Week 5b
Perceptron recap.
Clustering.

Fundamentals of Machine Learning

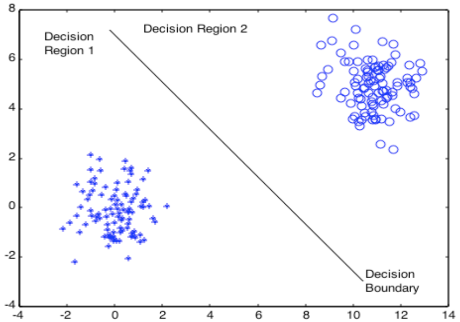
2023-24

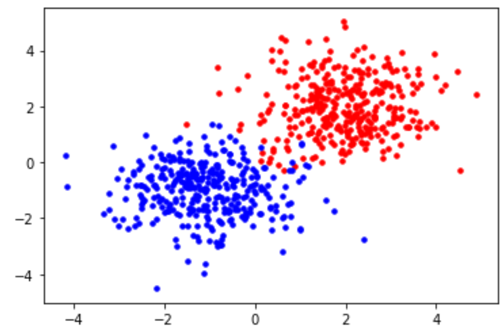
UNIVERSITY OF SUSSEX

Dr Benjamin Evans

Recap The Perceptron

- A single artificial neuron (perceptron) can do binary classification, if a linear decision boundary makes sense.



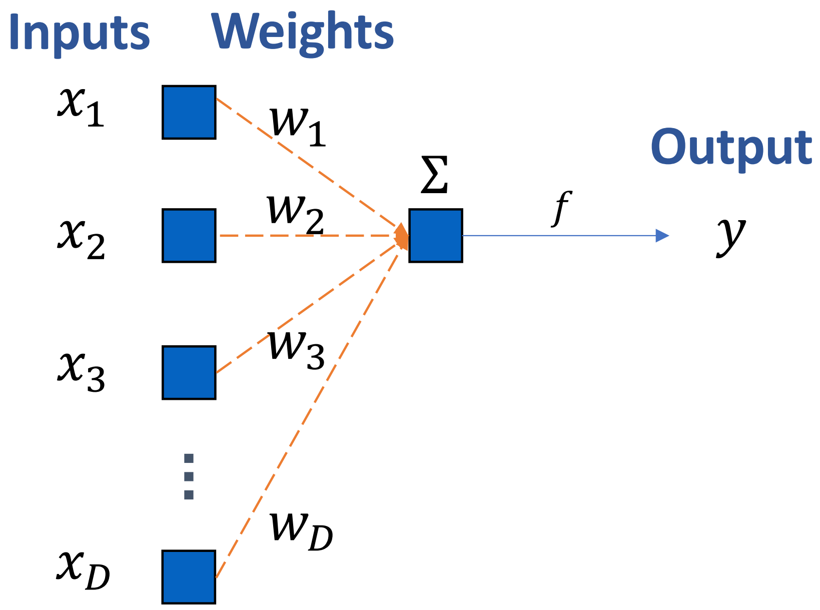


- The perceptron sums all the inputs, weighting each one according to the weight parameters, then classifies according to whether or not the sum exceeds the threshold of the activation function.

- The perceptron can do supervised learning, by updating weights using gradient descent.

The single perceptron

Can be viewed as a model…



… or a function.

$$y = f\left(\sum_{i=1}^{D} w_i x_i + w_0\right)$$

$$y = f(\boldsymbol{w} \cdot \boldsymbol{x})$$

- Takes the values for each of the features, x_i as **input**.
- Scales the features by their **weights** w_i and sums the products.
- Passes result through a *non-linear* activation function, f.
- Produces a binary classification as **output**, y, that is a function of the features: {0, 1} or {-1, 1}.

What are the parameters?

- The **weights**, $w_1, w_2, \ldots, w_D$
- The **bias**, w_0
- (The **activation function**, f)

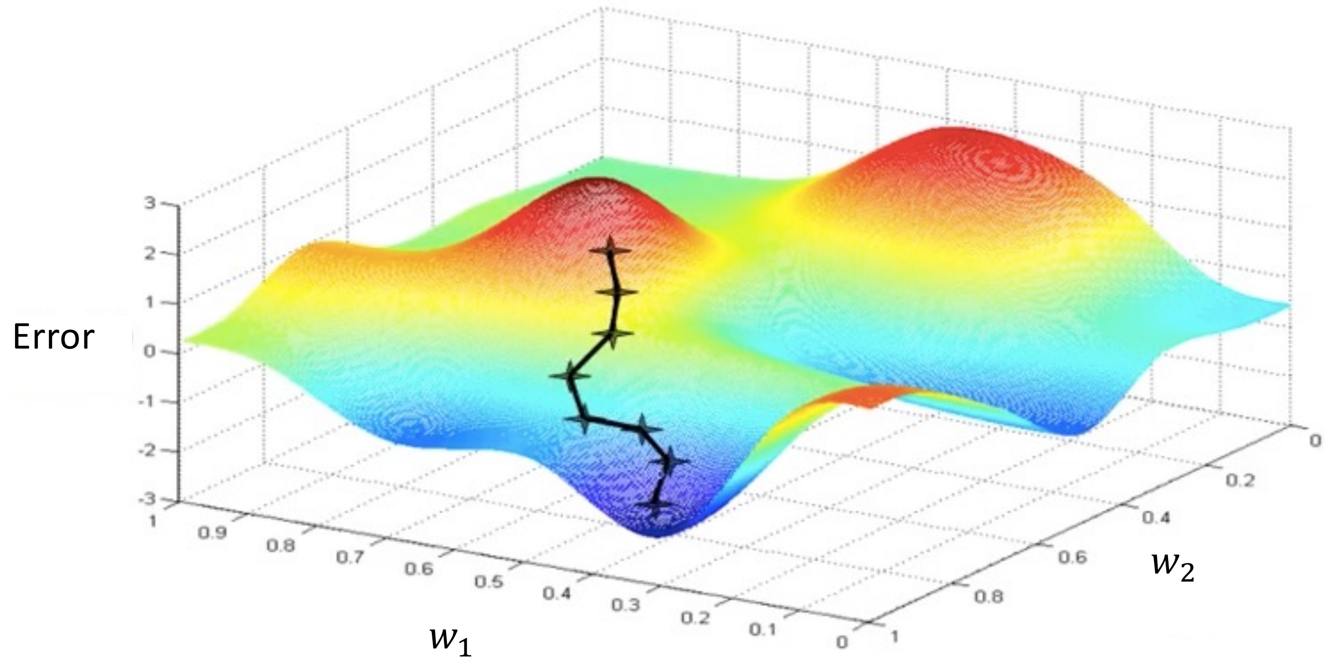
Learning for a perceptron

	Vector notation	Scalar notation
Error	$E(\mathbf{w}) = \frac{1}{2}(y - c)(\mathbf{w} \cdot \mathbf{x})$	$E(w) = \frac{1}{2}(y - c)(w_0 + w_1 x_1 + \ldots + w_d x_d)$
Gradient	$\nabla E(\mathbf{w}) = \frac{1}{2}(y - c)\mathbf{x}$	$\frac{\partial E(w)}{\partial w_i} = \frac{1}{2}(y - c)x_i$
Gradient descent Learning Rule	$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \frac{1}{2}(y - c)\mathbf{x}$	$w_i(t+1) = w_i(t) - \eta \frac{1}{2}(y - c)x_i$

On the error surface, each new weight is directly *downhill* from the old weight

η is how much to change in that direction

Visualisation of gradient descent

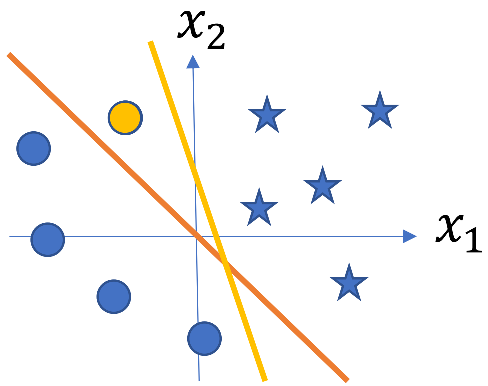


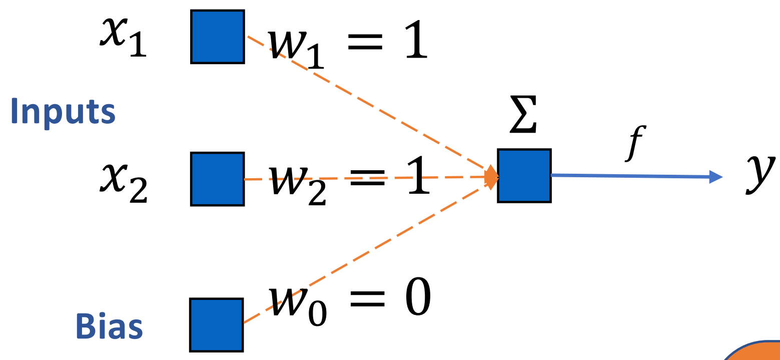
$$\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \nabla E(\mathbf{w})$$

[in reality there is w0 as well, but you can only picture the error surface for two parameters!]

Hyperparameters / settings for training a perceptron

- Set **learning rate**: η
- Set initial **weight values**: $\mathbf{w}$
- When to stop?
  - Training set shown repeatedly until **stopping criteria** are met e.g., the error drops below a threshold or plateaus
  - Note, each full presentation of all patterns := '*epoch*'
- Which type of **training regime**?
  - *Sequential* (on-line, stochastic, or per-pattern): Weights updated after each pattern is presented.
  - *Batch*: Calculate the derivatives/weight changes for each pattern in the training set. Calculate total change by summing individual changes.





Decision boundary where:
$w_0 + w_1 x_1 + w_2 x_2 > 0$
→ get $y = 1$

For this parameter initialization, the decision boundary is the straight line:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$$

i.e. $x_2 = -x_1$

We want:
$y = 1$ for ⭐ stars and
$y = -1$ for 🟡 circles

Present: $(x_1 = -1, \quad x_2 = 2)$
→ get $y = 1$, which is **wrong**!

$$E(w) = \frac{1}{2}(w_0 + w_1 x_1 + w_2 x_2)(y - c)$$
$$= \frac{1}{2}(0 - 1 + 2)(1 - -1) = 1$$

Apply the learning rule:

$$w_i \to w_i - \eta \frac{\partial E}{\partial w_i}$$

Differentiate error w.r.t. each weight…

$$\frac{\partial E}{\partial w_0} = 1$$
$$\frac{\partial E}{\partial w_1} = x_1 = -1$$
$$\frac{\partial E}{\partial w_2} = x_2 = 2$$

$w_0 \to w_0 - \eta$ w_0 ⬇ decreases

$w_1 \to w_1 + \eta$ w_1 ⬆ increases

$w_2 \to w_2 - 2\eta$ w_2 ⬇ decreases

➡ Decision boundary gets steeper and moves up (if η is not too big).

Learning outcomes for today's main topic: *Clustering*.

- Understand first unsupervised learning problem: clustering
- Understand the steps of the *k*-means algorithm

Clustering

- Learn a model from *unlabelled* instances $x_1, x_2, \ldots, x_N$

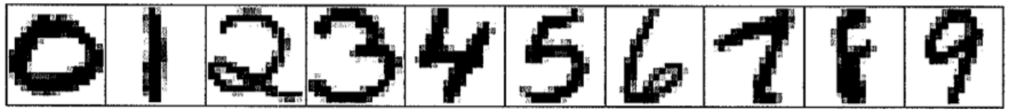
- Learning is unsupervised (we have no labelling function, beforehand): Requires data, but no labels

- Detects patterns / discover structure in the data
  - e.g. in customer shopping behaviours, search results, regions of images, etc.

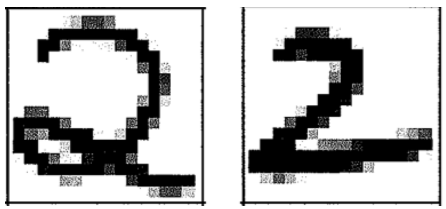
- Useful when don’t know what you’re looking for

Clustering vs. Classification

In classification, we have data for which the groups are known, and we try to learn what differentiates these groups to properly classify future data.



In clustering, we look at data for which groups are unknown and undefined, and try to learn the groups themselves, as well as what differentiates them.



When poll is active, respond at **pollev.com/bdevans**
Text **BDEVANS** to **07480 781235** once to join

How many clusters are there?

2

3

4

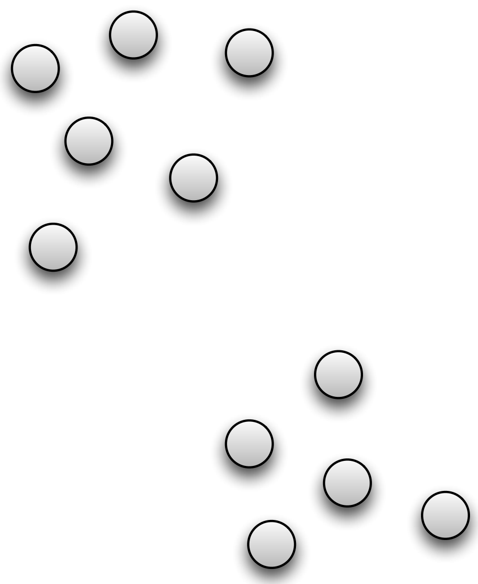
27

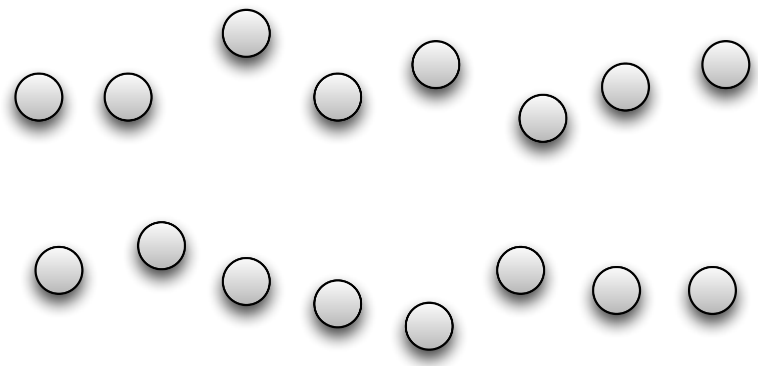
None of the above

Clustering

- Basic idea: group together similar instances

- Example 2-D Point patterns

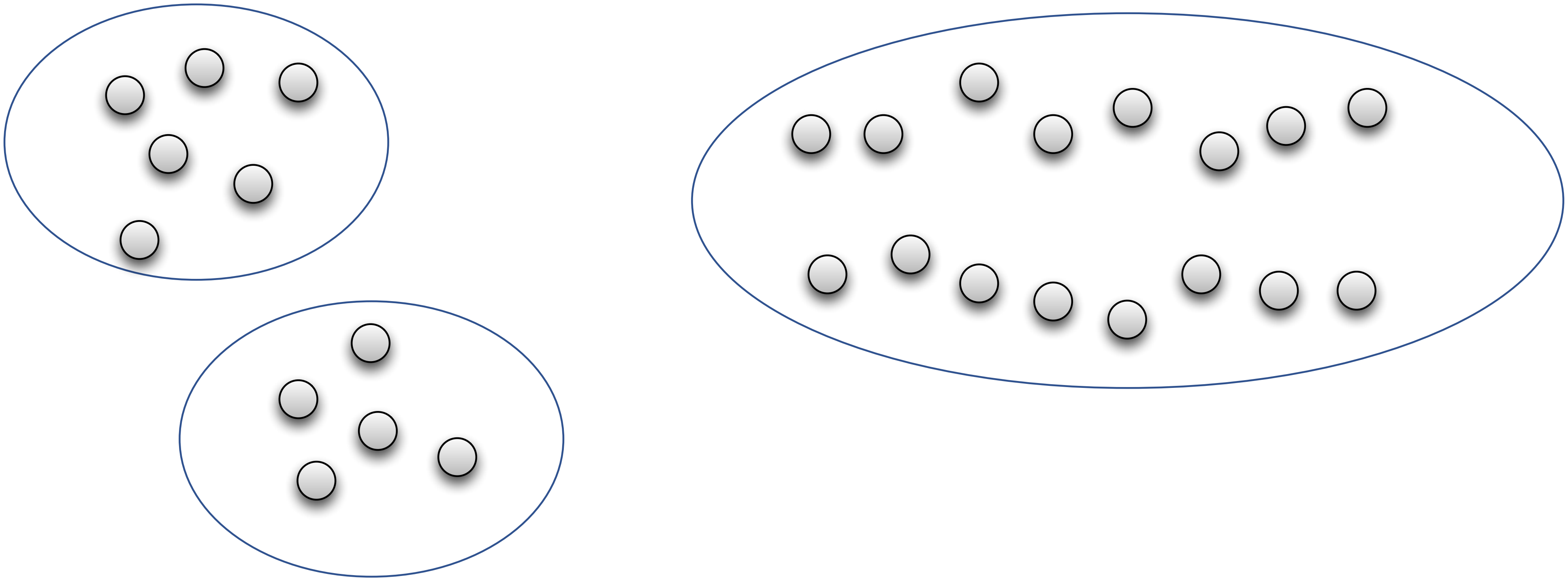




Clustering

- Basic idea: group together similar instances

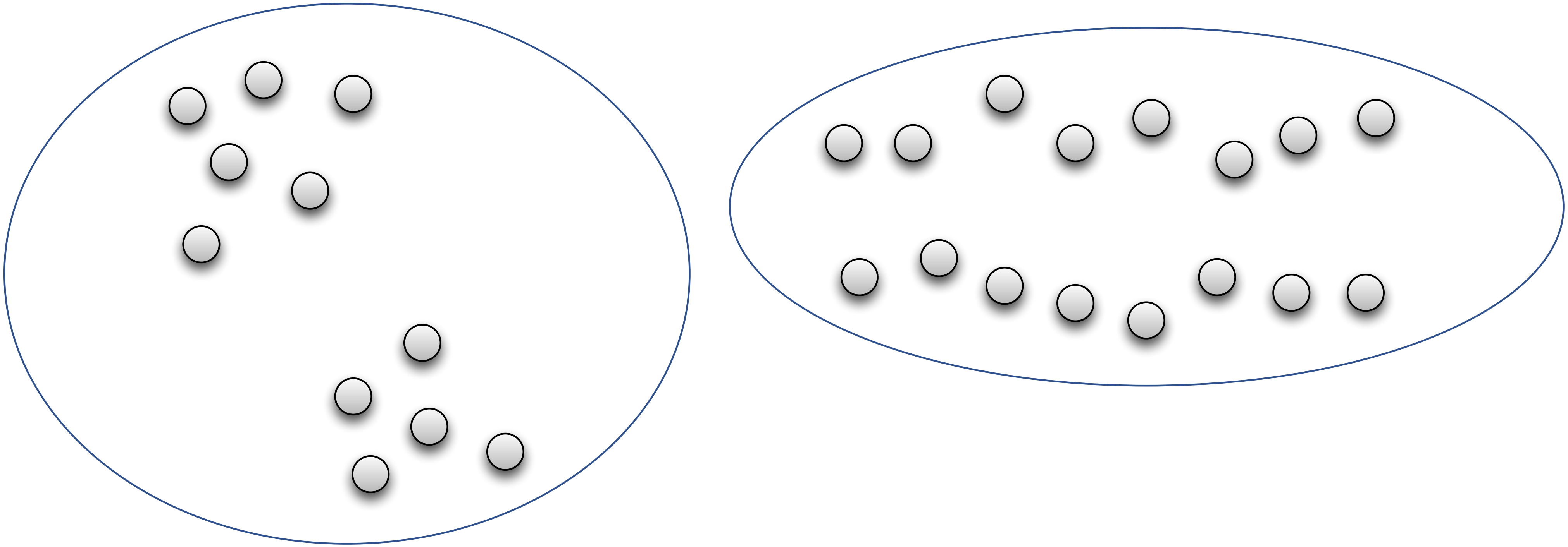
- Example 2-D Point patterns



Clustering

- Basic idea: group together similar instances

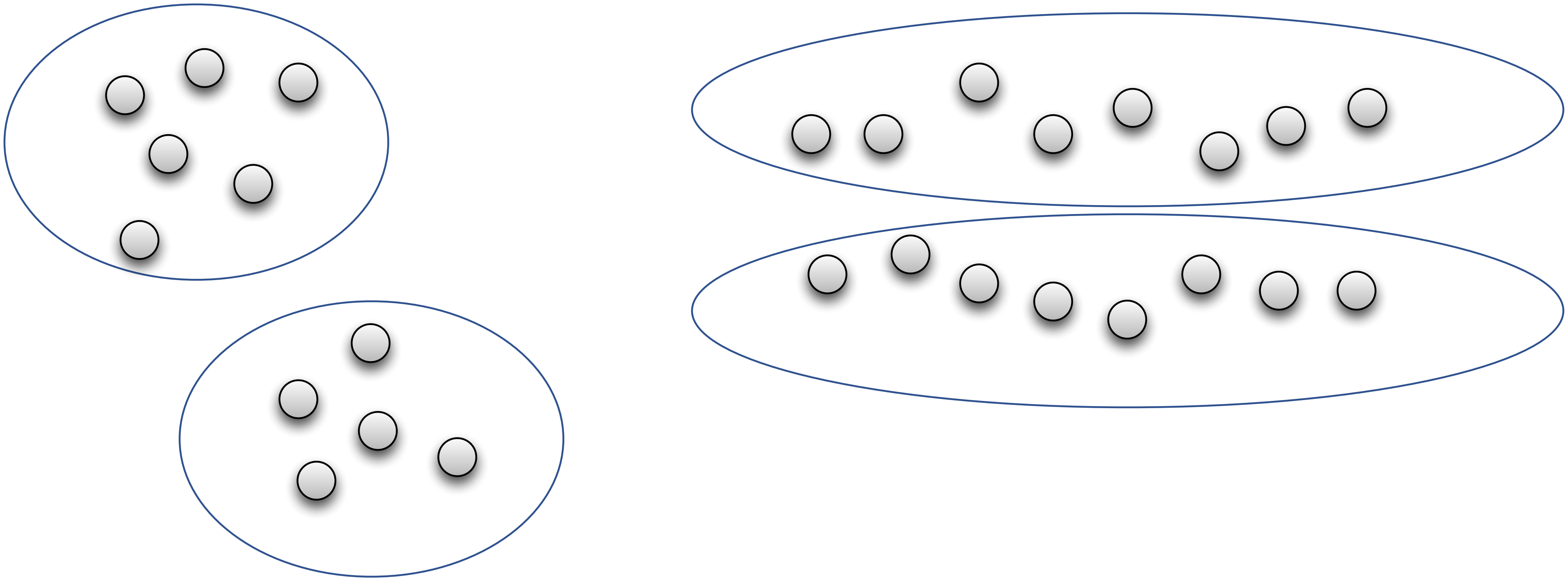
- Example 2-D Point patterns



Clustering

- Basic idea: group together similar instances

- Example 2-D Point patterns



Clustering

- What could "similar" mean?

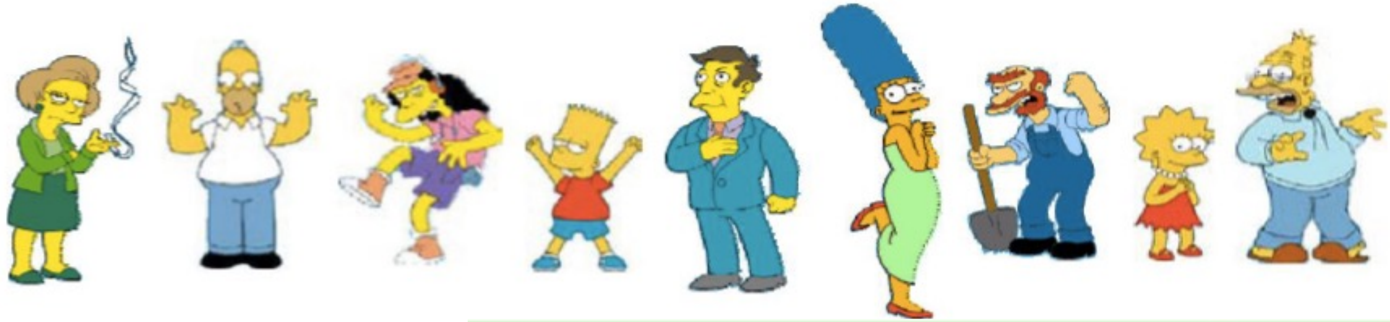
  ○ One option: small squared Euclidean distance

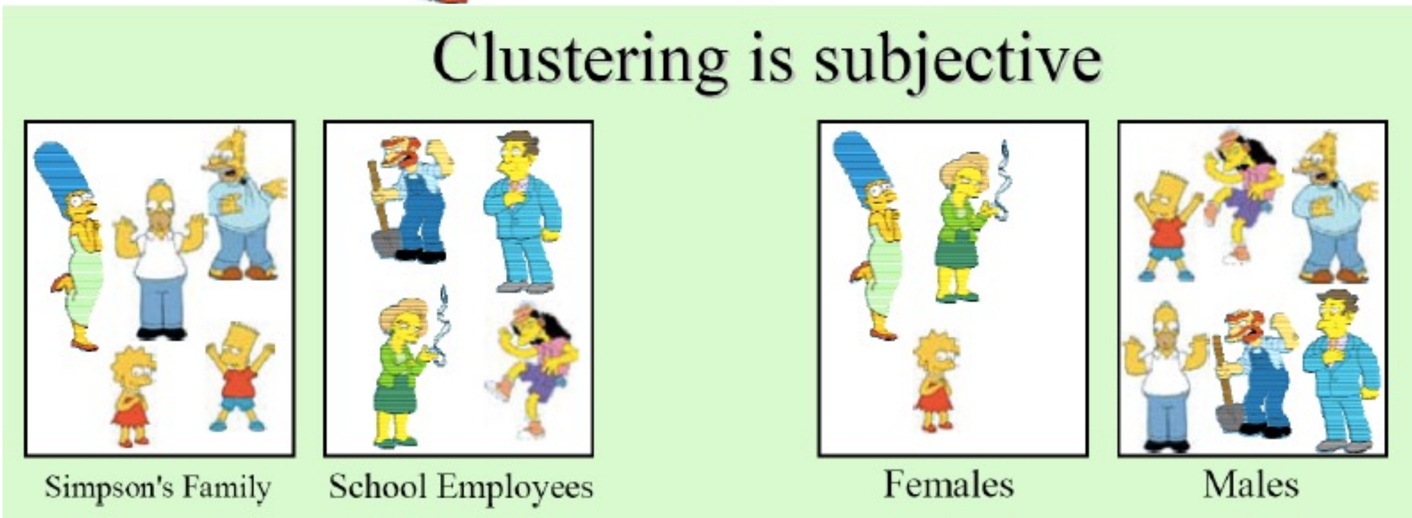
$$Dis(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|^2 = [\ (x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_d - y_d)^2\ ]$$

where ||·|| denotes the Euclidean distance

- Clustering results are crucially dependent on the measure of similarity (or distance) between "points" to be clustered

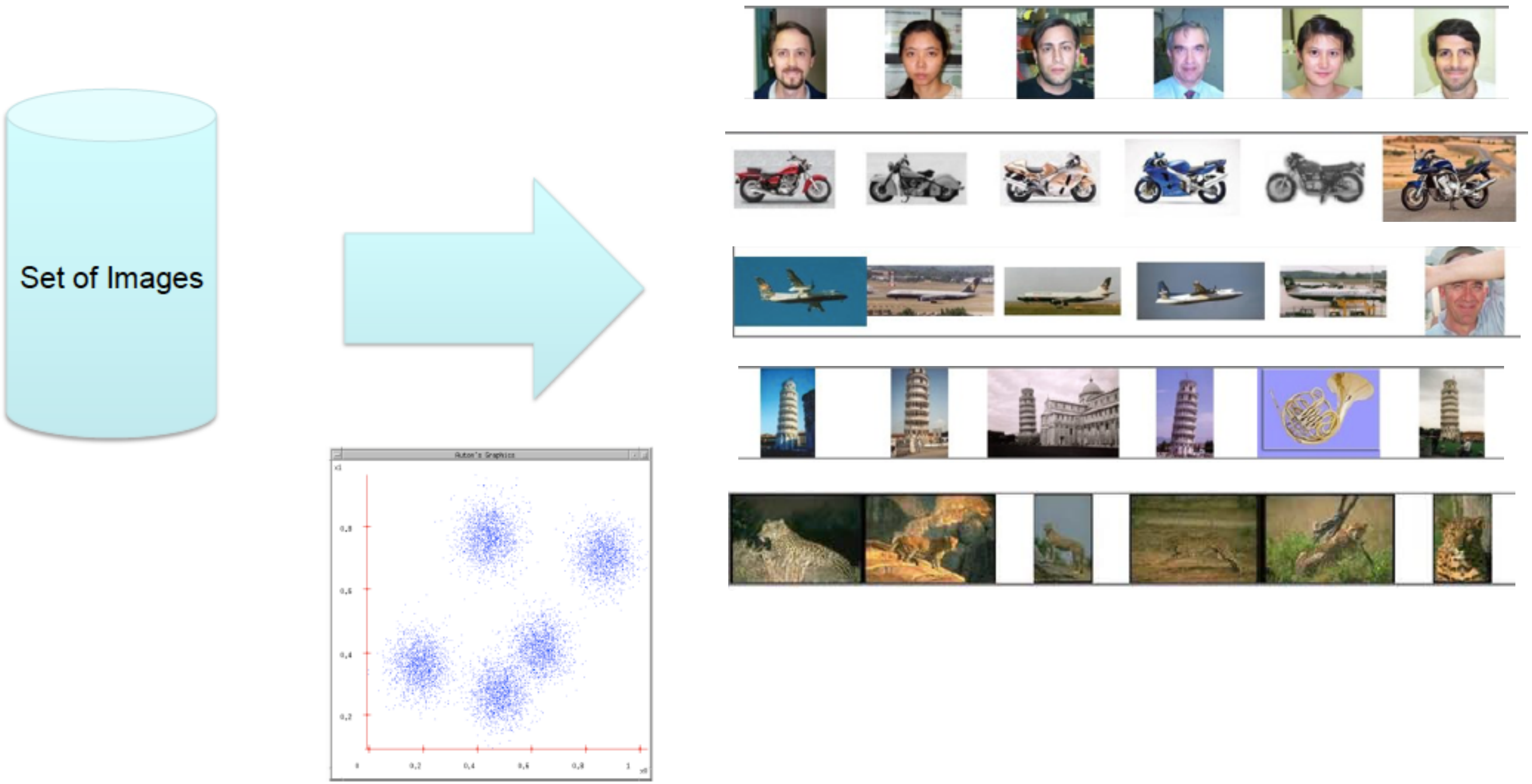
What is similar/dissimilar?





You pick your similarity/dissimilarity

Clustering examples



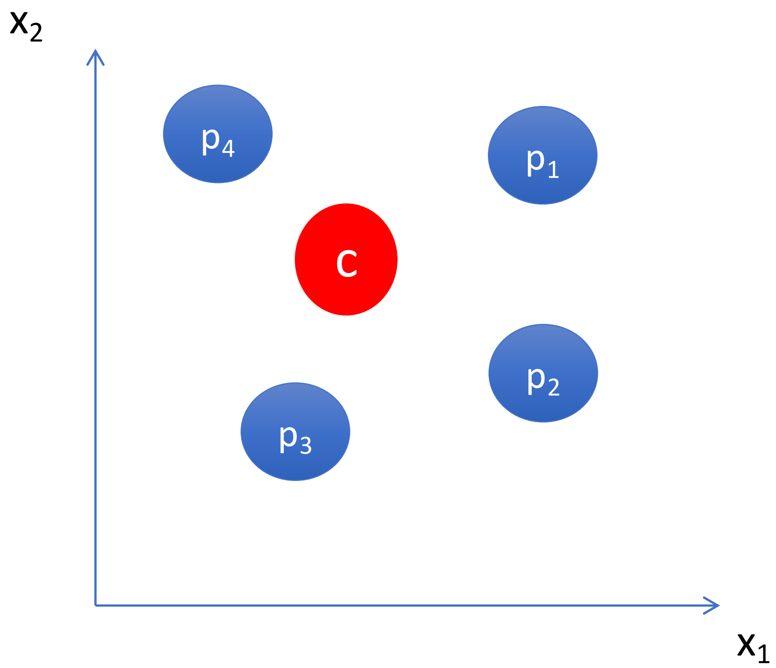
Clustering algorithms

- Partition algorithms
  - No hierarchy

- Hierarchical algorithms:
  - Bottom up – *agglomerative* (“merging”)
  - Top down – *divisive* (“splitting”)

k-means

- Simple partitioning approach based on the idea of centroids or prototypes

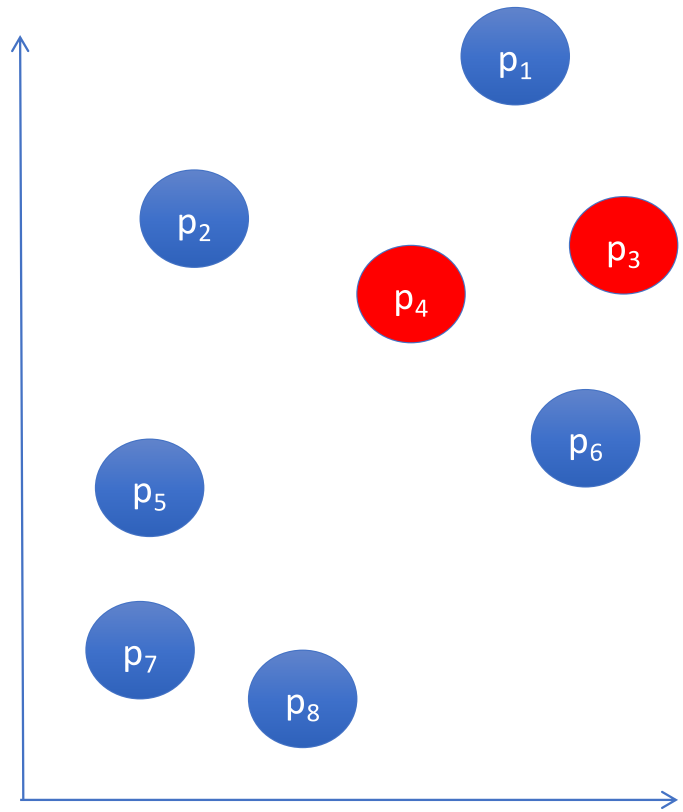


The centroid (middle) of any group of points in a Euclidean space can be found by taking the mean on every dimension.

$$C_j = \frac{\sum_{i=0}^{n} feature\ j\ of\ p_i}{n}$$

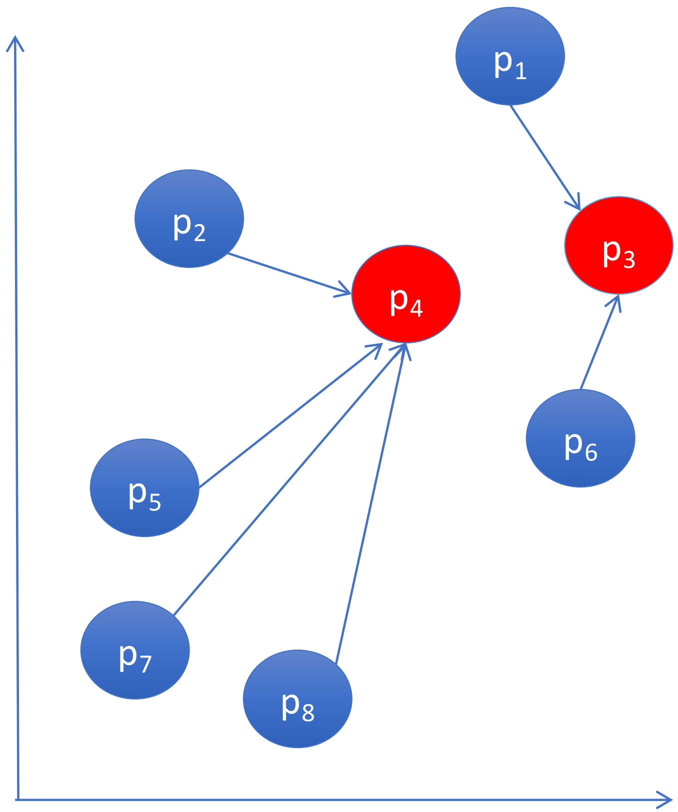
k-means

1. Select k points as initial centroids
2. while centroids changing:
    1. Form k clusters by assigning each point to its nearest centroid
    2. Recompute centroid of the cluster



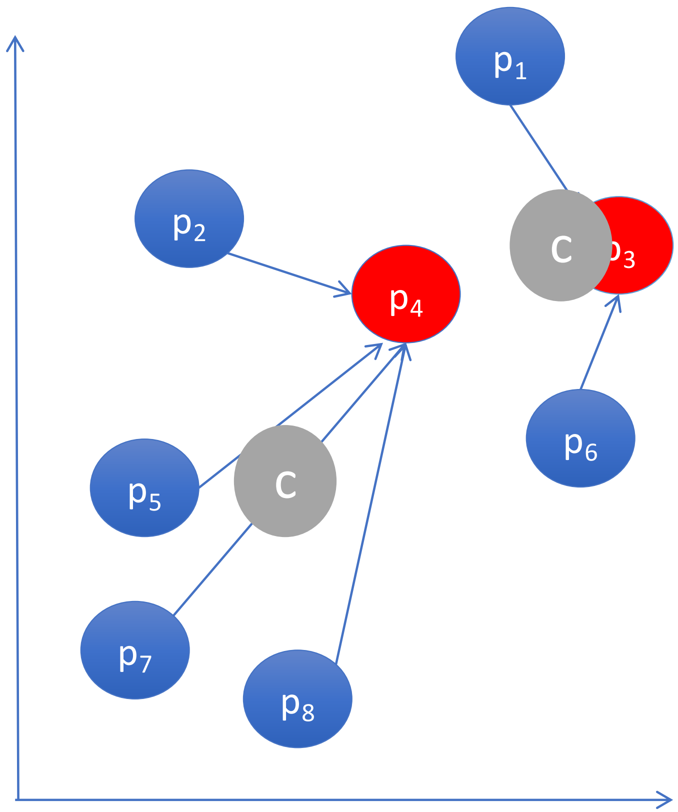
k-means

1. Select k points as initial centroids
2. while centroids changing:
    1. Form k clusters by assigning each point to its nearest centroid
    2. Recompute centroid of the cluster



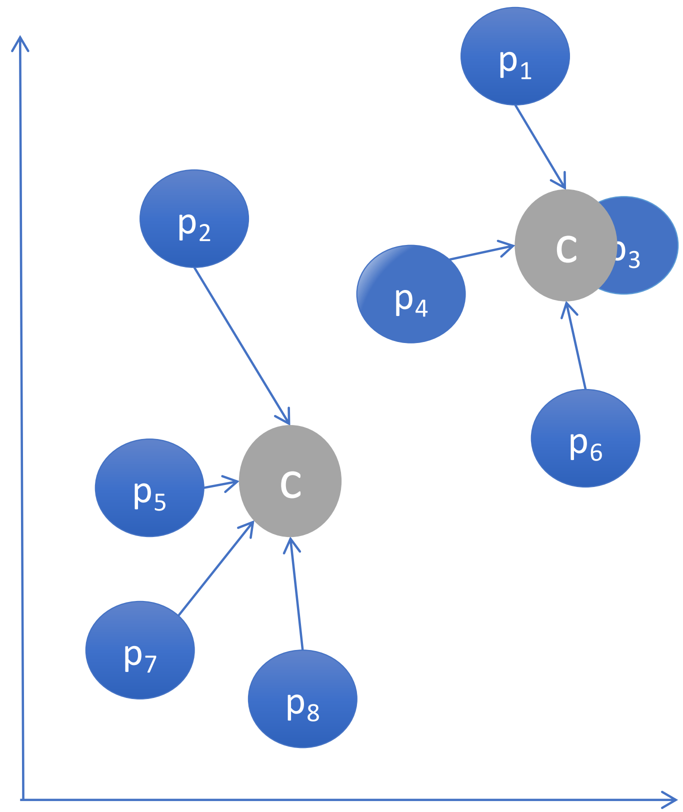
k-means

1. Select k points as initial centroids
2. while centroids changing:
    1. Form k clusters by assigning each point to its nearest centroid
    2. Recompute centroid of the cluster



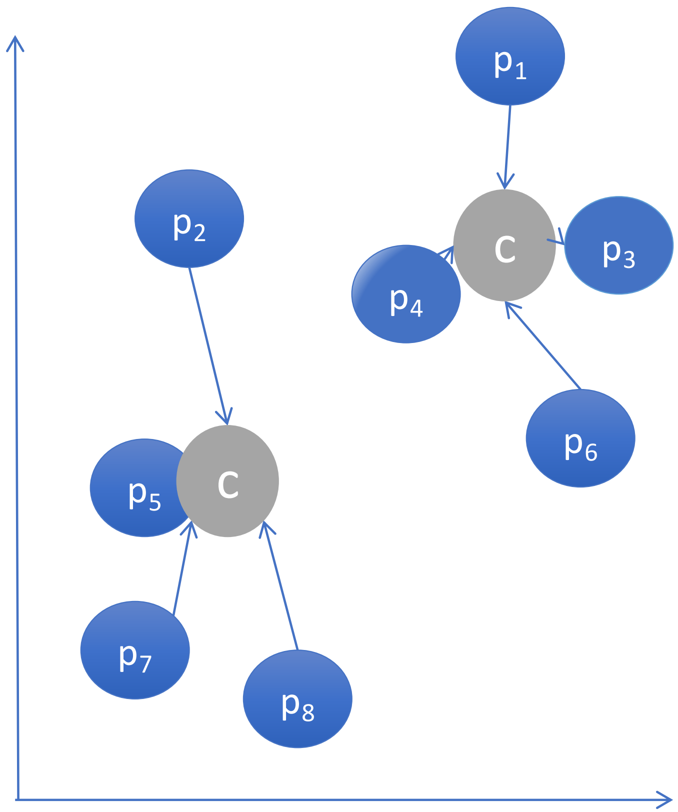
k-means

1. Select k points as initial centroids
2. while centroids changing:
    1. Form k clusters by assigning each point to its nearest centroid
    2. Recompute centroid of the cluster



k-means

1. Select k points as initial centroids
2. while centroids changing:
    1. Form k clusters by assigning each point to its nearest centroid
    2. Recompute centroid of the cluster



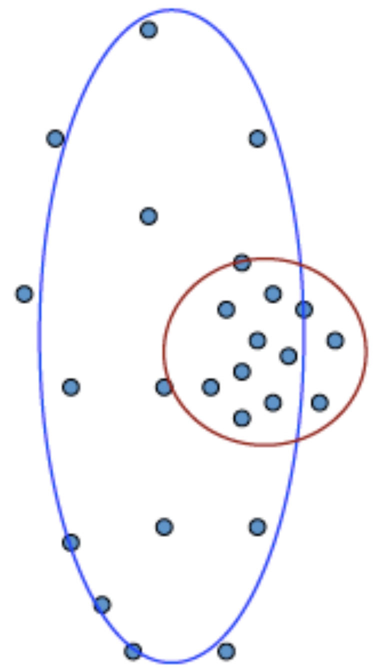
Pros and cons of *k*-means

Advantages

- Fast
  - ➢ Don't have to keep computing distances between all pairs of points.
- Simple to implement
- Intuitive

Disadvantages

- Need to choose/know *k* in advance
  - ➢ Could try out several different *k* and select the best?
- May converge on a local mimimum i.e., suboptimal clustering
  - ➢ Can be overcome to some extent by repeated random initialisations
- Hard clustering (each point is only assigned to a single cluster)
- Flat structure (no clusters within clusters)



What is *k*-means optimizing?

- Given data points X the goal is to choose *k* centroids and cluster assignments so that the average distance from centroids is minimised.

- That is minimise:

$$\sum_{i=1}^{N} \|x_i - C_i\|^2$$

with respect to all centroids and allocations (for a given *k*).

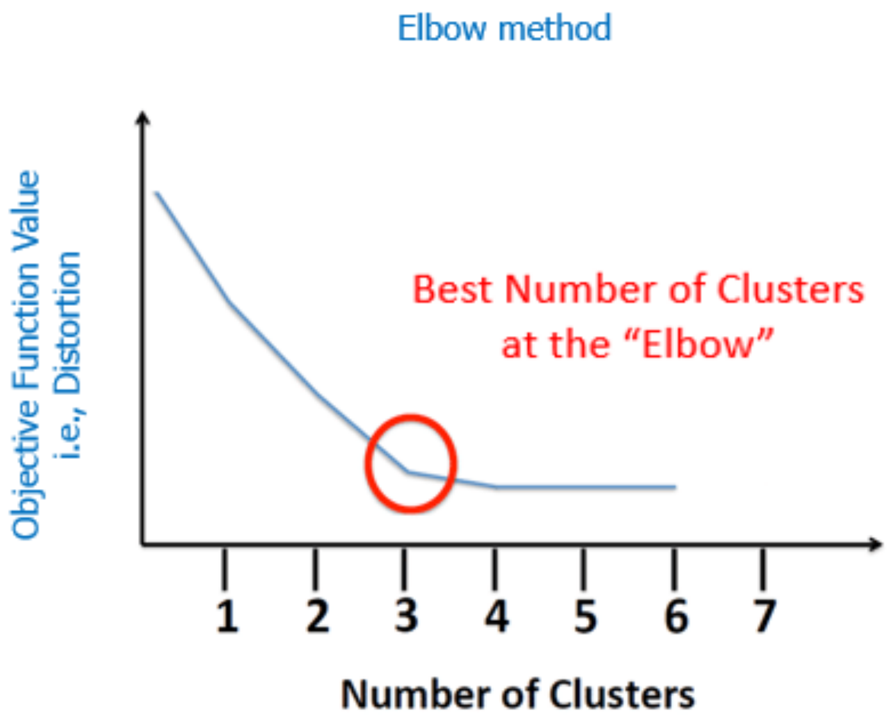
How many clusters?

- Minimize sum of distances to centroids? Called distortion or inertia:

  Distortion $=\sum_{i=1}^{N} ||x_i - C_i||^2$

- If you have N clusters, every point is at the centroid of its own cluster containing just itself.

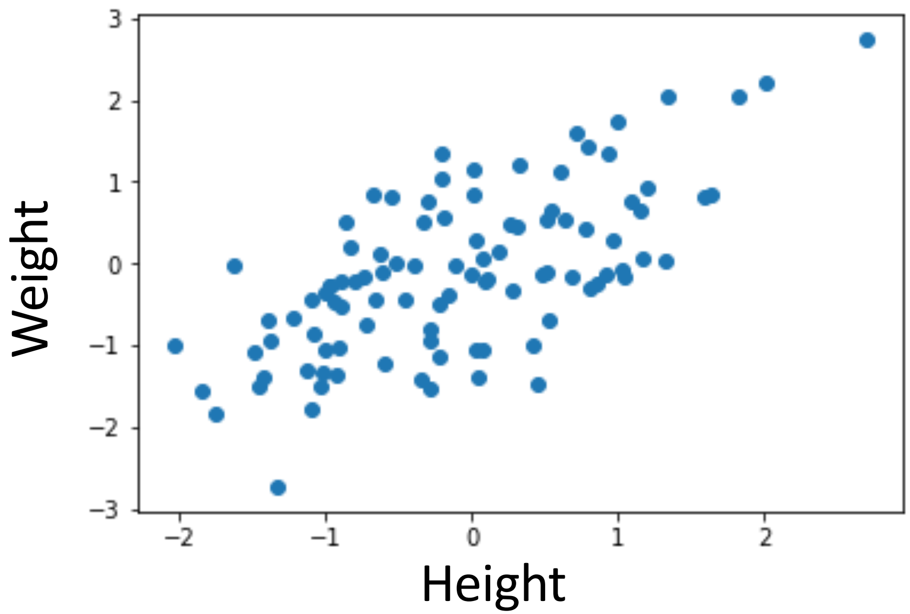
- Looking for substantial gain in having more clusters.



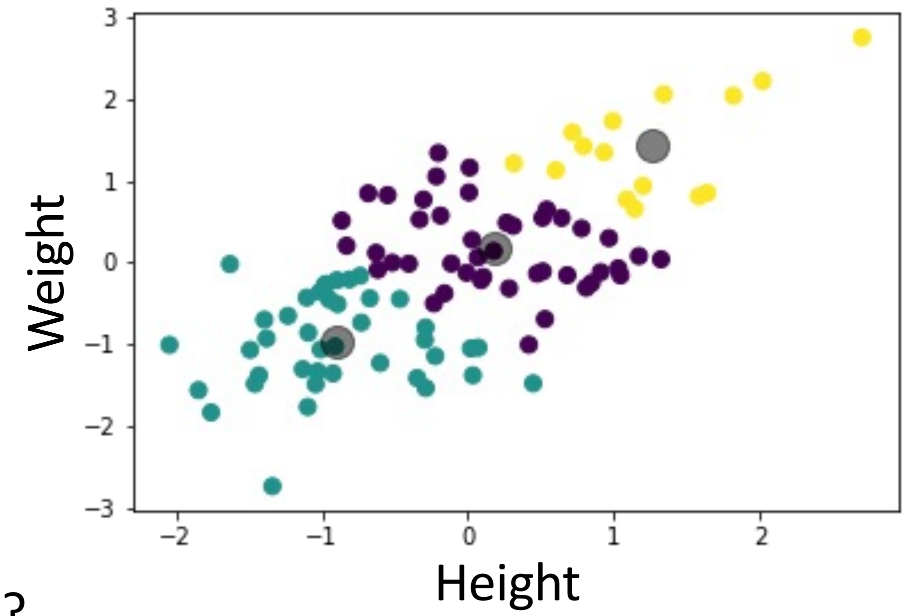
- E.g. Data are accident prone areas, centroids are where to put Hospital Emergency Units.

Example: T-shirts

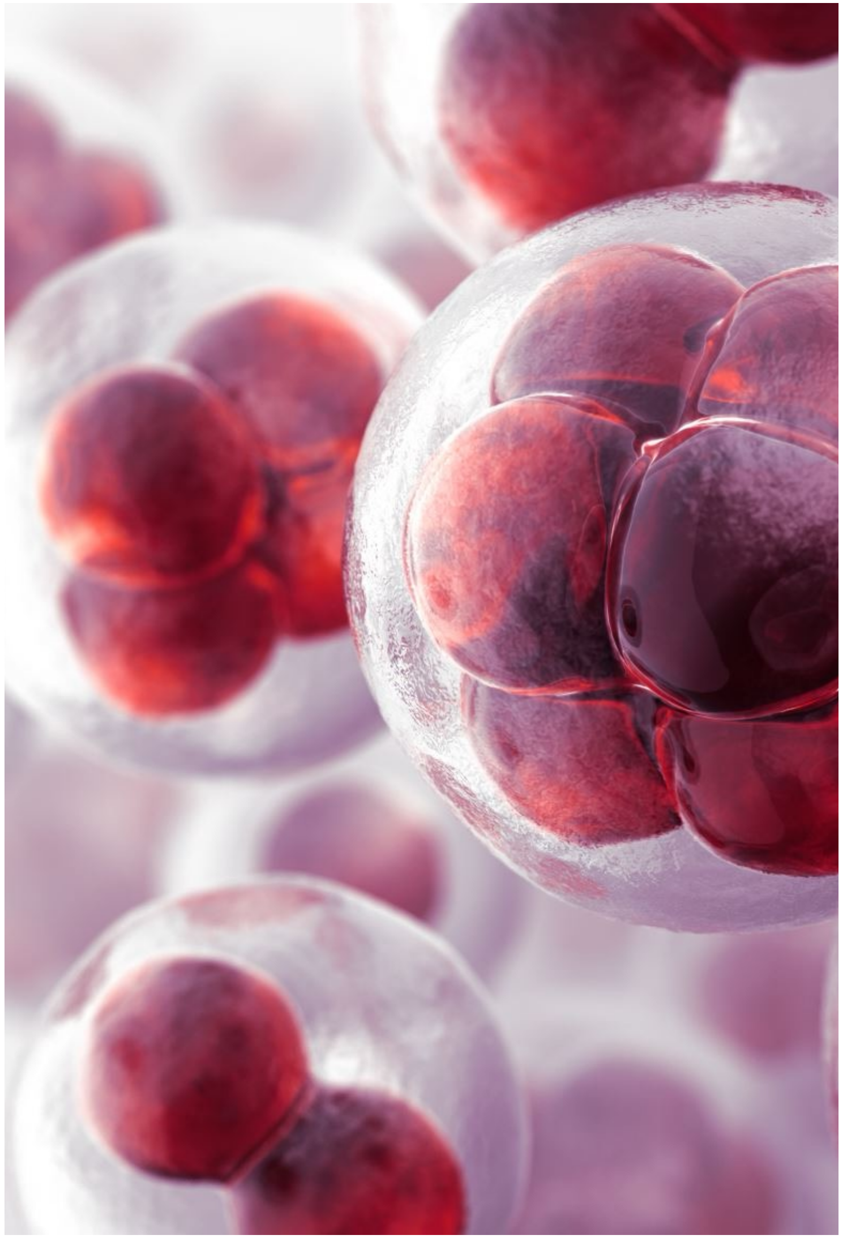
- Suppose I am a manufacturer of t-shirts and I want to make 3 different sizes. How do I optimise the sizes?



Example: T-shirts



- But is this optimal?
- Have to consider many factors...

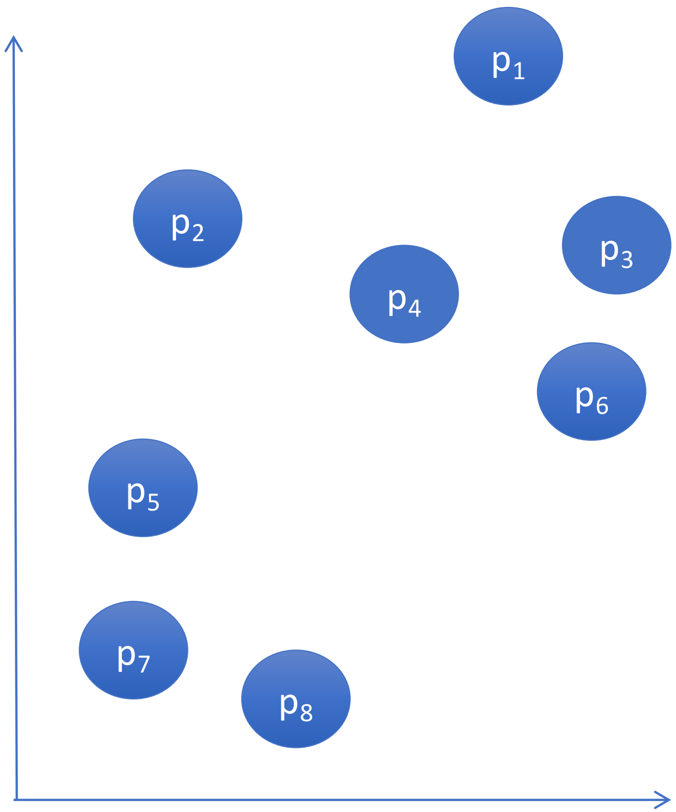


Extra material: Agglomerative hierarchical clustering

- AHC is an agglomerative technique which builds up clusters by repeatedly merging the closest pair of clusters
- Do not need to know the number of clusters in advance
- Hierarchical (clusters have internal structure)

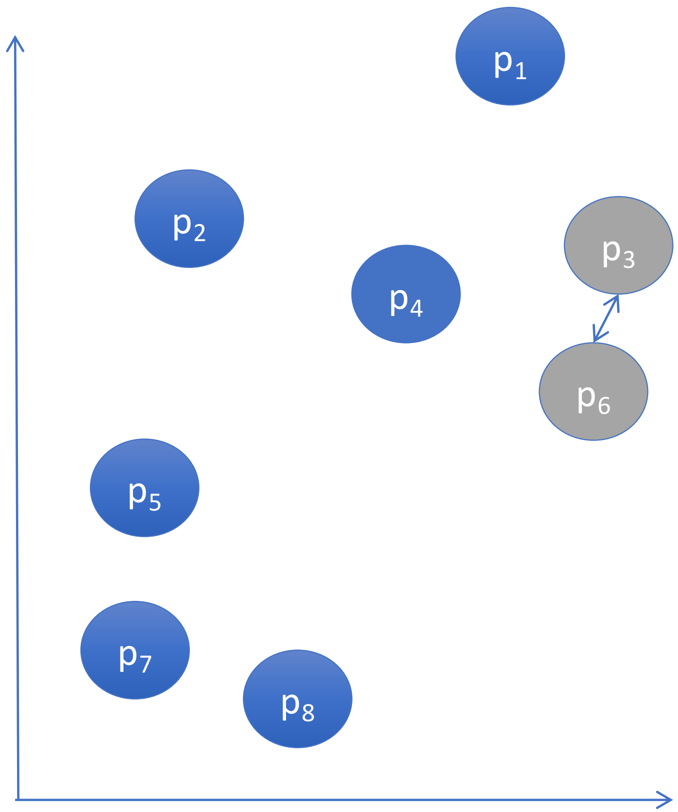
Agglomerative hierarchical clustering

1. Initialise n clusters as the n data points
2. Find closest pair <p_i,p_j> of clusters with distance d
3. while d < threshold:
   1. Merge clusters <p_i,p_j>
   2. Find closest pair <p_i, p_j> with distance d



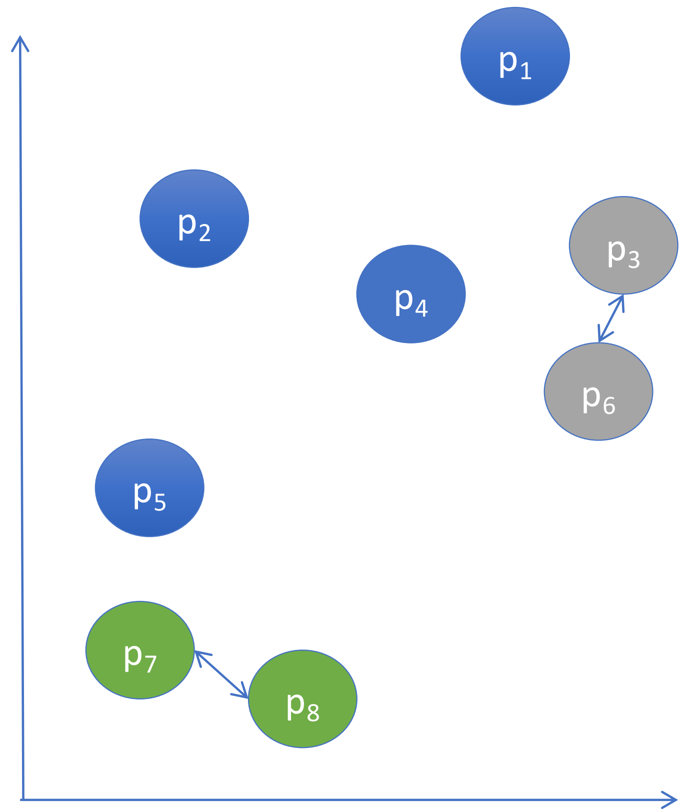
Agglomerative hierarchical clustering

1. Initialise n clusters as the n data points
2. Find closest pair $<p_i,p_j>$ of clusters with distance d
3. while d < threshold:
    1. Merge clusters $<p_i,p_j>$
    2. Find closest pair $<p_i, p_j>$ with distance d



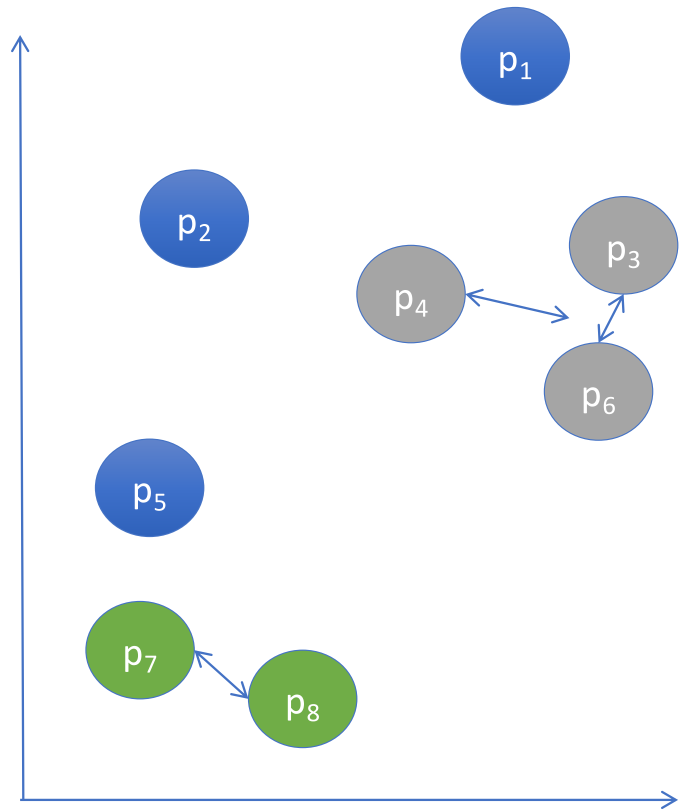
Agglomerative hierarchical clustering

1. Initialise n clusters as the n data points
2. Find closest pair <p_i,p_j> of clusters with distance d
3. while d < threshold:
    1. Merge clusters <p_i,p_j>
    2. Find closest pair <p_i, p_j> with distance d



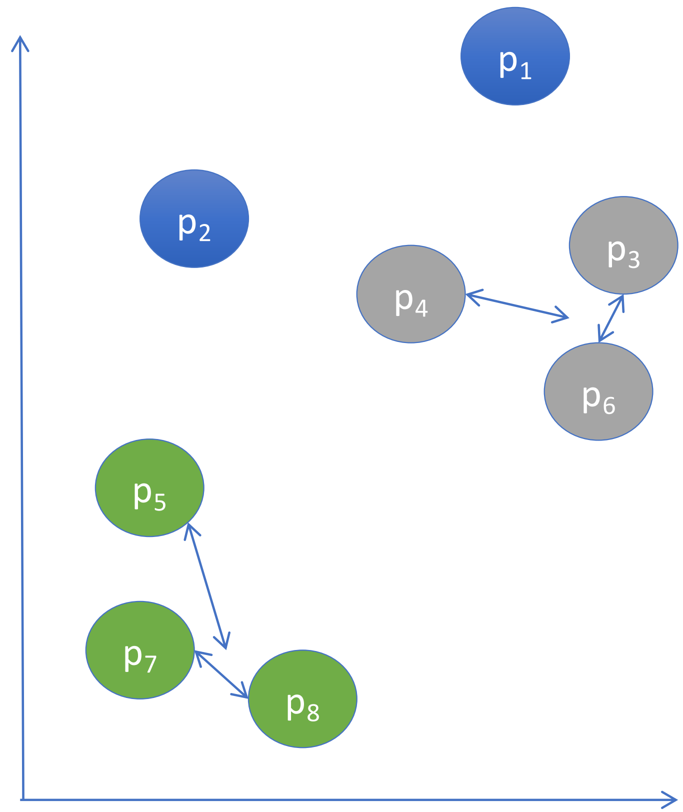
Agglomerative hierarchical clustering

1. Initialise n clusters as the n data points
2. Find closest pair $<p_i,p_j>$ of clusters with distance d
3. while d < threshold:
    1. Merge clusters $<p_i,p_j>$
    2. Find closest pair $<p_i, p_j>$ with distance d



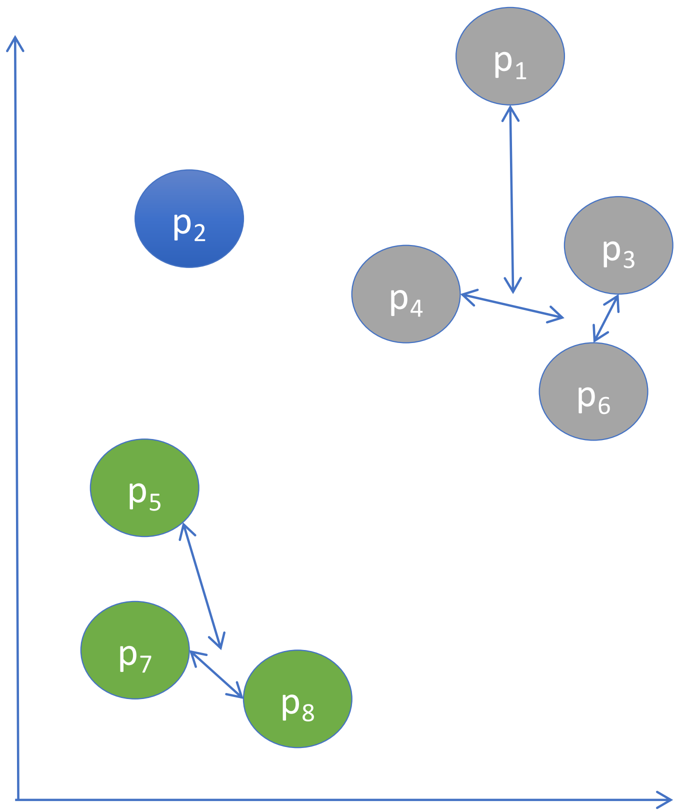
Agglomerative hierarchical clustering

1. Initialise n clusters as the n data points
2. Find closest pair <p_i,p_j> of clusters with distance d
3. while d < threshold:
   1. Merge clusters <p_i,p_j>
   2. Find closest pair <p_i, p_j> with distance d



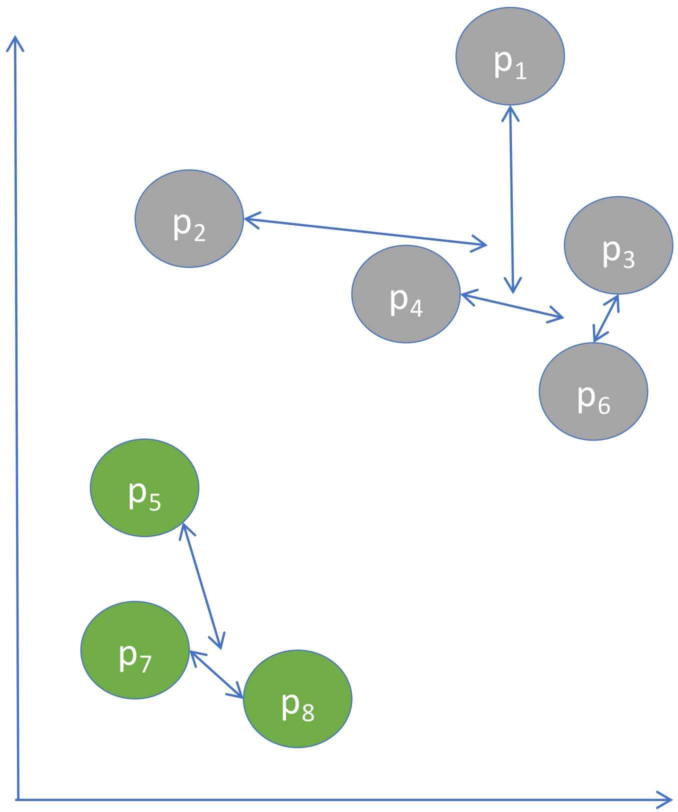
Agglomerative hierarchical clustering

1. Initialise n clusters as the n data points
2. Find closest pair $<p_i,p_j>$ of clusters with distance d
3. while d < threshold:
    1. Merge clusters $<p_i,p_j>$
    2. Find closest pair $<p_i, p_j>$ with distance d



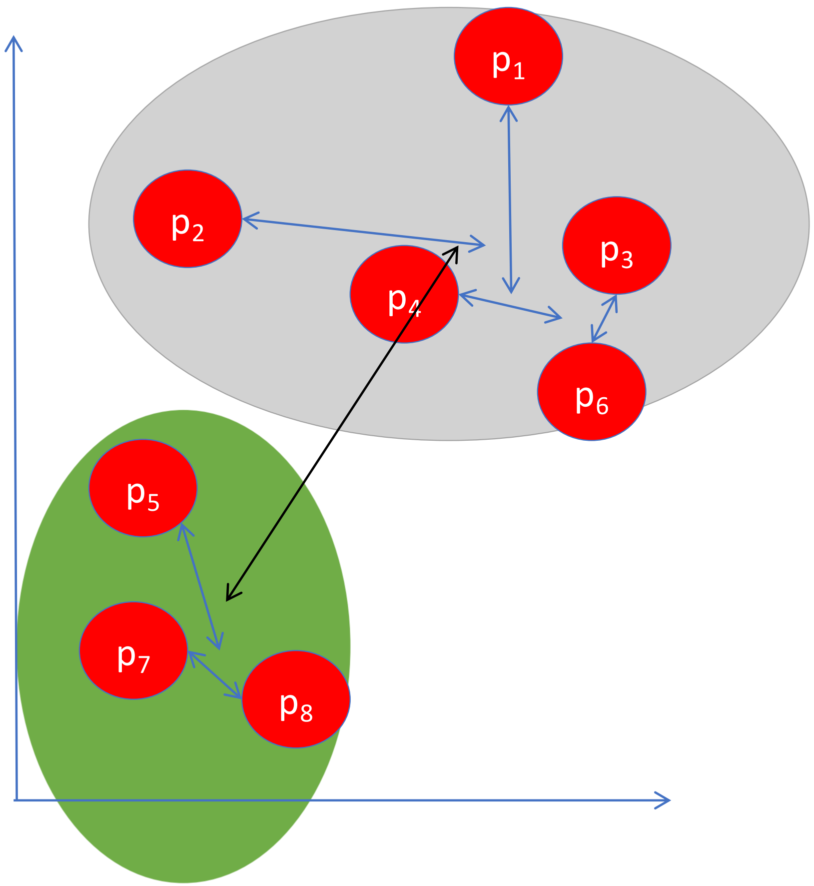
Agglomerative hierarchical clustering

1. Initialise n clusters as the n data points
2. Find closest pair <p_i,p_j> of clusters with distance d
3. while d < threshold:
   1. Merge clusters <p_i,p_j>
   2. Find closest pair <p_i, p_j> with distance d



Agglomerative hierarchical clustering

1. Initialise n clusters as the n data points
2. Find closest pair $<p_i,p_j>$ of clusters with distance d
3. while d < threshold:
    1. Merge clusters $<p_i,p_j>$
    2. Find closest pair $<p_i, p_j>$ with distance d



Example: teacher dividing up a class of students

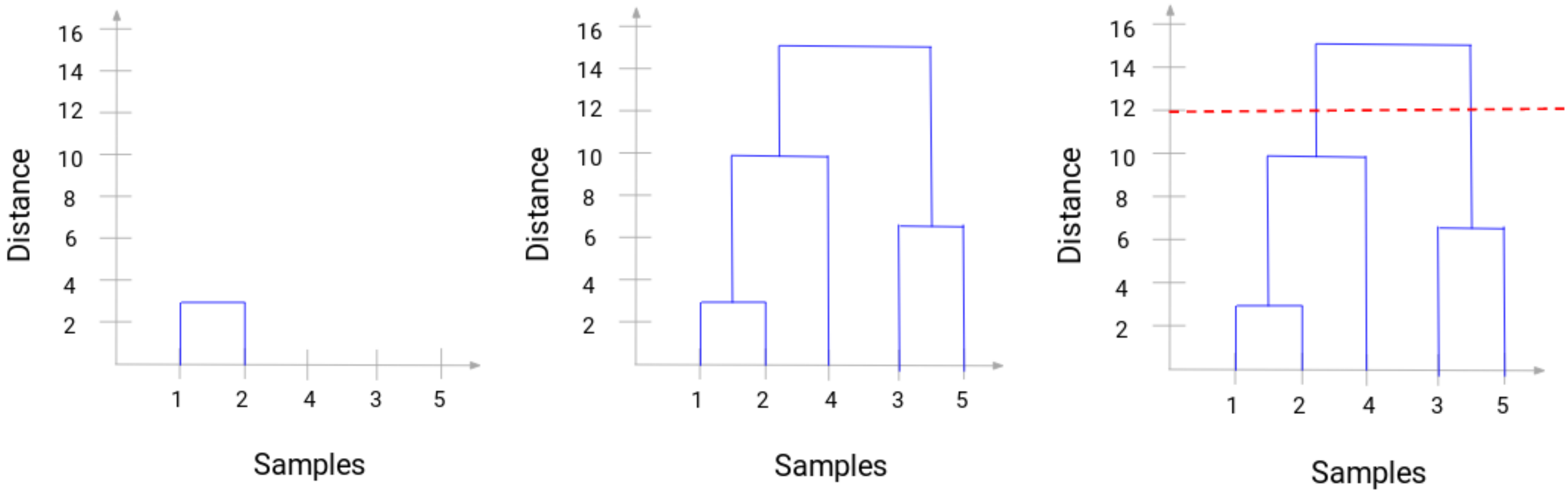
Student_ID	Marks
1	10
2	7
3	28
4	20
5	35

ID	1	2	3	4	5
1	0	3	18	10	25
2	3	0	21	13	28
3	18	21	0	8	7
4	10	13	8	0	15
5	25	28	7	15	0

Student_ID	Marks
(1,2)	10
3	28
4	20
5	35

ID	(1,2)	3	4	5
(1,2)	0	18	10	25
3	18	0	8	7
4	10	8	0	15
5	25	7	15	0

Build the dendrogram



- If you're lucky this will tell you how many clusters to have.

Agglomerative hierarchical clustering versus *k*-means

- *k*-means is quicker – just go through all N data points and calculate distances from centroids.
- Agglomerative hierarchical clustering – need similarity of all pairs, $N \times N$ similarity calculations each iteration. If N is large, this is a lot bigger!
- But with AHC, get a thorough snap-shot of data from one pass of algorithm, and don't need to specify number of clusters.

What have you learned today?

- Unsupervised learning.
  - Clustering.
    - *k*-means.
    - Agglomerative hierarchical
      - Dendrograms show the hierarchy and possibilities for cuts

- Next week: reading / catch up week.
  - Tuesday: No lecture.
  - Thursday: optional MCQ revision lecture.
  - Labs as usual (for this week's content)

Week 7: Pre-processing.

Week 8: Neural networks II.