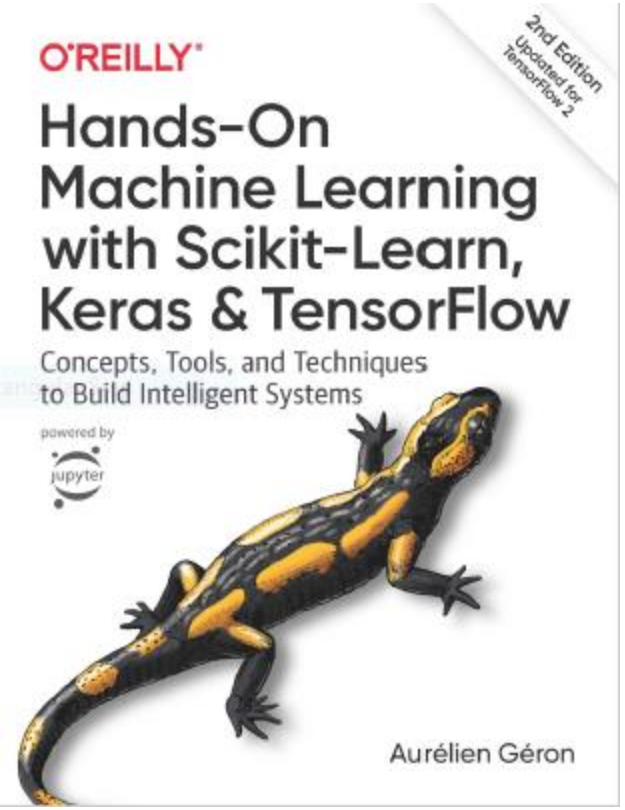
Unsupervised Learning and Clustering

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, 2nd Edition, 2019
  - Material: https://github.com/ageron/handson-ml2

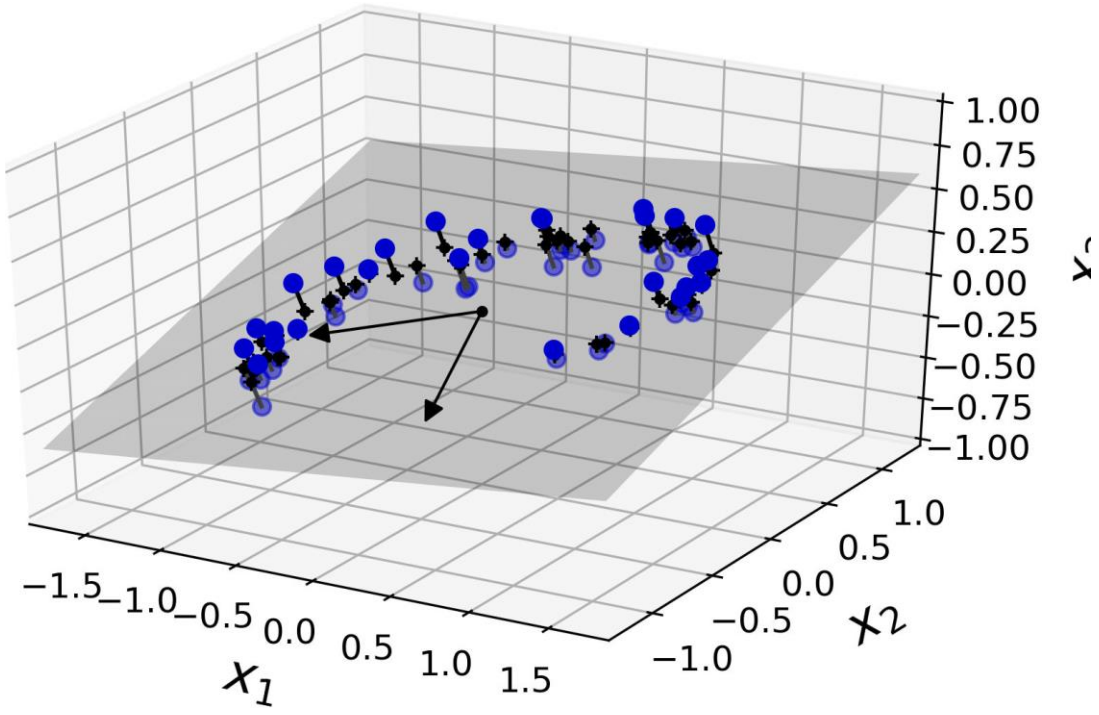
Outline

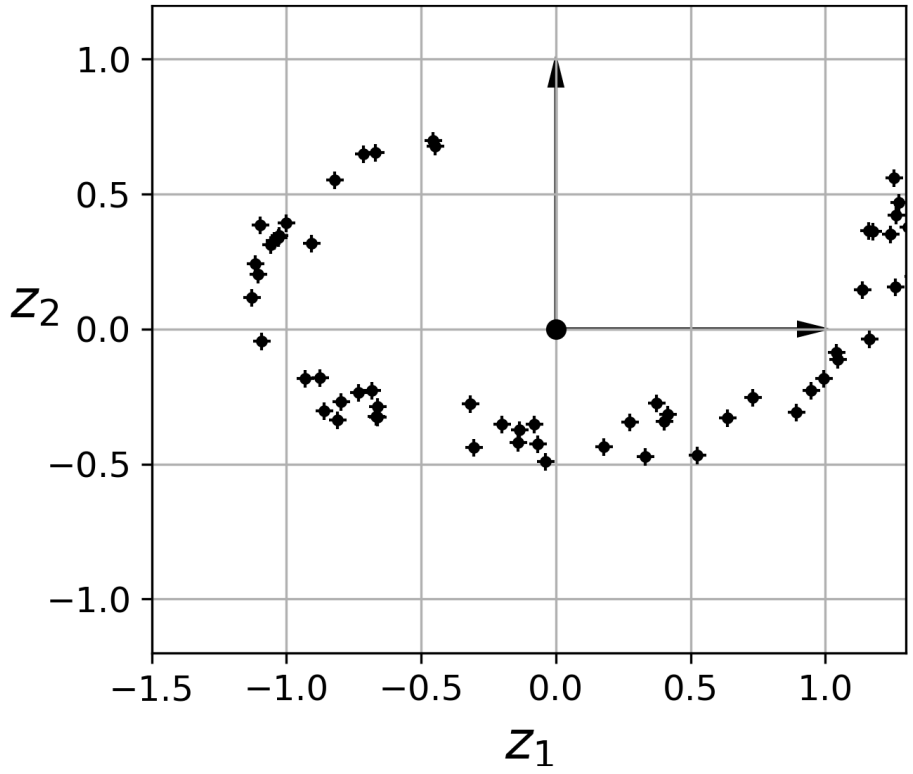
- Dimensionality Reduction
  - Projection and Manifold
  - Principal Component Analysis (PCA)
- Unsupervised Learning
- Clustering
  - K-Means
  - DBSCAN
- Gaussian Mixtures and Anomaly Detection
- Exercises

Dimensionality Reduction

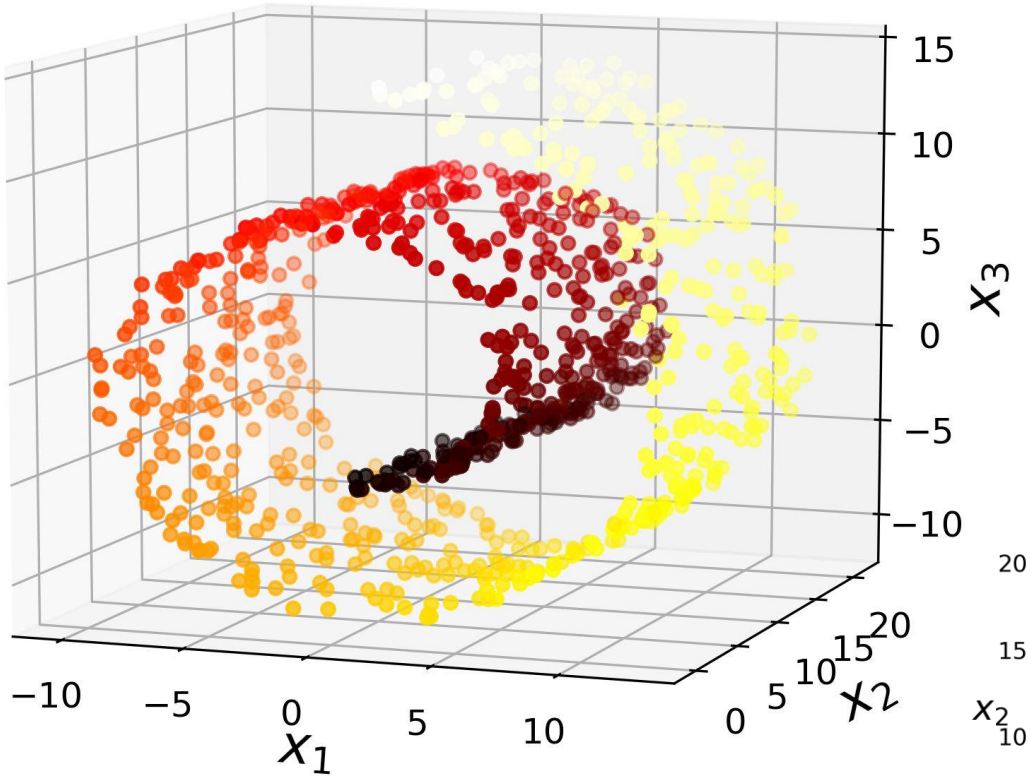
- Many Machine Learning problems involve **thousands** or even **millions of features** for each training instance.
- All these features make training extremely **slow** and make it much **harder** to find a good solution.
- This problem is often referred to as the **curse of dimensionality**.
- **Dimensionality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

Projection and Manifold

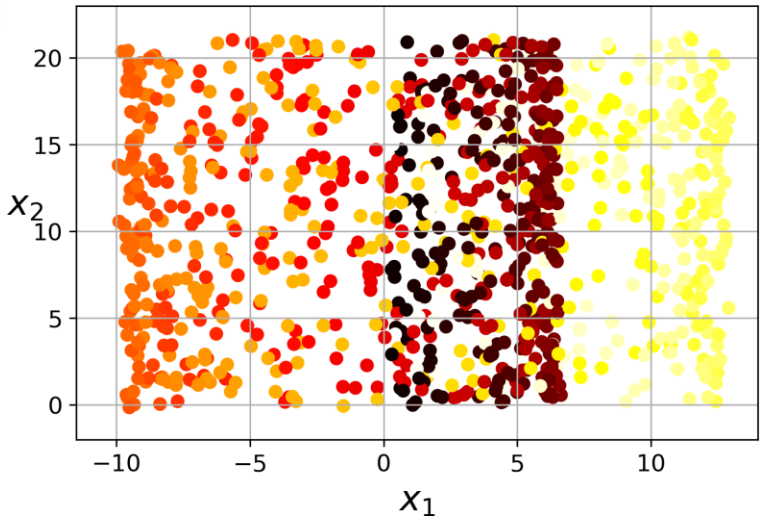


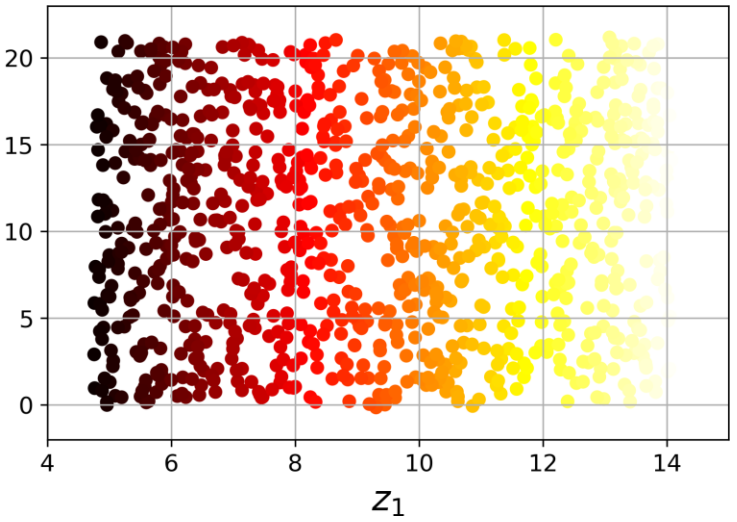


Projection and Manifold



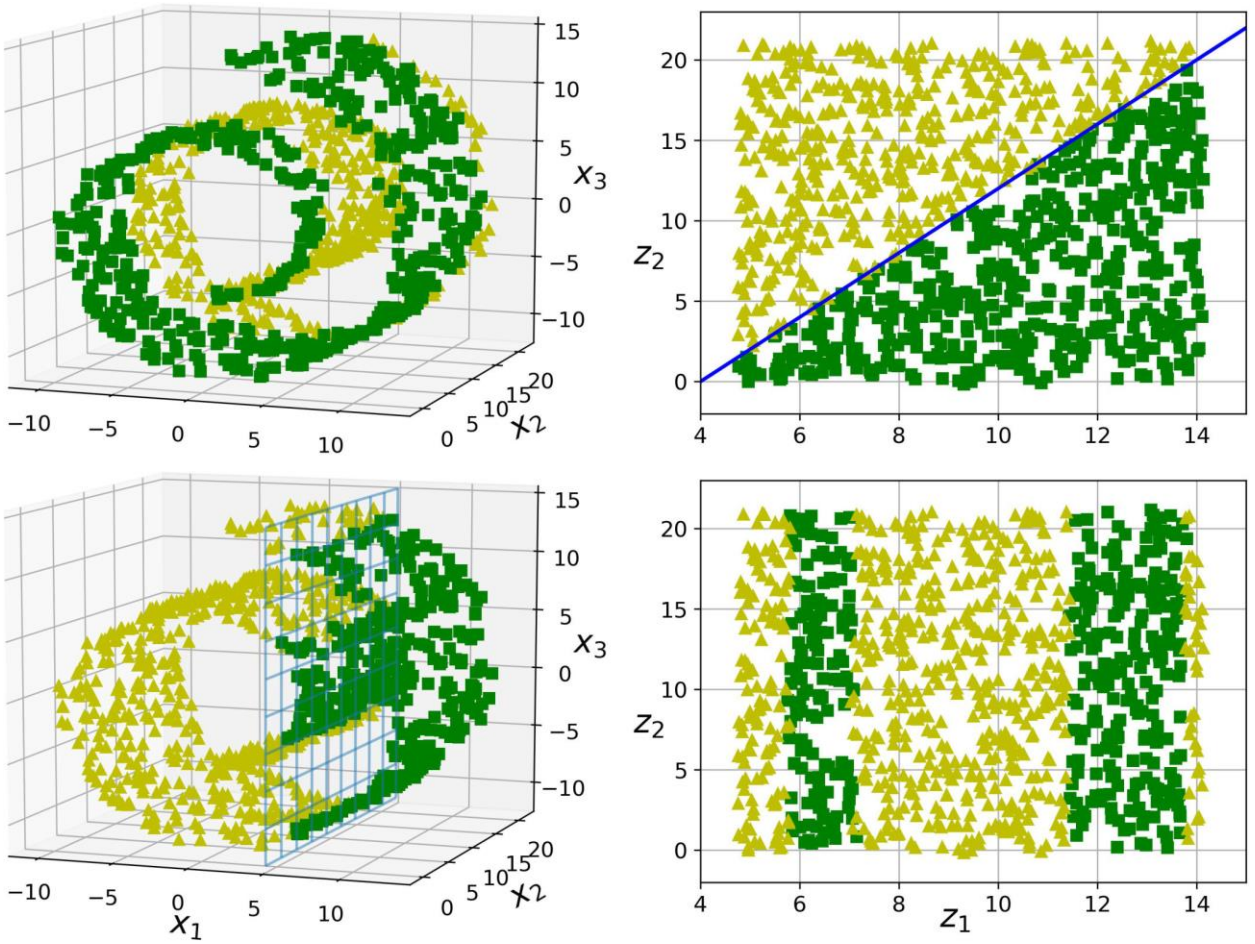
- Simply projecting onto a plane may not give better solution.
- Projecting to a proper manifold is better.





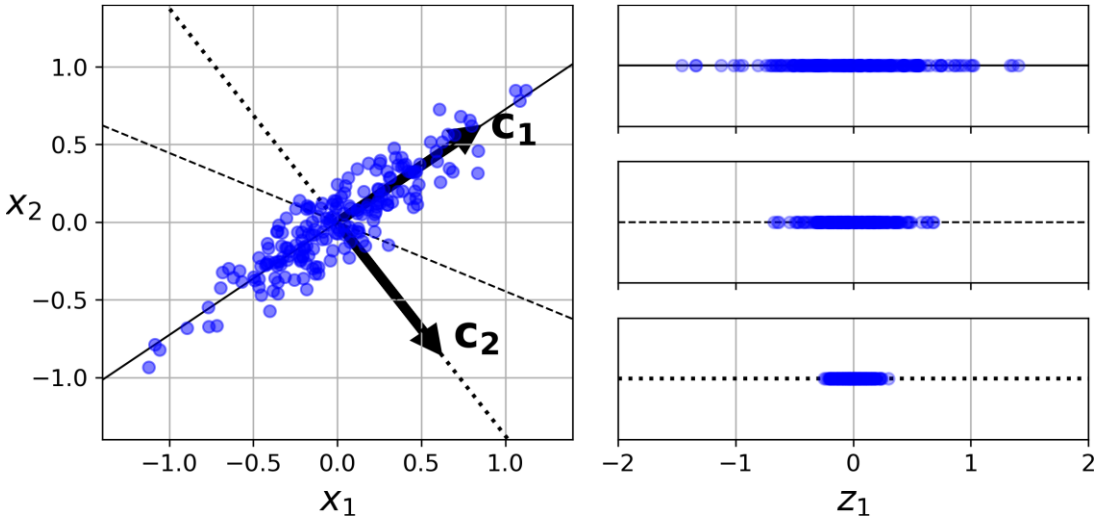
Projection and Manifold

- The decision boundary may not always be simpler with lower dimensions.



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)

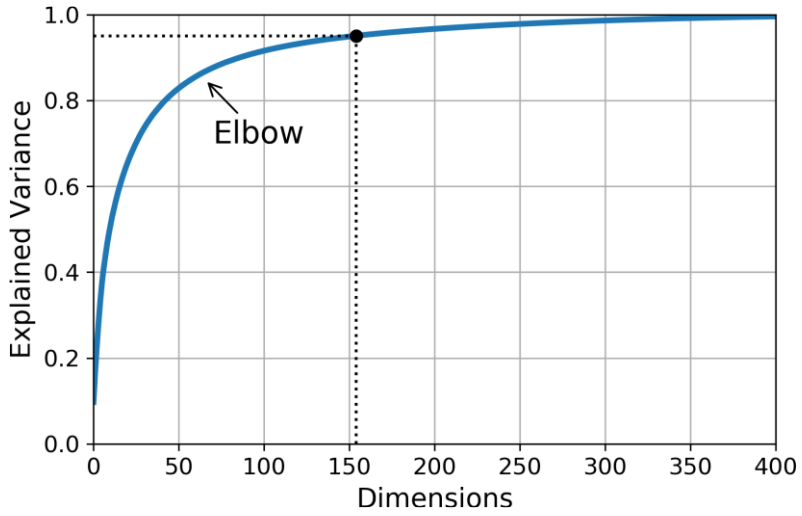
- Use PCA to reduce the dimensionality of the dataset down to **two dimensions**.
- Instead of specifying the number of principal components you want to preserve, you can set `n_components` to be a float between `0.0` and `1.0`, indicating the ratio of variance you wish to preserve.

```
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
  - Projection and Manifold
  - Principal Component Analysis (PCA)
- Unsupervised Learning
- Clustering
  - K-Means
  - DBSCAN
- Gaussian Mixtures and Anomaly Detection
- 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

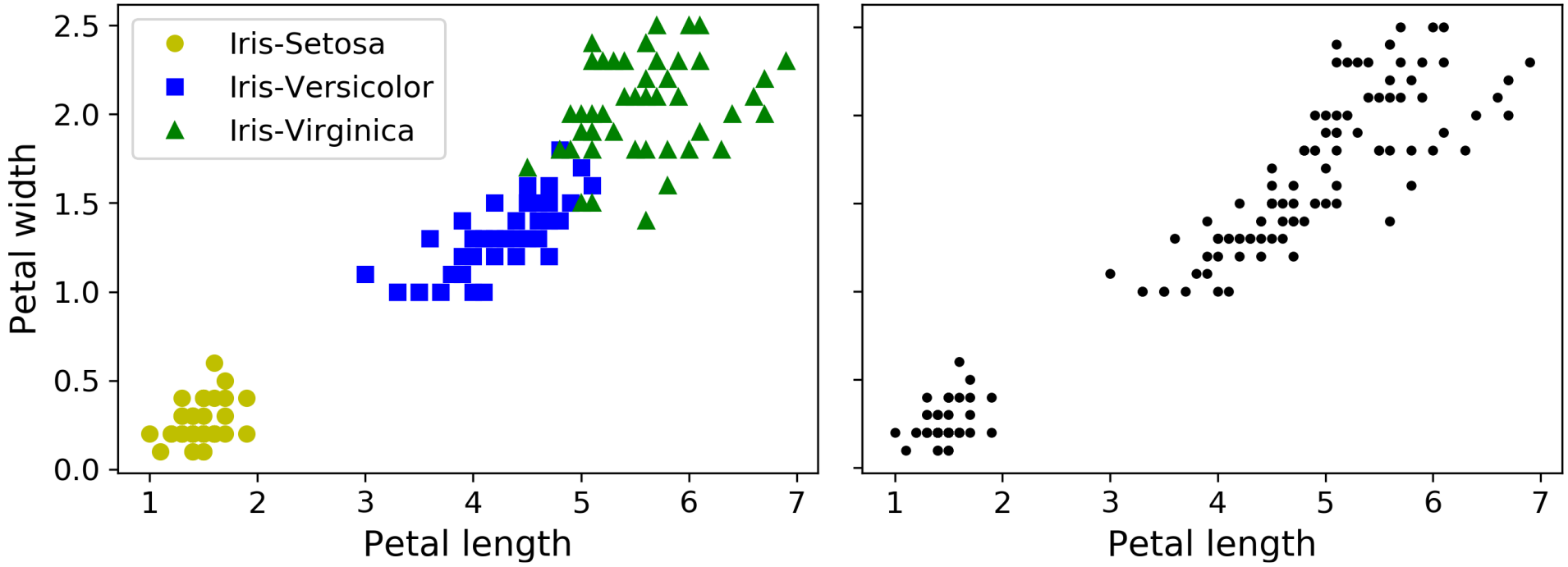
- **Example**: System that takes a few pictures of each item on a manufacturing production line and detects which items are defective.

Outline

- Dimensionality Reduction
  - Projection and Manifold
  - Principal Component Analysis (PCA)
- Unsupervised Learning
- Clustering
  - K-Means
  - DBSCAN
- Gaussian Mixtures and Anomaly Detection
- Exercises

Clustering

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

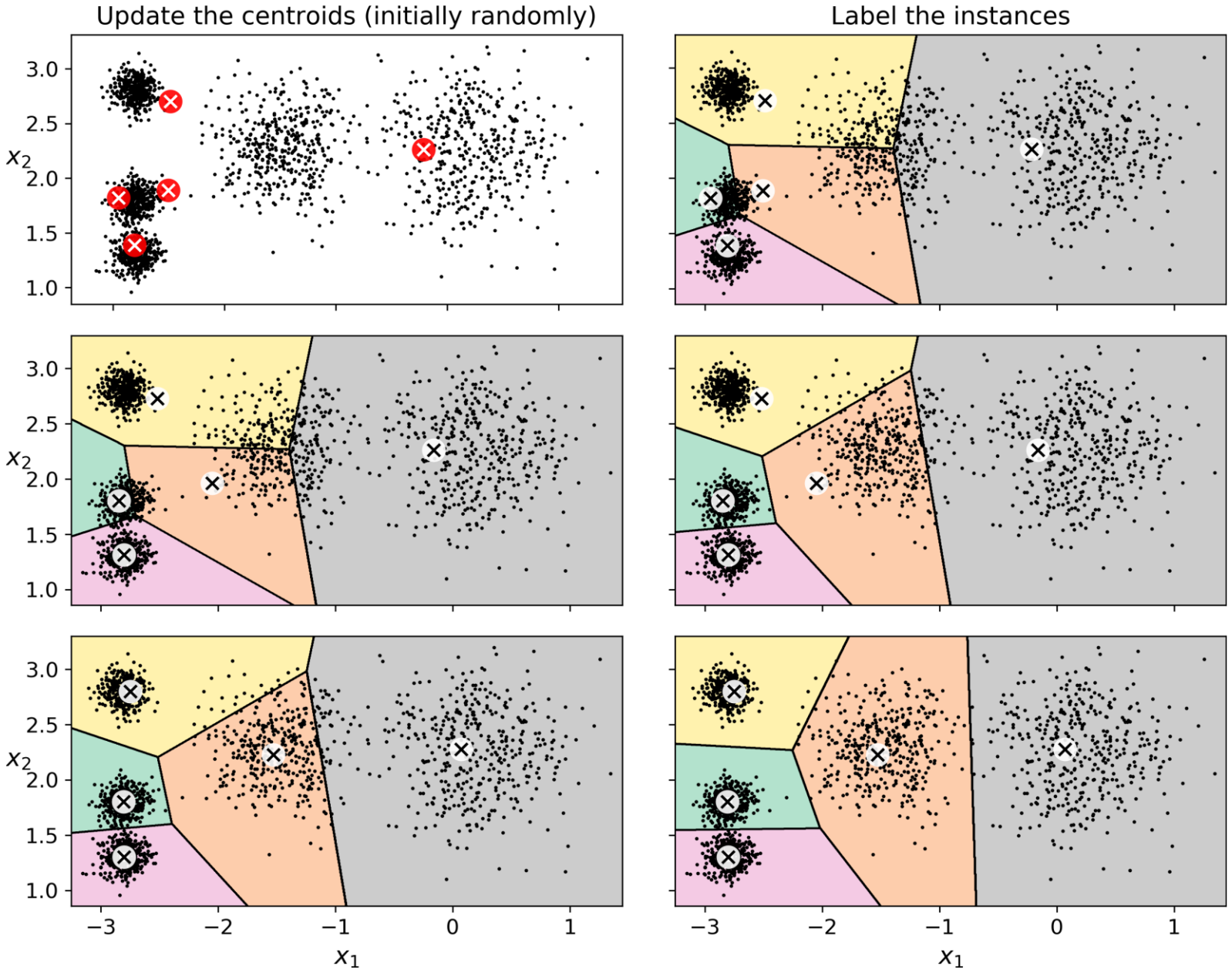


Clustering Applications

- **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**

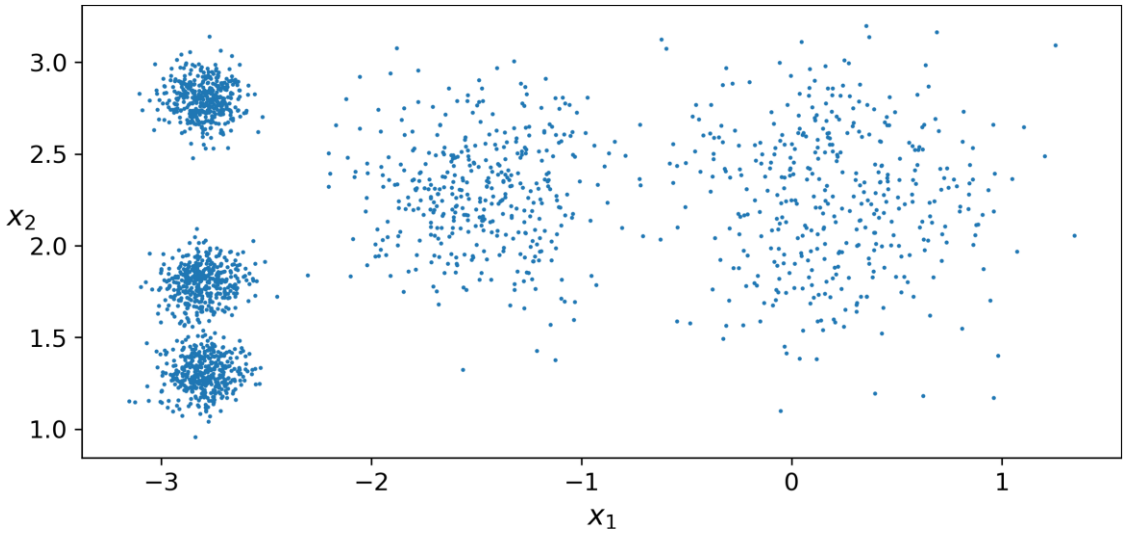
K-Means

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



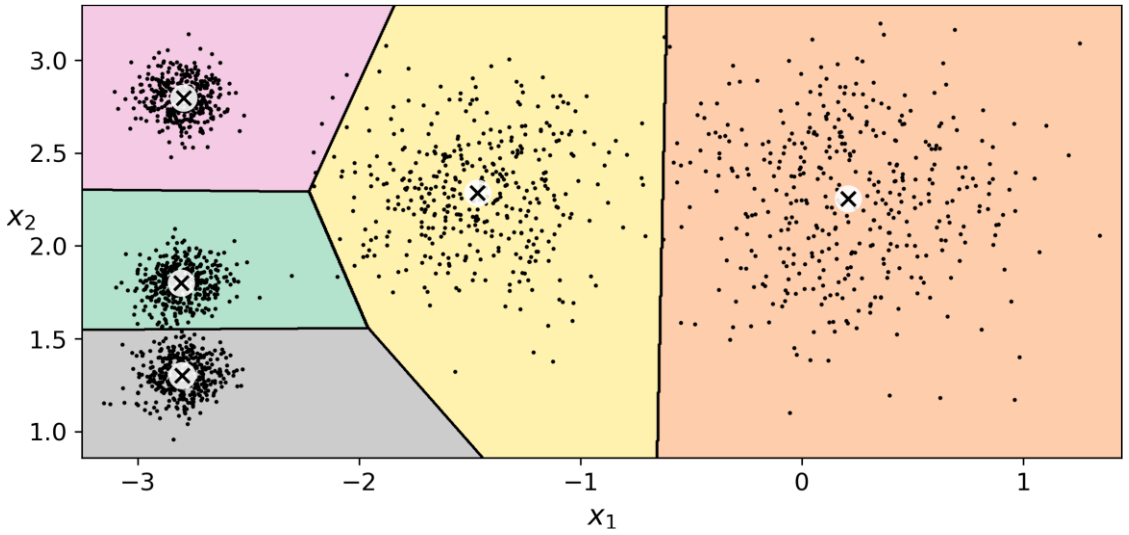
K-Means

- 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



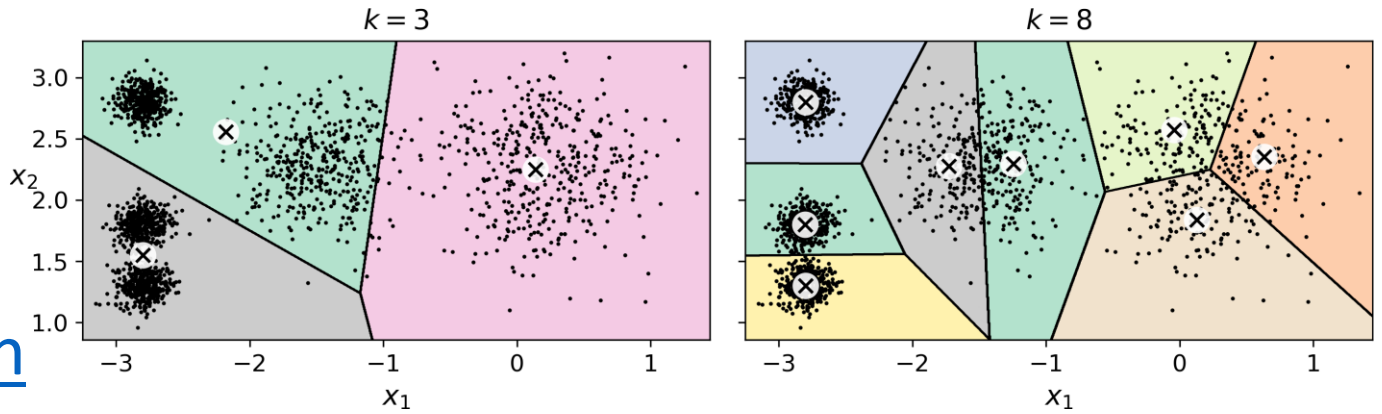

```
kmeans.cluster_centers_
array([[-2.80389616, 1.80117999],
       [ 0.20876306, 2.25551336],
       [-2.79290307, 2.79641063],
       [-1.46679593, 2.28585348],
       [-2.80037642, 1.30082566]])
```

Can be a dimensionality reduction technique.

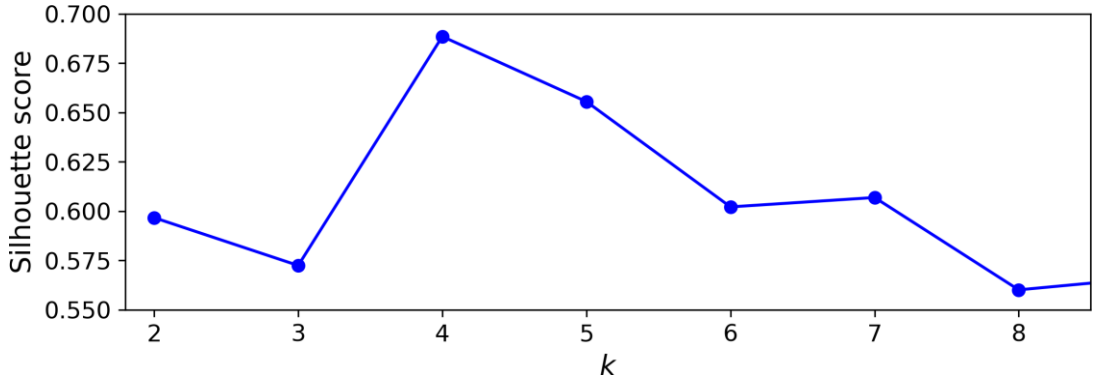
```
# Soft clustering, a score per
#  cluster:
kmeans.transform(X_new)
array([[2.81093633, 0.32995317,
        2.9042344 , 1.49439034,
        2.88633901],
       [1.21475352, 3.29399768,
        0.29040966, 1.69136631,
        1.71086031])
```

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
```



DBSCAN

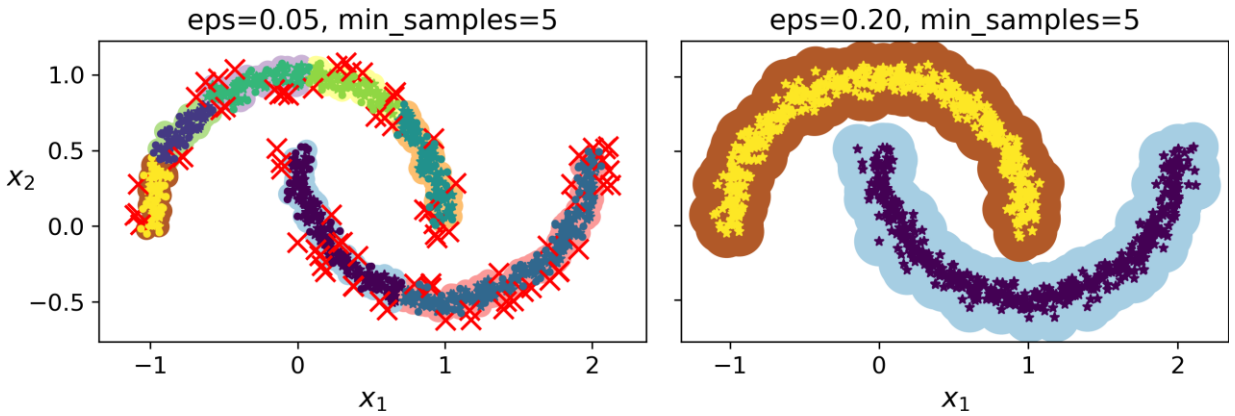
- Defines **clusters as continuous regions** of high density.
- Works well if all the **clusters are dense enough**, and they are well **separated by low-density regions**.
- Behaves well when the clusters have **varying sizes** or **non-spherical** shapes.

- **Algorithm**
  - For each instance, counts how many instances are located within a small distance **ε-neighborhood**.
  - If an instance has at least **min_samples** instances in its ε-neighborhood, then it is considered a **core instance**.
  - All instances in the neighborhood of a core instance belong to the same cluster. This may include other core instances; therefore, a long sequence of neighboring core instances forms a single cluster.
  - Any instance that is not a core instance and does not have one in its neighborhood is considered an **anomaly** (**-1**).

Can detect anomalies

DBSCAN

- Cluster the **moons** dataset

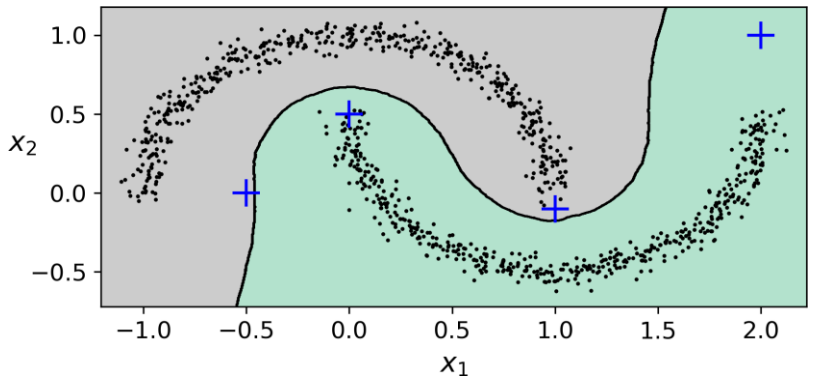



```
from sklearn.cluster import DBSCAN
from sklearn.datasets import
                              make_moons
X, y = make_moons(n_samples=1000,
                          noise=0.05)
dbscan = DBSCAN(eps=0.2,
                       min_samples=5)
dbscan.fit(X)
```

DBSCAN

- DBSCAN class does not have a **`predict()`** method.
- Can use other classifiers.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(dbscan.components_, dbscan.labels_[dbscan.core_sample_indices_])
X_new = np.array([[-0.5, 0], [0, 0.5], [1, -0.1], [2, 1]])
knn.predict(X_new)
array([1, 0, 1, 0])
```



Outline

- Dimensionality Reduction
  - Projection and Manifold
  - Principal Component Analysis (PCA)
- Unsupervised Learning
- Clustering
  - K-Means
  - DBSCAN
- Gaussian Mixtures and Anomaly Detection
- Exercises

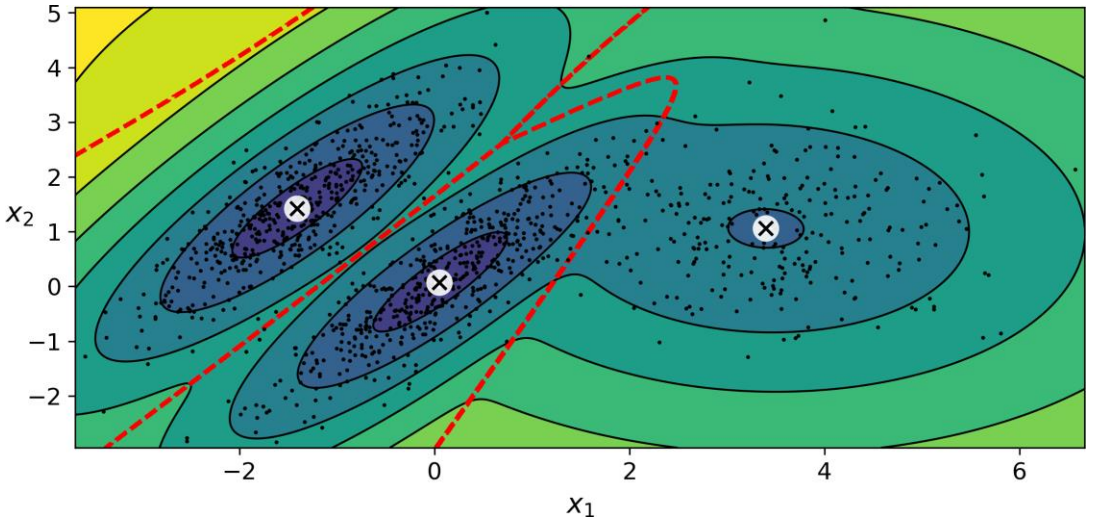
Gaussian Mixtures

- A **Gaussian mixture model** (**GMM**) is a probabilistic model that assumes that the instances were generated from a mixture of several Gaussian distributions whose parameters are unknown.
- Scikit-Learn's `GaussianMixture` class, given the dataset **X**, can estimate the weights $\boldsymbol{\varphi}$ and all the distribution parameters $\boldsymbol{\mu}^{(1)}$ to $\boldsymbol{\mu}^{(k)}$ and $\boldsymbol{\Sigma}^{(1)}$ to $\boldsymbol{\Sigma}^{(k)}$.

```
from sklearn.mixture import GaussianMixture
gm = GaussianMixture(n_components=3, n_init=10)
gm.fit(X)
```

Gaussian Mixtures

```
gm.converged_
True
gm.n_iter_
3
```

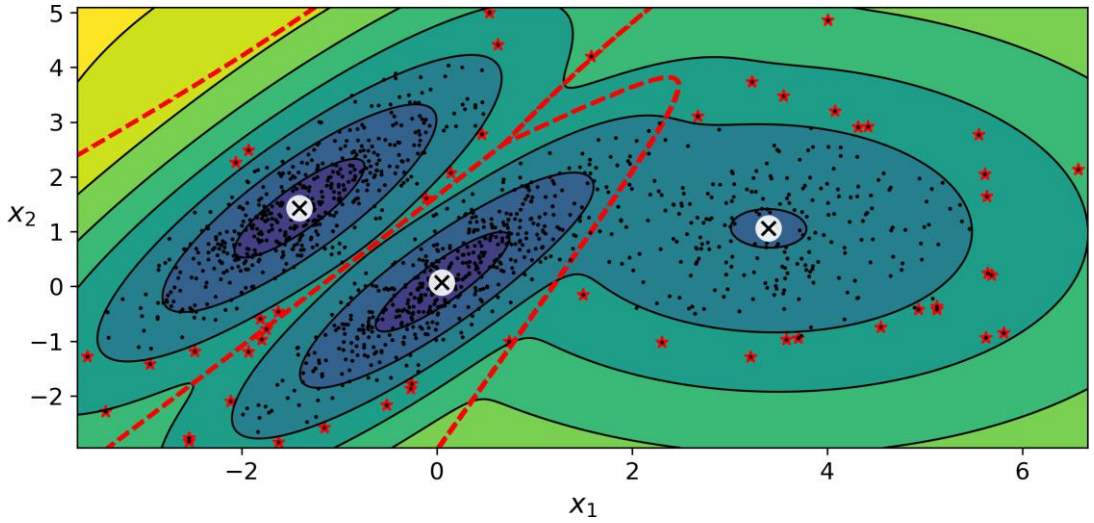



```
gm.weights_
array([0.20965228, 0.4000662,
       0.39028152])
gm.means_
array([[ 3.39909717, 1.05933727],
       [-1.40763984, 1.42710194],
       [ 0.05135313, 0.07524095]])
gm.covariances_
array([[[ 1.14807234, -0.03270354],
        [-0.03270354, 0.95496237]],
       [[ 0.63478101, 0.72969804],
        [ 0.72969804, 1.1609872 ]],
       [[ 0.68809572, 0.79608475],
        [ 0.79608475, 1.21234145]]])
```

Anomaly Detection using Gaussian Mixtures

- Any instance **located in a low-density region** can be considered an anomaly.
- **Identify the outliers** using the **4th percentile** lowest density as the threshold.

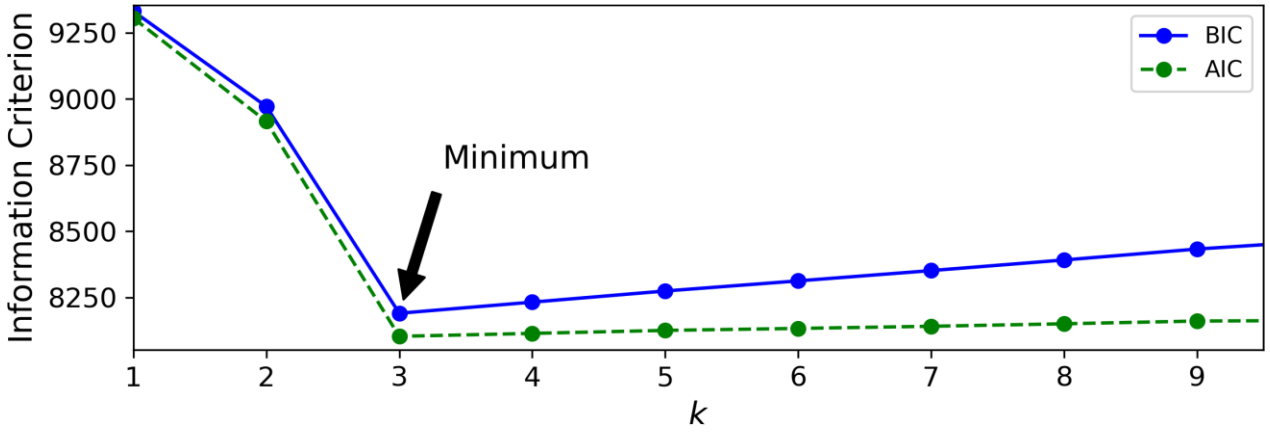
```
densities = gm.score_samples(X)
density_threshold = np.percentile(
                        densities, 4)
anomalies = X[densities <
                    density_threshold]
```



Selecting the Number of Components

- Minimize the **Bayesian information criterion** (**BIC**) or the **Akaike information criterion** (**AIC**).

```
gm.bic(X)
8189.74345832983
gm.aic(X)
8102.518178214792
```



Outline

- Dimensionality Reduction
  - Projection and Manifold
  - Principal Component Analysis (PCA)
- Unsupervised Learning
- Clustering
  - K-Means
  - DBSCAN
- Gaussian Mixtures and Anomaly Detection
- 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

9.3. Describe two techniques to **select the right number of clusters** when using **K-Means**.

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?

Summary

- Dimensionality Reduction
  - Projection and Manifold
  - Principal Component Analysis (PCA)
- Unsupervised Learning
- Clustering
  - K-Means
  - DBSCAN
- Gaussian Mixtures and Anomaly Detection
- Exercises