Maciej Zagórski
Informatyka, III rok

RAPORT Z PROJEKTU
Sprawdzenie działania i weryfikacja poprawności wybranych algorytmów grupujących

1. Przedmiot projektu

Przedmiotem projektu było sprawdzenie, jak działają następujące, poznane algorytmy grupujące:

- algorytm centroidów (k-średnich, *k-means*),
- algorytm gęstościowej klasteryzacji przestrzennej z uwzględnieniem szumu (*density-based spatial clustering of applicaions with noise,* DBSCAN).

Dodatkowo wykonano prostą implementację algorytmu hierarchicznego – klasteryzacji aglomeracyjnej (*agglomerative clustering*).

Ponadto celem realizacji projektu była weryfikacja poprawności (rezultatów zastosowania) przedmiotowych algorytmów.

Powyższego dokonano w oparciu o przykładowe zbiory danych „iris2D.csv" oraz „irisORG.csv", oba zawierające dane o kwiatach należących do rodzaju irysów. Zastosowanie algorytmów grupujących ma w tym przypadku na celu połączenie kwiatów w grupy – ich podział na gatunki, tj. przyporządkowanie poszczególnych kwiatów do kolejno wyodrębnianych grup (odpowiadających gatunkom kwiatów z rodzaju irysów).

2. Realizacja projektu

2.1. Analiza i przygotowanie zbiorów danych

Każdy z wymienionych powyżej zbiorów danych – „iris2D.csv" oraz „irisORG.csv" – zawiera informacje o 150 kwiatach należących do rodzaju irysów. Przy czym zbiór „irisORG.csv" zawiera (nieprzetworzone) informacje o parametrach poszczególnych kwiatów: długość i szerokość działki kielicha (odpowiednio: *sepal.length* i *sepal.width*) oraz długość i szerokość płatka (*petal.length* i *petal.width;* w zbiorze nie zawarto informacji o jednostkach, w jakich wyrażane są poszczególne wartości).

Z kolei zbiór „iris2D.csv" również zawiera powyższe dane (w odniesieniu do każdego ze 150 kwiatów), ale w postaci skompresowanej do dwóch wymiarów. Taką postać przedmiotowych danych uzyskano dokonując w pierwszej kolejności ich standaryzacji (funkcja *StandardScaler* z modułu *sklearn.preprocessing*, z biblioteki *scikit-learn*), a następnie kompresji – przy wykorzystaniu analizy głównych składowych, *principal component analysis,* PCA (funkcja *PCA* z modułu *sklearn.decomposition* ze wspomnianej biblioteki).

```
stdsc = StandardScaler()
mms = MinMaxScaler()
pca = PCA(n_components=2)

data_standardized_uncompressed = stdsc.fit_transform(data_uncompressed.iloc[:, :-1])
data_standardized_compressed = pca.fit_transform(data_standardized_uncompressed)

data_standardized_compressed = pd.DataFrame(data_standardized_compressed,
                                            columns=['"PC1"', '"PC2"'])
```

```
print(data_standardized_compressed)
```

```
     sepal.length  sepal.width  petal.length  petal.width    variety
0             5.1          3.5           1.4          0.2     Setosa
1             4.9          3.0           1.4          0.2     Setosa
2             4.7          3.2           1.3          0.2     Setosa
3             4.6          3.1           1.5          0.2     Setosa
4             5.0          3.6           1.4          0.2     Setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  Virginica
146           6.3          2.5           5.0          1.9  Virginica
147           6.5          3.0           5.2          2.0  Virginica
148           6.2          3.4           5.4          2.3  Virginica
149           5.9          3.0           5.1          1.8  Virginica

[150 rows x 5 columns]

        "PC1"     "PC2"
0   -2.264703  0.480027
1   -2.080961 -0.674134
2   -2.364229 -0.341908
3   -2.299384 -0.597395
4   -2.389842  0.646835
..        ...       ...
145  1.870503  0.386966
146  1.564580 -0.896687
147  1.521170  0.269069
148  1.372788  1.011254
149  0.960656 -0.024332

[150 rows x 2 columns]
```

W zbiorze „irisORG.csv” zawarto ponadto informację o tym, do jakiego gatunku irysów należy określony kwiat (o określonych w zbiorze parametrach) – w kolumnie *variety*. Informacja ta może przyjmować jedną z trzech wartości: *Setosa*, *Versicolor* lub *Virginica* (rozkład gatunków jest równomierny, tzn. do każdego z nich należy 50 próbek). Oznacza to, że każdy z kwiatów, których dane zawarto w zbiorach, należy do jednego z tych trzech gatunków irysów. Kolumna z tymi informacjami ze zbioru „irisORG.csv” posłuży do weryfikacji poprawności (rezultatów zastosowania) algorytmów grupujących w ramach projektu. Z tego względu została ona wyodrębniona jako podzbiór danych, do wykorzystania na dalszym etapie projektu (zob. p. 2.4 niżej).

```
data_iris_results = pd.read_csv('../data/report_2_clustering_algorithms_iris_org.csv')

y_true = data_iris_results.iloc[:, -1]
```

Zbiór danych „iris2D.csv”, który zostanie wykorzystany do sprawdzenia, jak działają algorytmy grupujące, zawiera dane porządkowe – w postaci pierwszej, nienazwanej kolumny zawierającej liczbę porządkową poszczególnych rekordów. Aby takie dane nie wpłynęły nieadekwatnie na efekty zastosowania algorytmów, zostały one w pierwszej kolejności usunięte.

```
data_iris = pd.read_csv('../data/report_2_clustering_algorithms_iris_2d.csv')

data_iris.drop([data_iris.columns[0]], axis=1, inplace=True)
```

Dane zawarte w zbiorze „irisORG.csv” są przy tym prawidłowe w tym znaczeniu, że w przypadku żadnego z kwiatów długość czy szerokość działki kielicha bądź płatka nie jest wartością zerową (co oznaczałoby brak dokonania pomiaru w tym zakresie). Jak już wskazano, zbiór „iris2D.csv” jest natomiast rezultatem odpowiedniego skompresowania danych z tego zbioru. Również kolumna *variety* w zbiorze „irisORG.csv”, zawierająca informację o gatunku każdego z kwiatów, nie jest pusta, tzn. każdy z kwiatów jest przyporządkowany do któregoś z trzech wymienionych wcześniej gatunków irysów.

2.2. Wybór i prezentacja zbiorów danych

Jak już wskazano, w celu realizacji projektu w pierwszej kolejności zaimportowano do programu zbiór danych „iris2D.csv” oraz usunięto z niego kolumnę zawierającą liczby porządkowe poszczególnych rekordów (wykorzystując do tego bibliotekę *pandas*).

Aby sprawdzić działanie wybranych algorytmów grupujących, zaimportowane dane skopiowano i przygotowano w trzech postaciach:

- „surowych danych” (określanych dalej na wykresach jako *raw data*), tj. danych w takiej postaci, w jakiej zostały przedstawione w zbiorze „iris2D.csv”,
- danych znormalizowanych (*normalized data*), tj. powyższych danych, które znormalizowano przy wykorzystaniu funkcji *MinMaxScaler* z modułu *sklearn.preprocessing* z biblioteki *scikit-learn*,
- danych ustandaryzowanych (*standardized data*), tj. danych ze zbioru „iris2D.csv”, które ustandaryzowano przez ponowne wykorzystanie funkcji *StandardScaler* z modułu *sklearn.preprocessing*.

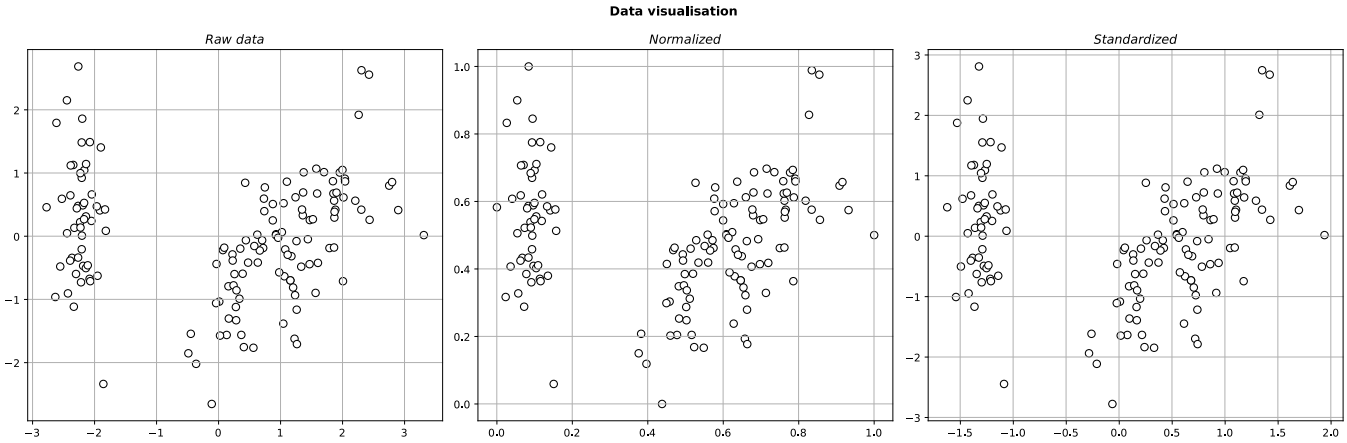
Z powyższych zestawów danych został następnie utworzony słownik (*dictionary*) o nazwie *X_data_sets*, zawierający również informację o nazwie (rodzaju) każdego zestawu danych.

```
stdsc = StandardScaler()
mms = MinMaxScaler()
# [...]

X = np.array(data_iris)
X_norm = mms.fit_transform(data_iris)
X_stand = stdsc.fit_transform(data_iris)

X_data_sets = {
    "Raw data": X,
    "Normalized": X_norm,
    "Standardized": X_stand,
}
```

W następnym kroku powyższe dane zostały zwizualizowane w formie wykresu punktowego – przy wykorzystaniu modułu *matplotlib.pyplot* z biblioteki *matplotlib* oraz własnych funkcji *plot_data* i *plot_data_loop*. Każdy punkt na wykresie to odrębny kwiat (irys – rekord, wiersz, próbka w zbiorze danych).

```
def plot_data(data_set, axis, data_set_name):
    axis.scatter(data_set[:, 0], data_set[:, 1], c='white', marker='o',
                 edgecolor='black', s=50)
    axis.set_title(data_set_name, style="italic")
    axis.grid()
```

```
def plot_data_loop(data_sets, plot_title):
    f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 6))
    axes = [ax1, ax2, ax3]
    i = 0

    for data_set_name, data_set in data_sets.items():
        plot_data(data_set, axes[i], data_set_name)
        i += 1

    f.suptitle(plot_title, weight='bold')
    plt.tight_layout()
    # [...]


plot_data_loop(X_data_sets, "Data visualisation")
# [...]

plt.show()
```



2.3. Zastosowanie algorytmów grupujących

2.3.1. Uwagi ogólne

Do powyższych zbiorów danych zastosowano algorytmy grupujące wskazane w p. 1 wyżej – każdy z algorytmów, z różnymi parametrami, osobno dla danych „surowych", znormalizowanych i ustandaryzowanych. Rezultaty ich zastosowania – pogrupowanie rekordów ze zbioru danych – przedstawiono w formie wykresów punktowych, osobno dla każdego algorytmu (i jego różnych parametrów) oraz rodzaju danych. W tym celu zdefiniowano tablice z nazwami kolorów i znaczników wykorzystanych na wykresach (które będą wykreślane przy pomocy modułu *matplotlib.pyplot* z biblioteki *matplotlib*) oraz własne funkcje *plot_fit_data* i *plot_fit_data_loop.*

```
colors = ['red', 'green', 'orange', 'blue', 'yellow', 'indigo', 'violet', 'cyan',
          'sienna', 'teal', 'purple', 'olive', 'lime', 'crimson', 'aqua', 'pink']
markers = ['^', 's', 'p', '>', 'o', 'h', 'v', 'd', '*', '<', '8', 'P', 'H', 'X', 'D']


def plot_fit_data(data_set, axis, data_set_name, algorithm, results):
    if algorithm == 'km':
        y = km.fit_predict(data_set)
    elif algorithm == 'db':
        y = db.fit_predict(data_set)
    else:
        y = ac.fit_predict(data_set)
```

```
    for i in range(min(y), max(y) + 1):
        if i == -1:
            lb = "Noise"
        else:
            lb = f'Cluster {i + 1}'

        axis.scatter(data_set[y == i, 0],
                     data_set[y == i, 1],
                     s=50, c=colors[i],
                     marker=markers[i], edgecolor='black',
                     label=lb)

    if algorithm == 'km':
        axis.scatter(km.cluster_centers_[:, 0],
                     km.cluster_centers_[:, 1],
                     s=250, marker='*',
                     c='grey', edgecolor='black',
                     label='Centroids')

    axis.set_title(data_set_name, style="italic")
    axis.legend(scatterpoints=1)
    axis.grid()

    results.append(y)


def plot_fit_data_loop(data_sets, plot_title, algorithm):
    f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 6))
    axes = [ax1, ax2, ax3]
    i = 0
    results = []

    for data_set_name, data_set in data_sets.items():
        plot_fit_data(data_set, axes[i], data_set_name, algorithm, results)
        i += 1

    f.suptitle(plot_title, weight='bold')
    plt.tight_layout()
    # [...]

    return results
```

2.3.2. Algorytm centroidów (k-średnich, *k-means*)

Algorytm *k-means* zastosowano w programie wykorzystując funkcję *KMeans* z modułu *sklearn.cluster* (z biblioteki *scikit-learn*). Wartość parametru k (*n_clusters*) – liczby klastrów, grup, na które zostaną podzielone rekordy w zbiorze danych – określono jako 3 – biorąc pod uwagę, że kwiaty objęte zbiorem danych mogą należeć do jednego z trzech gatunków irysów (*Setosa*, *Versicolor* lub *Virginica*).

Ponadto, parametr *init* – określający sposób doboru początkowych centroidów – został określony w pierwszej kolejności jako *random* – losowy wybór początkowych centroidów – a następnie jako *k-means++* – początkowe centroidy zostają rozmieszczone jak najdalej od siebie.

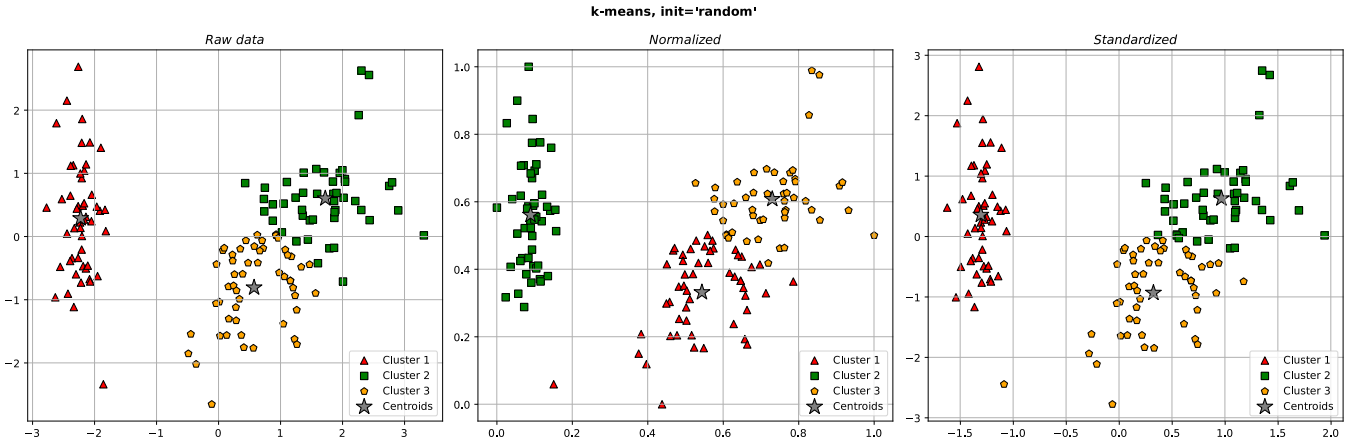
Pozostałe parametry algorytmu – *n_init*, *max_iter*, *tol*, *random_state* – zostały określone w wartościach domyślnych, zgodnie z odpowiednią dokumentacją biblioteki *scikit-learn*[1].

- Algorytm *k-means* z parametrem *init* określonym jako *random*:

```
km = KMeans(n_clusters=3, init='random', n_init=10, max_iter=300, tol=1e-04,
            random_state=0)
km_results = plot_fit_data_loop(X_data_sets, "k-means, init=random", "km")
# [...]
```

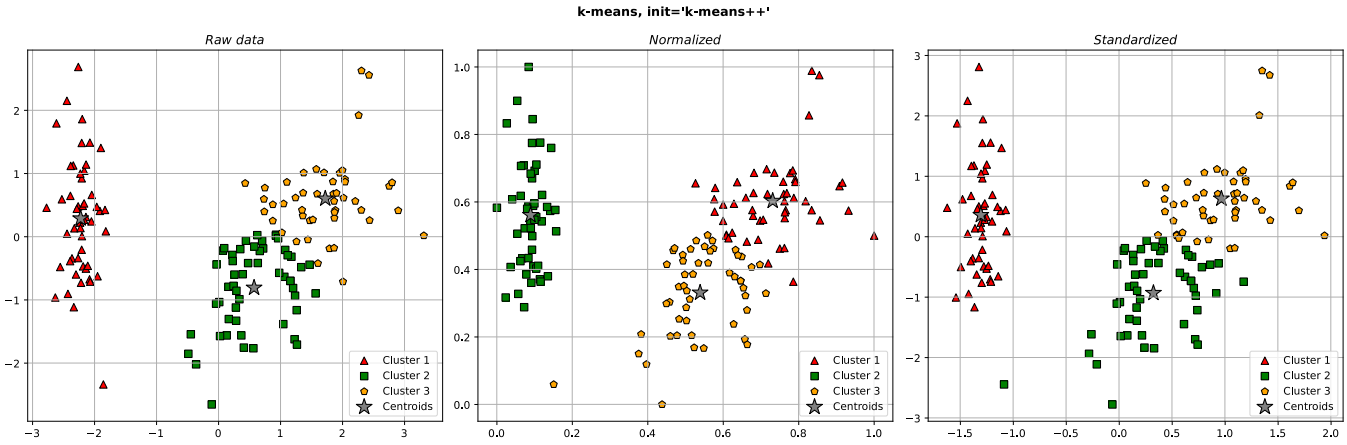
[1] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html [dostęp: 06.05.2023 r.].

```
plt.show()
```



- Algorytm *k-means* z parametrem *init* określonym jako *k-means++*:

```
km.set_params(init='k-means++')
km_pp_results = plot_fit_data_loop(X_data_sets, "k-means, init=k-means++", "km")
# [...]

plt.show()
```



2.3.3. Algorytm gęstościowej klasteryzacji przestrzennej z uwzględnieniem szumu (*density-based spatial clustering of applicaions with noise,* DBSCAN)

W algorytmie DBSCAN grupowanie (analiza skupień) punktów (próbek, rekordów) odbywa się w oparciu obszary zagęszczenia tych punktów – na podstawie tego zagęszczenia każdemu punktowi jest przyporządkowywana określona etykieta klastra (jest on przyporządkowywany do określonej grupy; w realizowanym projekcie: do określonego gatunku irysów).

Gęstość w tym wypadku jest rozumiana jako liczba punktów znajdujących się w określonym promieniu ε (od wybranego punktu). Jeżeli w tak określonym promieniu od wybranego punktu znajduje się również określona minimalna liczba innych punktów, to taki punkt jest uznawany za punkt rdzeniowy (*core point*). Jeżeli w promieniu od wybranego punktu znajduje się mniejsza liczba punktów niż określona minimalna

liczba, ale on sam znajduje się w promieniu punktu rdzeniowego, to jest on określany mianem punktu granicznego (*border point*). Jeśli natomiast punkt nie jest ani punkem rdzeniowym, ani punktem granicznym, to jest on uznawany za szum (punkt zaszumienia; *noise point*).

W oparciu o tak dokonaną klasyfikację punktów algorytm DBSCAN wyodrębnia skupienie (tworzy klaster) dla każdego punktu rdzeniowego (lub grupy połączonych punktów rdzeniowych, tj. znajdujących się względem siebie w promieniu ε). Do tak utworzonych klastrów odpowiednio przyporządkowywane są punkty graniczne – graniczne względem punktów rdzeniowych, znajdujących się w tych klastrach. Pozostałe punkty są natomiast traktowane jako szum (nie są przyporządkowywane do żadnego z klastrów).

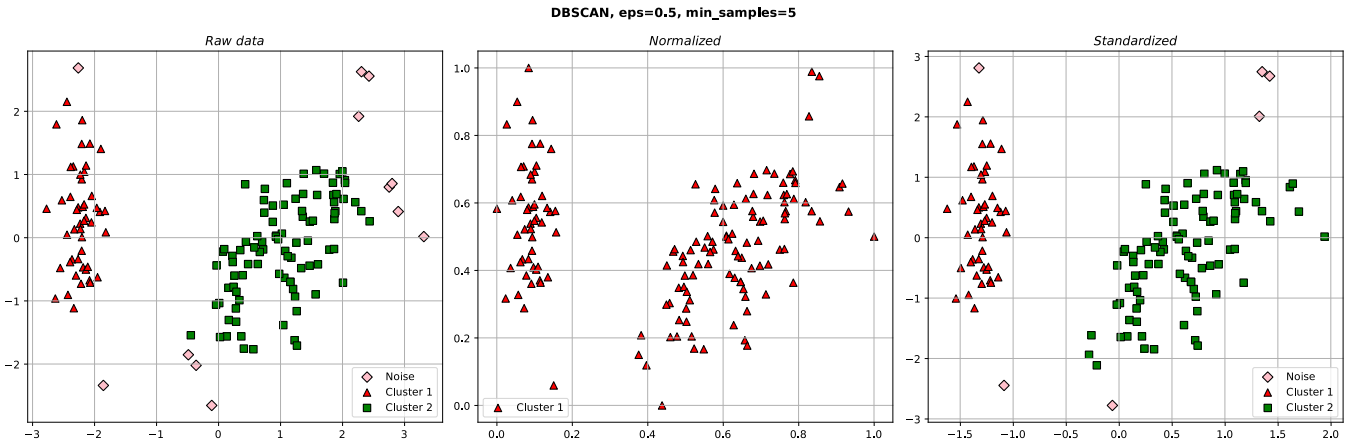
W przypadku algorytmu DBSCAN nie występują założenia dotyczące kulistości skupień (tak jak ma to miejsce w przypadku np. algorytmu *k-means*) – wyodrębniane w ramach tego algorytmu klastry, zwizualizowane przy pomocy dwuwymiarowego wykresu punktowego, mogą przyjmować inne kształty niż kuliste bądź eliptyczne.

Algorytm DBSCAN zastosowano w programie wykorzystując funkcję *DBSCAN*, również pochodzącą z modułu *sklearn.cluster* (z biblioteki *scikit-learn*). Należało przy tym określić długość promienia ε – parametr *eps* – oraz minimalną liczbę punktów, w oparciu o którą wyodrębnia się punkt rdzeniowy – parametr *min_samples*. Ponadto wykorzystano (domyślną) metrykę euklidesową (*metric='euclidean'*) do wyliczania odległości między punktami.

W przypadku algorytmu DBSCAN nie określa się uprzednio (jako parametru jego zastosowania), na ile grup mają zostać podzielone rekordy w zbiorze danych – jest to określane automatycznie, w ramach działania algorytmu – na podstawie zagęszczenia punktów (próbek).

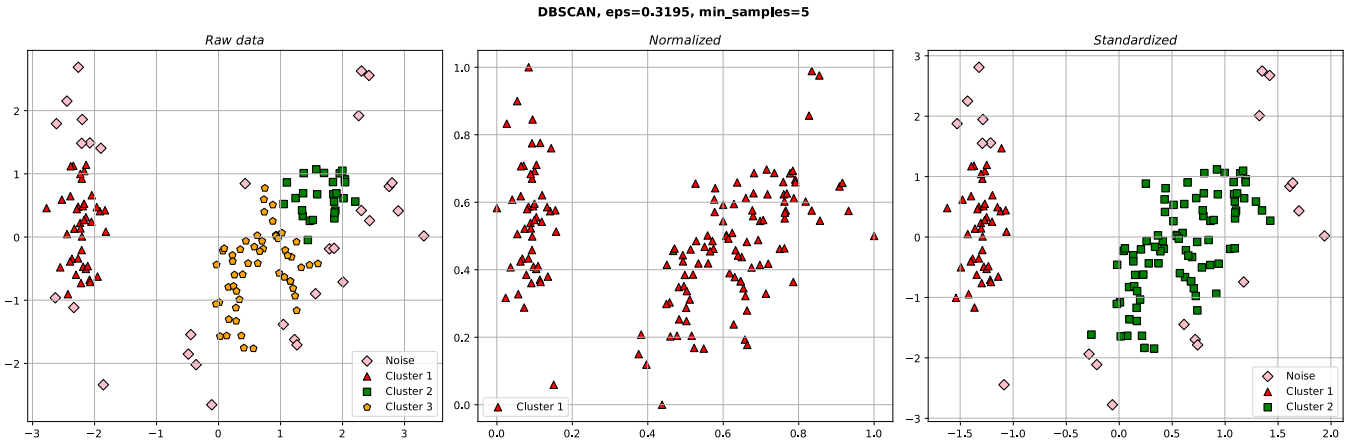
- Algorytm DBSCAN z parametrem *eps* i *min_samples* o wartościach domyślnych, tj. (odpowiednio) 0,5 oraz 5:

```
db = DBSCAN(eps=0.5, min_samples=5, metric='euclidean')
db_results = plot_fit_data_loop(X_data_sets, "DBSCAN, eps=0.5, min_samples=5", "db")
# [...]

plt.show()
```



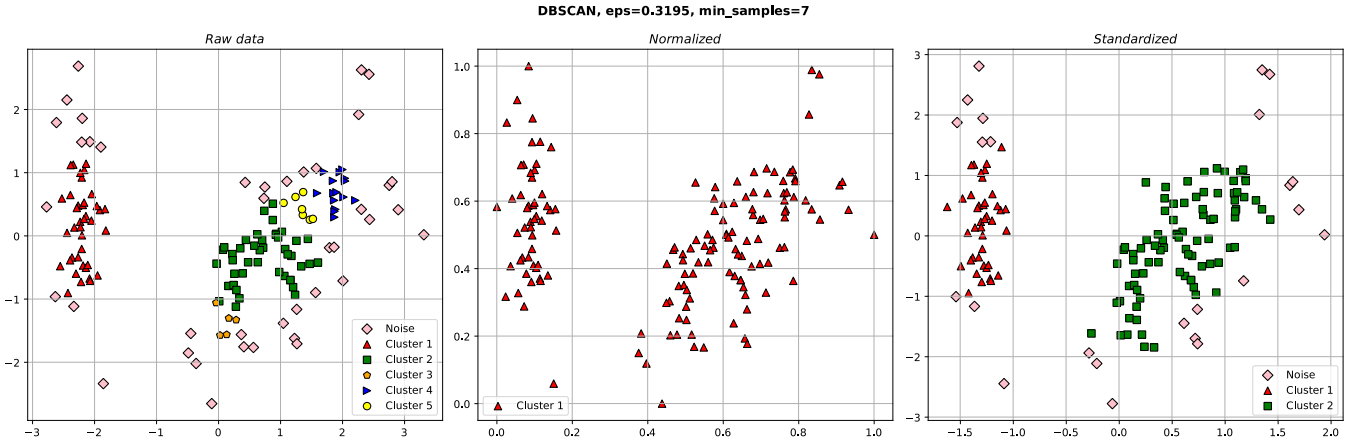
- Algorytm DBSCAN z parametrem *eps* o wartości 0,3195 i *min_samples* równym 5:

```
db.set_params(eps=0.3195)
db_eps_results = plot_fit_data_loop(X_data_sets, "DBSCAN, eps=0.3195, min_samples=5",
                                    "db")
# [...]

plt.show()
```



- Algorytm DBSCAN z parametrem *eps* o wartości 0,3195 i *min_samples* równym 7:

```
db.set_params(eps=0.3195, min_samples=7)
db_eps_min_samp_results = plot_fit_data_loop(X_data_sets, "DBSCAN, eps=0.3195, "
                                                          "min_samples=7", "db")
# [...]

plt.show()
```

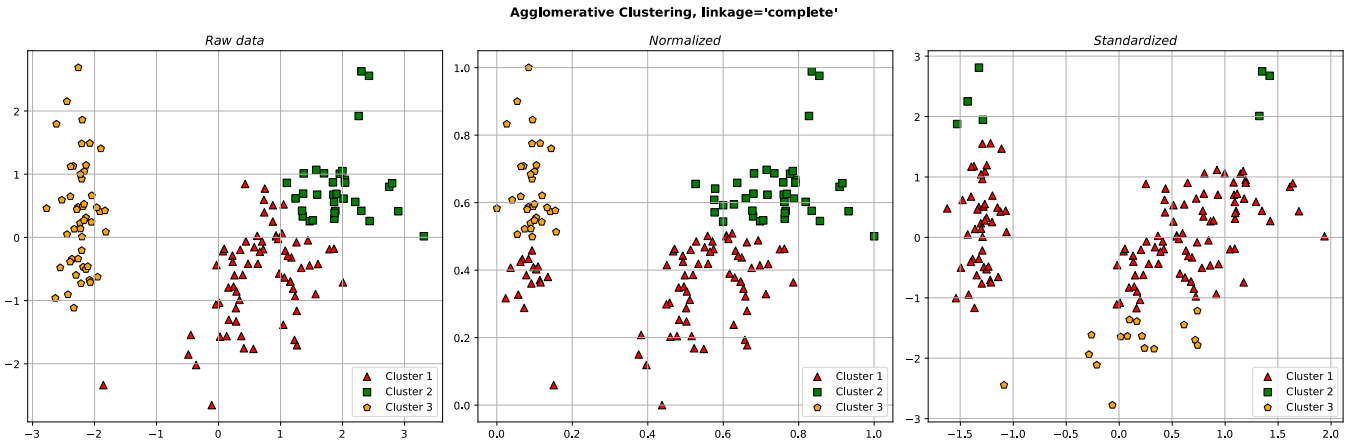


2.3.1. Algorytm hierarchiczny – klasteryzacji aglomeracyjnej (*agglomerative clustering*)

Dodatkowo w projekcie zastosowano algorytm *agglomerative clustering*, wykorzystując do tego funkcję *AgglomerativeClustering*, również z modułu *sklearn.cluster* (z biblioteki *scikit-learn*). Tak samo jak w

przypadku algorytmu *k-means*, wartość parametru *n_cluster* – liczby klastrów, które zostaną wydorębnione – została określona jako 3. Do mierzenia odległości między punktami wykorzystano metrykę euklidesową (*metric='euclidean'*), a jako sposób aglomeracji klastrów wybrano metodę pełnego wiązania (*linkage='complete'*).

```
ac = AgglomerativeClustering(n_clusters=3, metric='euclidean', linkage='complete')
ac_results = plot_fit_data_loop(X_data_sets, "Agglomerative Clustering, "
                                             "linkage='complete'", "ac")

plt.show()
```



2.4. Weryfikacja poprawności wybranych algorytmów grupujących

2.4.1. Uwagi ogólne

Weryfikacja poprawności (rezultatów zastosowania) algorytmów grupujących może zostać dokonana z dwóch perspektyw:

- perspektywy wewnętrznej – ewaluacja wewnętrzna – w ramach której weryfikacja grupowania dokonanego w wyniku zastosowania algorytmu jest dokonywana w oparciu o same grupowane dane (bez wykorzystania danych „zewnętrznych”),
- perspektywy zewnętrznej – ewaluacja zewnętrzna – w ramach której dokonane grupowanie jest oceniane w oparciu o dane zewnętrzne, np. informacje o klastrach, do których poszczególne próbki powinny były zostać przyporządkowane.

W ramach projektu dokonano weryfikacji poprawności algorytmów grupujących *k-means,* DBSCAN oraz *agglomerative clustering* z perspektywy zewnętrznej, wykorzystując do tego zbiór „irisORG.csv” – zawarte w tym zbiorze (w kolumnie *variety*) informacje o tym, do jakiego gatunku irysów należy określony kwiat (zob. p. 2.1 wyżej). Dokonana weryfikacja miała zatem charakter ewaluacji zewnętrznej.

W oparciu o przedmiotowe dane (i wykonane grupowania) dokonano oceny:

- „Czystości” (*purity*) klastrów będących wynikiem wykonanego grupowania – tj. oceny, w jakim stopniu wyodrębnione klastry zawierają próbki należące do jednej grupy.
- Zgodności klastrów z prawdziwymi klasami (*rand index*) – tj. oceny, w jakim stopniu klastry wyodrębnione w wyniku zastosowania algorytmów są podobne do klasyfikacji wzorcowej.

2.4.2. Wyliczenie wskaźnika *purity*

Aby wyliczyć wskaźnik *purity* dla każdego z wyodrębnionych (w wyniku zastosowania algorytmu grupującego) klastrów, należy policzyć największą liczbę punktów należących do jednej grupy (w oparciu o dane zewnętrzne) – liczbę punktów należących do grupy najczęściej występującej w danym klastrze. Następnie takie wartości należy zsumować dla wszystkich klastrów i podzielić przez całkowitą liczbę próbek.

Powyższego dokonano w ramach projektu poprzez wykorzystanie funkcji *contingency_matrix* z modułu *sklearn.metrics.cluster* (z biblioteki *scikit-learn*). Wskazana funkcja jako parametry przyjmuje dane zewnętrzne (wskazujące prawidłowe grupowanie próbek) oraz dane (grupowania) uzyskane w wyniku zastosowania algorytmu grupującego. W rezultacie jej zastosowania uzyskuje się informację o tym, ile próbek z określonych („prawdziwych") grup (wskazanych w wierszach) mieści się w poszczególnych klastrach (wymienionych w kolumnach).

```
matrix = contingency_matrix(labels_true, labels_pred)
print(matrix)
```

```
[[50  0  0]
 [ 0 11 39]
 [ 0 36 14]]
```

Z powyższej macierzy wynika, że: w pierwszym wyodrębnionym klastrze znajduje się 50 próbek należących do jednej grupy; w drugim klastrze – 11 próbek należących do drugiej grupy i 36 próbek należących do trzeciej grupy; w trzecim klastrze znajduje się 39 próbek należących do drugiej grupy i 14 próbek należących do trzeciej grupy (łączna liczba próbek: 150).

Następnie, przy pomocy zdefiniowanej funkcji *purity_score* zsumowano liczbę punktów występujących najczęściej w poszczególnych klastrach (w przykładzie powyżej: 50 + 36 + 39) i podzielono przez łączną liczbę próbek (150).

```
def purity_score(labels_true, labels_pred):
    matrix = contingency_matrix(labels_true, labels_pred)
    return np.sum(np.amax(matrix, axis=0)) / np.sum(matrix)
```

2.4.3. Wyliczenie wskaźnika zgodności klastrów z prawdziwymi klasami (*rand index*)

Oceny zgodności klastrów z prawdziwymi klasami dokonuje się poprzez zliczenie przypadków *true positive* i *true negative* i stosunku ich liczby do wszystkich przypadków łącznie (poza wymienionymi, także *false positive* i *false negative*).

Poszczególne przypadki ocenia się w oparciu o pary punktów (próbek): jeżeli dwie próbki, należące do jednej grupy zgodnie z danymi zewnętrznymi (wzorcowymi) zostały przyporządkowane do jednego klastra w wyniku zastosowania algorytmu, to występuje przypadek *true positive*; analogicznie w sytuacji, gdy próbki należące do różnych grup zostały przyporządkowane przez algorytm do różnych klastrów – jest to przypadek *true negative.* Nie ma przy tym znaczenia, do jakich klastrów poszczególne próbki są przyporządkowywane – istotna jest ocena ich grupowania względem siebie (w ramach poszczególnych par próbek – czy należą do tego samego klastra, czy do różnych klastrów).

W celu wyliczenia wskaźnika *rand index* posłużono się funkcją *rand_score* z modułu *sklearn.metrics* (z biblioteki *scikit-learn*). Podobnie jak *contingency_matrix,* przedmiotowa funkcja jako parametry przyjmuje dane zewnętrzne, wskazujące prawidłowe grupowanie próbek oraz dane (grupowania) uzyskane w wyniku zastosowania algorytmu grupującego.

```
rand_score(labels_true, labels_pred)
```

3. Prezentacja wyników

Do wyliczenia przedstawionych powyżej wartości wykorzystano zdefiniowaną funkcję własną *score_loop*

```
def score_loop(data_sets, score_title, labels_true, results):
    i = 0
    for data_set_name in data_sets:
        print(score_title + ": " + data_set_name + ": Purity [%%]: %.2f" %
              (purity_score(labels_true, results[i]) * 100))
        print(score_title + ": " + data_set_name + ": Rand index [%%]: %.2f" %
              (rand_score(labels_true, results[i]) * 100))
        print()
        i += 1


# [...]
score_loop(X_data_sets, "K-means, init='random'", y_true, km_results)
score_loop(X_data_sets, "K-means, init='k-means++'", y_true, km_pp_results)
score_loop(X_data_sets, "DBSCAN, eps=0.5, min_samples=5", y_true, db_results)
score_loop(X_data_sets, "DBSCAN, eps=0.3195, min_samples=5",  y_true, db_eps_results)
score_loop(X_data_sets, "DBSCAN, eps=0.3195, min_samples=7",  y_true,
           db_eps_min_samp_results)
score_loop(X_data_sets, "Agglomerative Clustering, linkage='complete'",  y_true,
           ac_results)
```

Uzyskane wartości wskaźników *purity* oraz *rand index* (zaokrąglone do dwóch miejsc po przecinku) przedstawiono poniżej w tabeli – odrębnie dla poszczególnych algorytmów grupujących (*k-means*, DBSCAN, oba z odpowiednimi parametrami, oraz *agglomerative clustering*) oraz typów danych poddanych działaniu algorytmów („surowe dane”, znormalizowane oraz ustandaryzowane).

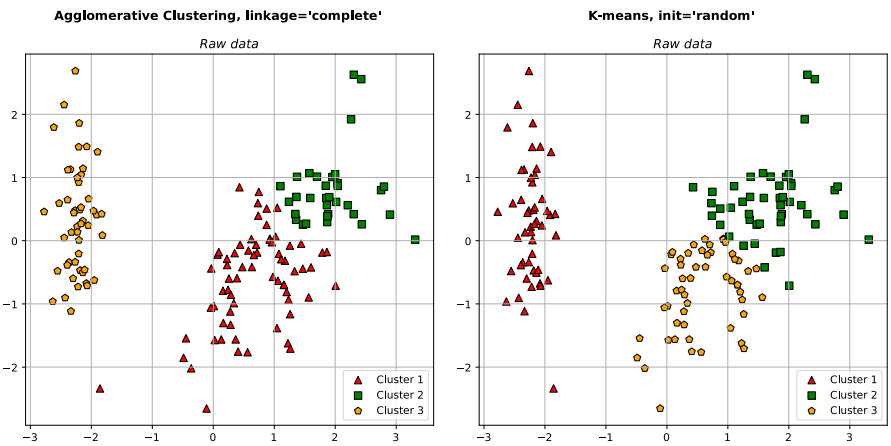
	Algorithm	***Parameters***	***Data***	***Purity [%]***	***Rand index [%]***
1.	*k-means*	*init='random'*	*Raw data*	83.33	83.22
2.			*Normalized*	82.67	82.33
3.			*Standardized*	81.33	81.47
4.		*init='k-means++'*	*Raw data*	83.33	83.22
5.			*Normalized*	83.33	82.79
6.			*Standardized*	81.33	81.47
7.	*DBSCAN*	*eps=0.5, min_samples=5*	*Raw data*	68.00	76.73
8.			*Normalized*	33.33	32.89
9.			*Standardized*	66.67	76.73

	Algorithm	***Parameters***	***Data***	***Purity [%]***	***Rand index [%]***
10.	*DBSCAN (cd.)*	*eps=0.3195, min_samples=5*	*Raw data*	76.67	76.64
11.			*Normalized*	33.33	32.89
12.			*Standardized*	64.67	74.20
13.		*eps=0.3195, min_samples=7*	*Raw data*	74.00	73.41
14.			*Normalized*	33.33	32.89
15.			*Standardized*	64.00	73.34
16.	*Agglomerative clustering*	*linkage='complete'*	*Raw data*	84.00	83.11
17.			*Normalized*	68.67	67.06
18.			*Standardized*	40.67	42.88

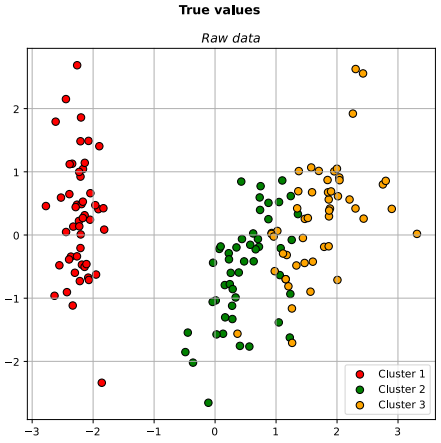
4. Komentarze i wnioski

W oparciu o uzyskane wyniki można wskazać, że:

- **Najlepsze wyniki udało się uzyskać w rezultacie zastosowania algorytmu *k-means*** (zarówno z parametrem *init='random'*, jak i *init='k-means++'*) **oraz algorytmu *agglomerative clustering*,** w obu przypadkach w odniesieniu do „surowych danych" (*raw data*):
  - w przypadku algorytmu *k-means* udało się uzyskać *rand index* o wartości 83,22% (oraz *purity* w wysokości 83,33%),
  - w przypadku algorytmu *agglomerative clustering* udało się uzyskać *purity* w wysokości 84% (oraz *rand index* o wartości 83,11%).
- W oparciu o wykresy punktowe wygenerowane dla zastosowania tych algorytmów można wskazać, że algorytm *agglomerative clustering* przyporządkował więcej punktów (próbek) do klastra numer 1 („łączącego się" z klastrem numer 2), podczas gdy algorytm *k-means* podzielił próbki między tymi klastrami bardziej równomiernie. Może to przekładać się na większą „czystość" zastosowania algorytmu *agglomerative clustering*, a jednocześnie mniejszą wartość *rand index* (zbyt dużo próbek, które powinny być w różnych klastrach, przyporządkowana do tego samego klastra).



- **Najgorsze rezultaty (wskaźniki) uzyskano w przypadku zastosowania algorytmu DBSCAN** – przy użyciu parametrów *eps* o wartości 0.3195, i *min_samples* w wysokości 5 (w odniesieniu do „surowych danych") udało się uzyskać co najwyżej *purity* w wysokości 76,67% i *rand index* o wartości 76,64%. Na wskazane rezultaty mogło wpłynąć to, że poszczególne próbki występują w dużym zagęszczeniu, pomiędzy dwoma klastrami (spośród trzech) nie występuje wyraźna granica, a trzeci z klastrów znajduje się w dużo większym oddaleniu od dwóch pozostałych. Okoliczności te ilustruje poniższy wykres punktowy, sporządzony w oparciu o informacje z kolumny *variety* ze zbioru „irisORG.csv" – wskazujące, do jakiego gatunku irysów należy określony kwiat.



- Na powyższe rezultaty nie wpłynęły zmiany parametrów stosowanego algorytmu DBSCAN – *eps* lub *min_samples*. Ich zwiększanie lub zmniejszanie prowadziło bądź to do wyodrębnienia zbyt małej liczby klastrów (tak było w przypadku *eps=0.5* i *min_samples=5*), bądź też do wyodrębniania ich zbyt dużej liczby (tak przy *eps=0.3195* i *min_samples=7*)[2].
- **Normalizacja i standaryzacja danych ze zbioru „iris2D.csv" nie pozwoliła na osiągnięcie lepszych rezultatów** niż bezpośrednie zastosowanie wartości z tego zbioru. Przeciwnie: zastosowanie znormalizowanych lub ustandaryzowanych danych prowadziło najczęściej do uzyskania gorszych rezultatów niż w przypadku „surowych danych" (tak było w przypadku algorytmów DBSCAN i *agglomerative clustering*) bądź też, co najwyżej, do podobnych rezultatów (algorytm *k-means*). Przyczyną takiego stanu rzeczy mogło być to, że „surowe dane" zastosowane w projekcie zostały już wcześniej ustandaryzowane (przed ich kompresją); późniejsza normalizacja lub (kolejna) standaryzacja mogły spowodować pogorszenie „jakości" danych w tym znaczeniu, że stały się one trudniejsze w interpretacji dla algorytmów grupujących („przedobrzono").
- Zastosowanie algorytmu DBSCAN z parametrami przyjętymi w projekcie w przypadku danych znormalizowanych w ogóle nie pozwoliło na wyodrębnienie więcej niż jednego klastra. W odniesieniu do tych danych – a więc takich danych, w których współrzędne każdej próbki mieszczą się w zakresie od 0 do maksymalnie 1 – przyjęte wartości parametru *eps,* choć i tak ułamkowe, mogły być jednak zbyt duże.

[2] Jak wskazują S. Raschka i V. Mirjalili, *„Znalezienie dobrej kombinacji tych [eps i min_samples] parametrów może być kłopotliwe w przypadku, gdy różnice w gęstości zestawu danych są względnie duże"* (tychże, *Machine learning i deep learning. Biblioteki scikit-learn i TensorFlow 2*, wyd. 3, Gliwice 2021, s. 367).

Dopiero zmiana tego parametru do wartości np. równego 0.1 mogłaby pozwolić na wyodrębnienie większej liczby klastrów.

- **Ocenianie działania algorytmów grupujących z perspektywy „zewnętrznej”** – tj. posiadając dane zewnętrzne, wskazujące na prawidłowe grupowanie próbek, do których zastosowano algorytmy grupujące – **może budzić pewne wątpliwości.** Grupowanie jest bowiem techniką uczenia nienadzorowanego – a w sytuacji, gdy dysponujemy informacją o prawidłowym grupowaniu określonych próbek, stosowanie do nich algorytmów grupujących może być bezcelowe. Dyskusyjne może być bowiem grupowanie czegoś, co zostało już pogrupowane (o pogrupowaniu czego mamy już wiedzę)[3].
- Celowości takiego działania można jednak szukać np. w sytuacji, gdy dysponujemy bardzo dużą liczbą próbek, a tylko w odniesieniu do części z nich posiadamy informację o prawidłowym grupowaniu – w takim wypadku na takiej części próbek może zostać sprawdzone zastosowanie określonych algorytmów, z określonymi parametrami. Chociaż sam dobór takich „zweryfikowanych” próbek i ich reprezentatywność dla całości zbioru może także rodzić kolejne problemy.
- Na podstawie ostatniego, przedstawionego powyżej wykresu punktowego, ilustrującego właściwe grupowanie próbek, do których stosowano algorytmy grupujące w ramach projektu, można stwierdzić, że **żaden z zastosowanych algorytmów nie pozwolił na osiągnięcie „właściwego” grupowania –** i to pomimo faktu, że każdy z nich osiągnął wysokie (powyżej 80%), , jak się wydaje, wartości wskaźników *purity* i *rand index*. Jest to w szczególności widoczne przy podziale dwóch „stykających się” klastrów, między którymi granica przebieg wzdłuż osi Y (w rezultacie każdego zastosowanego algorytmu granica ta przebiegała po skosie).

5. Źródła

[b.a.], *Cluster analysis - Wikipedia,* https://en.wikipedia.org/wiki/Cluster_analysis [dostęp: 08.05.2023 r.]

[b.a.], *Matplotlib – Visualization with Python*, https://matplotlib.org/, [dostęp: 08.05.2023 r.]

[b.a.], *pandas - Python Data Analysis Library*, https://pandas.pydata.org/, [dostęp: 08.05.2023 r.]

[b.a.], *scikit-learn: machine learning in Python – scikit-learn 1.2.2 documentation*, https://scikit-learn.org/stable/, [dostęp: 08.05.2023 r.]

S. Raschka, V. Mirjalili, *Python. Machine learning i deep learning. Biblioteki scikit-learn i TensorFlow 2*, wyd. 3, Gliwice 2021.

[3] *„Additionally, from a knowledge discovery point of view, the reproduction of known knowledge may not necessarily be the intended result”* ([b.a.], *Cluster analysis - Wikipedia,* https://en.wikipedia.org/wiki/Cluster_analysis#External_evaluation [dostęp: 08.05.2023 r.].