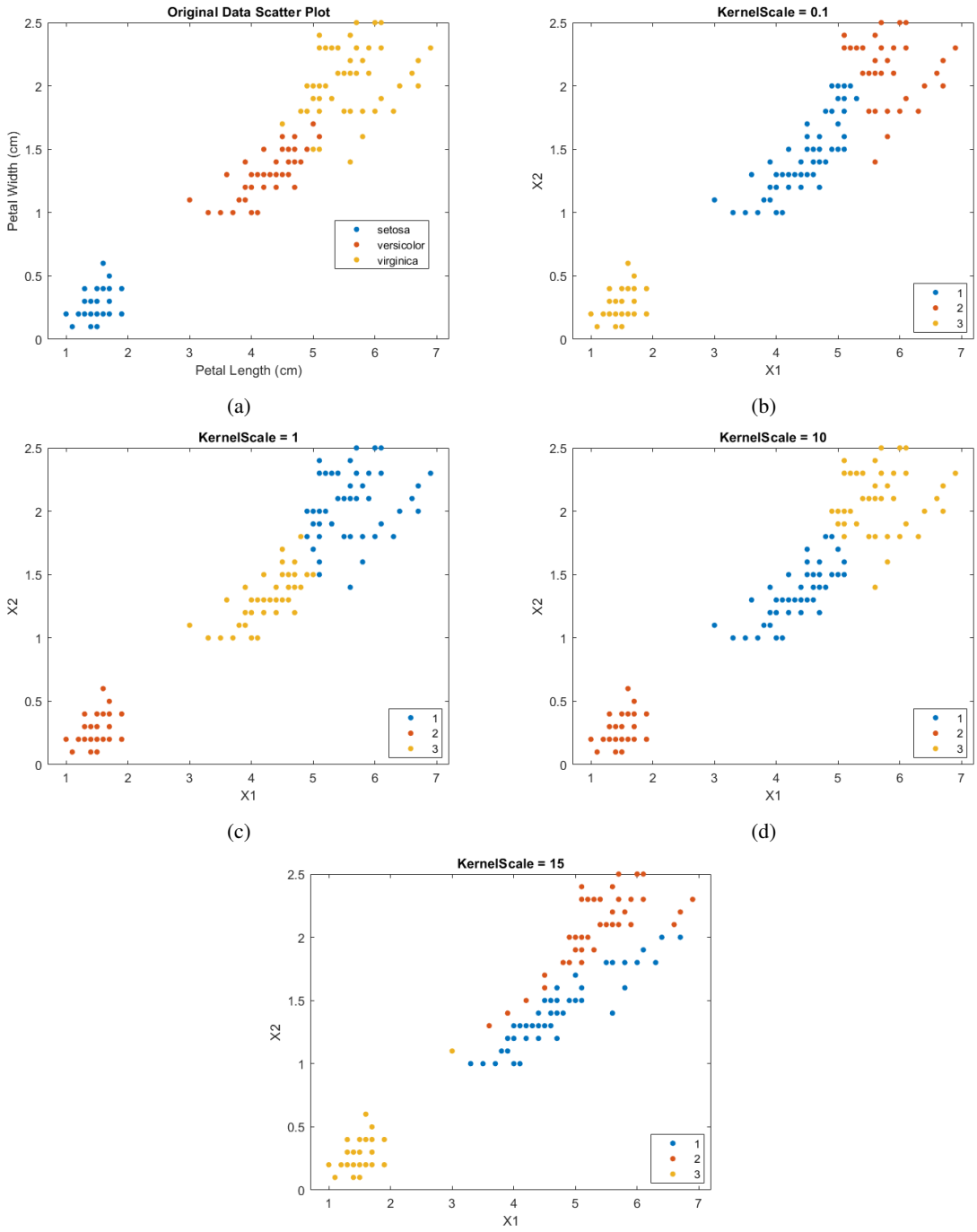


Fig. 9: Spectral Clustering for Different Kernel Scale Sizes

For kernel scale 15, Clustering accuracy is the lowest. For kernel scale 0.1, Setosa cluster (blue) is well defined due to its separation. For other classes, small clusters are formed within each group, which causes misclassification. Kernel scale 1 and 10 show similar performance but, as can be seen in Figure 8, kernel size 1 provides better clustering performance.

```
clear; clc; close all;

% Load Fisher's iris data set. Use the petal lengths and widths as predictors.
load fisheriris
X = meas(:,3:4);

figure;
plot(X(:,1),X(:,2),'k*','MarkerSize',5);
title 'Fisher''s Iris Data';
xlabel 'Petal Lengths (cm)';
ylabel 'Petal Widths (cm)';

rng(1); % For reproducibility

C = rand(3,2); % Randomly initialized centroids
x1 = min(X(:,1)):0.01:max(X(:,1));
x2 = min(X(:,2)):0.01:max(X(:,2));
[x1G,x2G] = meshgrid(x1,x2);
XGrid = [x1G(:),x2G(:)]; % Defines a fine grid on the plot

% K-Means with random centroids
idx2Region = kmeans(XGrid,3,'MaxIter',25,'Start',C);
[idx,C_1,sumd] = kmeans(X,3,"Distance","sqeuclidean","Start",C);

% Assigns each node in the grid to the closest centroid
figure;
gscatter(XGrid(:,1),XGrid(:,2),idx2Region,...
    [0,0.75,0.75;0.75,0,0.75;0.75,0.75,0],'..');
hold on;
plot(X(:,1),X(:,2),'k*','MarkerSize',5);
title(['Fisher''s Iris Data Random Mean Values (C = [', num2str(C(:)'), '])']);
xlabel 'Petal Lengths (cm)';
ylabel 'Petal Widths (cm)';
legend('Region 1','Region 2','Region 3','Data','Location','SouthEast');
hold off;

% Measure Clustering Quality
Quality = class_quality(X,idx,C);

% K-Means with better initial centroids
C_better = [4.5, 1.5; 6, 2; 1.5, 0.25];
idx2Region_better = kmeans(XGrid,3,'MaxIter',1,'Start',C_better);
[idx2,C_better1,sumd2] = kmeans(X,3,'Start',C_better,'Distance','sqeuclidean');

% Assigns each node in the grid to the closest centroid
figure;
gscatter(XGrid(:,1),XGrid(:,2),idx2Region_better,...
    [0,0.75,0.75;0.75,0,0.75;0.75,0.75,0],'..');
hold on;
plot(X(:,1),X(:,2),'k*','MarkerSize',5);
title(['Fisher''s Iris Data Better Initialized Mean Values (C_{better} = [',↙
num2str(C_better(:)'), '])']);
xlabel 'Petal Lengths (cm)';
```

```
ylabel 'Petal Widths (cm)';
legend('Region 1','Region 2','Region 3','Data','Location','SouthEast');
hold off;

% Measure Clustering Quality
Quality_better = class_quality(X,idx2,C_better1);

% K-Means with centroids based on class means
C_better2 = [mean(X(1:50,:)); mean(X(51:100,:)); mean(X(101:end,:))];
[idx3,C_better3,sumd3] = kmeans(X,3,'Start',C_better2,'Distance','sqeuclidean');
idx2Region_better2 = kmeans(XGrid,3,'MaxIter',1,'Start',C_better2);

% Assigns each node in the grid to the closest centroid
figure;
gscatter(XGrid(:,1),XGrid(:,2),idx2Region_better2,...
    [0,0.75,0.75;0.75,0,0.75;0.75,0.75,0],'..');
hold on;
plot(X(:,1),X(:,2),'k*','MarkerSize',5);
title(['Fisher''s Iris Data Better Initialized Mean Values 2 (C_{better2} = [',↙
num2str(C_better2(:)'), '])']);
xlabel 'Petal Lengths (cm)';
ylabel 'Petal Widths (cm)';
legend('Region 1','Region 2','Region 3','Data','Location','SouthEast');
hold off;

% Measure Clustering Quality
Quality_better2 = class_quality(X,idx3,C_better3);
Quality_random = class_quality(X,idx3,C_1);

clear; clc; close all;
load fisheriris
classes = unique(species);

% Use principal component analysis to reduce the dimension of the data to two↙
dimensions for visualization.
[~,score] = pca(meas(:,3:4),'NumComponents',1);

GMModels = cell(3,1); % Preallocation
options = statset('MaxIter',1000);
rng(1); % For reproducibility

for j = 1:4
    GMModels{j} = fitgmdist(score,j,'Options',options);
    fprintf('\n GM Mean for %i Component(s)\n',j)
    Mu = GMModels{j}.mu;
end

figure;
for j = 1:4
    subplot(2,2,j)
    h1 = gscatter(score,zeros(size(score)),species);
    h = gca;
    hold on
```

```
    gmPDF = @(x) arrayfun(@(x0) pdf(GMModels{j},x0),x);
    fplot(gmPDF, [h.XLim(1), h.XLim(2)]);
    % fcontour(gmPDF,[h.XLim h.YLim],'MeshDensity',100)
    title(sprintf('GM Model - %i Component(s)',j));
    xlabel('1st principal component');
    ylabel('2nd principal component');
    if(j ~= 3)
        legend off;
    end
    hold off
end
g = legend(h1);
g.Position = [0.7 0.25 0.1 0.1];


clear; clc; close all;
% Define the data matrix (each row is an observation, each column is a variable)
X = [0 1 2 3;
     1 0 4 5;
     2 4 0 6;
     3 5 6 0];

Z = linkage(X, 'centroid');
% Mahalanobis Distance
D_mahalanobis = pdist(X, 'mahalanobis'); % Precompute Mahalanobis distances
Z_mahalanobis = linkage(D_mahalanobis, 'centroid'); % Perform hierarchical clustering

% Minkowski Distance (with p = 3)
D_minkowski = pdist(X, 'minkowski', 1); % p = 1
Z_minkowski = linkage(D_minkowski, 'centroid');

% Standardized Euclidean Distance
D_seuclidean = pdist(X, 'seuclidean'); % Precompute Standardized Euclidean distances
Z_seuclidean = linkage(D_seuclidean, 'centroid');

% cosine Distance
D_cosine = pdist(X, 'cosine'); % Precompute Standardized Euclidean distances
Z_cosine = linkage(D_cosine, 'centroid');

% correlation Distance
D_corr = pdist(X, 'correlation'); % Precompute Standardized Euclidean distances
Z_correlation = linkage(D_corr, 'centroid');

% Display the linkage matrices
disp('Linkage Matrix (Mahalanobis):');
disp(Z_mahalanobis);
disp('Linkage Matrix (Minkowski):');
disp(Z_minkowski);
disp('Linkage Matrix (Standardized Euclidean):');
disp(Z_seuclidean);
```

```
% Plot dendrograms for each distance metric

figure;
subplot(2,3,1);
dendrogram(Z);
title('Dendrogram (Euclidean)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

subplot(2,3,2);
dendrogram(Z_mahalanobis);
title('Dendrogram (Mahalanobis)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

subplot(2,3,3);
dendrogram(Z_minkowski);
title('Dendrogram (Minkowski, p=1)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

subplot(2,3,4);
dendrogram(Z_seuclidean);
title('Dendrogram (Standardized Euclidean)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

subplot(2,3,5);
dendrogram(Z_cosine);
title('Dendrogram (Cosine)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

subplot(2,3,6);
dendrogram(Z_correlation);
title('Dendrogram (Correlation)');
xlabel('Leaf Nodes');
ylabel('Linkage Distance');

clear; clc; close all;

% Load Fisher Iris Dataset
load fisheriris
X = meas(:,3:4);

% Scatter plot of the original data
figure;
gscatter(X(:,1), X(:,2), species);
title('Original Data Scatter Plot');
xlabel('Petal Length (cm)');
ylabel('Petal Width (cm)');

% Compute distance matrix and similarity matrix
```

```
dist_temp = pdist(X);
dist = squareform(dist_temp);

% Similarity matrix
S = exp(-dist.^2);
disp(['Is Similarity Matrix Symmetric: ', num2str(issymmetric(S))]);
% Compute degree matrix
D = diag(sum(S, 2));

% Compute Laplacian Matrices
% 1. Unnormalized Laplacian
L_unnormalized = D - S;

% 2. Symmetric Normalized Laplacian
L_sym = eye(size(S)) - D^(-1/2) * S * D^(-1/2);

% Visualize Laplacian Matrices
figure;
imagesc(L_unnormalized);
colorbar;
title('Unnormalized Laplacian Matrix (L)');
xlabel('Nodes');
ylabel('Nodes');

figure;
imagesc(L_sym);
colorbar;
title('Symmetric Normalized Laplacian Matrix (L_{sym})');
xlabel('Nodes');
ylabel('Nodes');

figure;
imagesc(S);
colorbar;
title('Similarity Matrix (S)');
xlabel('Nodes');
ylabel('Nodes');

% Spectral clustering
k = 3; % Number of clusters
rng('default') % For reproducibility
idx = spectralcluster(S, k, 'Distance', 'precomputed', 'LaplacianNormalization',↙
'symmetric');

% Spectral clustering with unnormalized Laplacian
idx_unnormalized = spectralcluster(S, k, 'Distance', 'precomputed',↙
'LaplacianNormalization', 'none');

% Plot results with symmetric Laplacian
figure;
gscatter(X(:,1), X(:,2), idx);
title('Spectral Clustering with Normalized Laplacian');
xlabel('Petal Length (cm)');
```

```
ylabel('Petal Width (cm)');

% Plot results with unnormalized Laplacian
figure;
gscatter(X(:,1), X(:,2), idx_unnormalized);
title('Spectral Clustering with Unnormalized Laplacian');
xlabel('Petal Length (cm)');
ylabel('Petal Width (cm)');

% Mahalanobis Distance
[idx_mahalanobis, ~] = spectralcluster(X, k,'NumNeighbors', size(X,1), 'Distance',↙
'mahalanobis');
figure;
gscatter(X(:,1), X(:,2), idx_mahalanobis);
title('Spectral Clustering with Mahalanobis Distance');

% KernelScale = 0.1
idx1 = spectralcluster(X, k,'NumNeighbors', size(X,1), 'KernelScale', 0.1,↙
'LaplacianNormalization', 'symmetric');

% KernelScale = 1
idx2 = spectralcluster(X, k,'NumNeighbors', size(X,1), 'KernelScale', 1,↙
'LaplacianNormalization', 'symmetric');

% KernelScale = 10
idx3 = spectralcluster(X, k,'NumNeighbors', size(X,1), 'KernelScale', 10,↙
'LaplacianNormalization', 'symmetric');

% KernelScale = 15
idx4 = spectralcluster(X, k,'NumNeighbors', size(X,1), 'KernelScale', 15,↙
'LaplacianNormalization', 'symmetric');

% Visualize the results
figure;

gscatter(X(:,1), X(:,2), idx1);
title('KernelScale = 0.1');
xlabel('X1');
ylabel('X2');
figure;
gscatter(X(:,1), X(:,2), idx2);
title('KernelScale = 1');
xlabel('X1');
ylabel('X2');
figure;
gscatter(X(:,1), X(:,2), idx3);
title('KernelScale = 10');
xlabel('X1');
ylabel('X2');

figure;
gscatter(X(:,1), X(:,2), idx4);
title('KernelScale = 15');
```

```
xlabel('X1');
ylabel('X2');

% Convert species to numerical labels for comparison
true_labels = grp2idx(species); % 1 = setosa, 2 = versicolor, 3 = virginica

% Q_default = correct_classification(true_labels,idx,k);
% Q_unnormalized = correct_classification(true_labels,idx_unnormalized,k);
% Q_mahalanobis = correct_classification(true_labels,idx_mahalanobis,k);
% Q_kernel01 = correct_classification(true_labels,idx1,k);
% Q_kernel1 = correct_classification(true_labels,idx2,k);
% Q_kernel10 = correct_classification(true_labels,idx3,k);


% Compute distance matrix
dist_temp = pdist(X);
dist = squareform(dist_temp);

% Define different KernelScale values
kernel_scales = [0.1, 1, 10, 15];

% Plot similarity matrices for each KernelScale
figure;
for i = 1:length(kernel_scales)
    % Compute similarity matrix with given KernelScale
    kernel_scale = kernel_scales(i);
    S = exp(-dist.^2 / (2 * kernel_scale^2));

    % Plot the similarity matrix
    subplot(2, 2, i);
    imagesc(S);
    colorbar;
    title(['Similarity Matrix (KernelScale = ', num2str(kernel_scale), ')']);
    xlabel('Data Points');
    ylabel('Data Points');
end


function [correct_class1, correct_class2,correct_class3] = correct_classification↙
(true_labels,idx,k)

    % Initialize counts
    correct_class1 = 0;
    correct_class2 = 0;
    correct_class3 = 0;

    % Compare true labels with predicted clusters
    for i = 1:k
        % Find majority cluster for each class
        class_indices = (true_labels == i);
        predicted_clusters = idx(class_indices);

        % Majority voting: most frequent cluster in true class
```

```
        most_frequent_cluster = mode(predicted_clusters);
        correct_count = sum(predicted_clusters == most_frequent_cluster);

        % Store correct counts
        if i == 1
            correct_class1 = correct_count;
        elseif i == 2
            correct_class2 = correct_count;
        elseif i == 3
            correct_class3 = correct_count;
        end
    end

    % Display results
    fprintf('Correct classifications:\n');
    fprintf('Class 1 (Setosa): %d / %d\n', correct_class1, sum(true_labels == 1));
    fprintf('Class 2 (Versicolor): %d / %d\n', correct_class2, sum(true_labels ==↙
2));
    fprintf('Class 3 (Virginica): %d / %d\n', correct_class3, sum(true_labels ==↙
3));
end



% Function to compute clustering quality
function Quality = class_quality(X,idx,C)
Quality = 0;
    for i = 1:length(X)
        Quality = Quality + norm(X(i,:) - C(idx(i),:)).^2;
    end
end
```