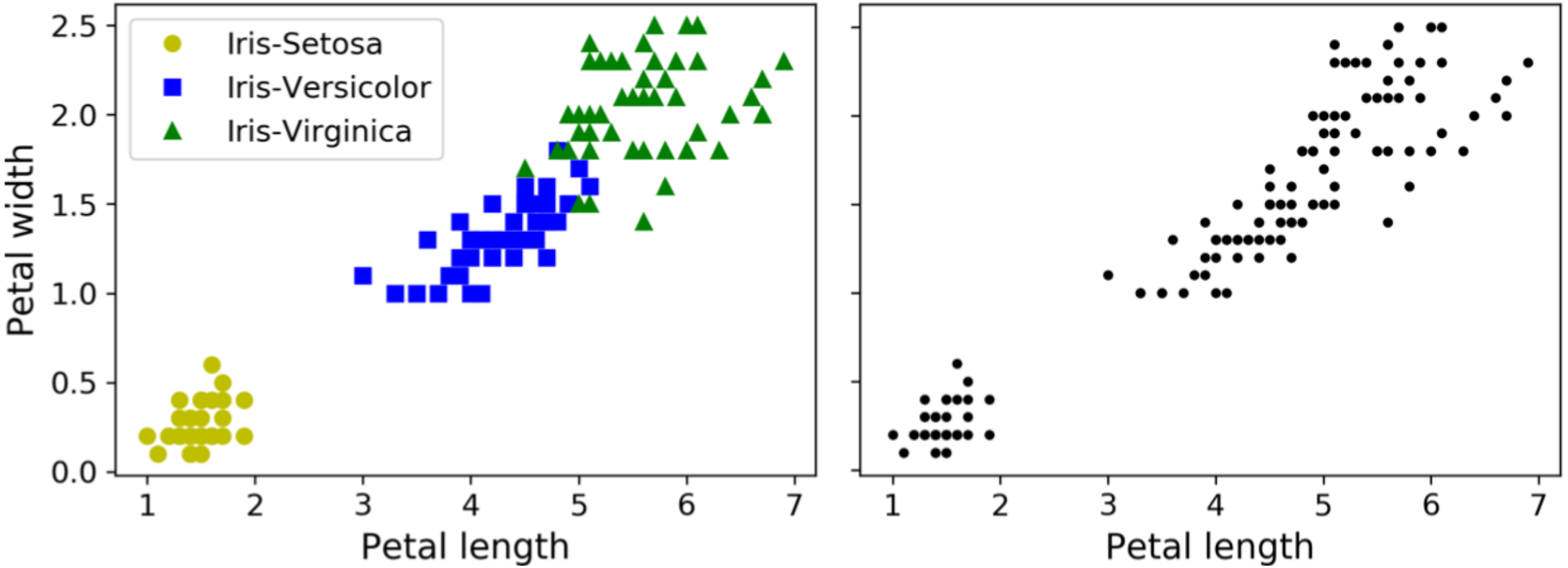
CSY2082 Introduction to Artificial Intelligence

Unsupervised Learning

Unsupervised Learning Techniques

- **Learning with no label**
- **Clustering**: the goal is to group similar instances together into clusters. This is a great tool for data analysis, customer segmentation, recommender systems, search engines, image segmentation, semi-supervised learning, dimensionality reduction, and more.
- **Anomaly detection**: the objective is to learn what “normal” data looks like, and use this to detect abnormal instances, such as defective items on a production line or a new trend in a time series.
- **Density estimation**: this is the task of estimating the probability density function (PDF) of the random process that generated the dataset. This is commonly used for anomaly detection: instances located in very low-density regions are likely to be anomalies. It is also useful for data analysis and visualization.

Classification versus Clustering



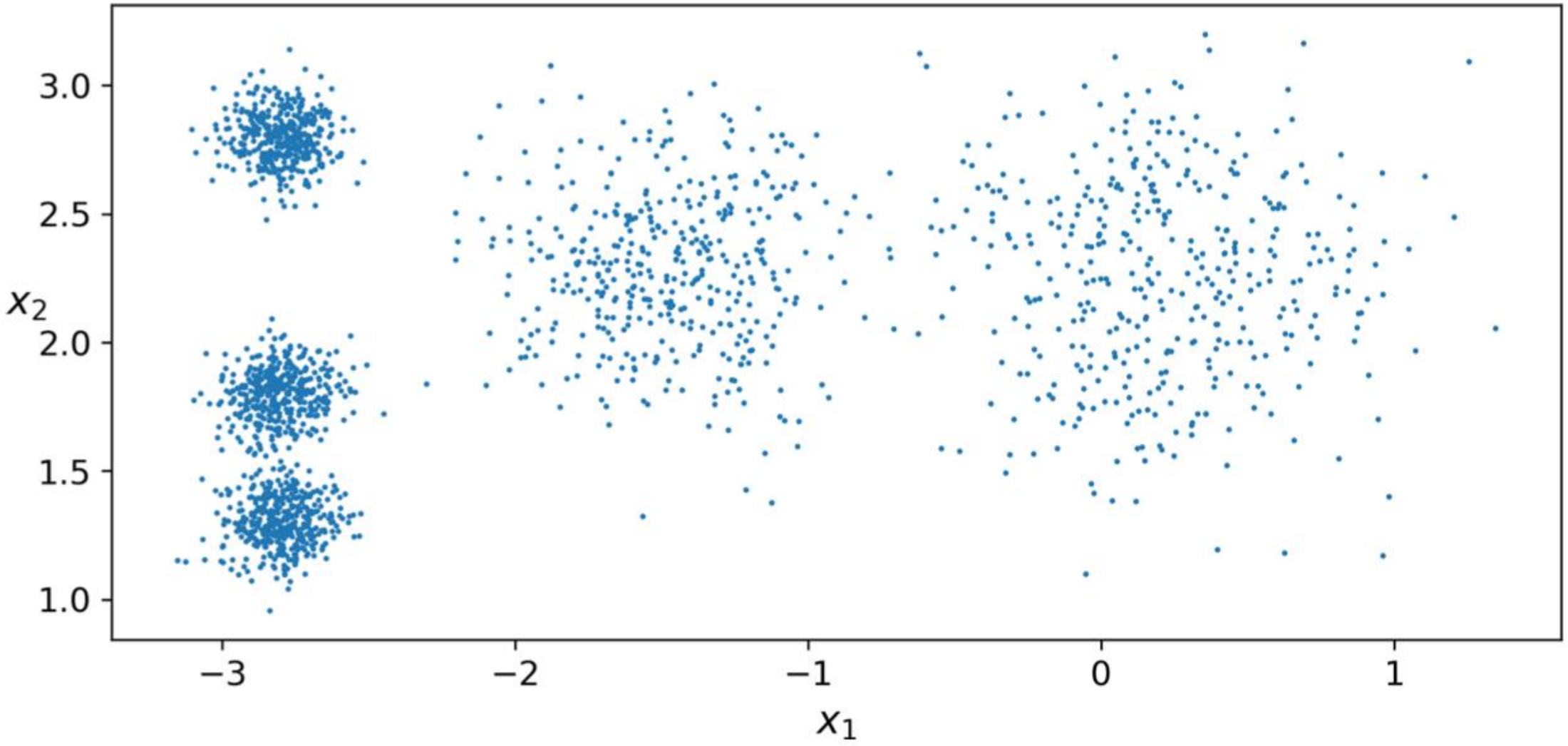
Clustering

Clustering

- **For customer segmentation:** you can cluster your customers based on their purchases, their activity on your website, and so on. This is useful to understand who your customers are and what they need, so you can adapt your products and marketing campaigns to each segment. For example, this can be useful in recommender systems to suggest content that other users in the same cluster enjoyed.
- **For data analysis:** when analysing a new dataset, it is often useful to first discover clusters of similar instances, as it is often easier to analyse clusters separately.
- **For anomaly detection (also called outlier detection):** any instance that has a low affinity to all the clusters is likely to be an anomaly. For example, if you have clustered the users of your website based on their behaviour, you can detect users with unusual behaviour, such as an unusual number of requests per second, and so on. Anomaly detection is particularly useful in detecting defects in manufacturing, or for fraud detection.
- **For semi-supervised learning:** if you only have a few labels, you could perform clustering and propagate the labels to all the instances in the same cluster. This can greatly increase the amount of labels available for a subsequent supervised learning algorithm, and thus improve its performance.
- **For search engines:** for example, some search engines let you search for images that are similar to a reference image. To build such a system, you would first apply a clustering algorithm to all the images in your database: similar images would end up in the same cluster. Then when a user provides a reference image, all you need to do is to find this image's cluster using the trained clustering model, and you can then simply return all the images from this cluster.
- **To segment an image:** by clustering pixels according to their colour, then replacing each pixel's colour with the mean colour of its cluster, it is possible to reduce the number of different colours in the image considerably. This technique is used in many object detection and tracking systems, as it makes it easier to detect the contour of each object.

Clustering
K-Means

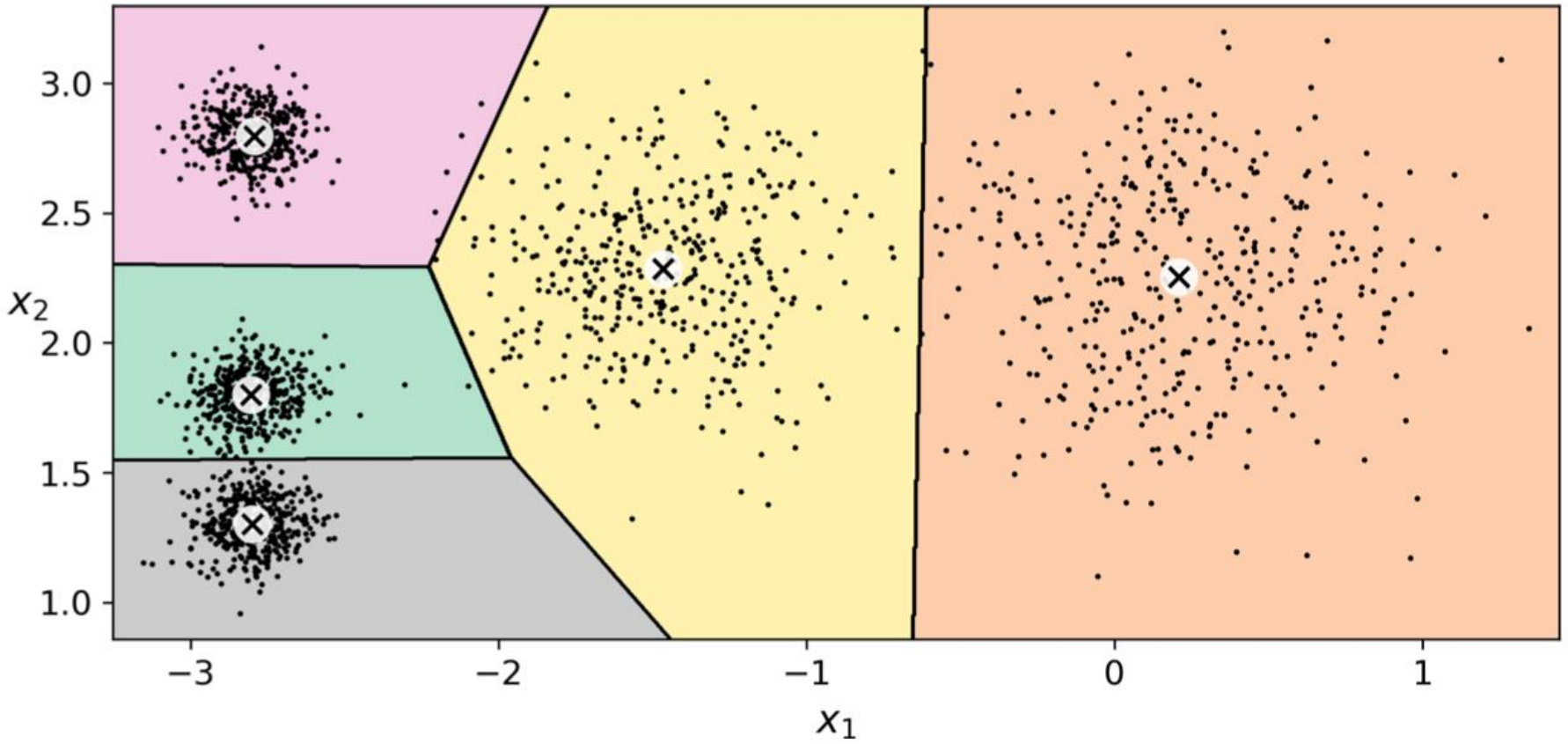
K-Means



K-Means

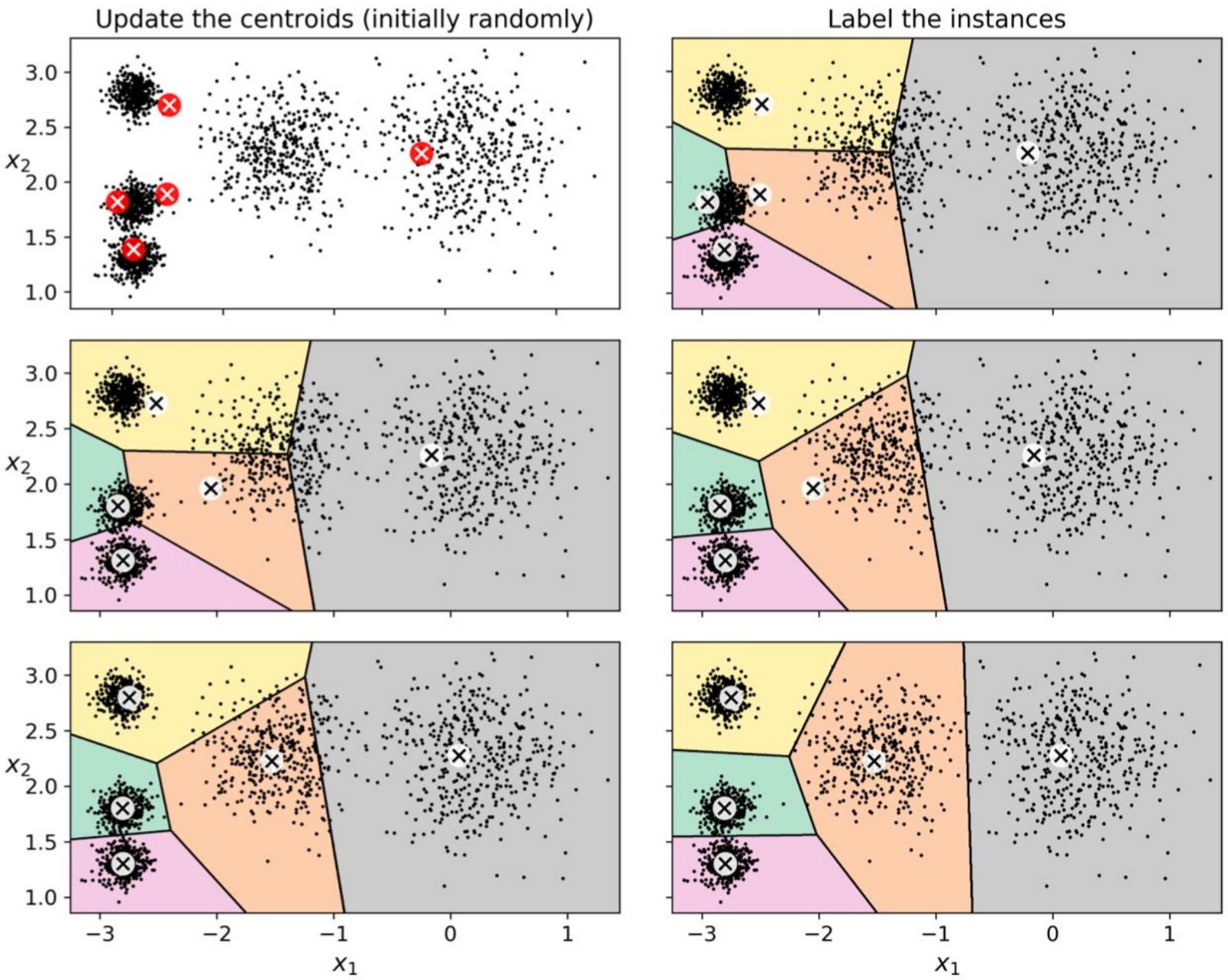
```
from sklearn.cluster import KMeans
k=5
kmeans = KMeans(n_clusters=k)
y_pred = kmeans.fit_predict(X)
```

- It will try to find each blob's centre and assign each instance to the closest blob.
- Note that you have to specify the number of clusters k that the algorithm must find.



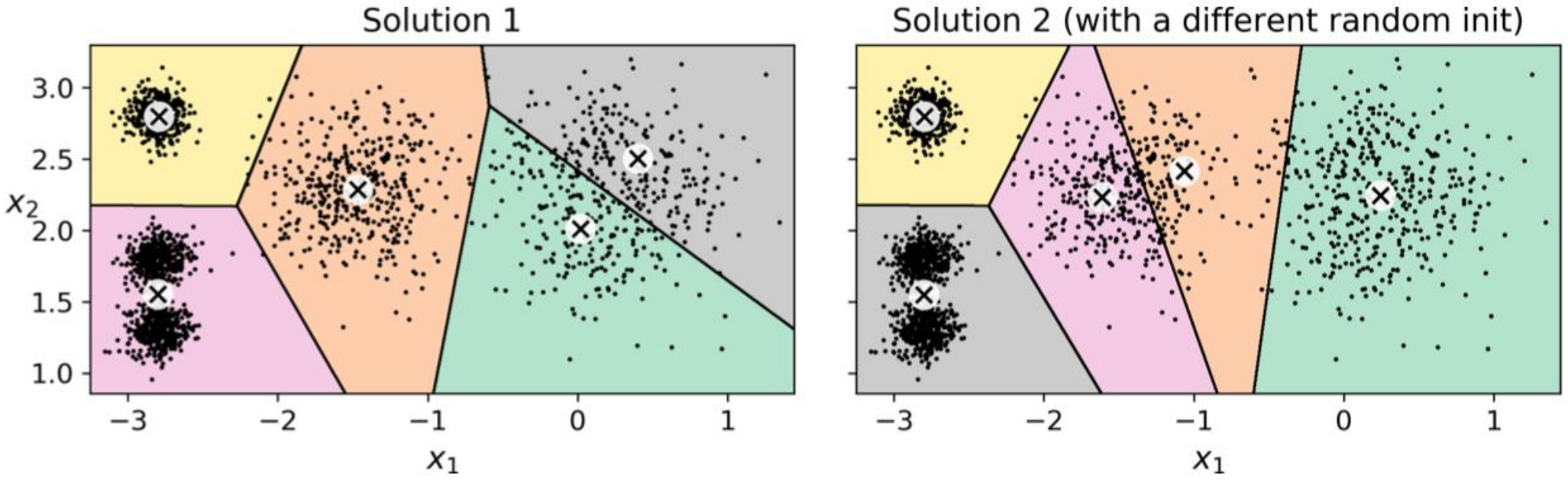
K-Means

- It starts by placing the centroids **randomly** (e.g., by picking k instances at random and using their locations as centroids).
- Then label the instances, update the centroids (calculate a new centroid based on all assigned instances), label the instances, update the centroids, and so on until the centroids **stop moving**.



K-Means

- Unfortunately, although the algorithm is guaranteed to converge, it may not converge to the right solution (i.e., it may converge to a local optimum): this depends on the centroid initialization.



Sub-optimal solutions due to unlucky centroid initializations

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Centroid Initialization Methods

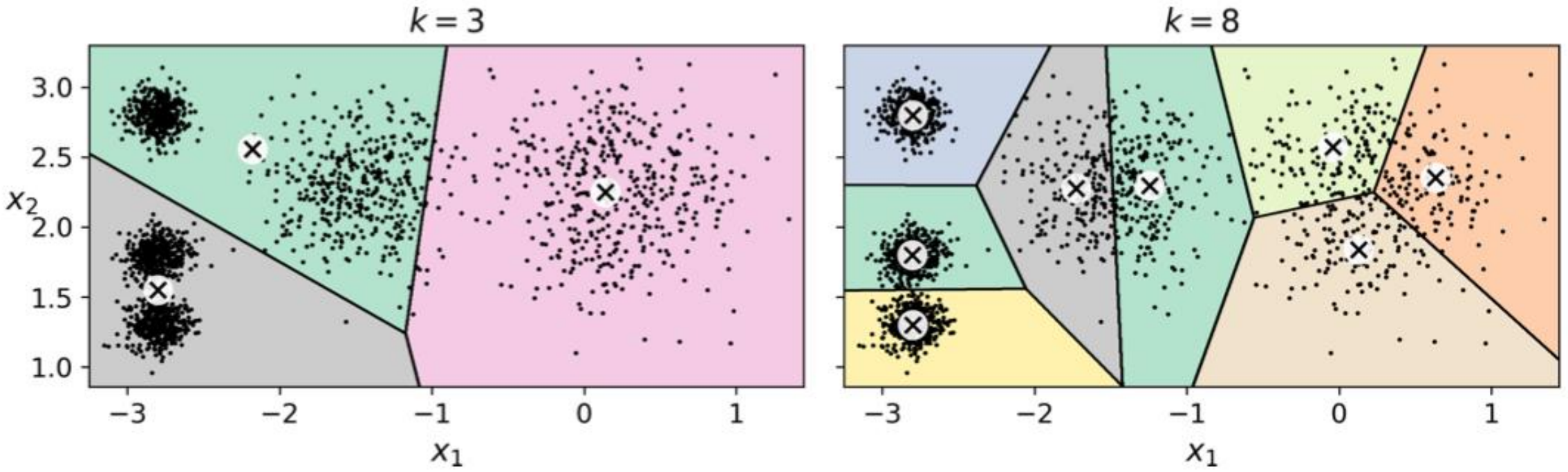
- If you happen to know approximately where the centroids should be, then you can set the init hyperparameter to a NumPy array containing the list of centroids, and set n_init to 1:

```
good_init = np.array([[-3, 3], [-3, 2], [-3, 1], [-1, 2], [0, 2]])
kmeans = KMeans(n_clusters=5, init=good_init, n_init=1)
```

- Another solution is to run the algorithm multiple times with different random initializations and keep the best solution. This is controlled by the n_init hyperparameter: by default, it is equal to 10.
  - it uses a performance metric called the model's inertia: this is the mean squared distance between each instance and its closest centroid.

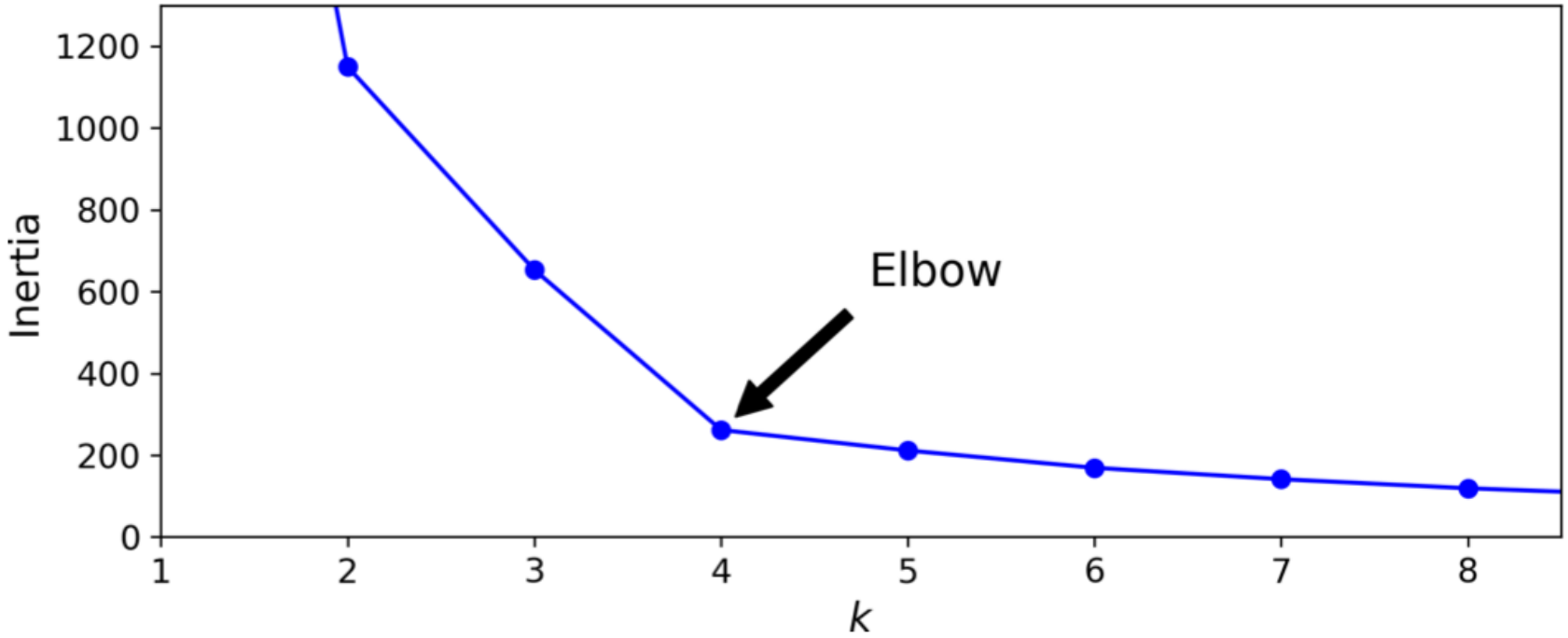
Finding the Optimal Number of Clusters

- So far, we have set the number of clusters k to 5 because it was obvious by looking at the data that this is the correct number of clusters. But in general, it will not be so easy to know how to set k, and the result might be quite bad if you set it to the wrong value.



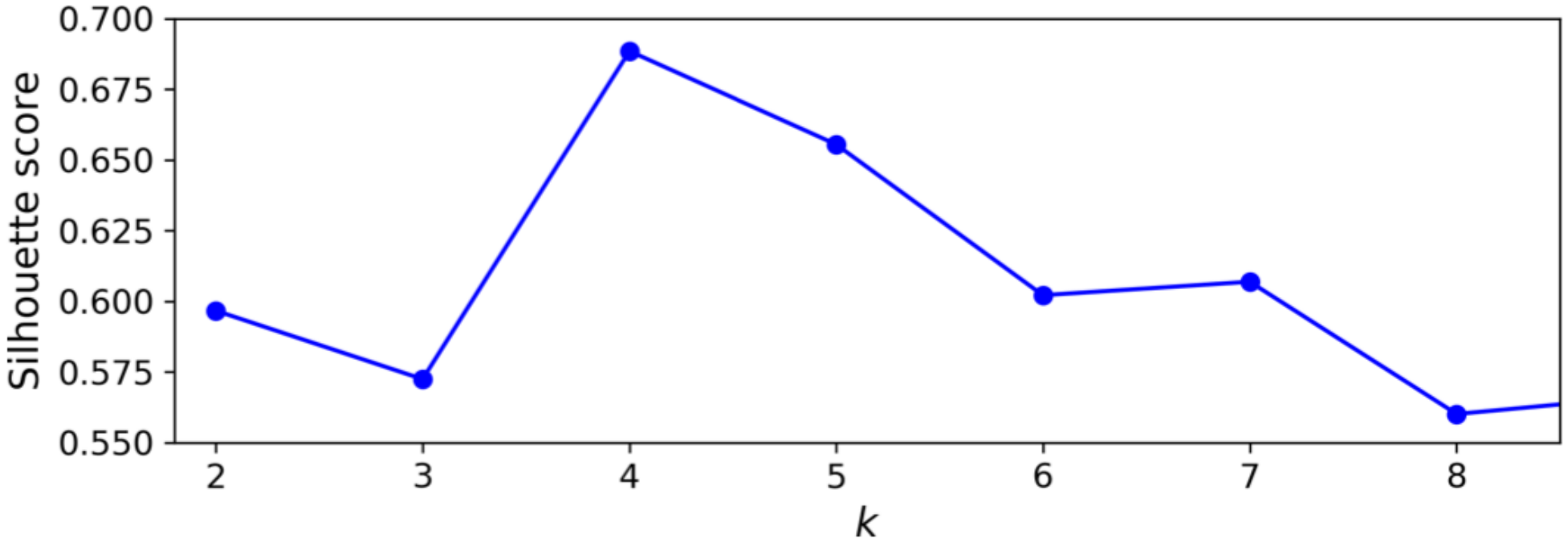
Finding the Optimal Number of Clusters

- You might be thinking that we could just pick the model with the lowest inertia. Unfortunately, the inertia is not a good performance metric when trying to choose k since it keeps getting lower as we increase k. Indeed, the more clusters there are, the closer each instance will be to its closest centroid, and therefore the lower the inertia will be.
- As you can see, the inertia drops very quickly as we increase k up to 4, but then it decreases much more slowly as we keep increasing k. This curve has roughly the shape of an arm, and there is an “elbow” at k=4 so if we did not know better, it would be a good choice: any lower value would be dramatic, while any higher value would not help much, and we might just be splitting perfectly good clusters in half for no good reason.



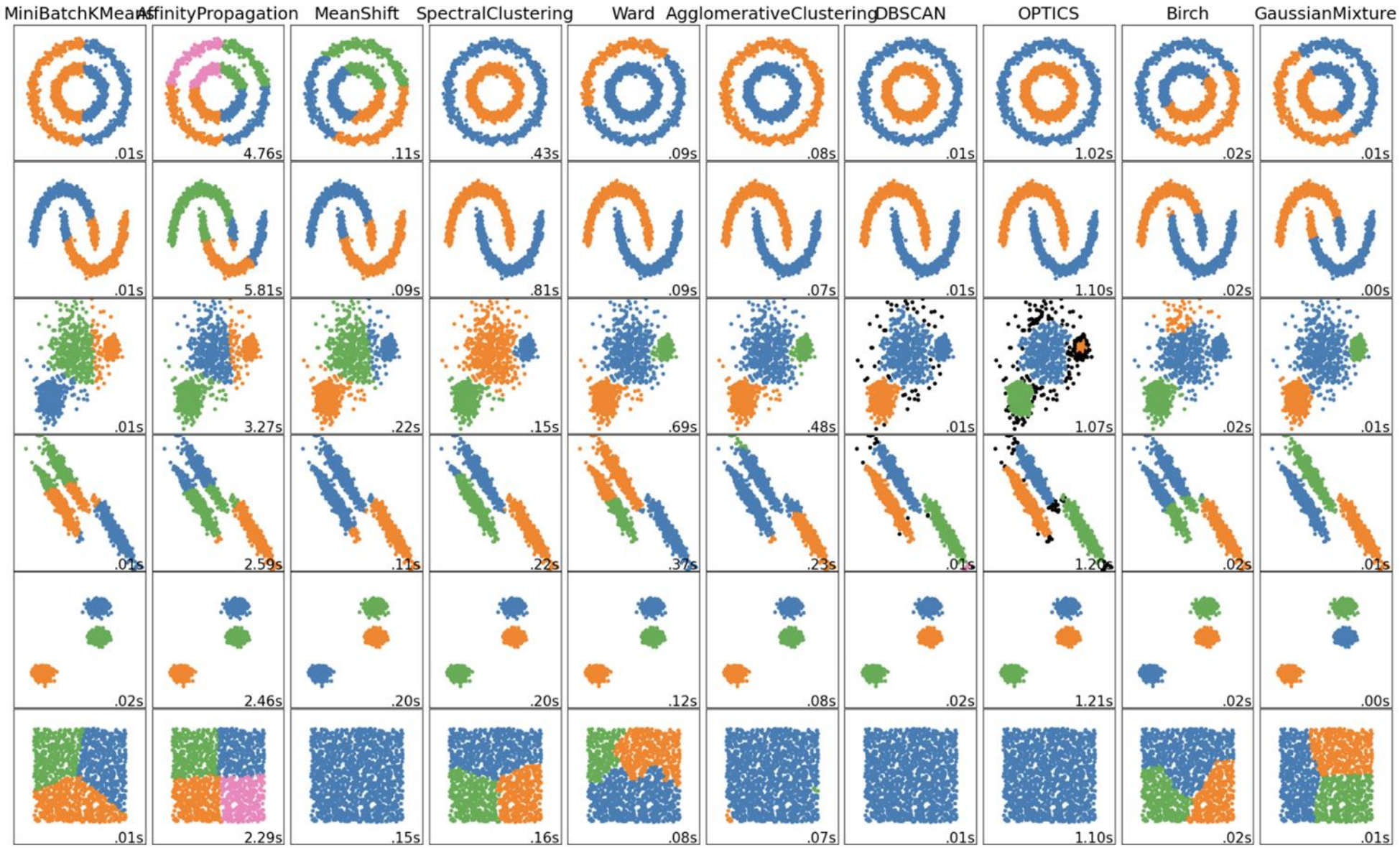
Finding the Optimal Number of Clusters

- A more precise approach (but also more computationally expensive) is to use the silhouette score, which is the mean silhouette coefficient over all the instances.
- An instance's silhouette coefficient is equal to (b – a) / max(a, b) where a is the mean distance to the other instances in the same cluster (it is the mean **intra-cluster** distance), and b is the mean nearest-cluster distance, that is the mean distance to the instances of the **next closest cluster** (defined as the one that minimizes b, excluding the instance's own cluster).
  - The silhouette coefficient can vary between -1 and +1: a coefficient close to +1 means that the instance is well inside its own cluster and far from other clusters, while a coefficient close to 0 means that it is close to a cluster boundary, and finally a coefficient close to -1 means that the instance may have been assigned to the wrong cluster.



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

Other clustering methods



A comparison of the clustering algorithms in scikit-learn

Exercise

- Use K-Means clustering to group houses by price. (Assignment 1 dataset)
- Visualise clustering results and select the best configurations.
- Choose another clustering method and compare results.

Use cases of clustering

Using clustering for image segmentation

- Image segmentation is the task of partitioning an image into multiple segments. In semantic segmentation, all pixels that are part of the same object type get assigned to the same segment.

Clustering faces

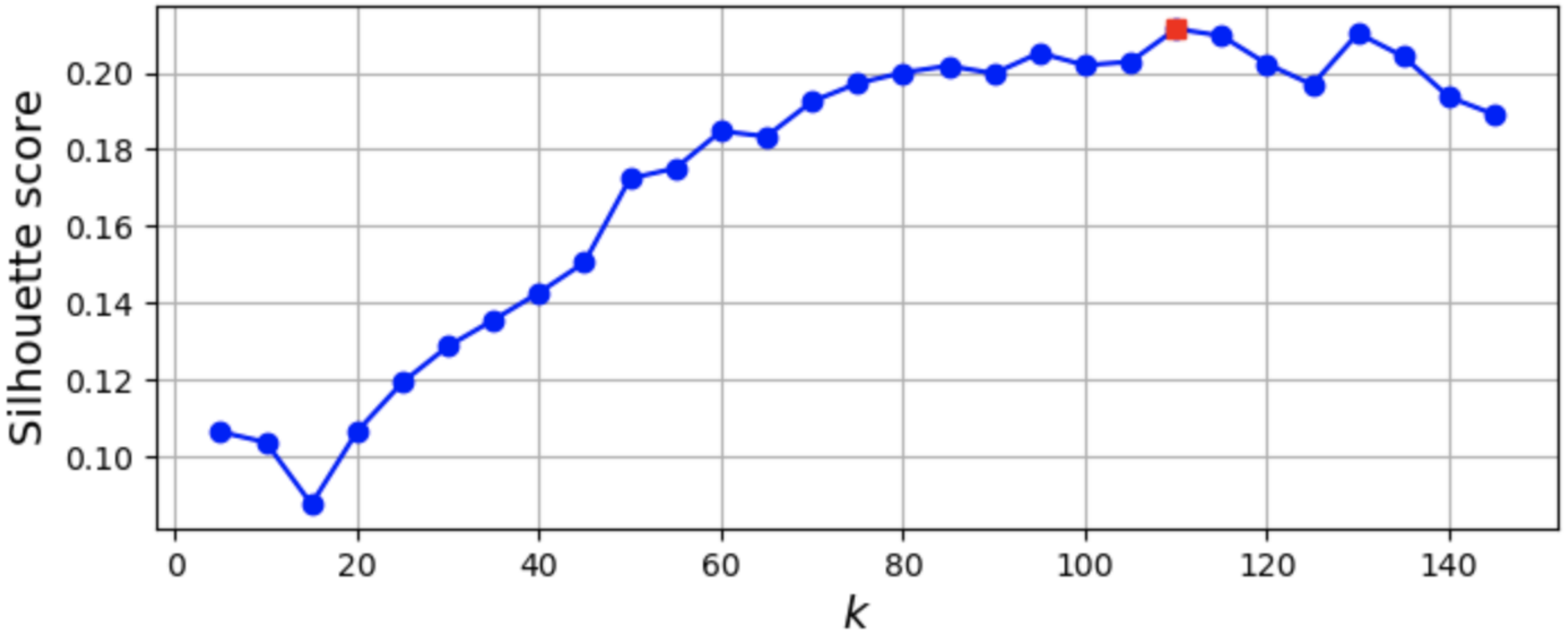
- The classic Olivetti faces dataset contains 400 grayscale 64 × 64–pixel images of faces. Each image is flattened to a 1D vector of size 4,096. The usual task is to train a model that can predict which person is represented in each picture.

Clustering faces

```
X=olivetti['data']
#Number of clusters
k=40
km_face = KMeans(n_clusters=k).fit(X)
```

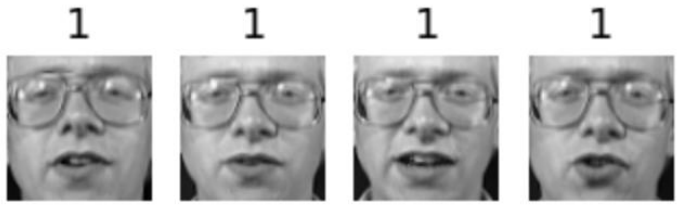
Clustering faces

- Best cluster size?



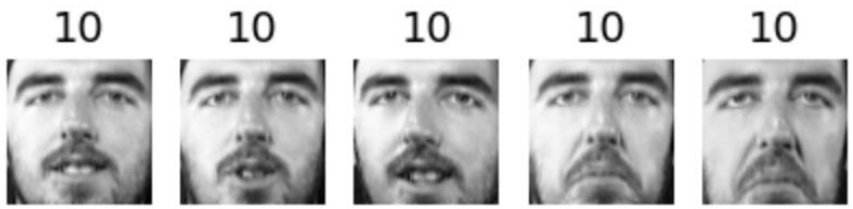
Cluster 32

Cluster 33



Cluster 34

Cluster 35



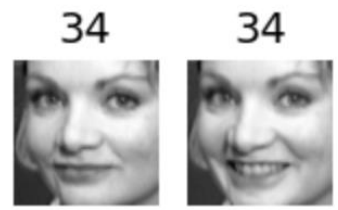
Cluster 36

Cluster 42

Cluster 43

Cluster 44

Cluster 45



Cluster 46