Uczenie nienadzorowane

Igor Wojnicki

April 21, 2024

Plan prezentacji

Uczenie nienadzorowane

K-Means

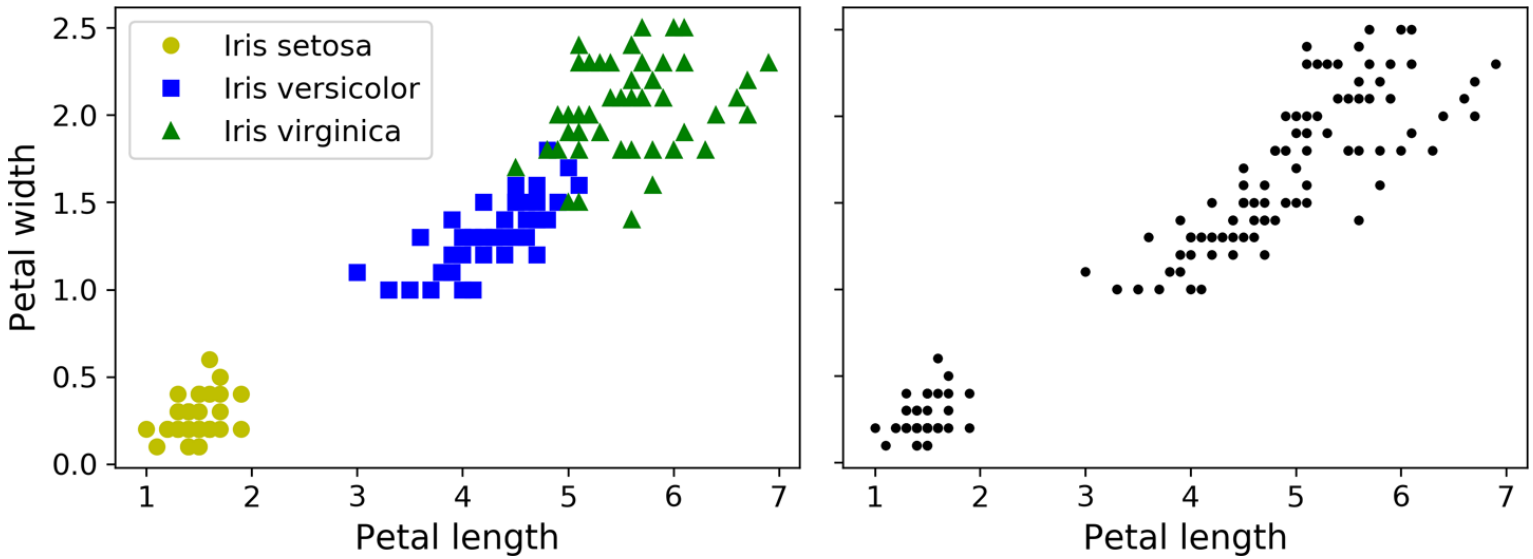
DBSCAN

Kategorie uczenia nienadzorowanego

- Klasteryzacja, *clustering*
  - identyfikacja klas (segmentacja klientów)
  - redukcja wymiarowości
  - analiza danych (po klasteryzacji, dla każdego klastra z osobna)
  - uczenie częściowo nadzorowane
  - segmentacja obrazu, detekcja, kompresja
- Detekcja anomalii, *anomaly detection*
  - detekcja wartości odstających, *outliers*
- Estymacja gęstości, *density estimation*

Klasteryzacja

Podobne do klasyfikacji, ale nie wiadomo ile jest klas.



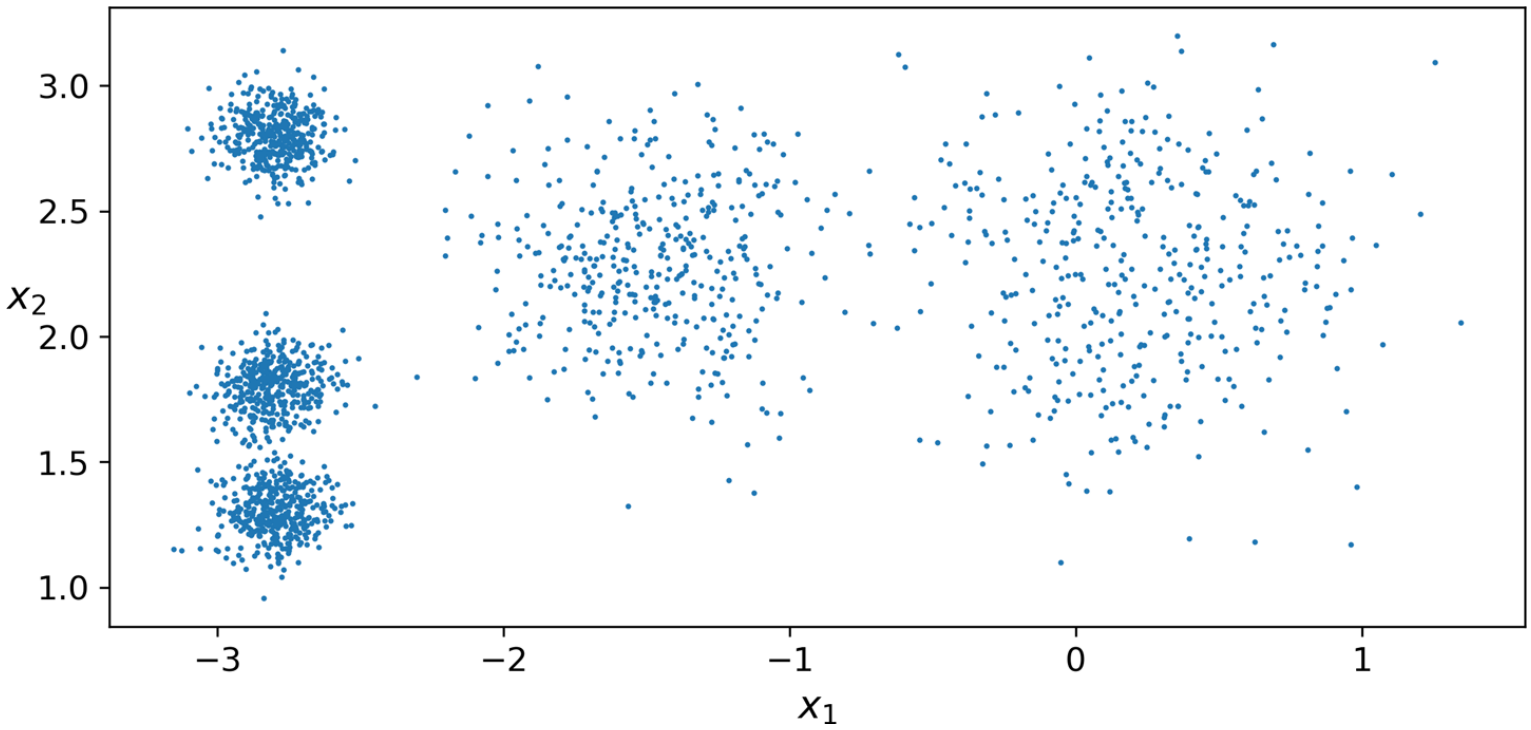
Plan prezentacji

Uczenie nienadzorowane

K-Means

DBSCAN

K-Means



- Algorytm centroidów (Wikipedia)
- Algorytm stara się znaleźć środek każdego z k skupisk.
- k jest parametrem algorytmu.

K-Means, przykład, dane

czyli jak zrobić rysunek z poprzedniego slajdu...

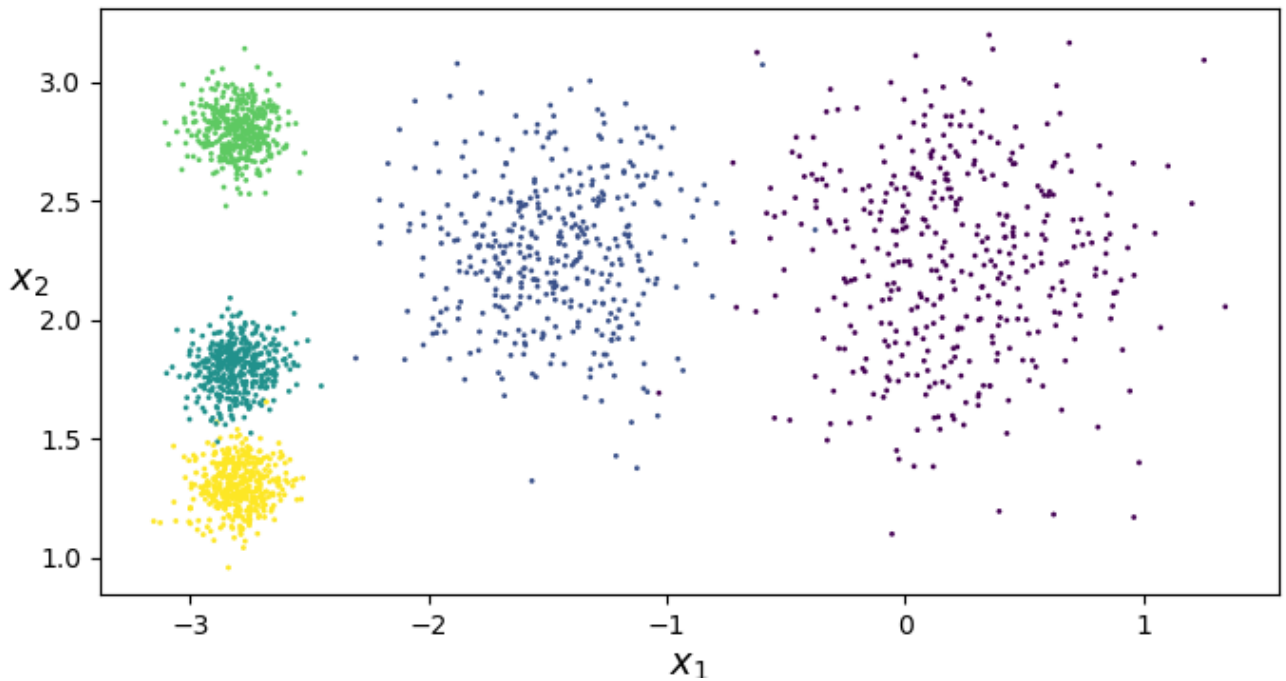
```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from sklearn.datasets import make_blobs
4  blob_centers = np.array(
5      [[ 0.2,  2.3],
6       [-1.5 ,  2.3],
7       [-2.8,  1.8],
8       [-2.8,  2.8],
9       [-2.8,  1.3]])
10 blob_std = np.array([0.4, 0.3, 0.1, 0.1, 0.1])
11 X, y = make_blobs(n_samples=2000, centers=blob_centers,
12                    cluster_std=blob_std, random_state=7)
```

K-Means, wizualizacja

```
1  def plot_clusters(X, y=None):
2      plt.scatter(X[:, 0], X[:, 1], c=y , s=1)
3      plt.xlabel("$x_1$", fontsize=14)
4      plt.ylabel("$x_2$", fontsize=14, rotation=0)
5
6  plt.figure(figsize=(8, 4))
7  plot_clusters(X,y) # zobaczmy skupiska
8  f = 'blobs_plot.png'
9  plt.savefig(f)
10 print(f)
```

K-Means, wizualizacja

Zwykle przynależność do skupisk jest nieznana, kolory dla celów dydaktycznych :)



K-Means, uczenie, wyniki

```
1  from sklearn.cluster import KMeans
2
3  k = 5
4  kmeans = KMeans(n_clusters=k, random_state=42)
5  y_pred = kmeans.fit_predict(X)
6  print(kmeans.labels_) # to samo co predict
7  print(y_pred)
8  print(kmeans.cluster_centers_)
```

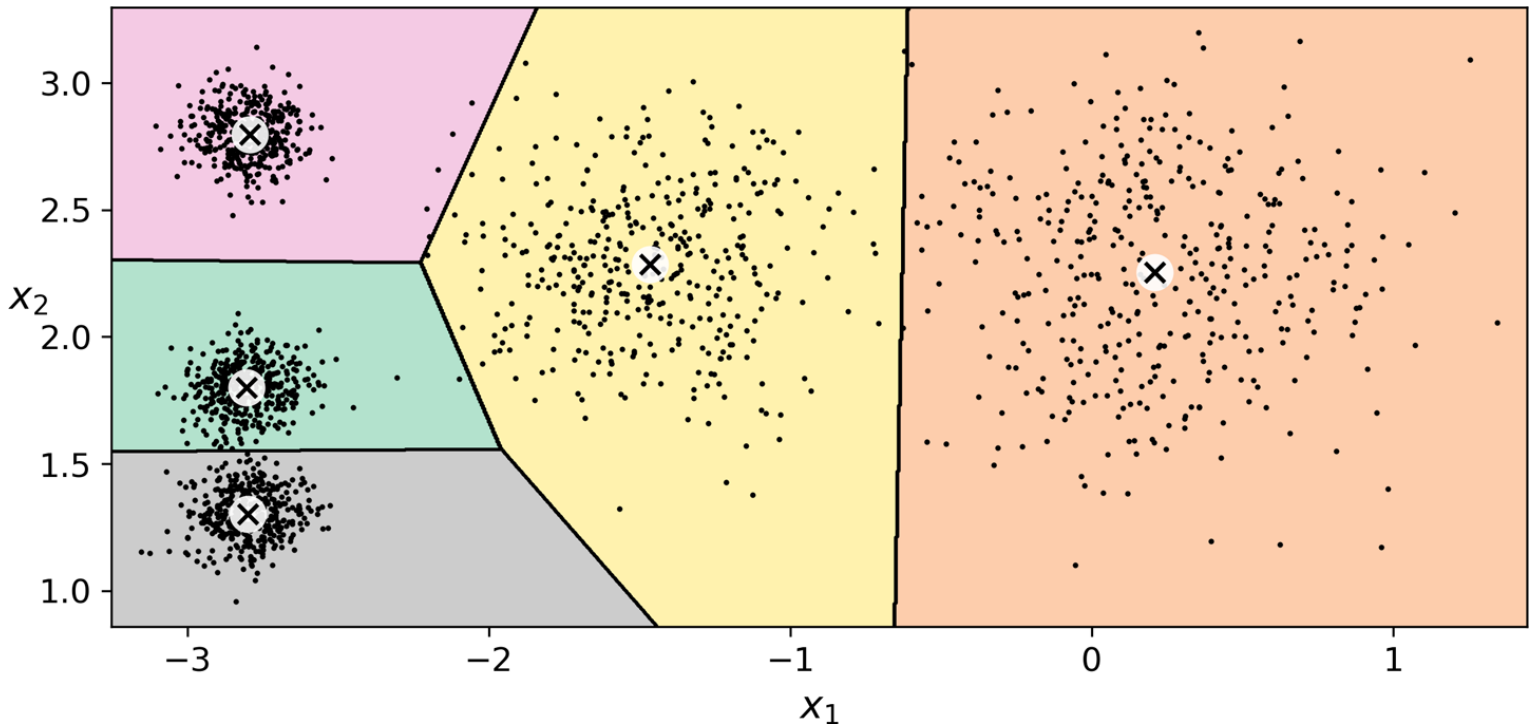
```
[4 1 0 ... 3 0 1]
[4 1 0 ... 3 0 1]
[[ 0.20876306  2.25551336]
 [-2.80389616  1.80117999]
 [-1.46679593  2.28585348]
 [-2.79290307  2.79641063]
 [-2.80037642  1.30082566]]
```

- `fit_predict()` = `fit()` + `predict()`

K-Means, wyniki cd.

```
1  print(kmeans.predict([[-3,2],[1,3]]))
```

```
[1 0]
```



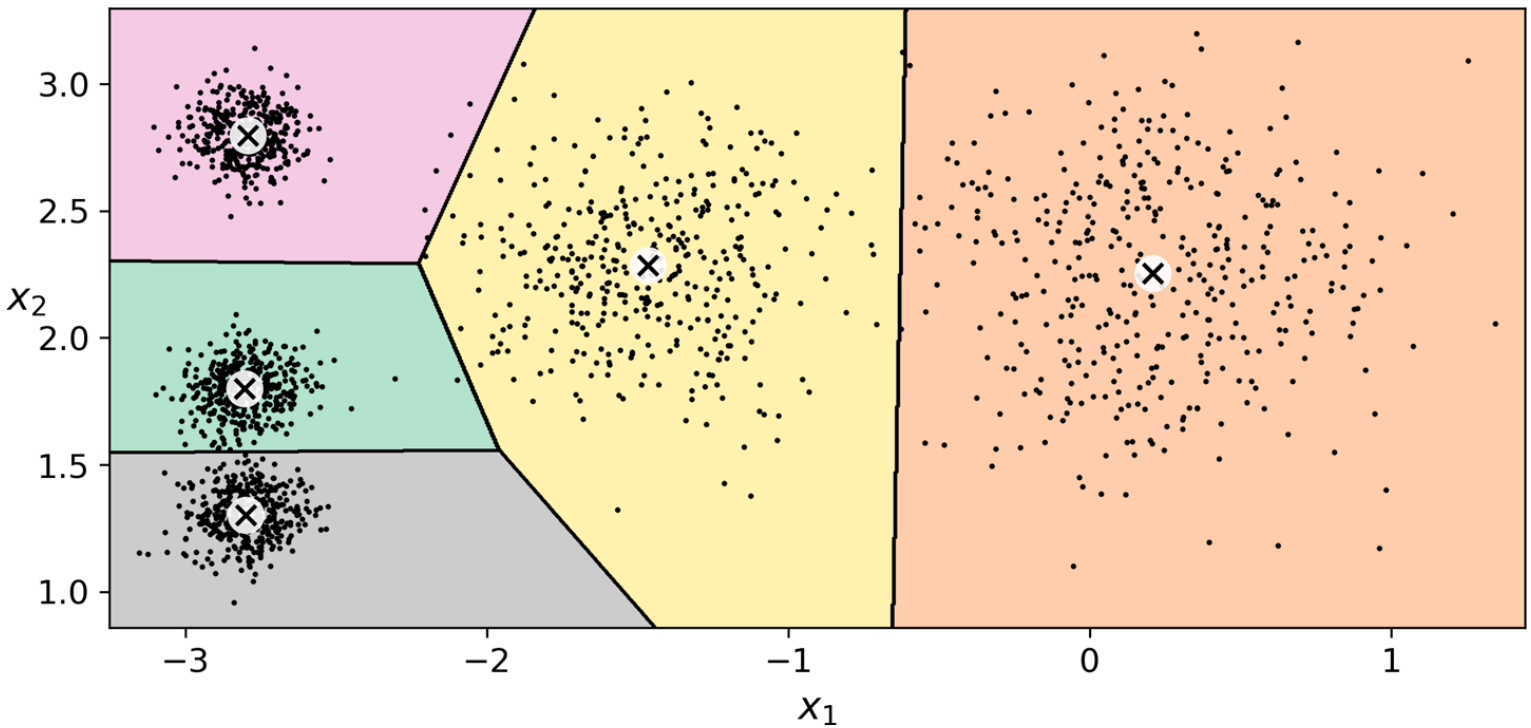
- diagram Woronoja (Voronoi)

K-Means, soft clustering

Transformacją do przestrzeni odległości od centroidów.

```
1 print(kmeans.transform([[-3,2],[1,3]]))
```

```
[[3.21892023 0.27925993 1.55962398 0.82289673 0.7271137 ]
 [1.08642361 3.98833241 2.56809022 3.79836311 4.16293819]]
```

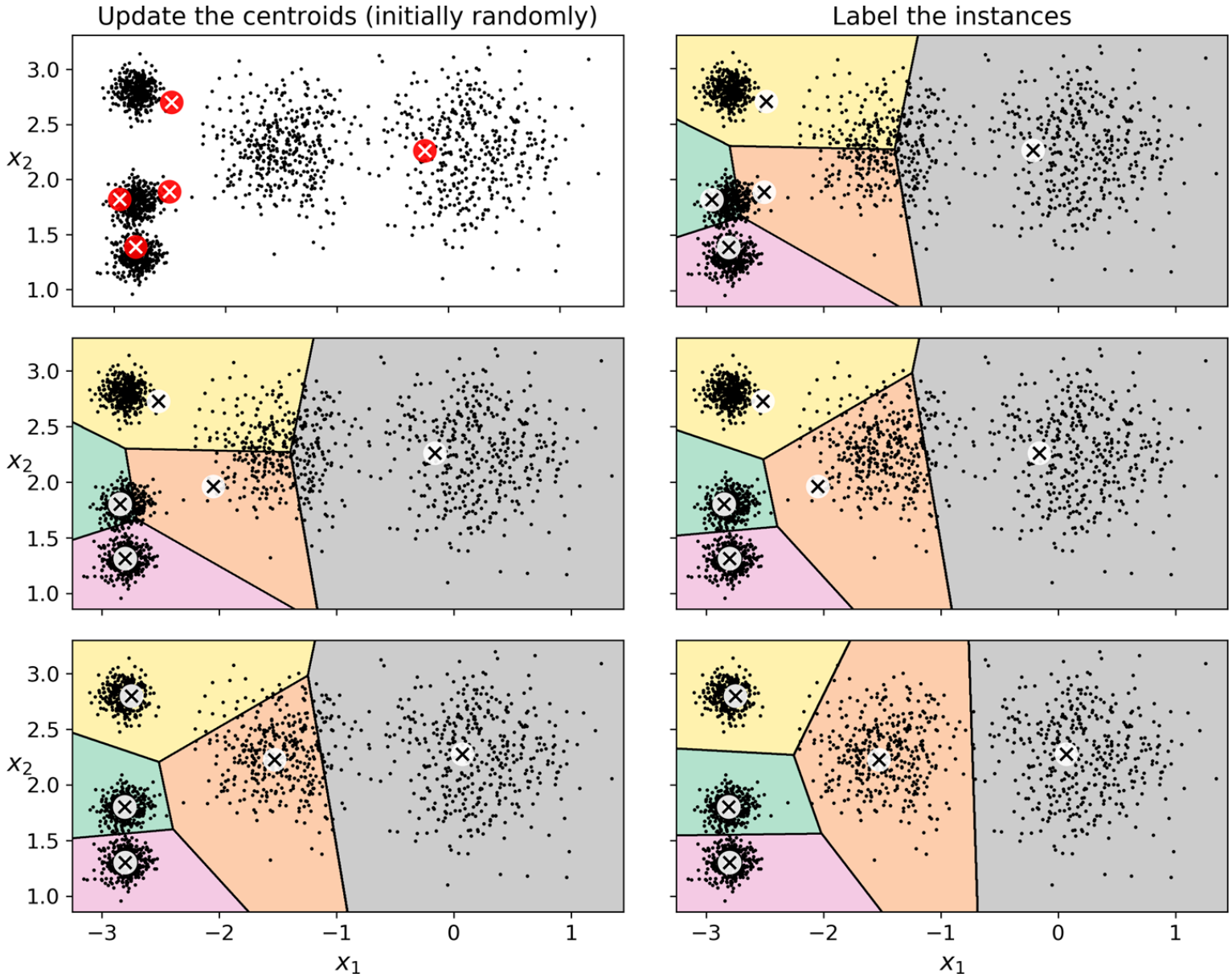


K-Means, algorytm

1. wylosuj lokalizacje k centroidów,
2. przyporządkuj instancje do najbliższego centroidu,
3. oblicz nową lokalizację każdego centroidu jako średnią z lokalizacji instancji należących do niego,
4. jeżeli zmieniła się pozycja centroidu to idź do: 2

`https://en.wikipedia.org/wiki/K-means_clustering#/media/File:K-means_convergence.gif`

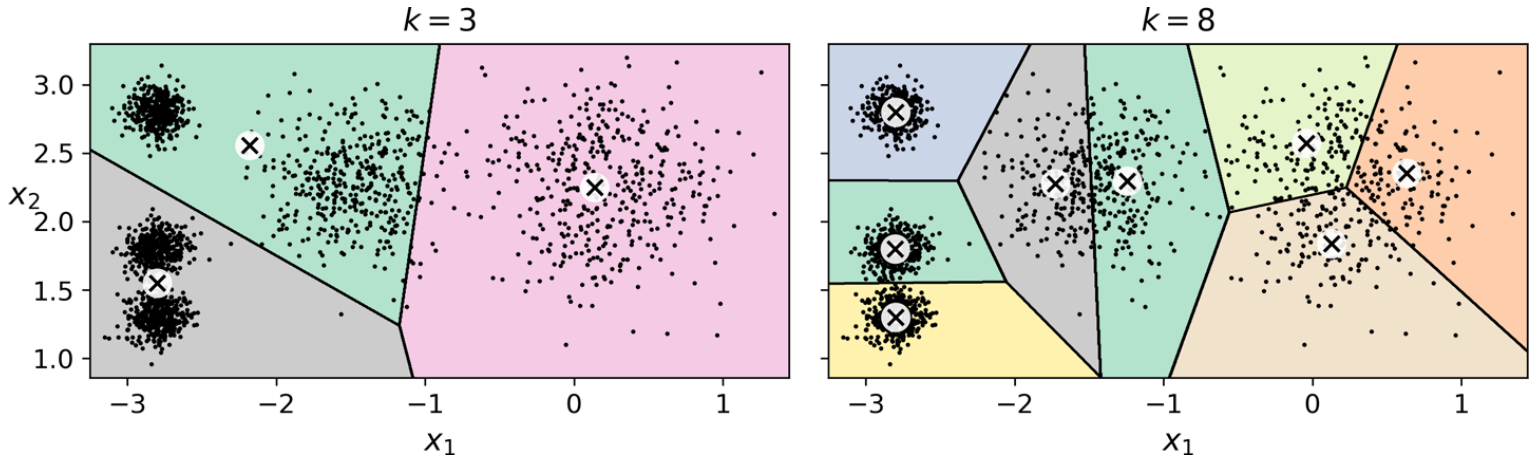
K-Means, algorytm



K-Means, algorytm

- Zbieżny,
- nie gwarantuje znalezienia optimum – zależy od kroku 1,
  - domyślnie algorytm uruchamiany jest 10 razy (parametr: `n_init=10`),
  - wybierany jest model z najmniejszą *inercją*: średnio-kwadratowa odległość między instancjami i centroidami.
    - pomierz odległość pomiędzy każdą instancją, a jej centroidem,
    - zsumuj kwadraty w/w odległości w ramach klastra,
    - zsumuj wartości inercji dla wszystkich klastrów.
    - `KMeans.inertia_`

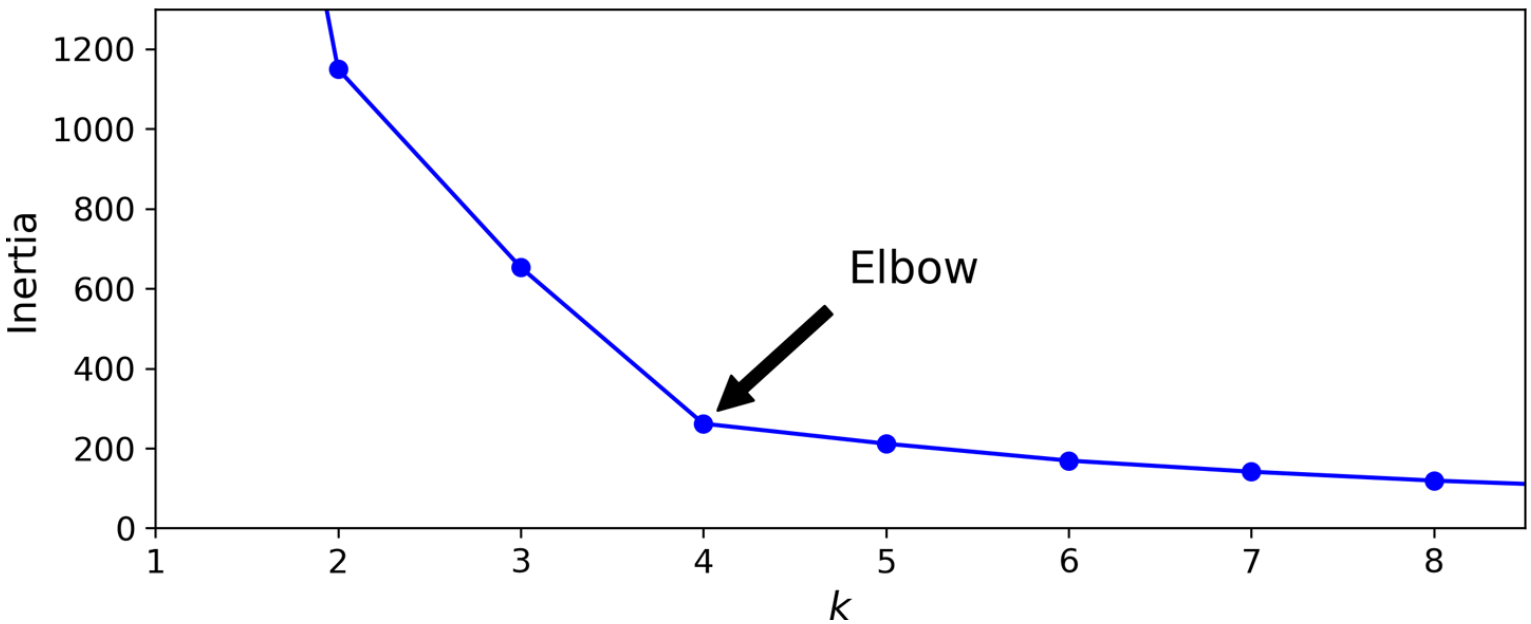
Ile klastrów/grup?



- *inercja* wprost nie pomoże... im więcej klastrów tym mniejsza.

Ile klastrów/grup?

- Ale można jej użyć:

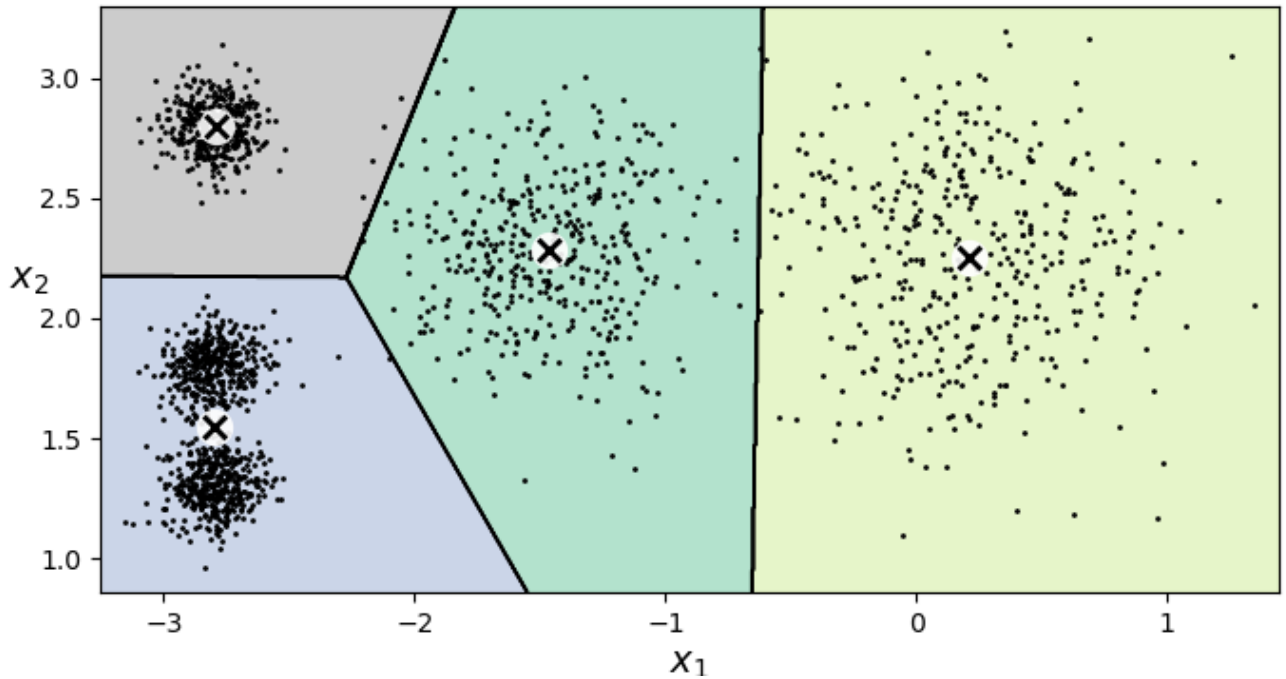


Jak wyglądają granice dla k=4?

```
1  from sklearn.cluster import KMeans
2
3  kmeans4 = KMeans(n_clusters=4)
4  y_pred = kmeans4.fit(X)
5
6  plt.cla()
7  plot_decision_boundaries(kmeans4, X)
8  f = 'kmeans4.png'
9  plt.savefig(f)
10 print(f)
```

- plot_decision_boundaries():
  https://github.com/ageron/handson-ml2/blob/master/09_unsupervised_learning.ipynb

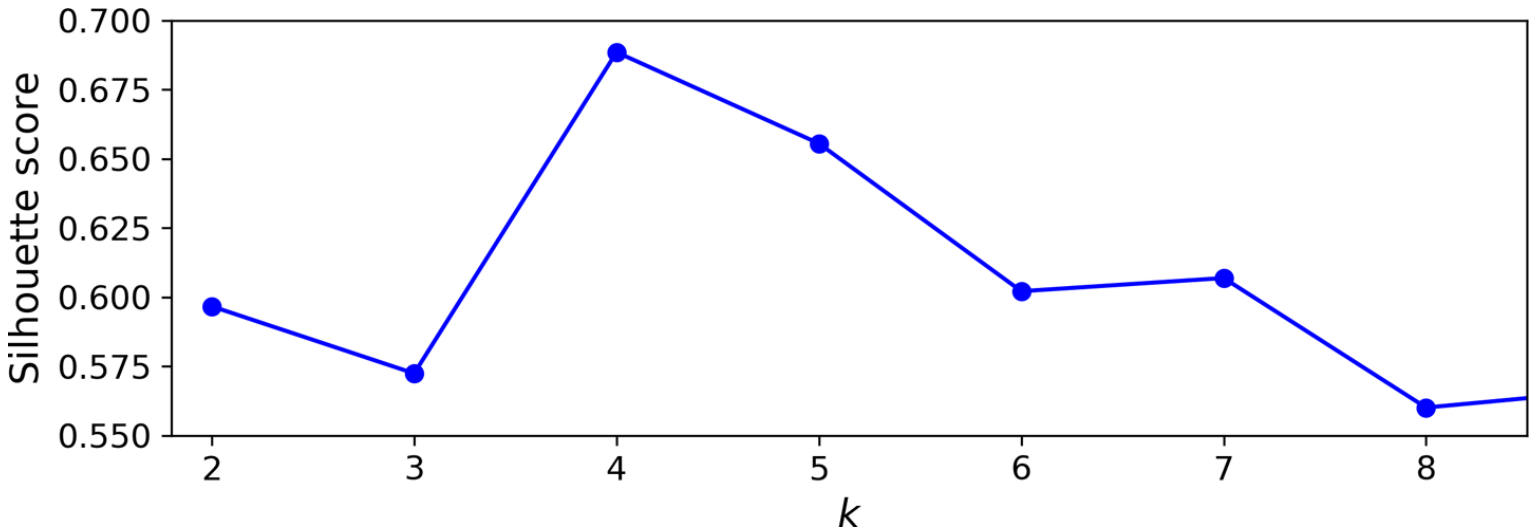
K-Means, k=4



- Nie musi być dokładnie taki podział - zależy od 1-go kroku algorytmu!

Ile klastrów/grup?

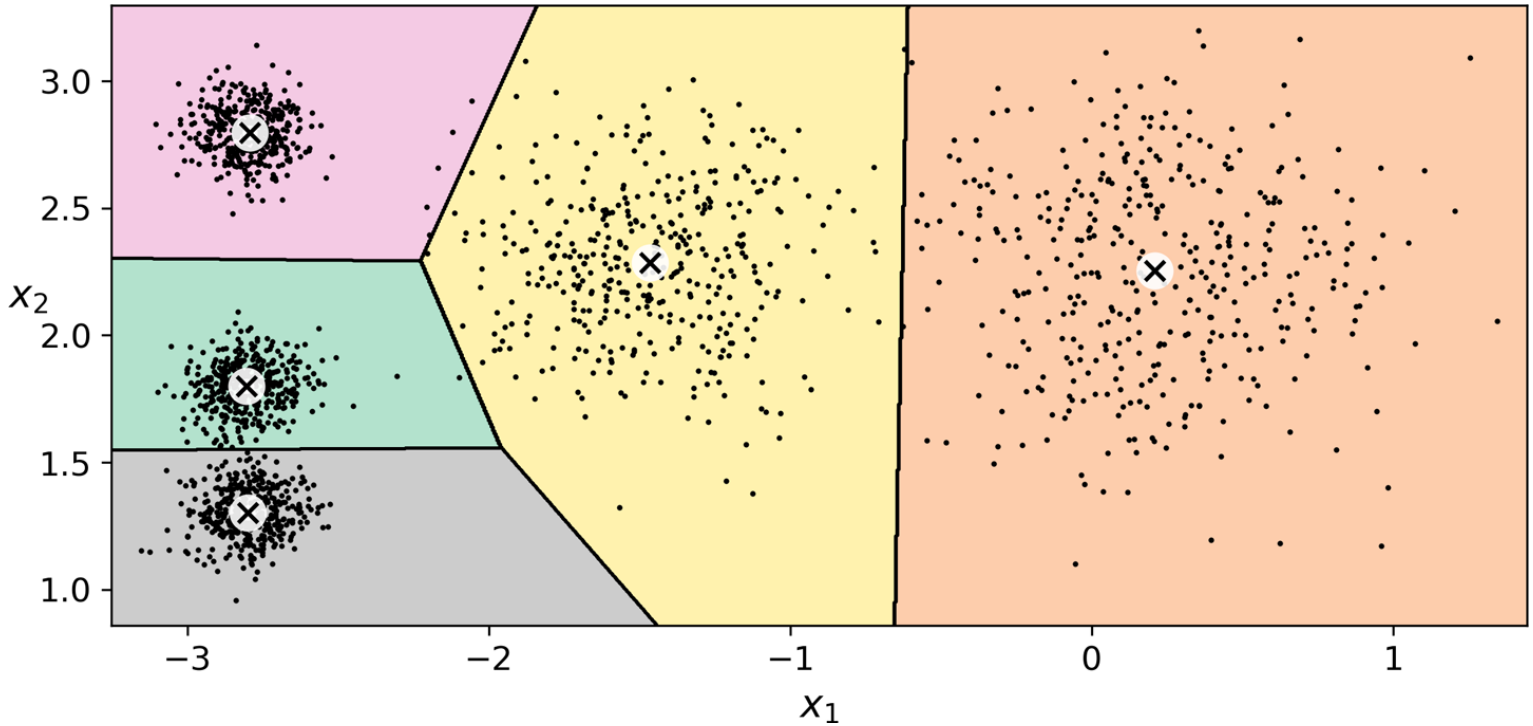
- Wskaźnik sylwetkowy, *silhouette score*.
  Średnia odległość pomiędzy obserwacjami wewnątrz grupy (a) i średnią odległość obserwacji do najbliższej „obcej” grupy (b).
  Silhouette obliczany jest dla każdej obserwacji w następujący sposób:
  $s = (a - b)/max(a, b)$.
  Uśredniona dla całego zbioru.

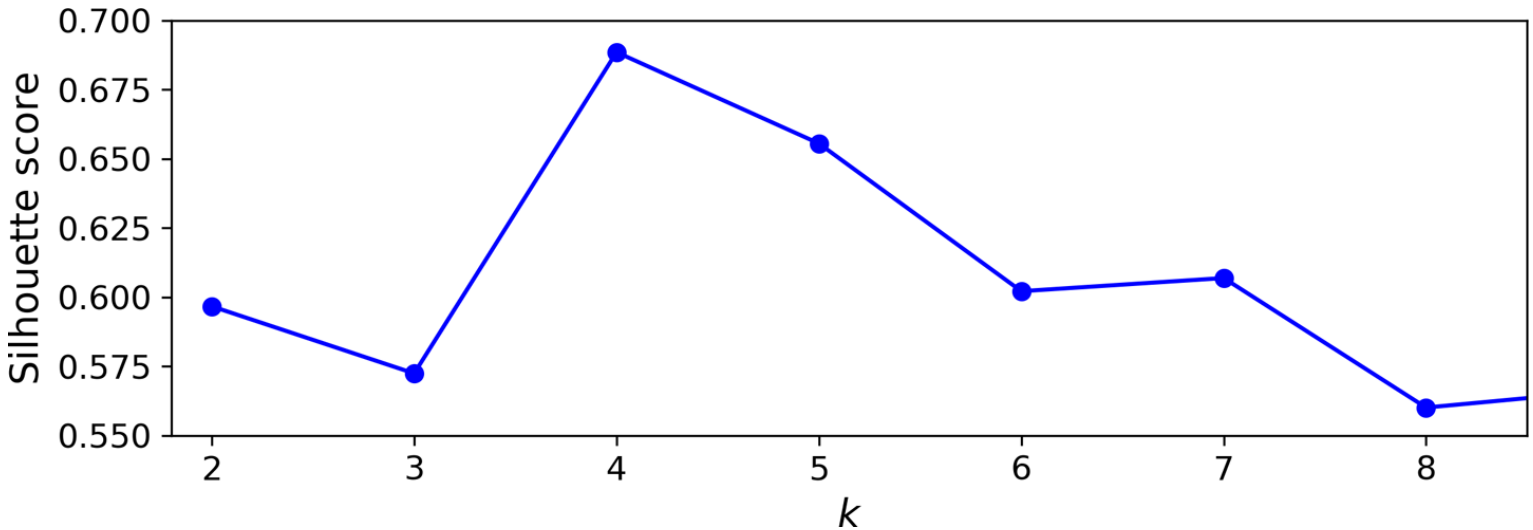



```
1 from sklearn.metrics import silhouette_score
2 print(silhouette_score(X, kmeans.labels_))
```

```
0.655517642572828
```

To ile w końcu grup?





Prosty przykład K-Means

```
1  from sklearn.cluster import KMeans
2
3  X = [ [1], [1], [2], [5], [6], [10] ]
4
5  kmeans = KMeans(n_clusters=2)
6  y_pred = kmeans.fit_predict(X)
7  print(kmeans.labels_)
8  print(y_pred)
9  print(kmeans.cluster_centers_)
```

```
[0 0 0 1 1 1]
[0 0 0 1 1 1]
[[1.33333333]
 [7.        ]]
```

Plan prezentacji

Uczenie nienadzorowane

K-Means

DBSCAN

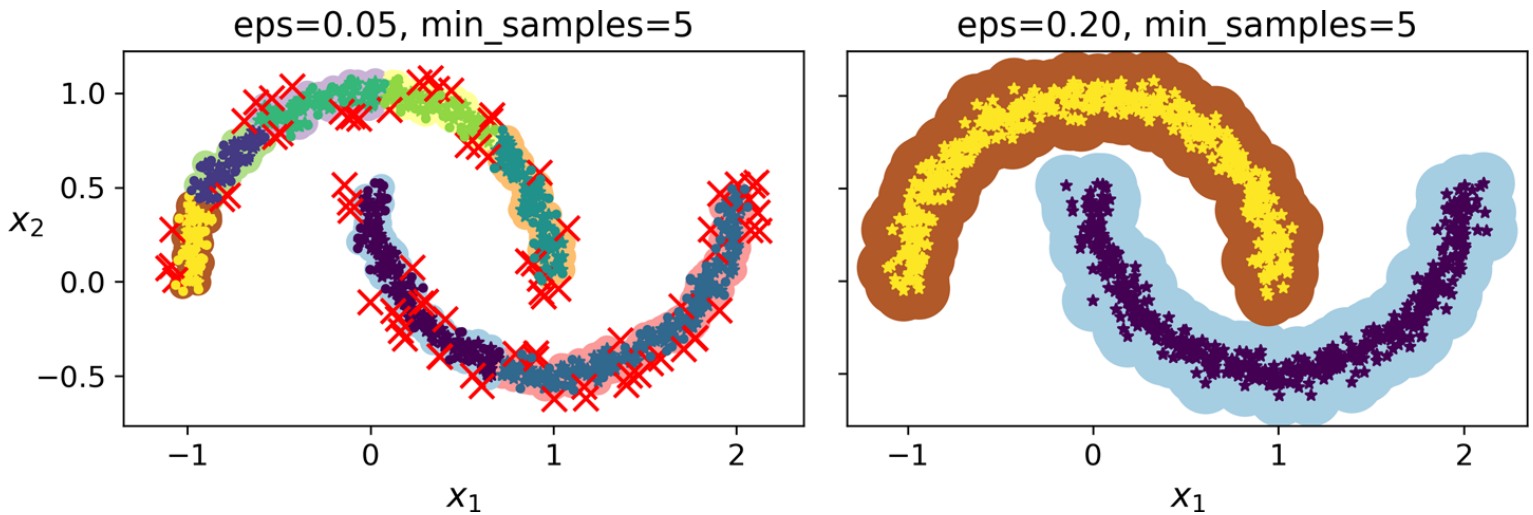
DBSCAN

1. Dla każdej instancji liczone jest ile instancji jest odległych o ϵ (wliczając tą instancję), tzw. sąsiedztwo-ϵ.
2. Jeżeli instancja ma przynajmniej `min_samples` w swoim sąsiedztwie-ϵ jest oznaczana jako rdzeń (*core*); tzw. instancja w "gęstym" obszarze.
3. Wszystkie instancje w sąsiedztwie rdzenia należą do tego samego klastra; sąsiednie, ϵ odległe rdzenie wchodzą w skład tego samego klastra.
4. Instancja, która nie jest rdzeniem i nie przynależy do żadnego rdzenia jest anomalią (wartością odstającą) (*outlier*).

DBSCAN, przykład

```
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=1000, noise=0.05,
                  random_state=42)

from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.20, min_samples=5)
dbscan.fit(X)
```



DBSCAN, wyniki

```
print(dbscan.labels_[:15])
```

```
[0 0 0 0 1 0 0 0 0 1 0 1 0 0 0]
```

- Wartości ujemne dla anomalii.

```
dbscan = DBSCAN(eps=0.05, min_samples=5)
dbscan.fit(X)
print(dbscan.labels_[:15])
```

```
[ 0  2 -1 -1  1  0  0  0  2  5  2  3  0  2  2]
```

Prosty przykład DBSCAN

```
1  from sklearn.cluster import DBSCAN
2
3  X = [ [1], [1], [2], [5], [6], [10] ]
4
5  dbscan = DBSCAN(eps=1, min_samples=2)
6  dbscan.fit(X)
7  print(dbscan.labels_)
8  print(dbscan.components_)
```

```
[ 0  0  0  1  1 -1]
[[1]
 [1]
 [2]
 [5]
 [6]]
```