Co-funded by the
Erasmus+ Programme
of the European Union

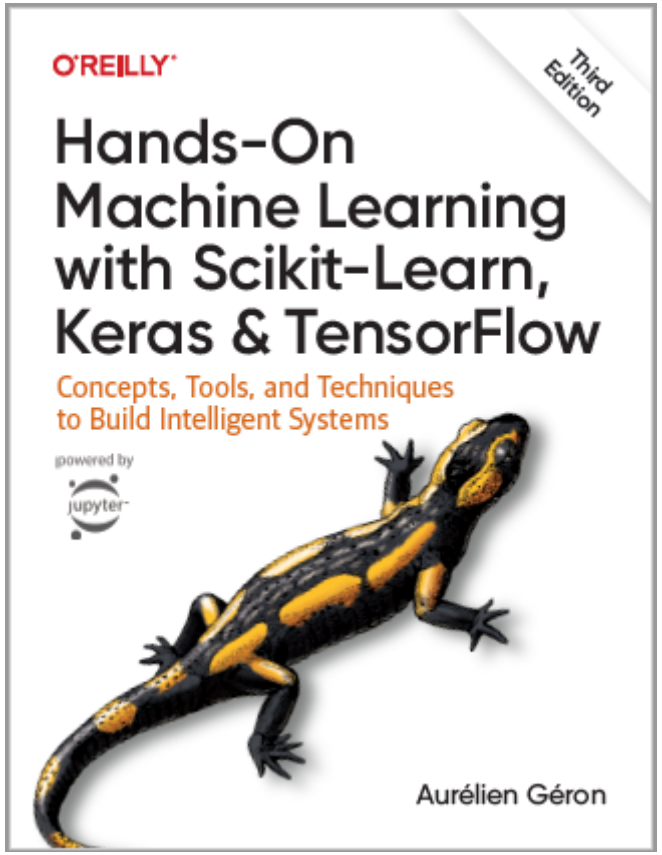
DeCAIR

Unsupervised Learning and Clustering

Prof. Gheith Abandah

Reference

- Chapter 8: **Dimensionality Reduction**
- Chapter 9: **Unsupervised Learning Techniques**



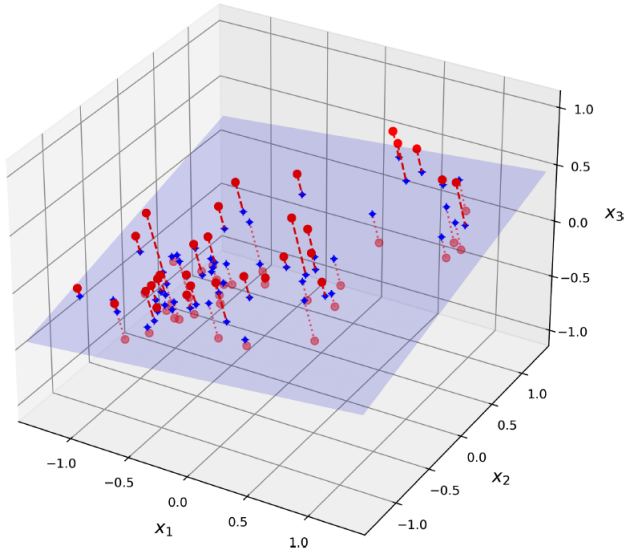
- Aurélien Géron, **Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow**, O'Reilly, 3rd Edition, 2022
  - Material: https://github.com/ageron/handson-ml3

Outline

- Dimensionality Reduction
- Unsupervised Learning
- Clustering
  - K-Means
- Exercises

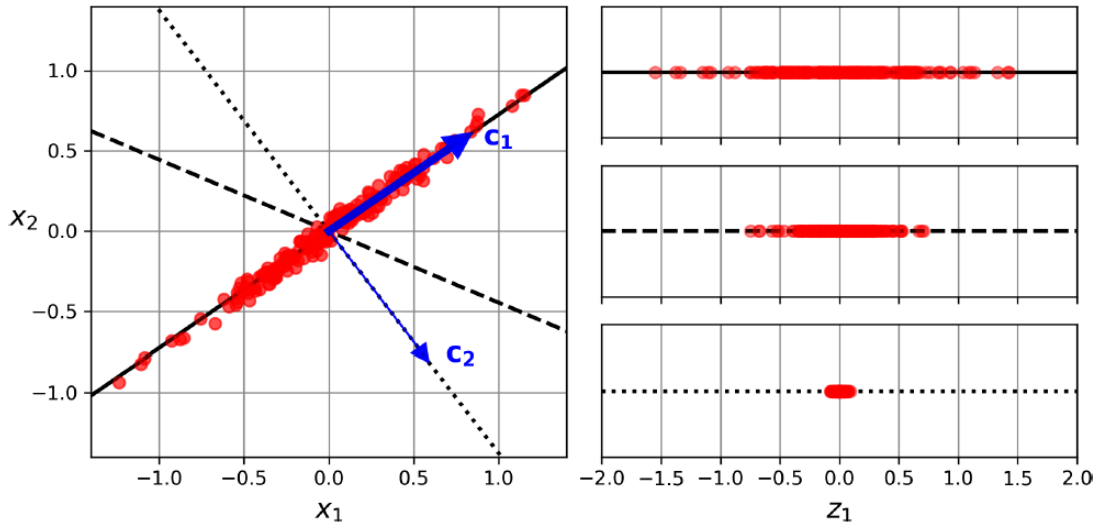
Dimensionality Reduction

- Many Machine Learning problems involve **thousands** or even **millions of features** for each training instance.
- All these features make training extremely **slow** and make it much **harder** to find a good solution.
- This problem is often referred to as the **curse of dimensionality**.
- **Dimensionality reduction approaches**
  - **Drop** not useful features
  - **Merge** correlated features
  - **Projection** and manifold
  - **Transform** features, *e.g.*, PCA



Principal Component Analysis (PCA)

- Is the most **popular** dimensionality reduction algorithm.
- First it identifies the **hyperplane** that lies **closest to** the **data**, and then it **projects** the data onto it.
- PCA identifies the **axis** that accounts for the **largest amount of variance** in the training set. Then it finds the **next orthogonal axes** that accounts for the largest amount of remaining variance.



Principal Component Analysis (PCA)

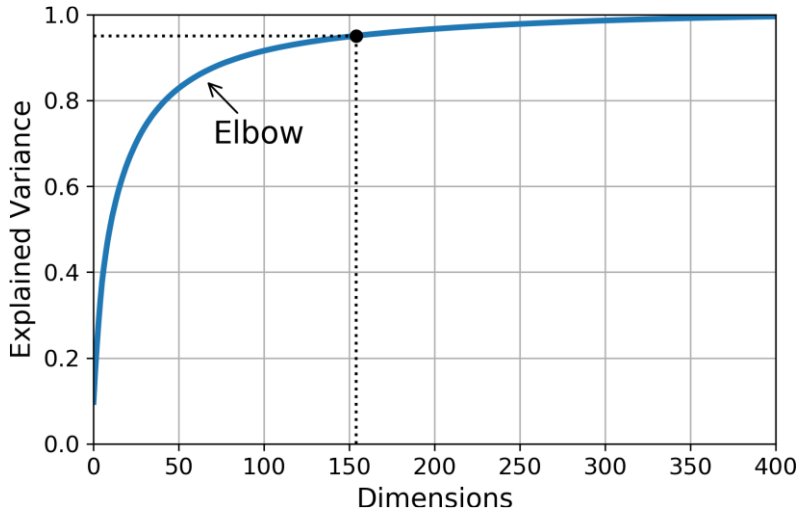
- Use PCA to reduce the dimensionality of the dataset down to **two dimensions**.
- Instead of specifying the number of principal components you want to preserve, you can set `n_components` to be a float between `0.0` and `1.0`, indicating the ratio of variance you wish to preserve.

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

3-D

```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```

MNIST



Outline

- Dimensionality Reduction
- Unsupervised Learning
- Clustering
  - K-Means
- Exercises

Unsupervised Learning

If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake.

Yann LeCun

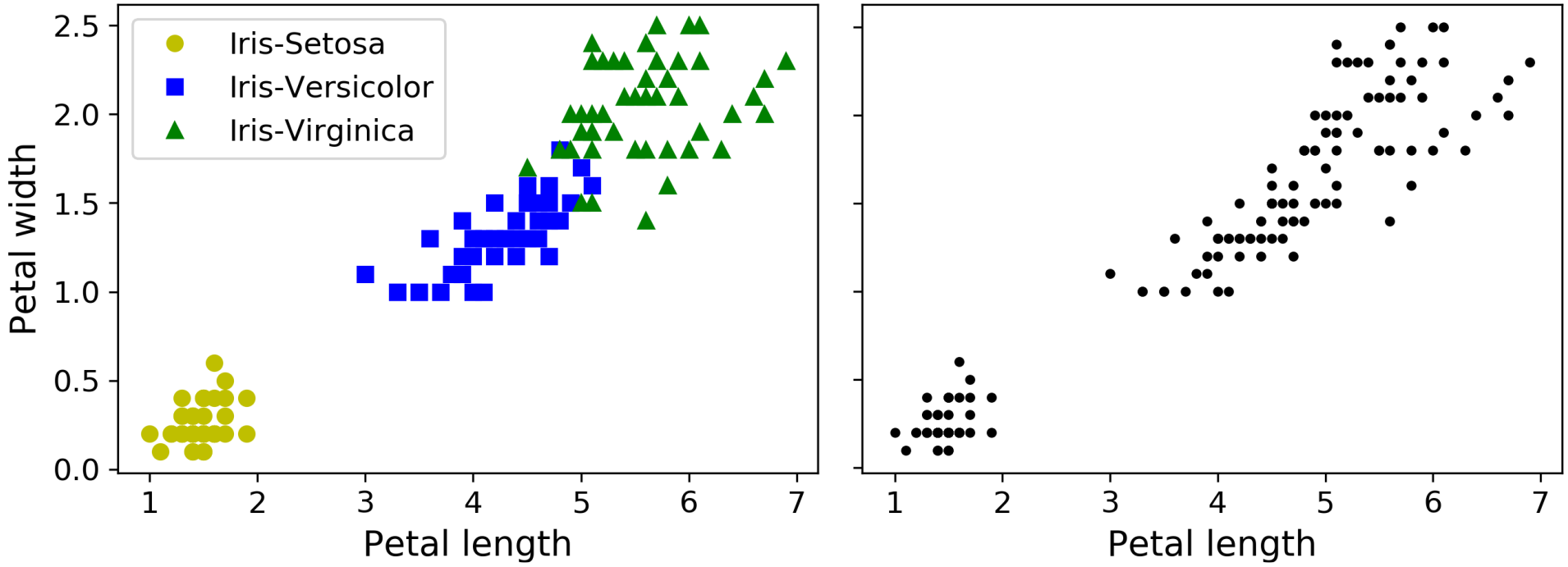
- **Example**: System that takes a few pictures of each item on a manufacturing production line and detects which items are defective.

Outline

- Dimensionality Reduction
- Unsupervised Learning
- Clustering
  - K-Means
- Exercises

Clustering

- The task of **identifying similar instances** and assigning them to clusters, i.e., groups of similar instances.
- **Classification** (left) versus **clustering** (right)

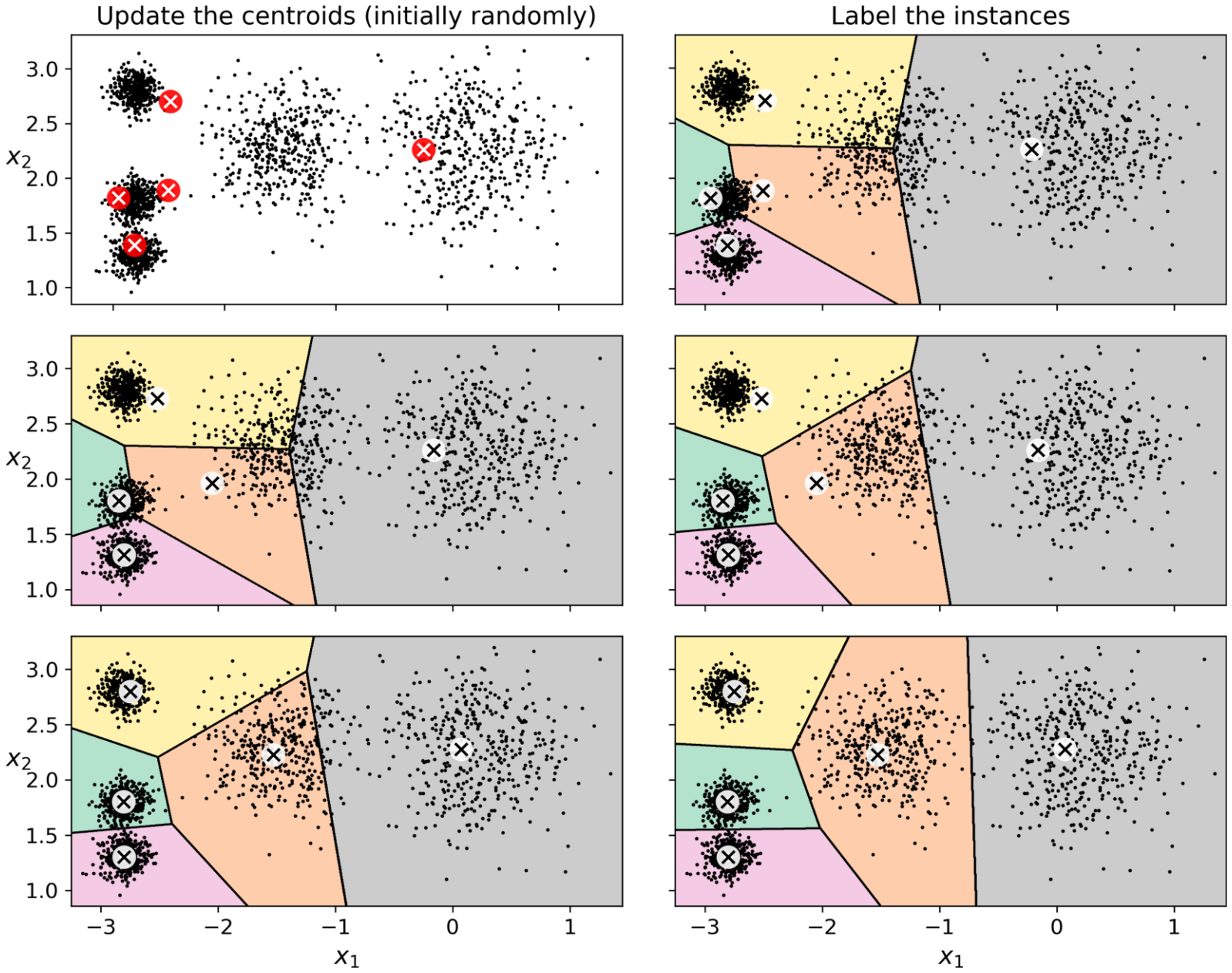


Clustering Applications

- **Customer segmentation**: useful for recommender systems.
- **Data analysis**: discover clusters of similar instances as it is often easier to analyze clusters separately.
- **Dimensionality reduction**: find affinity features to the found clusters
- **Anomaly detection**: any instance that has a low affinity to all the clusters is likely to be an anomaly.
- **Semi-supervised learning**: perform clustering and propagate the labels to all the instances in the same cluster.
- **Search engines** for images
- **Image segmentation**

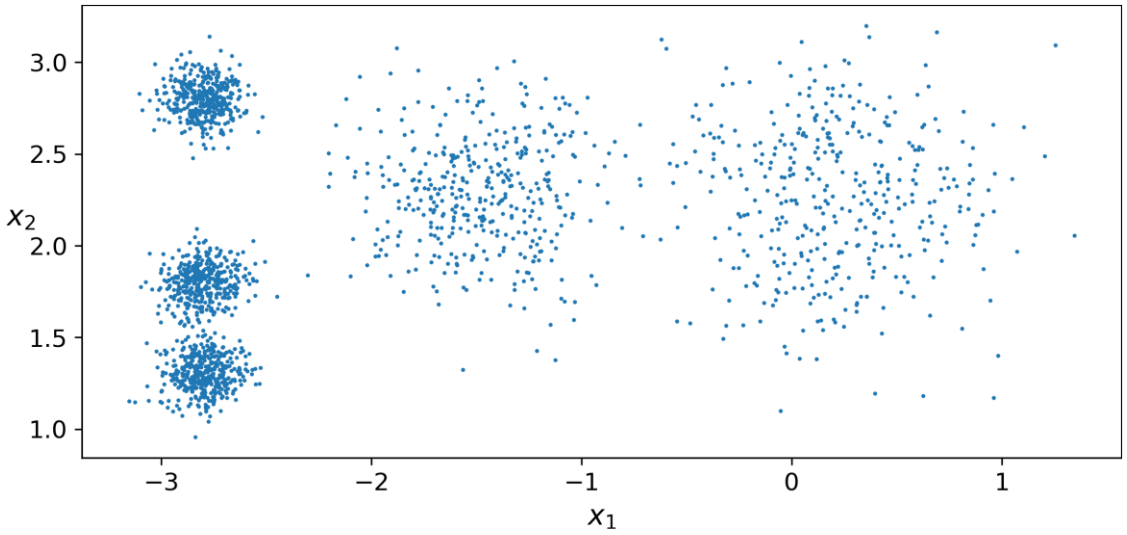
K-Means

- **Quick** and **efficient** algorithm
- **Scale** before clustering
- Need to **specify** the **number of clusters**



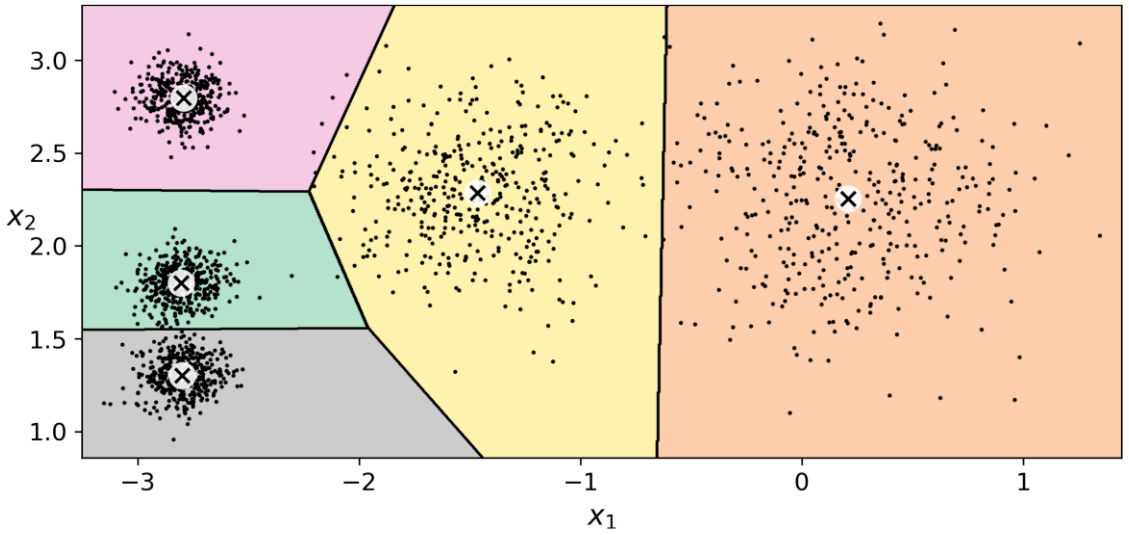
K-Means

- Cluster to 5 clusters




```
from sklearn.cluster import KMeans
k = 5
kmeans = KMeans(n_clusters=k)
y_pred = kmeans.fit_predict(X)
y_pred
array([4, 0, 1, ..., 2, 1, 0],
                        dtype=int32)
# Hard clustering:
X_new = np.array([[0, 2], [-3, 3]])
kmeans.predict(X_new)
array([1, 2], dtype=int32)
```

K-Means



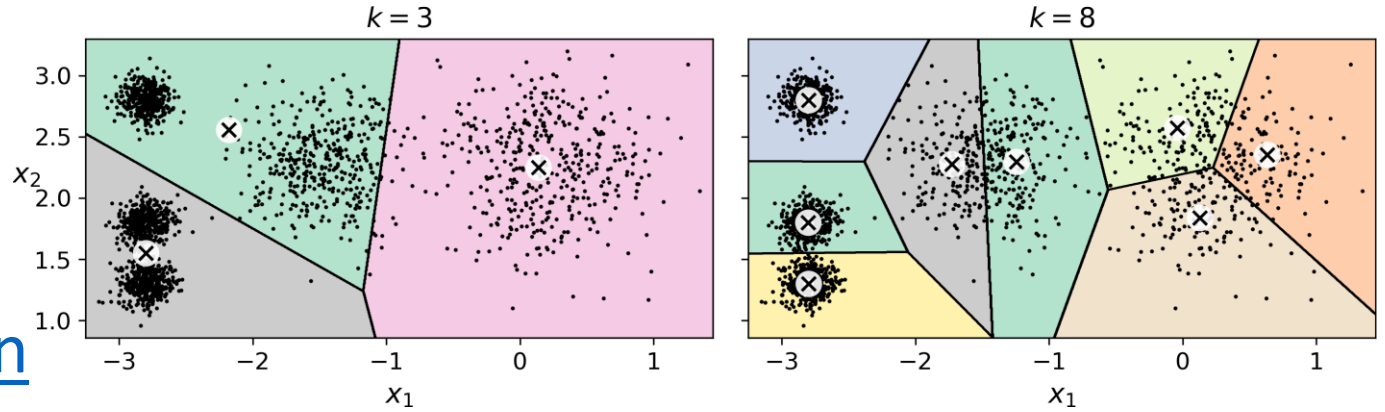

```
kmeans.cluster_centers_
array([[-2.80389616, 1.80117999],
       [ 0.20876306, 2.25551336],
       [-2.79290307, 2.79641063],
       [-1.46679593, 2.28585348],
       [-2.80037642, 1.30082566]])
```

Can be a dimensionality reduction technique.

```
# Soft clustering, a score per
#  cluster:
kmeans.transform(X_new)
array([[2.81093633, 0.32995317,
        2.9042344 , 1.49439034,
        2.88633901],
       [1.21475352, 3.29399768,
        0.29040966, 1.69136631,
        1.71086031])
```

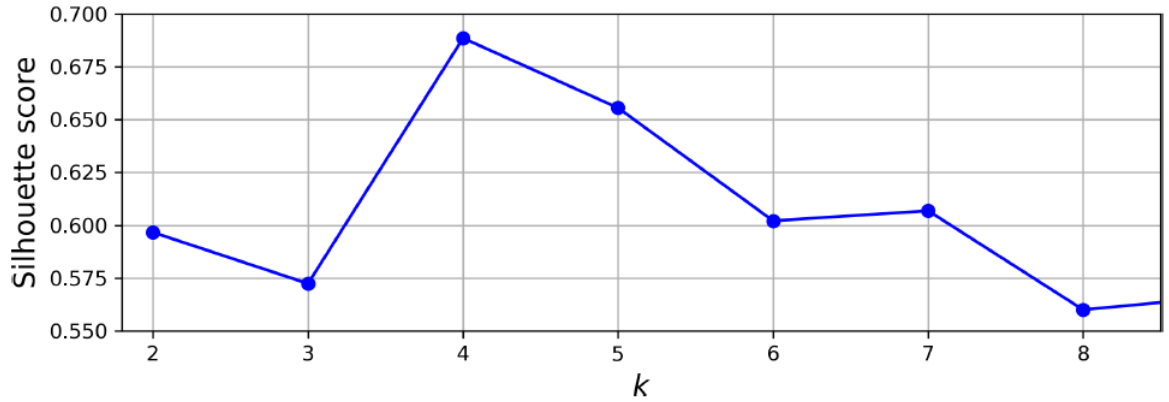
K-Means

- It is important to specify the **right** number of clusters *k*.



- Find *k* that gives highest mean silhouette coefficient.

```
from sklearn.metrics import
                        silhouette_score
silhouette_score(X, kmeans.labels_)
0.655517642572828
```



Summary

- Dimensionality Reduction
- Unsupervised Learning
- Clustering
  - K-Means
- Exercises

Exercises

8.9. Load the **MNIST dataset** (introduced in Chapter 3) and split it into a training set and a test set (take the first 60,000 instances for training, and the remaining 10,000 for testing). Train a Random Forest classifier on the dataset and time how long it takes, then evaluate the resulting model on the test set. Next, use **PCA** to reduce the dataset's dimensionality, with an explained variance ratio of **95%**. Train a new Random Forest classifier on the reduced dataset and see how long it takes. Was training much faster? Next evaluate the classifier on the test set: how does it compare to the previous classifier?

Exercises

9.10. 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. 40 different people were photographed (10 times each), and the usual task is to train a model that can predict which person is represented in each picture. Load the dataset using the `sklearn.datasets.fetch_olivetti_faces()` function, then split it into a training set, a validation set, and a test set (note that the dataset is already scaled between 0 and 1). Since the dataset is quite small, you probably want to use stratified sampling to ensure that there are the same number of images per person in each set. Next, **cluster the images** using **KMeans**, and ensure that you have a good number of clusters (using one of the techniques discussed in this chapter). Visualize the clusters: do you see similar faces in each cluster?