Ollscoil na Gaillimhe
University of Galway

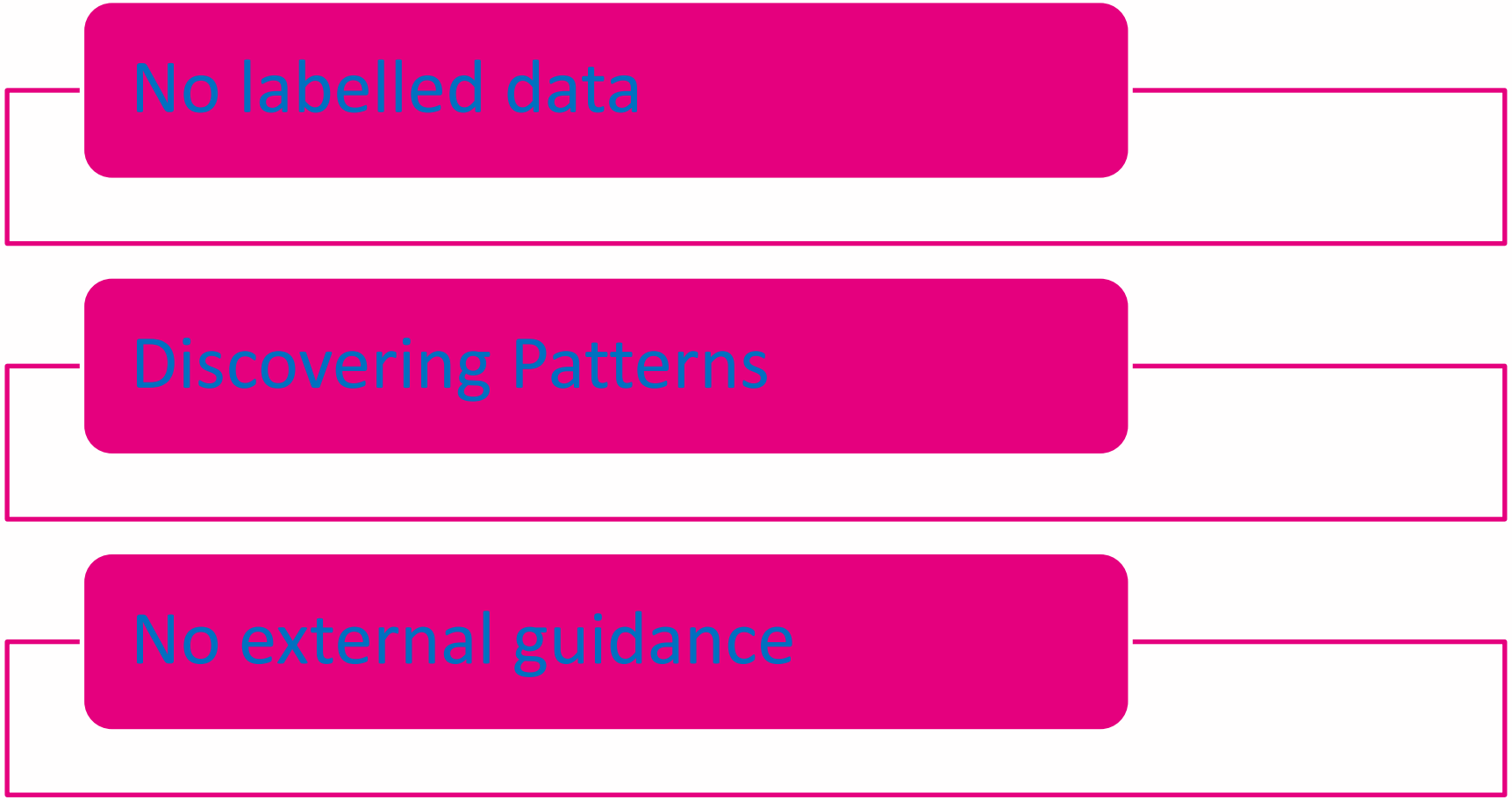
CT4101
Machine Learning

Dr. Frank Glavin
Room CSB-3004, Computer Science Building
Frank.Glavin@University*of*Galway.ie

School of Computer Science

University *of* Galway.ie

What is Unsupervised Learning?



Supervised vs. Unsupervised Learning – recap

- To date we have mainly looked only at supervised learning tasks, where we have **labelled** data giving ground truths that we can compare predictions against:
  - **Classification** tasks, where the labels take the form of one class from a finite number of possible discrete classes
  - **Regression** tasks, where the labels take the form of a floating-point number

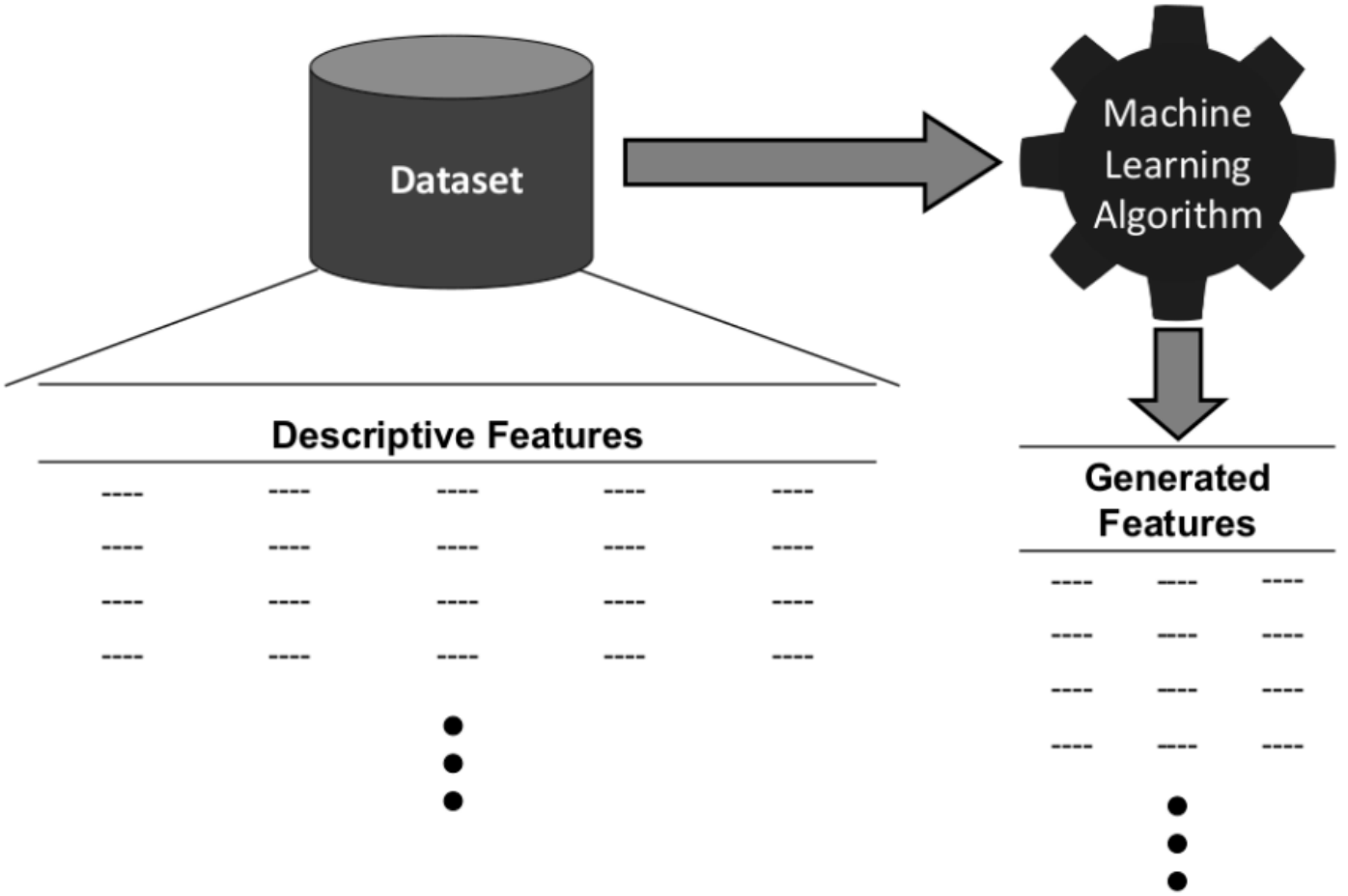
- In unsupervised learning tasks, there are no labels
- Our goal in unsupervised learning is to develop models based on the underlying structure within the descriptive features in a dataset.
- This structure is typically captured in new generated features that can be appended to the original dataset and so **augment** or **enrich** it

Supervised versus Unsupervised

Supervised Learning	
Purpose:	Task driven
Data:	Pre-categorised data
Objective:	Predictive models
Classification	**Regression**
Fraud detection	Market forecasting

Unsupervised Learning	
Purpose:	Data driven
Data:	Unlabelled data
Objective:	Pattern recognition
Clustering	**Association**
Targeted marketing	Customer recommendations

Unsupervised learning overview

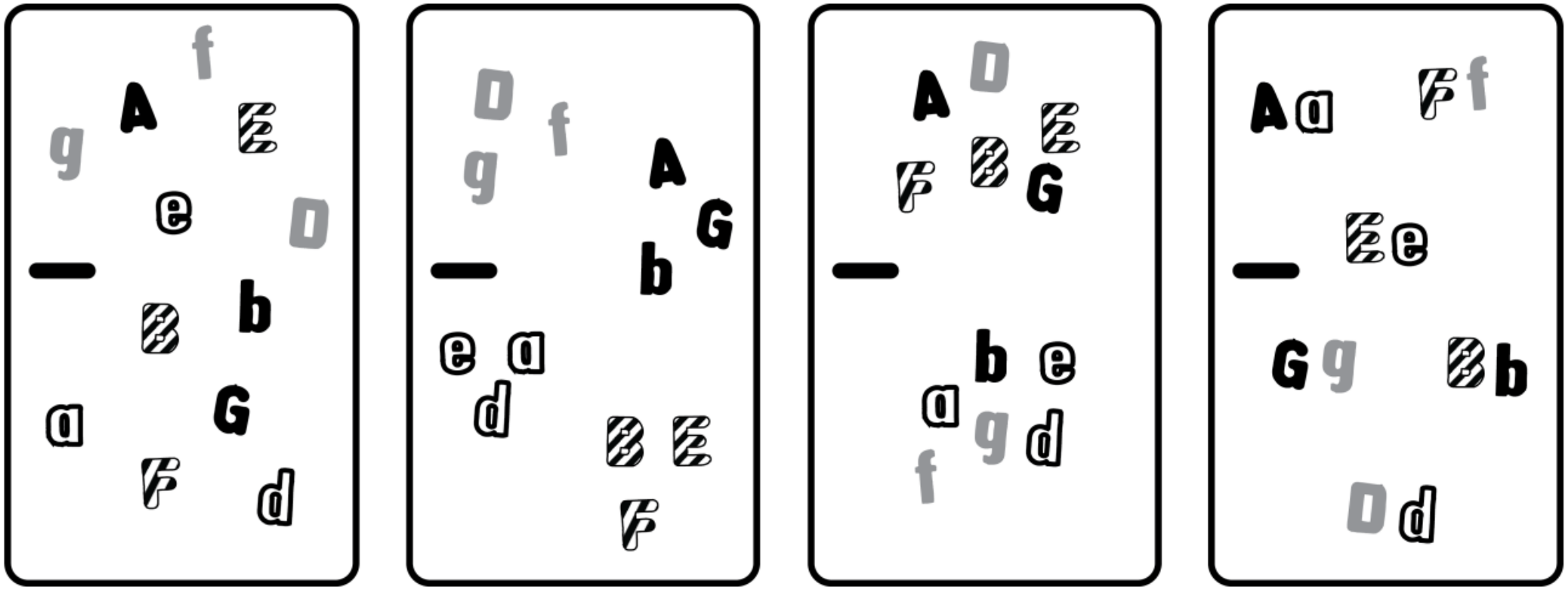


Unsupervised learning task examples

- **Clustering**:
  - Partitions the instances in a dataset into groups, or clusters, that are similar to each other. The end result of clustering is a single new generated feature that indicates the cluster that an instance belongs to, and the generation of this new feature is typically the end goal of the clustering task.
  - Common application: **customer segmentation** with which organizations attempt to discover meaningful groupings into which they can group their customers so that targeted offers or treatments can be designed.
  - Also commonly used in the domain of **information retrieval** to improve efficiency of the retrieval process. Documents that are associated with each other are assigned to the same cluster.

- **Representation learning**:
  - The goal is to create a new way to represent the instances in a dataset, usually with the expectation that this new representation will be more useful for a later, usually supervised, machine learning process.
  - Usually achieved using specific types of deep learning models called **auto-encoders**.
  - Advanced topic, we will not discuss in detail in this module due to time constraints.

Clustering example – how to organise letters?



The letters could be grouped by different features, e.g., **colour**, **case**, or **character**
Is there a ‘correct’ grouping? We don’t have a ground truth available – all groupings are valid in this case as they each highlight different characteristics present in the dataset

Clustering using the *k*-means algorithm

- We have already covered many of the fundamentals required to tackle clustering problems, e.g.:
  - Feature spaces
  - Measuring similarity using distance metrics (as we did with *k*-nearest neighbours)

- The ***k*-means** clustering algorithm is the most well-known approach to clustering
  - Relatively simple to understand
  - Computationally efficient
  - Simple to implement but also usually very effective

Clustering using the *k*-means algorithm

$$\sum_{i=1}^{n} \min_{\mathbf{c}_1,\ldots,\mathbf{c}_k} Dist(\mathbf{d}_i, \mathbf{c}_j)$$

- Given a dataset, D, consisting of *n* instances, $d_1 \ldots d_n$, where $\boldsymbol{d}_i$ is a set of *m* descriptive features
- The goal when applying *k*-means is to divide this dataset into *k* disjoint clusters, $C_1 \ldots C_k$.
- The number of clusters *k* **is an input to the algorithm**
- The division into clusters is achieved by minimising the result of the above sum
- $\boldsymbol{c}_1 \ldots \boldsymbol{c}_k$ are the centroids of the clusters; they are vectors containing the coordinates in the feature space of each cluster centroid
- $Dist$ is a distance metric (defined in the same way as we defined distance metrics previously when studying *k*-NN, i.e., a way to measure similarity between instances in a dataset)

Pseudocode description of *k*-means

Pseudocode description of the ***k*-means clustering** algorithm.

Require: a dataset $\mathcal{D}$ containing n training instances, $\mathbf{d}_1, \ldots, \mathbf{d}_n$
Require: the number of clusters to find k
Require: a distance measure, $Dist$, to compare instances to cluster centroids

1: Select k random cluster centroids, $\mathbf{c}_1$ to $\mathbf{c}_k$, each defined by values for each descriptive feature, $\mathbf{c}_i =< \mathbf{c}_i[1], \ldots, \mathbf{c}_i[m] >$
2: **repeat**
3: calculate the distance of each instance, $\mathbf{d}_i$, to each cluster centroid, $\mathbf{c}_1$ to $\mathbf{c}_k$, using $Dist$
4: assign each instance, $\mathbf{d}_i$, to belong to the cluster, $\mathcal{C}_i$, to whose cluster centroid, $\mathbf{c}_i$, it is closest
5: update each cluster centroid, $\mathbf{c}_i$, to the average of the descriptive feature values of the instances that belong to cluster $\mathcal{C}_i$
6: **until** no cluster reassignments are performed during an iteration

Issues to consider when applying *k*-means

Choice of distance metric?

Common to use Euclidean distance. Other distance metrics possible but may break convergence guarantees. Other clustering algorithms (e.g., *k*-medoids) have been developed to address this problem

Normalisation of data

As we are measuring similarity using distance, as with *k*-NN normalising the data beforehand is **very important** when the values of the attributes have different ranges, otherwise attributes with the largest ranges will dominate calculations

How to identify when convergence happens?

When no cluster memberships change during a full iteration of the algorithm

How to choose a value for *k*?

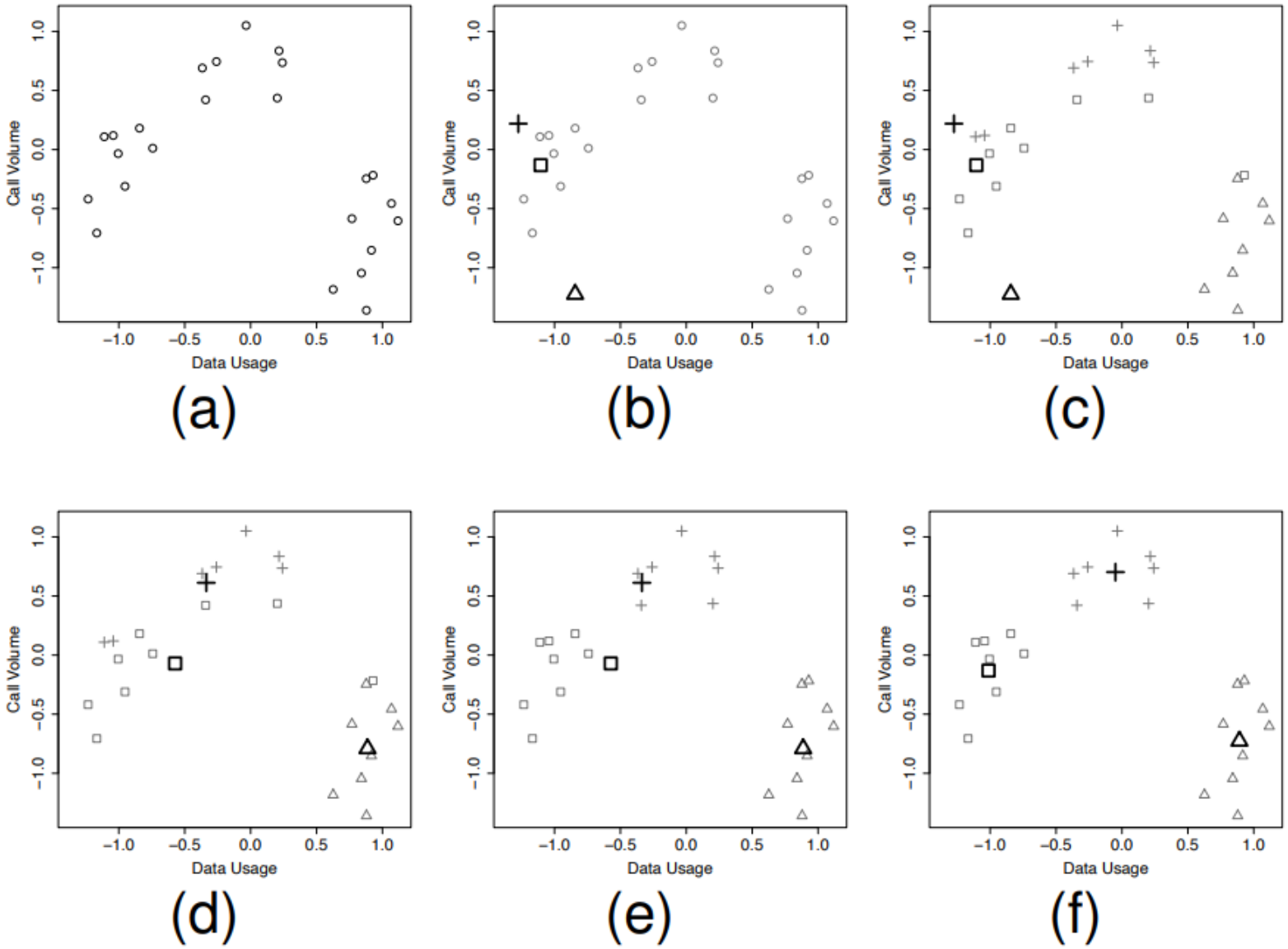
We will discuss this later in the coming slides

Example of **normalised** mobile phone customer dataset with first 2 iterations of *k*-means

ID	Data Usage	Call Volume	Cluster Distances Iter. 1 $Dist(\mathbf{d}_i, \mathbf{c}_1)$	$Dist(\mathbf{d}_i, \mathbf{c}_2)$	$Dist(\mathbf{d}_i, \mathbf{c}_3)$	Iter. 1 Cluster	Cluster Distances Iter. $Dist(\mathbf{d}_i, \mathbf{c}_1)$	$Dist(\mathbf{d}_i, \mathbf{c}_2)$
1	-0.9531	-0.3107	0.2341	0.9198	0.6193	$\mathcal{C}_1$	0.4498	1.9014
2	-1.1670	-0.7060	0.5770	0.6108	0.9309	$\mathcal{C}_1$	0.87	2.0554
3	-1.2329	-0.4188	0.3137	0.8945	0.6388	$\mathcal{C}_1$	0.7464	2.152
4	1.0684	-0.4560	2.1972	2.06	2.438	$\mathcal{C}_2$	1.6857	0.3813
5	-1.1104	0.1090	0.2415	1.3594	0.1973	$\mathcal{C}_3$	0.5669	2.1905
6	-0.8431	0.1811	0.4084	1.405	0.4329	$\mathcal{C}_1$	0.3694	1.9842
7	-0.3666	0.6905	1.1055	1.9728	1.0231	$\mathcal{C}_3$	0.7885	1.9406
8	0.9285	-0.2168	2.0351	2.0378	2.2455	$\mathcal{C}_1$	1.5083	0.5759
9	1.1175	-0.6028	2.2715	2.0566	2.529	$\mathcal{C}_2$	1.772	0.298
10	0.8404	-1.0450	2.1486	1.693	2.4636	$\mathcal{C}_2$	1.7165	0.258
11	-1.005	-0.0337	0.1404	1.2012	0.3692	$\mathcal{C}_1$	0.4339	2.0376
12	0.2410	0.7360	1.6017	2.2398	1.6013	$\mathcal{C}_3$	1.1457	1.6581
13	0.2021	0.4364	1.4253	1.9619	1.4925	$\mathcal{C}_1$	0.9259	1.4055
14	0.2153	0.8360	1.6372	2.3159	1.6125	$\mathcal{C}_3$	1.2012	1.7602
15	0.8770	-0.2459	1.985	1.9787	2.201	$\mathcal{C}_2$	1.4603	0.5454
16	-0.0345	1.0502	1.595	2.4136	1.4929	$\mathcal{C}_3$	1.2433	2.0589
17	0.8785	-1.3601	2.3325	1.727	2.6698	$\mathcal{C}_2$	1.9413	0.569
18	0.9164	-0.8517	2.1454	1.7984	2.4383	$\mathcal{C}_2$	1.6815	0.0674
19	-1.0423	0.1193	0.2593	1.3579	0.2525	$\mathcal{C}_3$	0.5065	2.133
20	-0.7426	0.0119	0.3899	1.2399	0.5706	$\mathcal{C}_1$	0.1889	1.8164
21	0.6259	-1.1834	2.0248	1.4696	2.3616	$\mathcal{C}_2$	1.6355	0.4709
22	0.7684	-0.5844	1.927	1.7338	2.195	$\mathcal{C}_2$	1.4362	0.2382
23	-0.2596	0.7450	1.2183	2.0535	1.1432	$\mathcal{C}_3$	0.8736	1.9167
24	-0.3414	0.4215	0.9432	1.7202	0.9548	$\mathcal{C}_1$	0.5437	1.7259

Plot of the mobile phone customer dataset

- Large symbols represent clusters, small symbols represent individual data points
- Clear to a human viewer that there are 3 natural clusters within this dataset, however we require an algorithm such as k-means clustering to find them automatically
- Generally, not possible to determine the correct number of clusters by eye in real world **high-dimensional** datasets



Worked example - determining the cluster centroid

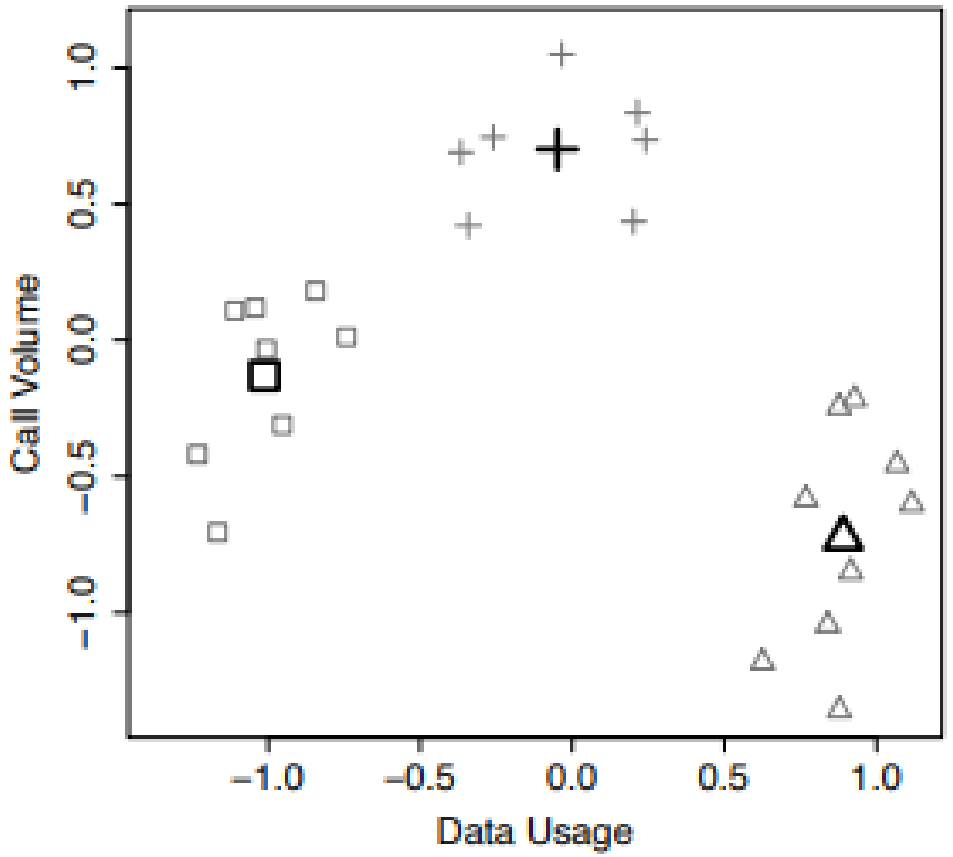
Each cluster centroid is then updated by calculating the mean value of each descriptive feature for all instances that are a member of the cluster

$$
\begin{aligned}
\mathbf{c}_1[\text{DATA USAGE}] &= (-0.9531 + -1.167 + -1.2329 + -0.8431 + 0.9285 \\
&\quad + -1.005 + 0.2021 + -0.7426 + -0.3414)/9 \\
&= -0.5727 \\
\mathbf{c}_1[\text{CALL VOLUME}] &= (-0.3107 + -0.706 + -0.4188 + 0.1811 + -0.2168 \\
&\quad + -0.0337 + 0.4364 + 0.0119 + 0.4215)/9 \\
&= -0.0706
\end{aligned}
$$

Final *k*-means clustering on the mobile phone customer dataset

- Once the algorithm has completed, its **two outputs** are a vector of assignments of each instance in the dataset to one of the clusters, $C_1 \ldots C_k$, and the *k* cluster centroids, $\boldsymbol{c}_1 \ldots \boldsymbol{c}_k$
- The assignment of instances to clusters can then be used to enrich the original dataset with a new generated feature, the cluster memberships

$$\mathcal{C}_1 = \{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_5, \mathbf{d}_6, \mathbf{d}_{11}, \mathbf{d}_{19}, \mathbf{d}_{20}\}$$
$$\mathcal{C}_2 = \{\mathbf{d}_4, \mathbf{d}_8, \mathbf{d}_9, \mathbf{d}_{10}, \mathbf{d}_{15}, \mathbf{d}_{17}, \mathbf{d}_{18}, \mathbf{d}_{21}, \mathbf{d}_{22}\}$$
$$\mathcal{C}_3 = \{\mathbf{d}_7, \mathbf{d}_{12}, \mathbf{d}_{13}, \mathbf{d}_{14}, \mathbf{d}_{16}, \mathbf{d}_{23}, \mathbf{d}_{24}\}$$



Choosing initial cluster centroids

How to choose the initial set of cluster centroids?

Could choose *k* random initial cluster centroids as per the pseudocode

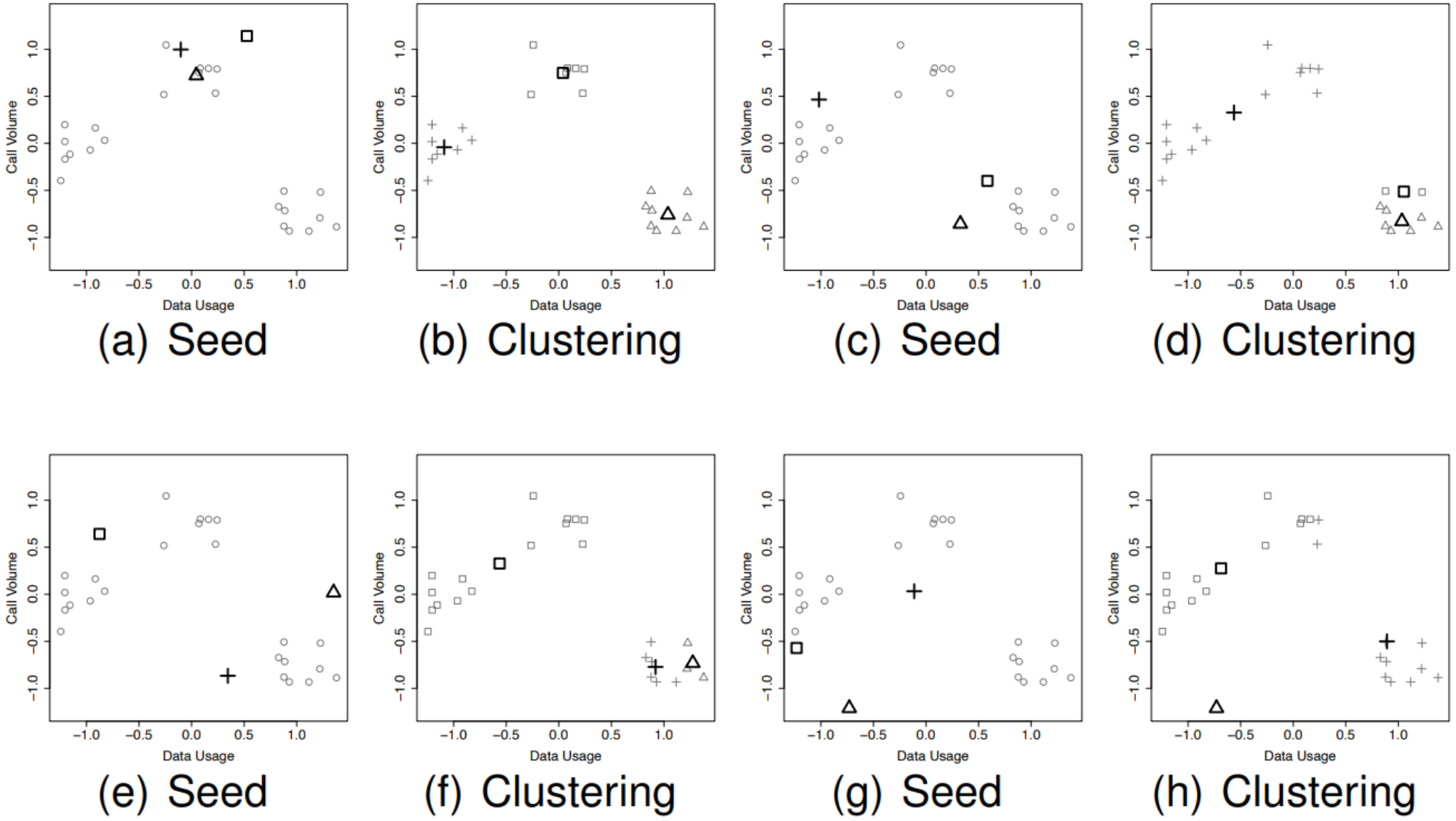
The choice of these initial cluster centroids (seeds), unfortunately, can have a big impact on the performance of the algorithm

Different randomly selected starting points can lead to different, often sub-optimal, clusterings (i.e., local minima)

As we will see in the example on the next slide, a particularly unlucky clustering could lead to a cluster having no members upon convergence! (easy fix – choose random instances in the dataset as the seeds)

An easy way to address this issue is to perform multiple runs of the k-means clustering algorithm starting from different initial centroids and then aggregate the results. In this way the most common clustering is chosen as the final result

The effect of different seeds on final clusters



More advanced approaches to select initial centroids

Main goals:

- Find initial centroids less likely to lead to sub-optimal clusterings
- Select initial centroids that allow the algorithm to converge much more quickly than when seeds are randomly chosen

E.g., ***k*-means++** algorithm (the default seed selection method in scikit-learn)

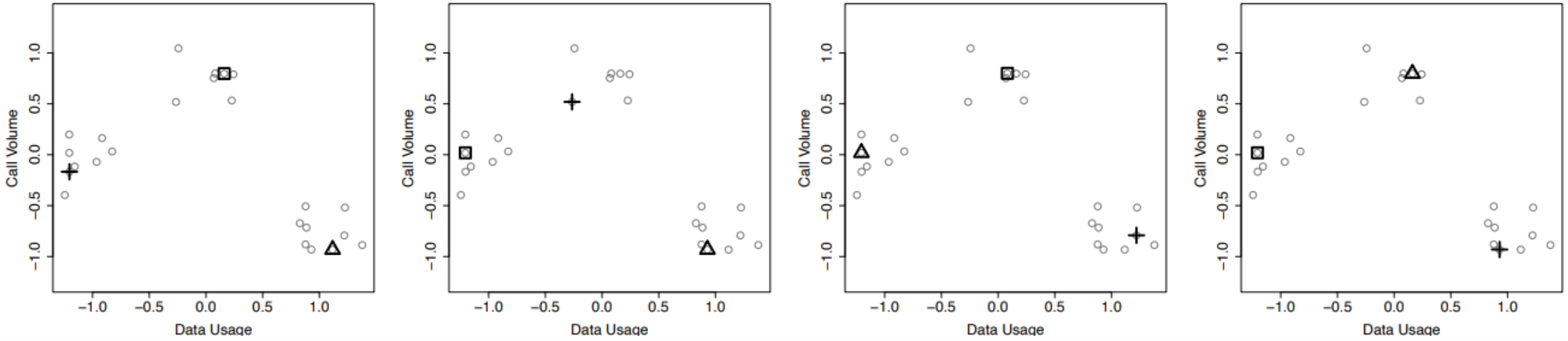
In this approach, an instance is chosen randomly (following a uniform distribution) from the dataset as the first centroid.

Subsequent centroids are then chosen randomly but following a distribution defined by the square of the distances between an instance and the nearest cluster centroid out of those found so far.

This means that instances far away from the current set of centroids are much more likely to be selected than those close to already selected centroids.

Examples of initial seeds chosen by *k*-means++

As before, using the mobile phone customers dataset with *k*=3
Instances from the dataset are being used as cluster centroids
Typically, there is good diversity across the feature space in the centroids selected
The *k*-means++ still stochastic; does not completely remove the possibility of a poor starting point that leads to a sub-optimal clustering
Therefore, should still try **running it multiple times** and pick the most common clustering



Pseudocode for *k*-means++

Pseudocode description of the **k-means++** algorithm.

Require: a dataset $\mathcal{D}$ containing n training instances, $\mathbf{d}_1, \ldots, \mathbf{d}_n$
Require: k, the number of cluster centroids to find
Require: a distance measure $Dist$ to compare instances to cluster centroids

1: choose $\mathbf{d}_i$ randomly (following a uniform distribution) from $\mathcal{D}$ to be the position of the initial centroid, $\mathbf{c}_1$, of the first cluster, $\mathcal{C}_1$
2: **for** cluster $\mathcal{C}_j$ in $\mathcal{C}_2$ to $\mathcal{C}_k$ **do**
3: for each instance, $\mathbf{d}_i$, in $\mathcal{D}$ let $Dist(\mathbf{d}_i)$ be the distance between $\mathbf{d}_i$ and its nearest cluster centroid
4: calculate a selection weight for each instance, $\mathbf{d}_i$, in $\mathcal{D}$ as $\frac{Dist(\mathbf{d}_i)^2}{\sum_{p=1}^{n} Dist(\mathbf{d}_p)^2}$
5: choose $\mathbf{d}_i$ as the position of cluster centroid, $\mathbf{c}_j$, for cluster $\mathcal{C}_j$ randomly following a distribution based on the selection weights
6: **end for**
7: proceed with *k*-means as normal using $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$ as the initial centroids.

Evaluating clustering

- More complicated evaluation challenge when conducting unsupervised learning compared to supervised learning

- All performance measures we discussed earlier for supervised learning relied on having **ground truth** labels available, which we can use to objectively measure performance using metrics like accuracy, error rate, RMSE, MAE, etc.

‘Good’ vs. ‘bad’ clusterings

We could use an idealised notion of what a ‘good’ clustering looks like, i.e.

- instances in the same cluster should all be relatively close together
- Instances belonging to different clusters should be far apart

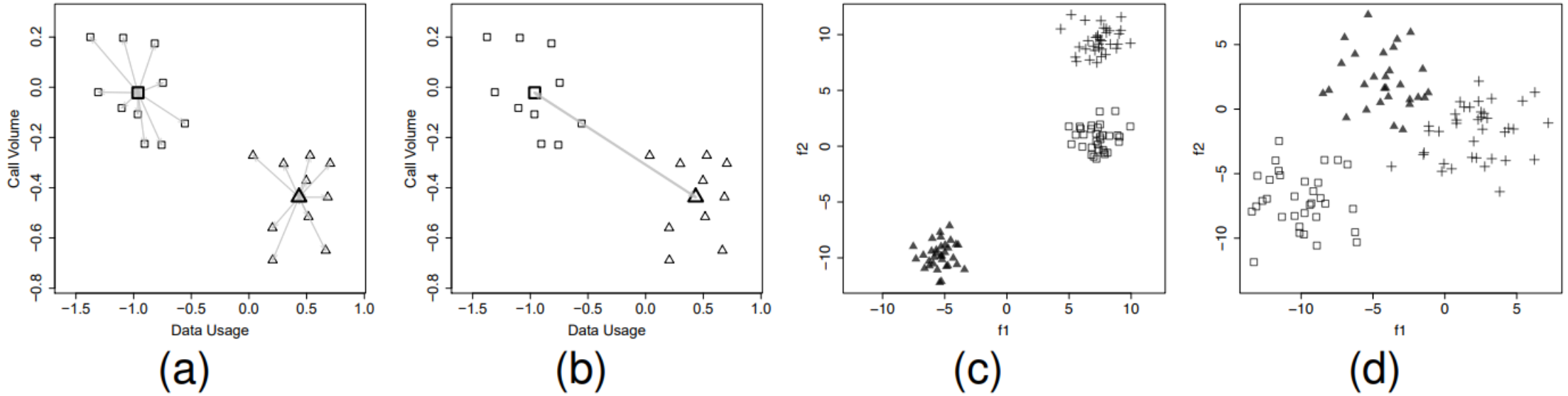


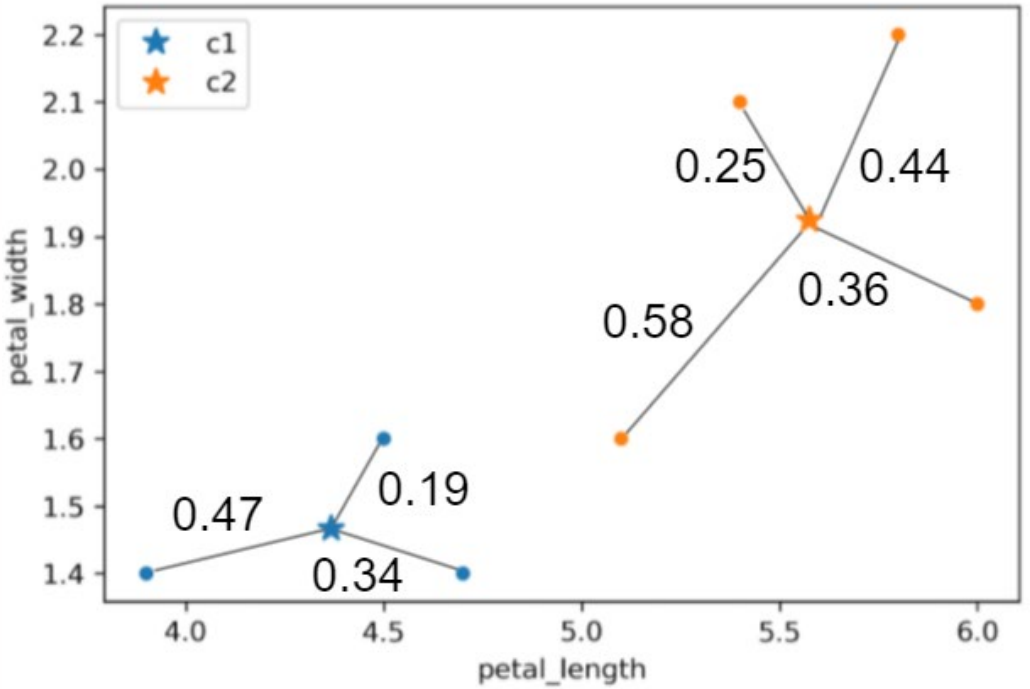
Figure 6: (a) Intra-cluster distance; (b) inter-cluster distance; (c) a *good* clustering; and (d) a *bad* clustering.

Loss function for clustering

A **loss function** is a function we wish to minimise to reduce error on a ML model

Inertia: sum of squared distances from each data point to its centre., alternatively referred to as the sum of squared errors (SSE)
The 'error' in this case is the distance between the centroid of the cluster and each data point in that cluster. To calculate, simply determine all distances from each data point to the centroid of its assigned cluster, square the distances and sum them up

Can compare different clusterings using *intertia* values



- E.g. on instances from the iris dataset
- Inertia/SSE = $0.47^2 + 0.19^2 + 0.34^2 + 0.25^2 + 0.58^2 + 0.36^2 + 0.44^2$

Using the elbow method to determine *k*

Plot the inertia/SSE (y-axis) vs the number of clusters *k* (x-axis)
Elbow method gives an indication of an appropriate *k* value – in this case *k*=3 looks appropriate
Other more complex methods available (e.g., silhouette) which we will not cover due to time constraints

N.B. – no definitive methods available to determine correct number of clusters. May need to be informed by deep knowledge of problem domain

Picking *k* – real world example (Andrew Ng)

Sometimes we can rely on *domain specific metrics* to guide the choice of *k*

E.g., perform 2 clusterings

- Cluster heights and weights of customers with K = 3 to design Small, Medium, and Large shirts.
- Cluster heights and weights of customers with K = 5 to design XS, S, M, L, and XL shirts.

To pick *k*:

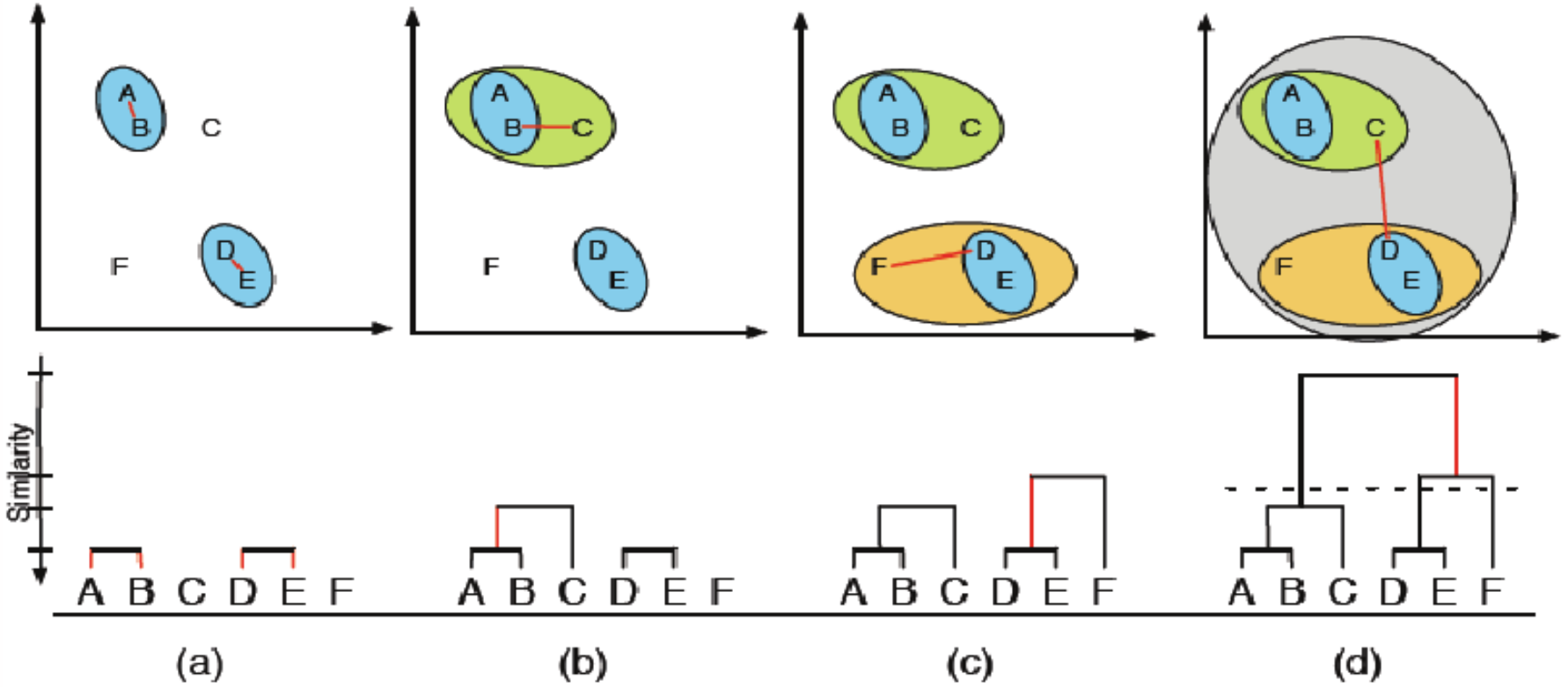
- Consider projected costs and sales for the 2 different *k* values
- Pick the value of *k* that maximizes profit.

Hierarchical clustering

Hierarchical clustering (HC) is a general family of clustering algorithms that build nested clusters by merging or splitting them successively.
This hierarchy of clusters is represented as a tree (or **dendrogram**).
The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample
Hierarchical clustering methods can sometimes be more computationally expensive than simpler algorithms (e.g., *k*-means)

Agglomerative hierarchical clustering (AHC) is an example of a HC method – we will not cover this algorithm in detail, just illustrate key differences with results given by *k*-means
AHC can find clusters that are not possible to find with *k*-means – examples on next slide
AHC starts by considering each instance in the dataset to be a member of its own individual cluster, then merging the most similar adjacent clusters in each iteration
We don't need to provide the number of clusters *k* in advance with AHC

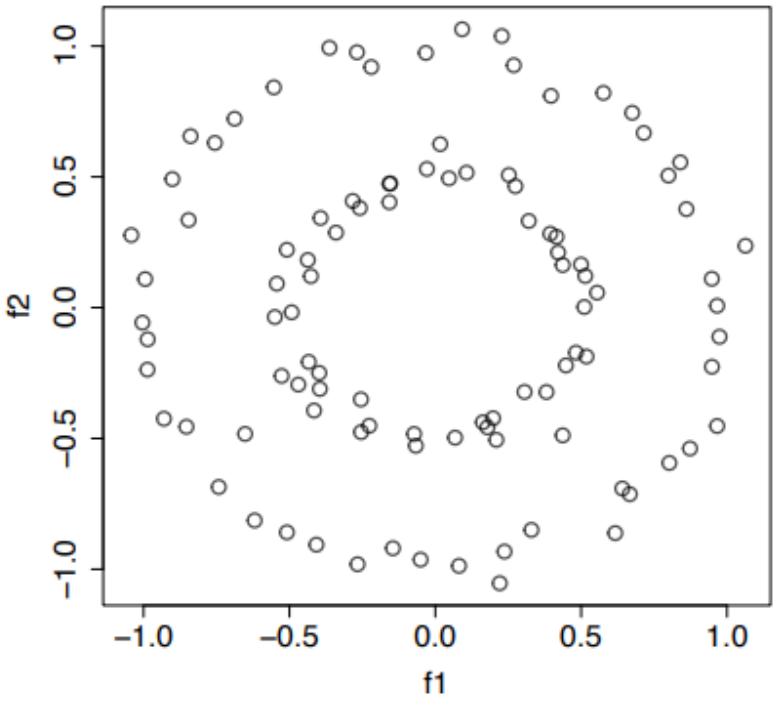
Agglomerative Hierarchical Clustering



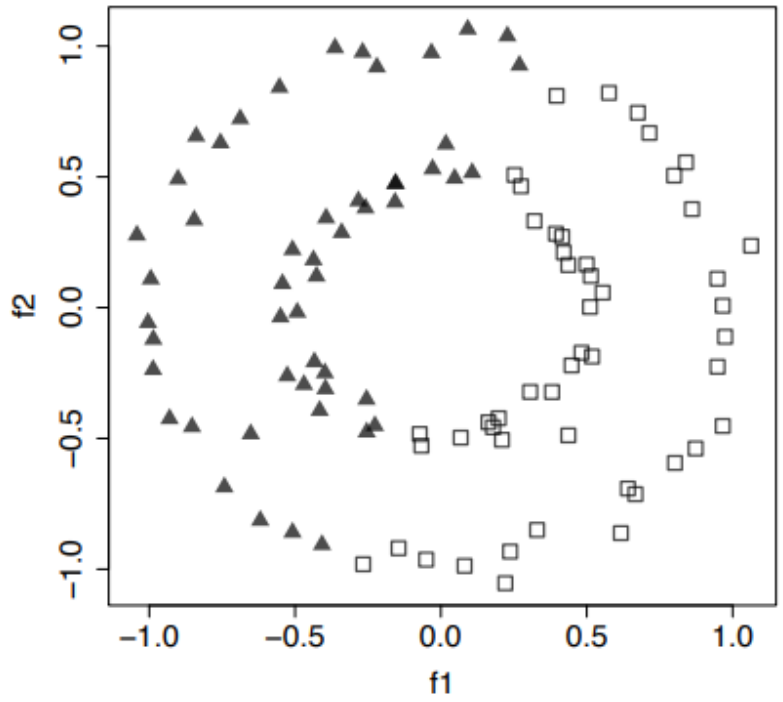
https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a

k-means vs. AHC on circles dataset

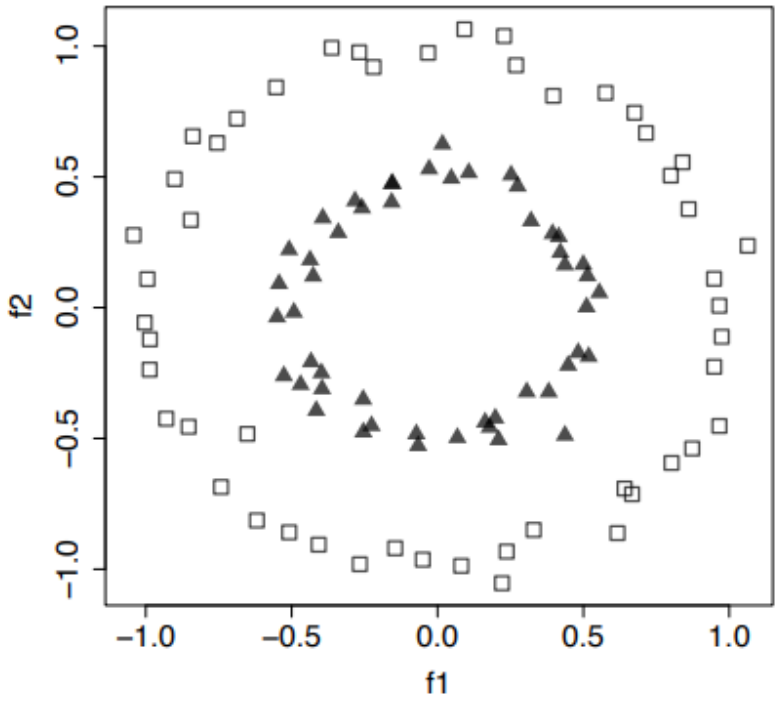
k-means cannot find the clusters that we might expect when ‘eyeballing’ the plots – this is due to the underlying assumptions made by *k*-means about how the points in a cluster should be distributed



(d) Circles Dataset

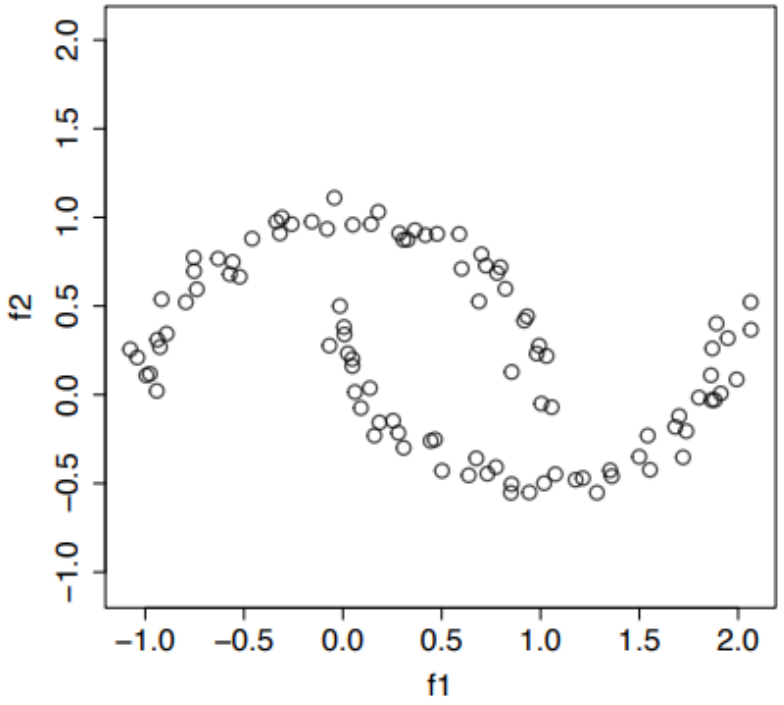


(e) *k*-means

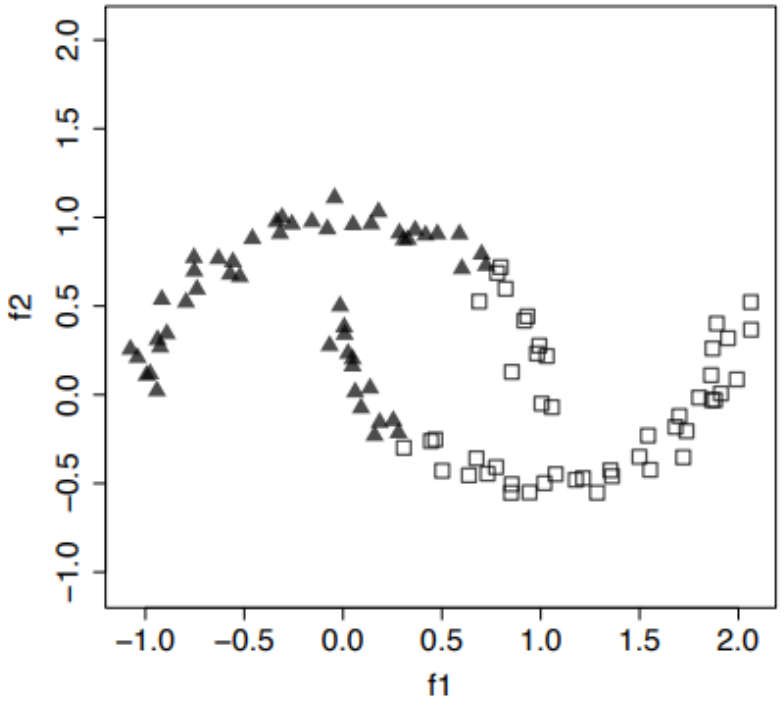


(f) AHC

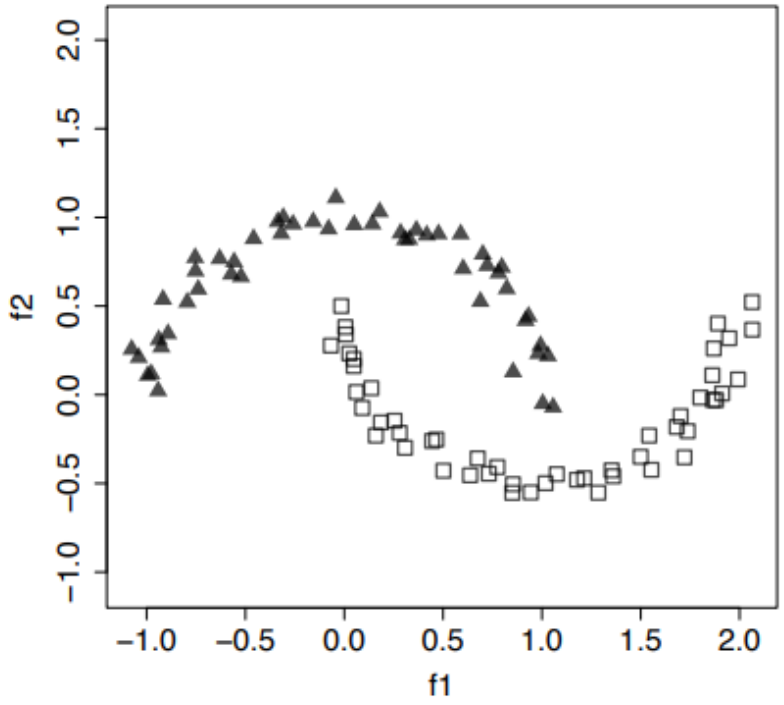
k-means vs. AHC on half-moons dataset



(g) Half-moons Dataset

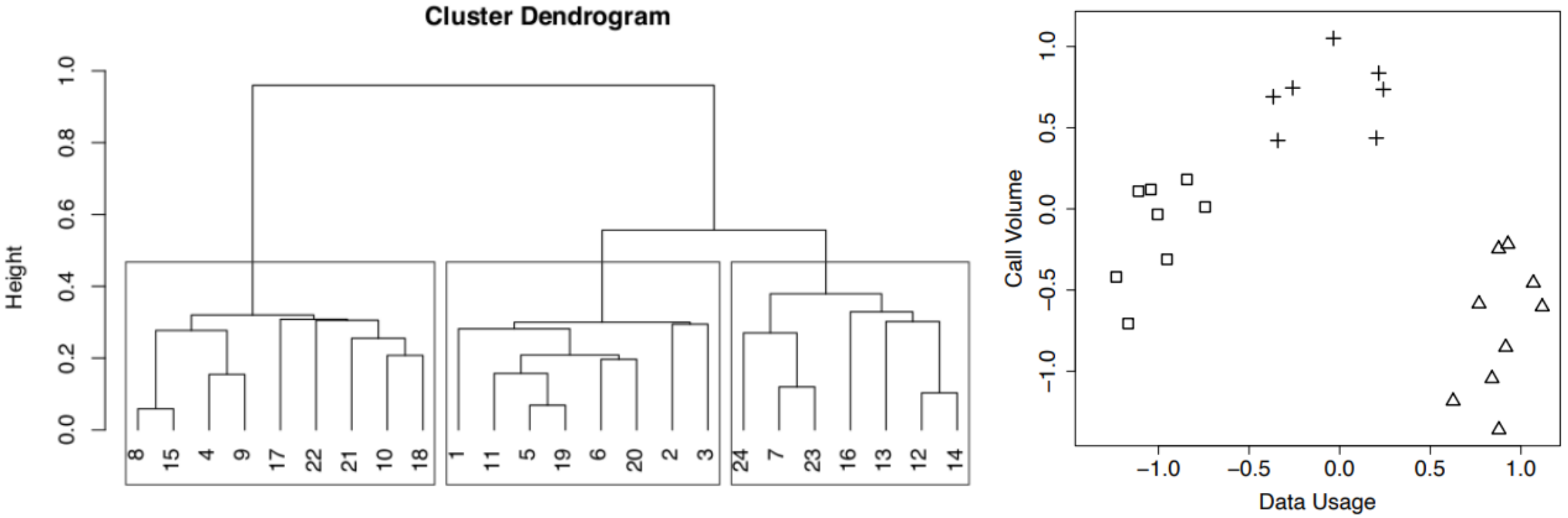


(h) *k*-means



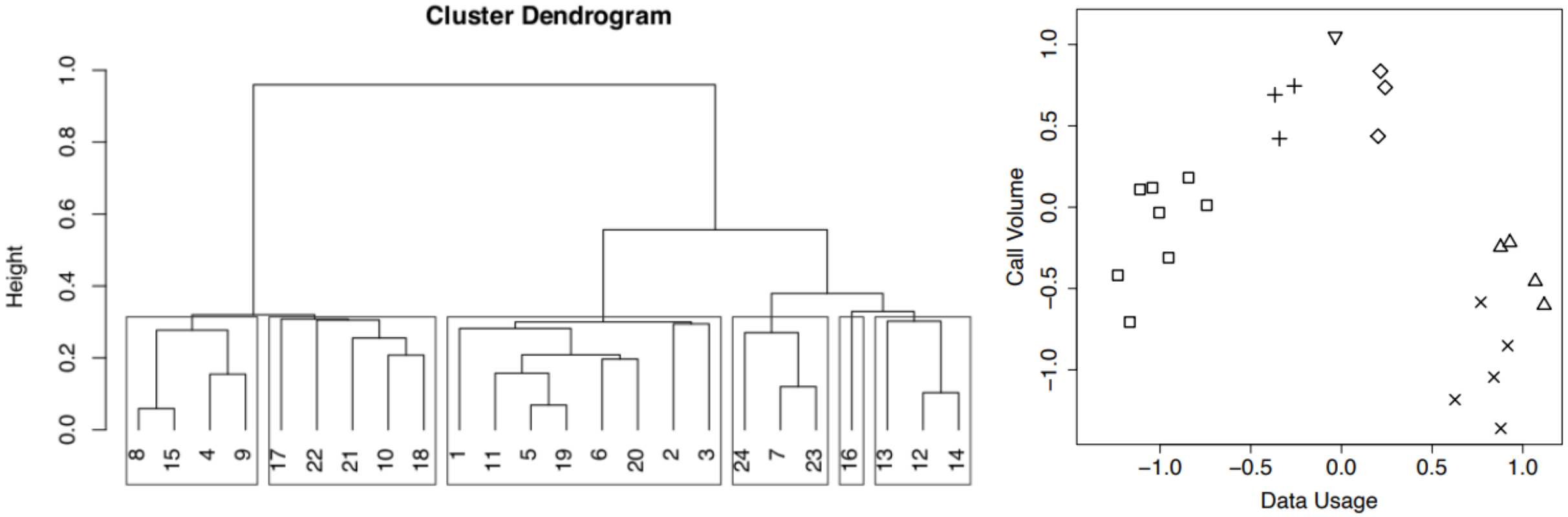
(i) AHC

Cutting an AHC cluster dendrogram, *k*=3



(b) Clustering cut at $k = 3$

Cutting an AHC cluster dendrogram, *k*=6



(c) Clustering cut at $k = 6$

Clustering algorithms in scikit-learn

Clustering of unlabelled data can be performed with the module `sklearn.cluster`

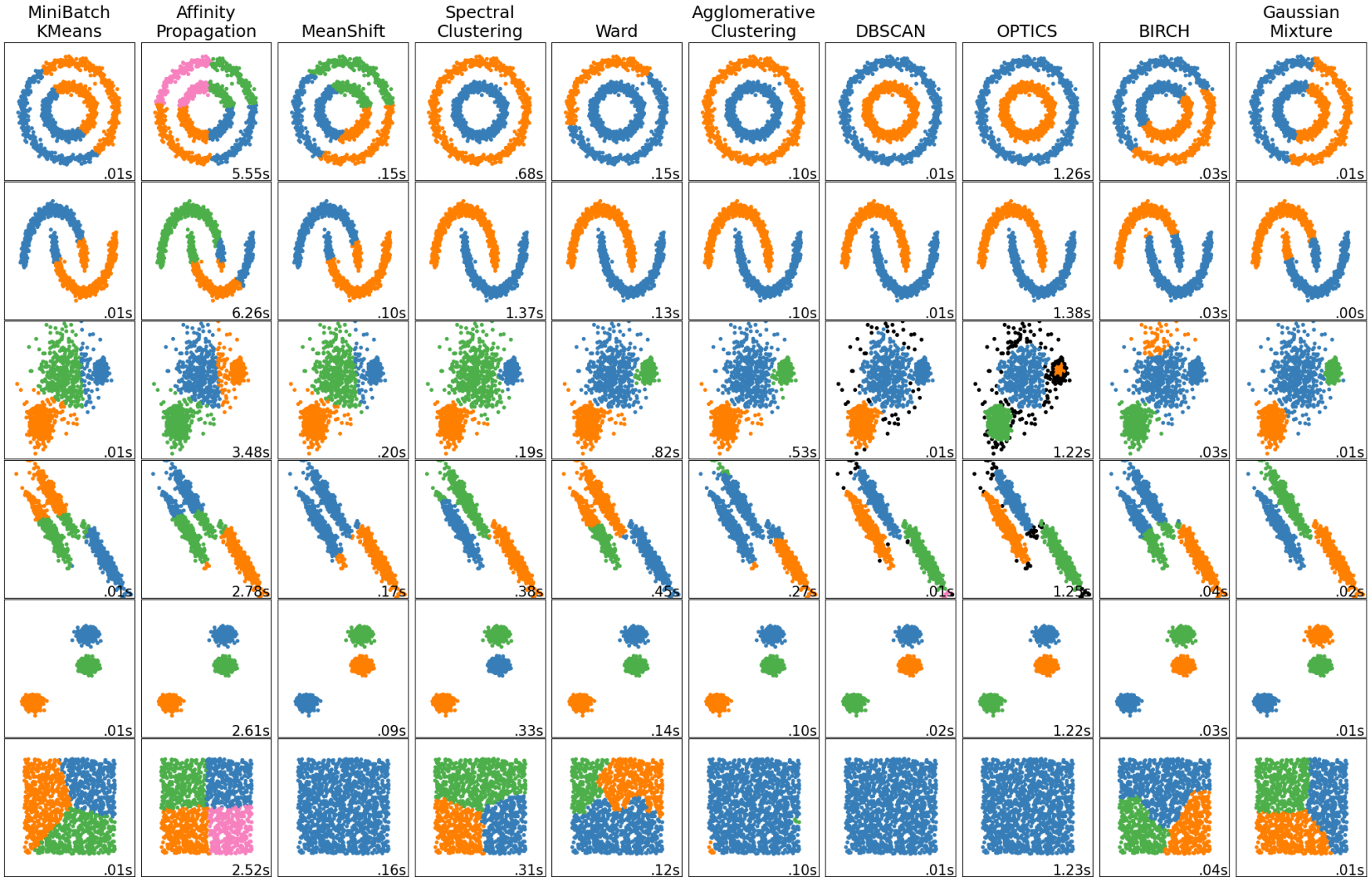
More info here:
https://scikit-learn.org/stable/modules/clustering.html

Class `sklearn.cluster.KMeans` implements the *k*-means algorithm
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Class `sklearn.cluster.AgglomerativeClustering` implements the agglomerative clustering algorithm
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

Scikit-learn - comparing clustering algorithms on toy datasets

https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html



Ollscoil na Gaillimhe
University of Galway