Lab11_Hierarchical_Clustering

October 23, 2024

1 Step 1: Prepare the dataset

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the dataset
points = np.array([
    [1, 1],   # p1
    [3, 2],   # p2
    [9, 1],   # p3
    [3, 7],   # p4
    [7, 2],   # p5
    [9, 7],   # p6
    [4, 8],   # p7
    [8, 3],   # p8
    [1, 4]    # p9
])
```

2 Step 2: Distance Functions

[2]:
```
def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

def manhattan_distance(a, b):
    return np.sum(np.abs(a - b))

def minkowski_distance(a, b, p=3):
    return np.power(np.sum(np.abs(a - b) ** p), 1/p)

def proximity_matrix(points, distance_func):
    size = points.shape[0]
    matrix = np.zeros((size, size))
    for i in range(size):
        for j in range(size):
            if i != j:
                matrix[i][j] = distance_func(points[i], points[j])
```

```
    return matrix
```

3 Step 3: Proximity Matrices

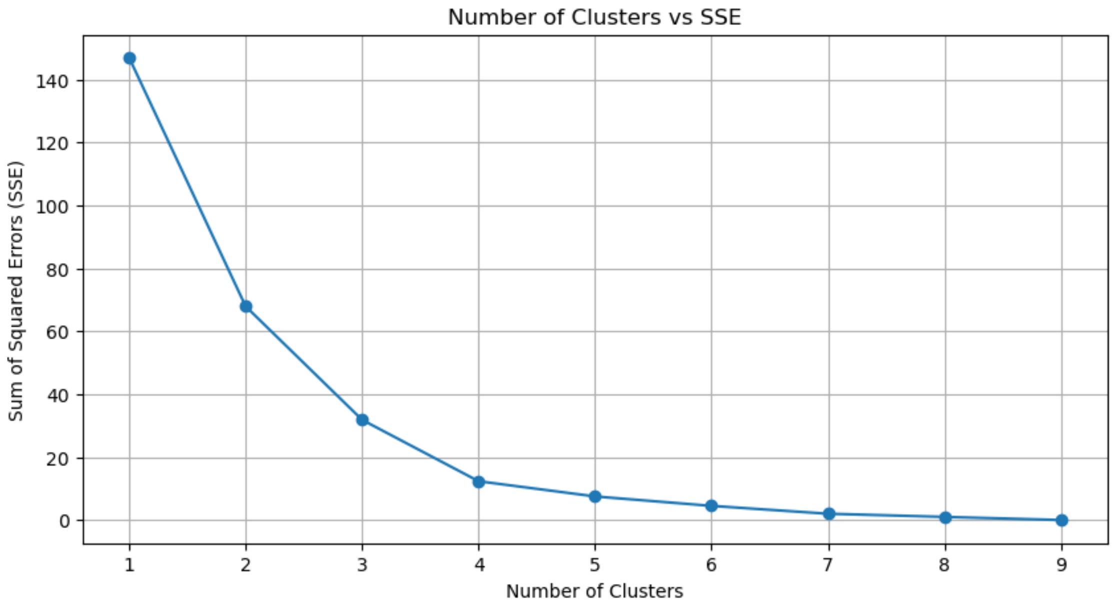
[3]:
```
euclidean_dist_matrix = proximity_matrix(points, euclidean_distance)
manhattan_dist_matrix = proximity_matrix(points, manhattan_distance)
minkowski_dist_matrix = proximity_matrix(points, minkowski_distance)

print("Euclidean Distance Matrix:\n", euclidean_dist_matrix)
print("\nManhattan Distance Matrix:\n", manhattan_dist_matrix)
print("\nMinkowski Distance Matrix:\n", minkowski_dist_matrix)
```

```
Euclidean Distance Matrix:
 [[ 0.          2.23606798  8.          6.32455532  6.08276253 10.
   7.61577311  7.28010989  3.        ]
 [ 2.23606798  0.          6.08276253  5.          4.          7.81024968
   6.08276253  5.09901951  2.82842712]
 [ 8.          6.08276253  0.          8.48528137  2.23606798  6.
   8.60232527  2.23606798  8.54400375]
 [ 6.32455532  5.          8.48528137  0.          6.40312424  6.
   1.41421356  6.40312424  3.60555128]
 [ 6.08276253  4.          2.23606798  6.40312424  0.          5.38516481
   6.70820393  1.41421356  6.32455532]
 [10.          7.81024968  6.          6.          5.38516481  0.
   5.09901951  4.12310563  8.54400375]
 [ 7.61577311  6.08276253  8.60232527  1.41421356  6.70820393  5.09901951
   0.          6.40312424  5.        ]
 [ 7.28010989  5.09901951  2.23606798  6.40312424  1.41421356  4.12310563
   6.40312424  0.          7.07106781]
 [ 3.          2.82842712  8.54400375  3.60555128  6.32455532  8.54400375
   5.          7.07106781  0.        ]]

Manhattan Distance Matrix:
 [[ 0.  3.  8.  8.  7. 14. 10.  9.  3.]
 [ 3.  0.  7.  5.  4. 11.  7.  6.  4.]
 [ 8.  7.  0. 12.  3.  6. 12.  3. 11.]
 [ 8.  5. 12.  0.  9.  6.  2.  9.  5.]
 [ 7.  4.  3.  9.  0.  7.  9.  2.  8.]
 [14. 11.  6.  6.  7.  0.  6.  5. 11.]
 [10.  7. 12.  2.  9.  6.  0.  9.  7.]
 [ 9.  6.  3.  9.  2.  5.  9.  0.  8.]
 [ 3.  4. 11.  5.  8. 11.  7.  8.  0.]]

Minkowski Distance Matrix:
 [[0.         2.08008382 8.         6.07317794 6.00924501 8.99588289
  7.17905435 7.05400406 3.        ]
 [2.08008382 0.         6.00924501 5.         4.         6.98636803
```

```
 6.00924501 5.01329793 2.5198421 ]
[8.         6.00924501 0.         7.5595263  2.08008382 6.
 7.76393608 2.08008382 8.13822304]
[6.07317794 5.         7.5595263  0.         5.73879355 6.
 1.25992105 5.73879355 3.27106631]
[6.00924501 4.         2.08008382 5.73879355 0.         5.10446872
 6.24025147 1.25992105 6.07317794]
[8.99588289 6.98636803 6.         6.         5.10446872 0.
 5.01329793 4.02072576 8.13822304]
[7.17905435 6.00924501 7.76393608 1.25992105 6.24025147 5.01329793
 0.         5.73879355 4.49794145]
[7.05400406 5.01329793 2.08008382 5.73879355 1.25992105 4.02072576
 5.73879355 0.         7.00679612]
[3.         2.5198421  8.13822304 3.27106631 6.07317794 8.13822304
 4.49794145 7.00679612 0.        ]]
```

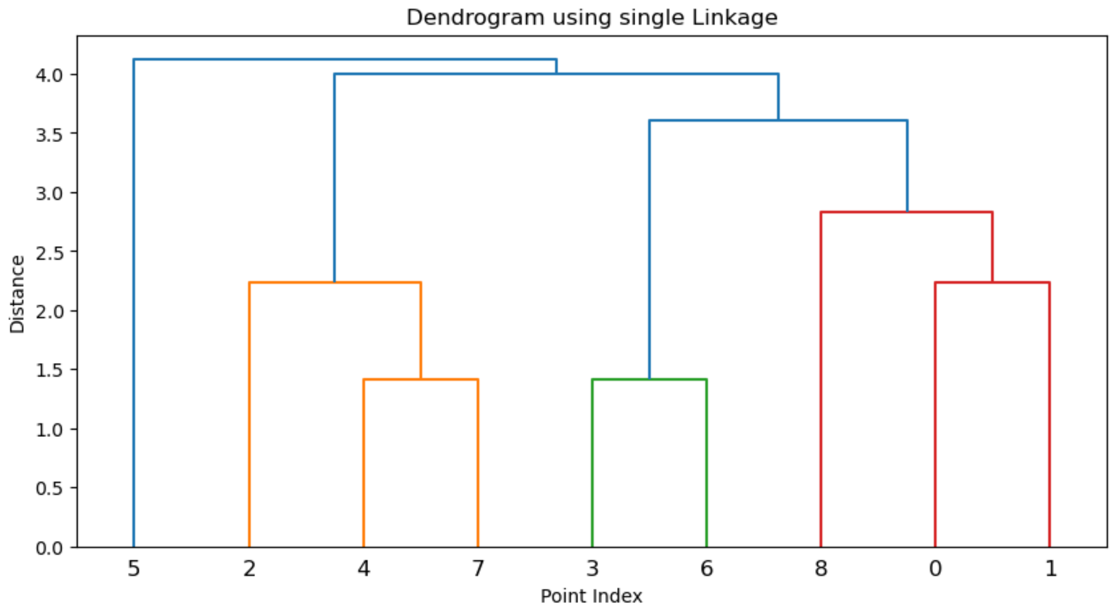
4 Step 4: Hierarchical Clustering and SSE Calculation

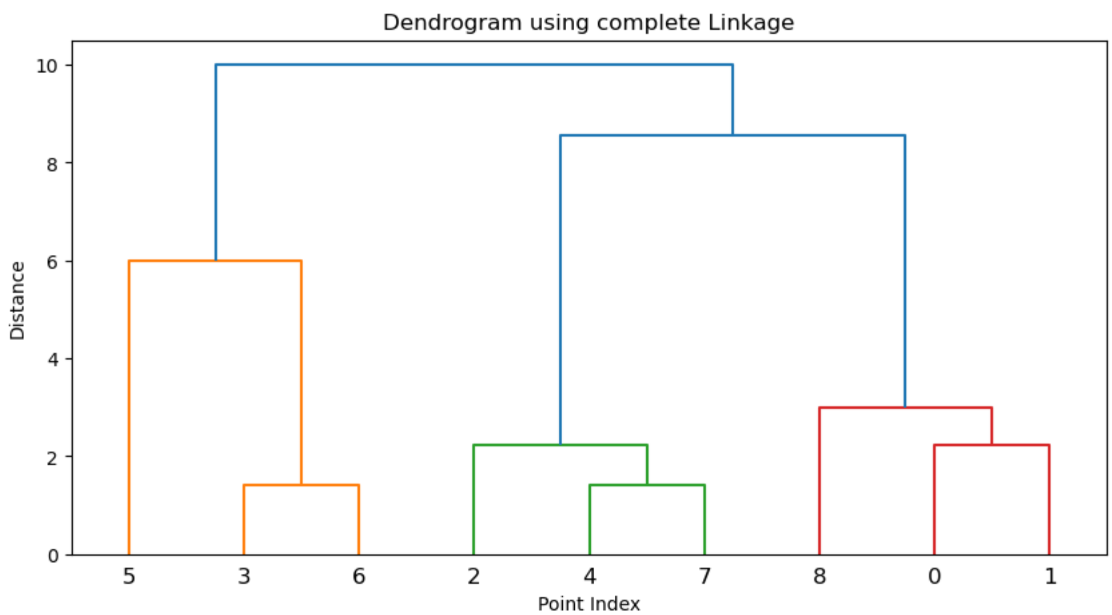
[9]:
```python
from sklearn.cluster import AgglomerativeClustering

def compute_sse(points, labels):
    sse = 0
    for cluster_id in np.unique(labels):
        cluster_points = points[labels == cluster_id]
        cluster_center = cluster_points.mean(axis=0)
        sse += np.sum((cluster_points - cluster_center) ** 2)
    return sse

# Determine the number of clusters and SSE
sse_values = []
num_clusters = range(1, 10)
for n_clusters in num_clusters:
    clustering = AgglomerativeClustering(n_clusters=n_clusters)
    labels = clustering.fit_predict(points)
    sse = compute_sse(points, labels)
    sse_values.append(sse)

# Plot SSE vs Number of Clusters
plt.figure(figsize=(10, 5))
plt.plot(num_clusters, sse_values, marker='o')
plt.title('Number of Clusters vs SSE')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Errors (SSE)')
plt.grid(True)
plt.show()
```

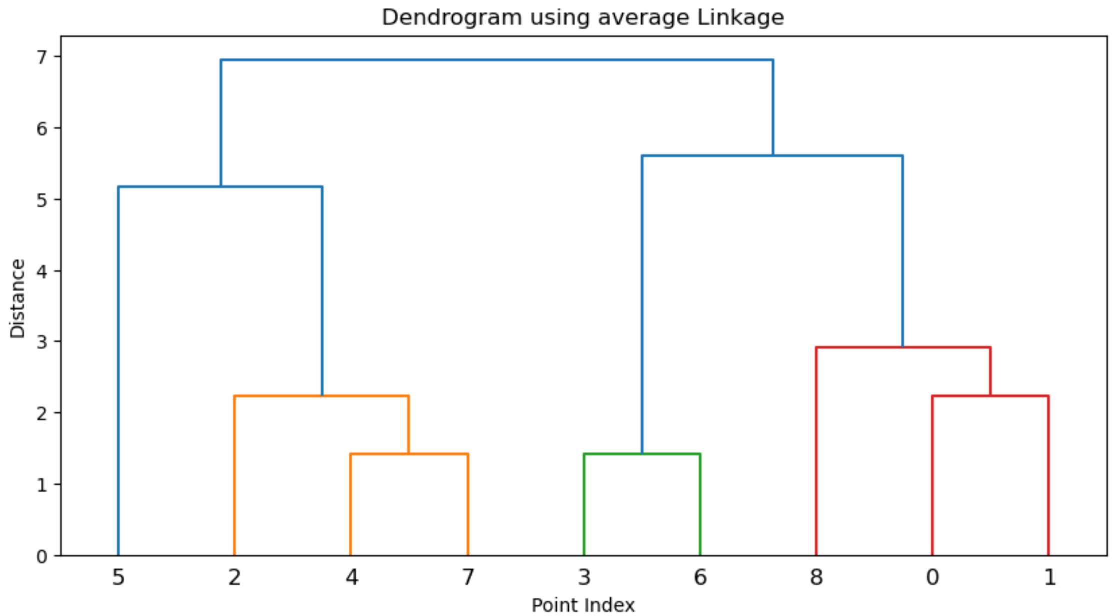


5 Step 5: Plotting the Dendrogram

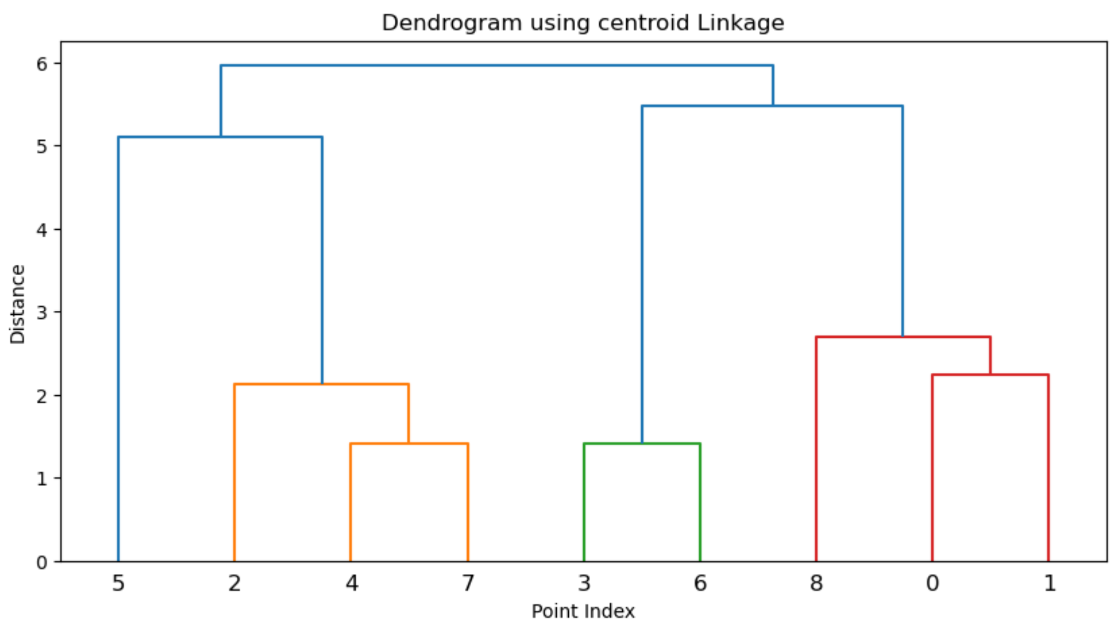
[10]:
```
import scipy.cluster.hierarchy as sch

def plot_dendrogram(method):
    plt.figure(figsize=(10, 5))
    Z = sch.linkage(points, method=method)
    sch.dendrogram(Z)
    plt.title(f'Dendrogram using {method} Linkage')
    plt.xlabel('Point Index')
    plt.ylabel('Distance')
    plt.show()

# Plot dendrograms for different linkage methods
linkage_methods = ['single', 'complete', 'average', 'centroid', 'ward']
for method in linkage_methods:
    plot_dendrogram(method)
```


Dendrogram using single Linkage
Distance
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
5
2
4
7
3
6
8
0
1
Point Index

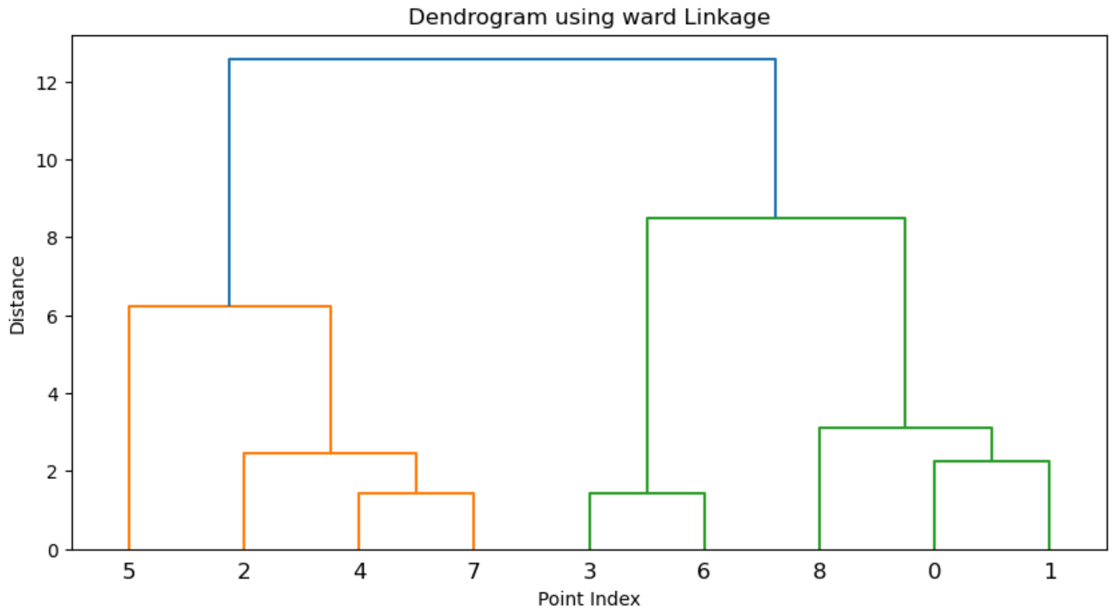

Dendrogram using complete Linkage
Distance
0
2
4
6
8
10
5
3
6
2
4
7
8
0
1
Point Index



Dendrogram using average Linkage
Distance
Point Index
5 2 4 7 3 6 8 0 1



Dendrogram using centroid Linkage
Distance
Point Index
5 2 4 7 3 6 8 0 1


Dendrogram using ward Linkage
Distance
0
2
4
6
8
10
12
5
2
4
7
3
6
8
0
1
Point Index

[ ]: