HierarchicalAgglomeretive_Class19-Completed

April 18, 2021

[56]:
```python
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
from plotnine import *

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import AgglomerativeClustering


from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

from sklearn.metrics import silhouette_score

# make sure you have these to make dendrograms!-------
import scipy.cluster.hierarchy as sch
from matplotlib import pyplot as plt
#-----------------------------------------------------

%matplotlib inline
```

1 0. Together

Hierarchical clustering (as the name suggests) assumes that the clusters in the data have a hierarchical relationship. For example, in a McDonald's food dataset we could have clusters like: Dessert, Drinks, Sandwhiches, Other. Within Sandwhiches we could have chicken sandwhiches, Burgers, Vegan Sandwhiches…within Burgers we could have smaller, lower calorie options, and bigger, more substantial burgers…etc.

Blood cells is another great example of a hierarchical relationship:

Hierarchical *Agglomeretive* Clustering (which we perform here), goes bottom up: every datapoint starts as it's own singleton cluster, and at each step, we merge the two closest clusters together until all data points are in one big cluster. In order to decide which clusters are closest, we need to choose two things:

- **distance metric**: this is a measure that helps us define how close together two *data points* are. Euclidean distance is a common distance metric that you may be familiar with, but there is also cosine distance, manhattan distance, hamming distance, and even custom distance functions (like levenshtein distance between two strings!)
- **linkage criteria**: this is a measure of how close two *clusters* are. Because (most) clusters have more than one point, we need to define what it means for two clusters to be close.

Single Linkage: the distance between two clusters (A and B) as the minimum distance between any point in A and any point in B

Average Linkage: the distance between two clusters (A and B) as the average distance between points in A and points in B.

Complete Linkage: the distance between two clusters (A and B) as the maximum distance between any point in A and any point in B.

Centroid Method: the distance between two clusters (A and B) as the distance between their respective mean vectors (centroids).

Ward's Method (default): the distance between two clusters (A and B) as the Sum of Squared Errors when combining the two clusters together.

and MORE! You could technically define this anyway you wanted.

2 1. Linkage Critera

Let's build a few functions that return the distance between two clusters. Each of these functions take in two dataframes, `A` and `B` that contain the datapoints for the two respective clusters (*number of features can vary*).

The functions take in two arguments:

- `A`: an N1 x P dataframe containing the data points in cluster A. (N1 is the number of data points in cluster A; P is the number of features used)
- `B`: an N2 x P dataframe containing the data points in cluster B. (N2 is the number of data points in cluster B; P is the number of features used)

The function should calculate and return the distance between the clusters (assume you're using euclidean distance for all of these) according to their respective linkage criterion (single, average, and complete).

2.0.1 *Question

To calculate the distance between two clusters `A` (N1 x P) and `B` (N2 x P), how many distances (between 2 data points) would you have to calculate?

[57]:
```python
def single(A,B):
    distances = []

    for A_index in range(0, A.shape[0]):
        for B_index in range(0, B.shape[0]):
            d = np.linalg.norm(A.iloc[A_index]-B.iloc[B_index])
            distances.append(d)

    return(min(distances))


def average(A,B):
    distances = []

    for A_index in range(0, A.shape[0]):
        for B_index in range(0, B.shape[0]):
            d = np.linalg.norm(A.iloc[A_index]-B.iloc[B_index])
            distances.append(d)

    return(np.mean(distances))

def complete(A,B):
    distances = []

    for A_index in range(0, A.shape[0]):
        for B_index in range(0, B.shape[0]):
            d = np.linalg.norm(A.iloc[A_index]-B.iloc[B_index])
            distances.append(d)

    return(max(distances))
```

[58]:
```
# check if your functions are working

df = pd.read_csv("https://raw.githubusercontent.com/cmparlettpelleriti/
 ↪CPSC392ParlettPelleriti/master/Data/HAC1.csv")

dA = df.loc[df.cluster == "A"] # cluster A
dB = df.loc[df.cluster == "B"] # cluster B
dC = df.loc[df.cluster == "C"] # cluster C
```

[59]:
```
# print(complete(dA[["x","y"]], dB[["x","y"]]))
# print(average(dA[["x","y"]], dB[["x","y"]]))
# print(single(dA[["x","y"]], dB[["x","y"]]))
# if complete() is correct, this will print true
completePASS = abs(complete(dA[["x","y"]], dB[["x","y"]]) - 4.718047025872837)␣
 ↪<= 0.01
print("Complete:", completePASS)

# if average() is correct, this will print true
averagePASS = abs(average(dA[["x","y"]], dB[["x","y"]]) - 2.734811240314461) <=␣
 ↪0.01
print("Average:", averagePASS)
```

```
# if single() is correct, this will print true
singlePASS = abs(single(dA[["x","y"]], dB[["x","y"]]) - 0.7361237342164363) <=␣
 ↪0.01
print("Single:", singlePASS)
```

```
Complete: True
Average: True
Single: True
```

Using the dataset `df` below,

1. plot the clusters using ggplot, color by cluster
2. use your functions `single()`, `average()`, and `complete()` to calculate the distances between each pair of clusters.

2.0.2 *Question*

3. Look at which clusters are considered "close" and "far" in different methods. Are there differences between which are furthest/closest between methods? What are they?

4. Describe why you think you see these differences.

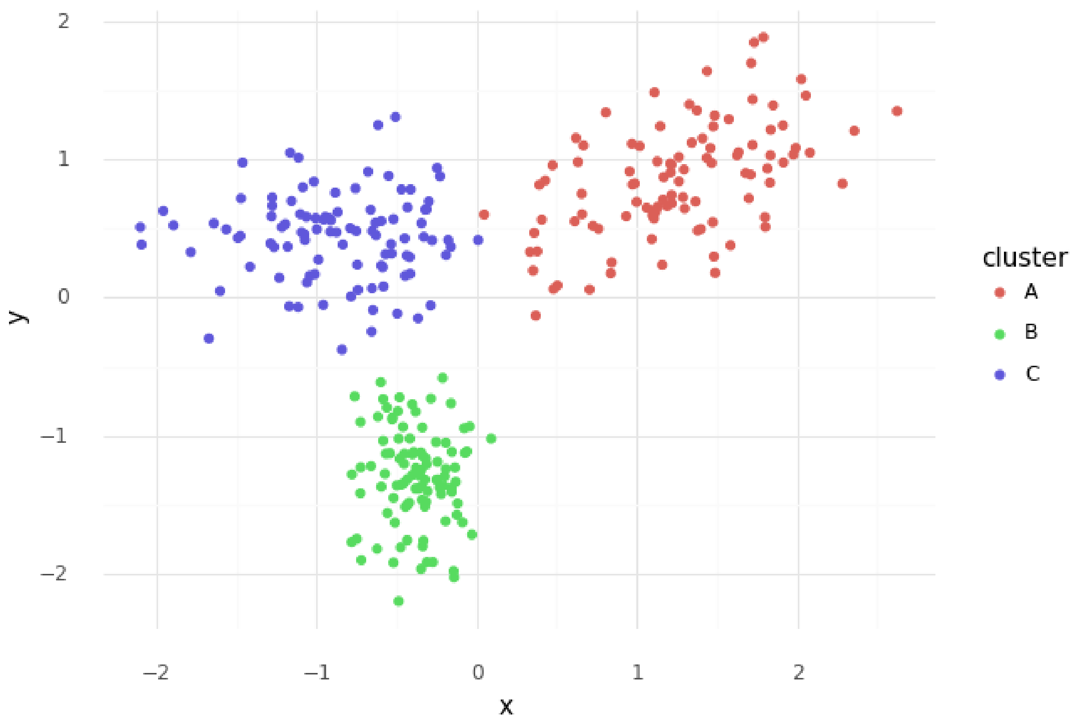
2.0.3 Answer

(example) The red cluster (A) is pretty spread out, so when you use methods like single linkage, Cluster C (blue) is closest to one end of cluster A. But the other end of cluster A is far away, so with complete linkage, Clusters A and C are furthest away from each other.

Average linkage is inbetween the two because while some points in cluster A are close to some points in cluster C, on average they're only moderately close.

The differences appear because the different linkage critera change WHICH points in each of the two clusters are influencing the measure of "closeness"

[67]:
```
# plot
ggplot(df, aes(x = "x", y = "y", color = "cluster")) + geom_point() +␣
 ↪theme_minimal()
```



```
[67]: <ggplot: (8795207390139)>
```

```
[61]: # calculate distances (this will likely take a few min to run)

      #---single------
      s_AB = single(dA[["x","y"]],dB[["x","y"]])
      s_AC = single(dA[["x","y"]],dC[["x","y"]])
      s_BC = single(dC[["x","y"]],dB[["x","y"]])

      print("AB:", s_AB)
      print("AC:", s_AC)
      print("BC:", s_BC)
      print("\n")
      #---average-----
      a_AB = average(dA[["x","y"]],dB[["x","y"]])
      a_AC = average(dA[["x","y"]],dC[["x","y"]])
      a_BC = average(dC[["x","y"]],dB[["x","y"]])

      print("AB:", a_AB)
      print("AC:", a_AC)
      print("BC:", a_BC)
      print("\n")
      #---complete----
```

```python
c_AB = complete(dA[["x","y"]],dB[["x","y"]])
c_AC = complete(dA[["x","y"]],dC[["x","y"]])
c_BC = complete(dC[["x","y"]],dB[["x","y"]])

print("AB:", c_AB)
print("AC:", c_AC)
print("BC:", c_BC)
print("\n")
```

```
AB: 0.7361237342164363
AC: 0.18809268564971213
BC: 0.33781439410788666


AB: 2.734811240314461
AC: 2.22451006115341
BC: 1.8723507863969513


AB: 4.718047025872837
AC: 4.81659588972283
BC: 3.5029348343614792
```

3 2. Exploring Linkage Critera

Using the predictors listed in `predictors`, perform HAC on the burger king dataset using sklearn and the following linkage critera:

- single linkage
- average linkage
- complete linkage
- ward's method

3.0.1 *Question*

Plot and compare the different clusters/dendrograms that you get. What do you notice is similar/different?

Think hard: what do the dendrograms tell you about the data?

(NOTE: see documentation if you need a refresher on how to set linkage criteria in sklearn, and here for how to set it for the dendrogram)

3.0.2 Answer

(example)

The dendrograms are incredibly different. Using single linkage, a lot of the clusters tend to be closer together (shown by the high density of connections in the lower part of the dedrogram). Complete linkage on the other hand gives you a dendrogram with a lot of density in the middle of the dendrogram (which suggests not so cohesive clusters). Average linkage (as the name suggests) Gives us a combination of Single and Complete linkages. There's still a lot of density towards the bottom, suggesting more cohesion, but you don't have as much separability (shown by longer distances in the top of the dendrogram).

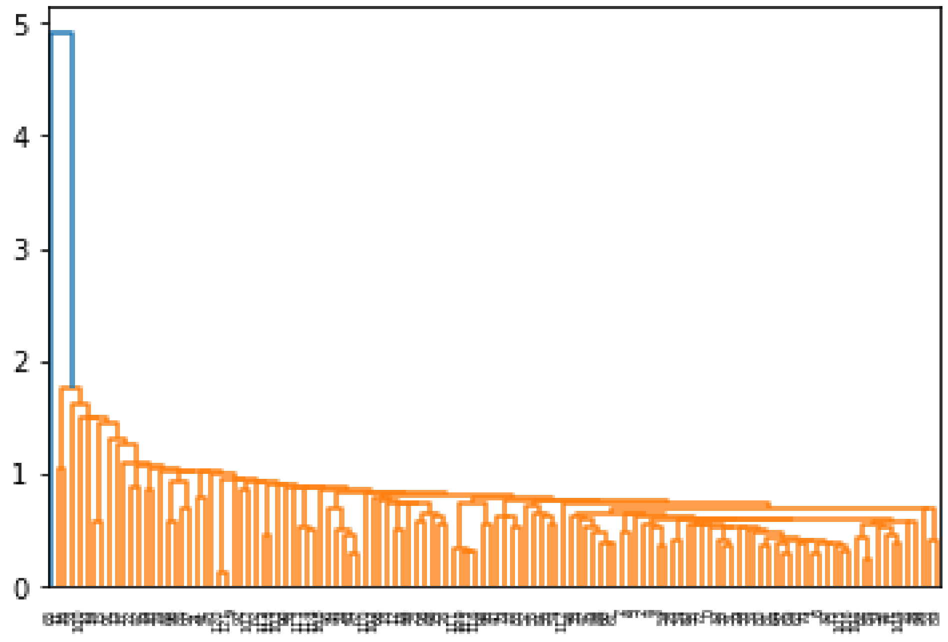
Ward's method has the nices looking dendrogram with high separability (long distances in the top of the dendrogram) and relatively good cohesion (short distances in the lower part of the dendrogram). While the orange cluster has a lot of variability, the Red and Green Clusters have pretty good cohesion.

The fact that single linkage gives you a lot of shorter distances compared to complete/average linkage suggests that edges of clusters might be close together while the rest of the cluster is quite spread out.

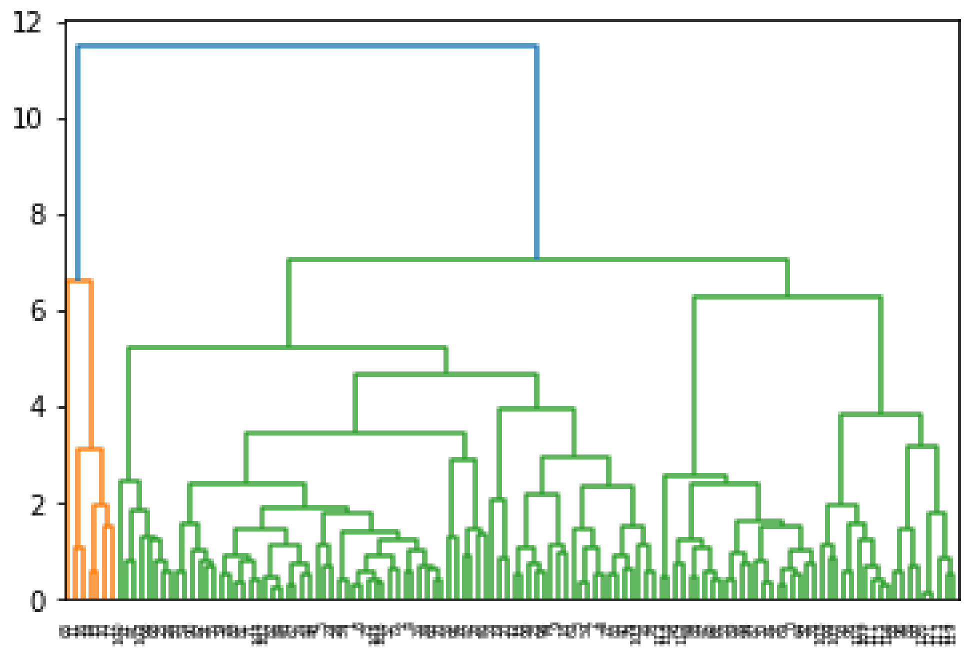
[62]:
```
burg = pd.read_csv("https://raw.githubusercontent.com/cmparlettpelleriti/
 ↪CPSC392ParlettPelleriti/master/Data/burger-king-items.txt",
                   sep = "\t")

predictors = ["Calories", "Protein(g)", "Fat(g)", "Sodium(mg)", "Carbs(g)",␣
 ↪"Sugar(g)"]
z = StandardScaler()
burg[predictors] = z.fit_transform(burg[predictors])

burg.head()
```

[62]:
```
                   Item  Serving.size  Calories  Fat.Cal  Protein(g)    Fat(g)  \
0             Hamburger         109.0 -0.835829     90.0   -0.370060 -0.919796
1          Cheeseburger         121.0 -0.661780    130.0   -0.145073 -0.670844
2       Double_Hamburger        146.0 -0.400706    160.0    0.304900 -0.421892
3   Double_Cheeseburger         171.0 -0.009095    230.0    0.604882  0.076012
4           Buck_Double         158.0 -0.183144    200.0    0.454891 -0.172940

   Sat.Fat(g)  Trans.fat(g)  Chol(mg)  Sodium(mg)  Carbs(g)  Fiber(g)  \
0         4.0           0.0        35   -0.801337 -0.580013       1.0
1         6.0           0.0        45   -0.390175 -0.580013       1.0
2         8.0           0.0        70   -0.745269 -0.580013       1.0
3        12.0           1.0        95    0.077055 -0.528996       1.0
4        10.0           0.5        85   -0.334107 -0.580013       1.0

   Sugar(g)  Meat  Breakfast  Not Breakfast  CarbsxMeat
0 -0.326804     1          0              1          28
1 -0.326804     1          0              1          28
2 -0.326804     1          0              1          28
3 -0.326804     1          0              1          29
4 -0.326804     1          0              1          28
```

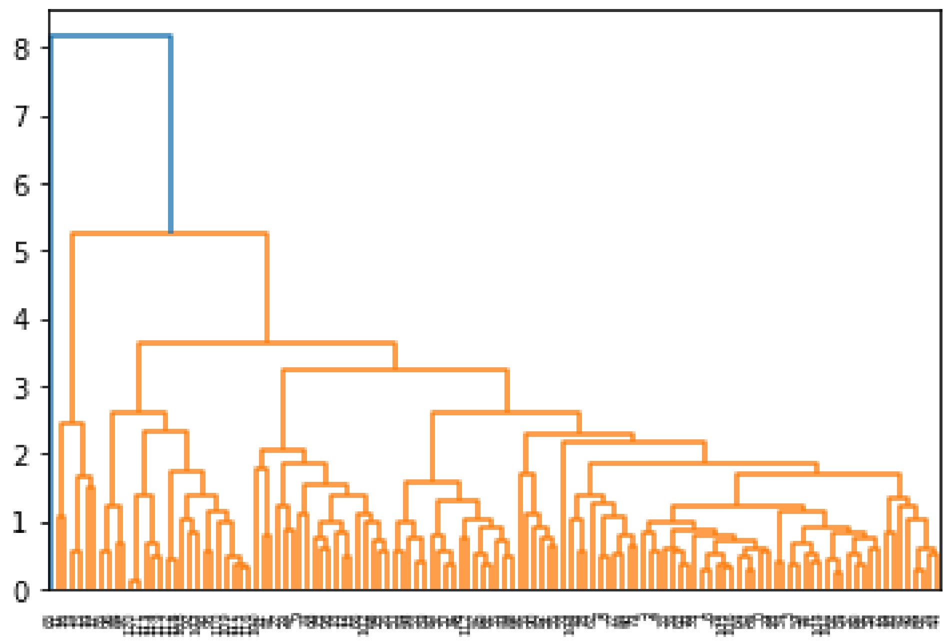
[63]:
```python
### YOUR CODE HERE ###

singleBK = AgglomerativeClustering(n_clusters = 3,
                                   affinity = "euclidean",
                                   linkage = "single")
singleBK.fit(burg[predictors])

dendro = sch.dendrogram(sch.linkage(burg[predictors], method='single'))
### \YOUR CODE HERE ###
```



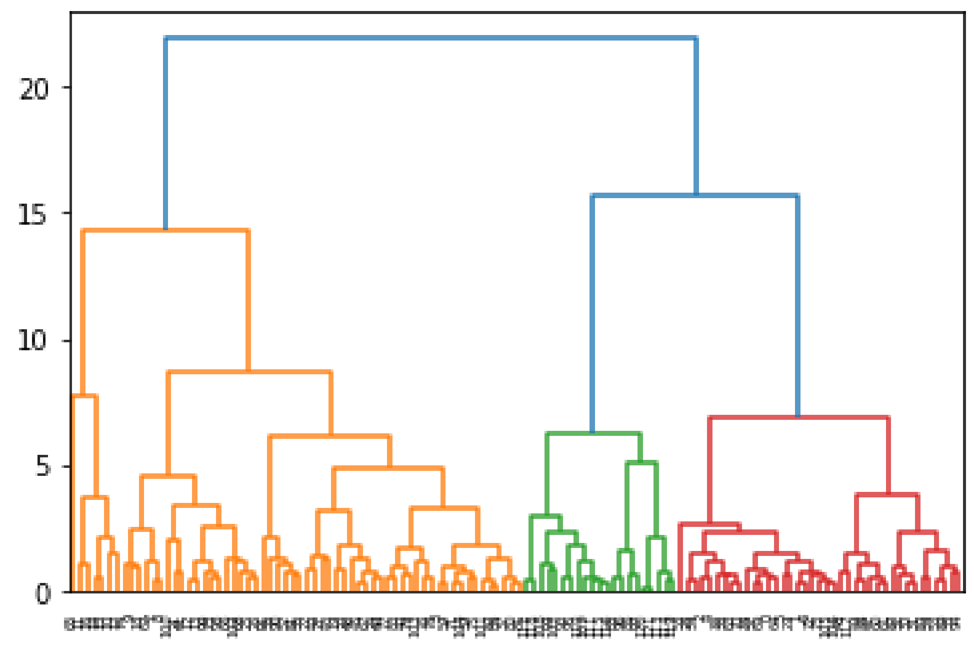
[64]:
```python
### YOUR CODE HERE ###
completeBK = AgglomerativeClustering(n_clusters = 3,
                                   affinity = "euclidean",
                                   linkage = "complete")
completeBK.fit(burg[predictors])

dendro = sch.dendrogram(sch.linkage(burg[predictors], method='complete'))
### \YOUR CODE HERE ###
```



[65]:
```
### YOUR CODE HERE ###

averageBK = AgglomerativeClustering(n_clusters = 3,
                                  affinity = "euclidean",
                                  linkage = "average")
averageBK.fit(burg[predictors])

dendro = sch.dendrogram(sch.linkage(burg[predictors], method='average'))
### \YOUR CODE HERE ###
```



```
[66]: ### YOUR CODE HERE ###

      wardsBK = AgglomerativeClustering(n_clusters = 3,
                                        affinity = "euclidean",
                                        linkage = "ward")
      wardsBK.fit(burg[predictors])

      dendro = sch.dendrogram(sch.linkage(burg[predictors], method='ward'))
      ### \YOUR CODE HERE ###
```


0
5
10
15
20