Textbook Exercises

MiningMassive Datasets Page 252 --- Exercise 7.2.2

Using points from example 7.2, calculating the distance between two clusters

a) the minimum of the distances between any two points, one from each cluster

b) the average of the distances between pairs of points, one from each of the two clusters.

In [57]:
```
# Points from Example 7.2
points = [(2,2), (5,2), (3,4), (9,3), (12,3), (11,4), (10,5), (12,6), (4,8), (6,8),
(4,10), (7,10)]
```

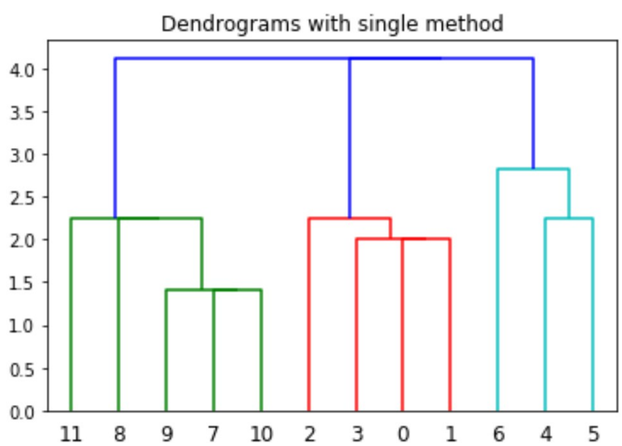
In [89]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as shc
```

In [90]:
```
points1 = np.array([[4,10], [4,8], [7,10], [6,8], [3,4], [2,2], [5,2], [10,5], [9,3],
[12,3], [11,4], [12,6]])
```

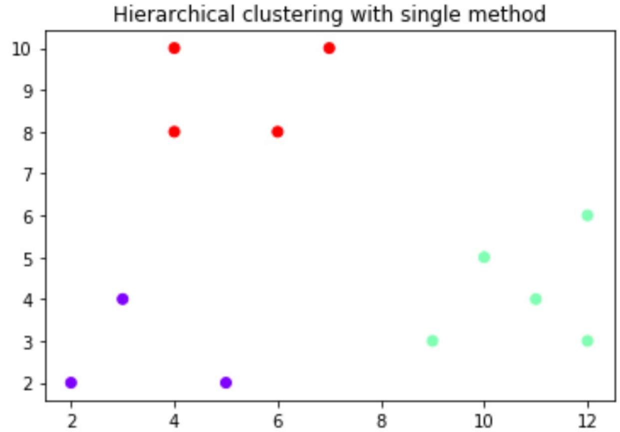
In [91]:
```
def dendrogram(data, method):
    plt.title('Dendrograms with {} method'.format(method))
    dend = shc.dendrogram(shc.linkage(data, method=method))
```

In [99]:
```
def h_cluster(n, method, data):
    cluster = AgglomerativeClustering(n_clusters=n, affinity='euclidean', linkage=meth
od)
    cluster.fit_predict(data)
    plt.scatter(points1[:,0],points1[:,1], c=cluster.labels_, cmap= 'rainbow')
    plt.title('Hierarchical clustering with {} method'.format(method))
```

In [97]:
```
dendrogram(points1, 'single')
```



In [100]:
```
h_cluster(3, 'single', points1)
```



In [101]:
```
cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
```

the clusters look very similar to what is displayed in figure 7.5 in the book. But the hierarchy is different due to changing the order of the pairings.

Section B: the average of the distances between pairs of points, one from each of the two clusters

In [61]:
```
# takes two lists of points and returns avg distance between the two
def AvgDistBetweenClusters(cluster1,cluster2):
    n = len(cluster1)*len(cluster2)
    sum_distances = 0
    for point1 in cluster1:
        for point2 in cluster2:
            sum_distances += abs(euclidean_length(np.array(point1)-np.array(point2)))
    average_distance = sum_distances / n
    return average_distance

# Takes a dictionary point:centroid and returns a list of points corresponding to a ce
rtain centroid
def GetPointsCluster(cluster_dct, centroid):
    points_in_cluster = []
    for point, centroid_ref in cluster_dct.items():
        if centroid_ref == centroid:
            points_in_cluster.append(point)
    return points_in_cluster

# Function for Part B
def hierarchical_clustering_b(points, target_cluster):
    cluster_dct = {}
    for i in range(len(points)):
        cluster_dct[points[i]] = points[i]

    # assign target clusters
    target = target_cluster

    # assign list unique values to clusters variable
    clusters = list(cluster_dct.values())
    unique = [x for x in set(tuple(x) for x in clusters)]
    num_clusters = len(unique)

    #While loop until desired number of clusters is reached
    while num_clusters > target:
        smallest_dist = 99999999

        for cluster1 in unique:
            points1 = GetPointsCluster(cluster_dct, cluster1)
            for cluster2 in unique:
                points2 = GetPointsCluster(cluster_dct, cluster2)
                distance = AvgDistBetweenClusters(points1, points2)
                if (cluster1 != cluster2) and (distance < smallest_dist):
                    smallest_dist = distance
                    combine_clusters = [cluster1, cluster2]

        new_centroid = 0
        points_new_centroid = []
        centroid_1 = combine_clusters[0]
        centroid_2 = combine_clusters[1]
        for point, centroid in cluster_dct.items():
            if centroid == centroid_1 or centroid == centroid_2:
                points_new_centroid.append(point)

        # average of all points from combined clusters
        new_centroid = tuple(np.sum(points_new_centroid, axis = 0)/len(points_new_cent
roid))

        # replace the centroid/cluster value for all points in the combined cluster
        for point, centroid in list(cluster_dct.items()):
            if centroid == centroid_1 or centroid == centroid_2:
                cluster_dct[point] = new_centroid

        clusters = list(cluster_dct.values())
        unique = [x for x in set(tuple(x) for x in clusters)]
        num_clusters = len(unique)
```

In [62]:
```
# Exercise b with 4 clusters
hierarchical_clustering_b(points, 4)
```

Out[62]:
```
{(2, 2): (3.3333333333333335, 2.6666666666666665),
 (5, 2): (3.3333333333333335, 2.6666666666666665),
 (3, 4): (3.3333333333333335, 2.6666666666666665),
 (9, 3): (10.5, 3.75),
 (12, 3): (10.5, 3.75),
 (11, 4): (10.5, 3.75),
 (10, 5): (10.5, 3.75),
 (12, 6): (12, 6),
 (4, 8): (5.25, 9.0),
 (6, 8): (5.25, 9.0),
 (4, 10): (5.25, 9.0),
 (7, 10): (5.25, 9.0)}
```

Mining Massive Datasets Page 260 --- Exercise 7.3.4

compute the representation of the cluster as in the BFR algorithm. Compute N, SUM, SUMSQ.

Compute variance and standard deviation for each cluster.

In [10]:
```
import pandas as pd
import numpy as np
```

In [11]:
```
# clusters from example 7.8
cluster1 = points[0:3]
cluster2 = points[3:8]
cluster3 = points[8:12]
```

In [12]:
```
clusters = [cluster1, cluster2, cluster3]

print('-'*40)#creating dashed line grid
for cluster in clusters:
    N = len(cluster)#calculating length of cluster
    SUM = tuple(np.sum(cluster, axis = 0)) # calculating the sum of all components in cluster
    SUMSQ = tuple(np.sum(np.square(cluster), axis = 0))# calculating the sum of squares
    variance = tuple(np.divide(SUMSQ,N) - np.square(np.divide(SUM, N))) # calculating the variance
    standard_deviation = tuple(np.sqrt(variance)) # calculating the standard deviation
    print('Cluster: {}'.format(cluster))
    print('N: {}'.format(N))
    print('SUM: {}'.format(SUM))
    print('SUMSQ: {}'.format(SUMSQ))
    print('Variance: {}'.format(variance))
    print('Standard Deviation: {}'.format(standard_deviation))
    print('-'*40)
```

```
----------------------------------------
Cluster: [(2, 2), (5, 2), (3, 4)]
N: 3
SUM: (10, 8)
SUMSQ: (38, 24)
Variance: (1.5555555555555536, 0.8888888888888893)
Standard Deviation: (1.247219128924646 4, 0.9428090415820636)
----------------------------------------
Cluster: [(9, 3), (12, 3), (11, 4), (10, 5), (12, 6)]
N: 5
SUM: (54, 21)
SUMSQ: (590, 95)
Variance: (1.3599999999999852, 1.3599999999999994)
Standard Deviation: (1.1661903789690538, 1.1661903789690597)
----------------------------------------
Cluster: [(4, 8), (6, 8), (4, 10), (7, 10)]
N: 4
SUM: (21, 36)
SUMSQ: (117, 328)
Variance: (1.6875, 1.0)
Standard Deviation: (1.299038105676658, 1.0)
----------------------------------------
```

Mining Massive Datasets Page 260 --- Exercise 7.3.5

Computing Mahalanobis Distance

In [13]:

```
## Function computes the Mahalanobis distances given the standard deviation and two po
int lists.
## lists are of equal length and dimension
def mahalanobis_distance(standard_dev, pointa, pointb):
    dimensions = len(standard_dev)
    temp_sum = 0
    for i in range(dimensions):
        temp_sum += ((pointa[i] - pointb[i])/standard_dev[i])**2
    distance = temp_sum**(1/2)
    return distance

point = [1,-3,4]
origin = [0,0,0]
std = [2,3,5]
mahalanobis_distance(std, origin, point)
```

Out[13]: 1.374772708486752

In [ ]: