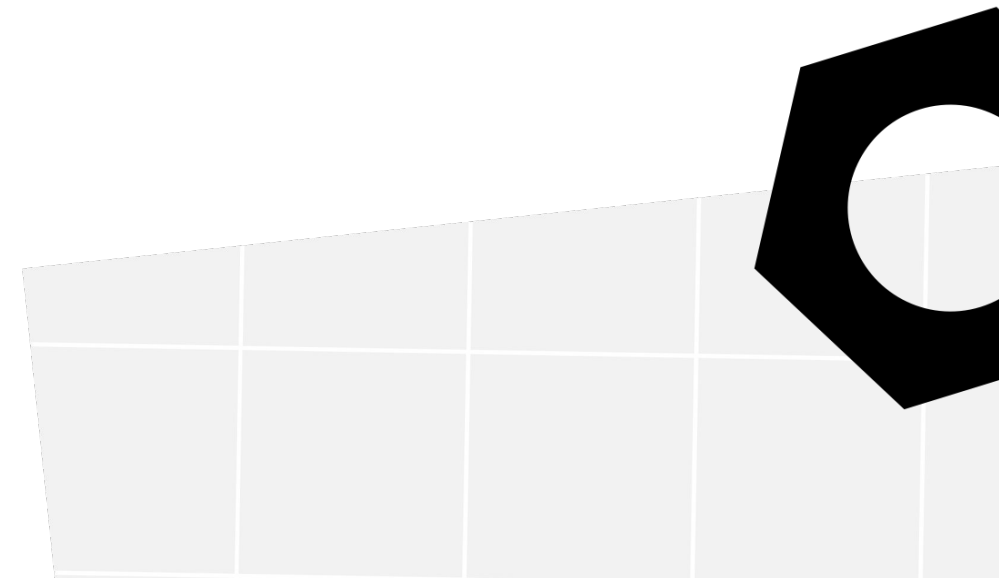




Machine Learning w praktyce: problemy klasteryzacji w uczeniu nienadzorowanym

Kurs Data Science





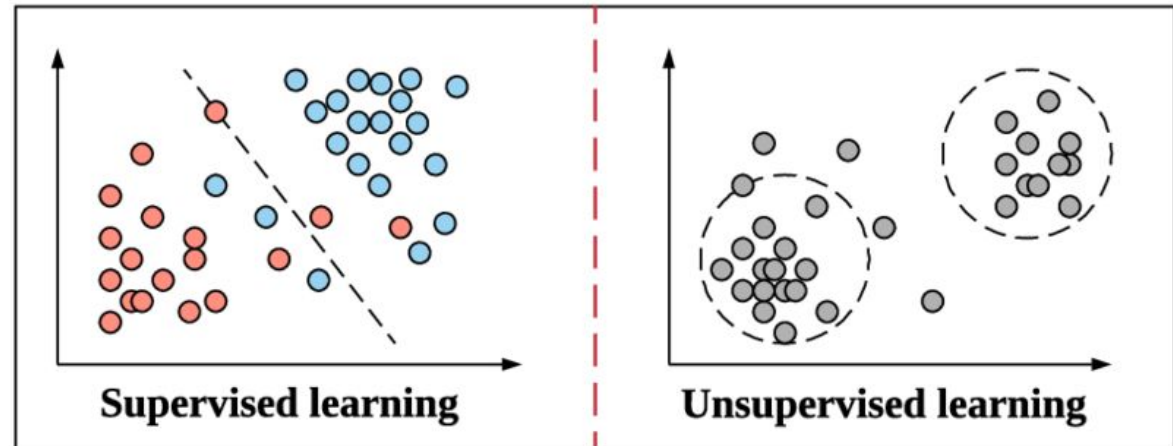
Dodatkowe materiały do zajęć

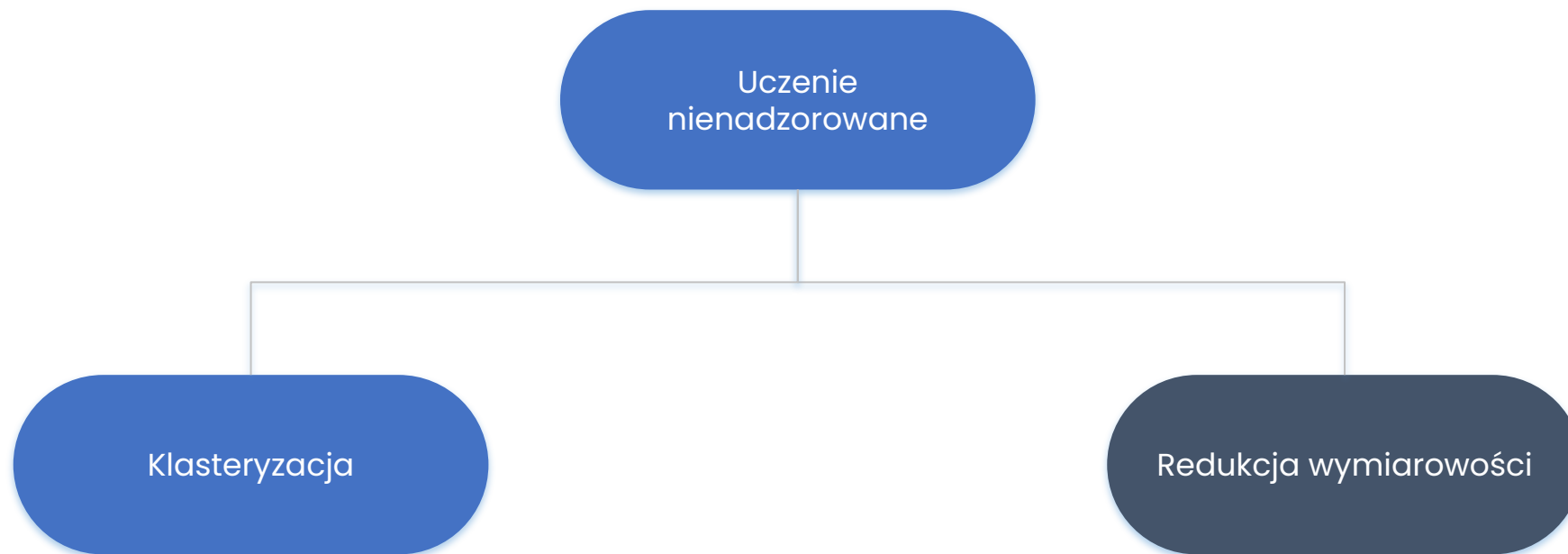
- [Dokumentacja sklearn](#)
- Dane i zeszyt klasteryzacja – możesz go pobrać z sekcji “Dodatkowe materiały do bloku” [tutaj](#).

Uczenie nadzorowane, a nie nadzorowane

Uczenie nienadzorowane:

Charakteryzuje się tym, że nie znamy docelowej "odpowiedzi". Naszym celem nie jest predykcja, żadnej wartości, a analiza "podobieństwa" w danych



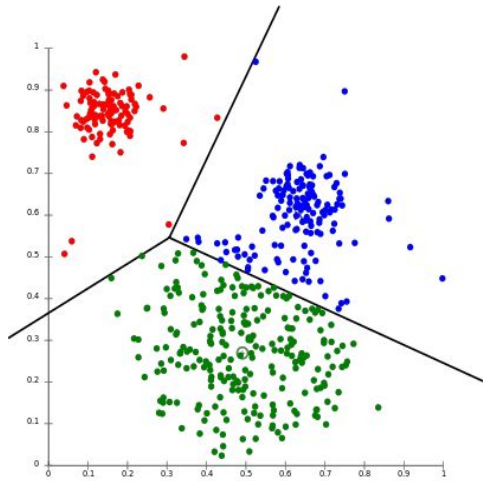


stara się znaleźć homogeniczne grupy wśród obserwowanych danych

pozwała znaleźć niskowymiarową reprezentację obserwowanych danych, która pozwala wyjaśnić dużą część zmienności (wariancji) w danych

Klasteryzacja (grupowanie, analiza skupień)

Cel: przypisanie podobnych w pewien sposób obiektów do jednej grupy, podczas gdy obiekty znacząco różniące się powinny zostać przypisane do innych grup. Powstałe grupy nazywamy **klastrami**.



sample



Cluster/group



Zastosowanie



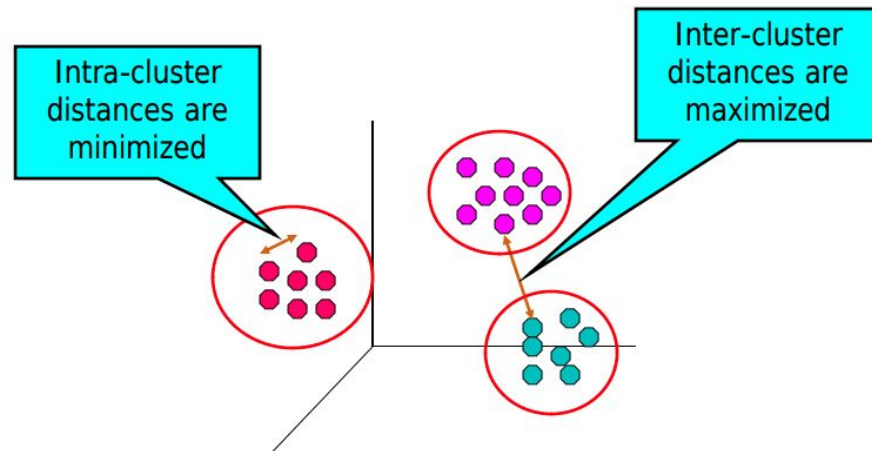
- Marketing – rozpoznawanie różnych grup klientów, ich preferencji, i wykorzystywanie tej wiedzy w celu stworzenia nowych programów marketingowych (np. analiza koszykowa)
- Ubezpieczenia – tworzenie nowych form ubezpieczeń oraz ocenianie średniego kosztu roszczeń dla zidentyfikowanych grup
- Ekonomia – Analiza rynku – rozpoznawanie grup producentów/firm
- Klasyfikacja dokumentów – Znajdowanie podobieństw wśród dokumentów

Metryki klasteryzacji - omówienie



Co to znaczy, że klasteryzacja jest dobra?

- Wysokie wewnątrz klastrowe podobieństwo
- Niskie zewnątrz klastrowe podobieństwo
- Zdolna do zauważania niektórych, lub wszystkich ukrytych wzorców





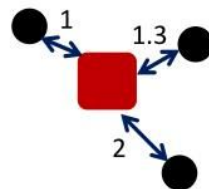
W jaki sposób ocenić klasteryzację?

- Miary wewnętrzne:
 - Wewnętrzna klastrowa suma odległości (SSE)
 - Silhouette
 - Davies-Bouldin Index
- Miary zewnętrzne:
 - Purity

Wewnątrz klastrowa suma odległości (SSE)

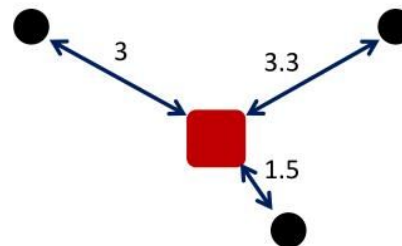
Suma kwadratów odległości obserwacji wewnątrz klastra od jego centrum

Cluster 1



$$\begin{aligned}SSE_1 &= 1^2 + 1.3^2 + 2^2 = \\ &= 1 + 1.69 + 4 = 6.69\end{aligned}$$

Cluster 2



$$\begin{aligned}SSE_2 &= 3^2 + 3.3^2 + 1.5^2 = 9 + \\ &= 10.89 + 2.25 = 22.14\end{aligned}$$

Wewnątrz klastrowa suma odległości (SSE)

Dzięki temu możemy “oceniać” każdy klaster osobno. Jednakże sumaryczną oceną jakości klasteryzacji będzie suma poszczególnych odległości dla każdego z klastrów:

Gdzie:

K – ilość klastrów

μ – reprezentant klastra i (centroid)

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} (\mu_i - x)^2$$



Wewnątrz klastrowa suma odległości (SSE)

Wadą wewnątrz klastrowej sumy odległości (SSE) jest niewątpliwie **brak uwzględniania zewnątrz klastrowego podobieństwa**, brane pod uwagę jest jedynie wewnątrz klastrowe podobieństwo.



Zaimplementujmy to razem!



- Załóżmy, że nasze dane będą dwuwymiarowe
- Jako wejście, będziemy otrzymywać listę klastrów, wraz z punktami, które do nich przynależą tj.
 - input - cluster [cluster1, cluster2, cluster3 ...]
 - clusterX - [[x1, x2, x3 ...], [y1, y2, y3 ...]]



Rozwiązanie



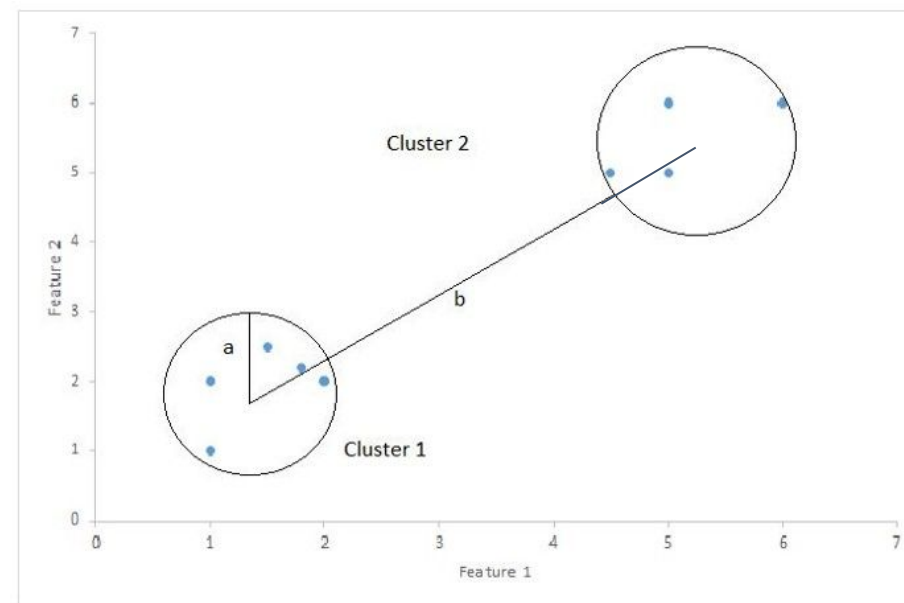
```
1 def sse(clusters):
2     result = 0
3     for xs, ys in clusters:
4         distance = calculate_distance(xs, ys)
5         result += distance / len(xs)
6     return result
7
8 def calculate_distance(xs, ys):
9     result = 0
10    for x1, y1 in zip(xs, ys):
11        for x2, y2 in zip(xs, ys):
12            result += np.sqrt((x1 - x2)**2 + (y1 - y2)**2)
13    return result
```

Silhouette

Silhouette wykorzystuje w swej ocenie średnią odległość pomiędzy obserwacjami wewnątrz grupy **(a)** i średnią odległość obserwacji do najbliższej „obcej” grupy **(b)**.

Wartość Silhouette dla pojedynczej instancji i :

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$





Silhouette

$a(i)$ – reprezentuje średni dystans punktu do wszystkich innych punktów znajdujących się w klastrze C. Duża wartość 'a' oznacza, że instancja jest niepodobna do innych instancji znajdujących się w klastrze:

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j)$$



Silhouette

$b(i)$ – reprezentuje średni dystans do wszystkich punktów w następnym **najbliższym** klastrze. Duża wartość **$b(i)$** implikuje, że instancja jest niepodobna do pobliskiego klastra.

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j)$$



Silhouette

Obliczanie **$b(i)$** składa się z dwóch kroków:

1. Znajdź średni dystans instancji (i) do innego klastra.
2. Weź wartość minimalną z tych średnich dystansów.

Dla przykładu: Jeżeli punkt należy do klastra 0, to znajdź **średnią** odległość pomiędzy punktem, a wszystkimi punktami klastra 1, 2, 3, a następnie zwróć minimum tych odległości (najmniejszą średnią odległość)



Silhouette

Warto zanotować:

- $-1 \leq s(i) \leq 1$
- Jeżeli $s(i) \approx 1$, to $a(i) \ll b(i)$ punkt znajduje się w dobrym klastrze jest zdecydowanie bardziej podobny do klastra, do którego należy niż do sąsiadującego klastra.
- Jeżeli $s(i) \approx -1$, to $a(i) \gg b(i)$ punkt znajduje się w złym klastrze jest zdecydowanie bardziej podobny do sąsiadującego klastra, niż do innych instancji swojego klastra.
- $s(i) = 0$ jeżeli $|C_i| = 1$ (to jest silhouette punktu w klastrze zawierającym tylko jedną instancję (właśnie ten punkt) wynosi 0
- Jeżeli $s(i) \approx 0$, możliwe, że punkt powinien należeć w innym klastrze



Silhouette

Ostatecznie Silhouette dla danego modelu określa się jako średnią $s(i)$ z wszystkich instancji

$$S = \frac{1}{N} \sum_i s_i$$

gdzie N – ilość instancji

$s(i)$ – wartość silhouette dla pojedynczego punktu



Zaimplementujmy to razem!



- Założmy, że nasze dane będą dwuwymiarowe
- Jako wejście, będziemy otrzymywać dict klastrów, wraz z punktami, które do nich przynależą tj.
 - input - cluster {1:cluster1, 2:cluster2, 3:cluster3 ...}
 - clusterX - [[x1, x2, x3 ...], [y1, y2, y3 ...]]

Silhouette – własna implementacja

```
1 def dist_to_cluster(point, cluster):
2     x_i, y_i = point
3     xs, ys = cluster
4     distances = 0
5     for x_j, y_j in zip(xs, ys):
6         distances += np.sqrt((x_j - x_i)**2 + (y_j - y_i)**2)
7     return distances
8
9 def a(point, cluster):
10     distances = dist_to_cluster(point, cluster)
11     return 1 / (len(cluster[0]) - 1) * distances
12
13 def b(point, other_clusters):
14     x_i, y_i = point
15     result = [1 / len(cluster[0]) * dist_to_cluster(point, cluster)
16               for cluster
17               in other_clusters]
18     return min(result)
19
20 def silhouette(point, cluster, other_clusters):
21     if len(cluster) == 1:
22         return 0
23     else:
24         ar = a(point, cluster)
25         br = b(point, other_clusters)
26         return (br - ar) / max(ar, br)
27
28 def global_silhouette(clusters):
29     if len(clusters) == 1:
30         return 0
31     results = []
32     for key, cluster in clusters.items():
33         other_clusters = [c for k, c in clusters.items() if k != key]
34         for point in zip(*cluster):
35             s = silhouette(point, cluster, other_clusters)
36             results.append(s)
37     return np.mean(results)
```



Davies-Bouldin Index

Davies-Bouldin, podobnie jak Silhouette bazuje na stosunku pomiędzy podobieństwem wewnątrz klastrowym i zewnątrz klastrowym.

$$DB(\mathcal{C}) = \frac{1}{k} \sum_{i=1}^k \max_{j \leq k, j \neq i} D_{ij}, \quad k = |\mathcal{C}|,$$



Davies-Bouldin Index



D_{ij} to “**within-to-between cluster distance ratio**” dla i-tego i j-tego klastra

$$D_{ij} = \frac{(\bar{d}_i + \bar{d}_j)}{d_{ij}},$$

Gdzie:

- \bar{d}_i jest to średni dystans pomiędzy każdym punktem w klastrze a jego centroidem
- \bar{d}_j to samo co \bar{d}_i tylko dla drugiego klastra
- d_{ij} to natomiast dystans pomiędzy klastrami

Davies-Bouldin Index

D_{ij} to “within-to-between cluster distance ratio” dla i-tego i j-tego klastra

$$D_{ij} = \frac{(\bar{d}_i + \bar{d}_j)}{d_{ij}},$$

Jeżeli 2 klastry są blisko siebie (małe d_{ij}), ale mają dużą rozpiętość (rozstrzał) – duże \bar{d}_i, \bar{d}_j to D_{ij} będzie duże, co może oznaczać, że te dwa klastry nie są od siebie zbytnio różne i może warto jest ze sobą **połączyć**.

Davies-Bouldin Index

$$DB(\mathcal{C}) = \frac{1}{k} \sum_{i=1}^k \max_{j \leq k, j \neq i} D_{ij}, \quad k = |\mathcal{C}|,$$

$\max D_{ij}$ oznacza "**worst-case within-to-between cluster ratio**" czyli wartość maksymalną D_{ij} dla i -tego klastra

- im mniejsze DB, tym lepiej



Porównanie

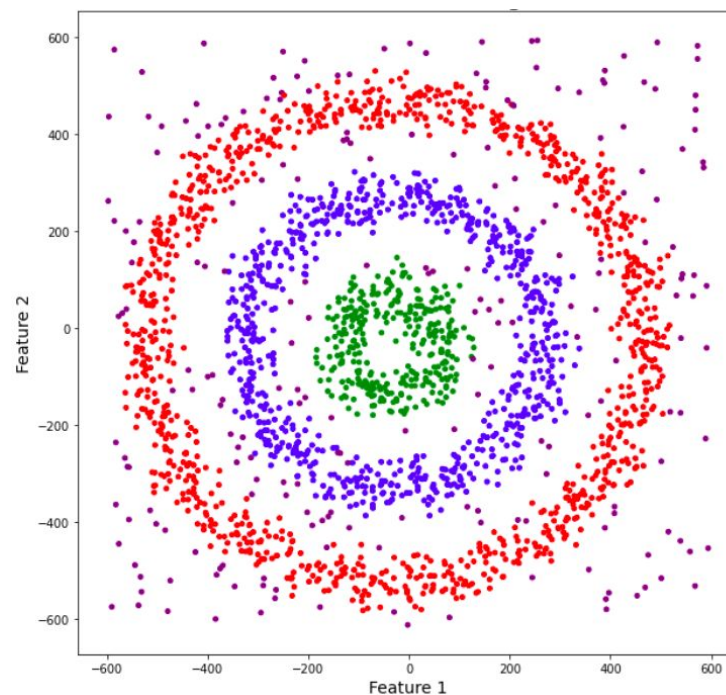


- Silhouette jest najbardziej wymagające obliczeniowo
- Silhouette bierze pod uwagę największą ilość informacji
- SSE nie bierze pod uwagę w ogóle stosunku klastrów pomiędzy sobą
- DB wydaje się być ciekawym tradeoffem pomiędzy SSE, a Silhouette

Podobnie jak w przypadku klasyfikacji nie ma czegoś takiego jak najlepsza i uniwersalna miara. Warto przeanalizować kilka miar i wyciągnąć z nich wnioski.

Pomyśl!

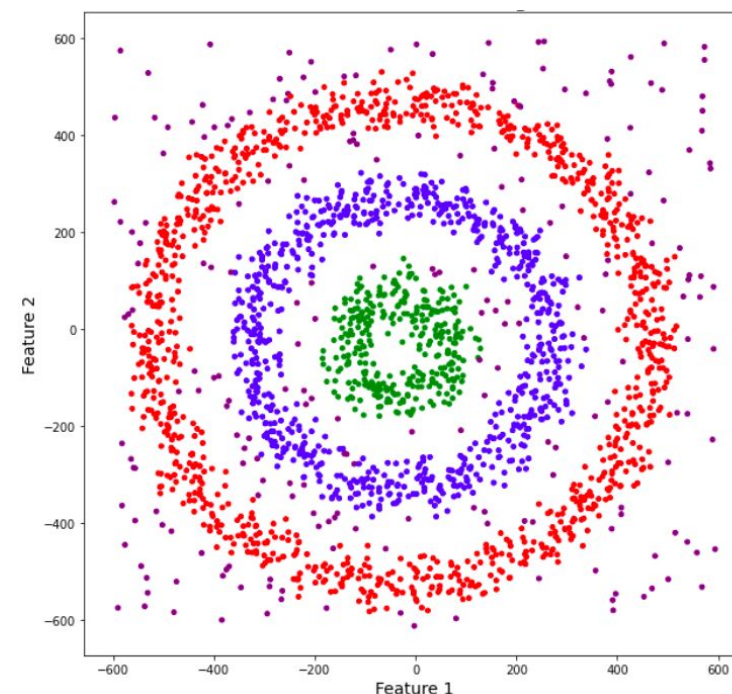
Jak poradzą sobie wprowadzone miary w poniższym przypadku?



Metody zewnętrzne

Wadą miar wewnętrznych jest fakt, że zupełnie nie radzą sobie z problemami, jak przedstawione na poniższej grafice.

W związku z tym pozostaje użycie jedynie miar zewnętrznych. Wymagają one niestety posiadania ground truth naszych danych.





Purity

Jest to procentowa wartość obiektów (instancji), które zostały zaklasyfikowane poprawnie. Przedstawia się ją w zakresie od 0 do 1.

$$Purity = \frac{1}{N} \sum_{i=1}^k \max_j |c_i \cap t_j|$$

Gdzie:

N – ilość wszystkich instancji

k – ilość klastrów

c_i – jest to jeden z klastrów

t_j – są to ilości danej klasy obiektów w klastrze

Elementy, które występują najczęściej są brane pod uwagę.



Purity

Kiedy mówimy “poprawnie zaklasyfikowane” oznacza to, że każdy klaster identyfikuje grupę obiektów tej samej klasy co w ground truth.

W celu obliczenia Purity możemy wykorzystać confusion matrix.

Przykład:

	T1	T2	T3
C1	10	53	10
C2	10	1	60
C3	10	16	0

$$\text{Purity} = (53 + 60 + 16) / 170 = 0.76$$



Purity

Cały proces przypomina poznane wcześniej accuracy, z tym zaznaczeniem, że klaster 1 nie musi odpowiadać pierwszej klasie. Określamy, że klaster reprezentuje daną klasę, jeżeli dana klasa jest klasą przeważającą w danym klastrze (stąd max we wzorze)

Warto również zauważyć, że Purity (czystość), nie uwzględnia tego, czy każda klasa jest w jakiś sposób reprezentowana. Sprawdza jedynie **jednorodność** poszczególnych klastrów.



Podsumowanie

Czy to wszystkie dostępne miary?

Oczywiście nie!

Ale z pewnością te najbardziej popularne.

Metryki klasteryzacji – Jak wybrać ilość klastrów?





Ilość klastrów

Wspomniane wcześniej metryki mogą posłużyć do:

- wyboru parametrów algorytmu
- wyboru algorytmu
- wyboru ilości klastrów

Ostatnie z tych, jest o tyle istotne, że większość algorytmów wymaga zdefiniowania z góry ilości poszukiwanych klastrów. Tym samym wymagane jest określenie ich optymalnej liczby.



Zaimplementujmy to wspólnie!



Inicjalizacja środowiska testowego:

1. Stwórz losowy rozkład z wyraźnie 3 różnymi grupami
2. Zwizualizuj utworzone dane
3. Przeprowadź dummy klasteryzację
4. Stwórz wizualizację klasteryzacji

Rozwiązanie:

Stwórz losowy rozkład z wyraźnie 3 różnymi grupami

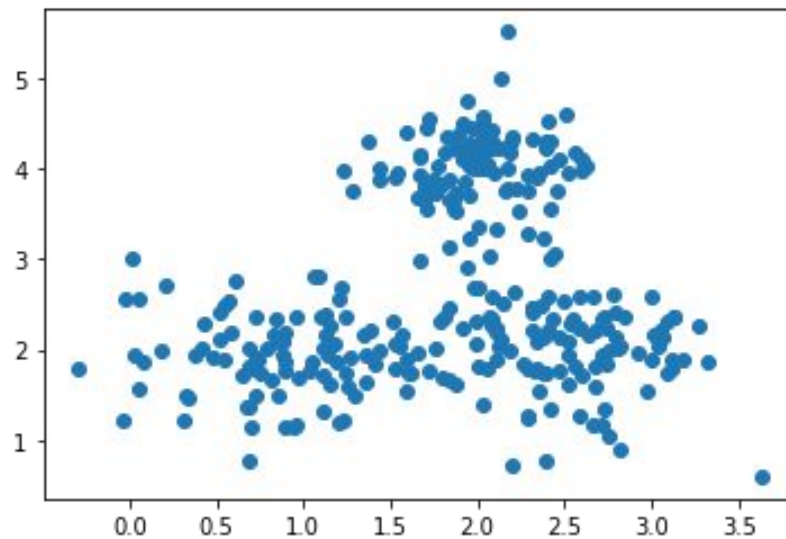
```
[ ] 1 import numpy as np
    2 from numpy.random import RandomState
    3 import matplotlib.pyplot as plt

[ ] 1 rnd = RandomState(25)
    2 x1 = sorted(list(rnd.normal(1.0, 0.5, size=100)))
    3 y1 = list(rnd.normal(2.0, 0.4, size=100))
    4 x2 = sorted(list(rnd.normal(2.0, 0.3, size=100)))
    5 y2 = list(rnd.normal(4.0, 0.4, size=100))
    6 x3 = sorted(list(rnd.normal(2.5, 0.4, size=100)))
    7 y3 = list(rnd.normal(2.0, 0.5, size=100))
```

Rozwiązanie:

Zwizualizuj utworzone dane

```
[ ] 1 fig, ax = plt.subplots()  
    2 ax.scatter(x1+x2+x3, y1+y2+y3)  
    3 plt.show()
```



Rozwiązanie:

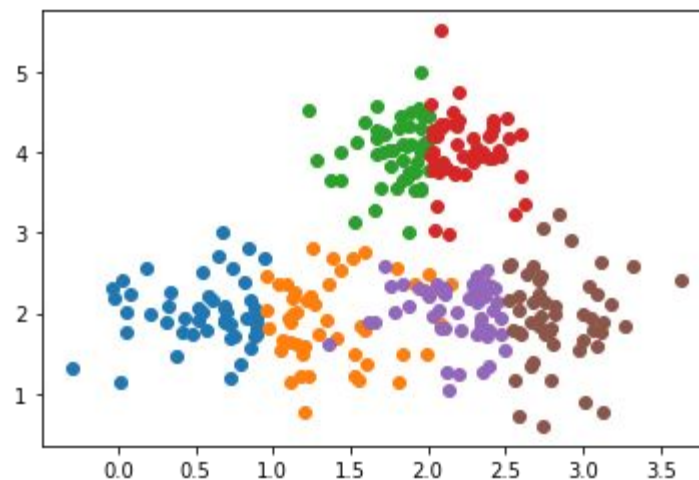
Przeprowadź dummy klasteryzację

```
1 models = []
2 # First situation - 1 cluster:
3 first_model = [(x1+x2+x3, y1+y2+y3)]
4
5 # Second situation - 2 clusters:
6 second_model = [(x1+x3, y1+y3), (x2,y2)]
7
8 # Third situation - 3 clusters:
9 third_model = [(x1, y1), (x2, y2), (x3, y3)]
10
11 # Fourth situation - 4 clusters:
12 fourth_model = [(x1[:int(len(x1)/2)], y1[:int(len(y1)/2)]),
13                 (x1[int(len(x1)/2):], y1[int(len(y1)/2):]),
14                 (x2, y2), (x3, y3)]
15
16 # Fifth situation - 5 clusters:
17 fifth_model = [(x1[:int(len(x1)/2)], y1[:int(len(y1)/2)]),
18                (x1[int(len(x1)/2):], y1[int(len(y1)/2):]),
19                (x2[:int(len(x2)/2)], y2[:int(len(y2)/2)]),
20                (x2[int(len(x2)/2):], y2[int(len(y2)/2):]),
21                (x3, y3)]
22
23 # Sixth situation - 6 clusters:
24 sixth_model = [(x1[:int(len(x1)/2)], y1[:int(len(y1)/2)]),
25                (x1[int(len(x1)/2):], y1[int(len(y1)/2):]),
26                (x2[:int(len(x2)/2)], y2[:int(len(y2)/2)]),
27                (x2[int(len(x2)/2):], y2[int(len(y2)/2):]),
28                (x3[:int(len(x3)/2)], y3[:int(len(y3)/2)]),
29                (x3[int(len(x3)/2):], y3[int(len(y3)/2):])]
30 models = [first_model, second_model, third_model,
31            fourth_model, fifth_model, sixth_model]
```

Rozwiązanie:

Stwórz wizualizację klasteryzacji

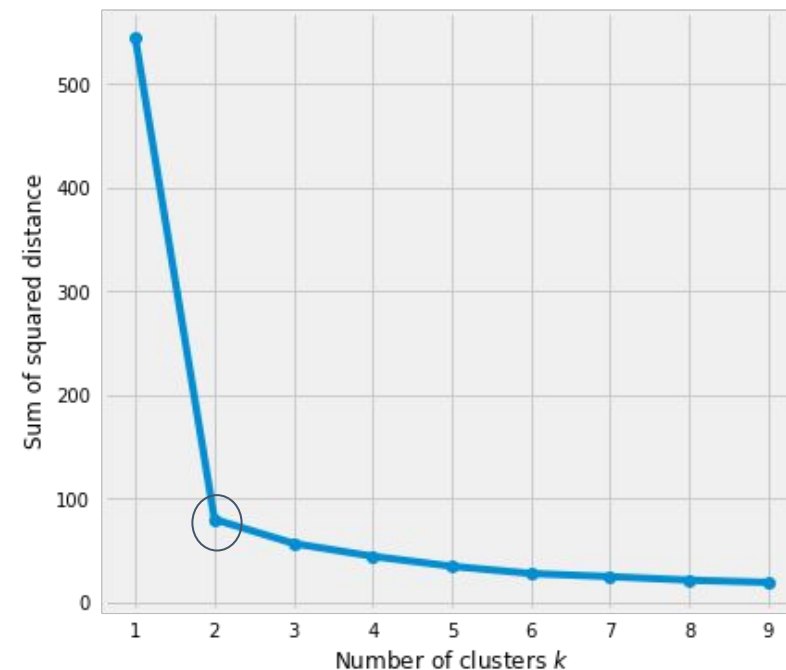
```
1 def show_clusters(clusters):  
2     fig, ax = plt.subplots()  
3     print(len(clusters))  
4     for x,y in clusters:  
5         ax.scatter(x, y)  
6     plt.show()  
  
1 show_clusters(sixth_model)
```



Kryterium „łokcia”

W celu określenia optymalnej ilości klastrow możemy posłużyć się miarą SSE. W tym celu:

- Implementujemy nasz algorytm dla różnej ilości klastrow
- Obliczamy SSE, dla każdego z powstałych rozwiązań
- Przedstawiamy wynik w formie wykresu



Optymalna ilość klastrow znajduje się w “zgięciu” łokcia.



Zaimplementujmy to wspólnie!



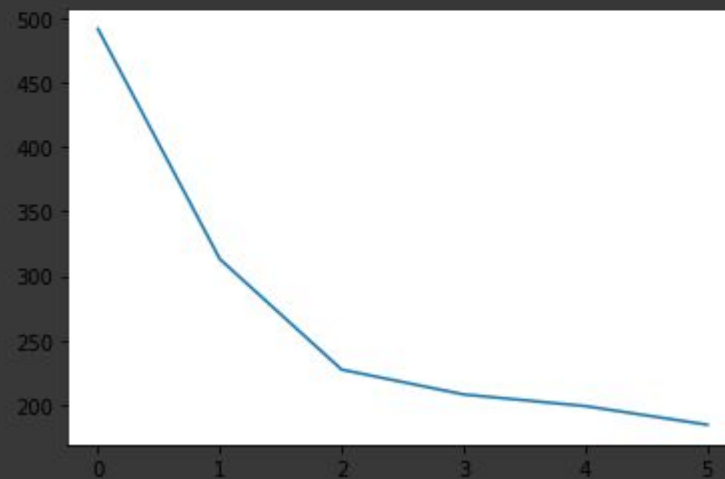
Metoda łokcia

1. Przy użyciu funkcji SSE sprawdź jaka jest optymalna ilość klastrów (na pewno 3?)

Rozwiązanie:

Wykres łokcia:

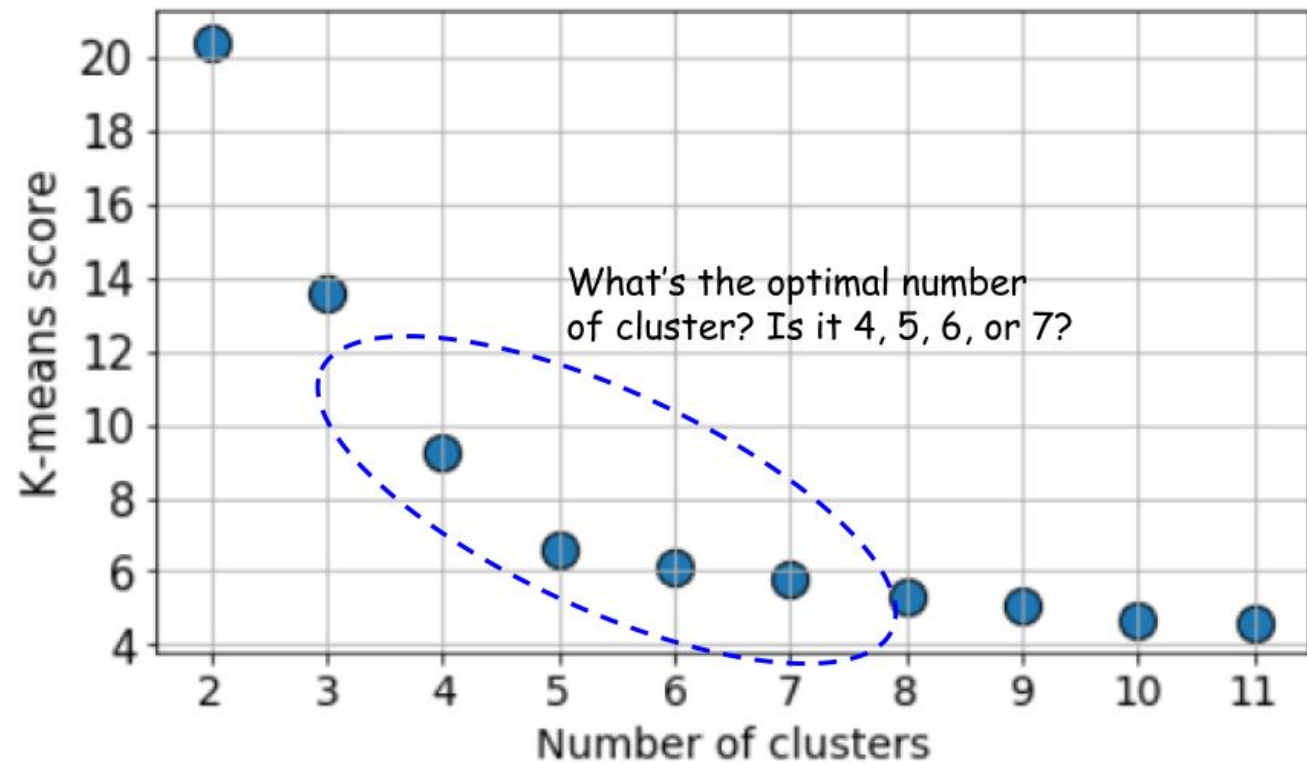
```
1 distances = []  
2 for model in models:  
3     distances.append(sse(model))  
4  
5  
6 plt.plot(list(range(len(distances))), distances)  
7 plt.show()
```



Kryterium „łokcia”

Nie zawsze określenie „łokcia” jest takie proste. Należy wtedy wykorzystać w tym celu inne metody.

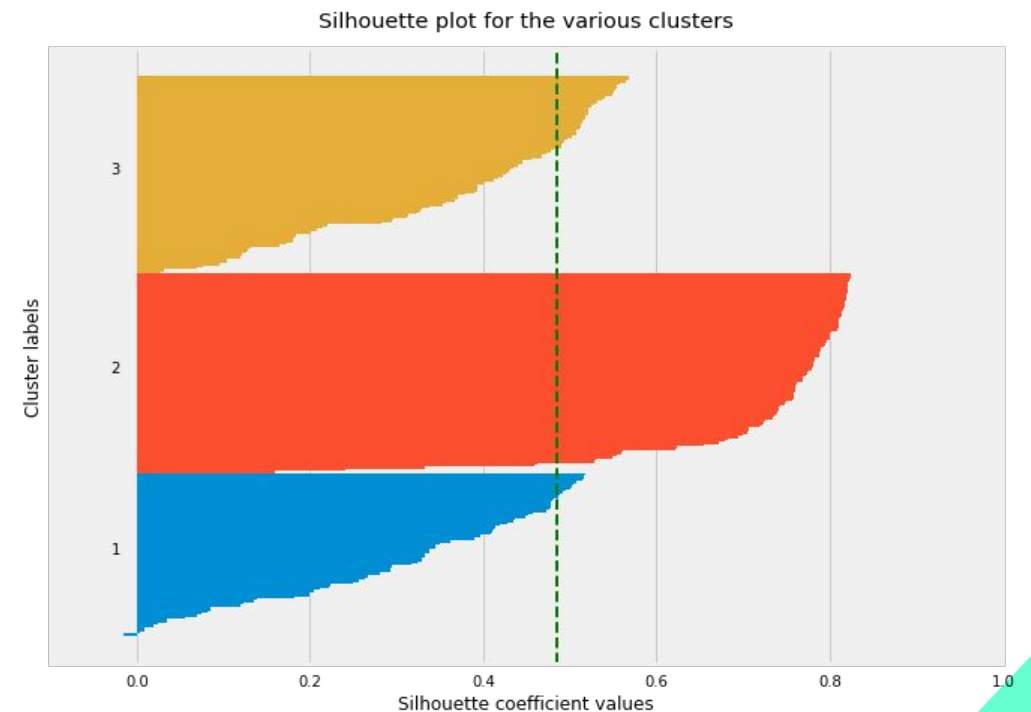
The elbow method for determining number of clusters



Silhouette

Określanie ilości klastrow za pomocą silhouette dokonuje się poprzez analizę wartości $s(i)$ dla poszczególnych klastrow ->

Kolor oznacza konkretny klaster, a pojedynczy "pasek" jest pojedynczą wartością $s(i)$

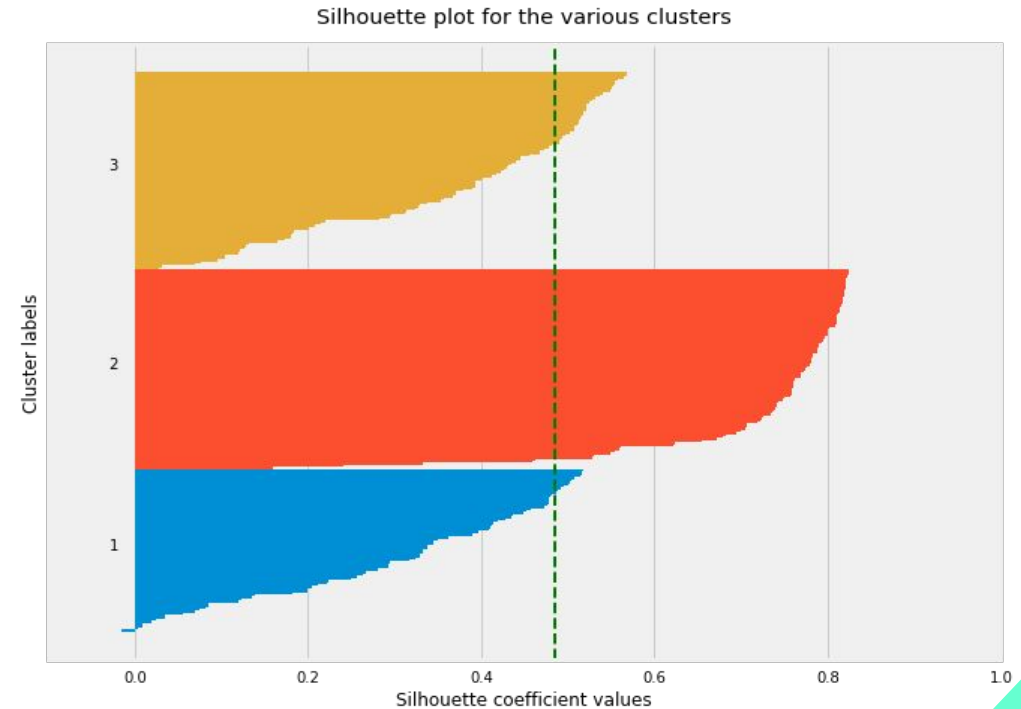


Silhouette

Należy przeanalizować wykresy tego typu dla każdej k liczby klas w poszukiwaniu:

- Możliwie dużej wartości średniej
- Braku klastrów, których maksymalna wartość $s(i)$ nie przekracza wartości średniej -> oznacza to złą ilość k

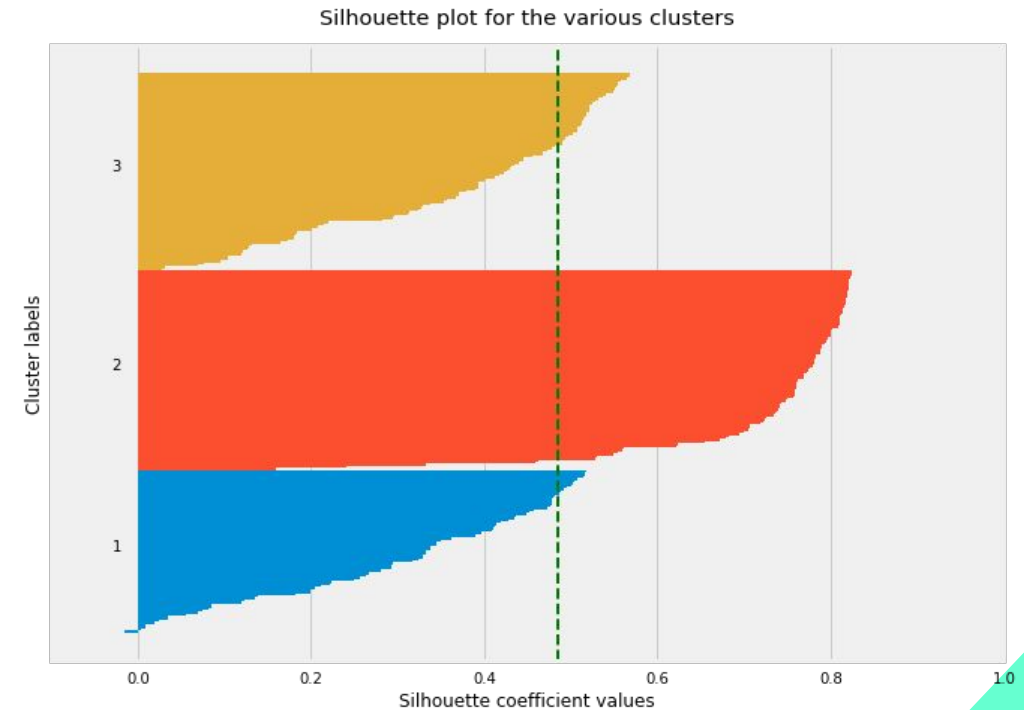
Dobra klasteryzacja będzie miała podobne charakterystyki dla każdego klastra, a także punkty będą podobne do siebie.



Silhouette

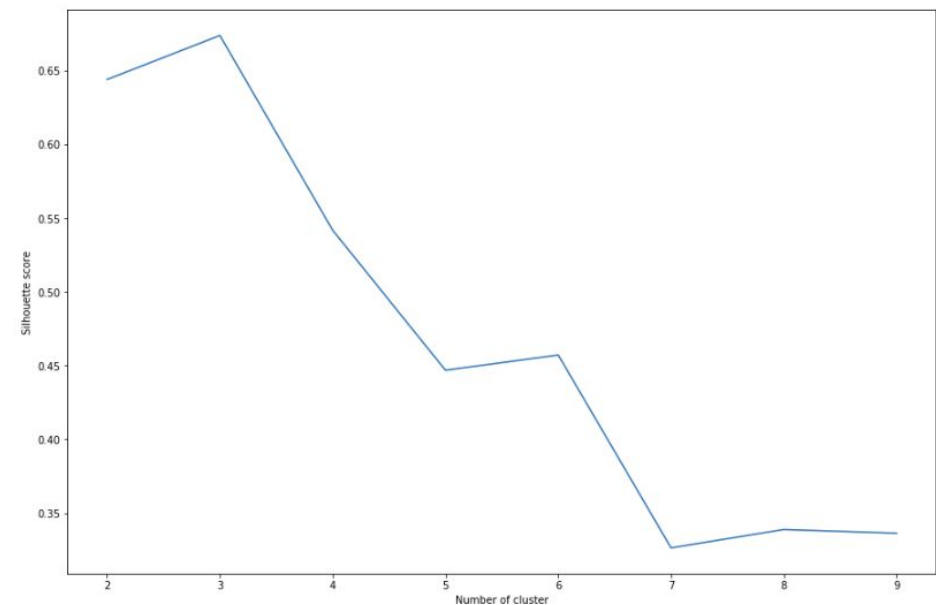
Przykładowo klasteryzacja pokazana po prawej:

- Ma raczej poprawną ilość klastrów (wartość maksymalna powyżej średniej)
- Klaster 2 jest z pewnością dobrze zgrupowany



Silhouette

- Oczywiście nic nie stoi na przeszkodzie, żeby również przyjrzeć się zbiorczej wartości silhouette
- W tym przypadku będziemy poszukiwać wartości maksymalnej





Przykład!

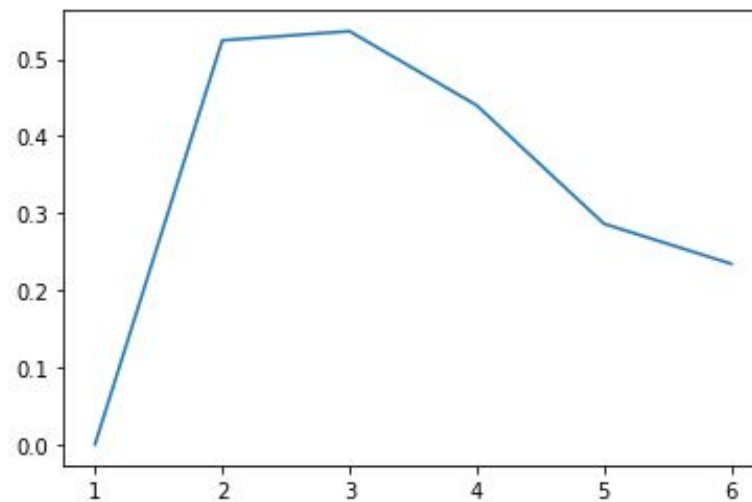
Przeanalizuj wcześniej utworzony zbiór za pomocą silhouette

1. Oceń za pomocą silhouette, czy faktycznie 3 klastry będą najlepszym pomysłem.

Rozwiązanie

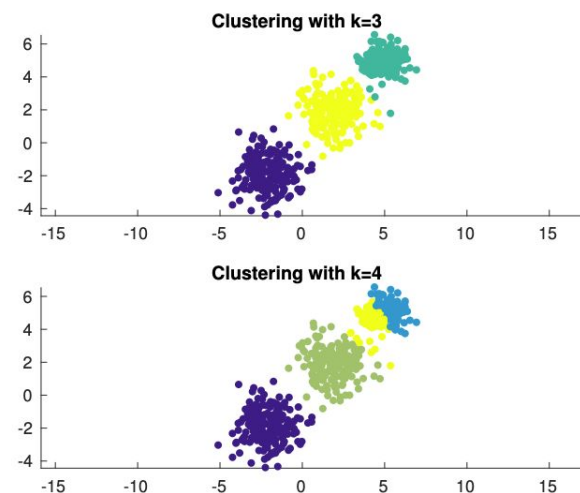
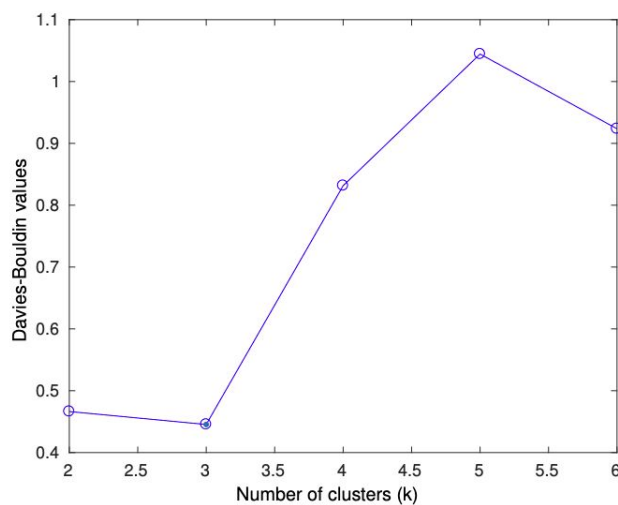
I sprawdzimy, jak wygląda wykres tej miary:

```
def global_silhouette(model):  
    if len(model) == 1:  
        return 0  
    results = []  
    for key, cluster in model.items():  
        other_clusters = [c for k, c in model.items() if k != key]  
        for point in zip(*cluster):  
            s = silhouette(point, cluster, other_clusters)  
            results.append(s)  
    return np.mean(results)  
  
sil = []  
for model in models:  
    to_dict = {}  
    for index, cluster in enumerate(model):  
        to_dict[index] = cluster  
  
    returned = global_silhouette(to_dict)  
    sil.append(returned)  
  
plt.plot(list(range(1, len(sil)+1)), sil)  
plt.show()
```



Davies–Bouldin index

Oceny optymalnej ilości klastrow za pomocą Davies–Bouldin index możemy dokonać przez analizę wykresu, tej wartości dla poszczególnej ilości klastrow. **Należy tutaj poszukiwać minimum.**



Podobieństwo instancji





Podobieństwo dwóch instancji

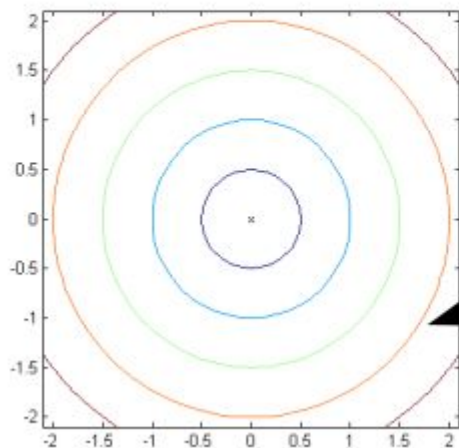
Najprościej – odległość pomiędzy nimi w n wymiarowej przestrzeni
gdzie n – ilość featurów/cech, które opisują instancję.

Przypomnienie z klasyfikacji!

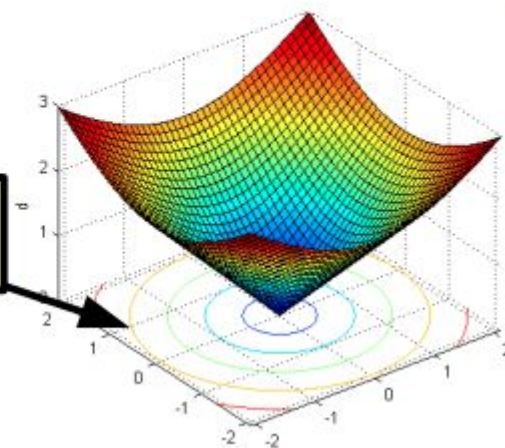
Euclidean

- **Odległość Euklidesa:**

$$d(a, b) = \left((a - b)^T (a - b) \right)^{1/2} = \sqrt{\sum_{i=1}^P (a_i - b_i)^2}$$



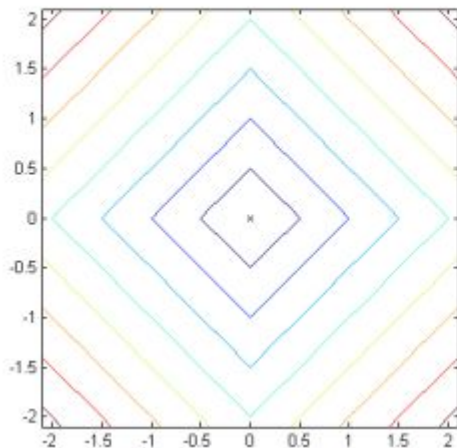
Krzywe równej odległości od punktu o współrzędnych (0,0)



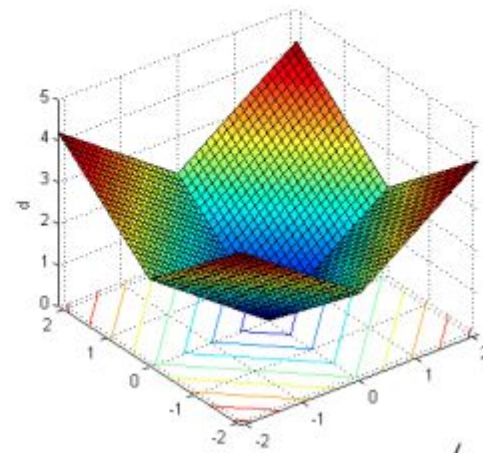
P

Manhattan

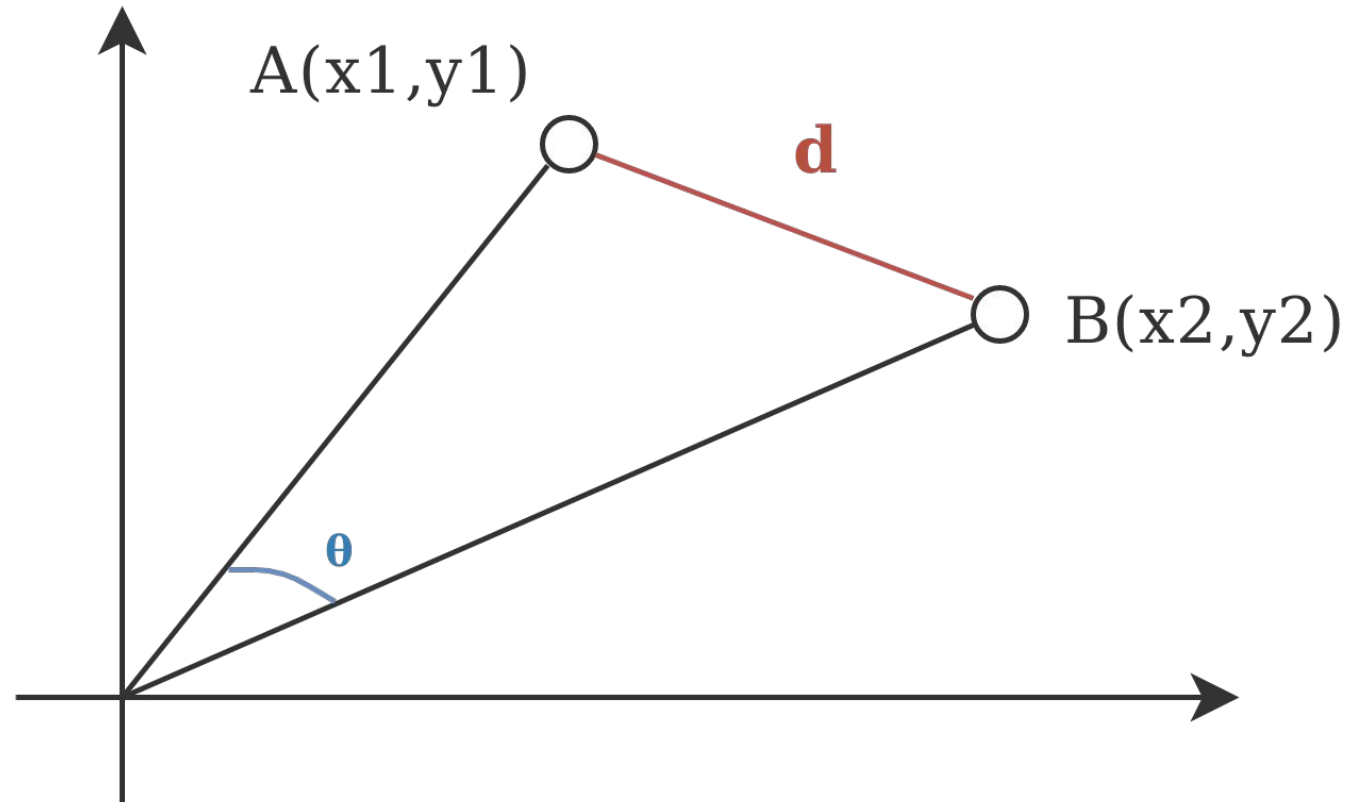
- **Odległość miejska (*manhattan, city-block*):**



$$d(a, b) = \sum_{i=1}^P |a_i - b_i|$$



Cosine



Klasteryzacja





Rodzaje klasteryzacji



- Connectivity-based (hierarchiczna)
- Centroid-based (K-means)
- Distribution-based (Gaussowski model mieszany)
- Density-based clustering (DBSCAN)

K-means





K-means – klasteryzacja niehierarchiczna



K-means – klastry formowane są poprzez przypisanie punktu do najbliższej centroidy (środka ciężkości klastra).

Algorytm:

1. Centroidy wybierane są w określony sposób (losowo, k-means ++)
2. Dystans pomiędzy punktami a centroidami jest obliczany. Punkt należy do najbliższego centroidy i jego klastra.
3. Po przypisaniu elementów do klastrów następuje wyznaczenie nowej centroidy, poprzez obliczenie "średniej" instancji przypisanej do danego klastra.
4. Kroki 2-3 powtarzamy tak długo, aż centroidy pomiędzy dwoma kolejnymi krokami nie zmienią się.




K-means – klasteryzacja niehierarchiczna



Zalety

- + prosty, efektywny
- + może być używany do różnych typów danych

Wady

- podatność na outliery
 - nie zawsze centroida dobrze reprezentuje klaster
 - nie radzi sobie z obsługą niesferycznych klastrów o różnych gęstościach i rozmiarach
- 

Przykład

Najpierw pobierzmy niewielki zbiór punktów:

```
!git clone https://github.com/matzim95/ML-datasets
```

```
%matplotlib inline
```

```
import pandas as pd
```

```
points = pd.read_csv('ML-datasets/points.csv', header = None)
```

```
points.head()
```

```
Cloning into 'ML-datasets'...
```

```
remote: Enumerating objects: 71, done.
```

```
remote: Counting objects: 100% (71/71), done.
```

```
remote: Compressing objects: 100% (59/59), done.
```

```
remote: Total 71 (delta 16), reused 59 (delta 12), pack-reused 0
```

```
Unpacking objects: 100% (71/71), done.
```

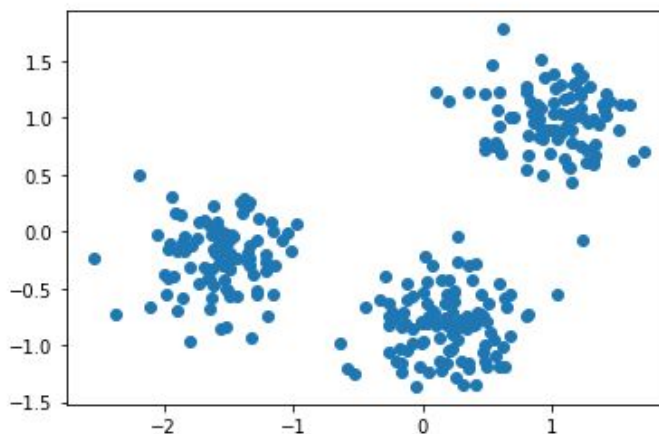
	0	1
0	0.065446	-0.768664
1	-1.529015	-0.429531
2	1.709934	0.698853
3	1.167791	1.012626
4	-1.801101	-0.318613

Rozwiązanie

Sprawdźmy ręcznie, na ile klastrów powinniśmy podzielić zbiór:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
xs = points.loc[:,0]
ys = points.loc[:,1]
ax.scatter(xs, ys)
plt.show()
```



wychodzi, że na 3 ;)

Rozwiązanie

A zatem, bierzmy się do roboty! I zobaczmy jak to wygląda:

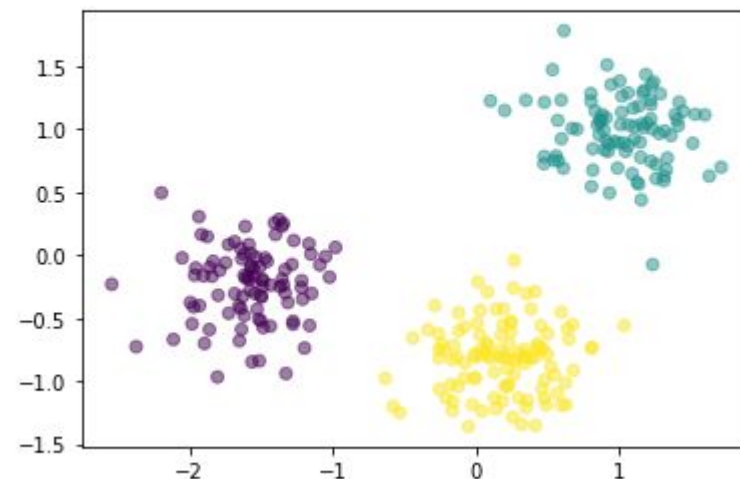
```
# Import KMeans
from sklearn.cluster import KMeans

# Create a KMeans instance with 3 clusters: model
model = KMeans(n_clusters=3)

# Fit model to points
model.fit(points)

labels = model.predict(points)
```

```
plt.scatter(xs, ys, c=labels, alpha=0.5)
plt.show()
```



Rozwiązanie

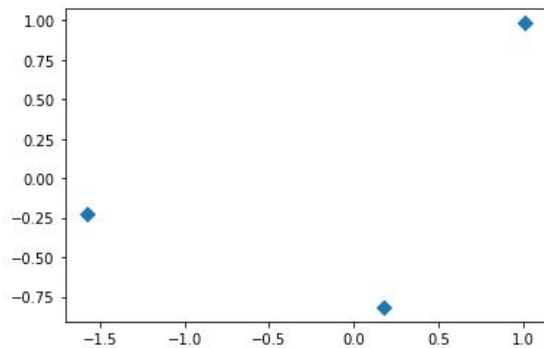
Pokażmy również centroidy naszych klastrów:

```
[18] # Assign the cluster centers: centroids
centroids = model.cluster_centers_

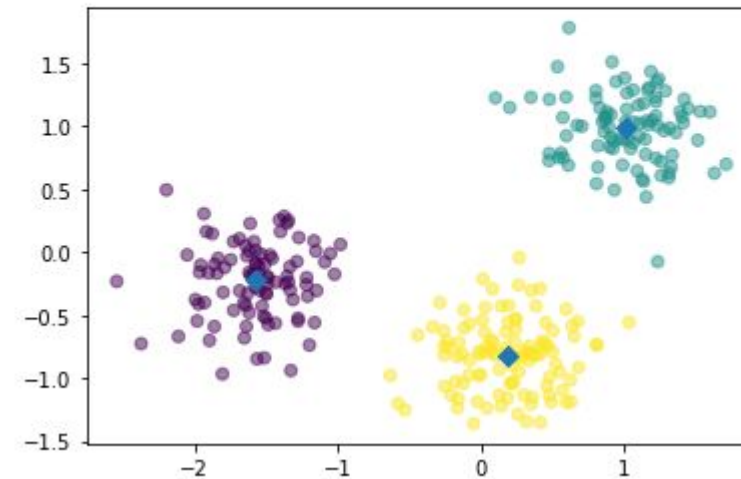
# Assign the columns of centroids: centroids_x, centroids_y
centroids_x = centroids[:,0]
centroids_y = centroids[:,1]

fig, ax = plt.subplots()

# Make a scatter plot of centroids_x and centroids_y
ax.scatter(centroids_x, centroids_y, marker='D', s=50)
plt.show()
```



```
fig, ax = plt.subplots()
ax.scatter(xs, ys, c=labels, alpha=0.5)
ax.scatter(centroids_x, centroids_y, marker='D', s=50)
plt.show()
```



K-means – pomiar optymalnej liczby klastrów

Najbardziej zależy nam na tym, żeby odległości punktów od środków klastrów były jak najmniejsze (klastry były skonsolidowane). Do tego służy wbudowana miara `inertias`. Pokazuje nam ona sumy odległości wszystkich punktów od ich centroidów. Jak widać, miara jest już dość mała dla 3 klastrów. Oczywiście dalej się zmniejsza, ale zależy nam żeby klastrów nie było też za dużo. Należy tutaj znaleźć pewien złoty środek.

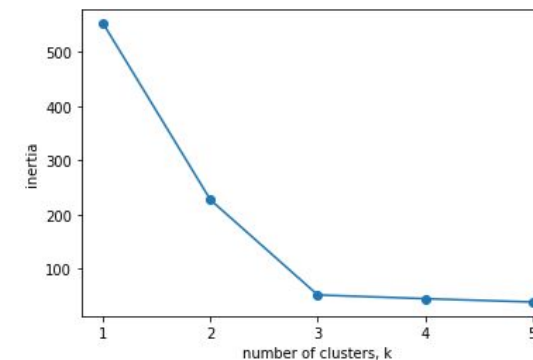
```
ks = range(1, 6)
inertias = []

for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(points)

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

# Plot ks vs inertias
fig, ax = plt.subplots()
ax.plot(ks, inertias, '-o')
ax.set_xlabel('number of clusters, k')
ax.set_ylabel('inertia')
ax.set_xticks(ks)
plt.show()
```



A teraz spróbujmy dla danych zaetykietowanych

```
wine, wine_classes = load_dataset('wine', 'Class')
```

```
y = wine.pop('class')
```

```
X = wine
```

```
X.head()
```

	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

Porównajmy wyniki modelu z prawdziwą wartością

Wyniki są, delikatnie mówiąc, kiepskie. Wynika to z faktu, że poszczególne atrybuty w zbiorze win mają bardzo różną wariancję (rozmieszczenie atrybutów). Co bardzo przeszkadza w dobrej klasteryzacji.

```
# Create a KMeans model with 3 clusters: model
model = KMeans(n_clusters=3)

# Use fit_predict to fit model and obtain cluster labels: labels
labels = model.fit_predict(X)

# Create a DataFrame with labels and varieties as columns: df
df = pd.DataFrame({'labels': labels, 'varieties': y})

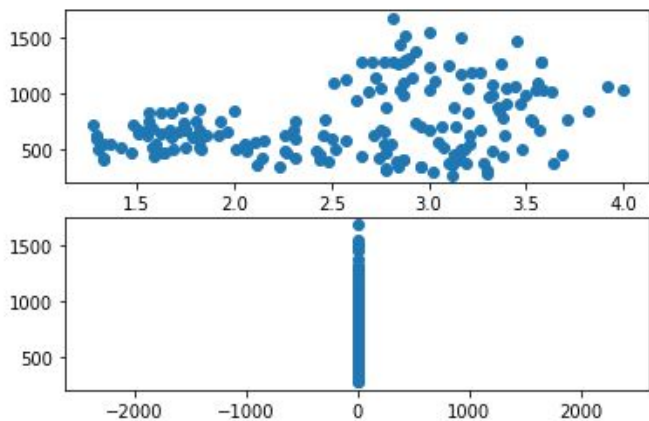
# Create crosstab: ct
ct = pd.crosstab(df['labels'], df['varieties'])

# Display ct
print(ct)
```

varieties	0	1	2
labels			
0	46	0	1
1	13	29	20
2	0	19	50

Problem: zróżnicowana wariancja atrybutów

```
fig, axs = plt.subplots(2, 1)
axs[0].scatter(X['OD280/OD315 of diluted wines'], X['Proline'])
axs[1].axis("equal")
axs[1].scatter(X['OD280/OD315 of diluted wines'], X['Proline'])
plt.show()
```



```
for column in X:
    print(X[column].var())
```

```
0.6590623278105763
1.2480154034152227
0.07526463530756043
11.152686155018094
203.9893353646925
0.3916895353266042
0.9977186726337837
0.015488633911001082
0.3275946676823461
5.374449383491404
0.05224496070589728
0.5040864089379803
99166.71735542428
```

Rozwiązanie: standaryzacja

```
from sklearn.preprocessing import StandardScaler

# Create scaler: scaler
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

labels = model.fit_predict(X_scaled)
```

Korzystamy z wbudowanej biblioteki. Wyniki są od razu dużo lepsze.

```
# Create a DataFrame with labels and varieties as columns: df
df = pd.DataFrame({'labels': labels, 'varieties': y})

# Create crosstab: ct
ct = pd.crosstab(df['labels'], df['varieties'])

# Display ct
print(ct)
```

varieties	0	1	2
labels			
0	0	48	3
1	59	0	3
2	0	0	65



Pipeline



Można również zrobić gotowy pipeline (ścieżkę, tak aby preprocesing i klasteryzacja następowały automatycznie po sobie):

```
from sklearn.pipeline import make_pipeline

pipeline = make_pipeline(scaler, model)
pipeline.fit(X)
labels = pipeline.predict(X)
```


Agglomerative Hierarchical Clustering





Klasteryzacja hierarchiczna - AHC

Agglomerative Hierarchical Clustering

1. Każdy z klastrów reprezentowany jest przez jeden punkt (liście).
2. Najbliższe pary klastrów są ze sobą łączone.
3. Operacja jest powtarzana aż do uzyskania jednego klastra (korzeń).

Podobieństwo pomiędzy klastrami może być wyznaczane w różny sposób (np. MIN, MAX, średnia dla grupy). Wybór metody podobieństwa odróżnia od siebie różne podejścia aglomeracyjne.



Klasteryzacja hierarchiczna – AHC

Zalety

- + obsługa klastrów o różnej długości
- + pozwala na opis od ogółu do szczegółu, taksonomia

Wady

- brak globalnej funkcji celu
- kiepski dla danych wielowymiarowych lub gdy występuje dużo szumu
- wysokie wymagania obliczeniowe

Przykład

Do AHC użyjemy zbioru Eurowizji 2016, w którym za wiersze służą kraje, a za kolumny ilości punktów, które przyznały poszczególnym piosenkom. Pobierzmy go ze zbioru.

```
eurovision = pd.read_csv('ML-datasets/eurovision-2016.csv')
eurovision = eurovision.pivot(index='From country', columns='To country')['Jury Rank']
eurovision = eurovision.fillna(0)
eurovision.index

Index(['Albania', 'Armenia', 'Australia', 'Austria', 'Azerbaijan', 'Belarus',
      'Belgium', 'Bosnia & Herzegovina', 'Bulgaria', 'Croatia', 'Cyprus',
      'Czech Republic', 'Denmark', 'Estonia', 'F.Y.R. Macedonia', 'Finland',
      'France', 'Georgia', 'Germany', 'Greece', 'Hungary', 'Iceland',
      'Ireland', 'Israel', 'Italy', 'Latvia', 'Lithuania', 'Malta', 'Moldova',
      'Montenegro', 'Norway', 'Poland', 'Russia', 'San Marino', 'Serbia',
      'Slovenia', 'Spain', 'Sweden', 'Switzerland', 'The Netherlands',
      'Ukraine', 'United Kingdom'],
      dtype='object', name='From country')
```

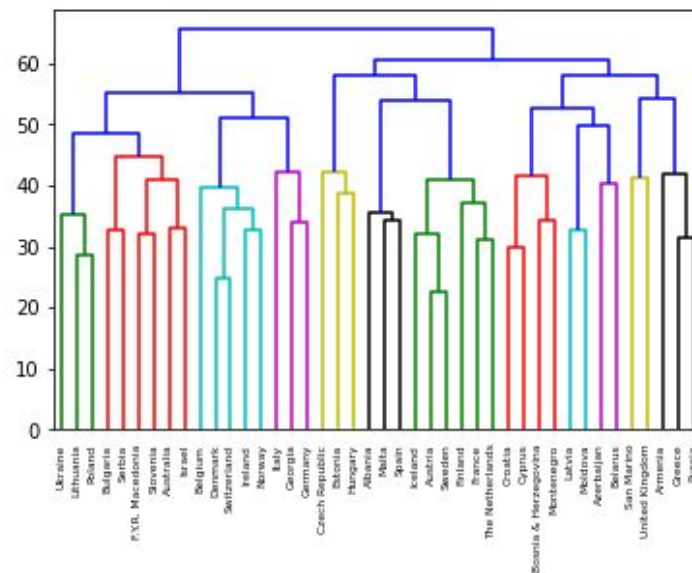
Rozwiązanie

W bibliotece scikit, metoda linkage odpowiada za klasteryzację hierarchiczną, natomiast metoda dendrogram pozwala wizualizować wynik jej pracy. Linkage 'complete' oznacza, że dystans pomiędzy klastrami będzie mierzony jako dystans pomiędzy najbardziej oddalonymi od siebie próbkami w obu klastrach.

```
from scipy.cluster.hierarchy import linkage, dendrogram

# Calculate the linkage: mergings
mergings = linkage(eurovision, method='complete')

# Plot the dendrogram, using varieties as labels
dendrogram(mergings,
            labels=eurovision.index,
            leaf_rotation=90,
            leaf_font_size=6,
            )
plt.show()
```



Rozwiązanie

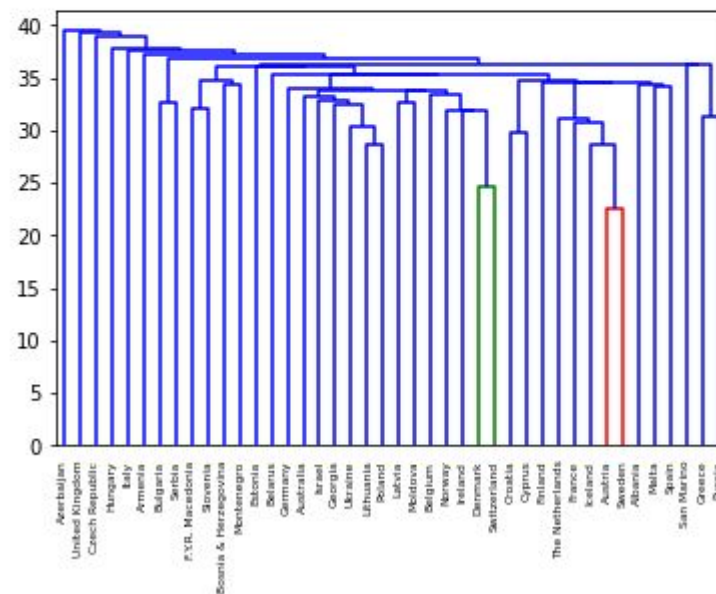
Wysokość dendrogramu pozwala określić stopień zbliżenia poszczególnych punktów (im niżej łączenie, tym bliżej siebie elementy). Można wybrać inną metodę łączenia np. 'single', wtedy odległość między klastrami mierzona za pomocą odległości najbliższej sobie położonych punktów obu klastrów.

Tutaj również możemy bawić się normalizacją próbek.

```
mergings = linkage(eurovision, method='single')

# Plot the dendrogram, using varieties as labels
dendrogram(mergings,
            labels=eurovision.index,
            leaf_rotation=90,
            leaf_font_size=6,
            )

plt.show()
```



DBSCAN



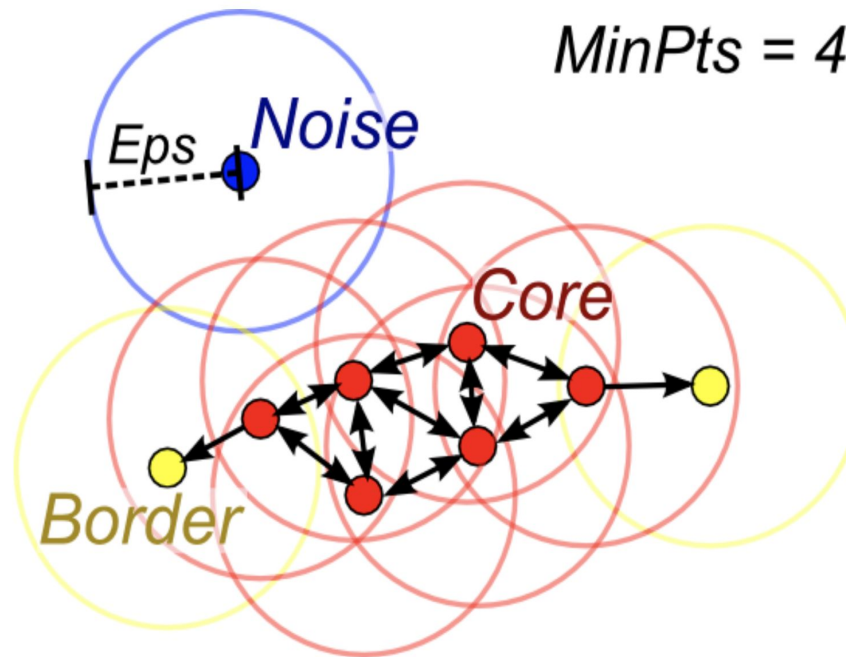


DBSCAN



- punkty są dzielone na rdzenie (core), punkty graniczne (border) i szumy (noise)
- punkty oznaczone jako szum są eliminowane
- rysowana jest krawędź pomiędzy wszystkimi punktami rdzenia, to znaczy punktami, których odległość od siebie jest mniejsza od pewnego epsilon
- każda grupa połączonych punktów rdzenia, to osobny klaster
- jeśli dany punkt łączy się tylko z jednym punktem rdzenia, oznacza się go jako granicę

DBSCAN



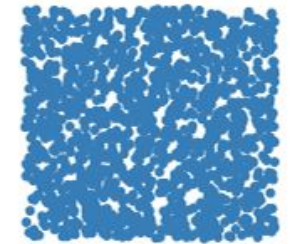
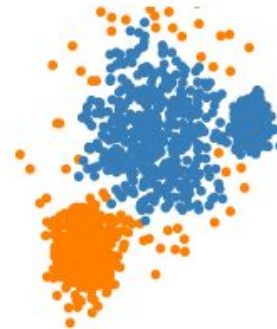
Red: Core Points

Yellow: Border points. Still part of the cluster because it's within epsilon of a core point, but not does not meet the min_points criteria

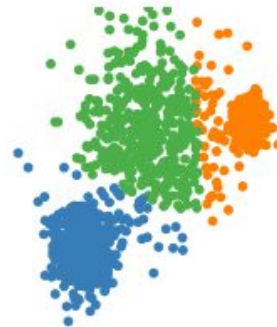
Blue: Noise point. Not assigned to a cluster

DBSCAN

DBSCAN



k-means






DBSCAN



Zalety

- + znajdowanie kształtów o różnych kształtach, nie tylko sferycznych
- + nie musimy znać liczby szukanych klastrów
- + dobrze radzi sobie z danymi zaszumionymi

Wady

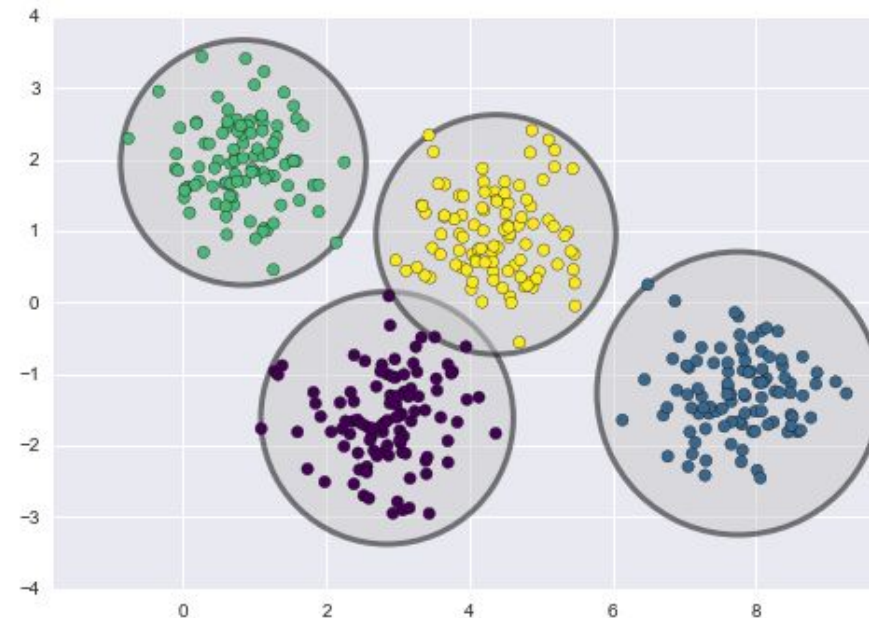
- czuły na dobór parametrów, minPoints i eps
 - słabo radzi sobie z klastrami o różnej gęstości zagęszczenia próbek
 - Słabo radzi sobie z klastrami blisko siebie
- 

Gaussowski model mieszany



Problemy z K-means

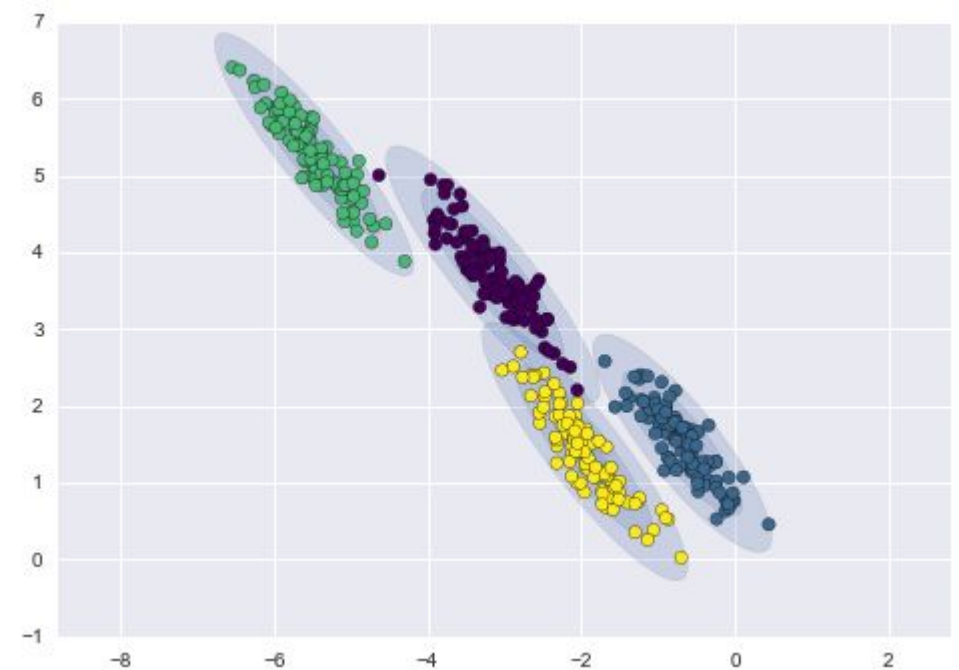
Jednym ze sposobów myślenia o K-meansie jest wyobrażenie sobie 2 wymiarowej przestrzeni, następnie wbicie cyrkla w środek z każdego z klastrów i zatoczenie okręgu.



Problemy z K-means

To działa świetnie dopóki dane faktycznie mają “okrągły” kształt (na hiperpłaszczyźnie należy sobie wyobrazić punkty znajdujące się w regularnych odstępach w każdym z wymiarów)

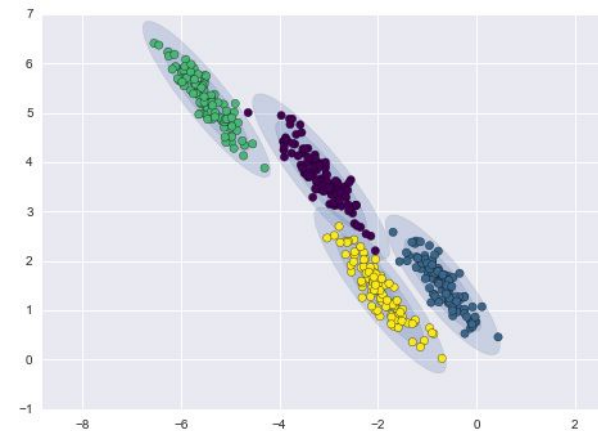
Jednak dane czasami mogą wyglądać tak
->



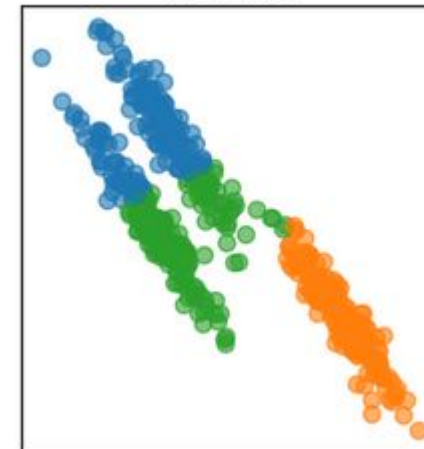
Problemy z K-means

K-means nie poradzi sobie w żaden sposób z tego typu danymi ->

Będzie działać na podobnej zasadzie jak obok ->

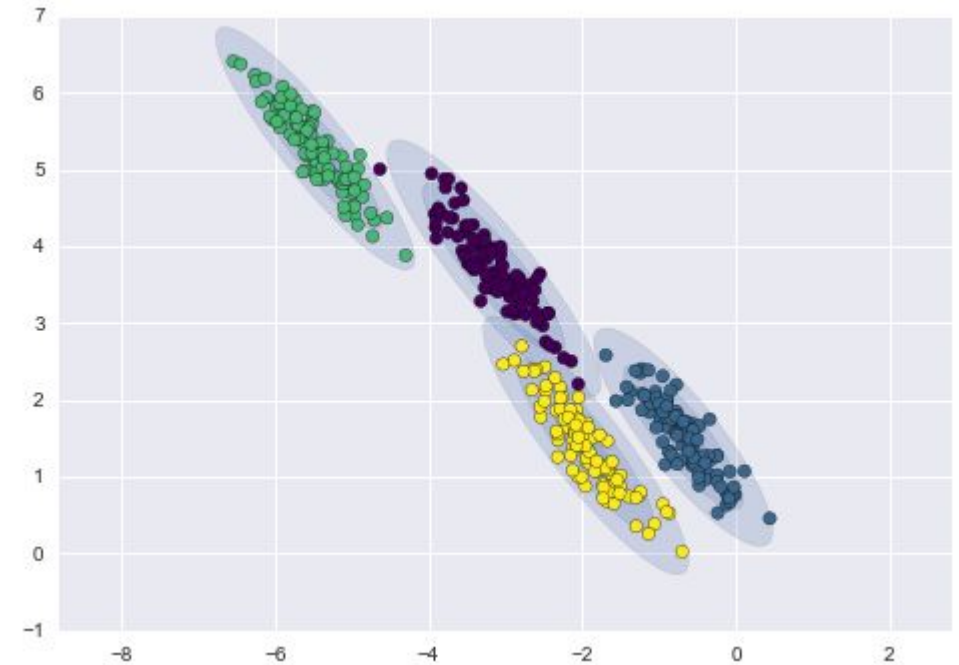


KMeans



Problemy z DBSCAN

Tak samo wyglądać będzie sytuacja z DBSCANEM, który ma spore problemy w przypadku blisko położonych siebie klastrów...





Wprowadzenie do GMMs

Gaussowski model mieszany zakłada, że w zbiorze danych występuje określona ilość rozkładów gaussowskich i każdy z nich reprezentuje klaster. Tym samym GMM grupuje ze sobą dane pochodzące z tego samego rozkładu.



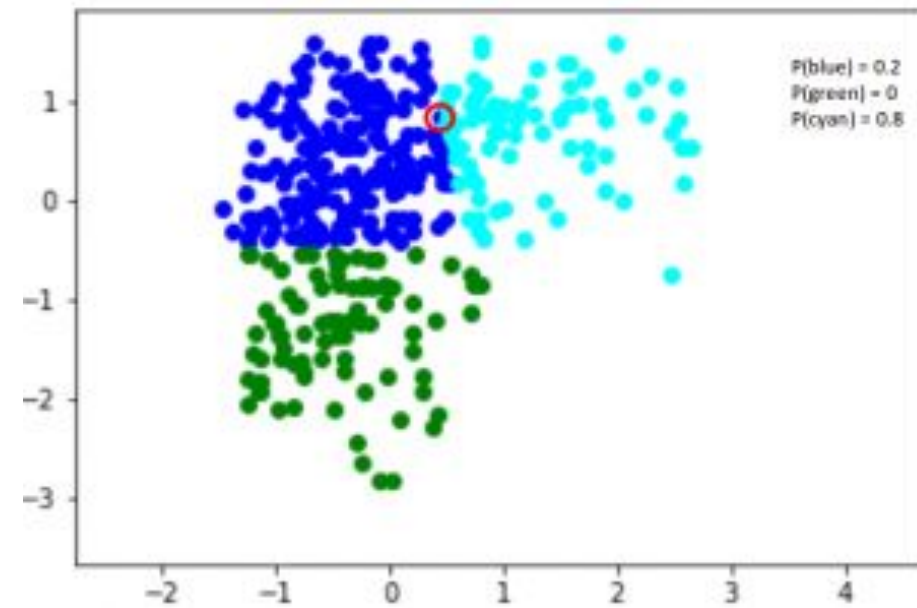
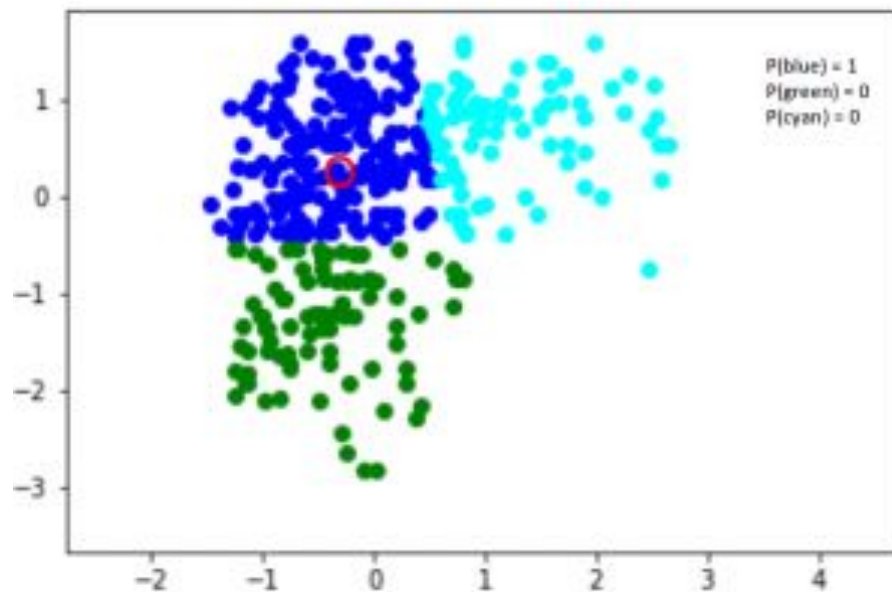
Grupowanie miękkie

Gaussowski model mieszany jest przykładem klasteryzacji miękkiej. To znaczy?

Założmy, że mamy 3 Gaussowskie rozkłady, każdy z nich posiada określoną średnią i wariancję. Dla danego zbioru danych GMM określi **prawdopodobieństwo** przynależności poszczególnych elementów do danego rozkładu.

Klasteryzacja Gaussa nie daje pewności, że dany obiekt należy do danej klasy, tylko że należy z pewną dozą prawdopodobieństwa!

Grupowanie miękkie





Ustalanie parametrów rozkładu

Klastry są reprezentowane przez odpowiednie rozkłady Gaussa, ale w jaki sposób określane są docelowe parametry tych rozkładów?

Wszystko dzięki technice Expectation-Maximization.



Expectation-Maximization



Expectation-Maximization (EM) jest statystycznym algorytmem do znajdowania parametrów modelu. Typowo używa się EM, kiedy dane posiadają missing values, czy też są niekompletne.

W tym przypadku missing values są jak w każdym uczeniu nienadzorowanym etykiety danych. Jeżeli byśmy je posiadali to z łatwością moglibyśmy określić parametry rozkładów.



Expectation-Maximization



Jako, że nie posiadamy tych informacji. EM stara się, z wykorzystaniem posiadanych danych, określić optymalne wartości rozkładów. Trochę na zasadzie działania K-means – z tym, że tutaj poza średnią wartością naszego zbioru, zależy nam również na pozyskaniu jego wariancji.



Expectation-Maximization



Algorytm EM składa się z następujących kroków:

1. Rozpocznij z losowymi wartościami rozkładów Gaussowskich
2. Powtarzaj aż do osiągnięcia zbieżności:
 - E-step – w tym kroku dostępne dane używane są do estymowania klas obiektów
 - M-step – Bazując na wartościach określonym w kroku E-step, zaktualizuj wartości rozkładów

E-step

Dla każdego punktu x_i , oblicz prawdopodobieństwo, że punkt należy do danego rozkładu:

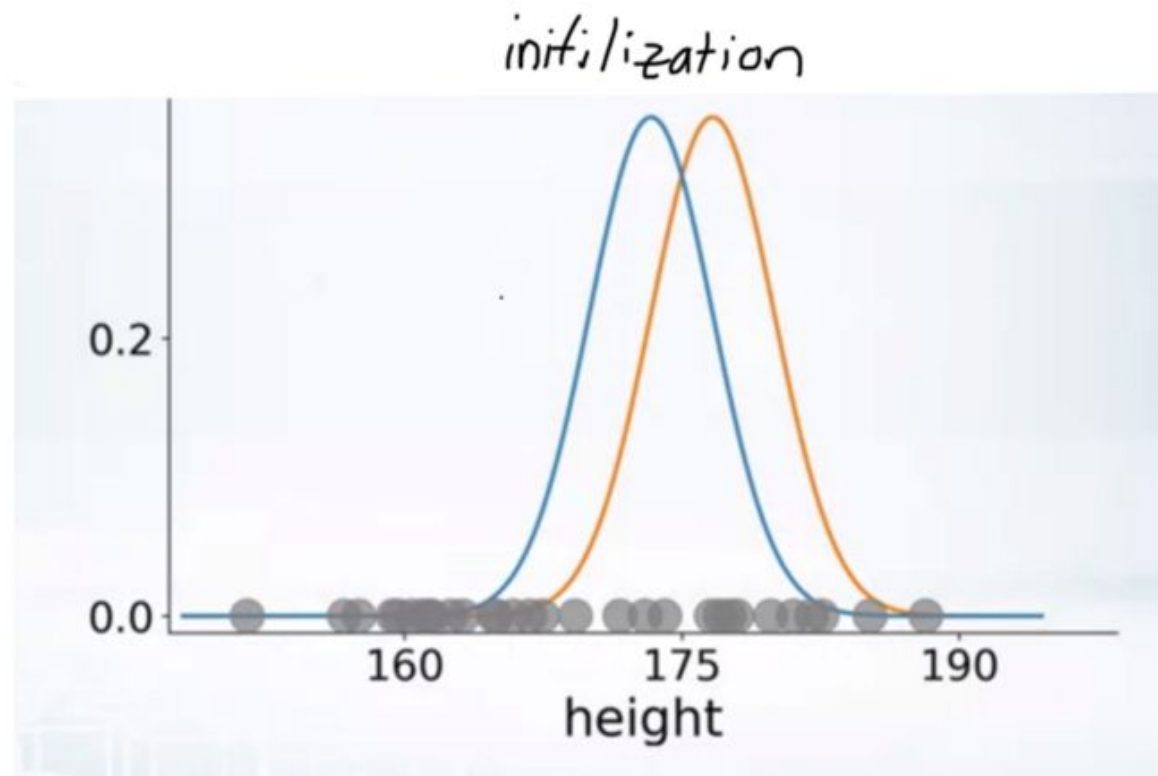
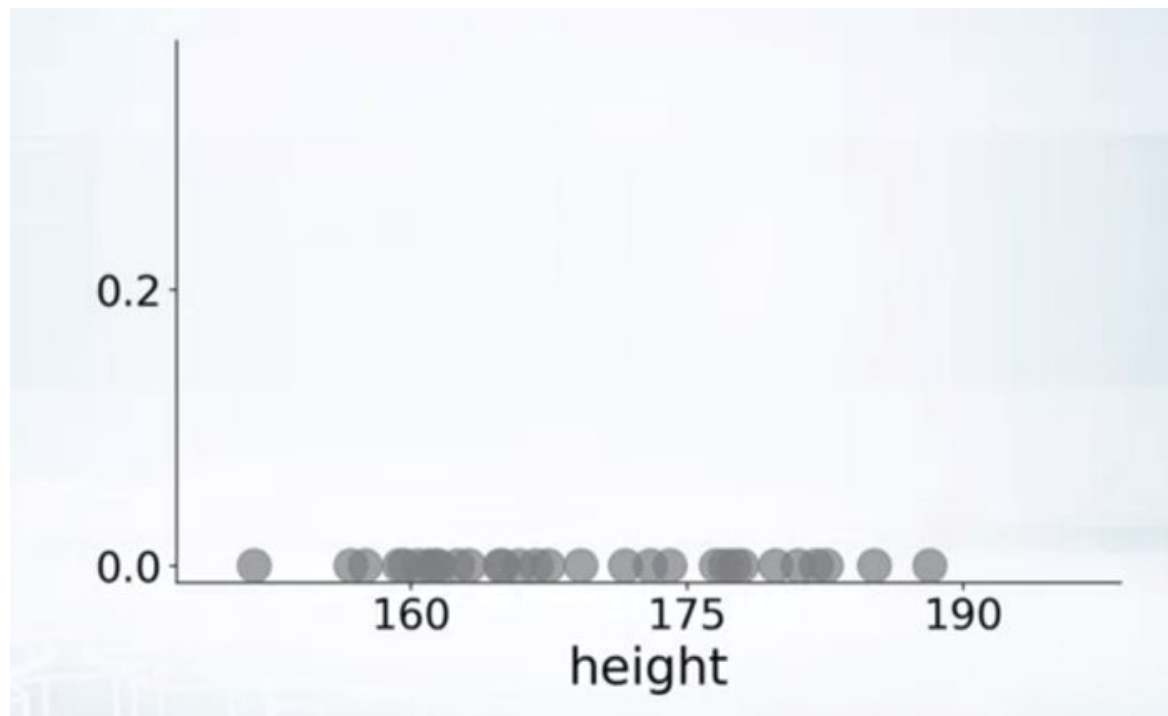
$$r_{ic} = \frac{\text{Probability } x_i \text{ belongs to } c}{\text{Sum of probability } x_i \text{ belongs to } c_1, c_2, \dots, c_k} = \frac{\pi_c \mathcal{N}(x_i ; \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} \mathcal{N}(x_i ; \mu_{c'}, \Sigma_{c'})}$$

M-Step

Z wykorzystaniem informacji z E-stepu możemy obliczyć nowe parametry rozkładów:

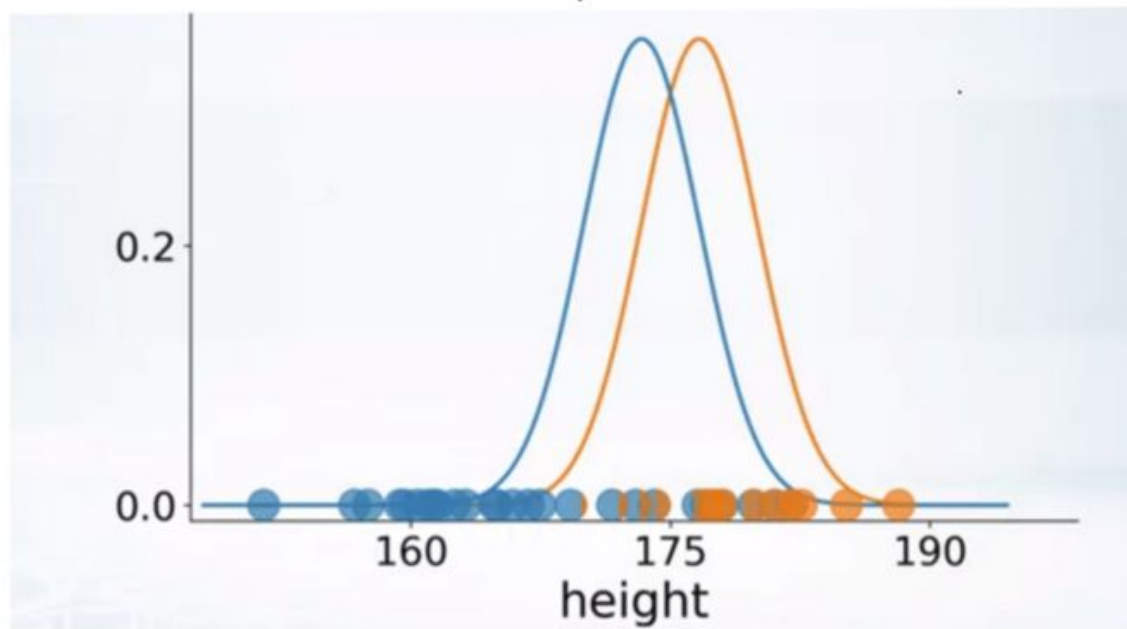
$$\mu = \frac{1}{\text{Number of points assigned to cluster}} \sum_i r_{ic} x_i$$
$$\Sigma_c = \frac{1}{\text{Number of points assigned to cluster}} \sum_i r_{ic} (x_i - \mu_c)^T (x_i - \mu_c)$$

Działanie algorytmu GMM na przykładzie 1 wymiarowych danych

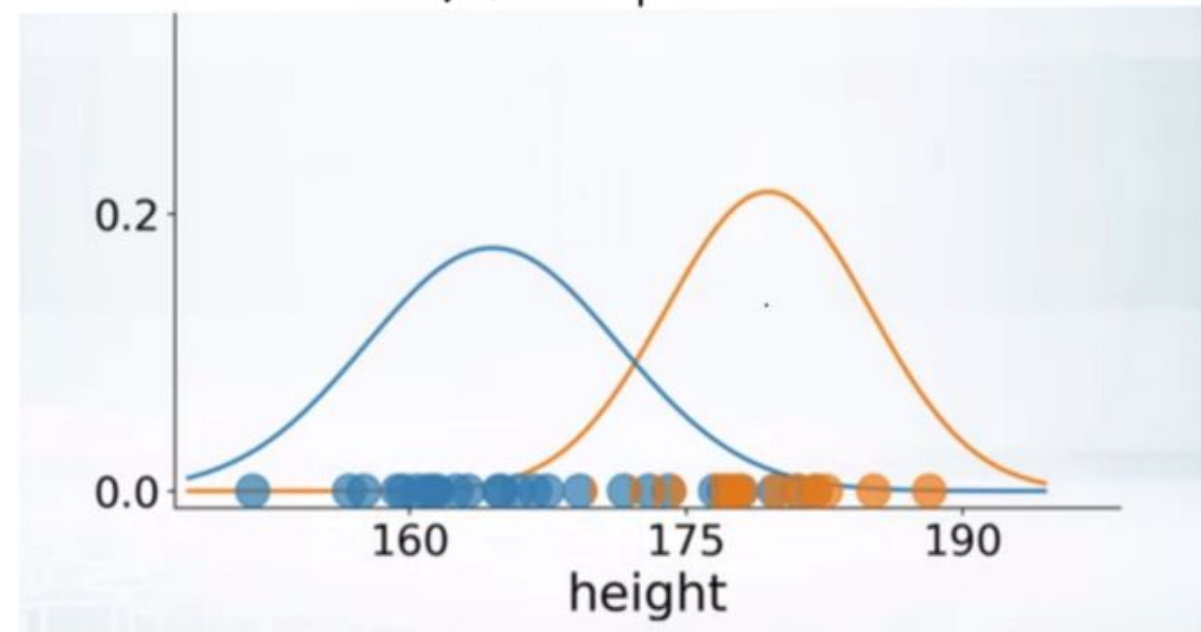


Działanie algorytmu GMM na przykładzie 1 wymiarowych danych

Iteration 1
E step

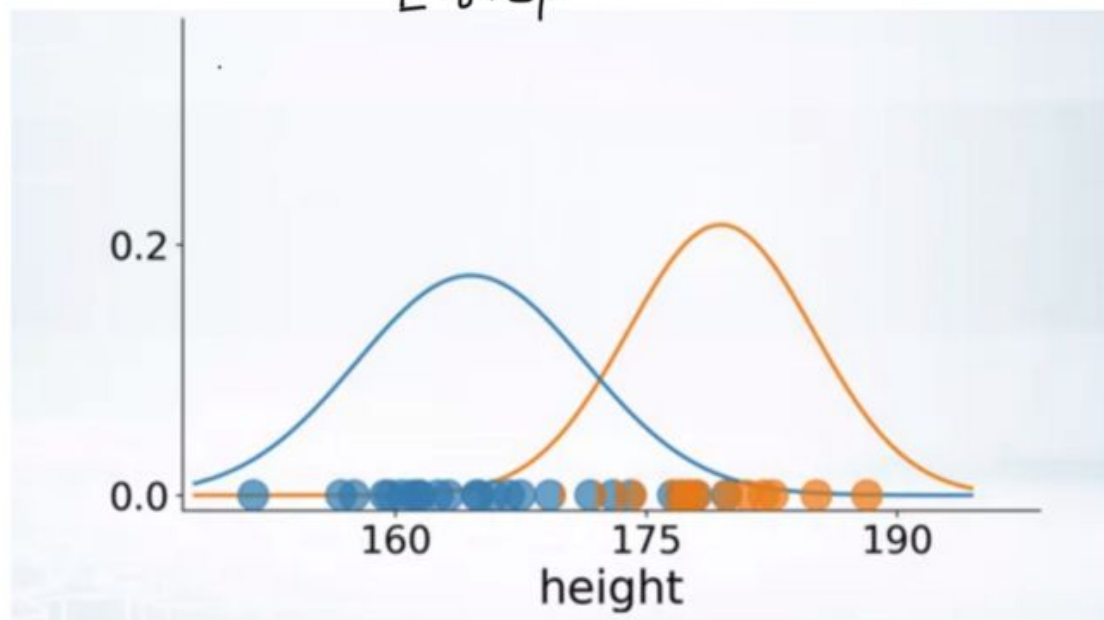


Iteration 1
M step

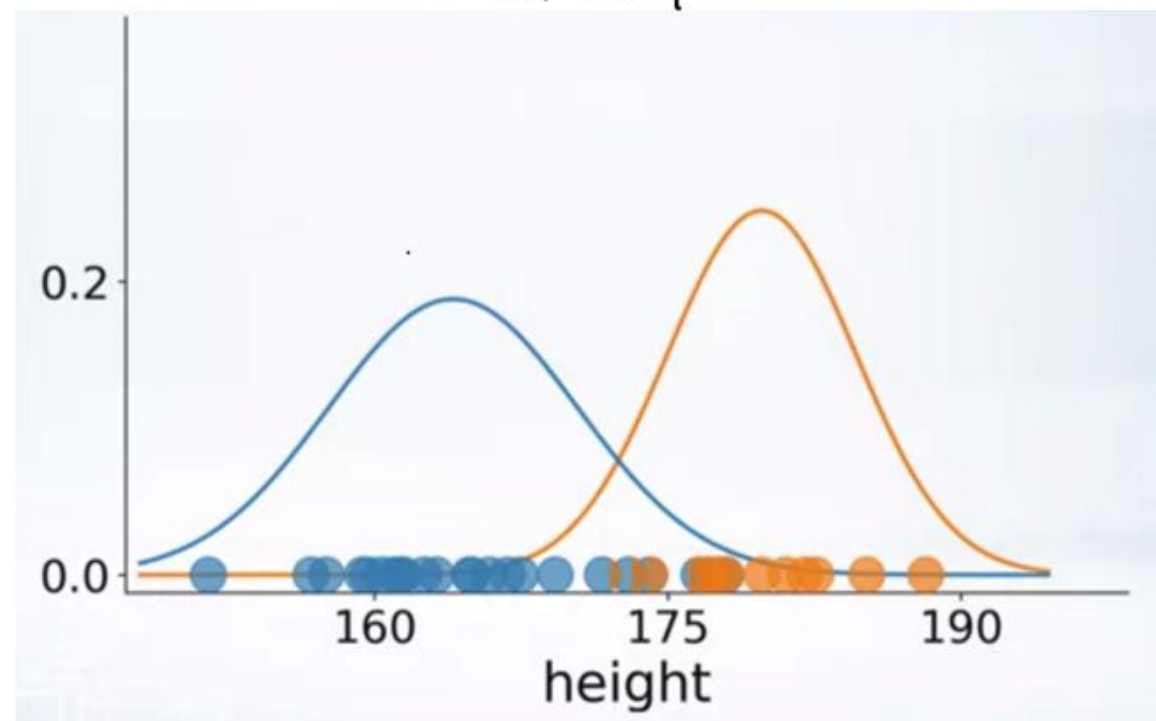


Działanie algorytmu GMM na przykładzie 1 wymiarowych danych

Iteration 2
E step



Iteration 2
M step





Podsumowanie

Warto pamiętać, z jakimi przykładami może poradzić sobie GMM. Ze względu na to, że w porównaniu do K-means używa nie tylko średniej wartości danych, ale i ich wariancji.

Redukcja wymiarowości





Czym jest klątwa wymiarowości i jak sobie z nią radzić?

Klątwa wymiarowości – im więcej atrybutów, tym więcej danych uczących potrzeba do procesu klasyfikacji. Ponadto wzrasta zapotrzebowanie na moc obliczeniową. Gdy wymiarowość dąży do nieskończoności obiekty stają się coraz bardziej odległe od siebie, przez co spada jakość klasyfikacji. Rozwiązanie: redukcja wymiarów.



Czym jest redukcja wymiarów?

Redukcja wymiarów danych – jest to proces przetwarzania danych prowadzący do redukcji ich wymiarowości.

Dzieli się na:

- **Feature selection** – w którym to staramy się wyłuskać z danych tylko te istotne informacje pomocne w rozwiązaniu danego problemu
- **Feature extraction** – polega na stworzeniu nowych featurów na podstawie tych obecnych i ich zapis w bardziej informatywnej i znaczącej formie. Np. Wydobywanie krawędzi z obrazu



Po co redukujemy wymiary?

- większa efektywność obliczeniowa
- mniej zajętego miejsca w pamięci
- eliminacja szumów i niepotrzebnych informacji
- mniejszy problem przy klasyfikacji, regresji itd.
- redukcja powielających się informacji

Feature extraction






PCA (Principal Component Analysis)



Wyobraźmy sobie, że mamy przed sobą banki w Polsce opisane przez kilka cech:

- liczba klientów,
- liczba aktywnych klientów,
- liczba klientów z aktywnym kontem,
- liczba kart debetowych,
- liczba klientów mobilnych,
- liczba placówek.

Jestem przekonany, że na podstawie intuicji każdy z nas jest w stanie przypuścić, że kategoria liczba klientów z aktywnym kontem będzie skorelowana z liczbą kart debetowych. Tak samo w wielu przypadkach liczba klientów będzie powiązana z liczbą placówek (choć czasami możemy się pozytywnie zdziwić). Po to aby nie powielać niektórych informacji i zyskać lepszą przejrzystość można zamienić dwie zmienne na jedną tak zwaną składową.





PCA (Principal Component Analysis)



Analiza składowych głównych (PCA) to najbardziej popularny algorytm redukcji wymiarów. W ogólnym skrócie **polega on na rzutowaniu danych do przestrzeni o mniejszej liczbie wymiarów tak, aby jak najlepiej zachować strukturę danych.**

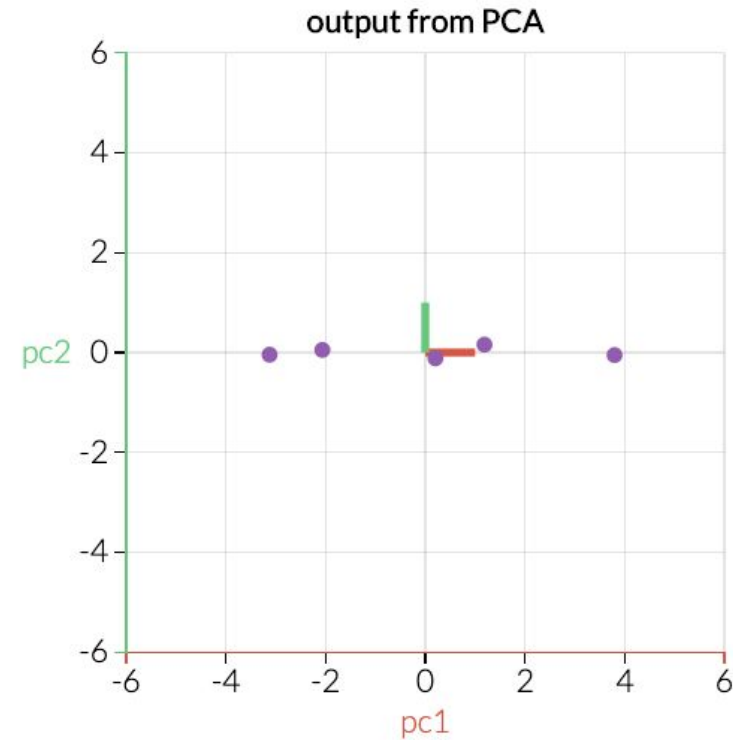
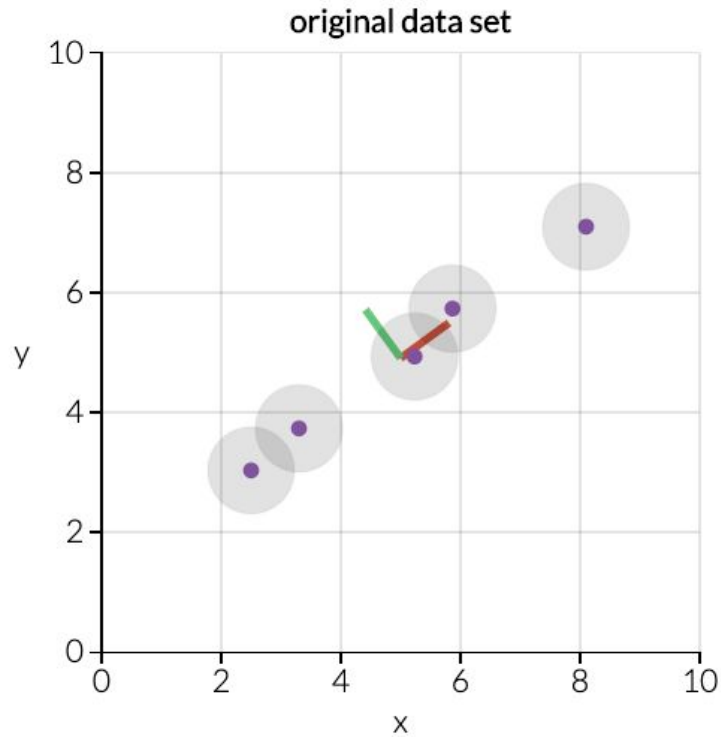
Służy głównie do redukcji zmiennych opisujących dane zjawisko oraz odkrycia ewentualnych prawidłowości między cechami. Dokładna analiza składowych głównych umożliwia wskazanie tych zmiennych początkowych, które mają duży wpływ na wygląd poszczególnych składowych głównych czyli tych, które tworzą grupę jednorodną.



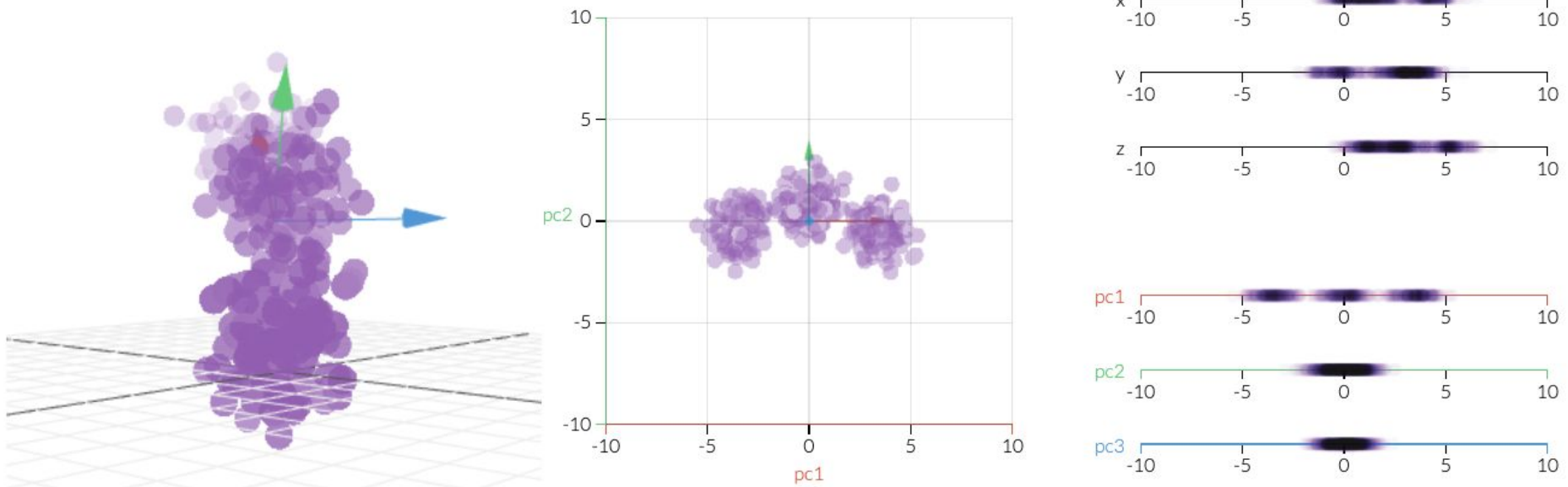
PCA (Principal Component Analysis)

- uwypuklenie wariancji
- wykrywanie silnych wzorców w zbiorze danych
- zalecana dla każdego datasetu o liczbie cech większej od 10
- **Principal Component** – liniowo nieskorelowane wektory, o wysokiej wariancji, utworzone na podstawie cech ze zbioru danych

PCA (Principal Component Analysis)



PCA (Principal Component Analysis)



A teraz spróbujmy dla danych zaetykietowanych

```
wine, wine_classes = load_dataset('wine', 'Class')
```

```
y = wine.pop('class')
```

```
X = wine
```

```
X.head()
```

	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

Przykład

Wczytajmy zbiór win i wyświetlmy korelację między dwoma kolumnami.

```
from scipy.stats import pearsonr

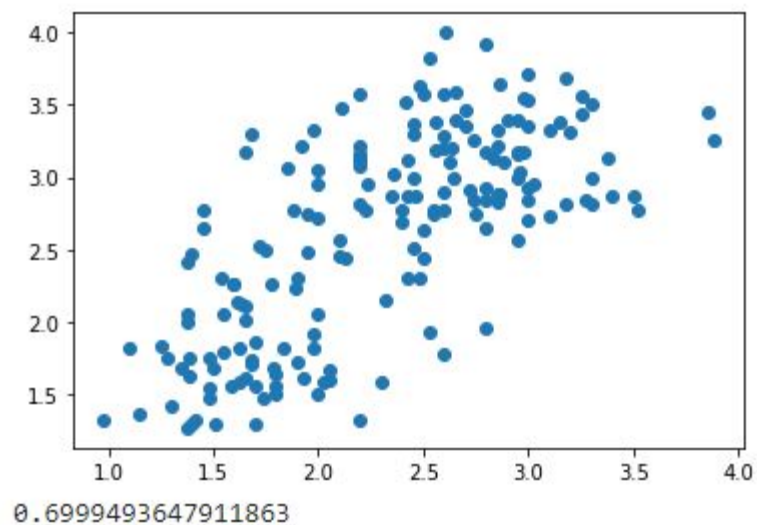
wine, wine_classes = load_dataset('wine', 'Class')

y = wine.pop('class')
X = wine

xs = wine['Total phenols']
ys = wine['OD280/OD315 of diluted wines']

fig, ax = plt.subplots()
ax.scatter(xs, ys)
plt.show()

correlation, pvalue = pearsonr(xs, ys)
print(correlation)
```



Przykład

Na początek za pomocą PCA możemy dokonać dekoreralizacji.

```
from sklearn.decomposition import PCA
model = PCA()

# Apply the fit_transform method of model to grains: pca_features
pca_features = model.fit_transform(wine)

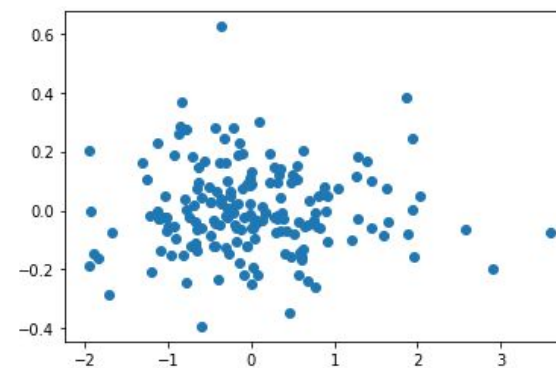
# Assign 5th column of pca_features: xs
xs = pca_features[:,5]

# Assign 11th column of pca_features: ys
ys = pca_features[:,11]

# Scatter plot xs vs ys
plt.scatter(xs, ys)
plt.show()

# Calculate the Pearson correlation of xs and ys
correlation, pvalue = pearsonr(xs, ys)

# Display the correlation
print(correlation)
```



-7.3075226425523e-17

Przykład

Po pozbyciu się korelacji możemy zająć się już redukcją wymiarów (metoda stratna już). Musimy znaleźć ile minimalnie parametrów potrzebuje dataset żeby być poprawnie rozpoznawanym.

Algorytm PCA szuka takich cech, które mają wysoką wariancję. W tym również będzie nam pomocna biblioteka PCA. Widzimy, że dla 3 komponentów wariancja jest już dość niska.

```
# Create scaler: scaler
scaler = StandardScaler()

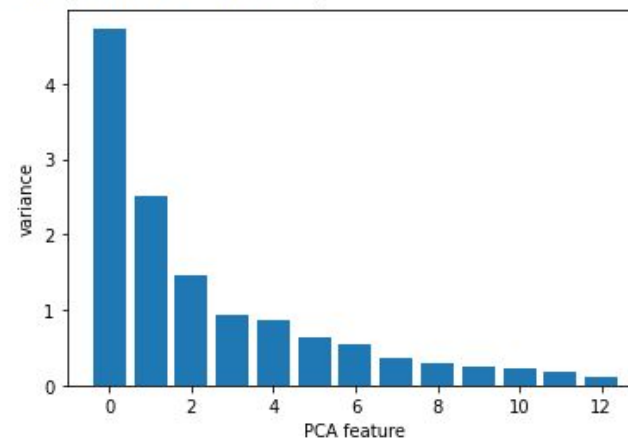
# Create a PCA instance: pca
pca = PCA()

pipeline = make_pipeline(scaler, pca)

# Fit the pipeline to 'samples'
pipeline.fit(wine)

# Plot the explained variances
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_)
plt.xlabel('PCA feature')
plt.ylabel('variance')
```

Text(0, 0.5, 'variance')



Przykład

Przejdźmy finalnie do redukcji wymiarowości. W tym celu przy definiowaniu modelu PCA musimy określić ilość komponentów jakie chcemy uwzględnić, czyli w naszym wypadku trzy.

```
scaler = StandardScaler()
scaler.fit(wine)
scaled_wine = scaler.transform(wine)

pca = PCA(n_components=3)

# Fit the PCA instance to the scaled samples
pca.fit(scaled_wine)

# Transform the scaled samples: pca_features
pca_features = pca.transform(scaled_wine)

# Print the shape of pca_features
print(pca_features.shape)

(178, 3)
```

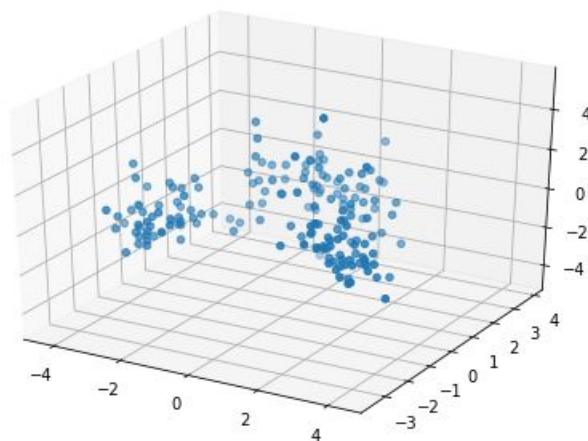
Przykład

Na koniec możemy obejrzeć nasze 3 wymiary na wykresie:

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = Axes3D(fig)

ax.scatter(pca_features[:,0], pca_features[:,1], pca_features[:,2])
plt.show()
```



tSNE – t-distributed stochastic neighbour embedding

- Inna metoda do wizualizacji i klasteryzacji danych.
- Mapuje na przestrzeń 2 lub 3 wymiarową.
- Świetna do odkrywania zbiorów.
- Przykładowo iris ma 4 atrybuty, ale możemy je przekształcić w 2 posiadające porównywalną wartość informacyjną
- metoda tSNE posiada parametr learning rate.

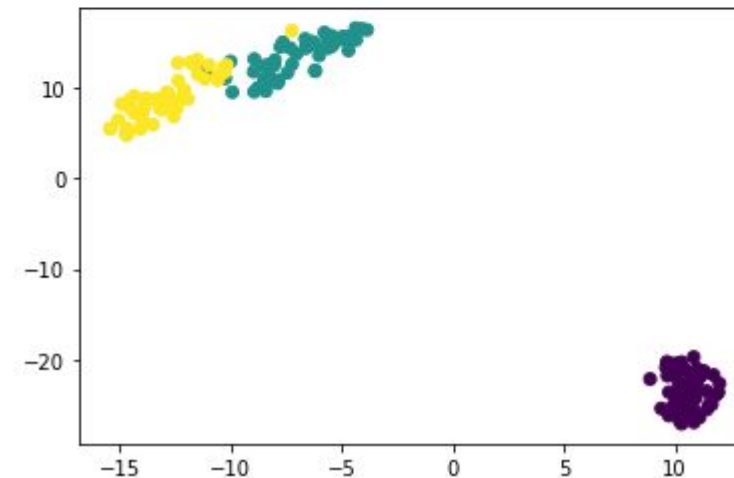
```
# Create a TSNE instance: model
model = TSNE(learning_rate=300)

# Apply fit_transform to samples: tsne_features
tsne_features = model.fit_transform(X)

# Select the 0th feature: xs
xs = tsne_features[:,0]

# Select the 1st feature: ys
ys = tsne_features[:,1]

# Scatter plot, coloring by variety_numbers
plt.scatter(xs, ys, c=y)
plt.show()
```

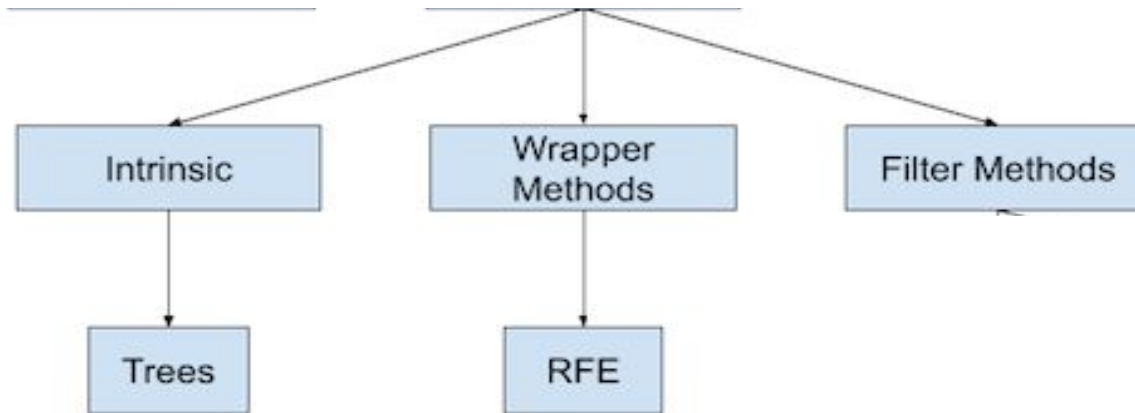


Feature selection



Podział technik Feature Selection

Feature Selection methods





Intrinsic methods

To między innymi metody opierające się o drzewa, są to algorytmy oferujące możliwość określenia tzw. "Feature Importance".

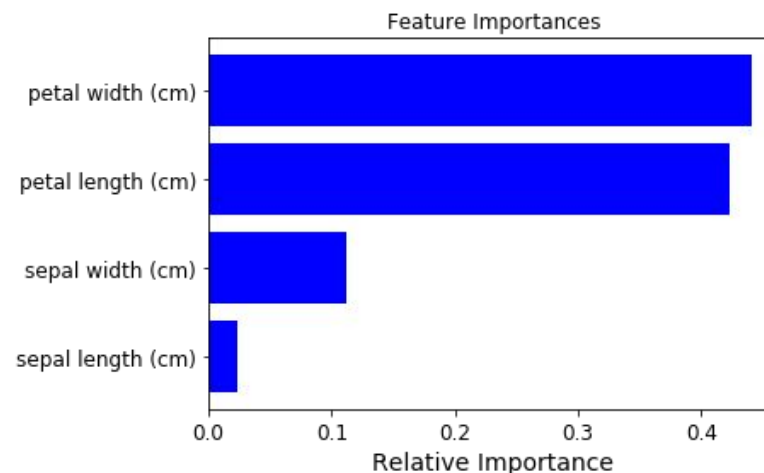
Feature Importance mówi nam, jakie cechy były najbardziej użyteczne do uzyskania predykcji. Informacje te możemy pozyskać z drzew decyzyjnych, czy lasów losowych.

Intrinsic methods

W Pythonie są gotowe metody, które umożliwiają uzyskanie takiej informacji:

[Feature importances with forests of trees — scikit-learn 0.23.2 documentation](#)

Przykładowo dla zbioru iris, możemy zauważyć, że informacje dotyczące badanych płatków kwiatu mają większe znaczenie, niż wymiary działki kielicha.





Wrapper-based methods



Działają poprzez ewaluację wybranej części featurów poprzez użycie ich w procesie uczenia maszynowego. **Metody te bazują na poszukiwaniu takiego podzbioru cech, który będzie gwarantował możliwe wysokie finalne osiągnięcia modelu.**

Sprawia to, że metody te są bardzo kosztowne obliczeniowo.



Wrapper-based proces działania



- Przy użyciu zdefiniowanego algorytmu szukania wybierz podzbiór z dostępnych featurów.
- Stwórz model uczenia maszynowego
- Zbadaj jakość stworzonego modelu
- Powtórz cały proces.



Wrapper-based kryterium stopu



W pewnym momencie należy zatrzymać taki proces. Kiedy? To zależy od przyjętego kryterium stopu:

- Jakość modelu wzrosła
- Jakość modelu spadła
- Osiągnięto zdefiniowaną wcześniej ilość poszukiwanych featurów



Wrapper-based metody poszukiwania

Wyróżnia się następujące rodzaje metod poszukiwania:

- Forward Feature Selection: metoda ta zaczyna od wybrania pierwszego featury i dodaje kolejne sprawdzając czy poprawiają one jakość działania algorytmu.
- Backward Feature Elimination: metoda ta zaczyna z pełnym zbiorem featurów, a następnie usuwa kolejne i sprawdza czy poprawiają one działanie modelu.
- Exhaustive Feature Selection: metoda ta sprawdza wszystkie możliwe kombinacje cech
- Bidirectional Search: Jednocześnie na przemian dodaje się i odejmuje featury i sprawdza się, czy przynosi to jakiś efekt.



Filter-based methods



Filter methods wyróżniają się tym, że mogą być zastosowane przed rozpoczęciem procesu uczenia naszego modelu. Przez co stanowią zazwyczaj pierwszy krok w selekcji cech w pipelinech.

Zalety:

- mogą być zastosowane do dowolnego algorytmu uczenia maszynowego
- są mało wymagające obliczeniowo

Są świetne do eliminacji cech nie mających większego znaczenia, powielanych informacji, wartości ciągłych, zduplikowanych i skorelowanych.



Filter-based methods



- Basic Filter Methods (usuwanie wartości stałych, lub też quasi-stałych, zdublowanych)
- Correlation Filter Methods (bazujące na korelacji pomiędzy zmiennymi)
- Statistical & Ranking Filter Methods (starają się zewaluować wartość informacyjną danej cechy indywidualnie przy braniu pod uwagę celu, który chcemy osiągnąć)



Filter-based methods



- Basic Filter Methods
- Correlation Filter Methods

W celu zastosowania większości method dostępnych w ramach tych dwóch rodzajów filter-based metod. Można wykorzystać bibliotekę pandas profiling.

Pandas profiling umożliwia odnalezienie korelacji pomiędzy danymi, wartościami stałymi, zdublowanymi.



Dla zainteresowanych

Więcej informacji na temat feature selection można znaleźć tutaj:
[Hands-on with Feature Selection Techniques: An Introduction](#)

Podsumowanie





Dlaczego warto w uczenie nienadzorowane?



- Potrafi odnaleźć nieznane do tej pory wzory i zależności w zbiorach danych.
- Pomaga w znajdowaniu cech, które będą pomocne w zadaniach kategoryzacji, co jeszcze szczególnie ważne przy bardzo dużej ilości cech – redukcja rozmiaru zbioru danych z zachowaniem możliwie dużej ilości informacji.
- Dane niezalabelowane są tańsze, aniżeli te, które posiadają etykiety (szczególnie, gdy trzeba je ręcznie dodawać).



Dziękuję za uwagę!

