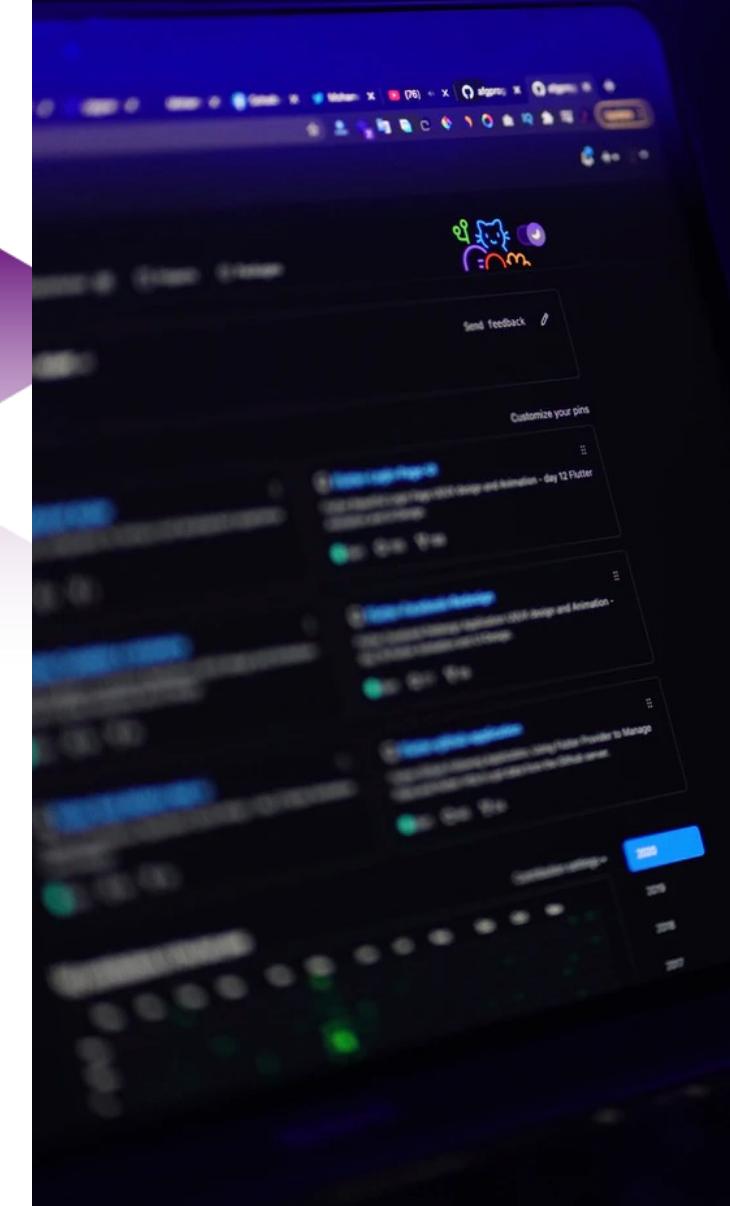


Machine Learning Part 2: Clustering

Louise Capener, Research Associate at the UK Data Service (Cathie Marsh Institute, UoM)



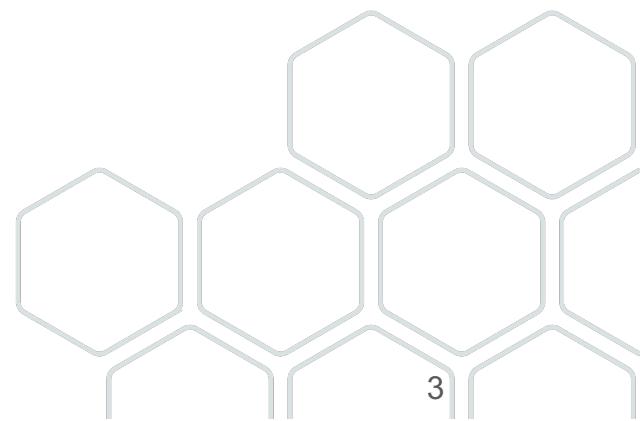
Interaction

- During this workshop you can use the Zoom chat for technical questions or comments (to chat with the facilitator – Nadia Kennar).
- If you want to ask questions in relation to the content of this talk, then you can use the Zoom Q+A function.



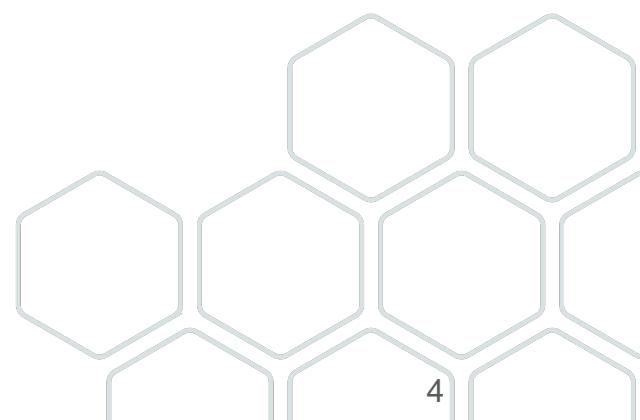
Can you hear us?

?



Troubleshooting audio problems

- Check your speaker/headset is plugged in / volume is on.
- Click on audio to change to listening via phone
- We are recording this webinar and will post it on YouTube
(<https://www.youtube.com/user/UKDATASERVICE>)
- This workshop is being run live on Youtube



Outline

- Recap
- What is clustering?
- Why bother with it?
- Types of clustering algorithms
- K-Means
- Hierarchical clustering

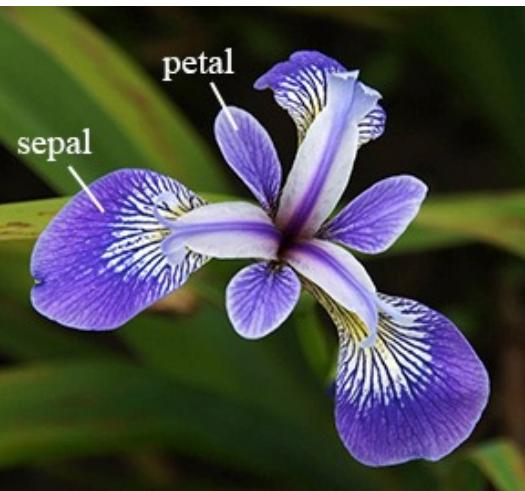


Recap

Supervised learning	Unsupervised learning
Input data is labelled	Input data is unlabelled
Data is classified based on the training dataset	Assigns properties of given data to classify it
Divided into Regression and Classification	Divided into Clustering and Association
Used for prediction	Used for analysis
Algorithms include: decision trees, logistic regressions, support vector machine	Algorithms include: k-means clustering, hierarchical clustering, apriori algorithm
A known number of classes	An unknown number of classes



Recap (contd.)

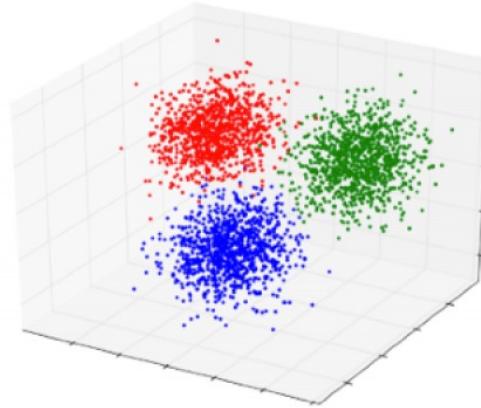


Supervised learning: used for prediction

Dps	Sepal length (cm)	Petal length (cm)	Petal width (cm)	Species
A	3.5	1.4	0.2	Iris-Versicolour
B	3.2	5.7	2.3	Iris-Setosa
C	3.2	5.9	2.3	Iris-Setosa
D	2.9	4.7	1.4	Iris-Virginica
E	3.7	1.5	0.4	Iris-Versicolour
F	3.1	5.5	2.2	?

Unsupervised learning: used for analysis

Dps	Sepal length (cm)	Petal length (cm)	Petal width (cm)
A	3.5	1.4	0.2
B	3.2	5.7	2.3
C	3.2	5.9	2.3
D	2.9	4.7	1.4
E	3.7	1.5	0.4

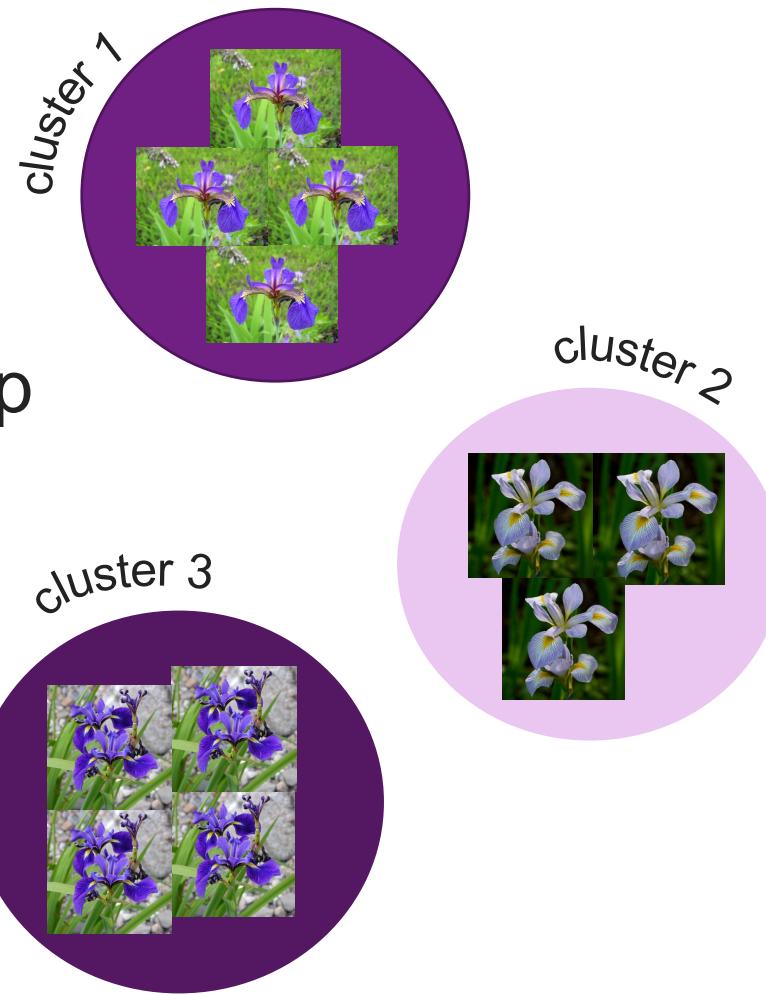


What is clustering?

“Clustering is the task of partitioning the dataset into groups, called clusters. The goal is to split up the data in such a way that points within a single cluster are very similar and points in different clusters are different.”

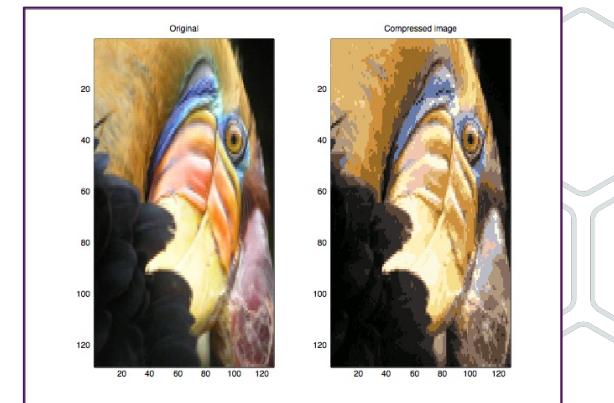
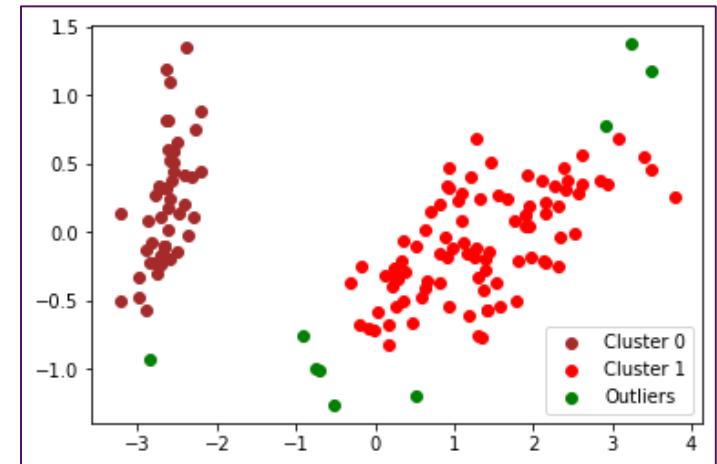
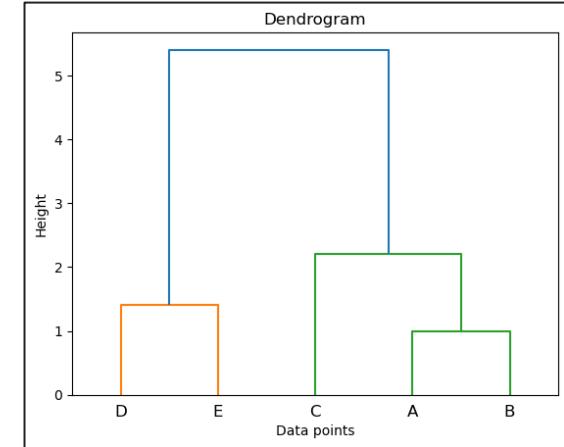
(Müller and Guido 2017)

Dps	Sepal length (cm)	Petal length (cm)	Petal width (cm)	cluster
A	3.5	1.4	0.2	1
B	3.2	5.7	2.3	2
C	3.2	5.9	2.3	2



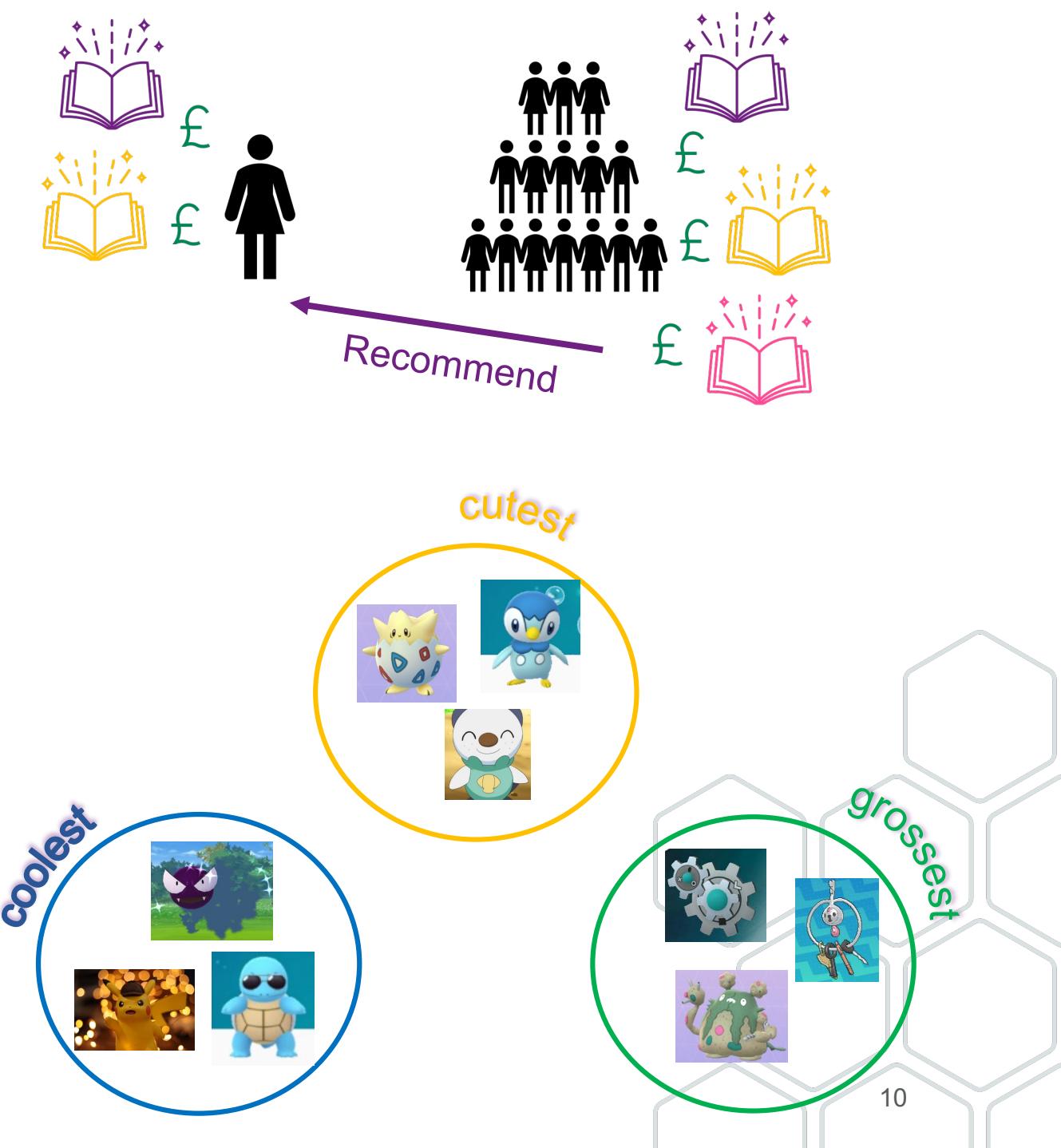
Why bother with it?

- It provides more information on the structure of the data → patterns
- It can help identify problems in the data, such as outliers
- It can be used to compress data



Other use cases

- Customer recommendation systems:
“People who bought *Harry Potter and the Philosopher’s Stone* also bought *The Hunger Games...*”
- Grouping DNA sequences of different strains of HIV into families of genetically similar viruses
- Identifying fake news by clustering the words used in articles. Certain words may appear more in sensationalized click-bait articles.
- And the more frivolous and fun side projects...



What is a cluster?

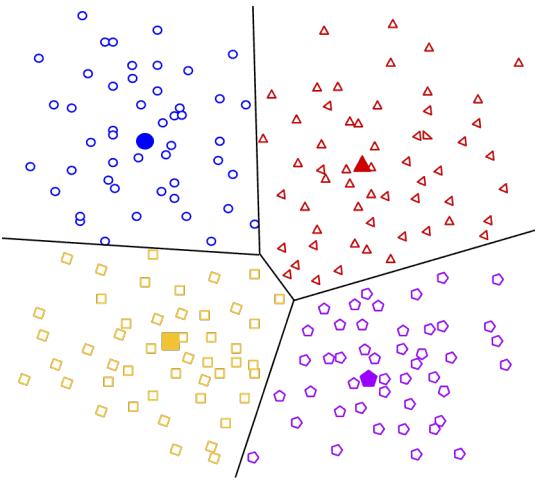
“There is no universal definition of what a cluster is: it really depends on the context, and different algorithms will capture different kinds of clusters.”

(Géron, 2019)

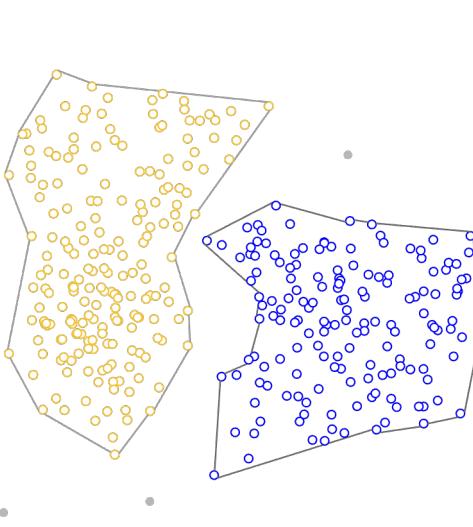


Types of clustering algorithms

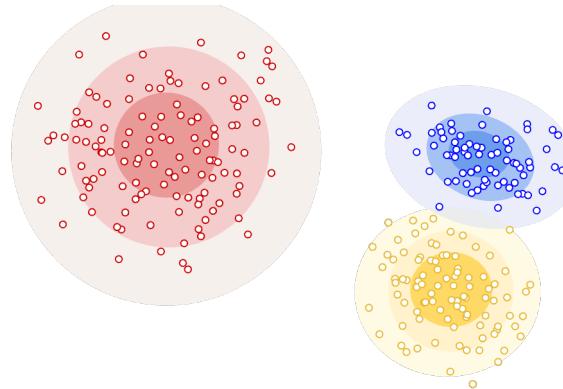
Centroid-based



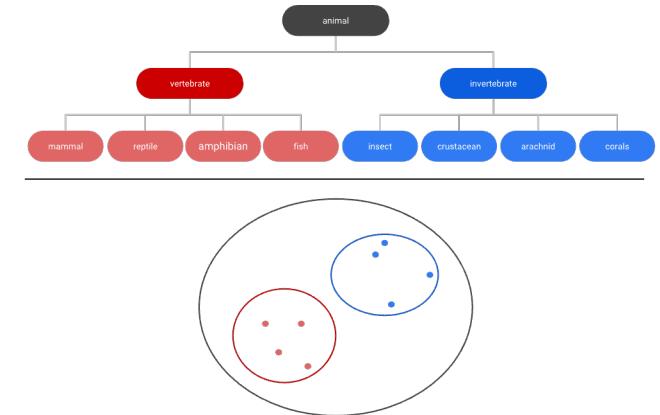
Density-based



Distribution-based



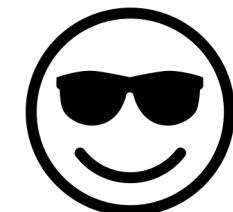
Hierarchical clustering



How do I know which type of algorithm is right for me?

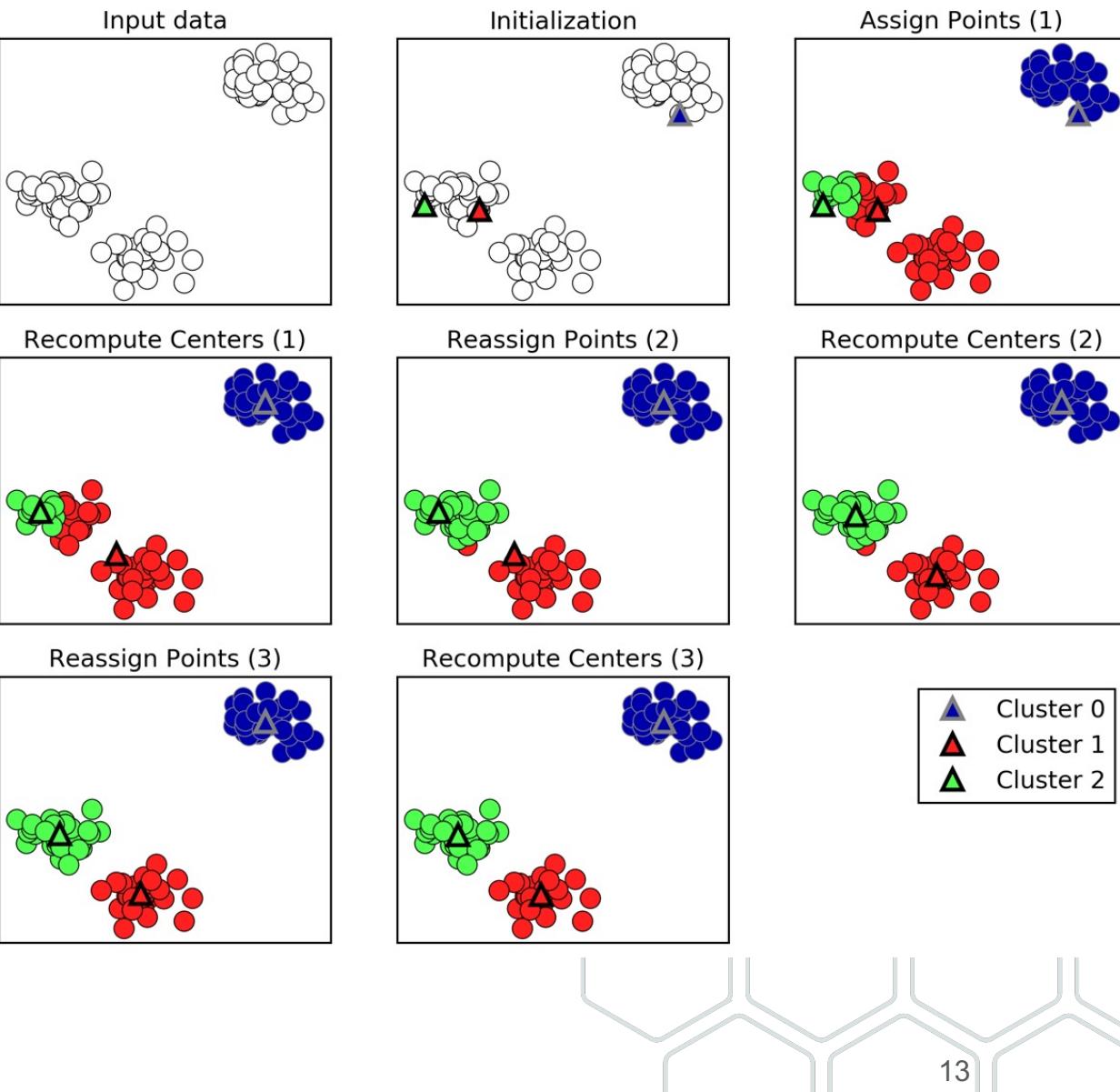


EXPLORE YOUR DATA



K-Means clustering

- We want to separate our data points into k clusters
- First, we initialize the algorithm with k random points (our centroids)
- Then, we assign each data point to its nearest initialisation point – using the Euclidean distance
- Once each data point is assigned, we relocate the initialisation point to the mean of the data points that were assigned to it
- Repeat the highlighted steps until the assignment of data points to centroids remains unchanged



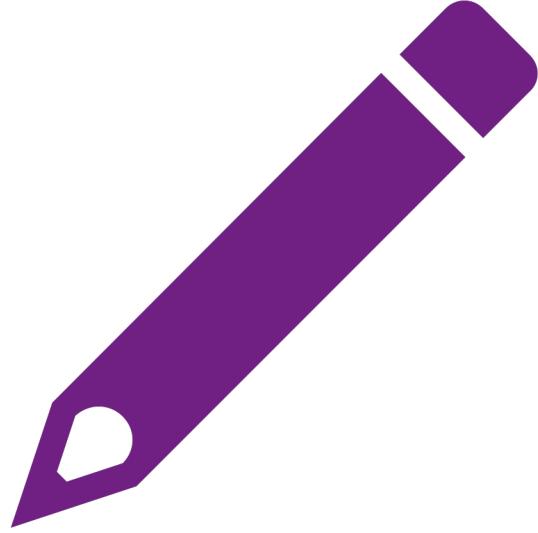
Introducing pseudocode...

K-MEANS(P, k)

Input: a dataset of points $P = \{p_1, \dots, p_n\}$, a number of clusters k

Output: centers $\{c_1, \dots, c_k\}$ implicitly dividing P into k clusters

```
1 choose  $k$  initial centers  $C = \{c_1, \dots, c_k\}$ 
2 while stopping criterion has not been met
3   do ▷ assignment step:
4     for  $i = 1, \dots, N$ 
5       do find closest center  $c_k \in C$  to instance  $p_i$ 
6         assign instance  $p_i$  to set  $C_k$ 
7   ▷ update step:
8   for  $i = 1, \dots, k$ 
9     do set  $c_i$  to be the center of mass of all points in  $C_i$ 
```



Pseudo English



Aim: Separate data points into k clusters

1. Initialise the algorithm with k random points (centroids)
2. Assign each data point to its nearest initialisation point, using the Euclidean distance
3. Relocate the initialisation point to the centre of its cluster (find the mean of the data points)
4. Repeat steps 2 and 3 until the assignment of data points to centroids remains unchanged



Pseudocode

K-MEANS(P, k)

Input: a dataset of points $P = \{p_1, \dots, p_n\}$, a number of clusters k

Output: centers $\{c_1, \dots, c_k\}$ implicitly dividing P into k clusters

```
1 choose  $k$  initial centers  $C = \{c_1, \dots, c_k\}$ 
2 while stopping criterion has not been met
3   do ▷ assignment step:
4     for  $i = 1, \dots, N$ 
5       do find closest center  $c_k \in C$  to instance  $p_i$ 
6         assign instance  $p_i$  to set  $C_k$ 
7   ▷ update step:
8   for  $i = 1, \dots, k$ 
9     do set  $c_i$  to be the center of mass of all points in  $C_i$ 
```



Code

```

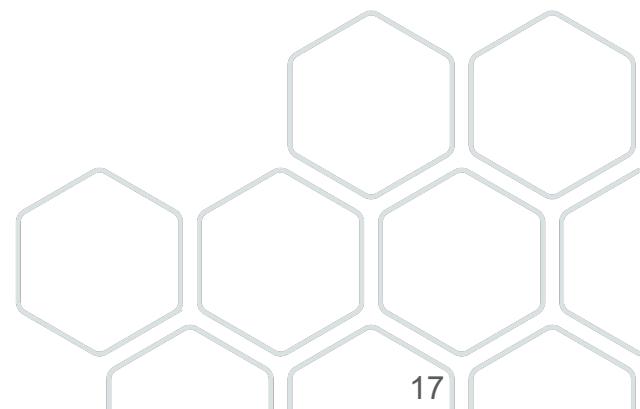
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import random
7
8 def load_data(fname):
9     features = []
10
11    with open(fname) as f:
12        for line in f:
13            p = line.strip().split(' ')
14            features.append(np.array(p[1::], dtype=float))
15    return np.array(features)
16
17 #This function returns the Euclidean distance squared between x and y
18 def distance(X, Y):
19     return np.linalg.norm(X-Y)**2
20
21 # normalize a vector in L2 norm
22 def l2_normalize(data):
23     return data / np.sqrt(np.sum(data ** 2))
24
25 """
26     loads all the required datasets
27     Input:
28         normalize - true/false - if normalization requires
29     Outputs:
30         dataset - concatenated datasets
31         target - list of all the individual data( used for evaluating performance)
32 """
33 def load_dataset(normalize = False):
34     # load the datasets from files
35     animals = load_data("animals")
36     countries = load_data("countries")
37     fruits = load_data("fruits")
38     veggies = load_data("veggies")
39
40     #normalize if required
41     if normalize:
42         animals = l2_normalize(animals)
43         countries = l2_normalize(countries)
44         fruits = l2_normalize(fruits)
45         veggies = l2_normalize(veggies)
46
47     # just a list of all datasets
48     targets = [animals, countries, fruits, veggies]
49
50     # concatenate all datasets into one
51     dataset = np.concatenate((animals, countries, fruits, veggies))
52     return dataset, targets
53
54 """
55     gets initial random centroids for the kmeans algo
56     Input:
57         k - num of centroids
58         dataset - the entire dataset
59     Output:
60         list of k centroid ( each is 300 long vector )
61 """
62 def get_random_centroids(k, dataset):
63     # Get Random centroids for k
64     cids = random.sample(range(0, len(dataset)), k)
65     centroids = []
66
67     # Then, the randomly chosen centroids are compiled into a list
68     for i in cids:
69         centroids.append(dataset[i])
70
71     print("cids ", cids)
72     # print("centroids ", centroids)
73     return centroids
74
75 """
76     Assian data points to clusters with given centroids

```

```

75 """
76     Assign data points to clusters with given centroids
77     Input:
78         dataset - the entre dataset
79         centroids - coordinates of k centroids
80     Output:
81         clusters - a list of cluster ids ( 0 to k-1 ) for all data points
82 """
83 def get_clusters(dataset, centroids):
84     clusters = []
85     # for every data point
86     for val in dataset:
87         min_dist = np.Inf
88         cluster_id = None
89         # find the closest centroid by checking the distance
90         # for every centroid
91         for cid, centroid in enumerate(centroids):
92             dist = distance(centroid, val)
93             if dist < min_dist:
94                 min_dist = dist
95                 cluster_id = cid
96
97         #record the closest centroid id as cluster id of this point
98         clusters.append(cluster_id)
99
100    #print("Clusters ", clusters)
101    return clusters
102
103 """
104     Finds new k centroids
105     Inputs:
106         dataset
107         clusters - current cluster assignments for all the data points
108         mean_fun - mean or median - function to use for calculating center of a cluster
109     Outputs:
110         list of modified k centroids
111 """
112 def find_new_centroids(dataset, clusters, mean_fun = np.mean):
113     unique_clusters = np.unique(clusters)
114
115     new_centroids = []
116     # go through all the clusters
117     for cid in list(unique_clusters):
118         cluster_members = []
119         pids = []
120         #go through all the data points and collect the point
121         #belonging to the particular cluster
122         for pid in range(len(dataset)):
123             # for all the points in that claster
124             if clusters[pid] == cid:
125                 cluster_members.append(dataset[pid])
126                 pids.append(pid)
127
128         #make a 2d matrix
129         cluster_members = np.stack(cluster_members, axis = 0)
130         # find the coordinates of the center by applying "mean" function
131         new_centroid = mean_fun(cluster_members, axis = 0)
132
133         new_centroids.append(new_centroid)
134
135     #print("New Centroids: ")
136     #print(new_centroids)
137     return new_centroids
138
139
140 def evaluate_clusters(clusters, targets):
141     #print("Evaluating..")
142     wrong = 0
143     total = 0
144     for i, d in enumerate(targets):
145         start_pos = total
146         end_pos = start_pos + len(d)
147         total += len(d)
148         cluster = clusters[start_pos:end_pos]
149         # cid = clusters[start_pos]
150         cid = max([int(c) for c in cluster], key=cluster.count)
151

```



Initialisation – how do we select our centroids?

- Forgy's method: choose k random data points from the dataset
- Random Partition method: Randomly assign data points to a cluster. Then calculate the mean of each cluster to get the initial centroids.
- K-means++: first centroid is a random datapoint, but remaining centroids are chosen based on the maximum squared distance
→ centroids are spread out evenly

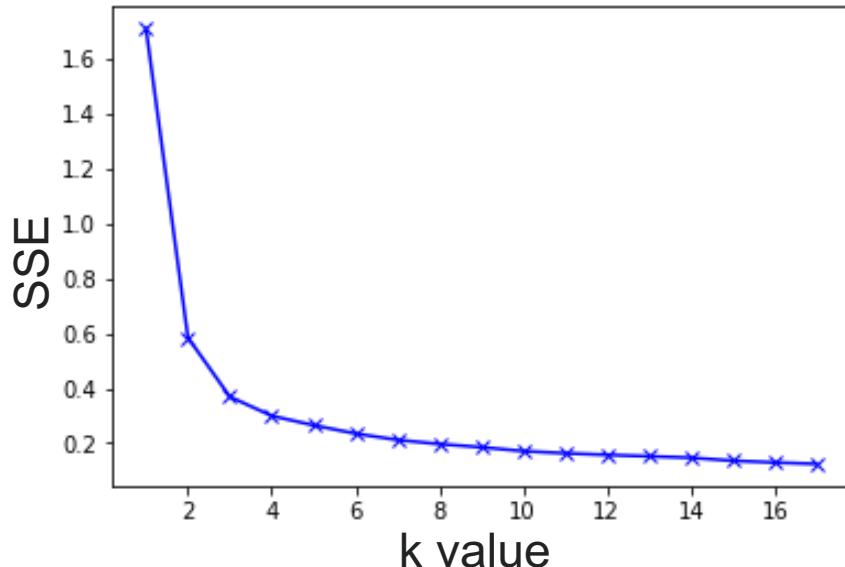


Okay... but how do we determine the number of clusters we want?

K = ?

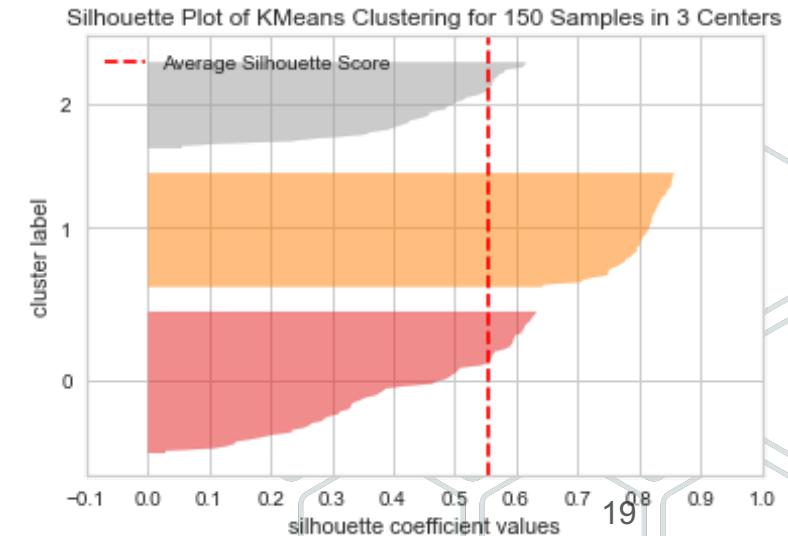


Elbow plot



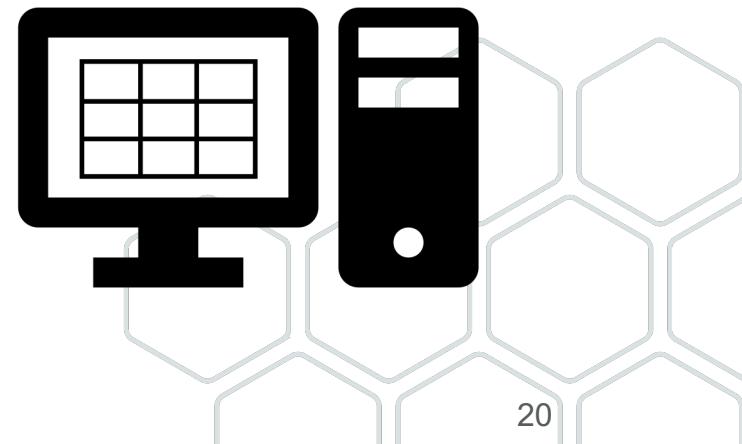
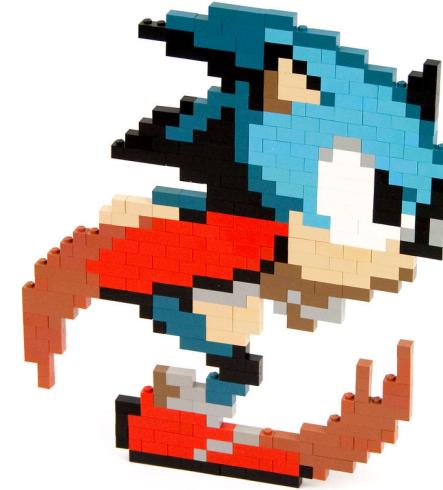
- Each time we increase the number of clusters → the SSE decreases
- Goal: select a small value of k that still has a low SSE
- Elbow represents where we start to have diminishing returns by increasing k

Sepal length (cm)	Petal length (cm)	Petal width (cm)
3.5	1.4	0.2
3.2	5.7	2.3
3.2	5.9	2.3
2.9	4.7	1.4
3.7	1.5	0.4



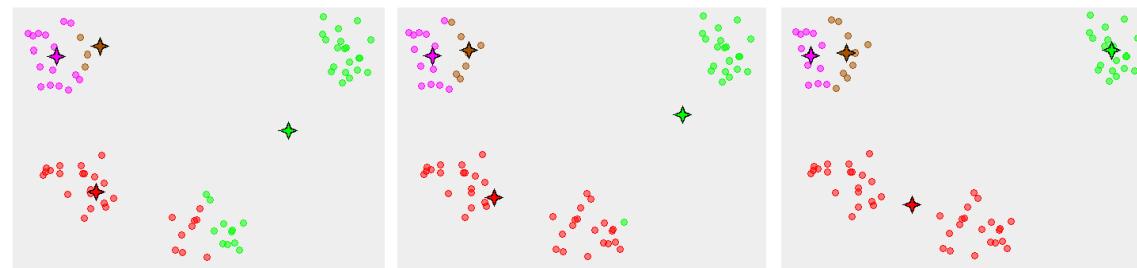
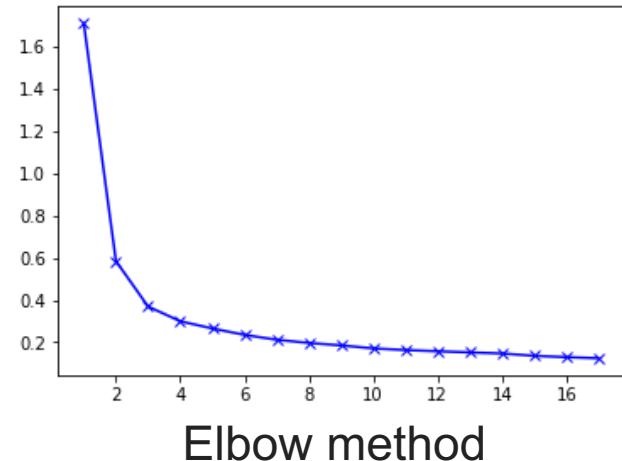
What are the strengths?

- Easy to understand and implement
- Fast
- Scalable

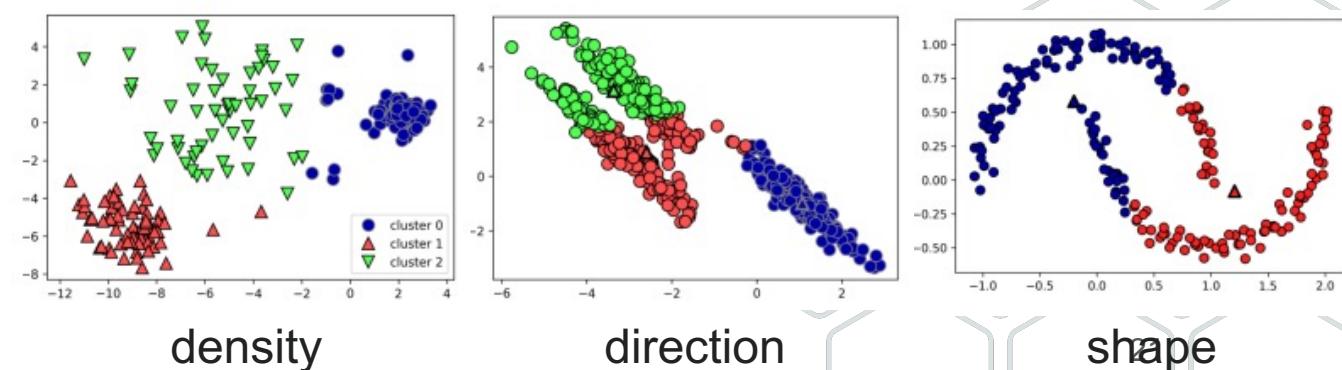


What are the limitations?

- Choosing k manually – it's a hassle!
- It is dependent on initial values: necessary to run the algorithm several times to avoid suboptimal solutions – converges to a local minimum
- Not good at clustering data of varying sizes, densities, or nonspherical shapes



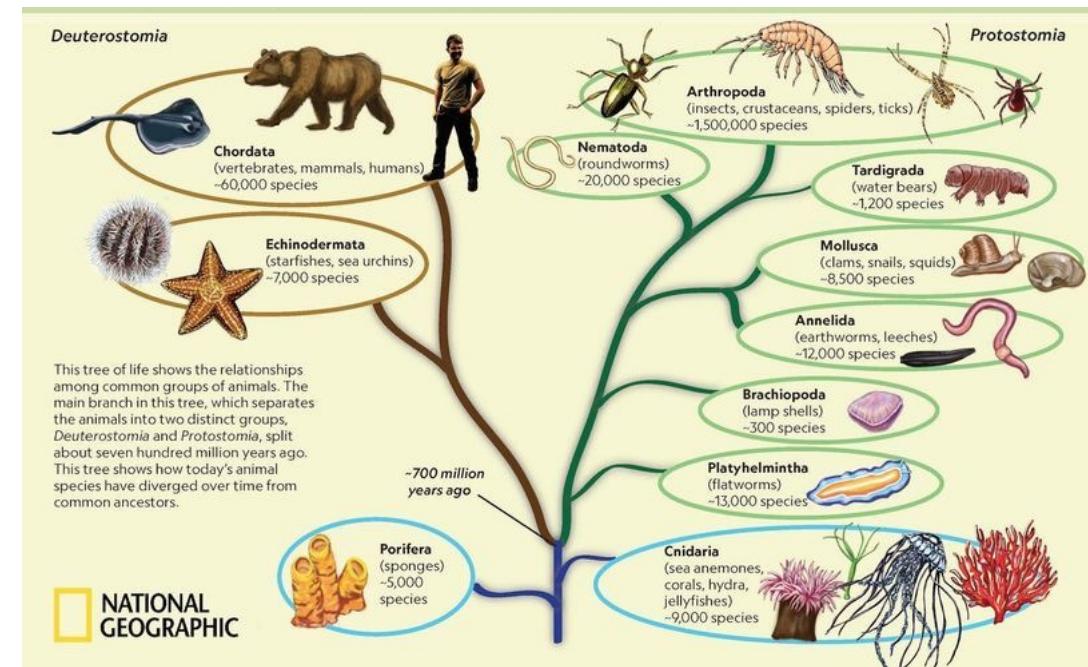
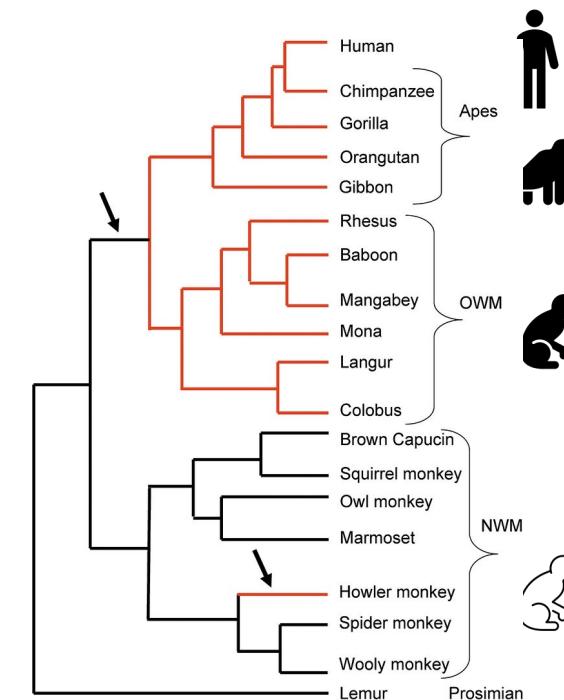
Bad centroid initialization → Suboptimal solution



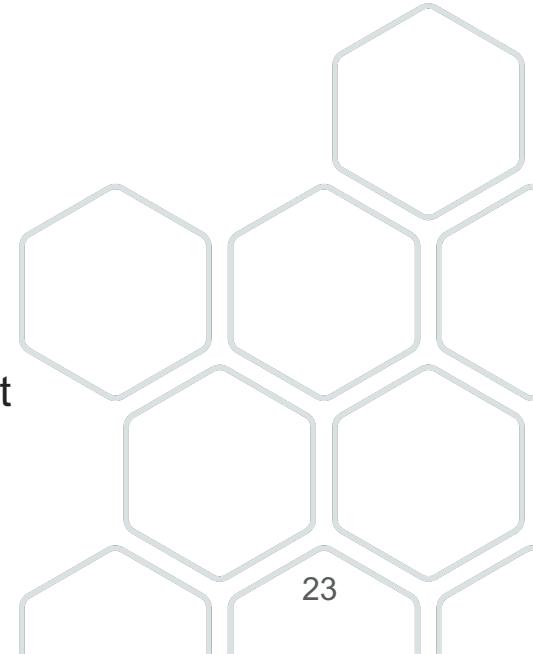
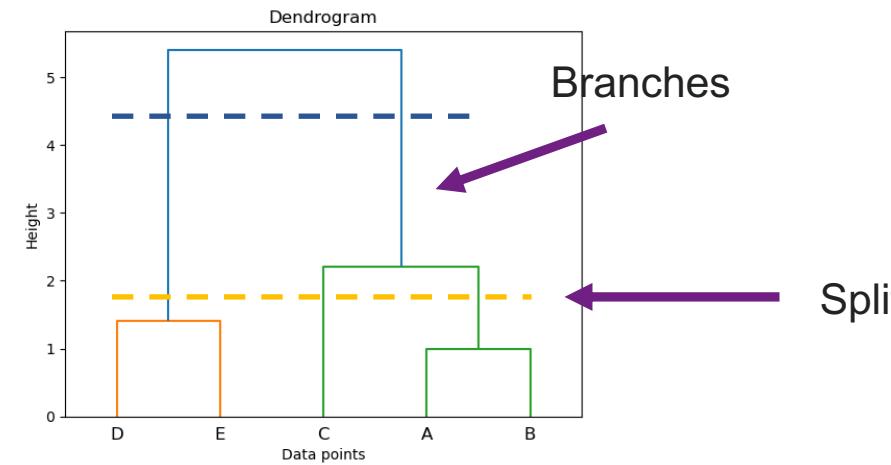
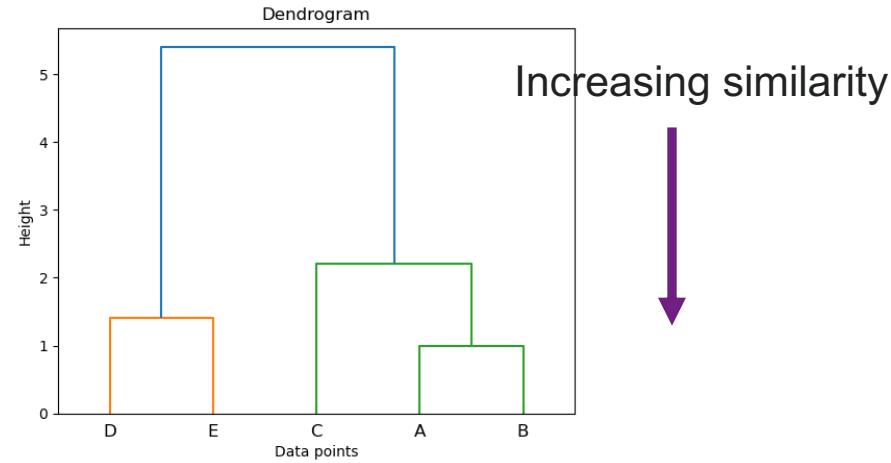
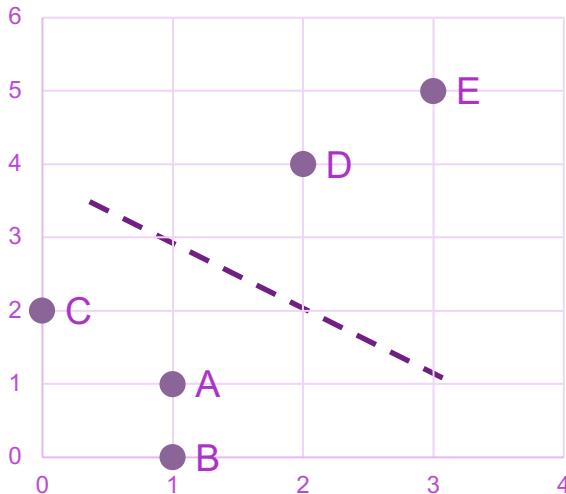
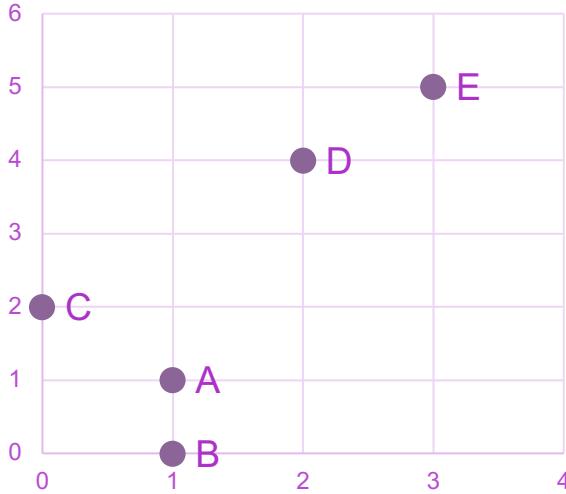
Hierarchical clustering

“Hierarchical clustering algorithms [...] approach the problem of clustering by developing a binary tree-based data structure called the dendrogram. Once the dendrogram is constructed, one can automatically choose the right number of clusters by splitting the tree at different levels to obtain different clustering solutions for the same dataset without rerunning the clustering algorithm again.”

(Reddy and Vinzamuri, 2015)

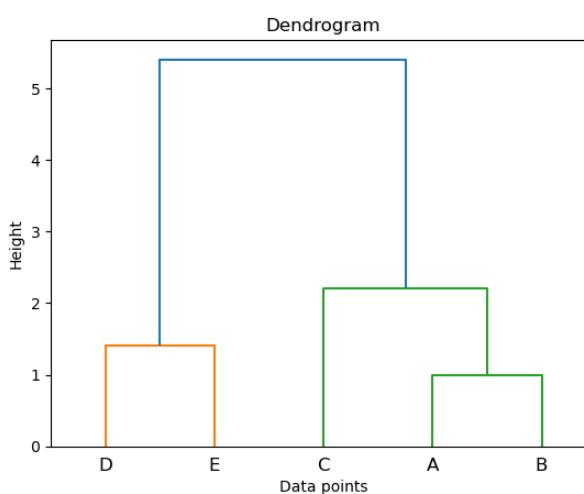
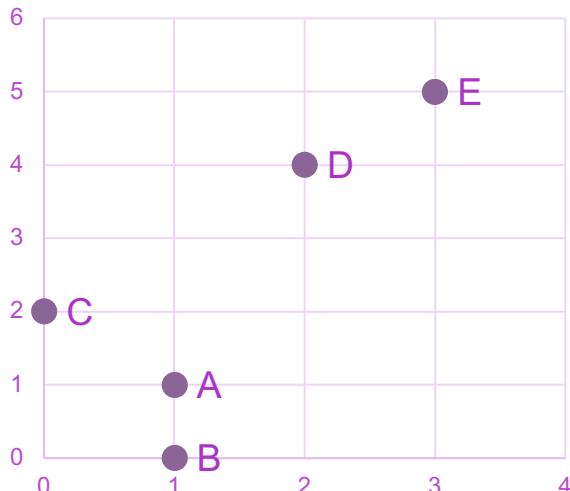
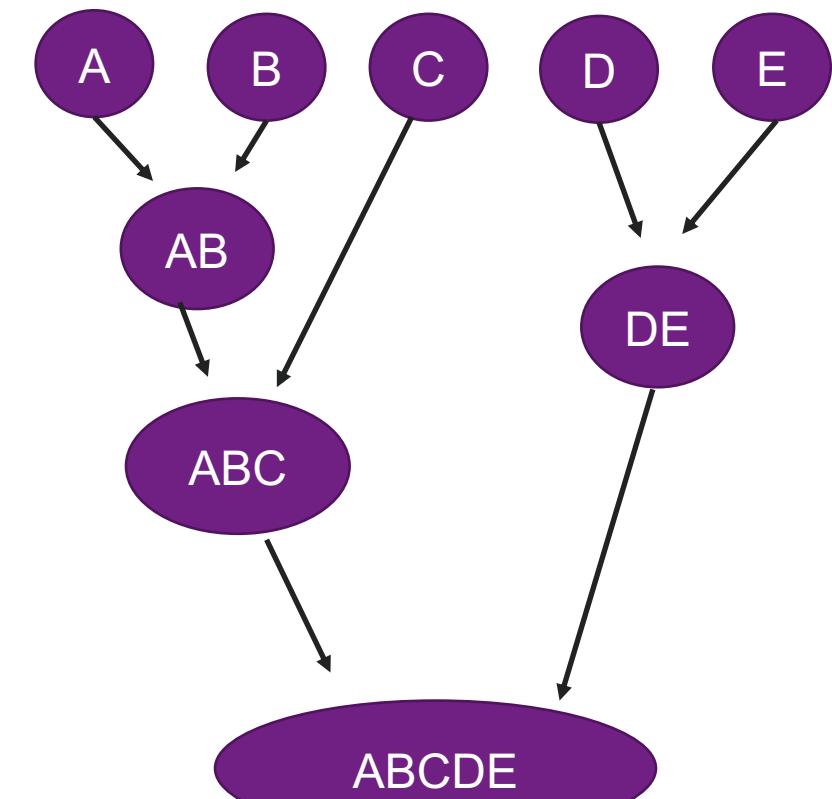


How do I read a dendrogram?

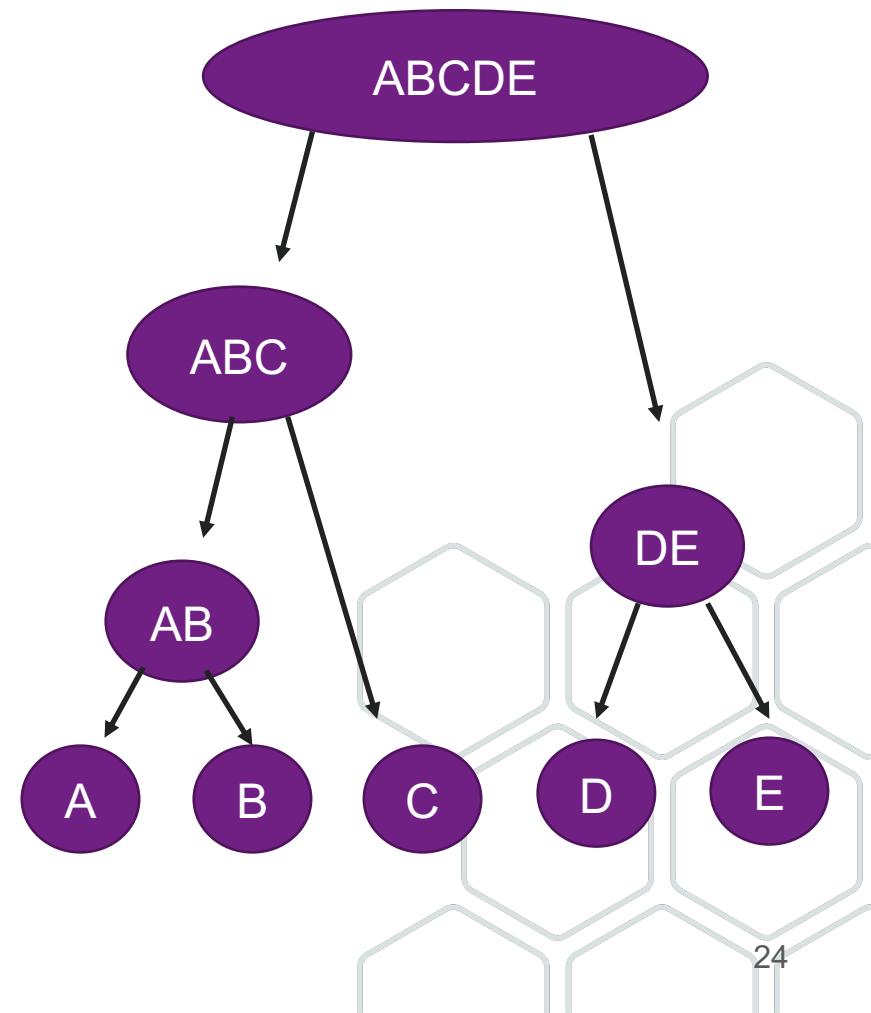


What are the 2 main approaches to hierarchical clustering?

1) Agglomerative



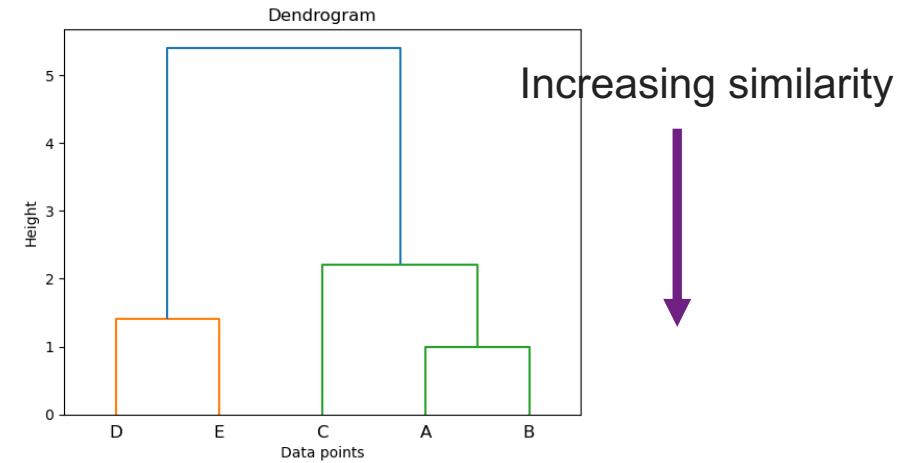
2) Divisive



...but how do we know which clusters should be combined, or split?

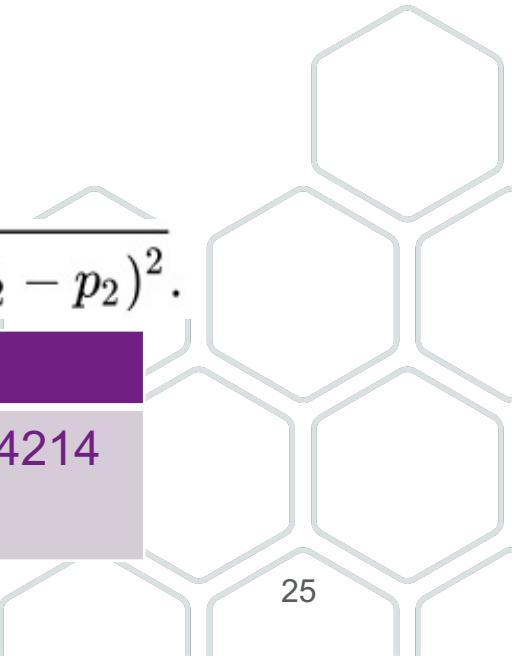
1) Measure of distance – some measure of similarity

- Hierarchical clustering is proximity-based
- Affects the shape of the clusters
- Used to build distance matrix
- Default is Euclidean distance, but other measures exist: correlation-based, Levenshtein distance etc.



$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}.$$

p	q	ED
3	4	1.414214
2	1	



2) Linkage criterion – different ways to link clusters based on distance

- A means of determining whether certain clusters should be merged
- Default is complete-linkage
- Other commonly used linkage criteria: single-linkage, average-linkage
- Used to update the distance matrix and merge clusters



Agglomerative hierarchical clustering: Using complete-linkage

PSEUDO ENGLISH

1. Turn each data point into a singleton, i.e., into a cluster of a single element
2. For each pair of clusters, calculate their distance
3. Merge the pair of clusters that take the smallest distance
4. Continue with step 2 and 3 until the termination criterion is satisfied
5. The termination criterion most commonly used is a threshold of the distance value

Algorithm Agglomerative (D)

Input: Dataset (D)

Output: Dendrogram

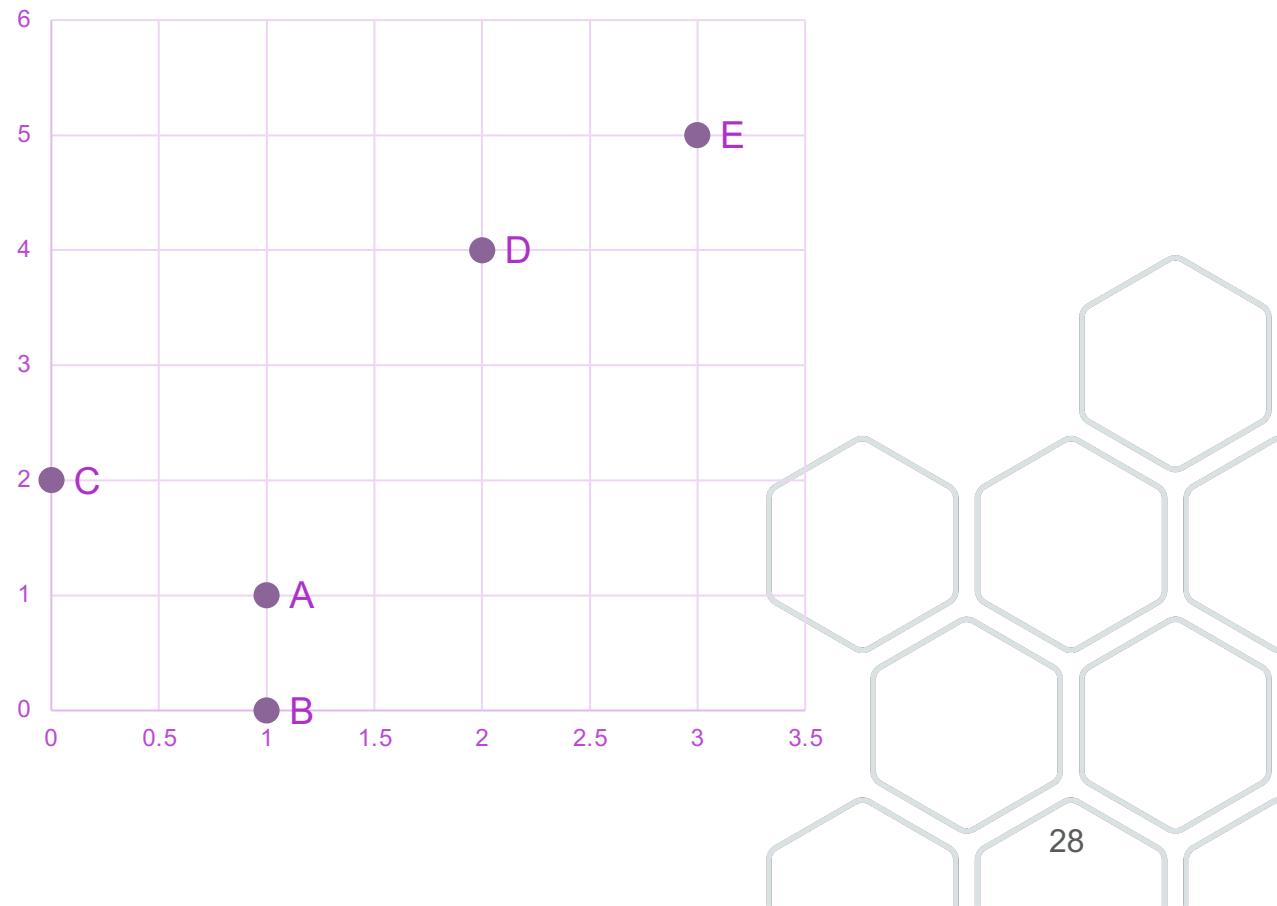
1. Make each data point in the dataset D a cluster,
2. Compute all pair-wise distances of $x_1, x_2, \dots, x_n \in$
3. **repeat**
4. find two clusters that are nearest to each other;
5. merge the two clusters to form a new cluster c
6. compute the distance from c to all other clusters;
7. **until** there is only one cluster left

PSEUDOCODE

Step by step...

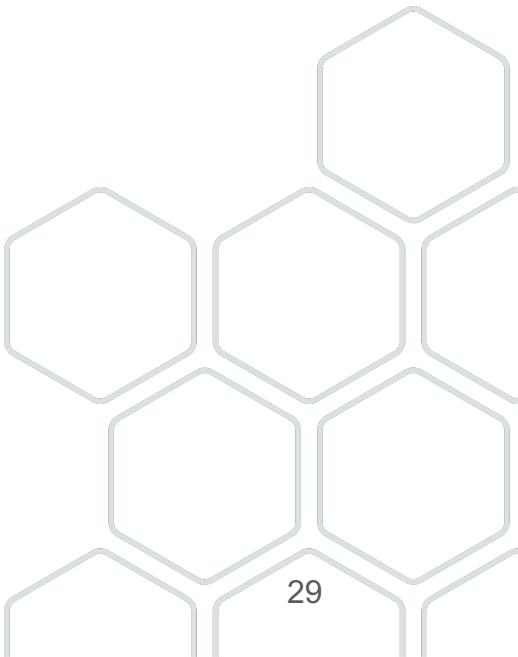
1) Load in dataset

Dps	x	y
Dps	sepal length (cm)	Petal length (cm)
A	1	1
B	1	0
C	0	2
D	2	4
E	3	5



2) Build distance matrix and identify smallest distance

	A	B	C	D	E
A	0	1	1.4	3.2	4.5
B	1	0	2.2	4.1	5.4
C	1.4	2.2	0	2.8	4.2
D	3.2	4.1	2.8	0	1.4
E	4.5	5.4	4.2	1.4	0



3) Perform merge and update distance matrix

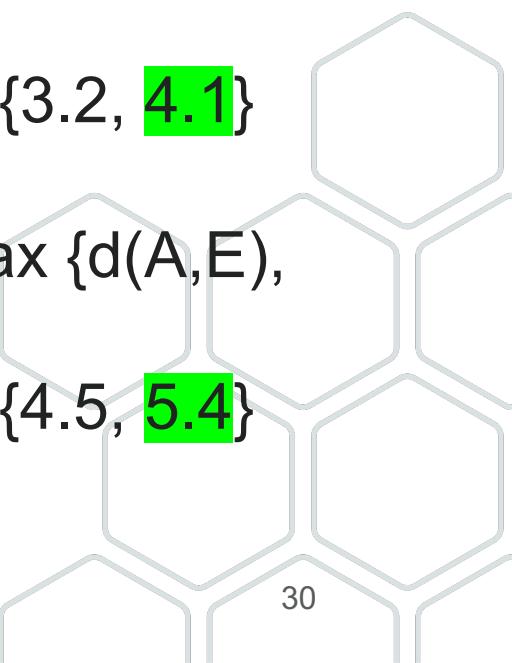
Updated distance matrix:

	AB	C	D	E
AB	0			
C	2.2	0		
D	4.1	2.8	0	
E	5.4	4.2	1.4	0

$$d[(A,B),C] = \max \{d(A,C), d(B,C)\}$$
$$= \max \{1.4, 2.2\}$$

$$d[(A,B),D] = \max \{d(A,D), d(B,D)\}$$
$$= \max \{3.2, 4.1\}$$

$$d[(A,B),E] = \max \{d(A,E), d(B,E)\}$$
$$= \max \{4.5, 5.4\}$$



Continue merging and updating the distance matrix...

	AB	DE	C
AB	0		
DE	5.4	0	
C	2.2	4.2	0

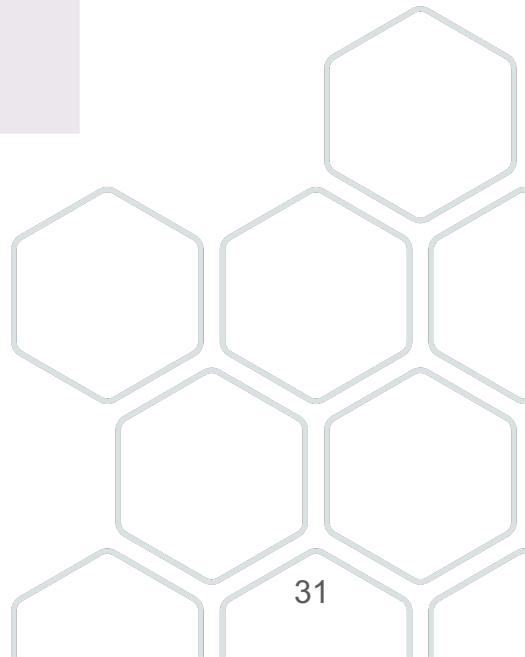


	ABC	DE
ABC	0	
DE	5.4	0

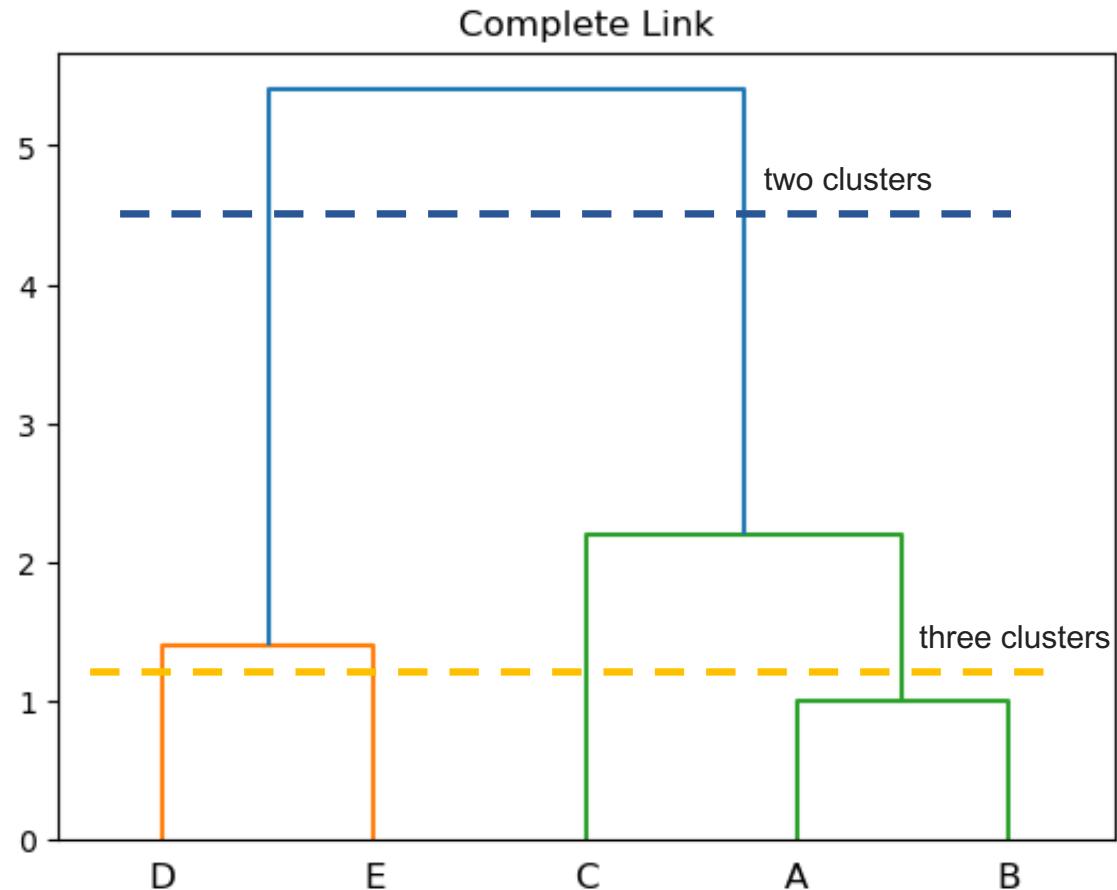
$$\begin{aligned} d[(A,B),(D,E)] &= \max \{d((A,B)D), d(A,B)E)\} \\ &= \max \{4.1, 5.4\} \end{aligned}$$

$$\begin{aligned} d[(C,(D,E))] &= \max \{d(C,D), d(C,E)\} \\ &= \max \{2.8, 4.2\} \end{aligned}$$

$$\begin{aligned} d[(A,B,C),(D,E)] &= \max \{d((D,E)(A,B), ((D,E),(C)) \\ &= \max \{5.4, 4.2\} \end{aligned}$$



RESULT

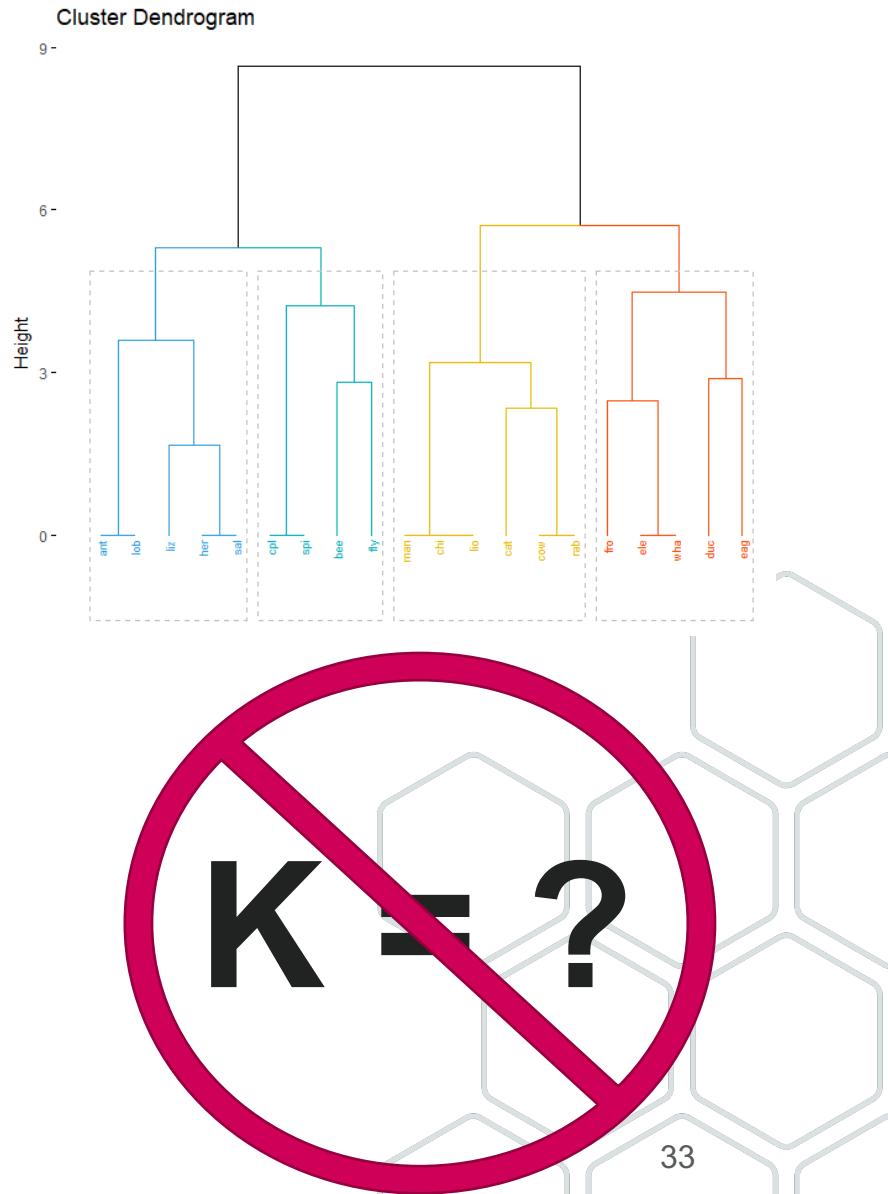


- Dendrogram: y-axis denotes when in the agglomerative algorithm two clusters get merged
- Y-axis also shows how far apart the merged clusters are → pay attention to the length of the branches



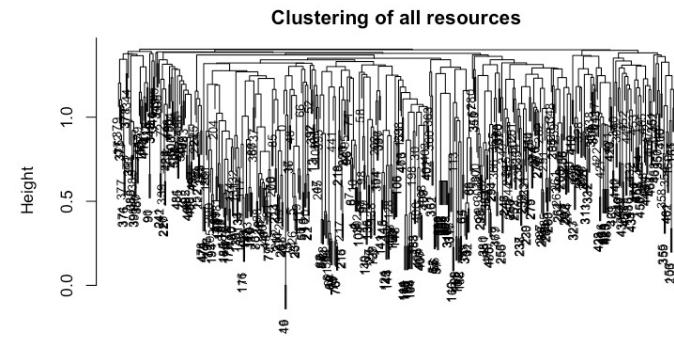
What are the strengths?

- Easy to understand and implement
- Most appealing output
- Can handle non-convex clusters
- No need to specify the number of clusters!



What are the limitations?

- Mathematically simple...but computationally expensive!
 - Hard to visualize results with a large dataset
 - Heavily driven by heuristics and arbitrary decisions
 - Algorithm can't undo previous step



K-Means vs Hierarchical clustering

	K-Means	Hierarchical clustering
Time complexity	$O(n)$	$O(n^2)$
Hyperparameters Tuning	Must specify the number of clusters (k) and retrain model for each k	No need to specify k value, can perform split wherever
Data structure	Better performance when dealing with convex clusters	Generates better results when dealing with non-convex clusters
Types/variations	Many variations (e.g., K-median, K-medoid)	Two approaches: Agglomerative and Divisive
Result Robustness	Result may be different on different runs	Same parameters generate the same result every time



Any Questions...

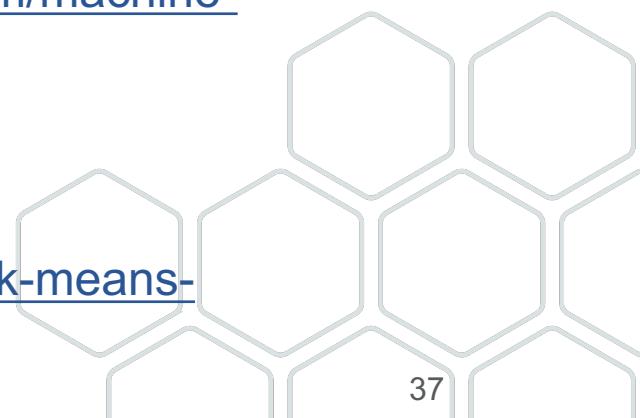
References

Geron, A (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.

Müller, A.C, and Guido, S (2017). Introduction to Machine Learning with Python: A Guide for Data Scientists, O'Reilly Media, Inc.

Reddy, C.K., and Vinzamuri, B (2015). A Survey of Partitional and Hierarchical Clustering Algorithms.

- Outlier detection using clustering (Outlier detection image)
<https://blogs.sap.com/2020/12/16/outlier-detection-by-clustering/>
- Image compression using K-Means clustering (parrot image)
<https://medium.com/@agarwalvibhor84/image-compression-using-k-means-clustering-8c0ec055103f>
- Clustering Algorithms: Types of Clustering (images) <https://developers.google.com/machine-learning/clustering/clustering-algorithms>
- K-Means clustering (image with centroids as triangles)
- K-Means pseudocode (image)
https://www.cms.waikato.ac.nz/~abifet/book/chapter_9.html#rfig9-1
- K-Means Elbow Method and Silhouette Analysis (image) <https://stackabuse.com/k-means-elbow-method-and-silhouette-analysis-with-yellowbrick-and-scikit-learn/>



Material for Tuesday the 2nd of November

GitHub:

https://github.com/UKDataServiceOpen/ML_Workshop



Survey

- When you leave the webinar, please complete our short survey
- Just click on 'continue' to access the survey.





Thank You.

Email:

louise.capener@Manchester.ac.uk

nadia.kennar@manchester.ac.uk,

Twitter:

@CapenerLouise

@NadiaKennar

