

# Uczenie nienadzorowane

Igor Wojnicki

April 11, 2022

# Plan prezentacji

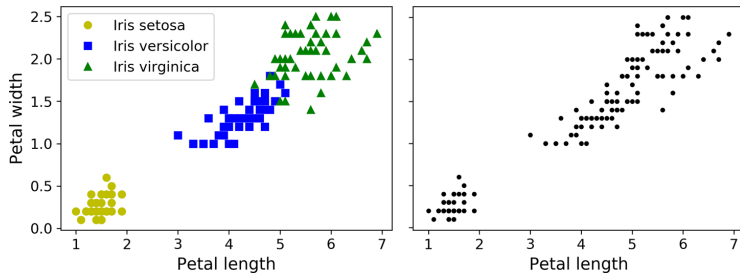
Uczenie nienadzorowane

# Kategorie uczenia nienadzorowanego

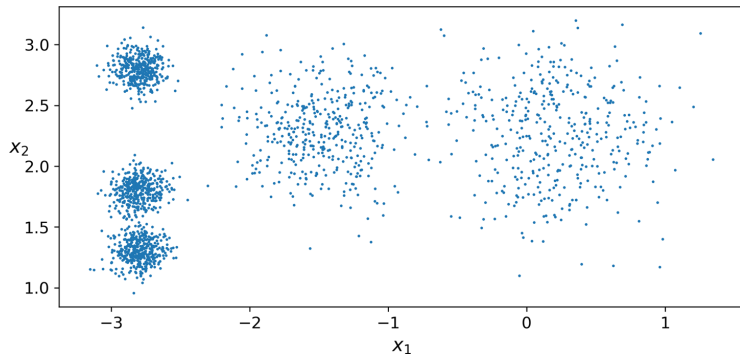
- ▶ Klasteryzacja, *clustering*
  - ▶ identyfikacja klas (segmentacja klientów)
  - ▶ redukcja wymiarowości
  - ▶ analiza danych (po klasteryzacji, dla każdego klastra z osobna)
  - ▶ uczenie częściowo nadzorowane
  - ▶ segmentacja obrazu, detekcja, kompresja
- ▶ Detekcja anomalii, *anomaly detection*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

# Klasteryzacja

Podobne do klasyfikacji, ale nie wiadomo ile jest klas.



# Algorytm centroidów, K-Means



- ▶ Algorytm\_centroidów
- ▶ Algorytm stara się znaleźć środek każdego z  $k$  skupisk.
- ▶  $k$  jest parametrem algorytmu.

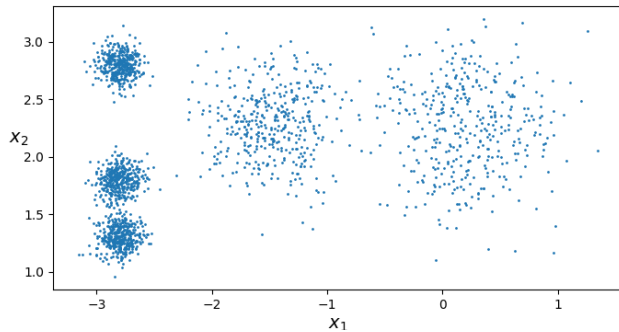
## K-Means, przykład, dane

```
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)
8  f = 'blobs_plot.png'
9  plt.savefig(f)
10 print(f)
```

# K-Means, wizualizacja





## 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_)
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]
```

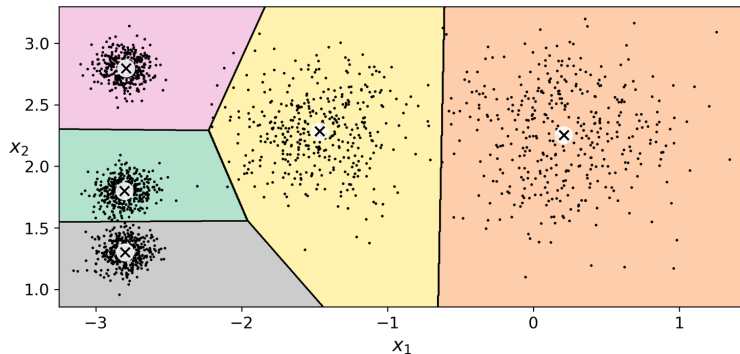
```
 [-2.80037642  1.30082566]]
```

► fit\_predict() = fit() + predict()

## K-Means, wyniki cd.

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

[1 0]

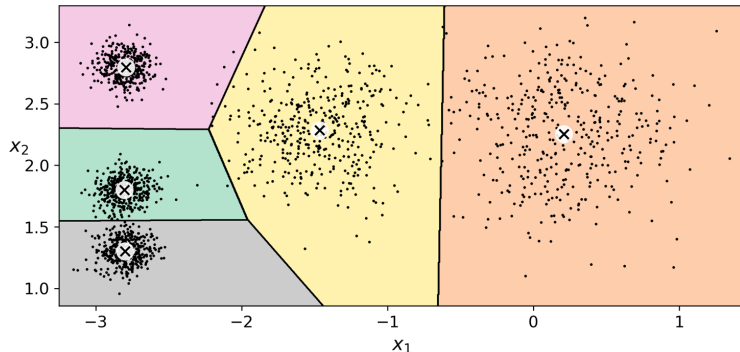


► diagram Woronoja (Voronoi)

# K-Means, soft clustering

```
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]]
```

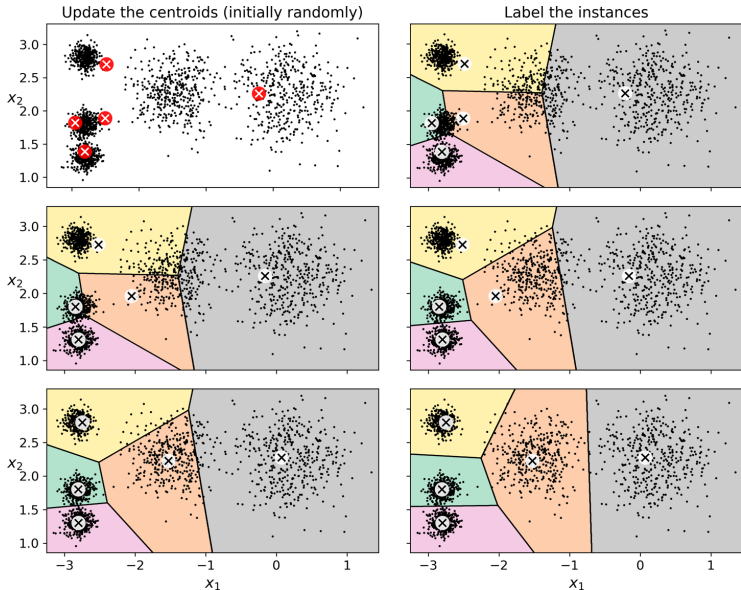


# K-Means, algorytm

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

Animacja z: [https://en.wikipedia.org/wiki/K-means\\_clustering](https://en.wikipedia.org/wiki/K-means_clustering)

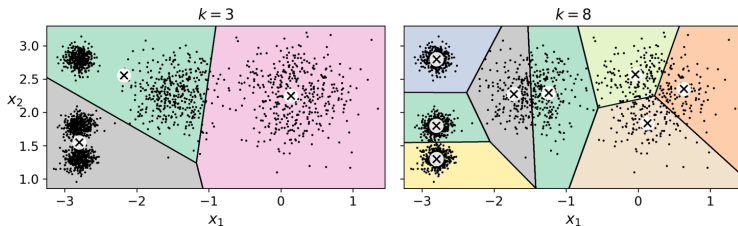
# K-Means, algorithm



# 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 jego centroidem,
    - ▶ zsumuj kwadraty w/w odległości w ramach klastra,
    - ▶ zsumuj wartości inercji dla wszystkich klastrów.

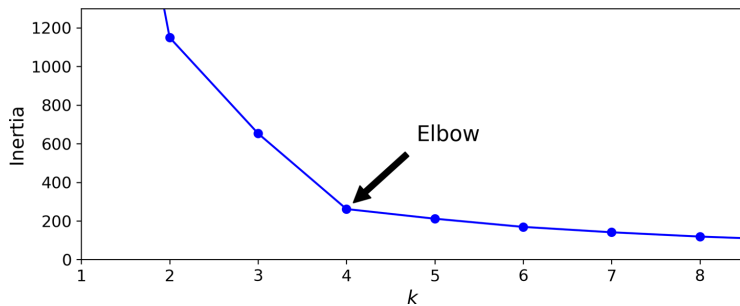
# Ile klastrow/grup?



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

# Ile klastrow/grup?

- ▶ Ale można jej użyć:



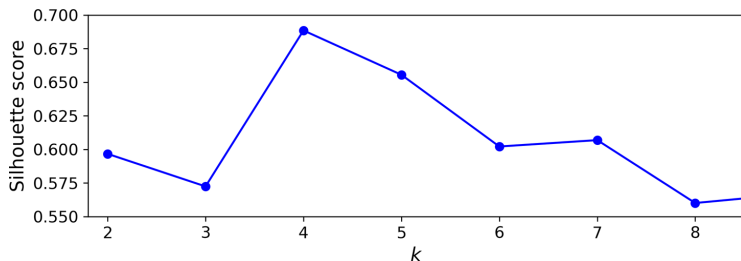


## Ile klastrów/grup?

- ▶ 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:

$$s = (a - b) / \max(a, b).$$

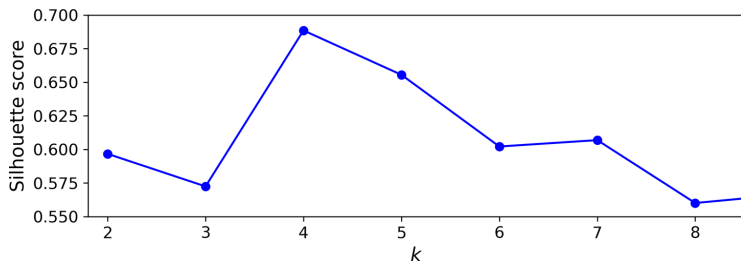
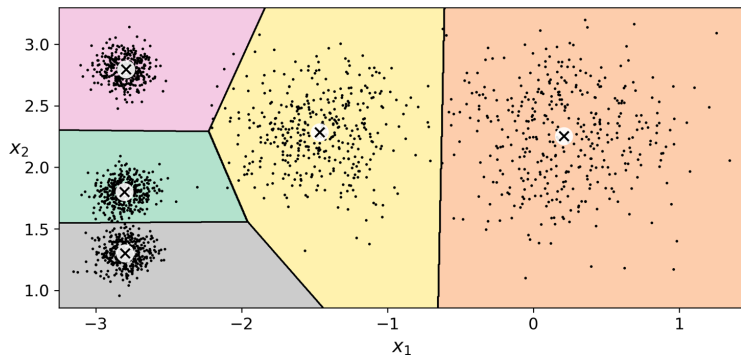
Uśredniona dla całego zbioru.



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

0.655517642572828

To ile w końcu grup?

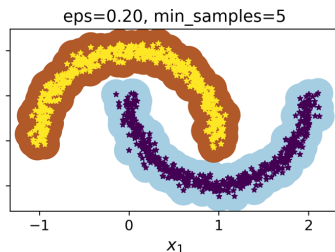
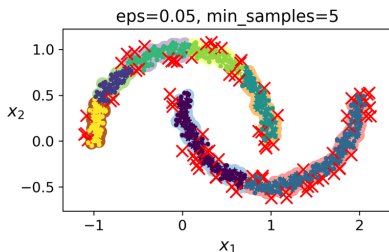


# DBSCAN

1. Dla każdej instancji liczone jest ile instancji jest odległych o  $\epsilon$ , tzw. sąsiedztwo- $\epsilon$ .
2. Jeżeli instancja ma przynajmniej `min_samples` w swoim sąsiedztwie- $\epsilon$  jest oznaczana jako rdzeń (*core*); tzw. instancja w "gęstym" obszarze.
3. Wszystkie instancje w sąsiedztwie rdzenia należą do tego samego klastra, włączając w to instancje przynależne do innych rdzeni; sąsiednie,  $\epsilon$  odległe rdzenie wchodzi w skład tego samego klastra.
4. Instancja, która nie jest rdzeniem i nie przynależy do żadnego rdzenia jest anomalią (*outlier*).

# DBSCAN, przykład

```
1 from sklearn.datasets import make_moons
2 X, y = make_moons(n_samples=1000, noise=0.05,
3                   random_state=42)
4
5 from sklearn.cluster import DBSCAN
6 dbscan = DBSCAN(eps=0.20, min_samples=5)
7 dbscan.fit(X)
```



## DBSCAN, wyniki

```
1 print(dbscan.labels_[:15])  
  
[0 0 0 0 1 0 0 0 0 1 0 1 0 0 0]
```

► Wartości ujemne dla anomalii.

```
1 dbscan = DBSCAN(eps=0.05, min_samples=5)  
2 dbscan.fit(X)  
3 print(dbscan.labels_[:15])  
  
[ 0  2 -1 -1  1  0  0  0  2  5  2  3  0  2  2]
```