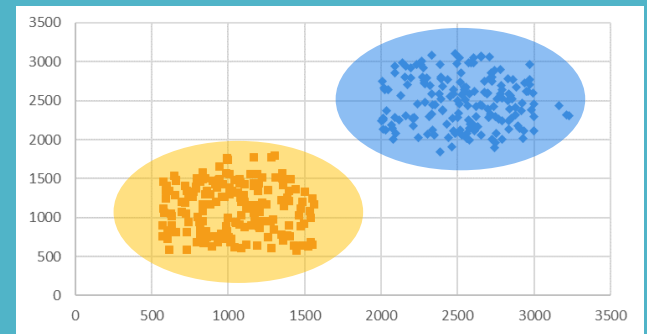
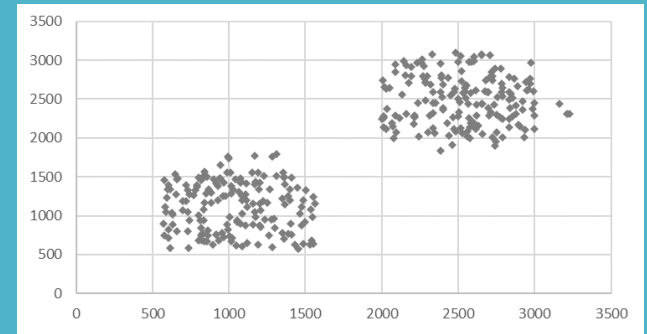


# Clustering



Dr. Jason Lines  
j.lines@uea.ac.uk

# Clustering

Clustering describes a number of methods that are used to group together records that are 'similar' and separate out records that are 'different'

Normally the data are unlabelled, i.e. no class labels, so clustering is an example of **unsupervised learning**

There is no notion of what is “correct” in advance: results are subjective and the algorithm has to determine what data to group together – i.e. determining similarity is key

A **cluster** is therefore a collection of data that are similar (which may implicitly define a class) and are dissimilar to data in other clusters.

# Examples of Clustering

## Business and Marketing

Grouping together customers with similar buying behaviours for targeted marketing

## Biology

Many applications, such as sequencing analysis and grouping together species based on their genetic makeup

## Medicine

Grouping together test results can lead to spotting anomalous results that may indicate illness

## Insurance

Estimating risk for new customers when creating policy quotes

## Network Security

Identifying intrusions based on patterns of network behaviour

And many more!

# How-to Clustering

There are a number of important considerations when clustering new data – and there are various clustering algorithms that we could use

But before we even decide which algorithm to use:

- How do we measure **similarity** between objects so that we can group/separate them?
- What if the data is not prepared in a suitable manner and similarity measures are biased by a subset of the attributes?

Both of these are important factors when clustering data so we must address them before discussing any of the algorithms

# How-to Clustering

To understand the whole picture we will talk look at:

1. Defining and measuring similarity
2. Data normalisation
3. Clustering Algorithms
  - a) Partitional
  - b) Hierarchical
  - c) Density
4. Further considerations

# Similarity

A critical part of any clustering algorithm is being able to determine how similar two instances of data are

However, similarity can be subjective and hard to define

The quality or state of being similar; likeness; resemblance; as, a similarity of features.

– **Webster's Dictionary**

Sometimes it's easy to describe similarity:



# Similarity

Sometimes similarity is more difficult to describe...



Similarity is hard to define, *but we know it when we see it*

The real meaning is philosophical, but we will take a more pragmatic approach

**Definition:** given objects A and B, the distance (i.e. dissimilarity) between two objects can be denoted as:  $D(A,B)$

There are certain rules that D should follow for clustering:

$$D(A,B) = D(B,A)$$

*Symmetry*

*Otherwise you could claim "Alex looks like Bob, but Bob looks nothing like Alex."*

$$D(A,A) = 0$$

*Constancy of Self-Similarity*

*Otherwise you could claim "Alex looks more like Bob, than Bob does."*

$$D(A,B) = 0 \text{ if } A=B$$

*Positivity (Separation)*

*Otherwise there are objects in your world that are different, but you cannot tell apart.*

$$D(A,B) \leq D(A,C) + D(B,C)$$

*Triangular Inequality*

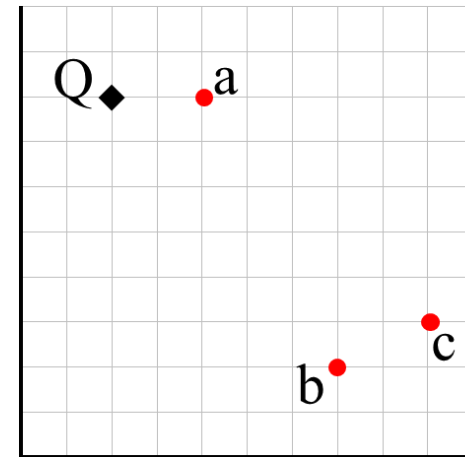
*Otherwise you could claim "Alex is very like Bob, and Alex is very like Carl, but Bob is very unlike Carl."*

# Why is Triangular Inequality so Important?

Virtually all techniques to index data require the triangular inequality to hold. Consider this example:

We have a query object,  $Q$ , and a dataset with 3 objects in it  $\{a, b, c\}$

We need to find the closest match to  $Q$



Suppose that we know triangular inequality holds, and we already have these pre-calculated distances:

Distances	a	b	c
a	0	6.7	7.07
b	6.7	0	2.3
c	7.07	2.3	0

# Why is Triangular Inequality so Important?

Step 1. Calculate  $D(Q,a)$

$$D(Q,a) = 2$$

The distance of 2 is now our best-so-far

Step 2. Calculate  $D(Q,b)$

$$D(Q,a) = 7.81$$

The distance of 2 is still our best-so-far

Step 3. Calculate  $D(Q,c)$

We don't need to!

We know (thanks to triangular inequality):

$$D(Q,b) \leq D(Q,c) + D(b,c)$$

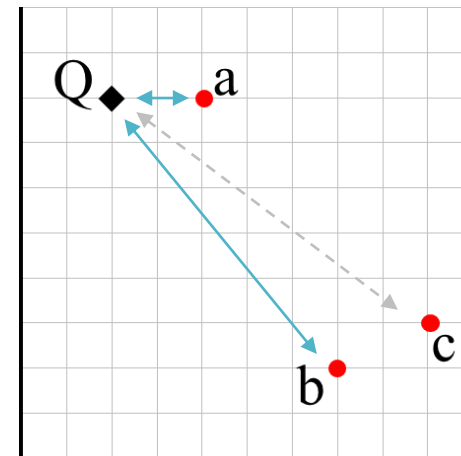
$$D(Q,b) - D(b,c) \leq D(Q,c)$$

$$7.81 - 2.3 \leq D(Q,c)$$

$$5.51 \leq D(Q,c)$$

$D(Q,c)$  must therefore be at least 5.51

5.51 is greater than 2, so we don't need to calculate the distance



Distances	a	b	c
a	0	6.7	7.07
b	6.7	0	2.3
c	7.07	2.3	0

# Calculating the Distance

Given two data:

$$\mathbf{x} = x_1 \dots x_n$$

$$\mathbf{y} = y_1 \dots y_n$$

The Euclidean Distance is defined as

$$D(\mathbf{x}, \mathbf{y}) \equiv \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

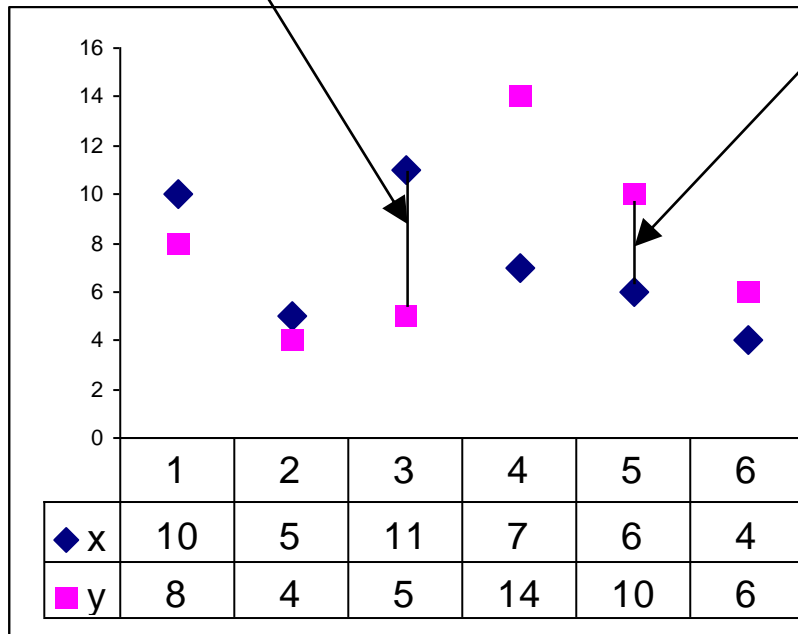
$$(x_3 - y_3)^2 = (11 - 5)^2 = 36$$

$$(x_5 - y_5)^2 = (6 - 10)^2 = 16$$

$$D(\mathbf{x}, \mathbf{y}) = 10.488$$

Euclidean distance has *Symmetry*, *Constancy*, *Positivity* and satisfies the *Triangle Inequality*

Keogh, E. and Kasetty, S. (2002). **On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration**. In the *8<sup>th</sup> ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. July 23 - 26, 2002. Edmonton, Alberta, Canada. pp 102-111.



# Other Distance Metrics

The Minkowski metric (a generalisation of Euclidean Distance)

$$d(a, b) = \sqrt[n]{\sum_{i=1}^n (a_i - b_i)^n}$$

If  $n=1$ , this is referred to as the Manhattan metric.

$$d(a, b) = \sum |a_i - b_i|$$

Correlation Distance

$$d(a, b) = \frac{\sum (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum (a_i - \bar{a})^2 \sum (b_i - \bar{b})^2}}$$

Mahalanobis distance

$$d(a, b) = (a - b)^T S^{-1} (a - b)$$

# Data Normalisation

We have seen how we can compute the distance between objects

However, this can be greatly affected by scale:

Height (cm)	Weight (lbs)	Query
185	215	

Data

Using Euclidean Distance:  $D(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$

Height (cm)	Weight (lbs)			
185	400	$(185-185)^2$	$(400-215)^2$	185
200	220	$(200-185)^2$	$(220-215)^2$	15.81
220	215	$(220-185)^2$	$(215-215)^2$	35
190	210	$(190-185)^2$	$(210-215)^2$	7.07

Someone is the same height as the query (185 cm), but weighs 215lbs more

Someone is the same weight as the query (215 lbs), but is 35cm taller

As you may expect, the closest match is compromise where the difference to each is small – 5cm taller and 5 pounds lighter

# Data Normalisation

But what if height isn't in cm, but is recorded in micrometres (um)?

Height (um)	Weight (lbs)	Query
185000	215	

Data

Using Euclidean Distance:  $D(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$

Height (um)	Weight (lbs)			
185000	400	$(185000 - 185000)^2$	$(400 - 215)^2$	185
200000	220	$(200000 - 185000)^2$	$(220 - 215)^2$	15,000
220000	215	$(220000 - 185000)^2$	$(215 - 215)^2$	35,000
190000	210	$(190000 - 185000)^2$	$(210 - 215)^2$	5,000

Someone is the same height as the query (185 cm), but weighs 215lbs more

Someone is the same weight as the query (215 lbs), but is 35cm taller

The very large difference in scale now means that weight is overwhelmed by the height in the calculation

This means that the person weighing 400lbs is now a closer match!!!

# Standard Normalisation

We can **normalise** the data so that attributes are on comparative scales, regardless of the original scale.

To do this, for each attribute:

1. Calculate the mean and standard deviation for all values of that attribute
2. For each instance, subtract the mean of that attribute and divide by the standard deviation

Height (cm)	Weight (lbs)
185	400
200	220
220	215
190	210

Mean	198.75	261.25
Stdev	15.48	92.59

$$a_i' = \frac{(a_i - \bar{a})}{s}$$

# Standard Normalisation

1. Calculate the mean and standard deviation for all values of that attribute
2. Subtract the mean and divide by the standard deviation

Height (cm)	Weight (lbs)
185	400
200	220
220	215
190	210

Mean	198.75	261.25
Stdev	15.48	92.59

$$a'_i = \frac{(a_i - \bar{a})}{s}$$



Height	Weight
-0.88833	1.498541
0.0807573	-0.44551
1.3728739	-0.49951
-0.565301	-0.55352

Mean	0	0
Stdev	1	1

The attributes have been *standardised* – each attribute now has a mean of 0 and standard deviation of 1

Now both attributes are on comparable scales, and neither will overwhelm the other in distance calculations

What about with the version of the data where height was on a much larger scale?

# Standard Normalisation

1. Calculate the mean and standard deviation for all values of that attribute
2. Subtract the mean and divide by the standard deviation

Height (cm)	Weight (lbs)
185	400
200	220
220	215
190	210

Mean	198.75	261.25
Stdev	15.48	92.59

$$a'_i = \frac{(a_i - \bar{a})}{s}$$



Height	Weight
-0.88833	1.498541
0.0807573	-0.44551
1.3728739	-0.49951
-0.565301	-0.55352

Mean	0	0
Stdev	1	1

Height (um)	Weight (lbs)
185000	400
200000	220
220000	215
190000	210

Mean	198750	261.25
Stdev	15478.48	92.59

$$a'_i = \frac{(a_i - \bar{a})}{s}$$



Height	Weight
-0.88833	1.498541
0.0807573	-0.44551
1.3728739	-0.49951
-0.565301	-0.55352

Mean	0	0
Stdev	1	1

After normalisation, both are identical

# Standard Normalisation

Even though the scale of the height attribute was very different in these two examples, the underlying pattern in the data was the same

After normalisation this is clear, as with standardised units, height is the same in both examples after **normalising** the data

Height (cm)	Weight (lbs)
-0.88833	1.498541
0.0807573	-0.44551
1.3728739	-0.49951
-0.565301	-0.55352

Mean	0	0
Stdev	1	1

Height (cm)	Weight (lbs)
-0.88833	1.498541
0.0807573	-0.44551
1.3728739	-0.49951
-0.565301	-0.55352

Mean	0	0
Stdev	1	1

# Standard Normalisation: Another Example

*(stolen from one of Tony's lectures)*

1. Calculate the mean and standard deviation of each attribute
2. For each data, subtract the mean and divide by the standard deviation

	a	b	c	a'	b'	c'
1	0.1	2	10000000	<b>0.76</b>	<b>-1.25</b>	<b>-1.11</b>
2	0.01	1	10001000	<b>-0.64</b>	<b>-1.28</b>	<b>2.38</b>
3	0.001	100	10000100	<b>-0.78</b>	<b>2.04</b>	<b>-0.76</b>
4	0.03	14	10000111	<b>-0.33</b>	<b>-0.85</b>	<b>-0.72</b>
5	0.01	23	10000854	<b>-0.64</b>	<b>-0.54</b>	<b>1.87</b>
6	0.14	50	10000231	<b>1.38</b>	<b>0.36</b>	<b>-0.30</b>
7	0.03	42	10000049	<b>-0.33</b>	<b>0.09</b>	<b>-0.94</b>
8	0.009	5	10000500	<b>-0.65</b>	<b>-1.15</b>	<b>0.64</b>
9	0.02	67	10000222	<b>-0.48</b>	<b>0.93</b>	<b>-0.33</b>
10	0.16	88	10000111	<b>1.69</b>	<b>1.64</b>	<b>-0.72</b>
mean	<b>0.051</b>	<b>39.2</b>	<b>10000318</b>			
st dev	<b>0.06434</b>	<b>29.76</b>	<b>286.3957</b>			

$$a_i' = \frac{(a_i - \bar{a})}{s}$$

# Similarity: Final Point – Categorical Attributes

Sometimes attributes may be discrete, e.g. {red, blue, green} or {sunny, cloudy, rainy}

A simple method for handling discrete values is to assume:

- 0 if both values are the same, and
- 1 if they are different

$$(0.76 + .64)^2 + (-1.25 + 1.28)^2 + 1$$

x	0.76	-1.25	RED
y	-0.64	-1.28	BLUE
z	-0.78	0.00	RED

	x	y	z
x	0	2.96	3.93
y		0	2.67
z			0

This could be adapted to use expert knowledge. For instance, is the difference between large and small the same as the difference between medium and small?

# A Generic Technique for Measuring Similarity

To measure the similarity between two objects, **transform one of the objects into the other**, and **measure how much effort it took**. The measure of effort becomes the distance measure.

The distance between Patty and Selma.

**Change dress color, 1 point**

**Change earring shape, 1 point**

**Change hair part, 1 point**

$D(\text{Patty}, \text{Selma}) = 3$

The distance between Marge and Selma.

**Change dress color, 1 point**

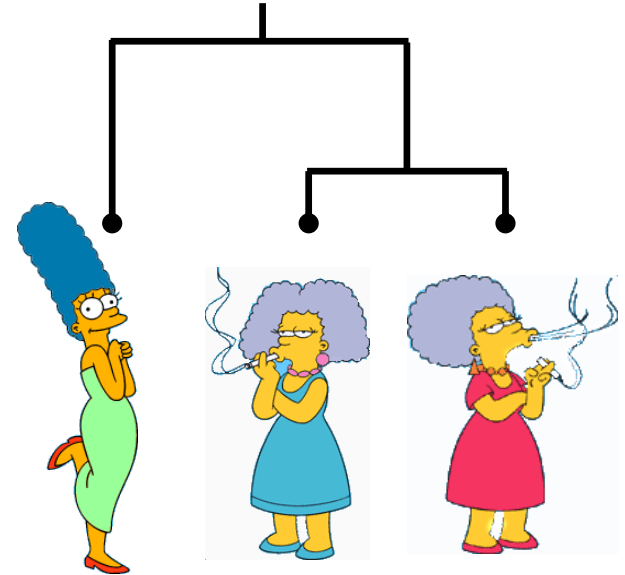
**Add earrings, 1 point**

**Decrease height, 1 point**

**Take up smoking, 1 point**

**Gain weight, 1 point**

$D(\text{Marge}, \text{Selma}) = 5$



This is called the “**edit distance**” or the “**transformation distance**”

# Edit Distance

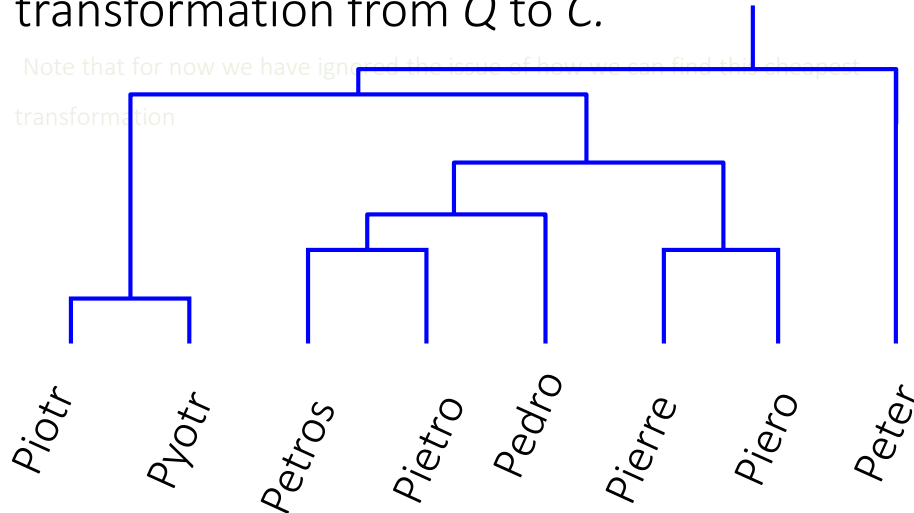
## Example

It is possible to transform any string  $Q$  into string  $C$ , using only *Substitution*, *Insertion* and *Deletion*.

Assume that each of these operators has a cost associated with it.

The similarity between two strings can be defined as the cost of the cheapest transformation from  $Q$  to  $C$ .

Note that for now we have ignored the cost of insertion and deletion.



How similar are the names “Peter” and “Piotr”?

Assume the following cost function

<i>Substitution</i>	1 Unit
<i>Insertion</i>	1 Unit
<i>Deletion</i>	1 Unit

$D(\mathbf{Peter}, \mathbf{Piotr})$  is 3

**Peter**



Substitution (i for e)

**Piter**



Insertion (o)

**Pioter**



Deletion (e)

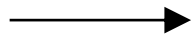
**Piotr**

# Summary: Measuring Distances

1. If all the attributes are real valued and largely uncorrelated then Euclidean distance is fine, but fields should be normalised.
2. If the attributes are categorical use:
  - Binary similarity
  - Value frequencies
  - Expert knowledge (e.g. edit distance)
3. If the attributes are correlated, the distance will be biased
  - Use a distance metric that allows for this (e.g. Mahalanobis )
  - or transform the variables so they are uncorrelated with each other (e.g. Principal Components Analysis)

# Clustering Algorithms

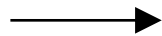
## Partitional



1. k-Means
2. Partitioning Around the Medoids

*use an iterative local search algorithm to find the best splits in the data, incrementally improving the splits*

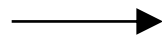
## Hierarchical



- Top down/Bottom up with:
1. Single Linkage
  2. Complete Linkage
  3. Average Linkage

*use a greedy algorithm to recursively join individual objects to clusters*

## Density



Density Peaks

*Group data together based on how many other data are "close"*

# k-means Clustering

K-means<sup>[1]</sup> is one of the simplest partitioning clustering algorithms in use

The procedure follows a simple and easy way to group a given data set into a **certain pre-defined number of clusters** (assume  $k$  clusters).

The main idea is to define  **$k$  centroids**, one for each cluster, and then **associate each data point with its closest centroid**

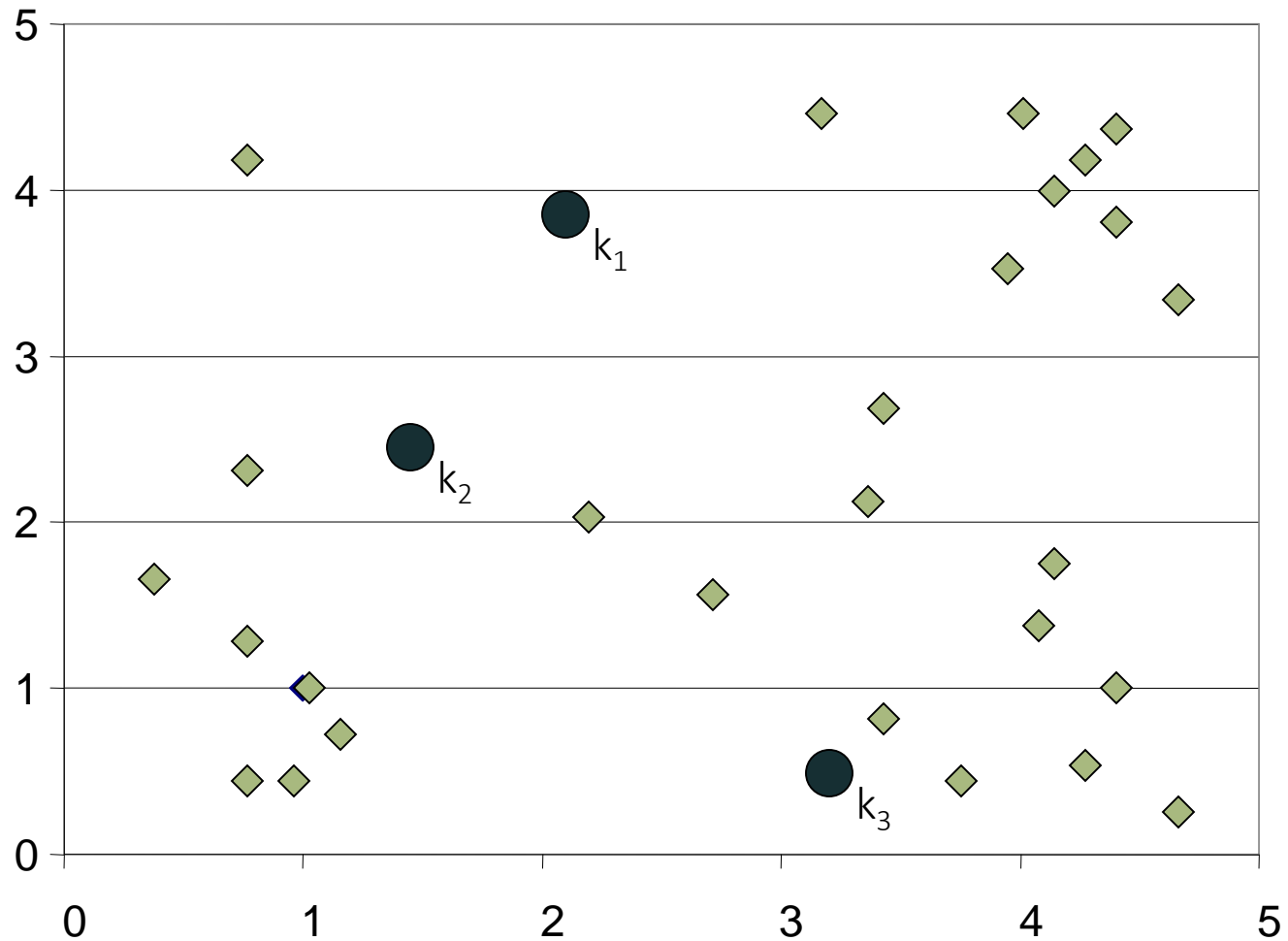
[1] J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations, *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*", Berkeley, University of California Press, 1:281-297

# k-means Clustering Algorithm

(Assuming  $k$  is given – more on that later)

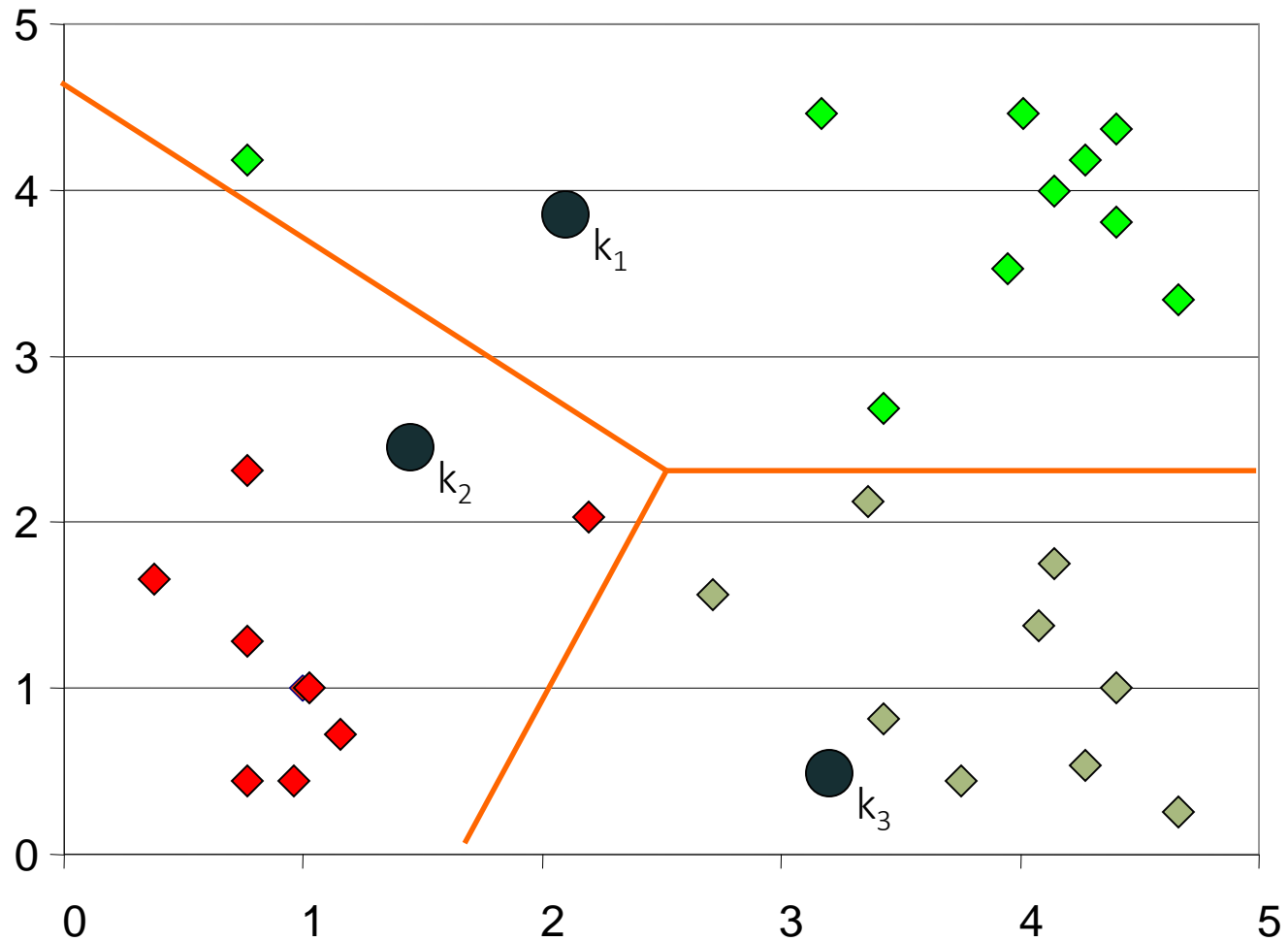
1. Initialize the  $k$  cluster centers (randomly, if necessary)
2. Decide the cluster memberships of the  $N$  objects by assigning them to the nearest cluster center.
3. Re-estimate the  $k$  cluster centers, with the assumption that the cluster memberships found in step 2 are correct
4. Recalculate the distance from each object to each centroid. If no object is now closer to a different centroid, finish. Otherwise, go back to step 2

# Example with k-means



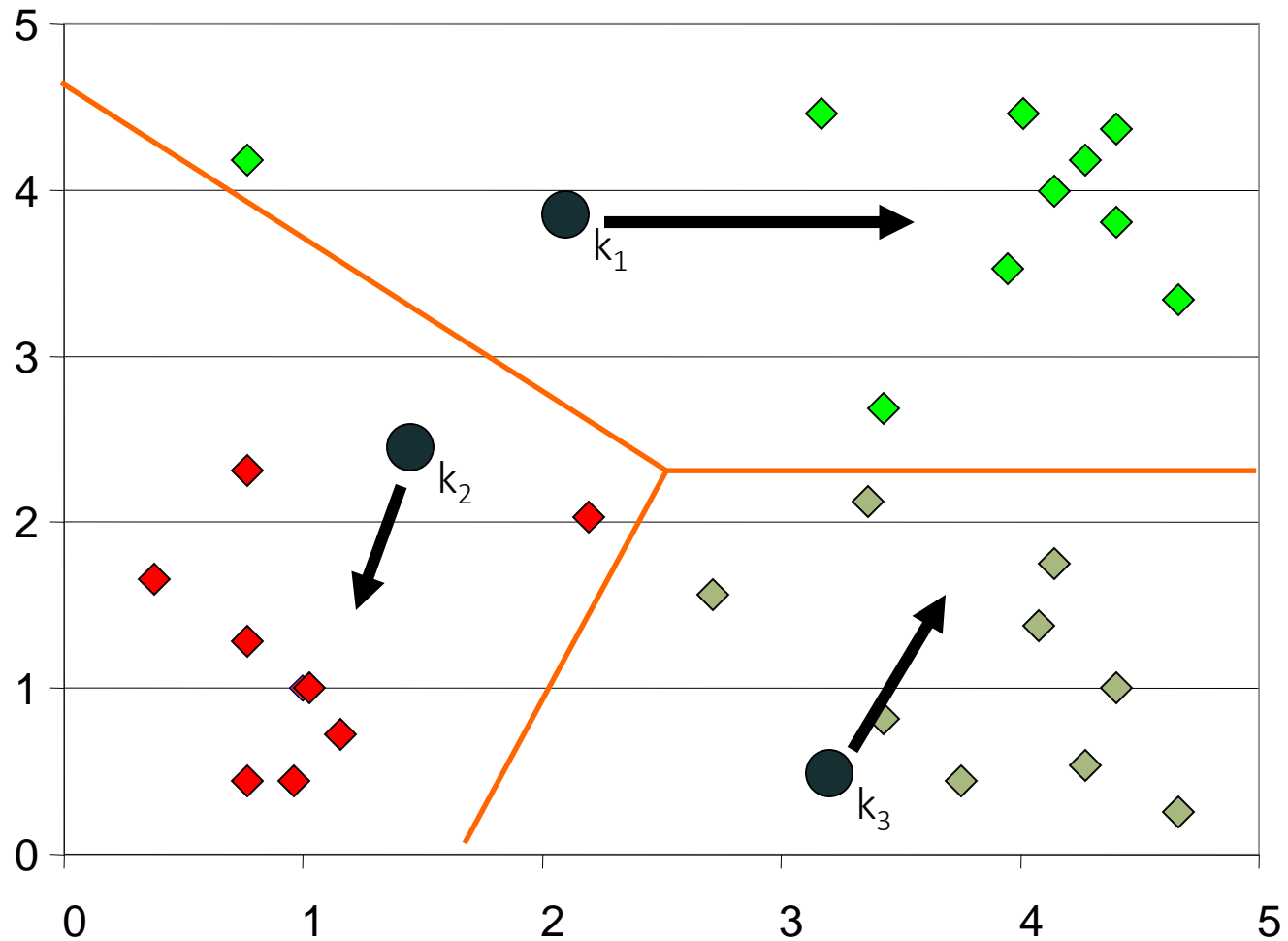
Initialize the  $k$  cluster centers ( $k=3$  here)

# Example with k-means



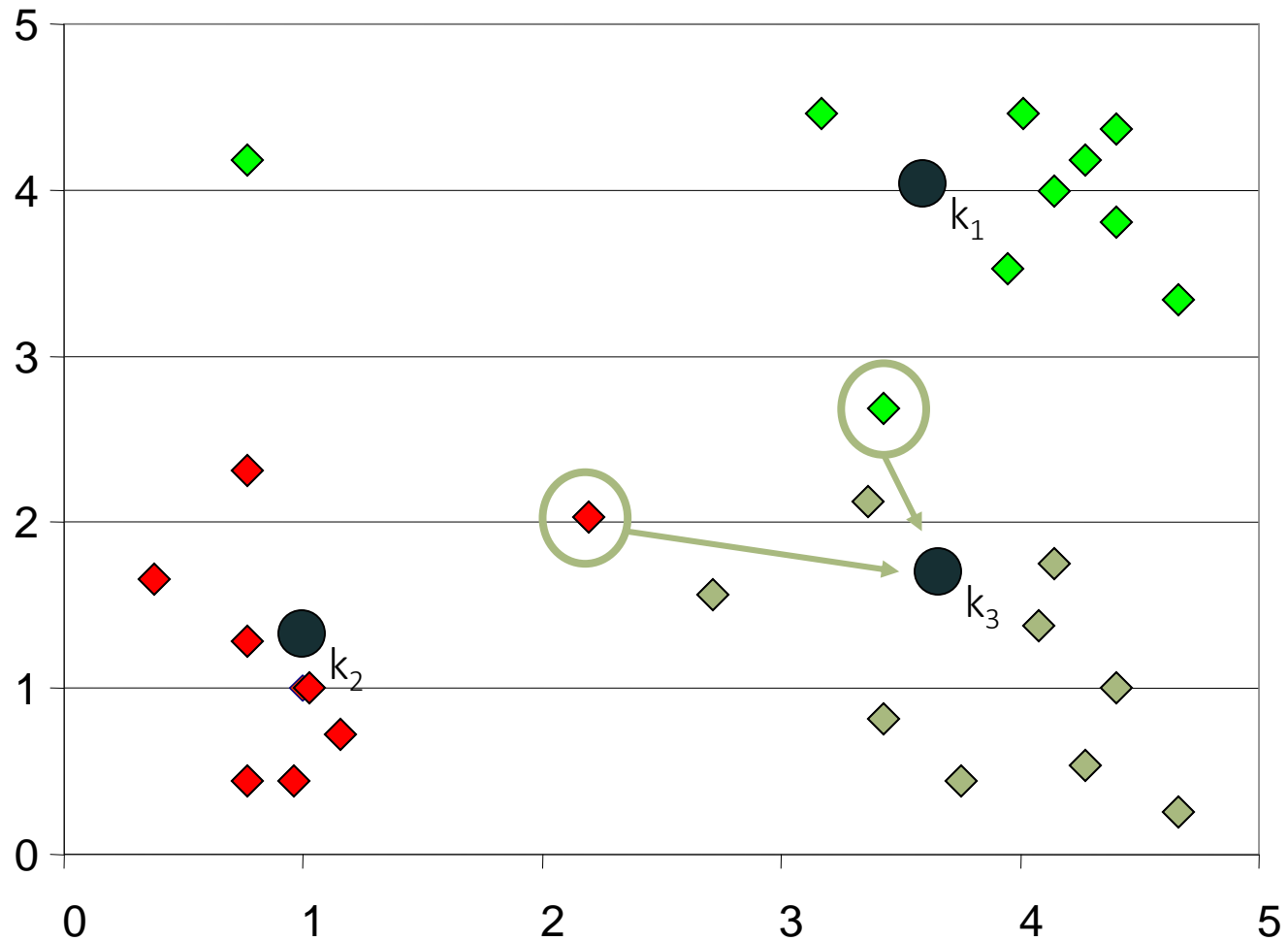
Assign each object to a cluster by finding the closest centroid to each

# Example with k-means



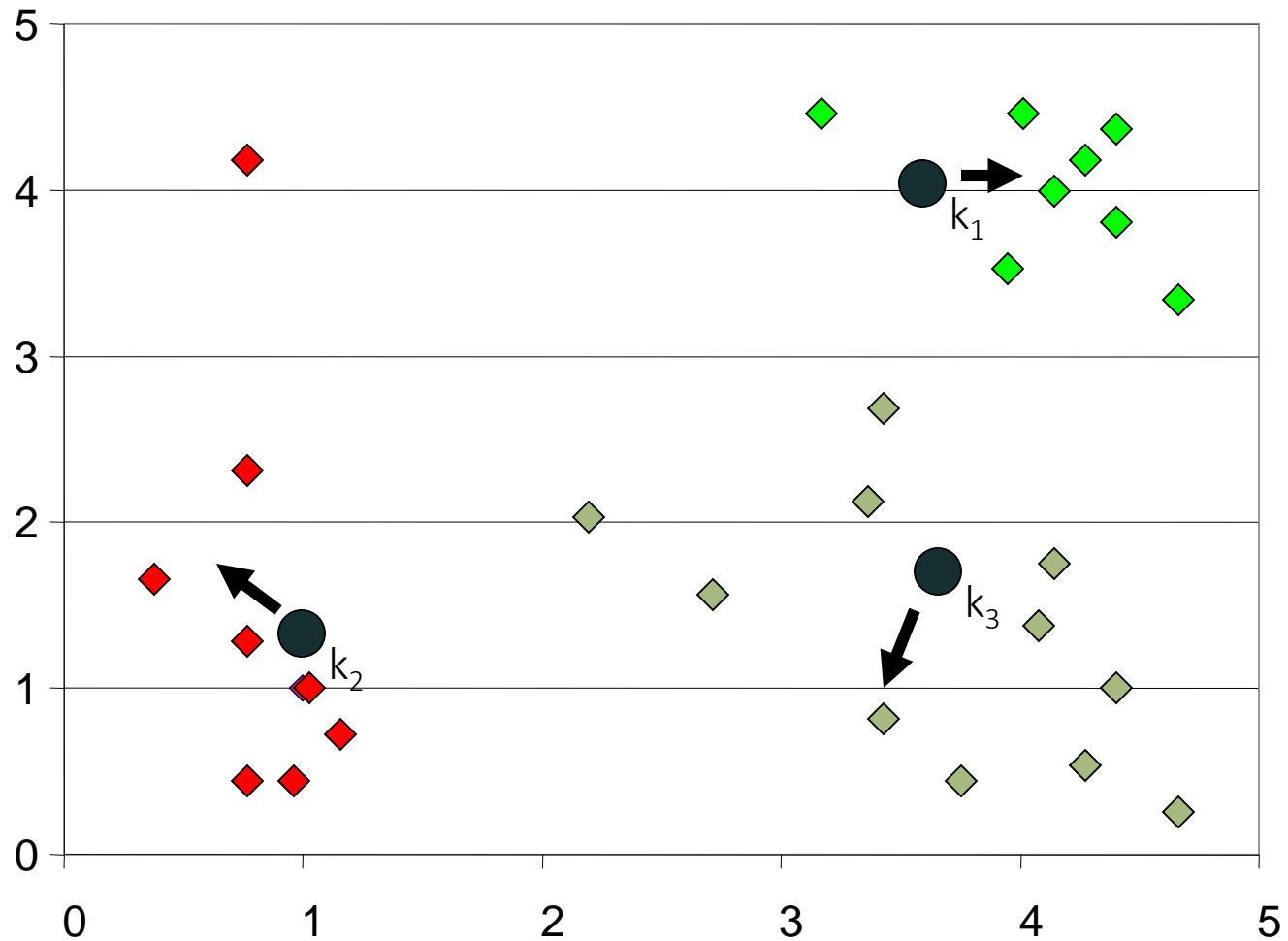
Re-estimate the  $k$  centroids

# Example with k-means



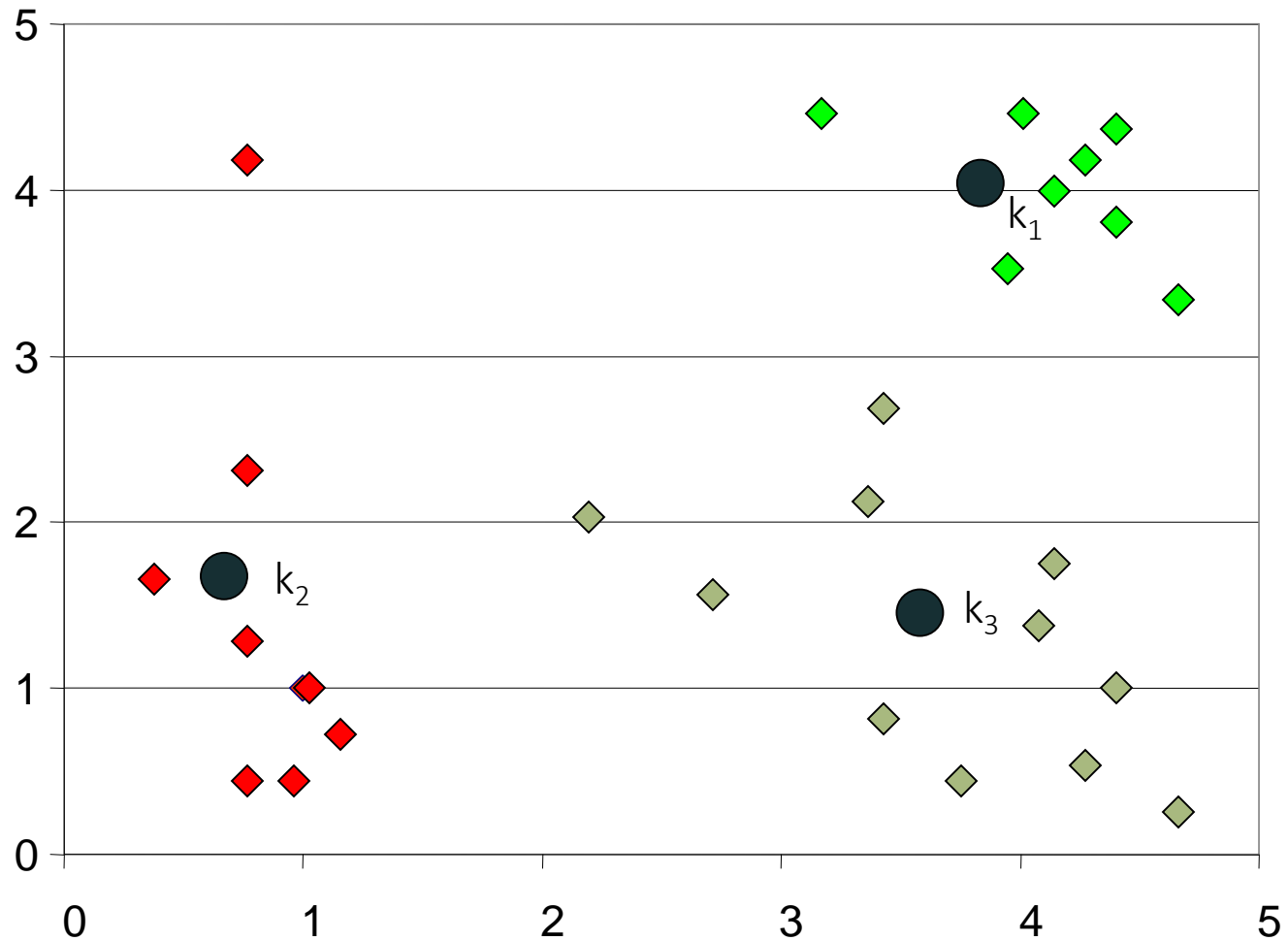
Recalculate memberships with new centroids, repeat algorithm if any change

# Example with k-means



Re-estimate the  $k$  centroids with updated memberships

# Example with k-means



Recalculate memberships – no change this time, so finish

# Comments on the k-means algorithm

## Strengths

Relatively efficient:  $O(tkn)$ , where  $n$  is the number of objects,  $k$  is the number of clusters, and  $t$  is the number of iterations (typically  $k, t \ll n$ )

The algorithm often terminates at a local optimum

## Weaknesses

Applicable *only when mean is defined*. What about categorical data?

*Need to specify  $k$* , the *number* of clusters, in advance

Unable to handle noisy data and *outliers*

Not suitable to discover clusters with *non-convex shapes*

# An Alternative Partitional Clustering Algorithm: k-medoids

Another common clustering approach is to partition data objects with the centroid of each cluster as an **actual data object**

- When the centroid is a data object it is known as a **medoid**

In other words,  **$k$  data objects** are selected to represent the  **$k$  clusters**

Given a data set with  $N$  objects, the problem is then deciding which  $k$  of the  $N$  objects are the best representatives

# k-medoids Clustering Algorithms

A large number of algorithms have been developed for this problem, including:

- PAM (partitioning around medoids)
- CLARA (clustering large applications)
- CLARANS (CLARA using neighbourhood search)
- CLAVANS (clustering large applications using variable neighbourhood search – Nguyen & Smith, 2006)
- And more...

# PAM

Partitioning around medoids (PAM) was introduced by Kaufman and Rousseeuw (1990).

PAM generally assumes the distance matrix has already been calculated

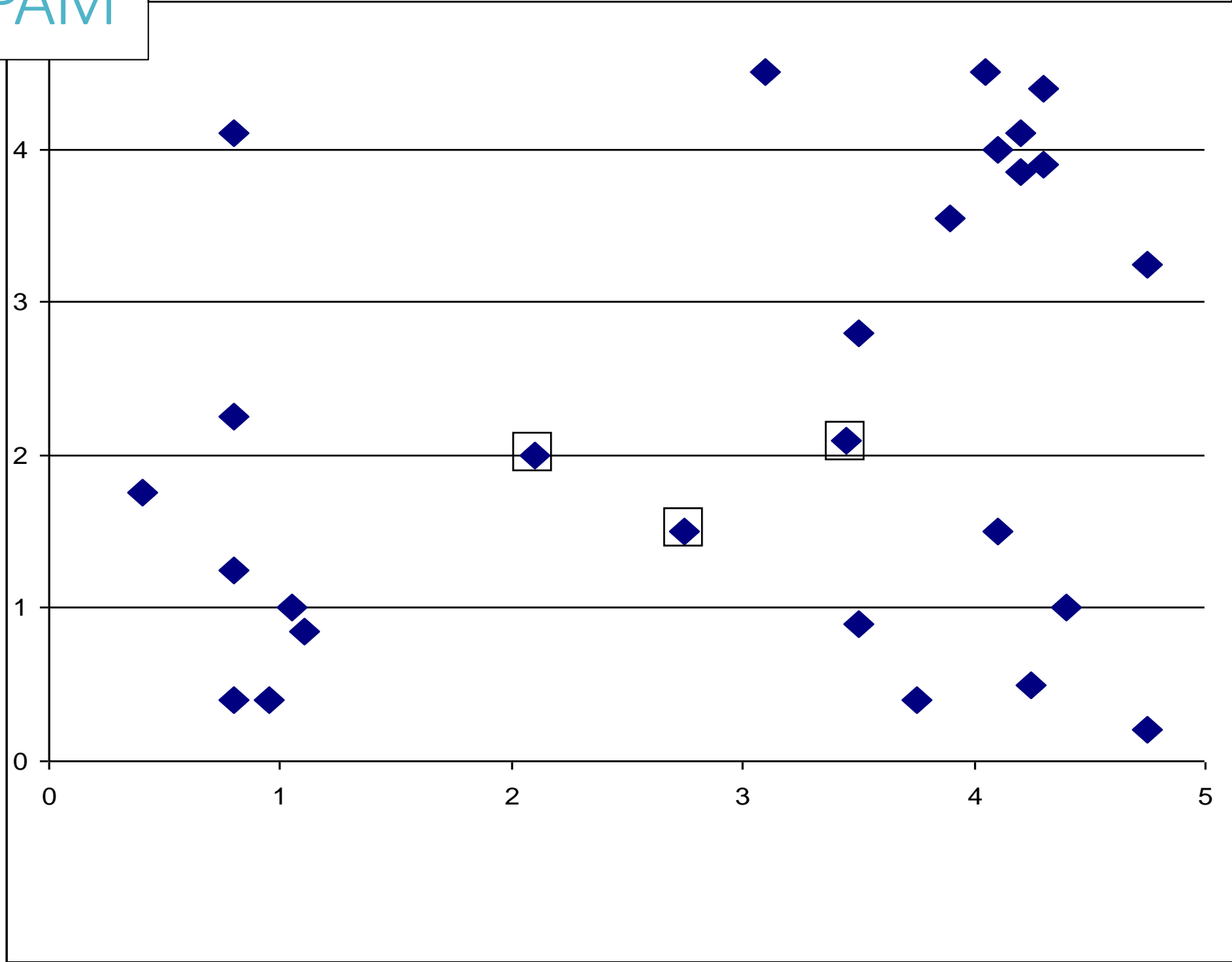
This means PAM can work with any defined distance matrix (no need to average discrete attributes, etc.)

## Algorithm:

1. Select  $k$  initial central medoids
2. Assign each data to the cluster associated with the closest medoid
3. Try swapping each cluster member with its medoid, select new medoid as that with the lowest within cluster sum of distance
4. If no medoids change, finish. Otherwise go to step 2

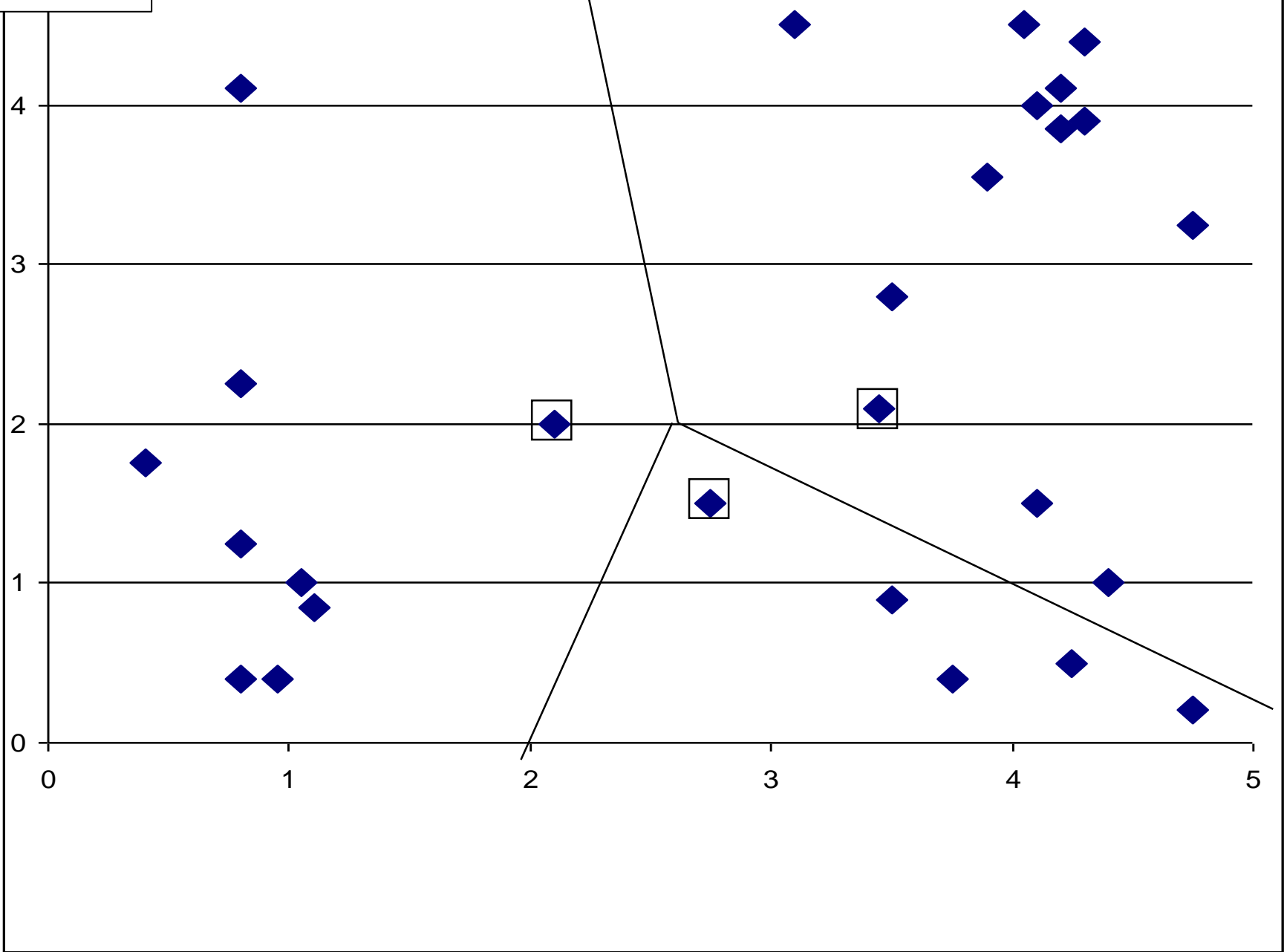
# PAM

Initialisation: Choose initial medoids (good idea to aim for the centre)



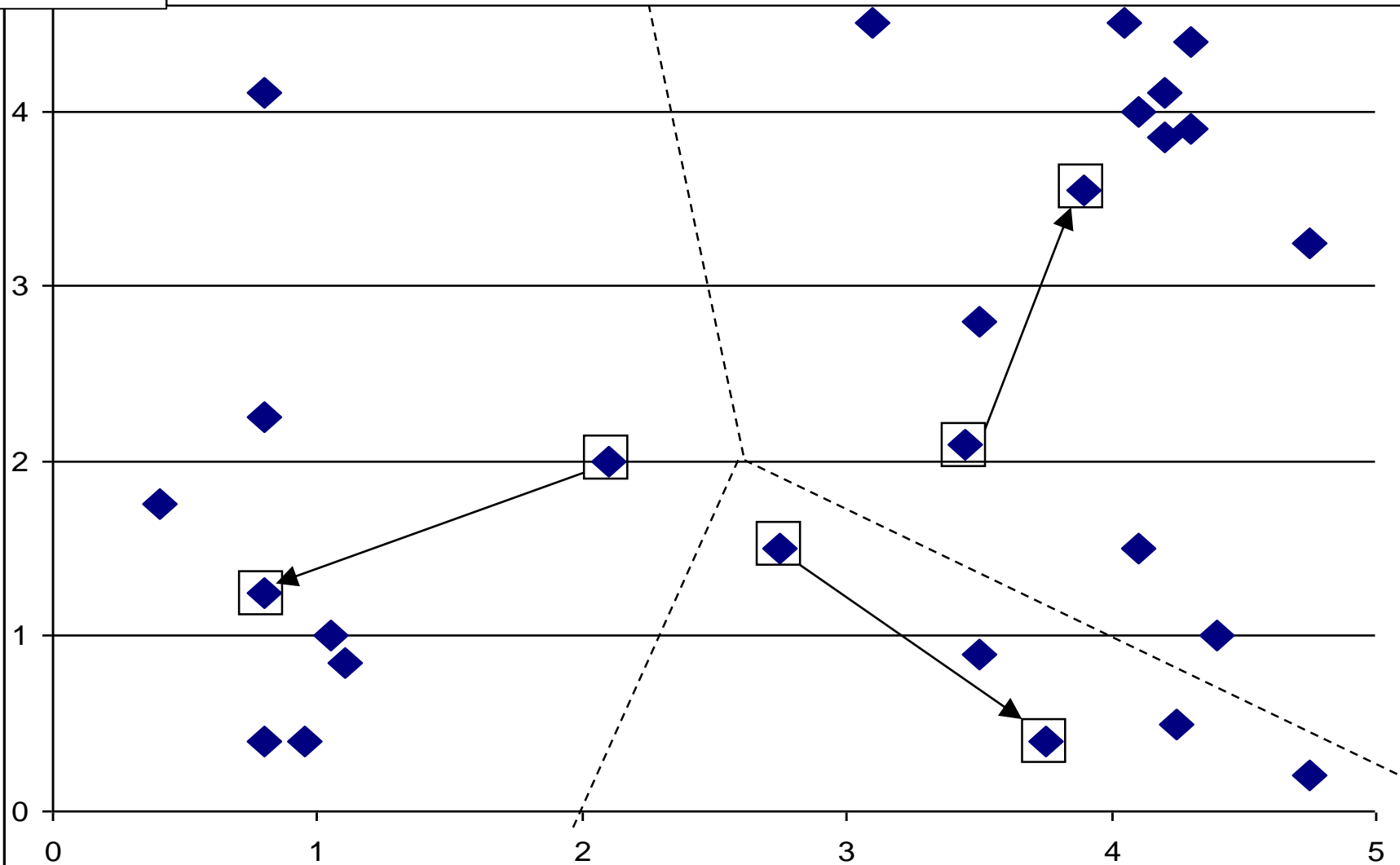
# PAM

Initialisation: Determine cluster membership by assigning each to the closest medoid



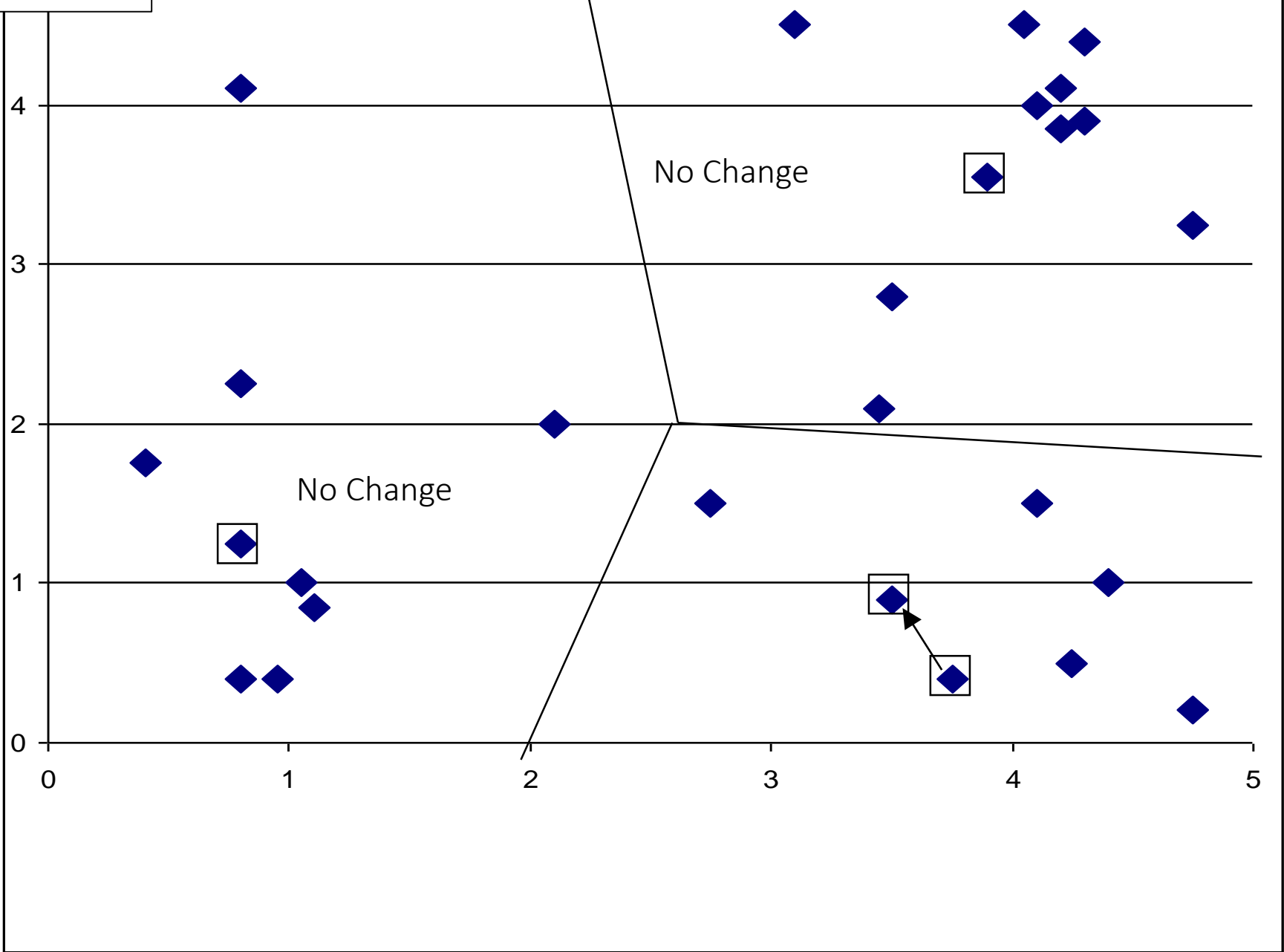
# PAM

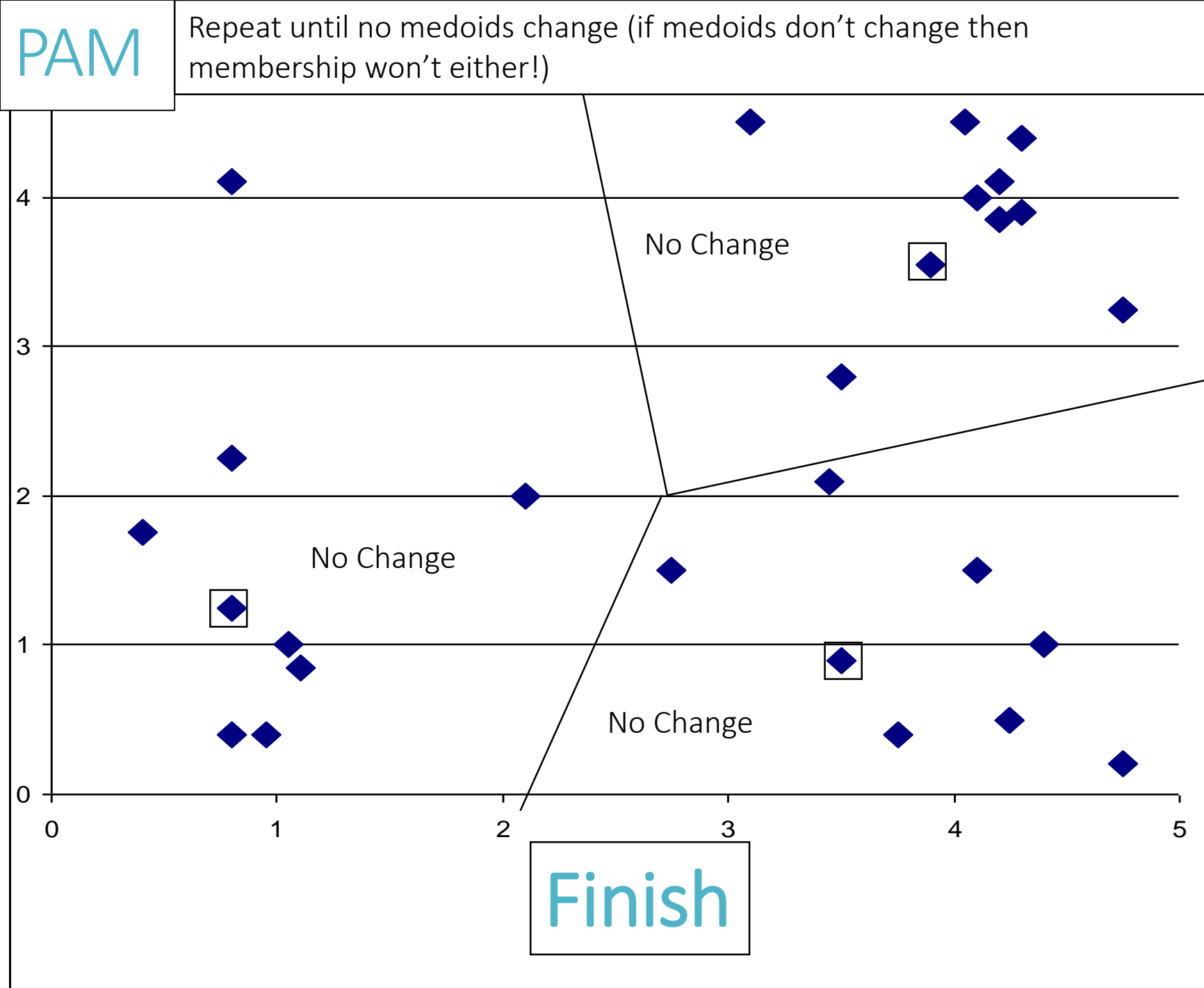
**Swap/test medoids:** For each cluster, trial each data point as a new medoid. Do this by summing the distances from the potential medoid to all other data points in the cluster. If any reduces the total sum, switch to that



# PAM

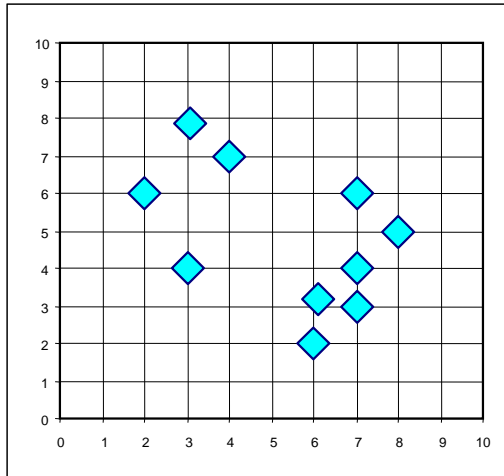
Swap/test medoids: Memberships changed in the bottom-right cluster, so recalculate cluster memberships for all elements (i.e. going back to step 2)



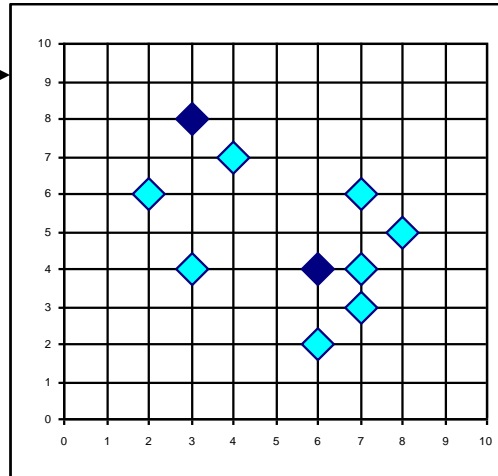


# PAM: Visual Summary

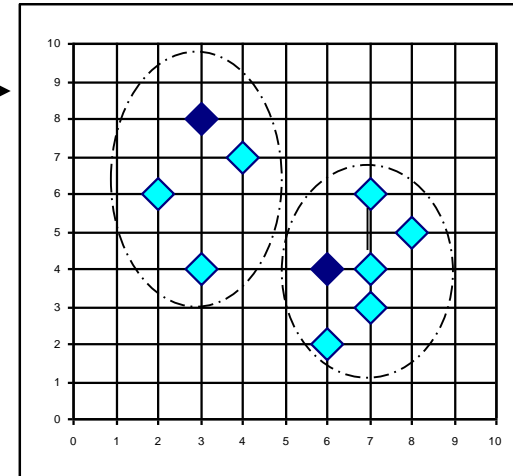
1. Start



2. Arbitrarily choose  $k$  object as initial medoids



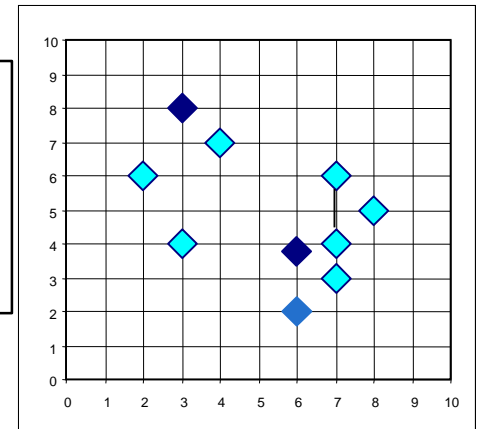
3. Assign each remaining object to nearest medoids



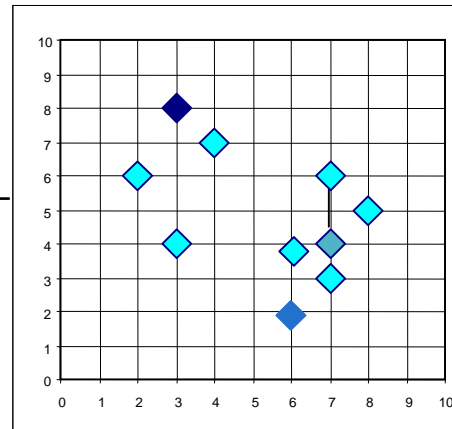
$K=2$

If any medoids changed

4a. Test each candidate medoid



4b. Swap any medoid and candidate that reduces within cluster sum



If no medoids changed

Finish

# PAM: Comments

## Strengths

- It is more robust, because it can work with any dissimilarity matrix (does not require a central point)

## Weaknesses

- Requires the entire distance matrix to be precalculated
- Each iteration requires evaluating  $n$  swaps so it is slow
- $K$  must still be set

# Other Medoid Based Clustering Algorithms (aka speeding up PAM)

## CLARA

- Proposed by the same team as PAM; essentially an extension of PAM that uses samples of the data, rather than the whole data
- They use five samples of  $(40+2k)$  objects, running PAM on each of the 5 samples.
- The final medoids are taken from the sample that produced the overall lowest sum of cluster distances

## CLARANS

- Similar to PAM – starts with  $k$  random medoids, and proceeds by randomly swapping a medoid with a random data point
- Stops when no improvement is found and restarts, but remembers the best-so-far solution

## CLAVANS

- Proposed by Nguyen and Smith in 2016
- Similar again, this time using variable neighbourhood searches rather than the whole data

# Hierarchical Clustering

Top-down

Bottom-up

Single/Complete/Average Linkage

# Hierarchical Clustering Overview


Partitional clustering algorithms use a greedy algorithm to search the space of all possible clusterings

Hierarchical clustering adopts a greedy algorithm to iteratively construct clusters

1. Represent clustering with **dendrograms**
2. **Bottom-up** vs **top-down** hierarchical clustering
3. Measuring distances between clusters: **linkage methods**

# Dendrograms

dendrogram

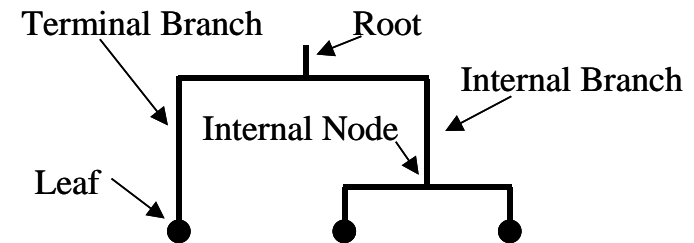
/ˈdendrə(v)ɡrɑːm/ 

*noun*

a tree diagram, especially one showing taxonomic relationships.



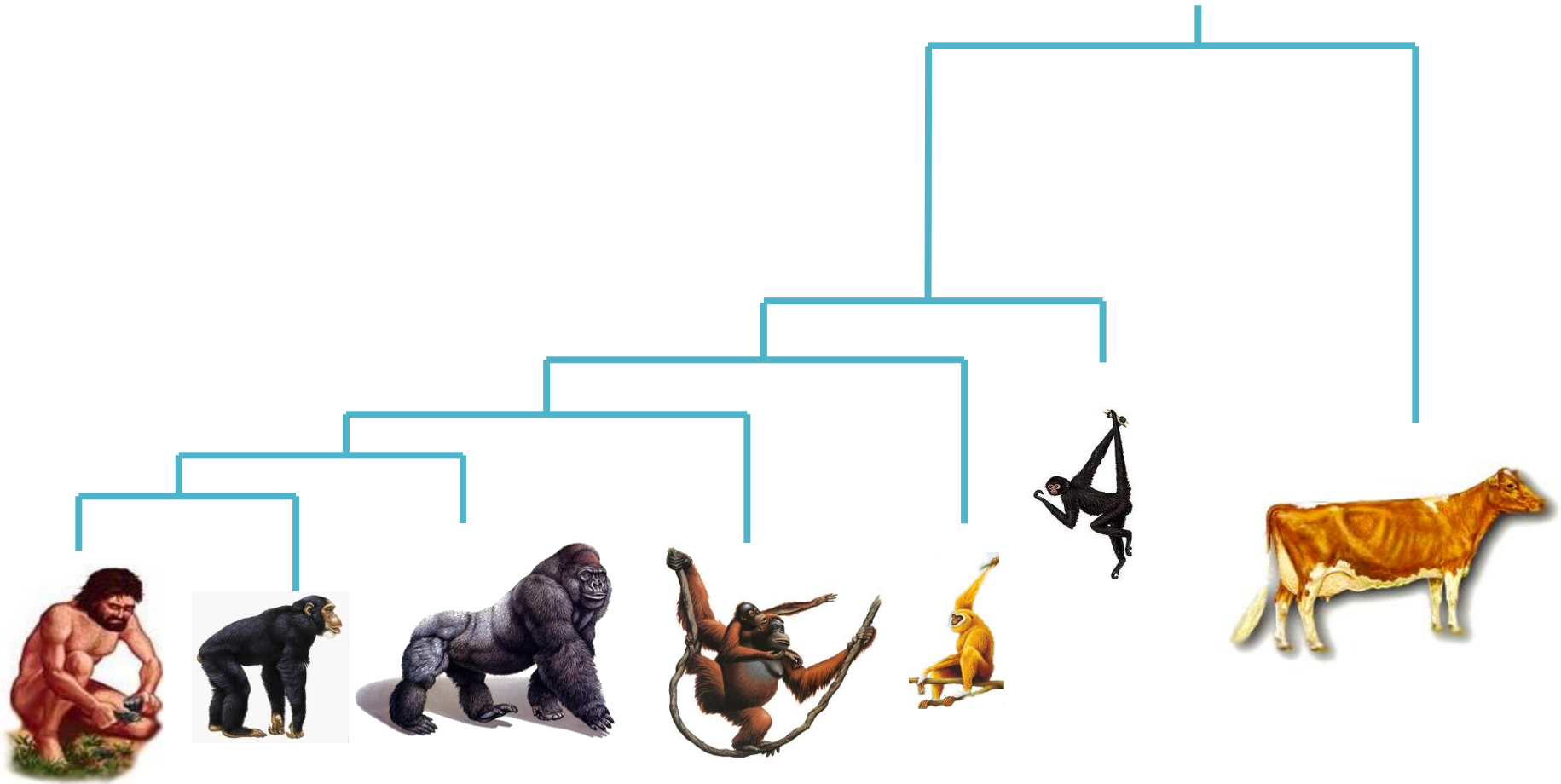
Dendrograms are tools that are used for representing clusters (and the similarity between clusters)



The similarity between two objects in a dendrogram is represented as the height of the lowest internal node they share.

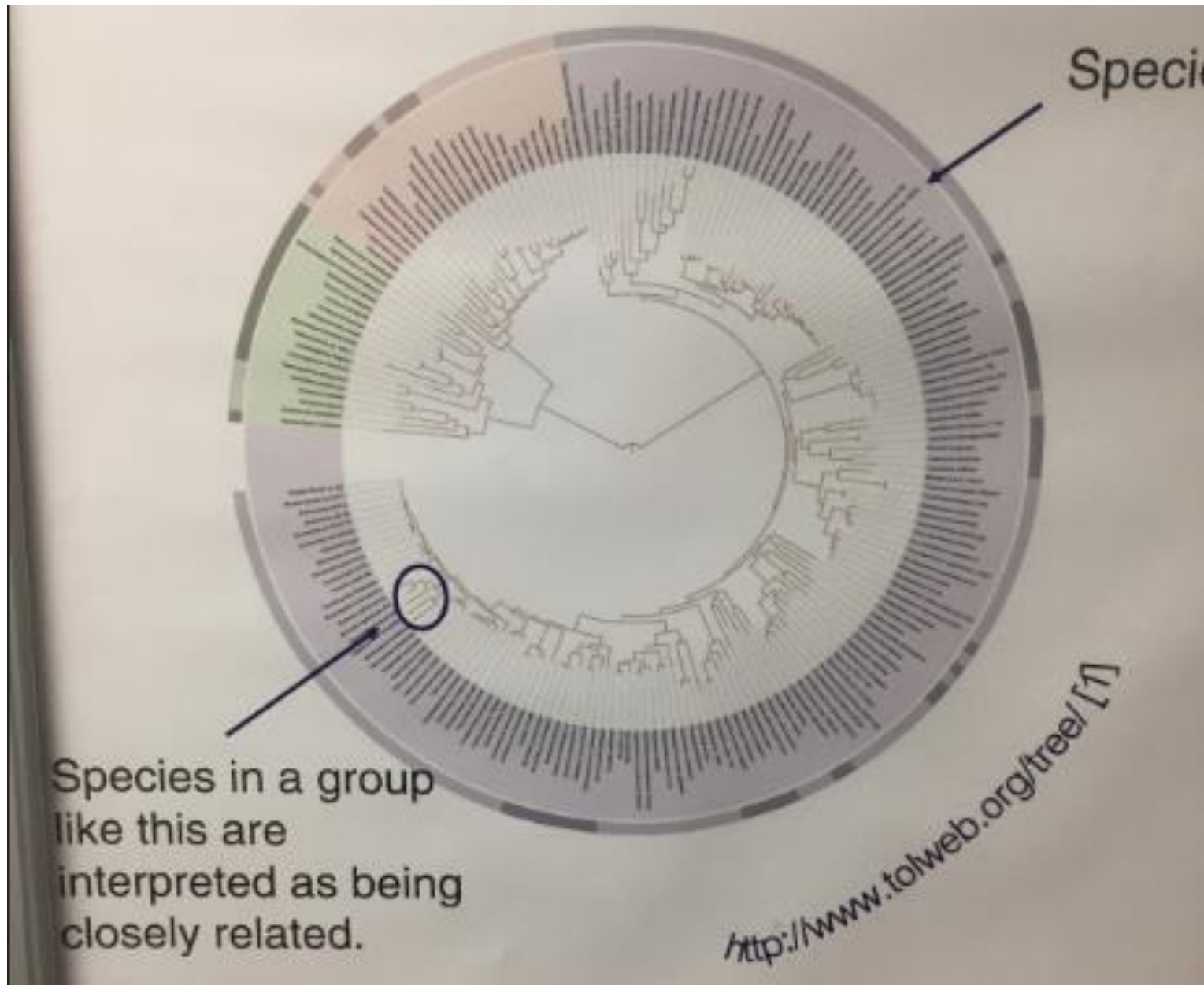


# Example: Phylogenetic Trees



(Bovine:0.69395, (Spider Monkey 0.390, (Gibbon:0.36079,(Orang:0.33636,(Gorilla:0.17147,(Chimp:0.19268, Human:0.11927):0.08386):0.06124):0.15057):0.54939);

Tree of Life: it's a little more complicated than that ...



## Example: Edit Distance (with Strings)

## Pedro (Portuguese)

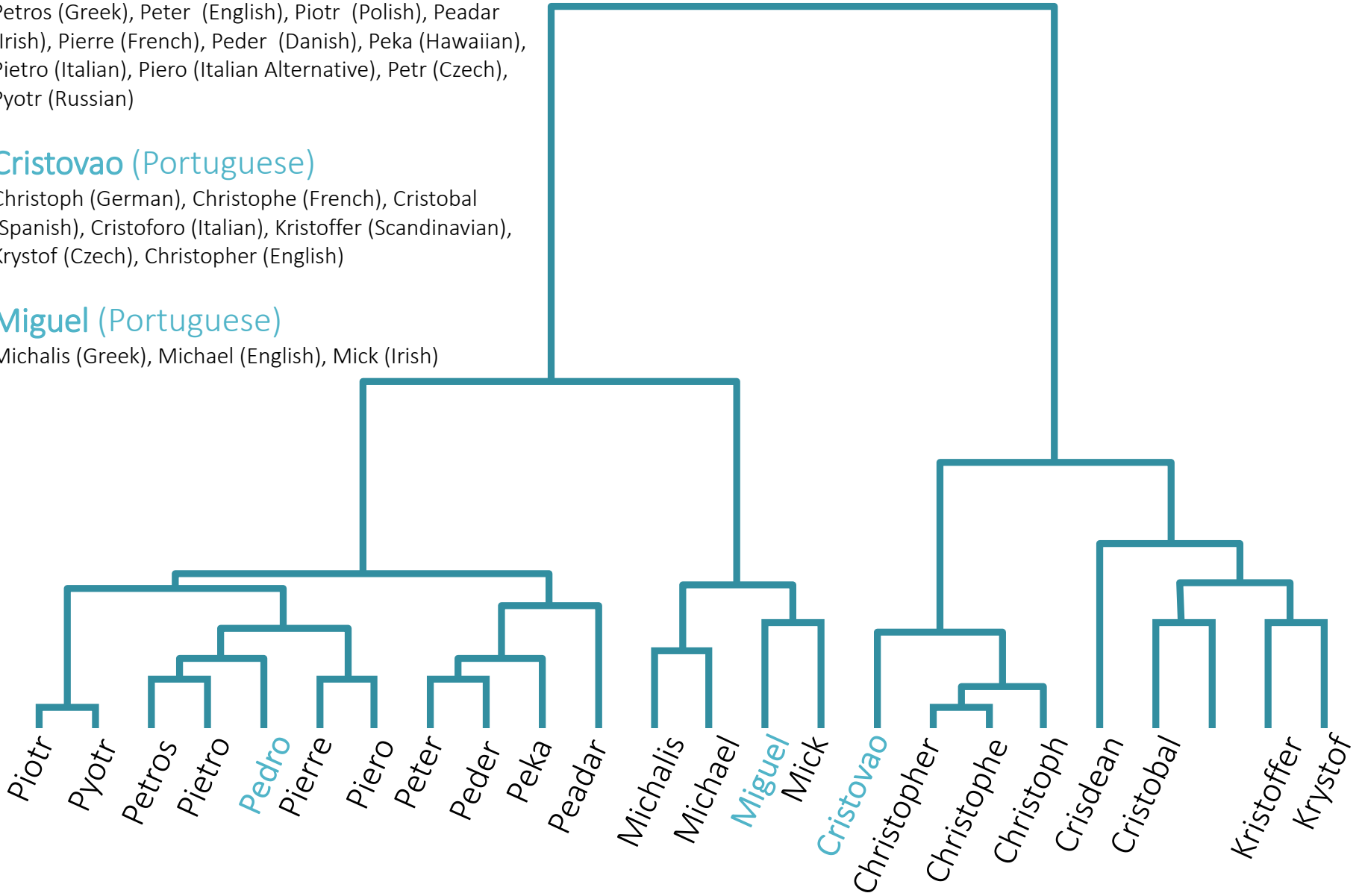
Petros (Greek), Peter (English), Piotr (Polish), Peadar (Irish), Pierre (French), Peder (Danish), Peka (Hawaiian), Pietro (Italian), Piero (Italian Alternative), Petr (Czech), Pyotr (Russian)

## Cristovao (Portuguese)

Christoph (German), Christophe (French), Cristobal (Spanish), Cristoforo (Italian), Kristoffer (Scandinavian), Krystof (Czech), Christopher (English)

## Miguel (Portuguese)

Michalis (Greek), Michael (English), Mick (Irish)



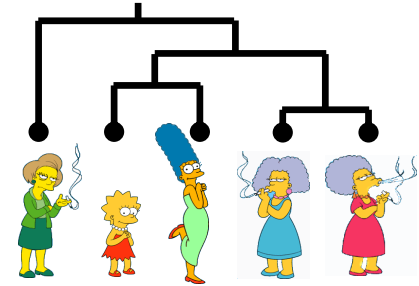
# How-to: Hierarchical Clustering

When forming dendrograms, the space of possible diagrams grows VERY quickly with  $n$

Given  $n$  objects, there are  $(2n - 3)! / [(2^{n-2}) (n - 2)!]$  possible dendrograms!

We cannot usually test all possible trees so we need to use heuristic approaches.

We can use greedy algorithms to construct dendrograms using either **bottom-up** or **top-down** approaches



Number of Leafs	Number of Possible Dendrograms
2	1
3	3
4	15
5	105
...	...
10	34,459,425

# Bottom-up vs. Top-down

## Bottom-Up (agglomerative)

1. Start with each item in its own cluster
2. Find the best pair best pair to merge into a new cluster
3. Repeat until all clusters are fused together

## Top-Down (divisive)

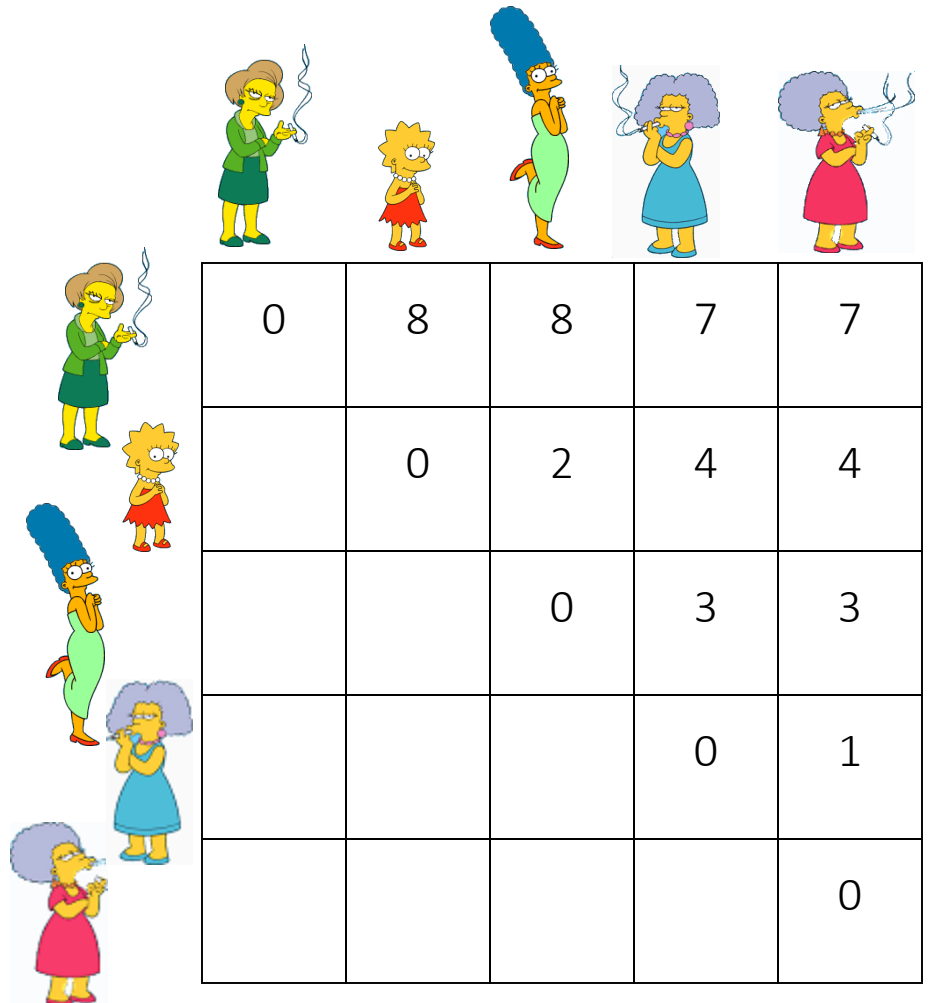
1. Start with all the data in a single cluster
2. Consider all possible ways to divide the cluster into two
3. Choose the best division, and recursively call operate on both sides until all objects are in their own clusters












# Examples: Bottom-up and Top Down

Start with a precalculated distance matrix containing distances between every pair of objects in our dataset

$$D(\text{Marge Simpson}, \text{Lisa Simpson}) = 8$$

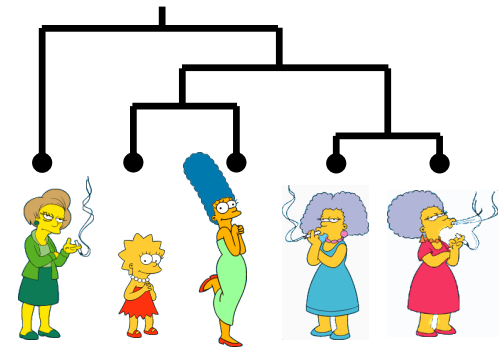
$$D(\text{Auntie Plaut}, \text{Auntie Plaut}) = 1$$



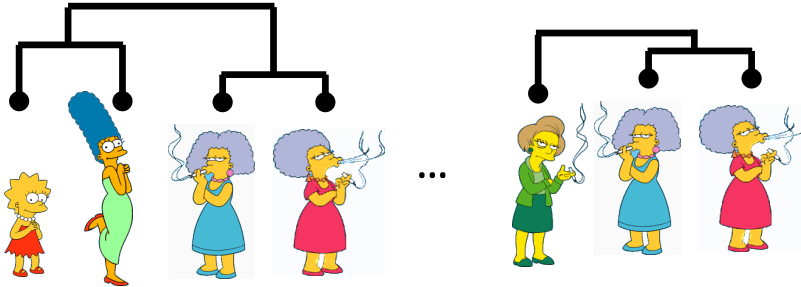
				
0	8	8	7	7
				
	0	2	4	4
				
		0	3	3
				
			0	1
				
				0

# Bottom-Up (agglomerative)

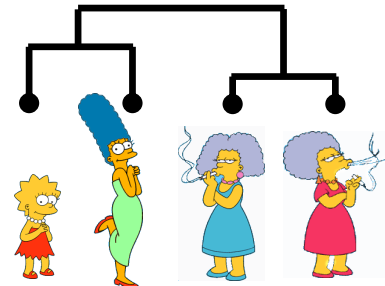
1. Start with each item in its own cluster
2. Find the best pair best pair to merge into a new cluster
3. Repeat until all clusters are fused together



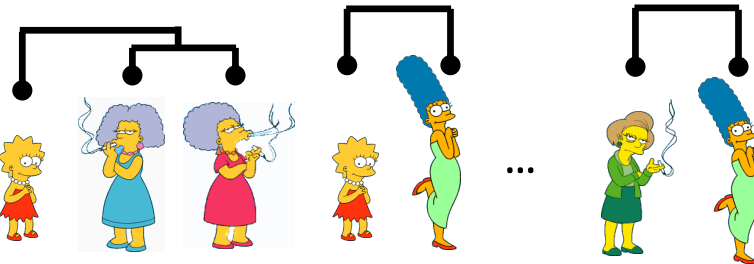
Consider all possible merges...



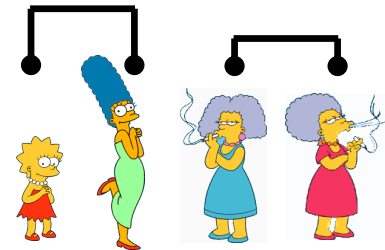
Choose the best



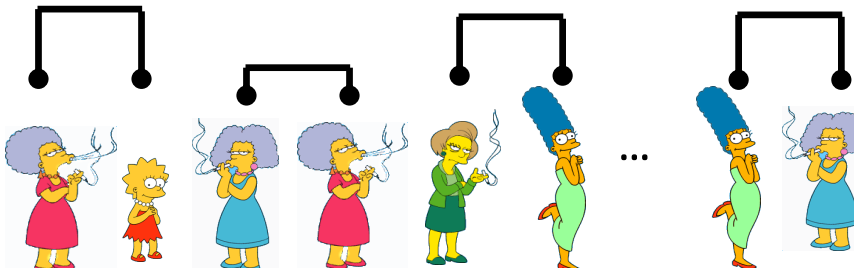
Consider all possible merges...



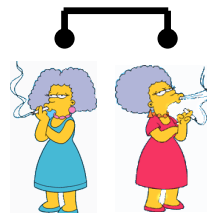
Choose the best



Consider all possible merges...

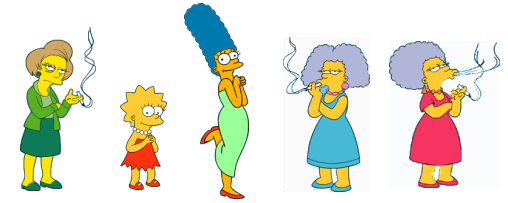


Choose the best

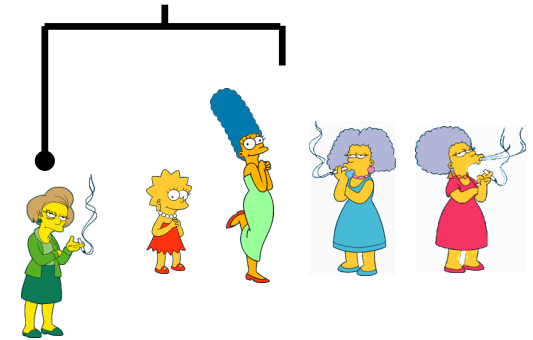


# Top-Down (divisive)

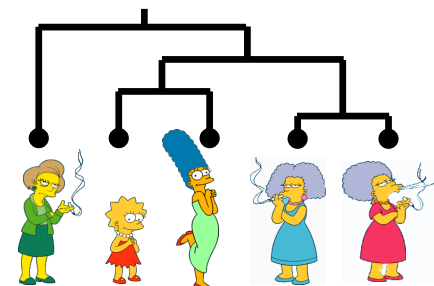
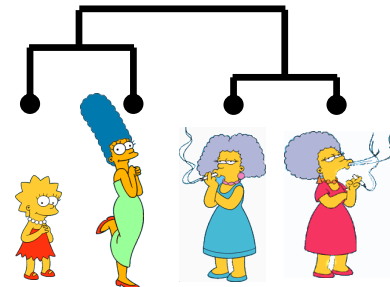
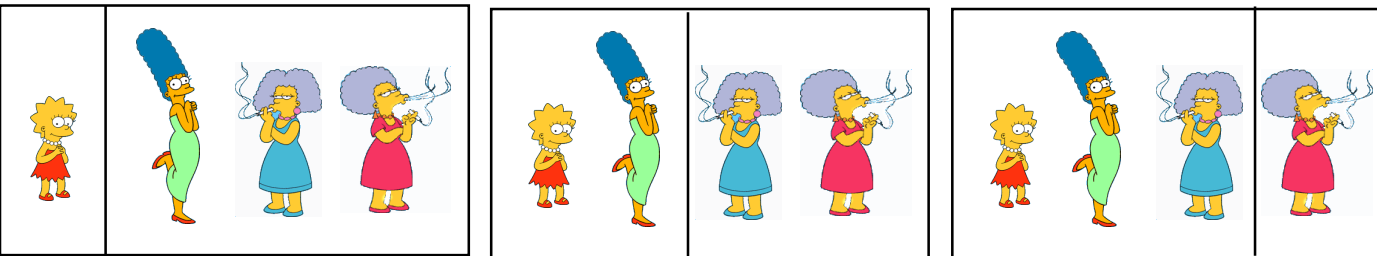
1. Start with all the data in a single cluster
2. Consider all possible ways to divide the cluster into two
3. Choose the best division, and recursively call operate on both sides until all objects are in their own clusters



Try all partitions and pick the best



Repeat recursively on non-leaf nodes



# Distance Between Clusters

Both of those approaches assume that we can measure the distance between two clusters.

We have a matrix of distances between objects, but what is the distance between:



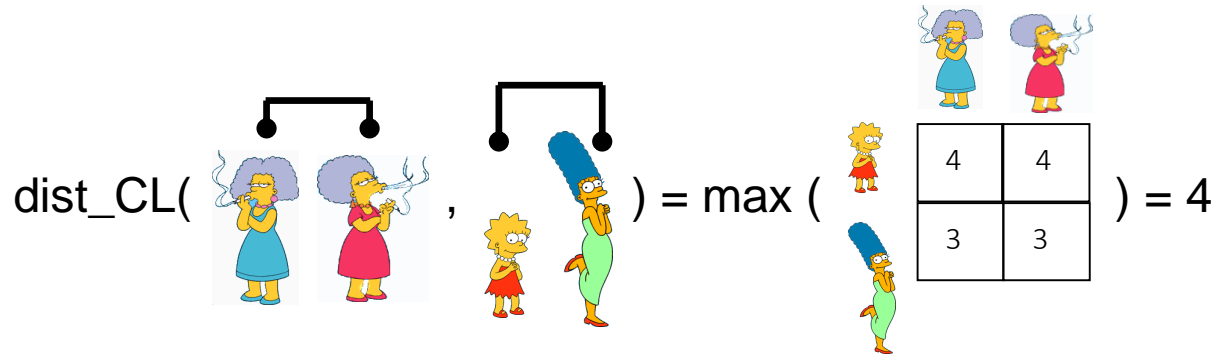
	0	8	8	7	7
		0	2	4	4
			0	3	3
				0	1
					0

**Single linkage (nearest neighbour):** With this approach, the distance between two clusters is determined by the distance of the two closest objects (nearest neighbours) in the different clusters.

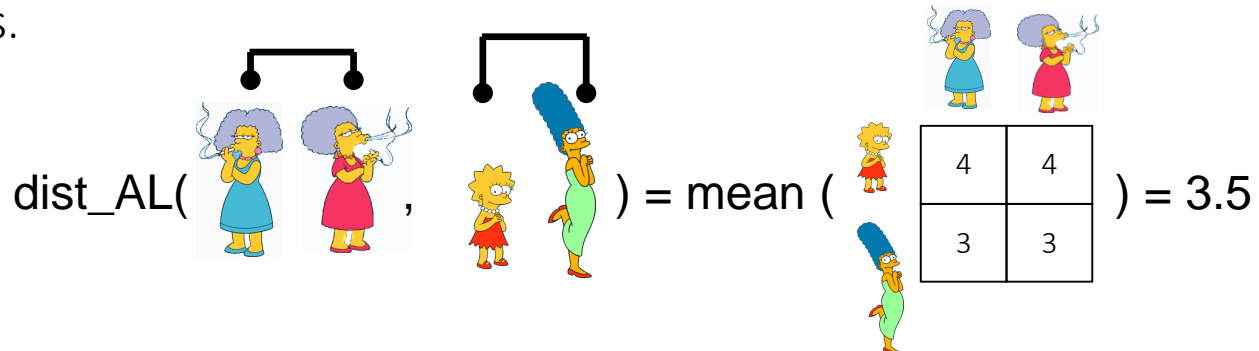
$$\text{dist\_SL}(\text{Cluster 1}, \text{Cluster 2}) = \min \left( \begin{array}{|c|c|} \hline \text{Marge Simpson} & \text{Lisa Simpson} \\ \hline 4 & 4 \\ \hline \text{Bart Simpson} & \text{Marge Simpson} \\ \hline 3 & 3 \\ \hline \end{array} \right) = 3$$

# Distance Between Clusters

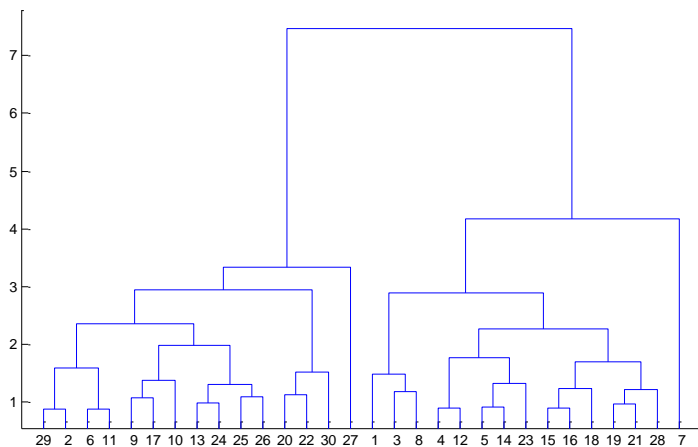
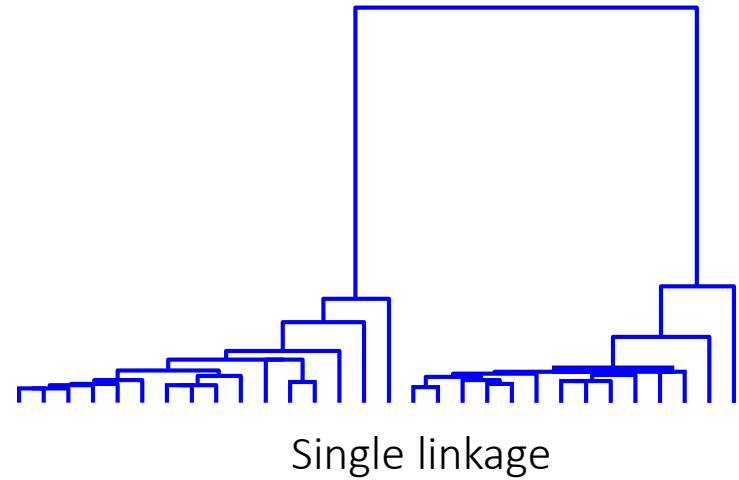
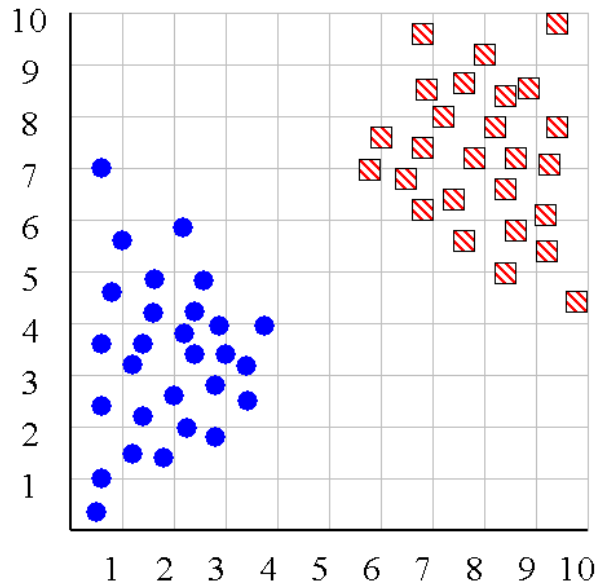
**Complete linkage (furthest neighbour):** In this method the distance between clusters is determined by the greatest distance between any two objects in the different clusters (i.e., by the *furthest* neighbours)


$$\text{dist\_CL}(\text{Cluster 1}, \text{Cluster 2}) = \max ( \begin{array}{|c|c|} \hline 4 & 4 \\ \hline 3 & 3 \\ \hline \end{array} ) = 4$$

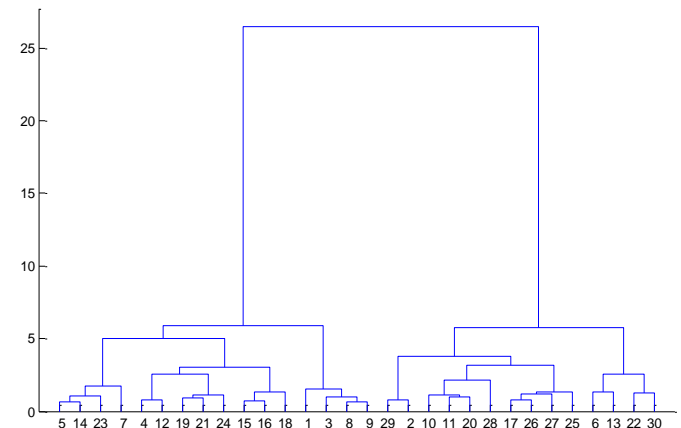
**Group average linkage:** This approach calculates the distance between two clusters as the average distance between all pairs of objects in the two different clusters.


$$\text{dist\_AL}(\text{Cluster 1}, \text{Cluster 2}) = \text{mean} ( \begin{array}{|c|c|} \hline 4 & 4 \\ \hline 3 & 3 \\ \hline \end{array} ) = 3.5$$

# Differences between Linkage Methods



Average linkage

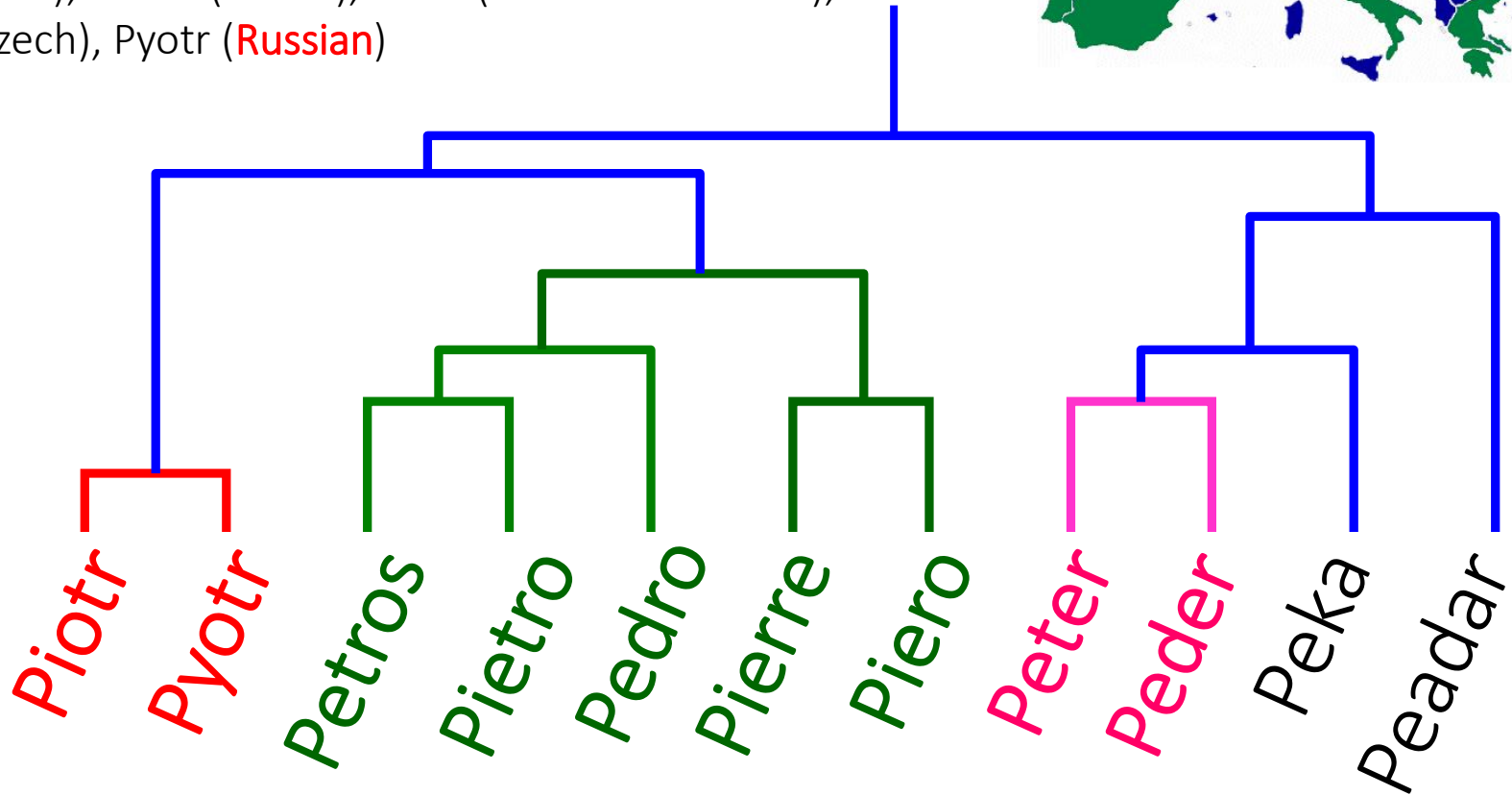
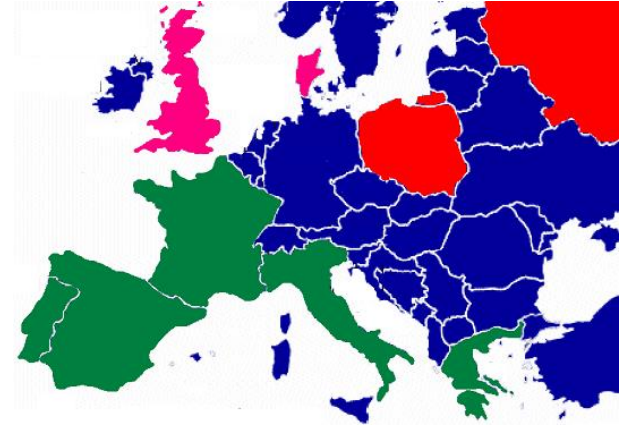


Complete linkage

# Results can be Informative and Interpretable

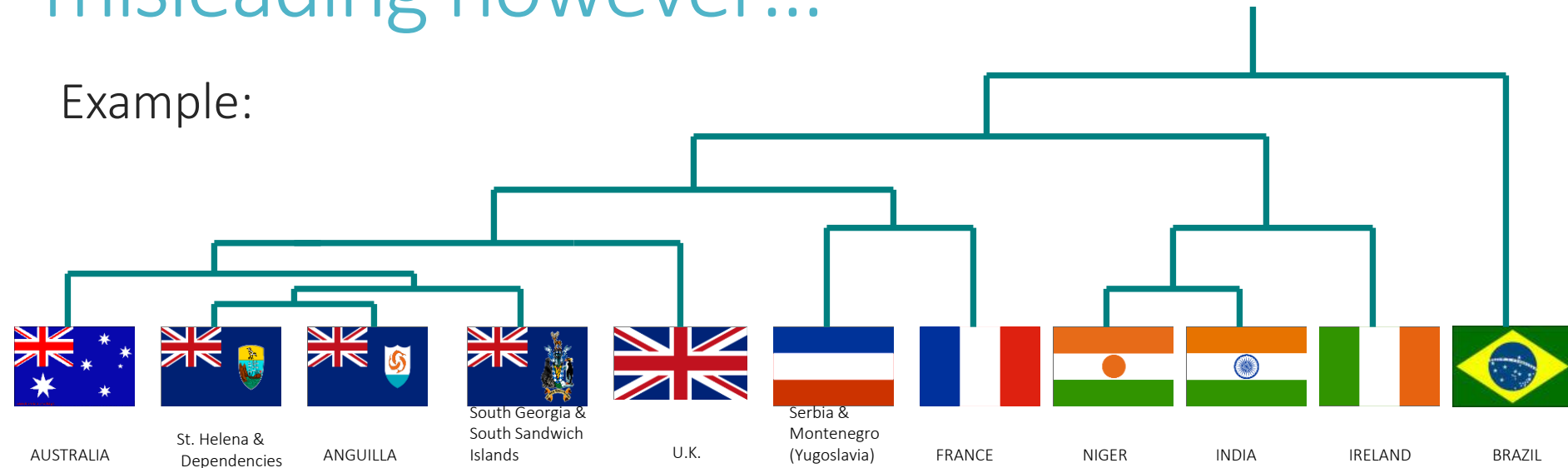
## Pedro (Portuguese/Spanish)

Petros (Greek), Peter (English), Piotr (Polish),  
Peadar (Irish), Pierre (French), Peder (Danish), Peka  
(Hawaiian), Pietro (Italian), Piero (Italian Alternative),  
Petr (Czech), Pyotr (Russian)



# Results can also be spurious and misleading however...

Example:



- The clustering of Australia, St. Helena, etc. is meaningful as each of these countries are former UK colonies
- Niger and India are one of the best matches too – but there is no link between these countries or their flags (other than the colours)

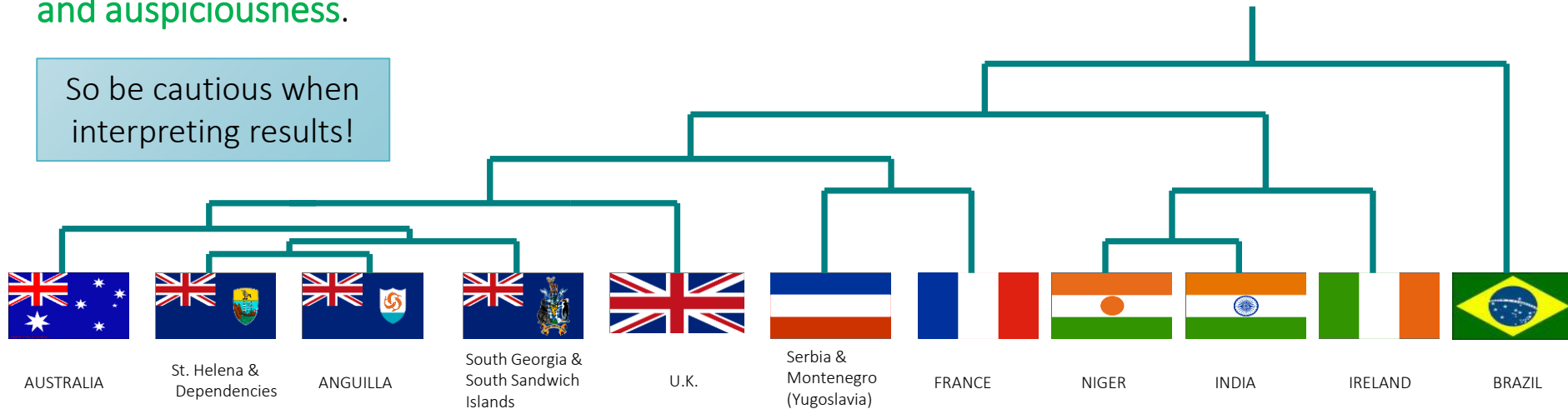
# Fun with Flags



The flag of Niger is **orange** over **white** over **green**, with an orange disc on the central white stripe, symbolizing the sun. **The orange stands the Sahara desert**, which borders Niger to the north. **Green stands for the grassy plains of the south** and west and for the River Niger which sustains them. It also stands for fraternity and hope. **White** generally symbolizes purity and hope.

The Indian flag is a horizontal tricolor in equal proportion of deep **saffron** on the top, **white** in the middle and **dark green** at the bottom. In the center of the white band, there is a wheel in **navy blue** to indicate the Dharma Chakra, the wheel of law in the Sarnath Lion Capital. This center symbol or the 'CHAKRA' is a symbol dating back to 2nd century BC. The **saffron stands for courage** and sacrifice; the **white**, for purity and truth; **the green for growth and auspiciousness**.

So be cautious when interpreting results!



# Hierarchical Clustering: Summary

## Strengths:

- No need to set  $K$  – everything is combined into one group, and clusters of similar objects can be observed in the dendrograms
- The hierarchical nature of the relationships map well onto human intuition and dendrograms are interpretable (be careful when inferring meaning however!)

## Weaknesses:

- They do not scale well – time complexity is at least  $O(n^2)$
- Like any heuristic search algorithms – local optima can be a problem
- As above – results are subjective so be careful about inferring meaning

# Density-based Clustering

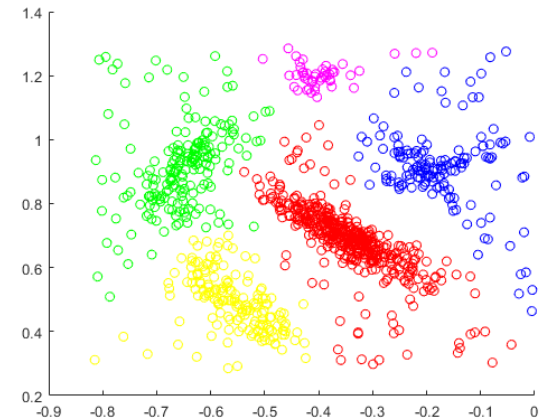
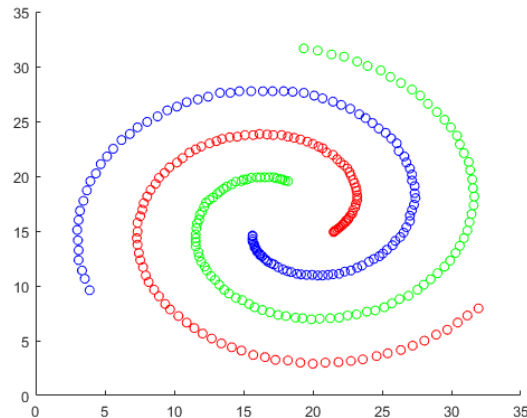
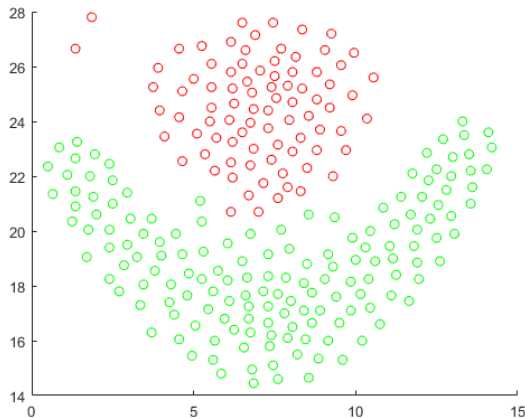
Density Peaks Algorithm

# Density-based

Partitional and hierarchical clustering algorithms are bad at finding non-convex clusters

Density-based methods use a localised measure of how close a point is to other points to form clusters

Popular density based approaches include DBScan, BIRCH and Density Peaks



# Density Peaks

A density based clustering algorithm with similarities to DBSCAN and PAM introduced in [1]

Relatively efficient, no need to repeat until convergence like K-Means and PAM

Easy to add solutions to deal with outliers

Allows for clusters of arbitrary sizes

No easy way to accurately select cluster centres automatically

Thresholds must be selected somehow

[1] Rodriguez, Alex, and Alessandro Laio. "Clustering by fast search and find of density peaks." *Science* 344.6191 (2014): 1492-1496.

# Informal Algorithm: Density Peaks

Five steps:

1. Calculate the **distance matrix** for the input data
2. Find the **local density** value for each data object ( $\rho$ ). This quantifies how many other objects are “close” to each object.
3. For each object, find the **distance to its nearest neighbour with a higher local density** ( $\delta$ )
4. Select cluster centres (medoid) using  $\rho$  and  $\delta$
5. Assign cluster labels to objects based on closest centre

# Calculating Densities

$\rho$  - density

$\delta$  - distance to nearest with higher  $\rho$

Two methods for finding the local density of a point, both using the distance to another point  $d$  and a distance threshold  $d_c$

Cutoff Kernel:

$$\rho = \sum \begin{matrix} \text{if } d < d_c: 1 \\ \text{else: } 0 \end{matrix}$$

i.e. in English: Cutoff kernel is a count of all other points that are closer than  $d_c$

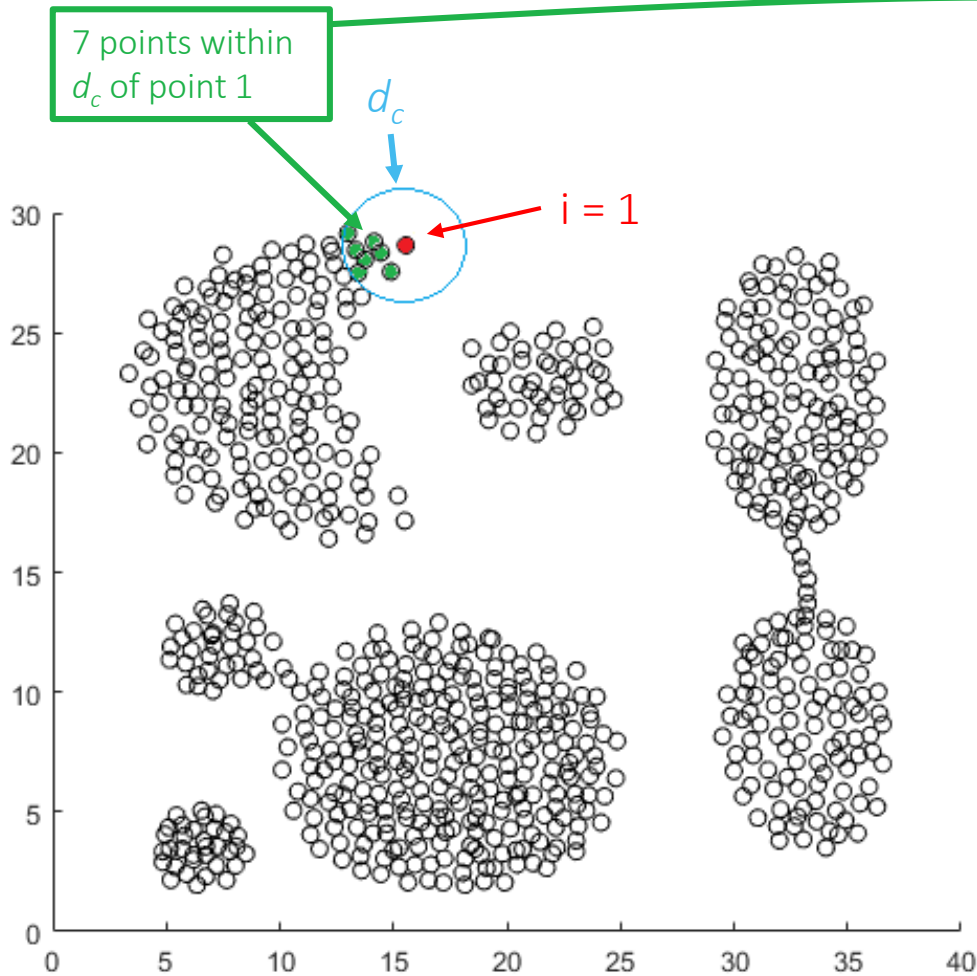
Gaussian Kernel:

$$\rho = \sum \exp(-(d^2/d_c^2))$$

You don't need to learn how to do this one but be aware that it exists and what it does on a high level

$d_c$  is chosen so that average number of neighbors is around 1-2% of the total number of points in the dataset

# Local Density of Individual Points



Cutoff Kernel

$i$	$\rho_i$
1	7
2	15
3	15
4	15
5	20
6	21
7	17
8	24
9	23
10	26

Gaussian Kernel

$i$	$\rho_i$
1	7.774
2	12.507
3	13.564
4	13.296
5	17.866
6	17.988
7	15.004
8	20.626
9	19.043
10	23.886

2. Find the local density value for each data object ( $\rho$ ). This quantifies how many other objects are "close" to each object.

# Higher Density Nearest Neighbours

Next the distance to points of higher local density must be found

This is most easily done by sorting the points by their local densities in descending order and looking at the ones above each point

Special case: the  $\delta$  value (distance) of the point with the highest local density is set to max  $\delta$  value from all points

$i$	$\rho_i$	$\delta_i$	$NN_i$
315	48.908	291.429	-1
319	48.898	0.529	315
314	48.884	0.429	315
348	48.879	0.686	314
347	48.819	0.492	348
260	48.792	7.534	314
346	48.777	0.178	347
259	48.743	0.355	260
382	48.702	0.702	259
320	48.665	0.178	319

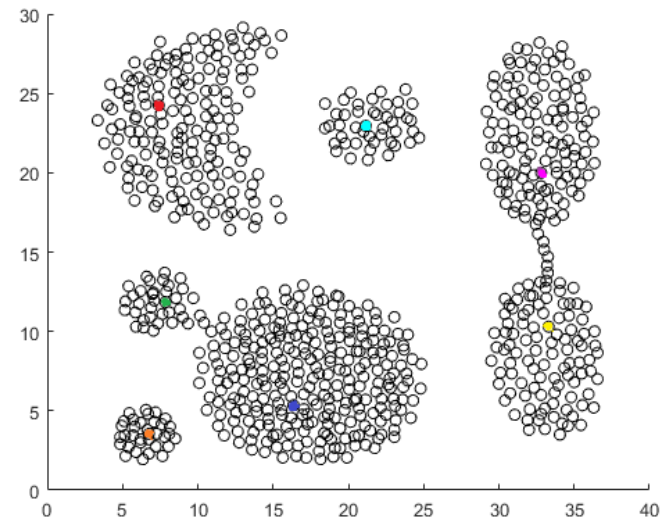
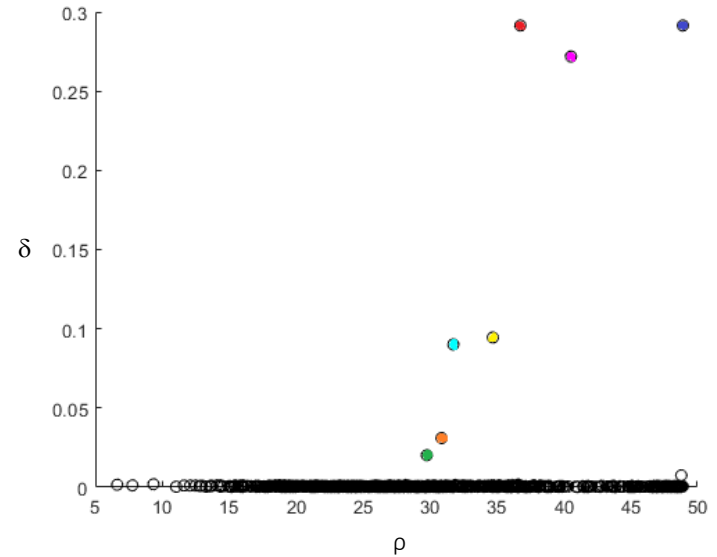
3. For each object, find the distance to its nearest neighbour with a higher local density ( $\delta$ )

# Selecting Cluster Centers

Points with a high  $\rho$  and  $\delta$  are likely candidates for cluster centres (this can still be subjective, of course)

Plotting  $\rho$  and  $\delta$  on a graph is a good way to visualise this; the closer to the top-right of the graph, the "better" candidate the point is as a medoid

Alternative: multiply  $\rho$  and  $\delta$  together into a 1d array and sort



# Selecting Cluster Centers

Points with a high  $\rho$  and  $\delta$  are likely candidates for cluster centres (this can still be subjective, of course)

Plotting  $\rho$  and  $\delta$  on a graph is a good way to visualise this; the closer to the top-right of the graph, the "better" candidate the point is as a medoid

Alternative: multiply  $\rho$  and  $\delta$  together into a 1d array and sort

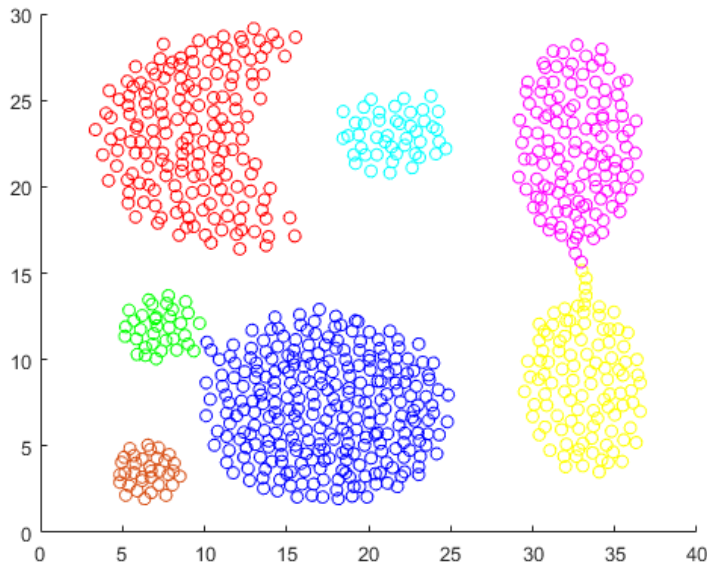
$i$	$\rho_i$	$\delta_i$	$\rho_i * \delta_i$
315	48.908	291.429	14253.21
319	48.898	0.529	25.87
314	48.884	0.429	20.97
348	48.879	0.686	33.53
347	48.819	0.492	24.02
260	48.792	7.534	367.60
346	48.777	0.178	8.68
259	48.743	0.355	17.30
382	48.702	0.702	34.19
320	48.665	0.178	8.66

# Assigning Cluster Labels

Finally cluster membership is assigned

If the point is a cluster centre add it to its own cluster, else add it to the cluster of its nearest neighbour of higher density

For example, say 315 and 260 we selected as cluster centres:



Medoid

Medoid

$i$	$\rho_i$	$\delta_i$	$NN_i$
315	48.908	291.429	-1
319	48.898	0.529	315
314	48.884	0.429	315
348	48.879	0.686	314
347	48.819	0.492	348
260	48.792	7.534	314
346	48.777	0.178	347
259	48.743	0.355	260
382	48.702	0.702	259
320	48.665	0.178	319

# Dealing with Outliers

Two proposed approaches:

Threshold:

Simply sets a lower threshold for the local density value of a point. If it is less than this value (i.e. not close to many data) then it is treated as an outlier and not assigned to a cluster

Halo:

The halo method separates the clusters into two components, the cluster core and cluster halo

For each cluster the highest density of the points bordering other clusters is taken, points with a local density less than this are part of the clusters halo and suitable to be considered as noise

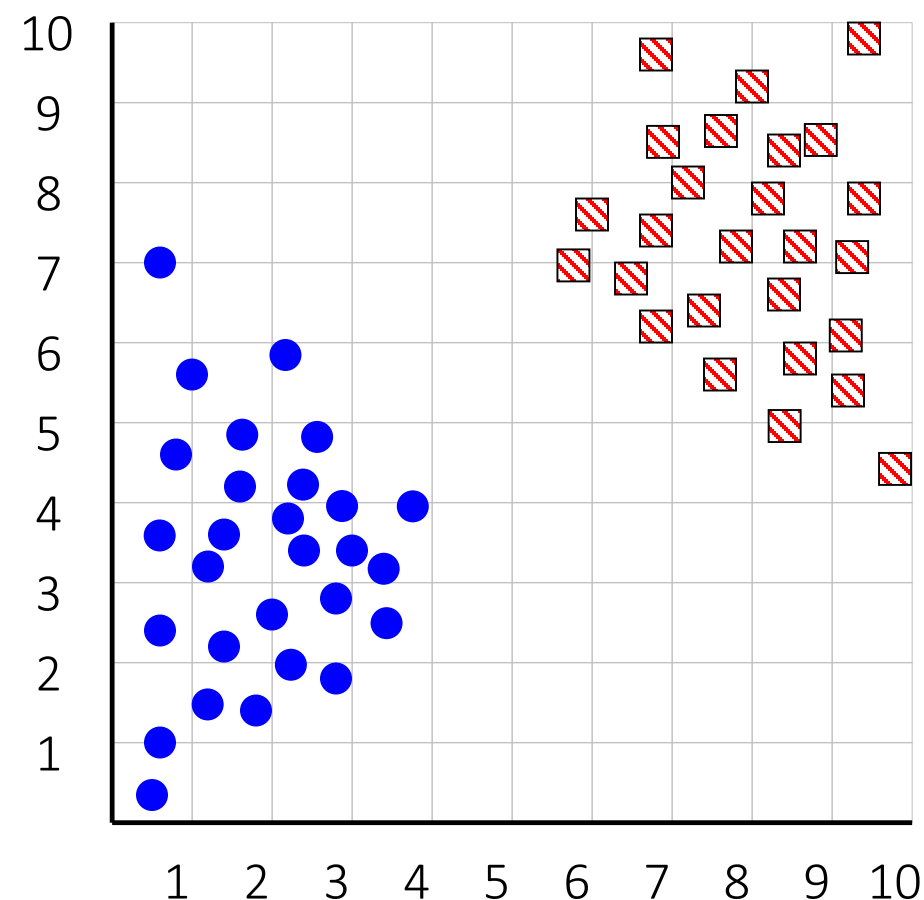
# Further Points and Final Matters

Finding the correct number of clusters ( $k$ )

Other types of clustering algorithm

# How do we find the *right* number of clusters?

This is subjective and generally unsolved. There are methods available for approximating this however.



Although this data is labelled, imagine that we do **NOT** know the class labels. We are only clustering on the X and Y axis values.

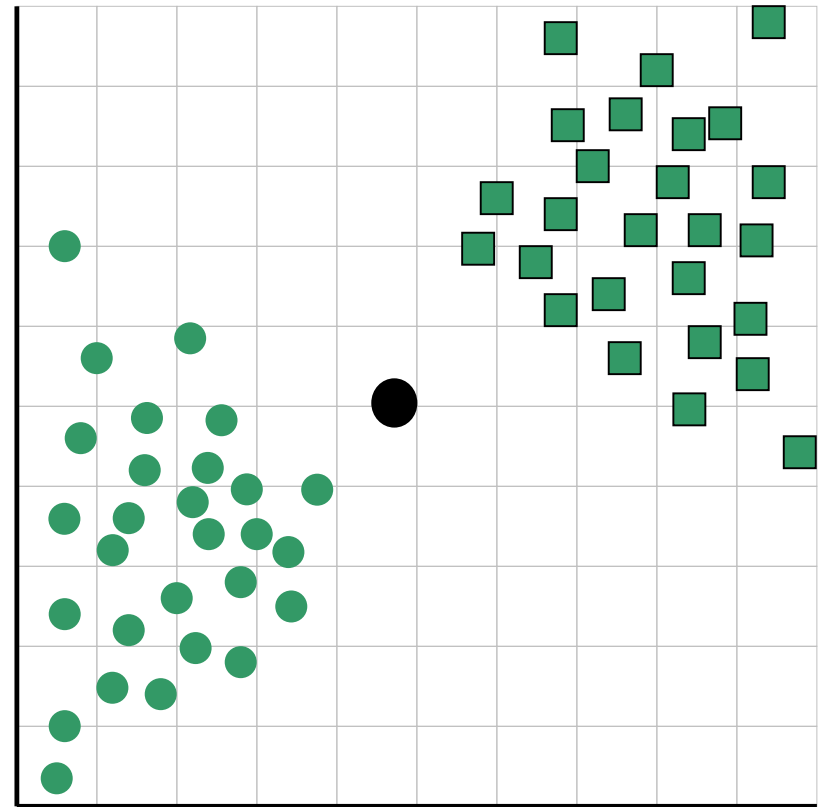
Can we detect that the correct number of clusters is 2?

# Objective Function

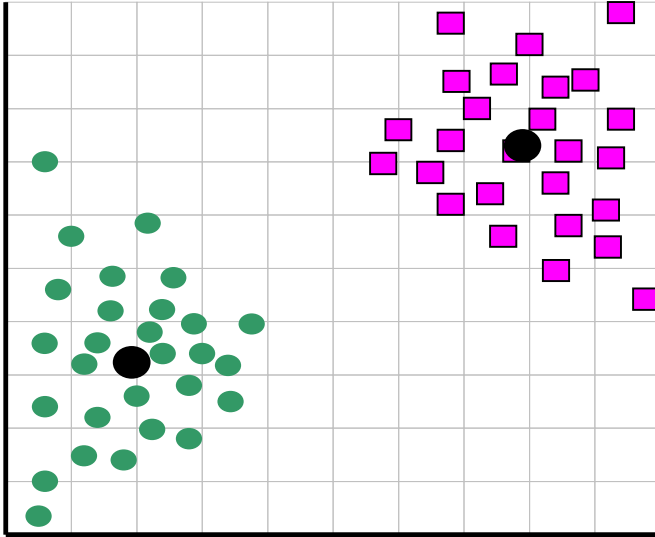
We need to define an **objective function** that summarises the quality of a clustering solutions

We could use the sum of distances from elements to the centroid (e.g. the within-cluster sum)

When  $k=1$ , there is only one centroid for all data. Let's say the sum in this case is **873.0**

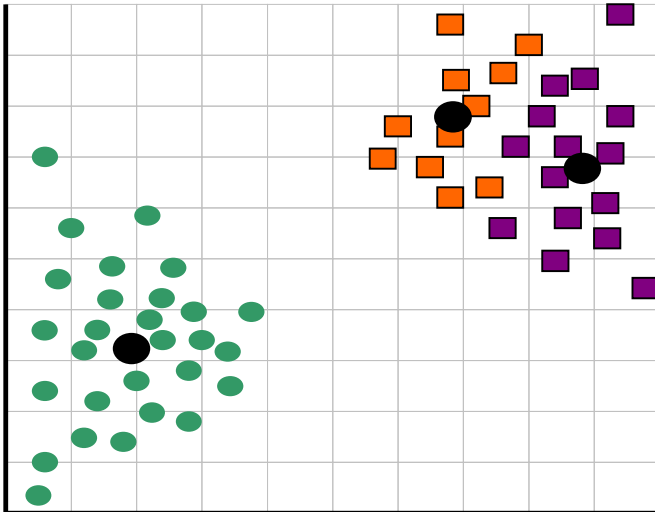


# Objective Function



When  $k = 1$ , the objective function was  
870.0

When  $k = 2$ , the objective function is  
173.1



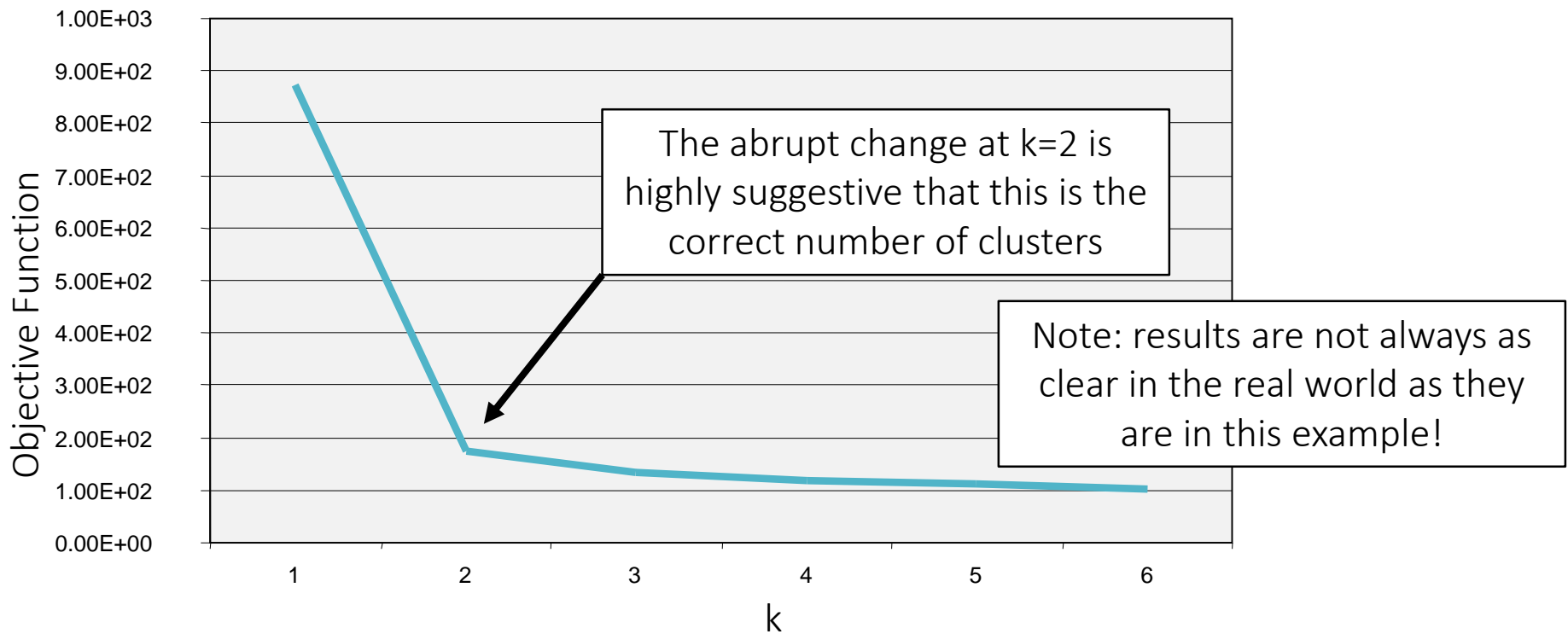
When  $k = 3$ , the objective function is  
133.6

Note: adding more clusters **will always decrease this objective function!**  
(when  $k=n$ , the objective function would be 0...)

# Plotting the Objective Function

It's not an exact approach, but if we plot the objective function for different values of  $k$  then we can look for a change point

This is known as **knee finding** or **elbow finding**



# Other Types of Clustering Algorithms

**Density-based** – e.g. Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)

- BIRCH initially scans the data and creates an in-memory R-tree (serving as a multi-level compression of the data that attempts to preserve the inherent clustering of the data)
- The notion is that similar items are compressed in similar ways
- An arbitrary clustering algorithm can then be applied to the leaf nodes of the tree

**Probabilistic** – e.g. Expectation Maximisation (EM)

- Iterates between two steps of assigning points to  $k$  clusters, then estimating model parameters for the clusters

**Spectral Clustering**

- Transform the data using an eigenvalue decomposition, cluster based on eigenvalues

# Summary

Measuring similarity/distance between data is key

Preparing the data adequately may be critical to ensure effective clustering

- E.g. standard normalisation

We've covered two common types of clustering:

- Partitional
  - K-means
  - K-medoid (PAM)
- Hierarchical
  - Top-down vs Bottom-up
  - Single/average/group linkage
- Density
  - Density Peaks with cut-off kernel