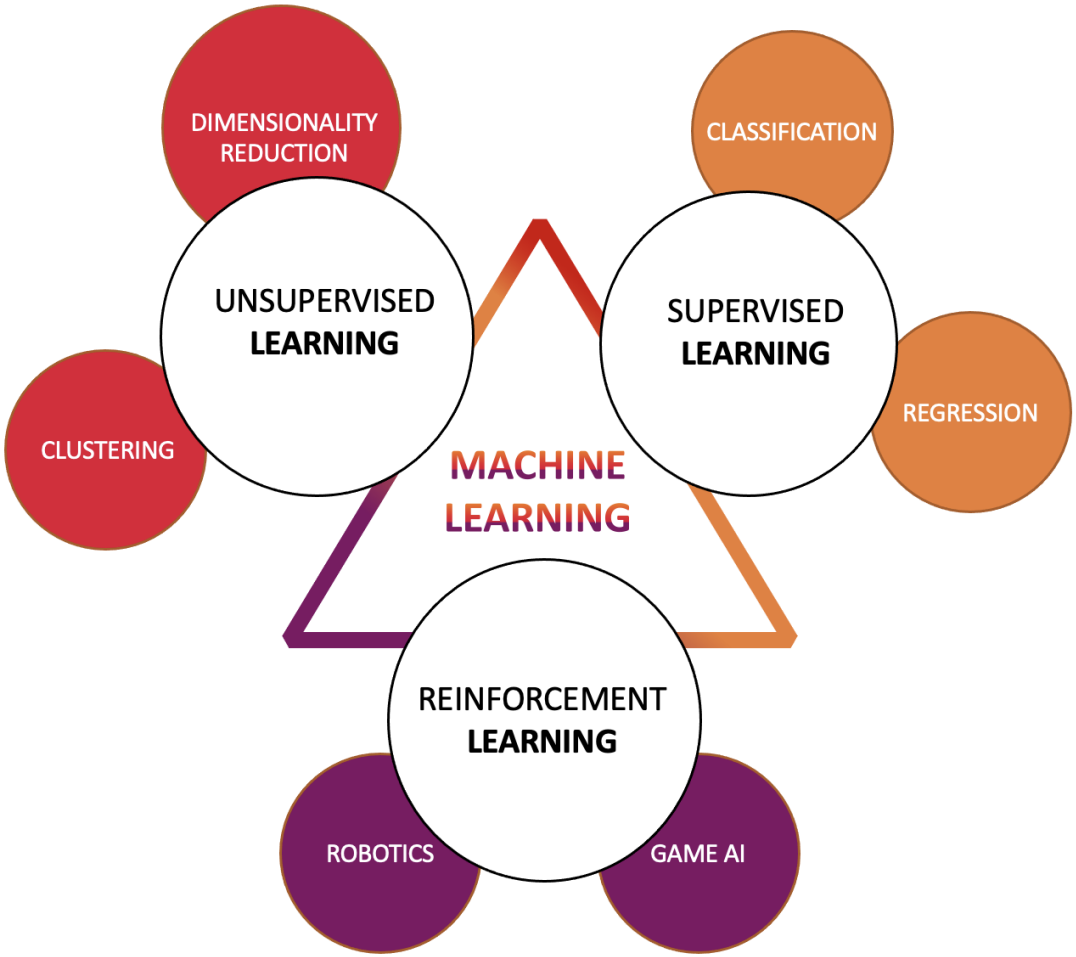
AIML430/COMP309: ML Tools and Techniques
Lecture 5: Clustering

Ali Knott
School of Engineering and Computer Science, VUW

Recap: Types of machine learning



Yesterday we did regression and dimensionality reduction...
Today we'll do clustering.

Clustering algorithms

A clustering algorithm separates an unlabelled dataset defined in a *numerical feature space* into separate groups, or 'clusters'.

- Essentially, it *creates classes*, and labels datapoints with classes.
- It works best if the data *actually has clusters in it*.
  That's not guaranteed!

When might you use a clustering algorithm?

Customer Segmentation in Retail:

- Clustering transaction data to identify different customer types

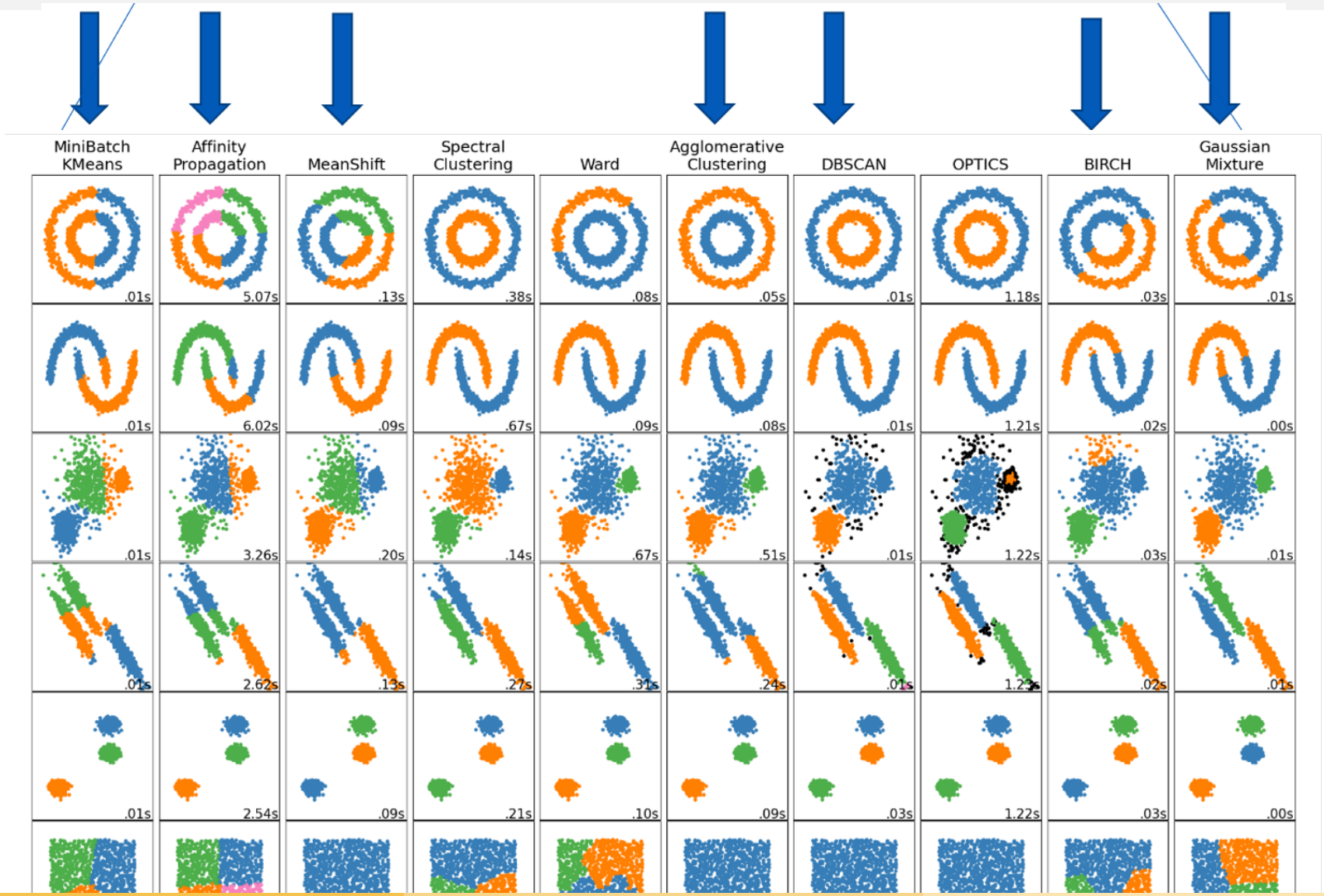
Social Media Analysis:

- Clustering users, by the content they interact with
  Clustering content, by the users who interact with it

Astronomical Data Analysis:

- Clustering stars and galaxies based on their properties to understand celestial structures

Different clustering algorithms give different results. . . We will look at 7 of them today.

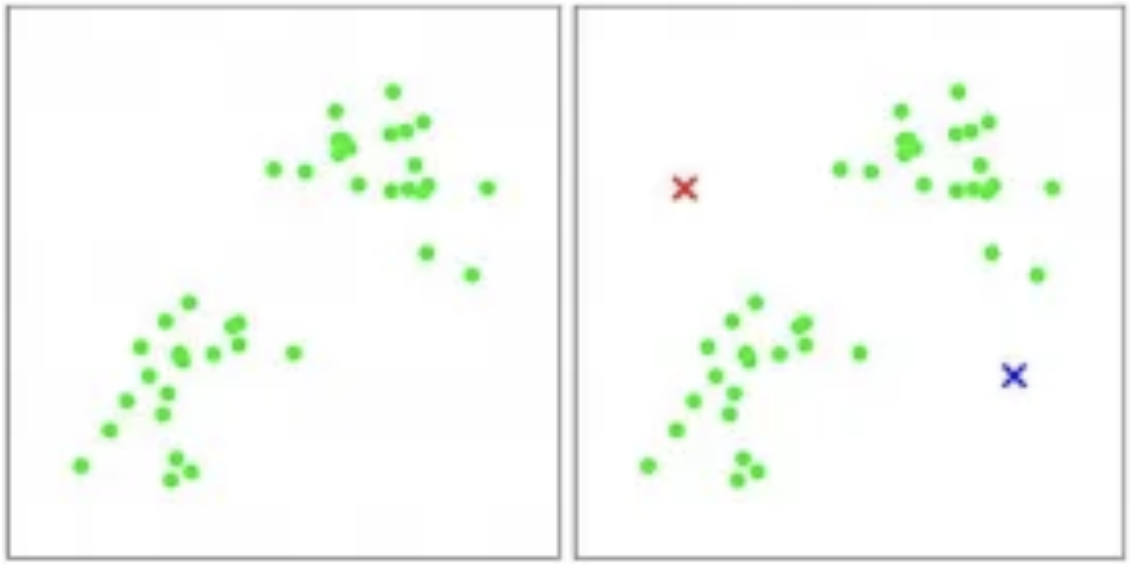


1. *k*-means clustering

K-means clustering partitions instances into *K* clusters.

- Each cluster is defined by a centroid—which is the *average position* of all its members.
- After clustering, each item belongs to the cluster with the nearest centroid.

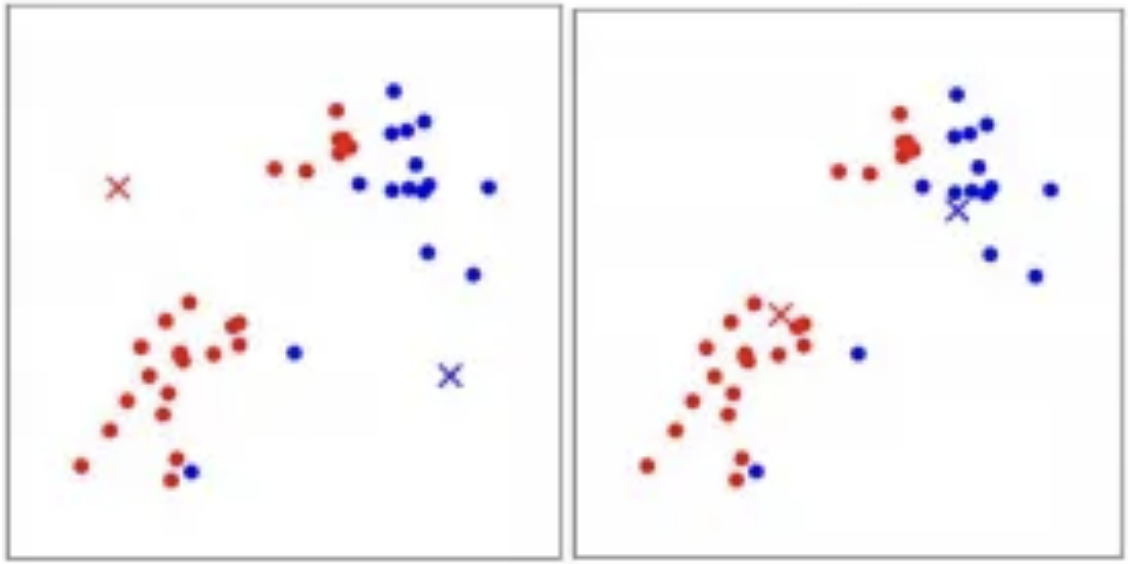
**The user must specify k in advance!*



1. *k*-means clustering

We start by placing centroids at random locations in feature space.

- Then we assign every item to its *nearest centroid*.
- Then we *recompute the centroid* of each cluster.
- And we *iterate* on that process...
  Until the centroids *stop moving*. (Called convergence.)

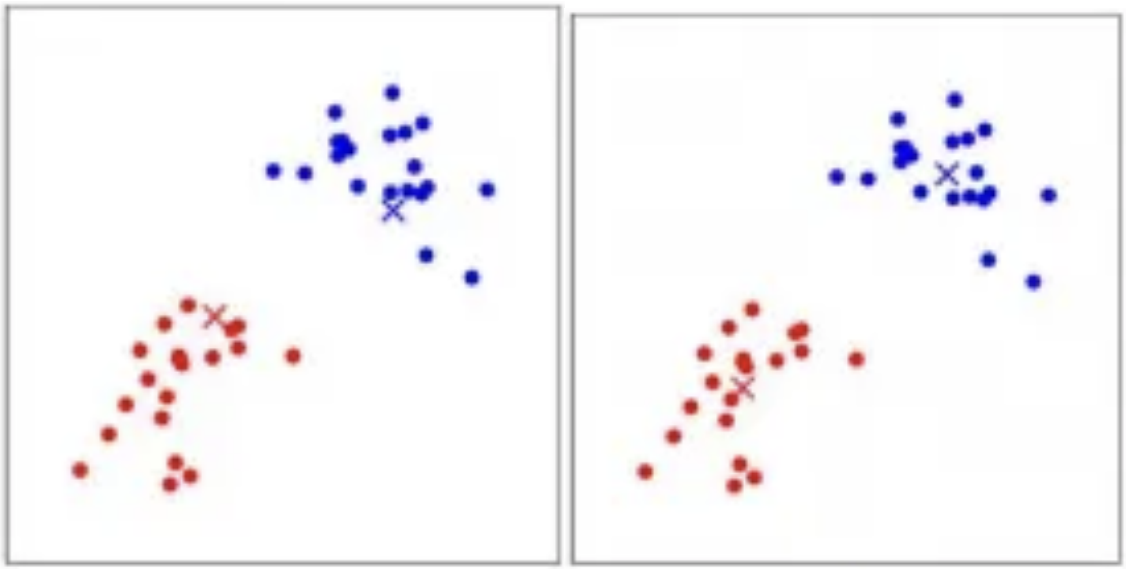


1. k-means clustering

To compute the centroid of cluster of n items, we just take the average value for each input dimension x.

If the positions of the n items for a given dimension are $p_1 \ldots p_n$, the centroid for that dimension is

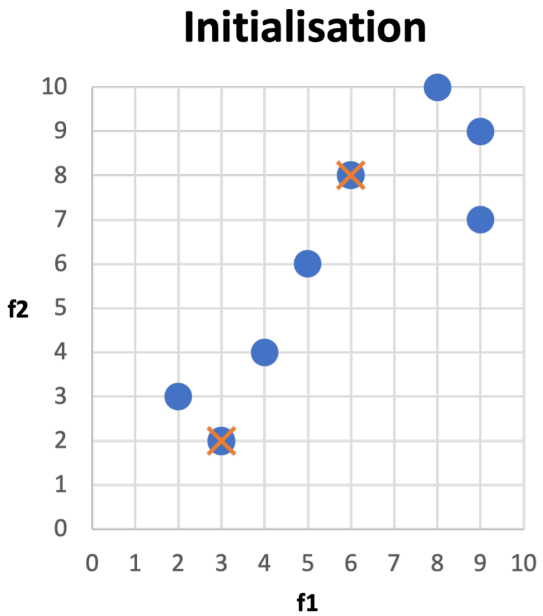
$$\frac{\sum_{i=1}^{n} p_i}{n}$$

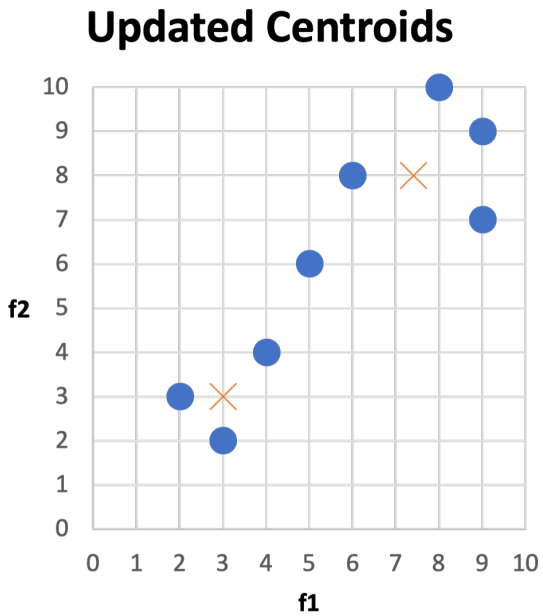


1.1. *k*-means clustering

Here's a dataset of 8 items, defined in 2D feature space.
Let's do k-means with $k = 2$, and assign our initial centroids like this.

Instance	f1	f2
1	2	3
2	3	2
3	4	4
4	5	6
5	9	7
6	9	9
7	6	8
8	8	10





Which items are grouped with the centroid at (3,2)?

- Items 1(2,3), 2(3,2) and 3(4,4).

What will the new centroid be for this group?

- $(\frac{2+3+4}{3}, \frac{3+2+4}{3}) = (3, 3)$.

A few points about *k*-means

k-means algorithm is an optimisation algorithm:|

- It optimises the average distance between training items and their nearest centroid.
- The inertia of a clustering result is the sum of squared distances between items and their nearest centroid.

There are many distance metrics that can be used...

- Euclidean is the most obvious...

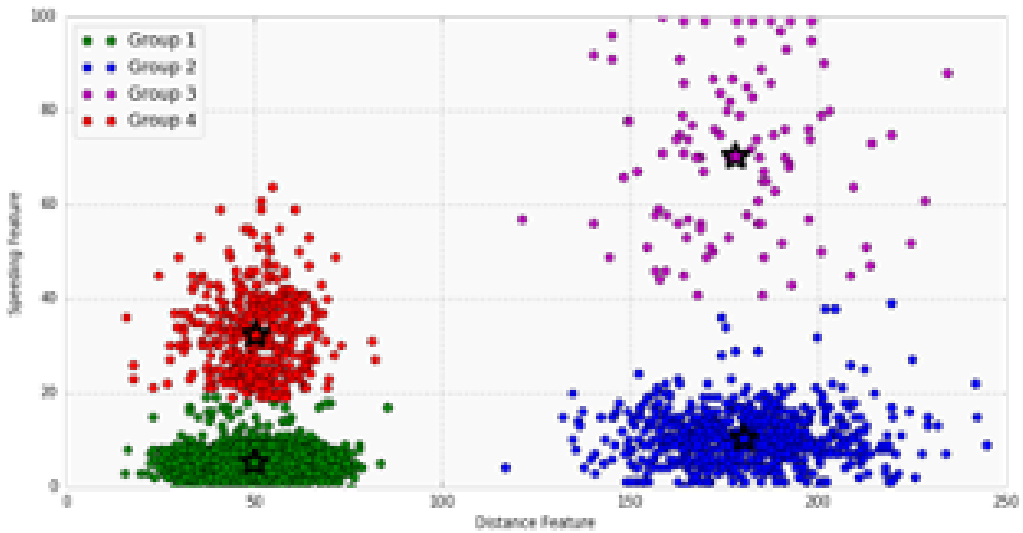
Choosing the right value of *k*

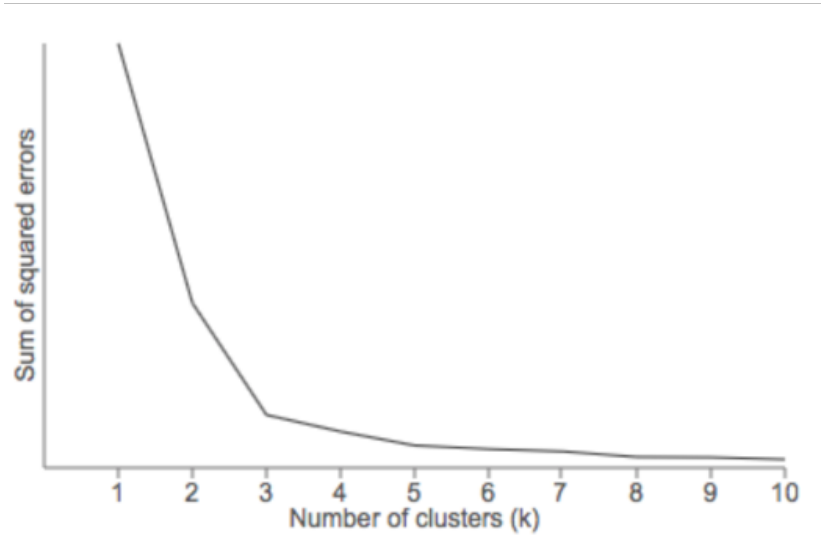
If you get *k* wrong, you can really misrepresent the data.

- There are lots of ways of estimating a good value for *k*.

One simple way is the elbow method.

- Run *k*-means for *a range* of values of *k*.
- For each run, calculate the *average distance between points and centroids*, and plot the result...
- The 'elbow' in the graph is a good choice for *k*.



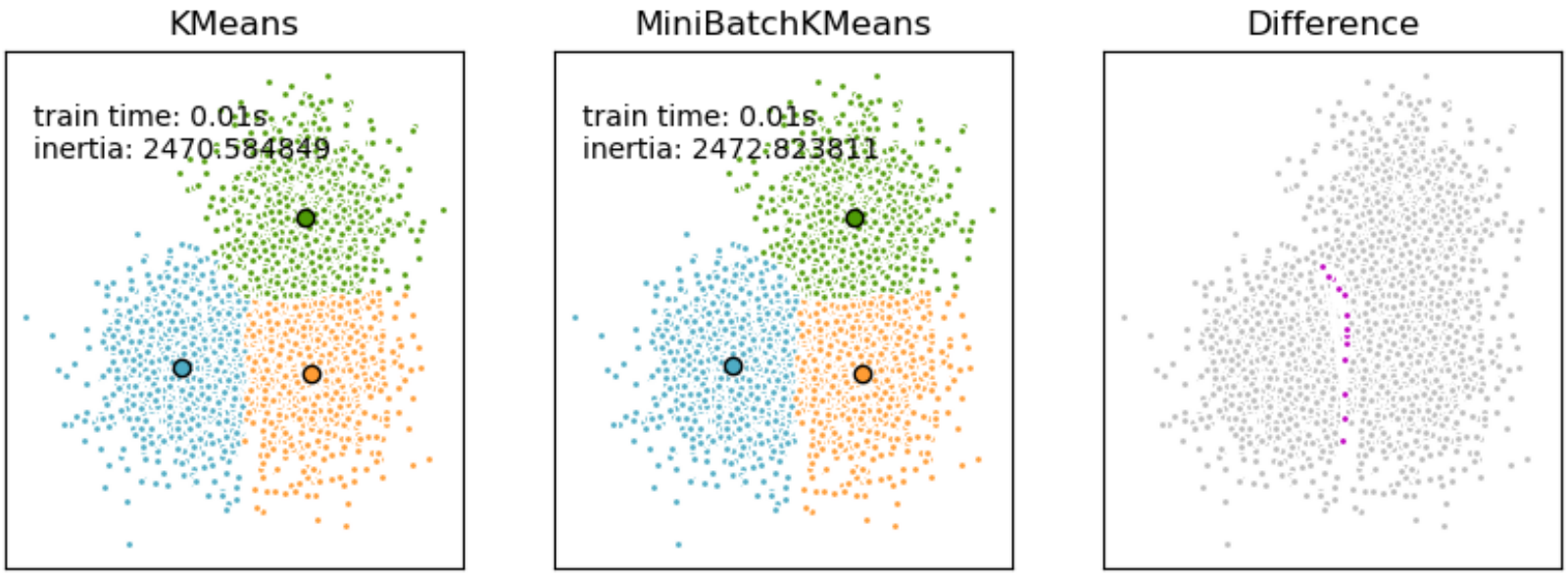


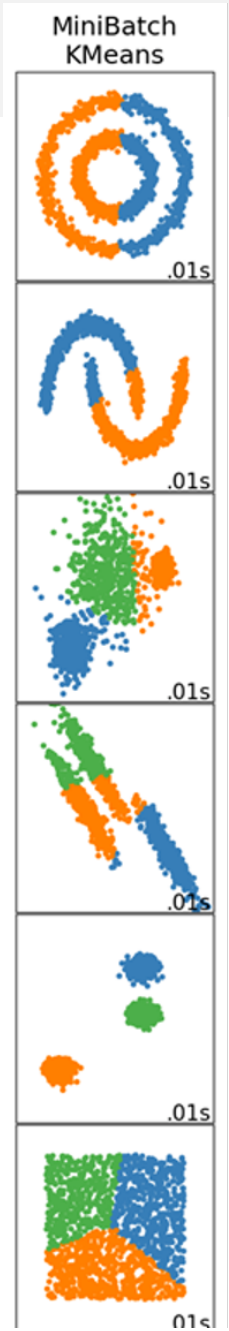
Mini-batch *k*-means

A variant of *k*-means uses mini-batches to reduce the computation time.

- Mini-batches are subsets of the input data, randomly sampled in each iteration.

Generally only slightly worse than full *k*-means.

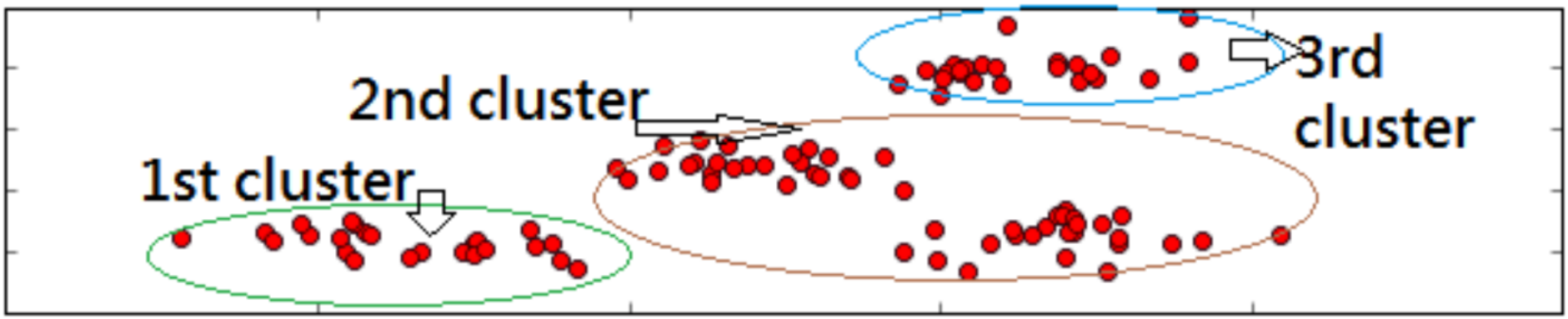




2. Mean shift clustering

In mean shift clustering, at each iteration, each point moves incrementally closer towards the centroid of points in its local neighbourhood.

- Neighbourhood is given by the bandwidth parameter.
- If you don't specify bandwidth, the scikit-learn fn *estimates* it.



- Mean shift doesn't scale well, because there's lots of distance calculation...
- But it doesn't require you to pick the number of clusters up front.

3. Gaussian mixture models

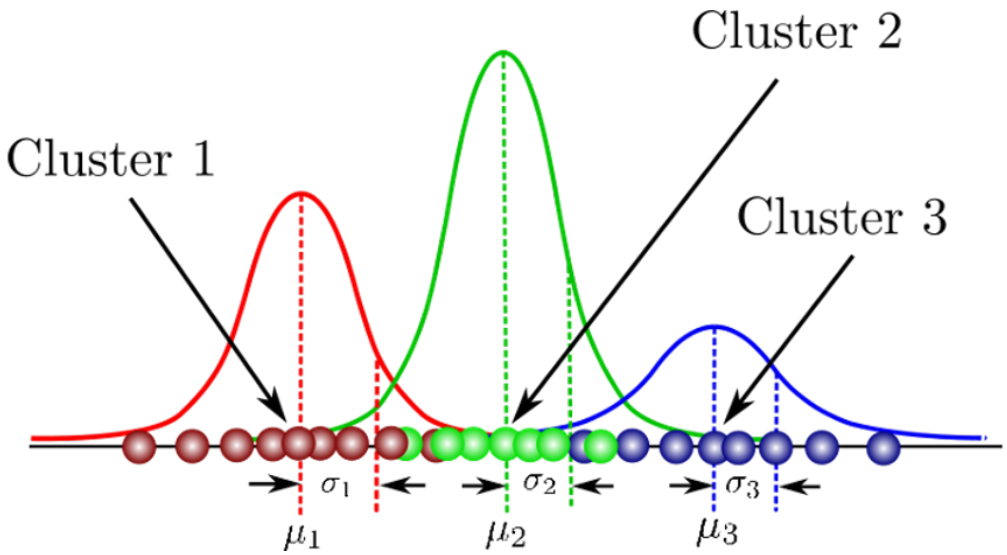
A few drawbacks with k-means:

- Clusters are defined by distance to centroids...
  - So clusters are encouraged to be *spherical*...
  - And *all the same size*.
- There's no *statistical principle* behind the model.
  - A cluster is best thought of as a statistical concept:
  - A random distribution of items, with a mean and a variance.

A Gaussian mixture model is a principled statistical method for clustering.

3. Gaussian mixture models

In 1 dimension, a Gaussian mixture model is just a collection of simple Gaussian functions, each with a mean and variance.



A few things to note:

- Each point now has a *probability* of belonging to a given cluster.
- That accounts nicely for clusters that *overlap* with one another.

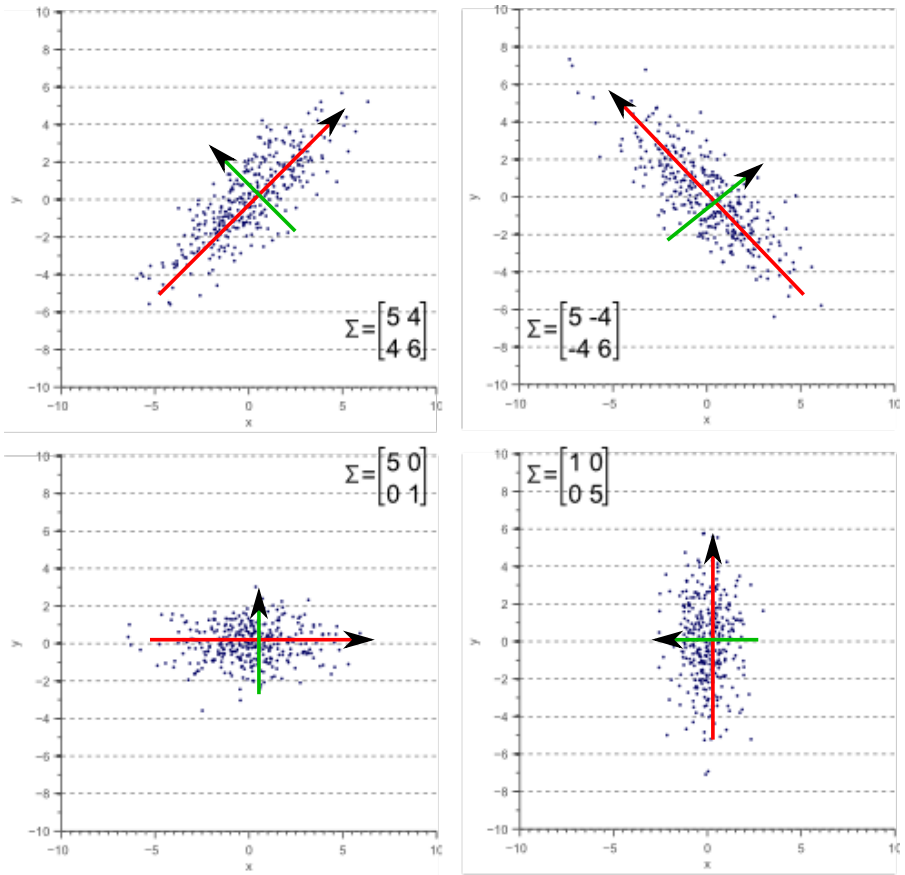
3. Gaussian mixture models

Now think of some *2D* clusters, of different ‘shapes’.

- We can represent each of these in a Gaussian model using a covariance matrix, just like we did for PCA.

Remember from last lecture:

- A covariance matrix defines the transformation that best maps a cloud of 2D Gaussian noise onto some given data.
- It *stretches* the cloud in each dimension, and *rotates* it.
- You can do this in *n* dimensions too!



3. Gaussian mixture models

In Gaussian mixture modelling, you still have to choose the number of Gaussians up-front.

- It's like *k*-means, in that sense.
- You also have to choose what *type* of covariance matrix it will use.
  - 'Spherical', 'diagonal', 'tied' or 'full'...

Having set those parameters, the Gaussian mixture algorithm searches for a good fit of its *k* Gaussians to the data.

It uses an expectation maximisation algorithm.

- The basic idea is to find the Gaussians that *maximise the probability of the observed training data*.

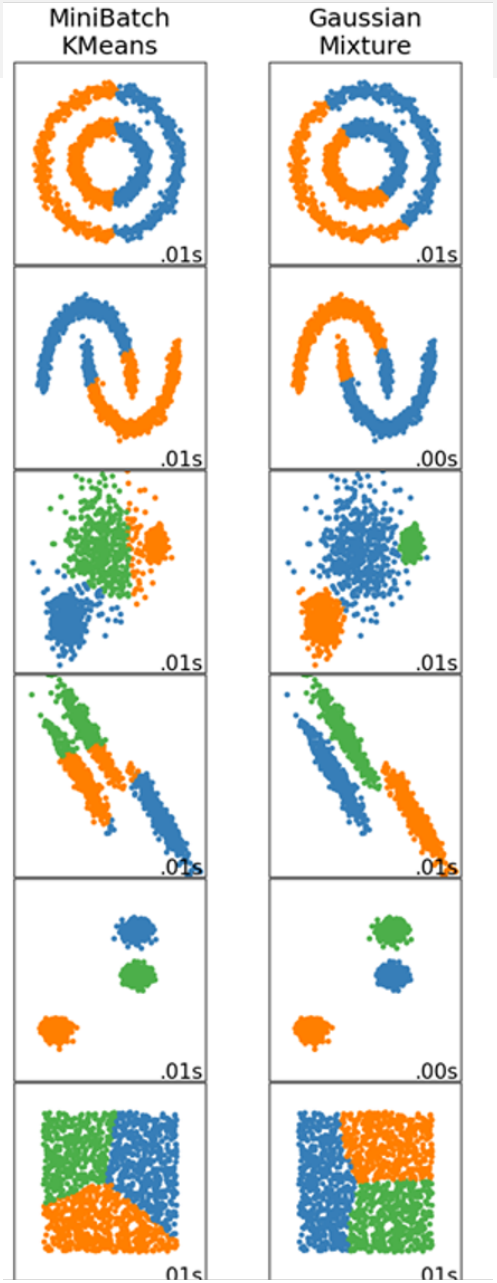
3. Gaussian mixture models

In expectation maximisation, we begin by setting the parameters of our Gaussians to random values.

Then:

- Make *soft assignments* of each data item to Gaussians.
- Incrementally *adjust* the parameters of each Gaussian to increase the likelihood of the observed data, *given those assignments*.
- *Iterate* until the parameters converge.

This model is good at finding clusters of different *shapes* and *sizes*.



4. Affinity propagation

The idea in the affinity propagation algorithm is to identify exemplars of items from different clusters.

- An exemplar is a point that effectively *represents* items in its cluster.

Exemplars are determined by a novel message-passing process, where points in the dataset are modelled as 'communicating with one another'.

- The model has a 'distributed', 'self-organising' character.

4. Affinity propagation

Affinity propagation is basically a dating agency for training points.

Each point *sends* messages to all other 'target points', informing each one of its 'attractiveness' to the sender.

- 'Attractiveness' is a function of *similarity*.

Each target point then responds to all senders, informing each sender of its 'availability' for associating with it.

- The 'availability' score reflects the offers received from all senders.
- Targets make themselves most available to the senders to whom they are most attractive.

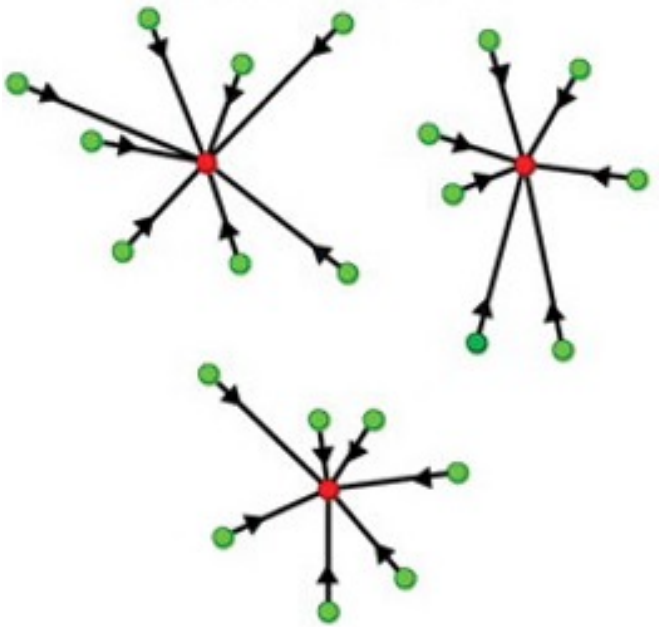
The process converges on a set of senders, which are each attractive to a particular cluster of points.

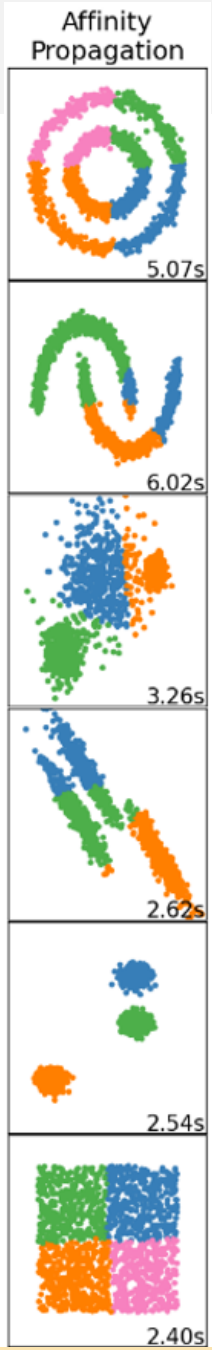
- These selected senders are 'exemplars' for their clusters.

4. Affinity propagation

Affinity propagation is different from the other algorithms we've seen so far, because each cluster is associated with a *data point*.

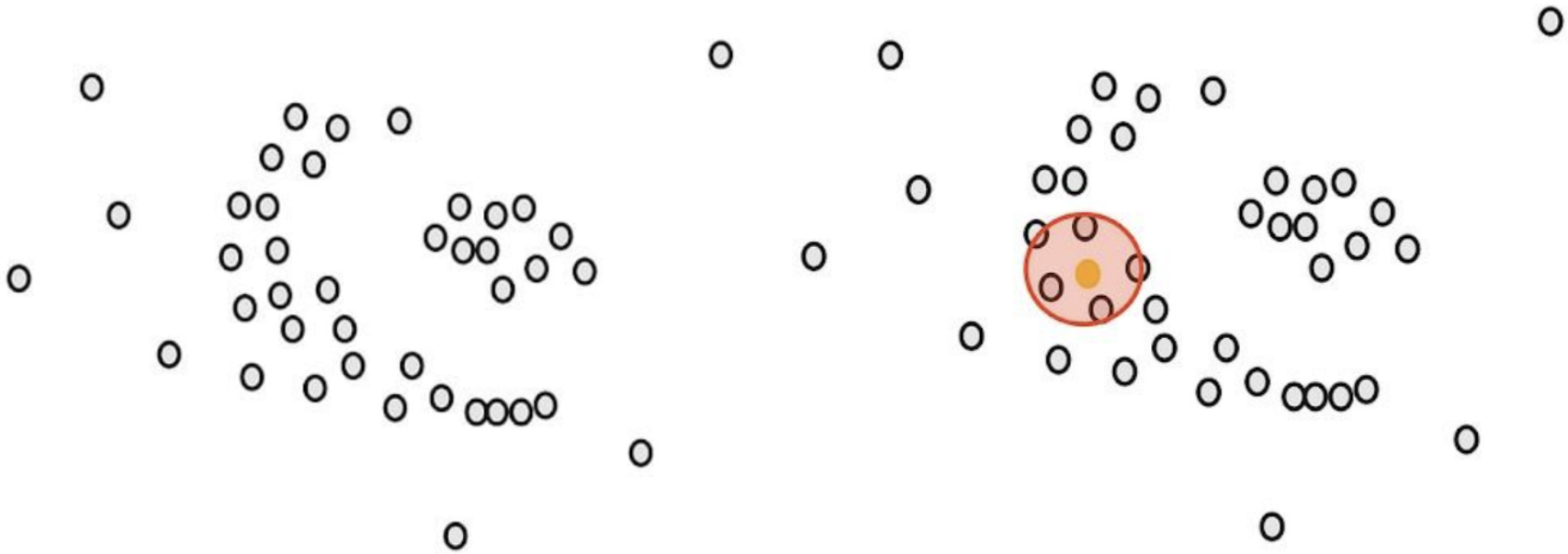
- *k*-means' centroid doesn't correspond to any point...
- Gaussian means needn't correspond to any point...





5. DBSCAN clustering

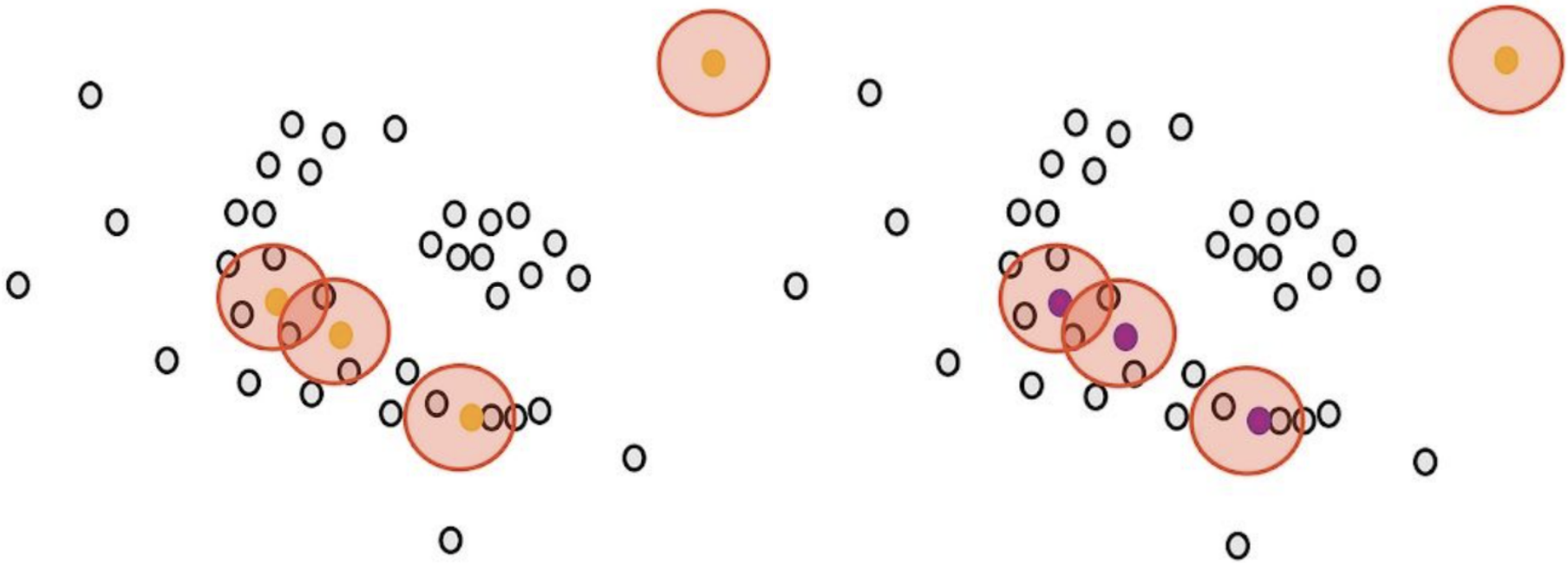
The core principle of DBSCAN is to associate clusters with *dense regions* of points.



5. DBSCAN clustering

To measure density, we define two parameters:

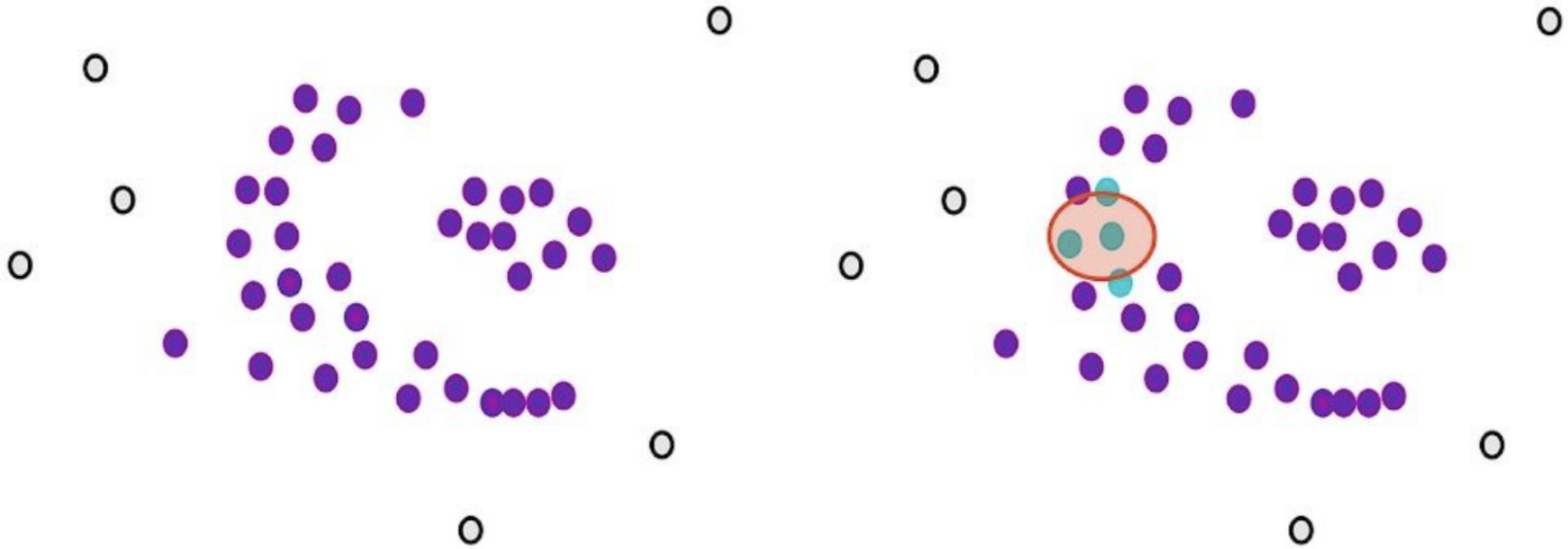
- Epsilon (eps) is the radius of a circle we place on top of each point.
- *minPts* is the minimum number of points that need to be *within that circle* to identify the point as being in a 'dense region'.
  - Let's set *minPts* to 3!



5. DBSCAN clustering

In a first pass, DBSCAN runs through *all points*, looking for points which have at least *minPts* neighbours.

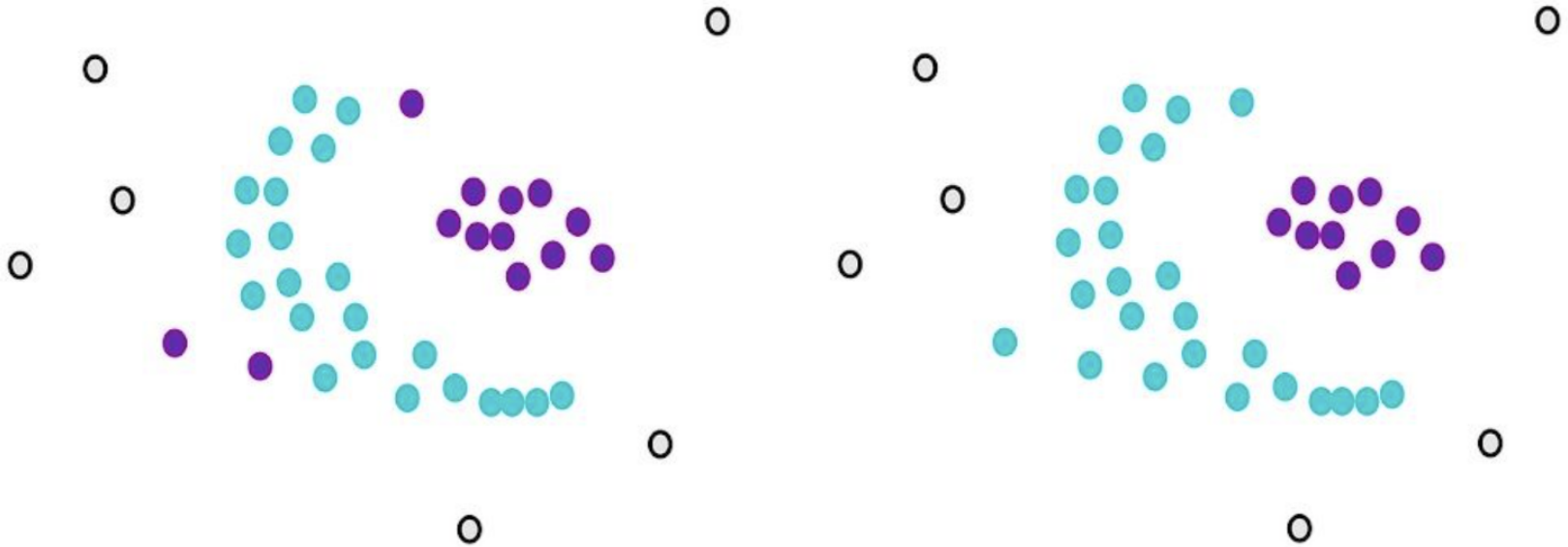
- Those that do are called core points. (Coloured purple here.)
- Here are all the core points in this example.



5. DBSCAN clustering

In a second pass, we separate our core points into different clusters.

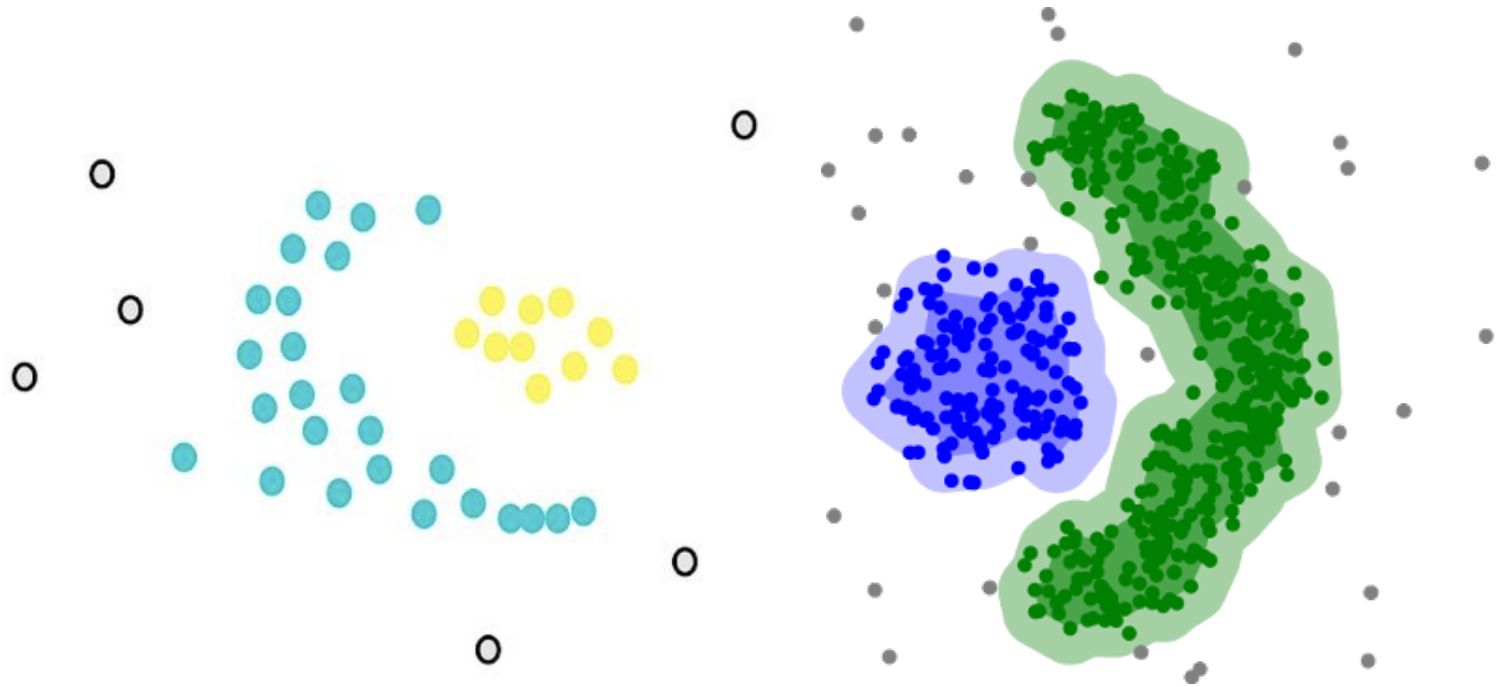
- To do that, we start at a random point, and 'grow' a cluster to all points reachable by circles...
- And we do that repeatedly for each cluster.



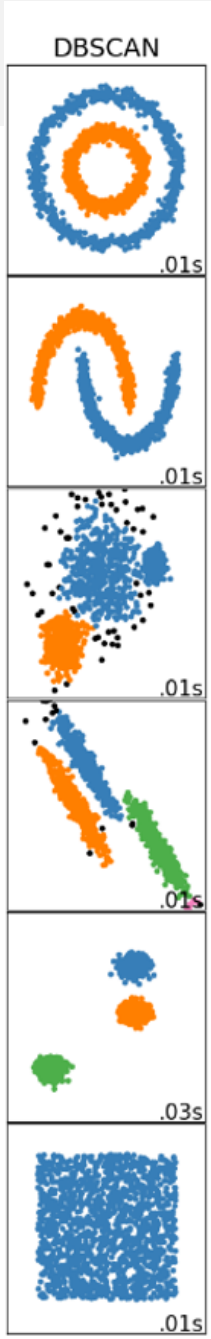
5. DBSCAN clustering

We end up with a set of clusters, and a set of outlier points, that don't belong to any cluster.

- There's also a set of border points, that aren't core, but end up at the edges of clusters.
- In this picture, the borders of clusters are shaded lighter..



DBSCAN advantages



1. We don't need to specify k in advance!
   - DBSCAN determines the number of clusters by itself.
     - Like mean shift and affinity propagation.

2. DBSCAN can detect *outliers*.
   - Like Gaussian mixture models.

3. DBSCAN is flexible about *cluster shapes*...
   - Even more flexible than Gaussian mixture models and mean shift...

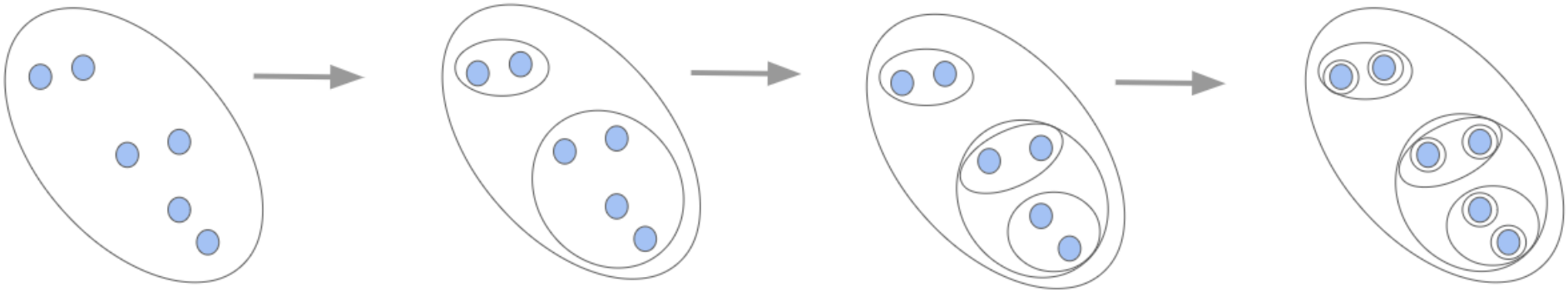
6. Hierarchical (agglomerative) clustering

Some datasets have clusters at different levels of hierarchy...

- 'Clusters *within clusters*.'
- To identify these, we can use a hierarchical clustering algorithm.

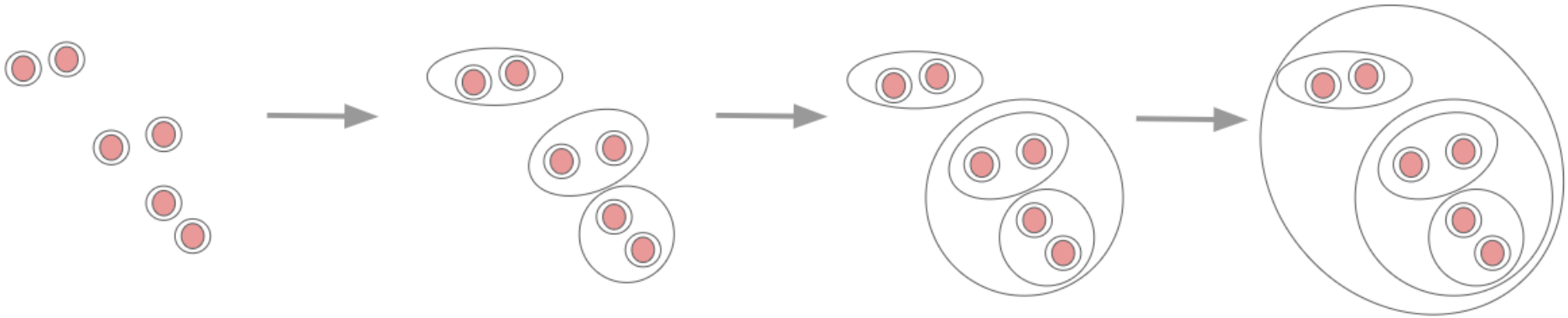
Two approaches to hierarchical clustering

Divisive clustering starts with the whole dataset, and recursively *splits*:



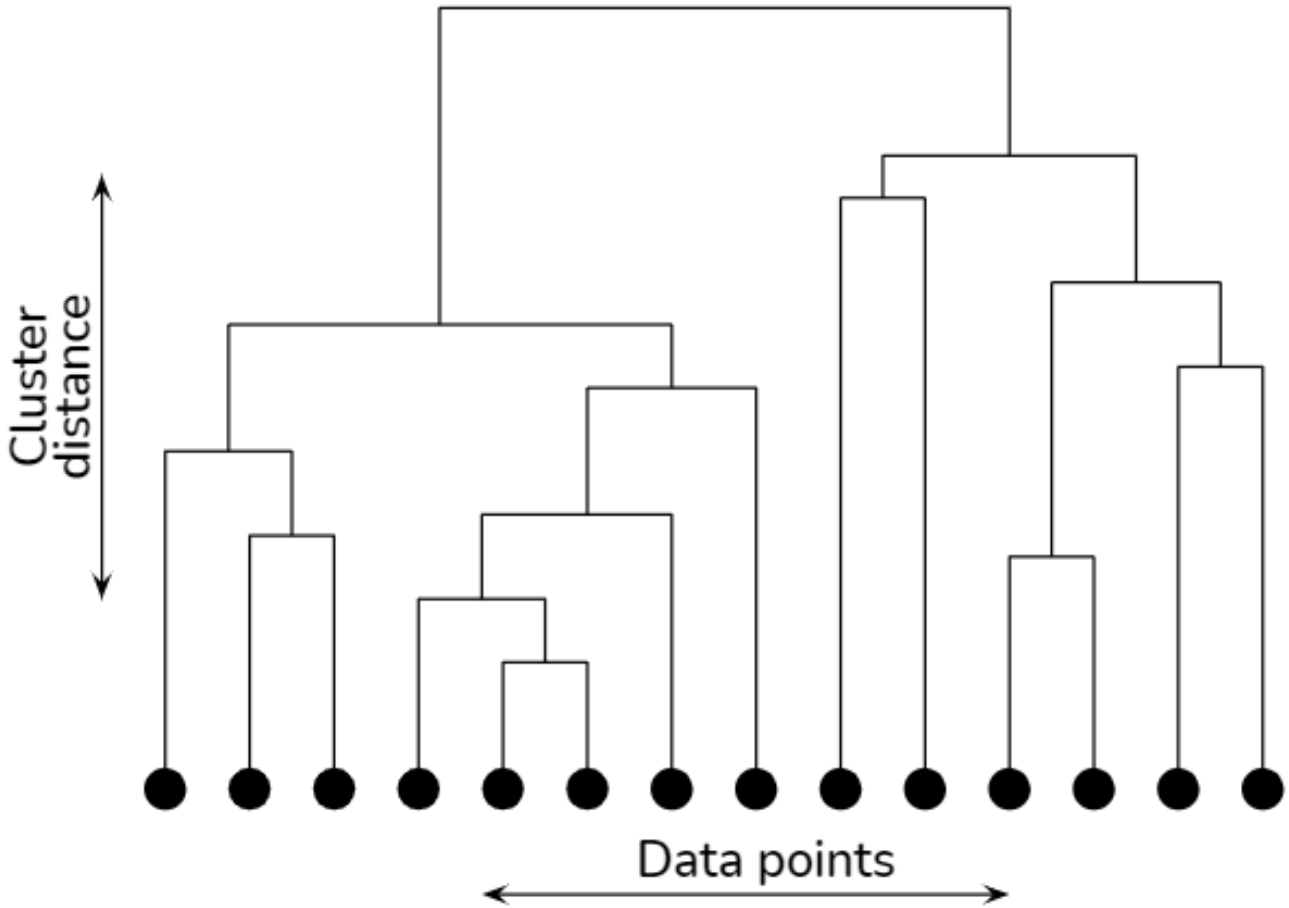
Agglomerative clustering finds the most similar items, and recursively *joins* into clusters.

- As we recurse, each cluster is represented by its centroid.

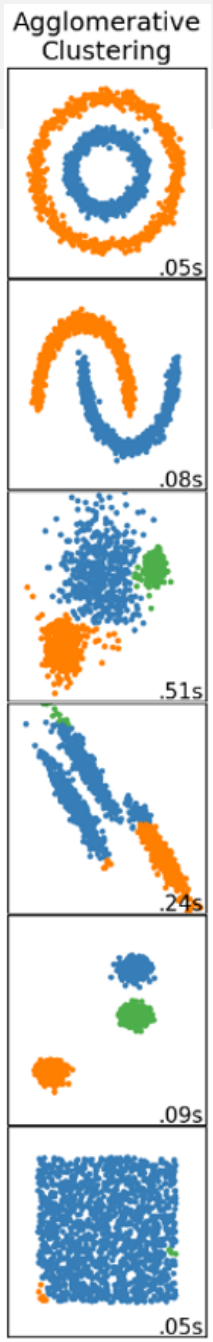


Dendrograms

Either way, we end up with a *tree* of clusters, called a dendrogram.



Performance of agglomerative clustering



It's pretty good at identifying weirdly-shaped clusters.

But it will find hierarchical structure *even if there isn't any.*

7. BIRCH

The algorithms so far don't scale well to very large datasets.

- An algorithm called BIRCH ('Balanced Iterative Reducing Clusters Using Hierarchies') is specially designed for this case.
- BIRCH is a type of hierarchical clustering algorithm.

BIRCH works by building a data structure called a CF tree.

- It looks for *dense regions* of points (or clusters).
- When it finds one of these, it creates a clustering feature (CF), that *summarises* this region, and adds it to the tree.
  - CFs *compress* data: they make the algorithm tractable.
  - The tree constructed by BIRCH is *balanced*, which again ensures efficiency.

When the CF tree is complete, leaf nodes can be further processed by more expensive clustering algorithms.