20210927 Hierarchical Clustering

Monday, September 27, 2021 2:41 PM

- BU Data Science Association two members from the club:
  - Workshop, research projects, competitions, and talks
  - Provide resources including resumes, networking with Prof and Students
  - Community of data science students
  - budatasci.com
  - Intro to python, intro to pandas, intro to matplotlib
  - Benefit from being involved, even as a mentor
  - SLACK CHANNEL

- Start homework 1 ASAP!!!!
  - Last questions run time will be considerable
  - In the future: requires notebooks
  - For HW1 just highly encouraged
  - When you run the notebook, you have the option of saving the output
    - Please do so
  - Last question will output an image, which takes a long time to run, so please run it before submitting it
- Ensure you compile and save the notebook with the results
- Homework 0 is almost graded
  - Noticed that a lot of you didn't think about what

we were asking
    - He was surprised
    - A lot of you missed the PURPOSE of the questions
    - You need to think about the data and why you are doing it
    - Replacing NAN values with row mean
        - Makes no sense if attributes are in COLUMNS
        - Want to replace with the mean age, income, etc. if it's missing
- Labwork
    - He took a look at some of the PRs in the lab
        - Some of you really did the minimum
        - Which is OK, but there is a purpose to these labs
        - Try to get the skills you need out of this class
- Lecture slides uploaded
- He uploaded some code to look at too
- Look at dbscan function
    - You need to implement it yourself
    - We will do it together next class
    - So if you are bored during lecture, try implementing it

Hierarchical Clustering

- Last time:
  - k-means and Lloyd's algorithm
  - Also discussed how to pick random numbers proportional to distances

Hierarchical Clustering

Two main types of hierarchical clustering:

Agglomerative:

1. Start with every point in its own cluster
2. At each step, merge the two closest clusters
3. Stop when every point is in the same cluster

Divisive:

1. Start with every point in the same cluster
2. At each step, split until every point is in its own cluster

---

- Two main ways
  - Agglomerative
    - Simpler, what we will mainly look at
  - Divisive

Hierarchical Clustering

Our main focus will be on **agglomerative** methods

Agglomerative Clustering Algorithm

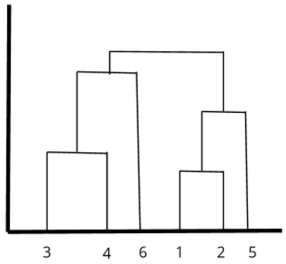
1. Let each point in the dataset be in its own cluster
2. Compute the distance between all pairs of clusters
3. Merge the two closest clusters
4. Repeat 3 & 4 until all points are in the same cluster

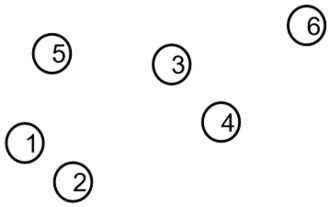
- Once you merge clusters, you will have one less cluster
- Repeat 2 & 3 (typo)
- Missing info:
  - What haven't we defined?
    - We defined distances between POINTS not CLUSTERS
      - Examples:
        - Compute center of the cluster, and do distances between centers
          - Means of the cluster
        - Sum of distances of all possible pairs between one cluster and another cluster

- Shortest path: the two points that are closest between two clusters
  - All pairwise distances between points, and take minimum
    - Conversely, take maximum

Hierarchical Clustering

At every step, we record which clusters were merged in order to produce a dendrogram:

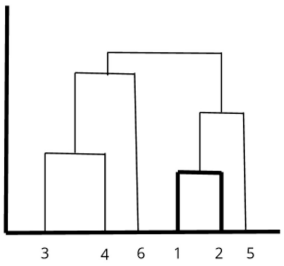


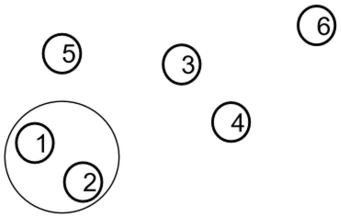


- Output: some sort of dendrogram with a hierarchy of clusters
  - Read from bottom to top
  - Based on distance of y-axis, you can tell which points merge at certain thresholds
  - 1 and 2 merge first

Hierarchical Clustering

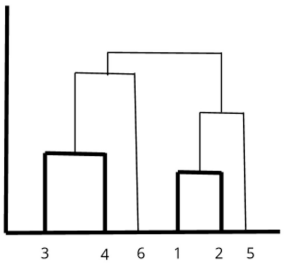
At every step, we record which clusters were merged in order to produce a dendrogram:

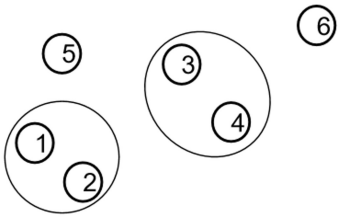




Hierarchical Clustering

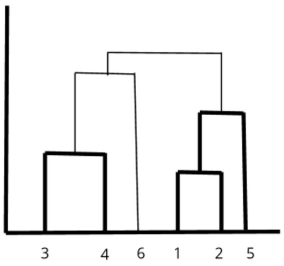
At every step, we record which clusters were merged in order to produce a dendrogram:

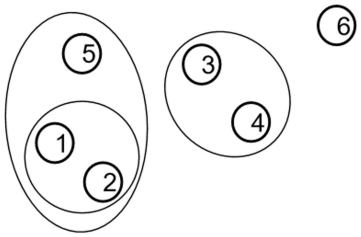




Hierarchical Clustering

At every step, we record which clusters were merged in order to produce a dendrogram:

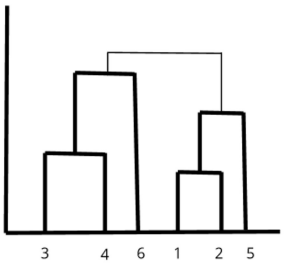


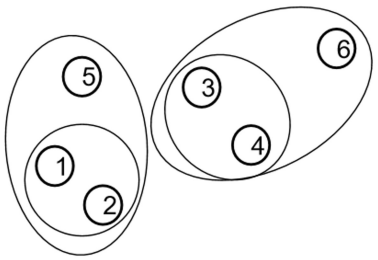


- Application:
  - Analyzing the DNA of different species and relate the species to each other

Hierarchical Clustering

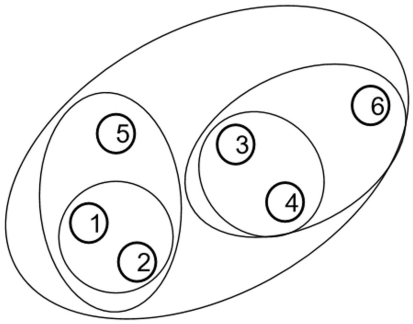
At every step, we record which clusters were merged in order to produce a dendrogram:





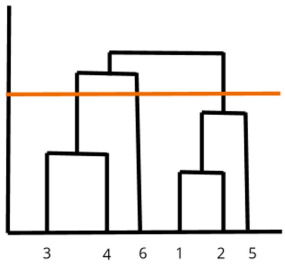
Hierarchical Clustering

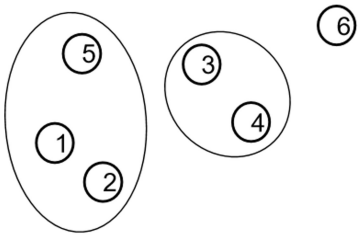
At every step, we record which clusters were merged in order to produce a dendrogram:



Hierarchical Clustering

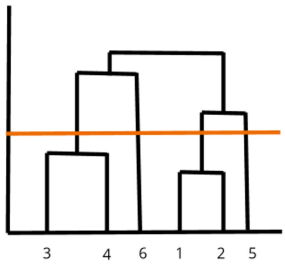
We can “cut” the dendrogram at any threshold to produce any number of clusters

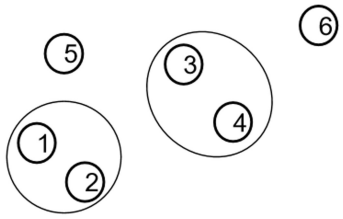




Hierarchical Clustering

We can “cut” the dendrogram at any threshold to produce any number of clusters

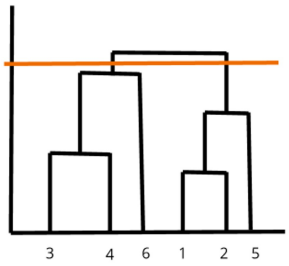


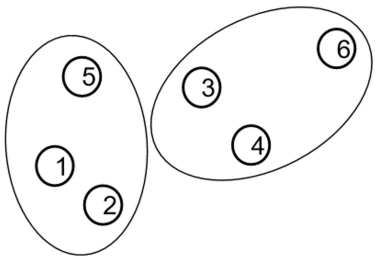


- Depending on where we cut the dendrogram (threshold), you get different number of clusters

Hierarchical Clustering

We can “cut” the dendrogram at any threshold to produce any number of clusters





Hierarchical Clustering

Can we implement this? Are we missing anything?

How do we compute the distance between clusters?

Distance between clusters can be thought of as distance between two sets of points. What ideas come to mind?

Hierarchical Clustering - Distance Functions

Let's first define:

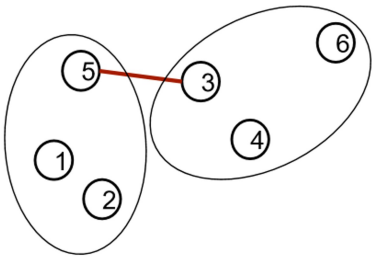
Distance between points: $\mathbf{d(p_1, p_2)}$

Distance between clusters: $\mathbf{D(C_1, C_2)}$

Single-Link Distance

Is the **minimum** of all pairwise distances between a point from one cluster and a point from the other cluster.

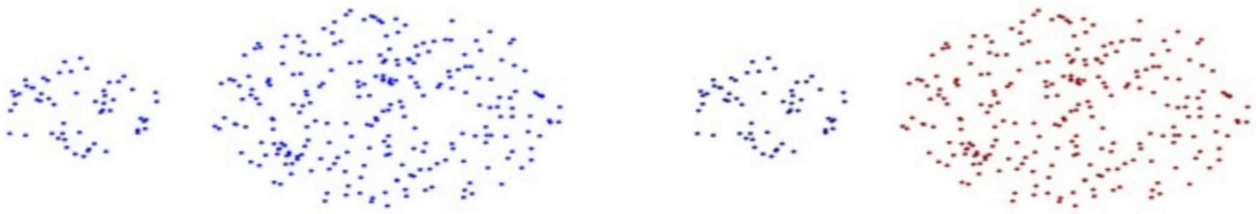
$$D_{SL}(C_1, C_2) = \min \{d(p_1, p_2) \mid p_1 \in C_1, p_2 \in C_2\}$$



Depends on choice of **d**

- Lowercase d is distance between points, uppercase D is distance between clusters
- Minimum of the Set of distances between p1 and p2 such that p1 is in C1 and p2 is in C2

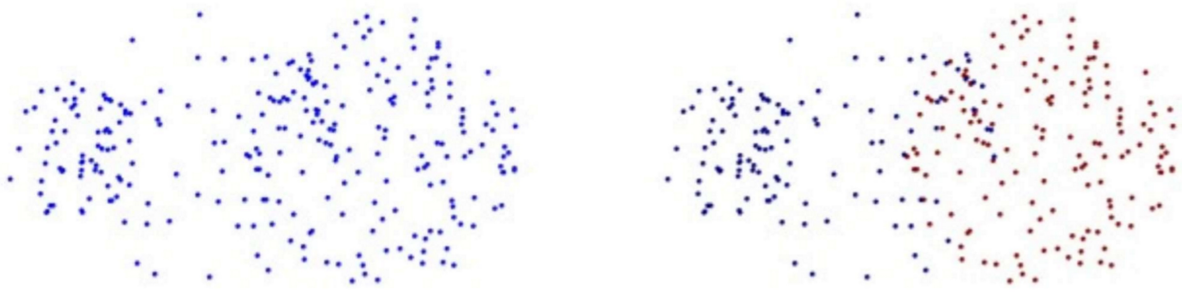
Single-Link Distance



Can handle clusters of different sizes

- Advantage:
  - Can handle clusters of different sizes
    - Unlike k-means

Single-Link Distance



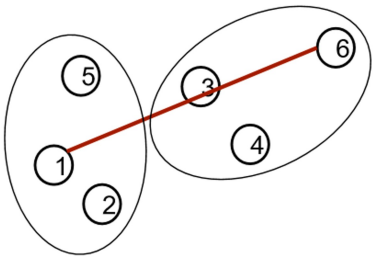
But... Sensitive to noise points
Tends to create elongated clusters

- If there is overlap between clusters, it gets jumbled up
  - Sensitive to noise points

Complete-Link Distance

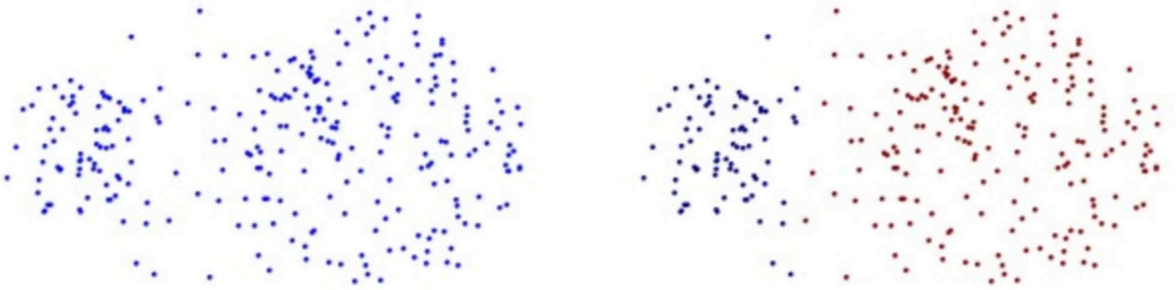
Is the **maximum** of all pairwise distances between a point from one cluster and a point from the other cluster.

$$D_{CL}(C_1, C_2) = \max \{d(p_1, p_2) \mid p_1 \in C_1, p_2 \in C_2\}$$



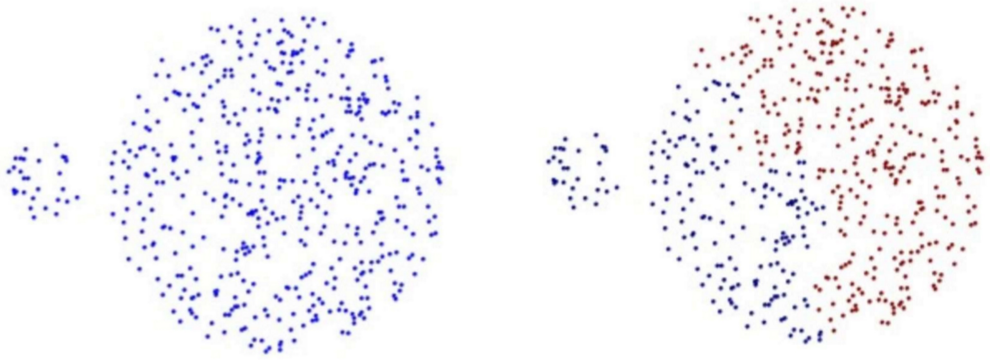
- Max of the set of all pairwise distances with …
- Also very sensitive to what "d" distance function chosen

Complete-Link Distance



Less susceptible to noise
Creates more balanced (equal diameter) clusters

Complete-Link Distance



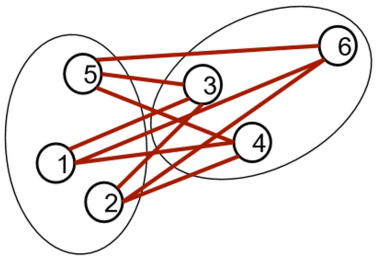
But... Tends to split up large clusters.
All clusters tend to have the same diameter

- Similar issue to kmeans of splitting up larger cluster

Average-Link Distance

Is the **average** of all pairwise distances between a point from one cluster and a point from the other cluster.

$$D_{AL}(C_1, C_2) = \frac{1}{|C_1| \cdot |C_2|} \sum_{p_1 \in C_1, p_2 \in C_2} d(p_1, p_2)$$



- Weighted by the number of points
- 1 over the cardinality of C1 and the cardinality of C2
  - Size of C1 times Size of C2:
    - Why times? because you want the pairwise distances
    - You have the C1 times C2 distances to compute
- All pairwise distances and compute average

Average-Link Distance

Less susceptible to noise and outliers.

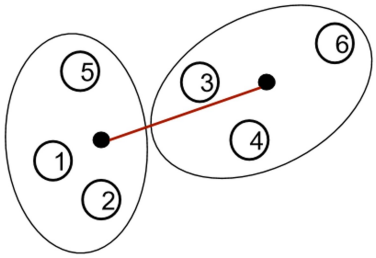
But... Tends to be biased toward globular clusters

-

Centroid Distance

The distance between the centroids of clusters.

$$D_C(C_1, C_2) = d(\mu_1, \mu_2)$$

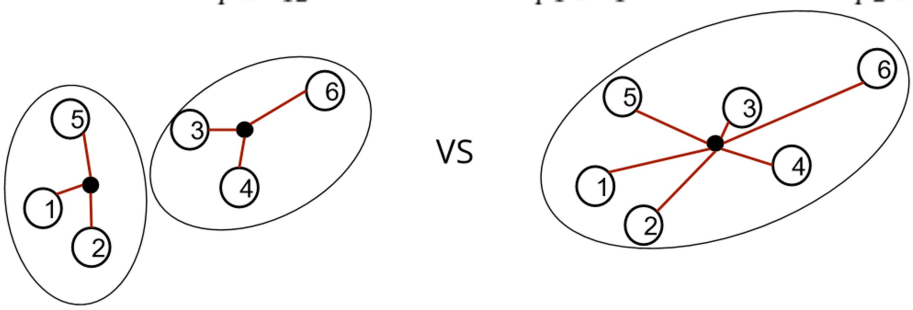


- mu1 centroid of cluster 1

Ward's Distance

Is the difference between the spread / variance of points in the merged cluster and the unmerged clusters.

$$D_{WD}(C_1, C_2) = \sum_{p \in C_{12}} d(p, \mu_{12}) - \sum_{p_1 \in C_1} d(p_1, \mu_1) - \sum_{p_2 \in C_2} d(p_2, \mu_2)$$



- Tuning parameter
  - Bit tricky
- Intuition nice
- You are trying to understand the variance
  - Not exactly variance, but similar
- If variance is really high when the two clusters are joined, then they probably shouldn't be merged
- If variance is small, you aren't losing much by merging them
- Can this be negative?
  - Maybe prove this for yourself for next time…
  - Convince yourself this is a distance function

Agglomerative Clustering Algorithm

1. Let each point in the dataset be in its own cluster
2. Compute the distance between all pairs of clusters
3. Merge the two closest clusters
4. Repeat 3 & 4 until all points are in the same cluster

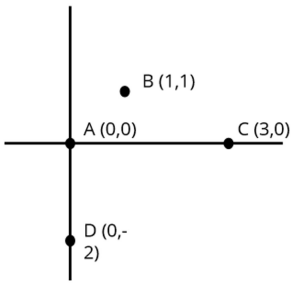
- Once you compare the pairwise distances between all clusters, then do merge, then recalculate
  - But if you track this, only need to recalculate distance for the NEW cluster, not the relationship between the un-merged clusters

Example

d = Euclidean
D = Single-Link

Distance Matrix

D = Single-Link



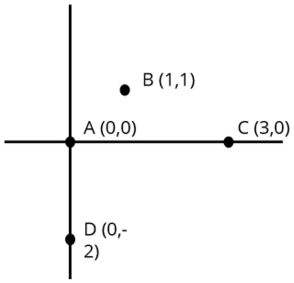
Distance Matrix

	A	B	C	D
A				
B				
C				
D				

- Keep track of these distances
  - In data structure, and populate it as you go

Example

d = Euclidean
D = Single-Link

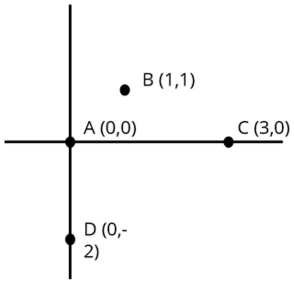


Distance Matrix

	A	B	C	D
A	0			
B		0		
C			0	
D				0

Example

d = Euclidean
D = Single-Link

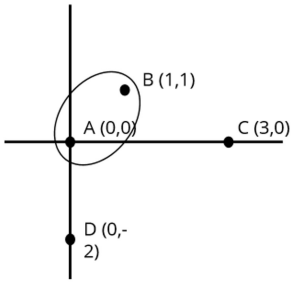


Distance Matrix

	A	B	C	D
A	0	√2	3	2
B	√2	0	√5	√10
C	5	√5	0	√13
D	2	√10	√13	0

Example

d = Euclidean
D = Single-Link



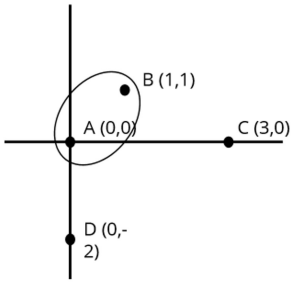
Distance Matrix

	A	B	C	D
A	0	√2	3	2
B	√2	0	√5	√10
C	5	√5	0	√13
D	2	√10	√13	0

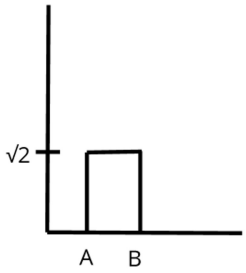
- Merge A and B first

Example

d = Euclidean
D = Single-Link



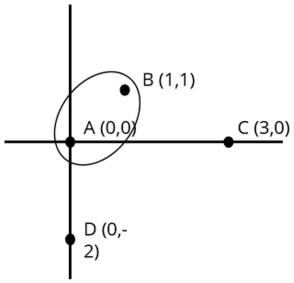
Dendrogram



- Really want to keep track of this in a dendrogram

Example

d = Euclidean
D = Single-Link



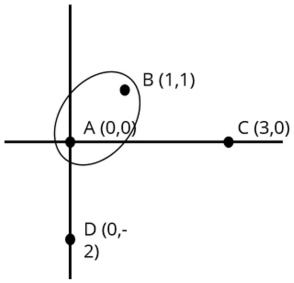
Distance Matrix

	A & B	C	D
A & B	0		
C		0	√13
D		√13	0

- Distances between C and D are unchanged, and don't need to be updated

Example

d = Euclidean
D = Single-Link



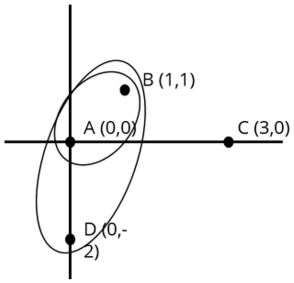
Distance Matrix

	A & B	C	D
A & B	0	√5	2
C	√5	0	√13
D	2	√13	0

- Doing the minimum of the distances because we are doing single-link
- Merge: A&B and D next

Example

d = Euclidean
D = Single-Link

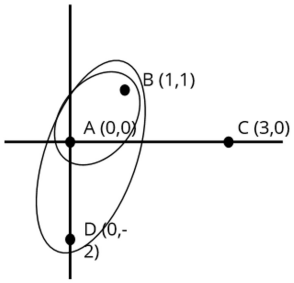


Distance Matrix

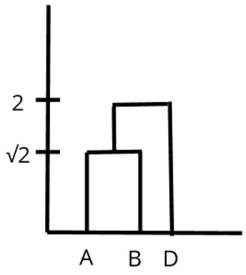
	A & B	C	D
A & B	0	√5	2
C	√5	0	√13
D	2	√13	0

Example

d = Euclidean
D = Single-Link

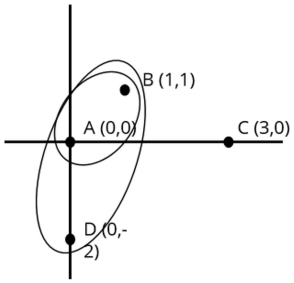


Dendrogram



Example

d = Euclidean
D = Single-Link

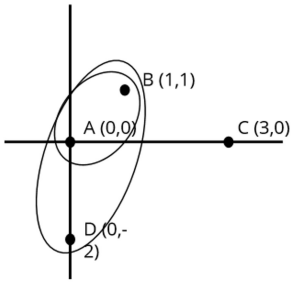


Distance Matrix

	A & B & D	C
A & B & D	0	
C		0

Example

d = Euclidean
D = Single-Link



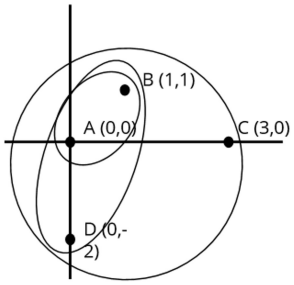
Distance Matrix

	A & B & D	C
A & B & D	0	√5
C	√5	0

- Still square root of 5

Example

d = Euclidean
D = Single-Link

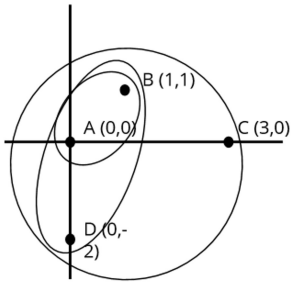


Distance Matrix

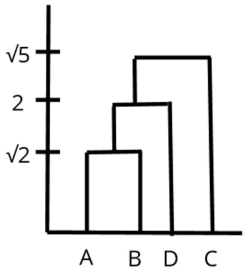
	A & B & D	C
A & B & D	0	√5
C	√5	0

Example

d = Euclidean
D = Single-Link



Dendrogram



- Everything in one cluster so we're done

Hierarchical Clustering

Finding the threshold with which to cut the dendrogram requires exploration and tuning. But in general hierarchical clustering is used to expose a hierarchy in the data (ex: finding/defining species via DNA similarity).

To capture the difference between clusterings you can use a cost function, or methods that we will discuss later when we look at clustering aggregation.

- Challenges:
  - The threshold to use is not clear
  - Can't see the full dendogram if there are too many points
- Cost function for k-means can't be used for hierarchical clustering… ??
- Ask yourself: do you have enough information to implement this yet?
  - Ask if you need more info

20210927 DB-Scan; Density-Based Clustering

Monday, September 27, 2021 2:43 PM

Density-Based Clustering

Density-Based Clustering

Goal: cluster together points that are densely packed together.

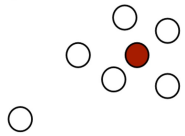
How should we define density?

Given a fixed radius **ε** around a point, if there are at least **min_pts** number of points in that area, then this section is dense.

- They need to be close AND have a lot of points in this small location
  - Within this specific threshold of distances, is there above a certain number of points?
  - Ratio between the number of points and the area they take up
    - There is a simpler way to do this
- Given a fixed radius, if there are at least a certain number of points, then this section is dense

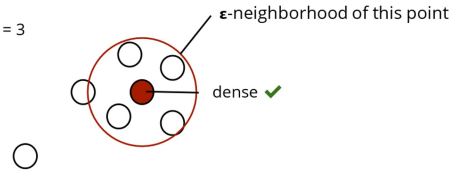
Example

Min_pts = 3



Example

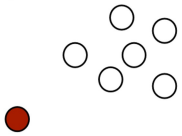
Min_pts = 3



- Epsilon neighborhood of this point
- Core point

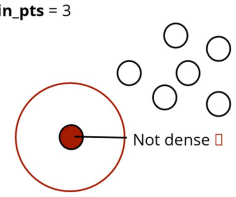
Example

Min_pts = 3



Example

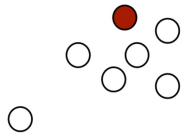
Min_pts = 3



- Noise point

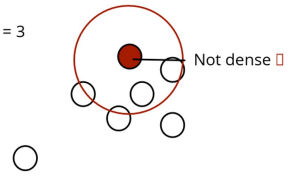
Example

Min_pts = 3



Example

Min_pts = 3



But... That point was part of a dense section earlier...

- This point isn't dense, but was in the other points dense region earlier
- This is a border point

Density-Based Clustering

We need to distinguish between points at the core of a dense region and points at the border of a dense region.

Let's define:

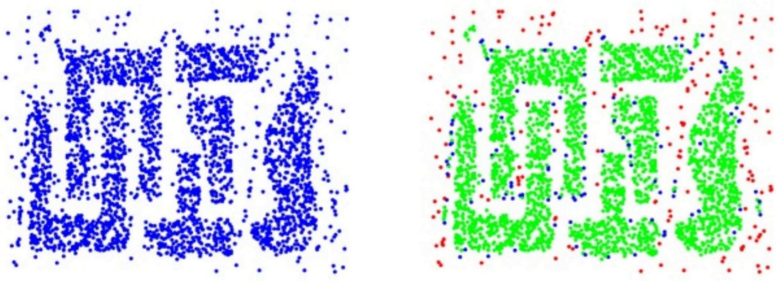
Core point: if its **ε**-neighborhood contains at least **min_pts**

Border point: if it is in the **ε**-neighborhood of a core point

Noise point: if it is neither a core nor border point

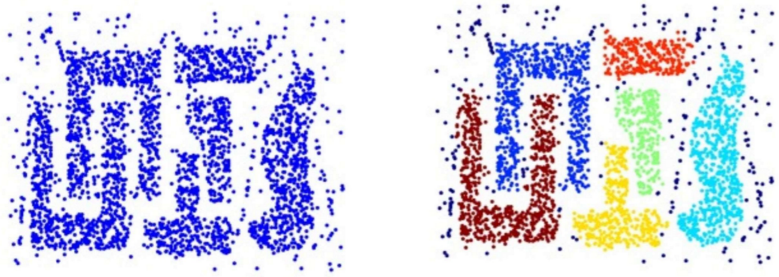
- DB Scan distinguish points that are core to a dense region and those that are a border point

Density-Based Clustering



Core | Border | Noise

Density-Based Clustering



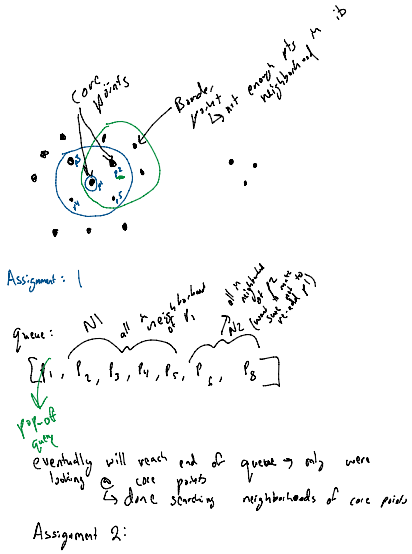
Create clusters by connecting core points

- K means would not be a good method for this type of example

DBScan Algorithm

ε and **min_pts** given:

1. Find the **ε**-neighborhood of each point
2. Label the point as **core** if it contains at least **min_pts**
3. Label points in its neighborhood that are not **core** as **border**
4. Label points as **noise** if they are neither **core** nor **border**
5. For each **core** point, assign to the same cluster all **core** points in its neighborhood
6. Assign border points to nearby clusters

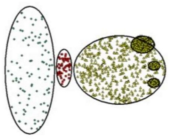


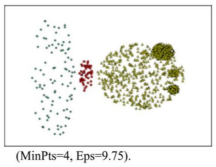
- We'll walk through it, and then figure out how to implement
- epsilon and min_pts are two parameters you can tune/optimize for your dataset (tinker with)
- Leave the noise points as they are
  - Normally don't associate a noise point with a label
- How would you do this?
  - Create a new column
  - For each points label it as core or border
  - Is it a core point? Which core points in its neighborhood? Rinse and repeat
  -

DBScan - Benefits

1. Can identify clusters of different shapes and sizes
2. Resistant to noise

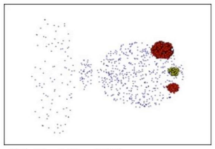
DBScan - Limitations





(MinPts=4, Eps=9.75).

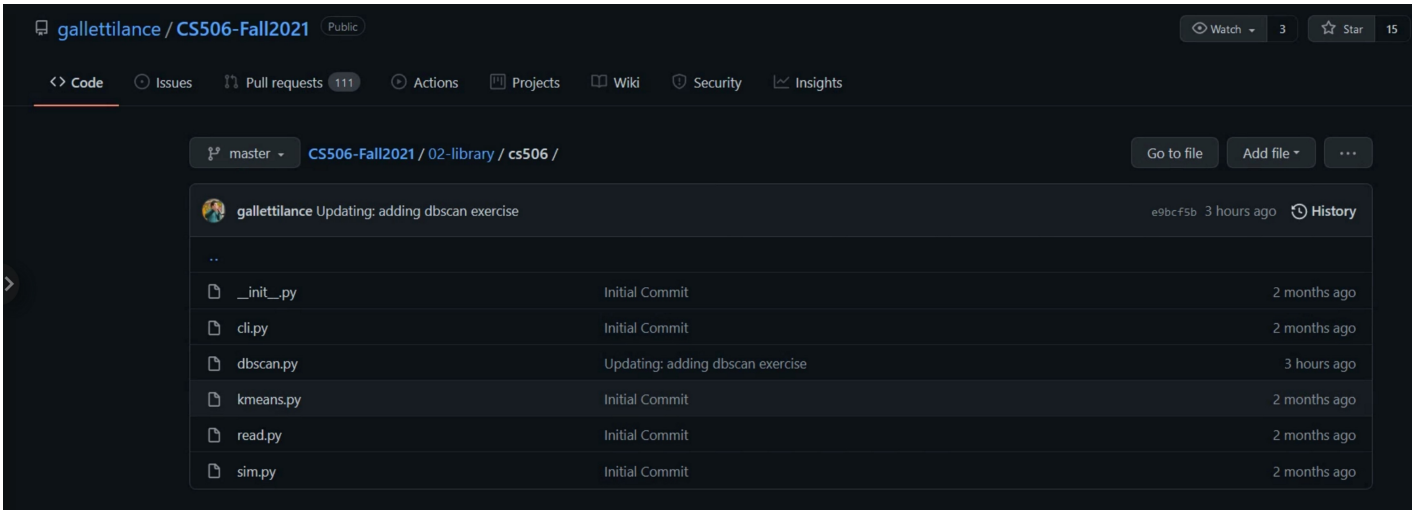
1. Can fail to identify clusters of varying densities.
2. Tends to create clusters of the same density.
3. Notion of density is problematic in high-dimensional spaces



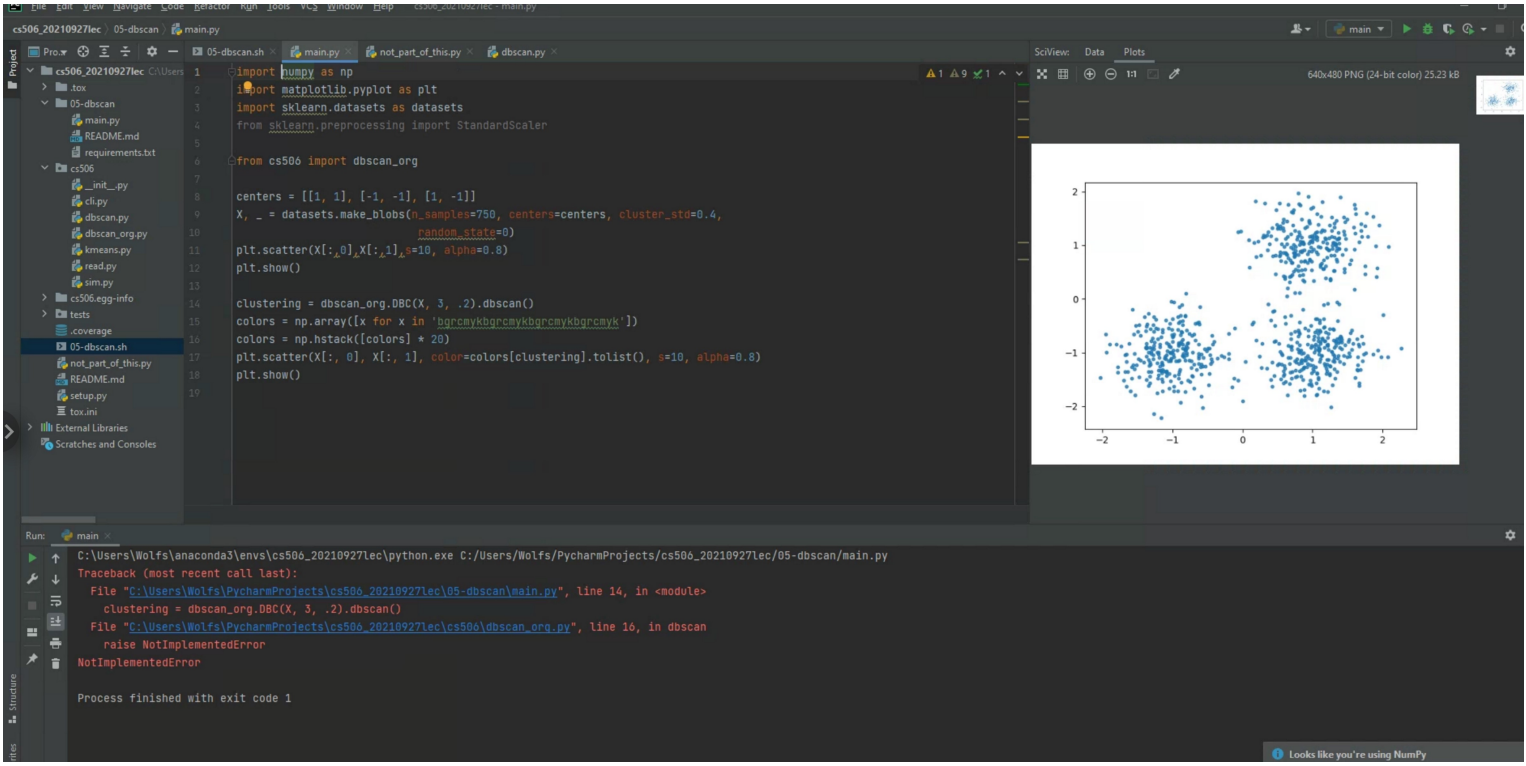
(MinPts=4, Eps=9.92)

- Varying densities is hard to optimize epsilon and MinPts
- Epsilon of only slight changes (9.745 vs 9.92) gave very different results
- Notion of density is problematic in high-dimensional space
  - e.g. pixels on your phones
    - Images lie in high dimensional space
    - In that many dimensions, takes a lot of points to population it
    - (Don't fully get this example)
  - If you have two points in two dimensions is NOT the same density of a million points in a million dimensions
    - As you increase in dimensions, you need more points to populate the space
  - e.g. Volume of sphere
    - Most of it is in the shell of the sphere

Demo


gallettilance / CS506-Fall2021
Public
Watch 3
Star 15
Code
Issues
Pull requests 111
Actions
Projects
Wiki
Security
Insights
master
CS506-Fall2021 / 02-library / cs506 /
Go to file
Add file
gallettilance Updating: adding dbscan exercise
e9bcf5b 3 hours ago
History
..
__init__.py Initial Commit 2 months ago
cli.py Initial Commit 2 months ago
dbscan.py Updating: adding dbscan exercise 3 hours ago
kmeans.py Initial Commit 2 months ago
read.py Initial Commit 2 months ago
sim.py Initial Commit 2 months ago

```
# Start coding the db-scan example

# As part of your lab this week, you will have to play around with this library
# Not published, so you can only pip install it locally

# Start by just walking through the code

# We have three centers, clusters we are generating
# Want to try to recover those 3 clusters

# Want to double check that everything in setup is currently working before starting to code further



# Start out by spinning out a virtual enviornment:
cat requirements.txt
pip install -r requirements.txt

# CS506 only available locally, so you will have to install in separte
cd 05-dbscan
python3 main.py

# Should be able to see a figure pop up
```



Incomplete, and I missed part of it:

File - C:\Users\Wolfs\PycharmProjects\cs506_20210927lec\cs506\dbscan.py

```python
# Always want to minimize debugging space

class DBC():

    def __init__(self, dataset, min_pts, epsilon):
        self.dataset = dataset
        self.min_pts = min_pts
        self.epsilon = epsilon

    def dbscan(self):
        """
            returns a list of assignments. The index of the
            assignment should match the index of the data point
            in the dataset.
        """
        # assignment vs assignments is not great naming
        assignments = [0 for _ in range(len(self.dataset))]
        assignment = 1
        # Iterate over points in dataset,
        # If you iterate over points themselves
        # You need to assignment of the point to match the index of the point
        # So this can get messy
        # So iterate over indexes instead
        # Or use enumerate to get index and the point
        for i, x in enumerate(self.dataset):
            # Need to determine whether or not X is a core point
            if self._get_epsilon_neighborhood(x) >= self.min_pts:
                # Core point
                # Explore x neighborhood for core points and give them the same assignment
                # Instead of doing that now, we are going to save:
                # Helper function
                assignments = self.
```

```
32 _explore_neighborhood(x, assignments, assignment)
33             # Without a core point, you can't really
   do anything
34             # Everything is initially labeled noise
35             # You know it is a border point if it's in
    a core point's neighborhood, but isn't a core point
36             # Will iterate over queue and pop off
37         return assignments
38
39 # Try to do on your own before next time
```