Introduction to Machine Learning
Assignment 2

Group 31
Stijn Kammer (s4986296) & Ramon Kits (s5440769)

September 28, 2022

Introduction

In this report, results of the second assignment of the course Machine Learning will be discussed. The assignment was to implement the agglomerative hierarchical clustering algorithm. Different linkage methods were implemented and tested on the dataset delivered by the course. The results of the different linkage methods and finding the optimal number of clusters are discussed in this report.

Methods

The agglomerative hierarchical clustering algorithm is an iterative algorithm which starts with each data point in its own cluster. The algorithm then iteratively merges the two closest clusters. The distance between two clusters is defined by the linkage method. The algorithm stops when all data points are in one cluster. After the algorithm has finished, the data points are clustered in a tree structure. The tree structure can be used to find the optimal number of clusters. Four linkage methods that may be used are single, complete, average and ward's linkage.

Single linkage uses the minimum distance between objects in two clusters. To determine the distance between two clusters, the minimum distance between all objects in the two clusters is used. This works well for clusters that are clearly separated. Complete linkage uses the maximum distance between objects in two clusters. To determine the distance between two clusters, the maximum distance between all objects in the two clusters is used, this is the opposite of single linkage. This can lead to clusters that are very spread out because clusters with closer objects may not be merged because it has objects that are further away. Both of the last two linkage methods are higly sensitive to outliers and noise. Average linkage uses the average distance between objects in two clusters. It uses the average distance between all objects in the two clusters to determine the distance between the two clusters. This makes it more robust to outliers. At last, ward's linkage is the most complex linkage method of the four. It uses the variance of the clusters to determine the distance between the clusters. The variance is the sum of the squared distances between the objects in the cluster and the mean of the cluster. When the increase in variance is at its minimum, the clusters will me berged. This linkage method is in many cases preferred over the other three because it is very robust to outliers and it is more sensitive to the shape of the clusters.

implementation

For the data clustering the agglomerative hierarchical clustering algorithm was implemented. The algorithm was implemented in Python and the results were plotted using matplotlib. All four

linkage methods were implemented and tested on the dataset delivered by the course. Also the amount of clusters was determined using the dendrogram and the silhouette score.

Listing 1: *Plotting the dendrogram*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score

AFFINITY = 'euclidean'
LINKAGES = ['single', 'complete', 'average', 'ward']

# Read data from csv file
data = pd.read_csv('data_clustering.csv', header=None)
data = data.values

for linkageMeasures in LINKAGES:
    # create hierarchical clustering model
    # generate and display dendrogram and color the clusters using the above set colors
    plt.figure(figsize=(10, 7))
    plt.title("Dendrogram for {} linkage".format(linkageMeasures))
    plt.xlabel("Data points")
    plt.ylabel("Euclidean distance")
    dendrogram(linkage(data, method=linkageMeasures,
                metric=AFFINITY), no_labels=True, color_threshold=0)
```

In Listing 1 the most important part of the code is shown which plots the dendrogram. The code imports the data from the csv file and and performs hierarchical clustering using the four linkage methods. The code then plots the dendrograms using the scipy library. This dendrogram can be used to determine the optimal number of clusters.

Listing 2: *Plotting the dendrogram*

```
def ownSilhouetteScore(data, labels):
    # Calculate the silhouette score for the given data and labels

    # Calculate intra-cluster distances
    intraClusterDistances = np.zeros(len(data))
    for i in range(len(data)):
        # Calculate the average distance to all other points in the same cluster
        intraClusterDistances[i] = np.mean(np.linalg.norm(
            data[labels == labels[i]] - data[i], axis=1))

    # Calculate inter-cluster distances
    interClusterDistances = np.zeros(len(data))
    for i in range(len(data)):
        # Calculate the average distance to all other points in the nearest cluster
        interClusterDistances[i] = np.min(
```

```
            np.mean(np.linalg.norm(data[labels != labels[i]] - data[i],
    axis=1)))

    # Calculate the silhouette score
    silhouetteScore = np.mean((interClusterDistances -
    intraClusterDistances) /
                              np.maximum(intraClusterDistances,
    interClusterDistances))

    return silhouetteScore


# Read data from csv file
data = pd.read_csv('data_clustering.csv', header=None)
data = data.values

# calculate silhouette score for different configuations of the model
results = {'single': [], 'complete': [], 'average': [], 'ward': []}
nclustersList = range(MINCLUSTERS, MAXCLUSTERS + 1)
for nClusters in nclustersList:
    for linkage in results.keys():
        hac = AgglomerativeClustering(
            n_clusters=nClusters, affinity=AFFINITY, linkage=linkage)
        hac.fit(data)
        results[linkage].append(silhouette_score(data, hac.labels_, ))

# display results in a bar chart
x = np.arange(len(nclustersList))
width = 0.20
fig, ax = plt.subplots()
plt.ylim(top=1)
rects1 = ax.bar(x - 2*width, results['single'], width, label='Single')
rects2 = ax.bar(x - width, results['complete'], width, label='Complete'
    )
rects3 = ax.bar(x, results['average'], width, label='Average')
rects4 = ax.bar(x + width, results['ward'], width, label='Ward')
for rect in rects1 + rects2 + rects3 + rects4:
    height = rect.get_height()
    ax.annotate(round(height, 2),
                xy=(rect.get_x() + rect.get_width() / 2, height),
                textcoords="offset points",
                ha='center', va='bottom', rotation=22.5)
ax.set_ylabel('Silhouette score')
ax.set_xlabel('Number of clusters')
ax.set_title('Silhouette score for different configurations')
ax.set_xticks(x)
ax.set_xticklabels(nclustersList)
ax.legend()
fig.tight_layout()
plt.show()
```

In Listing 2 is the code shown which uses agglomerative hierarchical clustering using the built-in

function of sklearn and calculates the silhouette scores. This code makes it possible to use either the silhouette calculation of the sklearn library as well as the possibility to calculate the silhouette score manually which uses the following formula:

$$S = \frac{1}{n} \sum_{i=1}^{n} \frac{(b_i - a_i)}{max(a_i, b_i)}$$

Where a_i is the average distance between the object and all other objects in the same cluster. b_i is the average distance between the object and all other objects in the next nearest cluster. And n is the number of objects in the dataset.

The avreage scores are calculated for the four linkage methods and a range of possible number of clusters. After the scores are calculated, the code plots the scores using matplotlib in a bar chart. This chart makes it easy to find the optimal number of clusters.

Listing 3: *Plotting the clusters for different linkage methods*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score

AFFINITY = 'euclidean'
LINKAGES = ['single', 'complete', 'average', 'ward']
MINCLUSTERS = 2
MAXCLUSTERS = 4

# Read data from csv file
data = pd.read_csv('data_clustering.csv', header=None)
data = data.values

# display data in scatter plot
plt.figure(figsize=(10, 7))
plt.title("Data without clustering")
plt.scatter(data[:, 0], data[:, 1], c='black', s=7)
plt.savefig('output/data_scatter.png')


def calculateWSS(data, labels):
    # Calculate the WSS for the given data and labels

    # calculate the centroids
    centroids = np.zeros((len(np.unique(labels)), len(data[0])))
    for i in range(len(np.unique(labels))):
        centroids[i] = np.mean(data[labels == i], axis=0)

    # calculate the cluster sum of squares
    wss = 0
    for i in range(len(data)):
        wss += np.sum((data[i] - centroids[labels[i]]) ** 2)

    return wss
```

```
38
39
40  def calculateBSS(data, labels):
41      # Calculate the BSS for the given data and labels
42
43      # calculate the centroids
44      centroids = np.zeros((len(np.unique(labels)), len(data[0])))
45      for i in range(len(np.unique(labels))):
46          centroids[i] = np.mean(data[labels == i], axis=0)
47
48      # calculate the cluster sum of squares
49      bss = 0
50      for i in range(len(centroids)):
51          bss += np.sum((centroids[i] - np.mean(centroids)) ** 2)
52
53      return bss
54
55
56  for linkageMeasure in LINKAGES:
57      for nClusters in range(MINCLUSTERS, MAXCLUSTERS + 1):
58          hac = AgglomerativeClustering(
59              n_clusters=nClusters, affinity=AFFINITY, linkage=
    linkageMeasure)
60          # fit the model
61          hac.fit(data)
62          # display clusters in scatter plot
63          plt.figure(figsize=(10, 7))
64          plt.title("Scatter plot for {} linkage and {} clusters".format
    (
65              linkageMeasure, nClusters))
66          plt.xlabel("x1")
67          plt.ylabel("x2")
68          plt.scatter(data[:, 0], data[:, 1], c=hac.labels_, cmap='
    rainbow')
69          print("{} WSS: {} with {} clusters".format(linkageMeasure,
70                round(calculateWSS(data, hac.labels_), 2), nClusters))
71          print("{} BSS: {} with {} clusters".format(linkageMeasure,
72                round(calculateBSS(data, hac.labels_), 2), nClusters))
73
74          # save the plot
75          plt.savefig(
76              'output/{}_{}_clusters.png'.format(linkageMeasure,
    nClusters))
```

In Listing 3 the code is shown which uses agglomerative hierarchical clustering with the four linkage methods and a range of number of clusters and plots the clusters using matplotlib. This way it is possible to see how the clusters are formed by the different linkage methods.

What the code also is able to do is manually calculating the Within Cluster Sum of Squares

$$WSS = \sum_{i=1}^{N_C} \sum_{x \in C_i} d(X, \bar{X}_{C_i})^2$$

which represents the sum of the squared distances between the objects within the cluster. And the Between Cluster Sum of Squares

$$BSS = \sum_{i=1}^{N_C} |C_i| \cdot d(\bar{X}_{C_i}, X)^2$$

Which represents the sum of the squared distances between the mean of the clusters and the objects in the dataset. This applied to the different linkage methods and range of number of clusters. These values are being printed in the console so they can be analyzed for each combination of linkage method and number of clusters.

Results

Add results

Work distribution

Work distribution