

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КІЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
Кафедра математичних методів системного аналізу

ЗВІТ

про виконання лабораторної роботи № 3
з дисципліни «Інтелектуальний аналіз даних»

Виконала:

студентка 3 курсу
групи КА-91
Оніщенко Д. С.

Перевірила:

Недашківська Н. І.

Київ – 2021

Умова задачі

1. Представити початкові дані графічно.
2. Побудувати модель кластеризації згідно з варіантом.
3. Виконати кластеризацію даних на основі моделі.
4. Представити розбиття на кластери графічно, наприклад, різними коліорами.
5. Розрахувати додаткові результати кластеризації (згідно з варіантом).
6. Побудувати декілька альтернативних моделей:
 - шляхом зміни значень параметрів основної моделі,
 - використати різні функції відстані,
 - задати різні значення кількості кластерів, в алгоритмах де кількість кластерів - параметр.
7. Дляожної альтернативної моделі розрахувати метрики якості кластеризації, що реалізовані в класі metrics (тільки метрики згідно з варіантом):
 - Estimated Number of Clusters.
 - Adjusted Rand Index.
 - Adjusted Mutual Information.
 - Homogeneity.
 - Completeness.
 - V-measure.
 - Silhouette Coefficient.
 - Calinski-Harabasz Index.
 - Davies-Bouldin index.
 - Contingency Matrix.
8. Виконати аналіз результатів кластеризації одним з неформальних методів (тільки методом згідно з варіантом):

- чи є розбиття стабільним на підвибірках даних,
- чи є розбиття стабільним після видалення окремих об'єктів,
- чи є розбиття стабільним після зміни порядку об'єктів у множині об'єктів,
- чи існує взаємозв'язок між результатами кластеризації і змінними, які не враховувалися при кластеризації,
- чи можна інтерпретувати результати кластеризації.

9. Зробити висновки про якість роботи моделей на досліджених даних. До слідити різні значення параметрів основної моделі, різні функції відстані та різну кількість кластерів в алгоритмах, де кількість кластерів слугує параметром.

10. Оцінити результати кластеризації на основі метрик якості та на основі неформальних методів. Для кожного набору даних вибрати найкращу модель.

Варіант 6

6. Алгоритм Affinity propagation. Відобразити графічно центри кластерів.

Метрики якості: Estimated number of clusters, Calinski-Harabasz Index, Davies-Bouldin index, Contingency Matrix.

Чи є розбиття стабільним після вилучення окремих об'єктів?

Початкові дані:

- (a) `sklearn.datasets.make_moons`
- (b) `from sklearn.datasets import make_blobs`
`X, y = make_blobs(n_samples=500,`
`n_features=2,`
`centers=4,`
`cluster_std=1,`
`center_box=(-10.0, 10.0),`
`shuffle=True,`
`random_state=1)`

Теоретичні відомості

Нехай від x_1 до x_n є набір точок даних без будь-яких припущення щодо їх внутрішньої структури, а s — функція, яка кількісно визначає подібність між будь-якими двома точками, так що $s(i, j) > s(i, k)$ якщо x_i більше схожий на x_j ніж на x_k . Для цього прикладу використовувався негативний квадрат відстані двох точок даних, тобто для точок x_i і x_k , $s(i, k) = -||x_i - x_k||^2$

Діагональ s (тобто $s(i, i)$) є особливо важливою, оскільки вона представляє перевагу екземпляра, тобто ймовірність того, що конкретний екземпляр стане зразком. Коли для всіх вхідних даних встановлено однакове значення, він контролює, скільки класів створює алгоритм. Значення, близьке до мінімально можливої подібності, створює менше класів, тоді як значення, близьке або більше максимально можливої подібності, створює багато класів. Зазвичай він ініціалізується до медіанної подібності всіх пар вхідних даних.

Алгоритм продовжує чергувати два кроки передачі повідомлень, які оновлюють дві матриці:

- Матриця «відповідальності» R має значення $r(i, k)$, які кількісно визначають, наскільки добре підходить x_k , щоб служити прикладом для x_i , порівняно з іншими прикладами-кандидатами для x_i .
- Матриця «доступності» A містить значення $a(i, k)$, які представляють, наскільки «відповідним» було б для x_i вибрати x_k як приклад, беручи до уваги перевагу інших точок щодо x_k як прикладу.

Обидві матриці ініціалізуються усіма нулями, і їх можна розглядати як логарифмічні таблиці ймовірностей. Потім алгоритм ітеративно виконує такі оновлення:

- По-перше, оновлення відповідальності надсилаються навколо:
$$r(i, k) \leftarrow s(i, k) - \max\{a(i, k') + s(i, k')\} k! = k'$$
- Потім доступність оновлюється за:

$$a(i, k) \leftarrow \min \left(0, r(k, k) + \sum_{i' \notin \{i, k\}} \max(0, r(i', k)) \right)$$

$$a(k, k) \leftarrow \sum_{i' \neq k} \max(0, r(i', k))$$

Ітерації виконуються до тих пір, поки або межі кластера не залишаються незмінними протягом певної кількості ітерацій, або доки не буде досягнуто певної заздалегідь визначені кількості (ітерацій). Приклади витягаються з остаточних матриць як ті, чия «відповідальність + доступність» для них є позитивною (тобто $(r(i, i) + a(i, i)) > 0$)

Коли варто використовувати:

У вас не дуже великий ($N < 10^5 - 10^6$) або в міру великий, але розжижений ($N < 10^6 - 10^7$) датасет

Заздалегідь відома функція близькості

Ви очікуєте побачити множину кластерів різної форми із можливою трохи різною кількістю елементів, що варіюються

Ви готові повозитися з постобробкою

Складність елементів датасету не має значення

Властивості функції близькості значення не мають

Calinski-Harabasz Index, також відомий як критерій відношення дисперсії, являє собою відношення суми дисперсії між кластерами та міжкластерної дисперсії для всіх кластерів, чим вищий бал, тим кращі результати.

$$\frac{SS_B}{SS_W} \times \frac{N - k}{k - 1}$$

Де k – кількість кластерів, а N – загальна кількість спостережень (точок даних), SSW – це загальна дисперсія всередині кластера (еквівалентна загальній сумі квадратів, розрахованої нижче), SSB – загальна дисперсія між кластерами .

Ви можете обчислити SSB , використовуючи загальну суму квадратів (tss) мінус SSW . Для даного набору даних загальна сума квадратів (tss) є квадратом відстані всіх точок даних від центру набору даних, цей показник не залежить від кількості кластерів.

Немає «прийнятного» граничного значення. Ви просто порівнюєте значення СН на око. Чим вище значення, тим «кращим» є рішення. Якщо на лінійному графіку значень СН з'являється, що одне рішення дає пік або принаймні різке лікоть, виберіть його. Якщо, навпаки, лінія плавна - горизонтальна або висхідна, або низхідна - то немає підстав віддавати перевагу одному рішенням іншим.

$$\sum_{k=1}^g \sum_{j=1}^n u_{kj}^2 d^2(c_k, \bar{i})$$

Міжкластерні відмінності
(є великими при оптимальному g)

\bar{i} – середнє арифметичне вхідних об'єктів

$$\sum_{k=1}^g \sum_{j=1}^n u_{kj}^2 d^2(c_k, i_j)$$

Відмінності всередині кластеру
(є малими при оптимальному g)

Переваги

Оцінка вища, коли кластери щільні та добре розділені, що відноситься до стандартної концепції кластера.

Оцінка швидко обчислюється.

Недоліки

Індекс Калінського-Харабаша, як правило, вищий для опуклих кластерів, ніж інші концепції кластерів, наприклад кластери на основі щільності, подібні до тих, які отримані за допомогою DBSCAN.

Критерій СН найбільш прийнятний у випадку, коли кластери більш-менш сферичні та компактні в своїй середині (наприклад, нормальну розподілені). За інших рівних умов СН, як правило, віддає перевагу кластерним рішенням з кластерами, що складаються приблизно з однакової кількості об'єктів.

`sklearn.metrics.calinski_harabasz_score(X, labels)`

Davies-Bouldin index. індекс, визначений як симетричний і невід'ємний. Через те, як воно визначається, як функція відношення розкиду всередині кластера, до поділу між кластерами, **нижче значення означатиме, що кластеризація краща.**

$$\bar{R} = \frac{1}{N} \sum_{i=1}^N R_i$$

де

N : загальна кількість кластерів

$R_{ij}=S_i+S_j M_{ij}$ де:

S_i : внутрішньокластерна дисперсія кластера i

S_j : внутрішньокластерна дисперсія кластера j

M_{ij} : відстань між центрами скучень i і j

В основному тут ми обчислюємо подібність між кластерами як суму двох внутрішньокластерних дисперсій, поділених на міру поділу.

Чим більше R_{ij} , тим більше подібних кластерів i і j . Ви, напевно, вже можете сказати, що найкращий можливий випадок, коли ці показники в цілому є якомога меншими.

$R_i \equiv \max_{j \neq i} (R_{ij})$

маючи: $i \neq j$ Для кожного кластера i ми знаходимо найвищий коефіцієнт з усіх розрахованих R_{ij} .

Наприклад, для кластера $i=1$ ми обчислили R_{11} , R_{12} і R_{13} . Продовжуйте з R_{12} і R_{13} (оскільки R_{11} не задовольняє обмеження $i \neq j$, і немає сенсу порівнювати кластер з самим собою).

Виявляється, середня подібність між кожним кластером та його найбільш подібним, усереднена за всіма кластерами, . Це підтверджує ідею, що жоден кластер не повинен бути схожим на інший, і, отже, найкраща схема кластеризації по суті мінімізує індекс Девіса-Болдіна. Цей індекс, визначений таким чином, є середнім за всіма кластерами, і, отже, хорошим показником для визначення кількості

кластерів у даних є побудова графіка щодо кількості кластерів, для яких він розраховується. Число i , для якого це значення є найнижчим, є хорошим показником кількості кластерів, на які можна ідеально класифікувати дані. Це має застосування при визначенні значення k в алгоритмі kmeans, де значення k невідоме попередньо.

Хід роботи

```
# create datasets
X_moons, y_moons = make_moons(n_samples=500, noise=0.15, random=1)
)
X_blobs, y_blobs = make_blobs(n_samples=500,
                               n_features=2,
                               centers=4,
                               cluster_std=1,
                               center_box=(-10.0, 10.0),
                               shuffle=True,
                               random_state=1)
```

Кількість кластерів значення не має

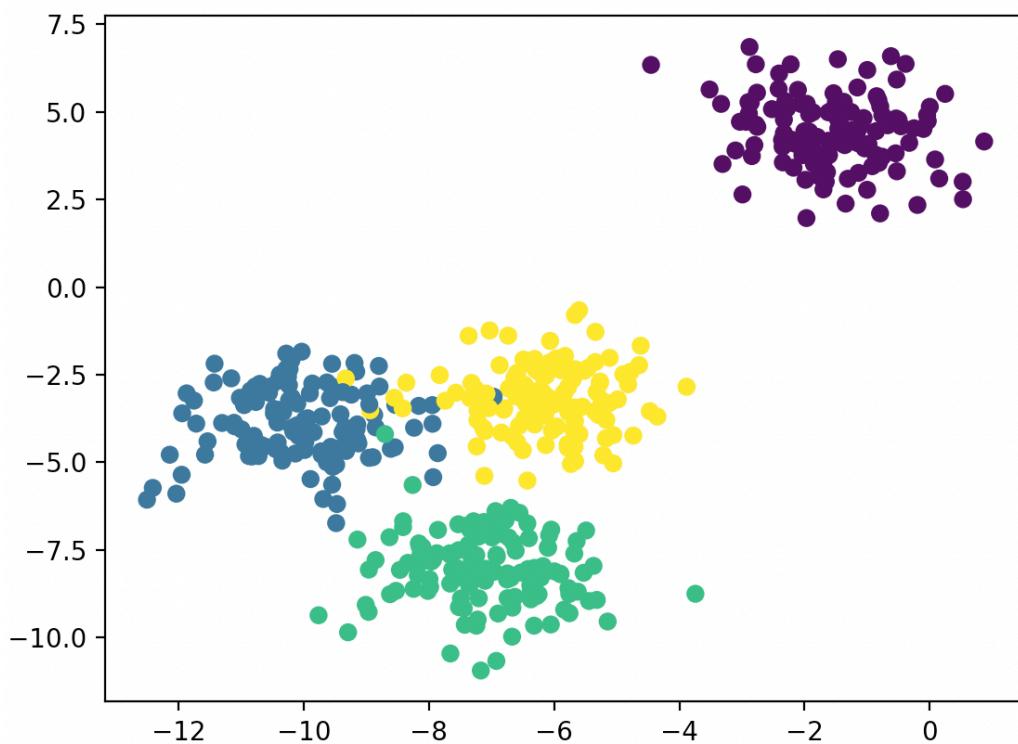
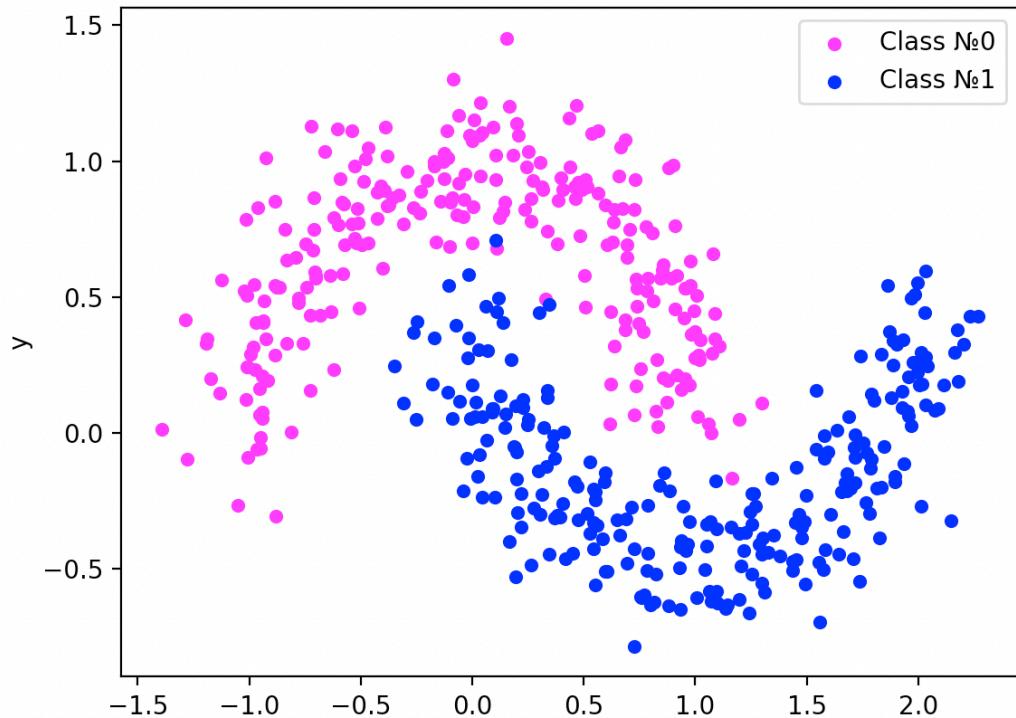
Недоліки алгоритму:

1. Хоча алгоритм не потребує задавати кількість центрів кластерів заздалегідь, він повинен встановити опорний рівень заздалегідь, а розмір опорного ступеня позитивно корелює з числом центрів кластерів;
2. Оскільки алгоритму AP необхідно оновлювати значення привабливості та значення атрибуції кожної точки даних на кожній ітерації, складність алгоритму відносно висока і для роботи з великим обсягом даних потрібно багато часу.

1) представити початкові дані графічно

```
# show dataset
def show_moons(X, y):
    df = DataFrame(dict(x=X[:, 0], y=X[:, 1], label=y))
    colors = {0: 'magenta', 1: 'blue'}
    fig, ax = plt.subplots()
    grouped = df.groupby('label')
    for key, group in grouped:
```

```
group.plot(ax=ax, kind='scatter', x='x', y='y', label='Class №' + str(key),  
color=colors[key])  
plt.show()  
show_moons(X_moons, y_moons)
```



```
plt.scatter(X_blobs[:, 0], X_blobs[:, 1], c=y_blobs)  
plt.show()
```

2) побудувати модель кластеризації згідно з варіантом

```
# create models:  
model_moons = AffinityPropagation().fit(X_moons)  
model_blobs = AffinityPropagation().fit(X_blobs)
```

3) виконати кластеризацію на основі моделі

+4) представити розбиття на кластери графічно

+5) Розрахувати додаткові результати кластеризації: відобразити графічно центри кластерів

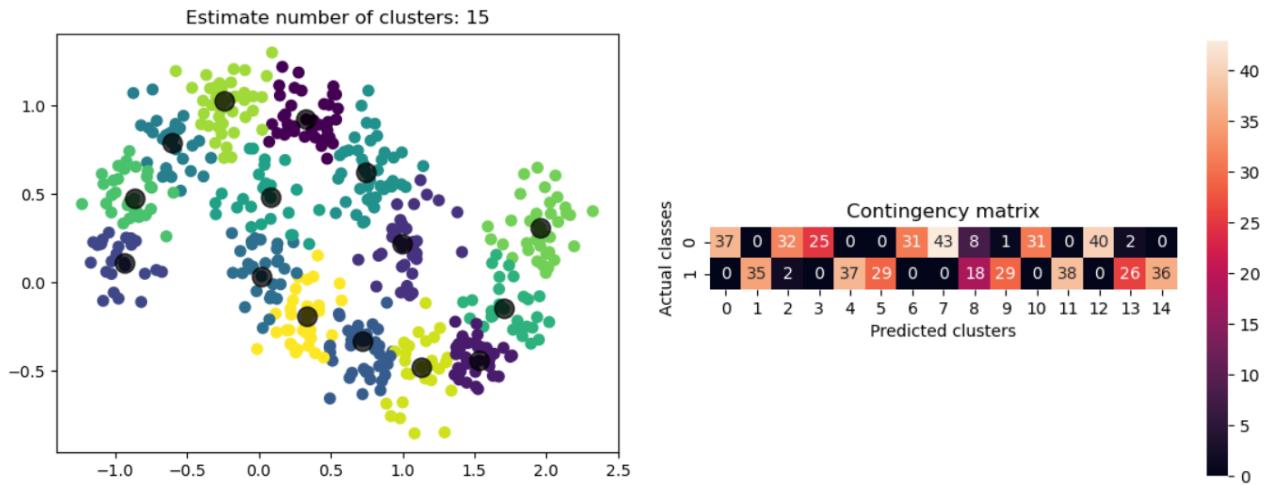
+7) Для кожної альтернативної моделі розрахувати метрики якості кластеризації, що реалізовані в класі metrics :

- Calinski-Harabasz Index.
- Davies-Bouldin index.
- Contingency Matrix.

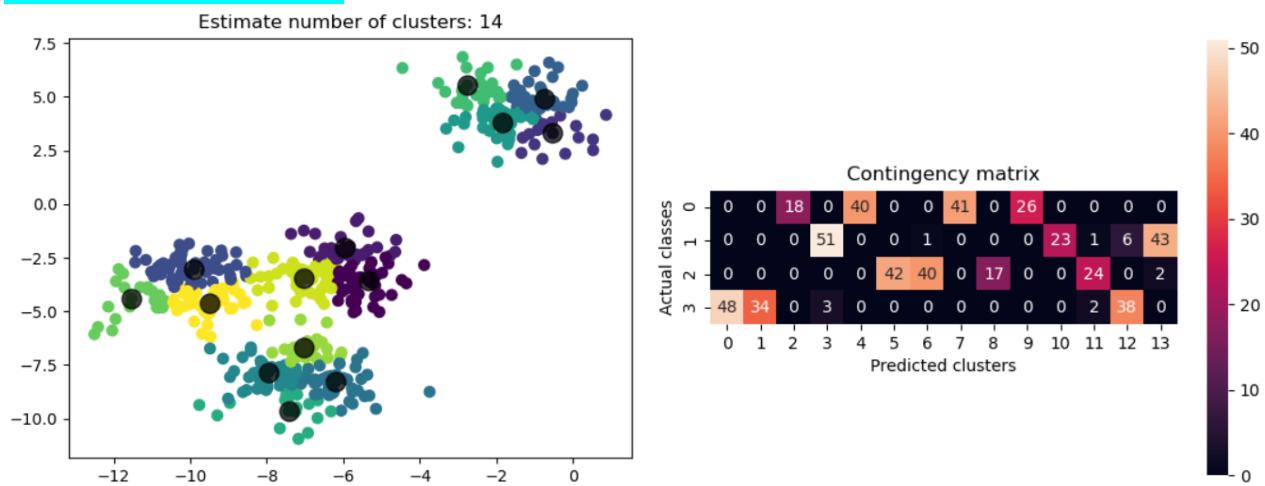
```
def show_clusters(model, X, y_pred):  
    plt.scatter(X[:, 0], X[:, 1], c=y_pred, s=45, cmap='viridis')  
    centers = model.cluster_centers_  
    plt.scatter(centers[:, 0], centers[:, 1], c='black', s=150, alpha=0.75)  
    plt.title(f'Estimate number of clusters: {len(centers)}')  
    plt.show()  
  
def create_matrix(y_true, y_predict):  
    cm = contingency_matrix(y_true, y_predict)  
    sns.heatmap(cm, annot=True, square=True)  
    plt.ylabel("Actual classes")  
    plt.xlabel("Predicted clusters")  
    plt.title("Contingency matrix")  
    plt.show()  
  
def get_scores(model, X, y_true):  
    y_predict = model.predict(X)  
    ch_index = calinski_harabasz_score(X, y_predict)  
    db_index = davies_bouldin_score(X, y_predict)  
    v_score = v_measure_score(y_true, y_predict)  
    num_clusters = len(model.cluster_centers_.indices_)  
    return y_predict, ch_index, db_index, v_score, num_clusters  
  
def metrics_plot(model, X, y_true):  
    y_predict, ch_index, db_index, _, _ = get_scores(model, X, y_true)  
    show_clusters(model, X, y_predict)  
    print(f'Calinski-harabasz index: {ch_index}')  
    print(f'Davies-bouldin index: {db_index}')  
    create_matrix(y_true, y_predict)
```

```
# show clusters  
print('* The first dataset *')  
metrics_plot(model_moons, X_moons, y_moons)  
print('* The second dataset *')  
metrics_plot(model_blobs, X_blobs, y_blobs)
```

* The first dataset *



* The second dataset *



Calinski-harabasz index: 1680.844203031026

Davies-bouldin index: 0.8857524394372467

Estimated number of clusters: 14

6) побудувати декілька альтернативних моделей:

- використати різні функції подібності

```
# change distances
new_preference1 = -1*npamax(euclidean_distances(X_moons))**1.2
new_preference2 = -1*npamax(euclidean_distances(X_blobs))**1.8

model_moons_change_pref =
AffinityPropagation(preference=new_preference1).fit(X_moons)
```

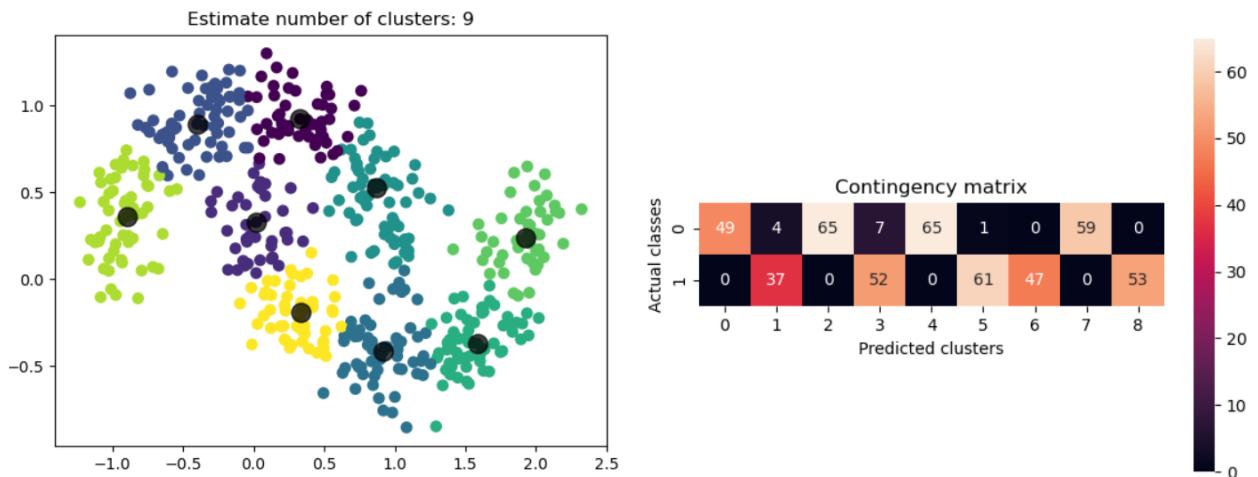
```

model_blobs_change_pref =
AffinityPropagation(preference=new_preference2).fit(X_blobs)

print('* The first dataset with changed preference *')
metrics_plot(model_moons_change_pref, X_moons, y_moons)
print('* The second dataset with changed preference *')
metrics_plot(model_blobs_change_pref, X_blobs, y_blobs)

```

* The first dataset with changed preference *

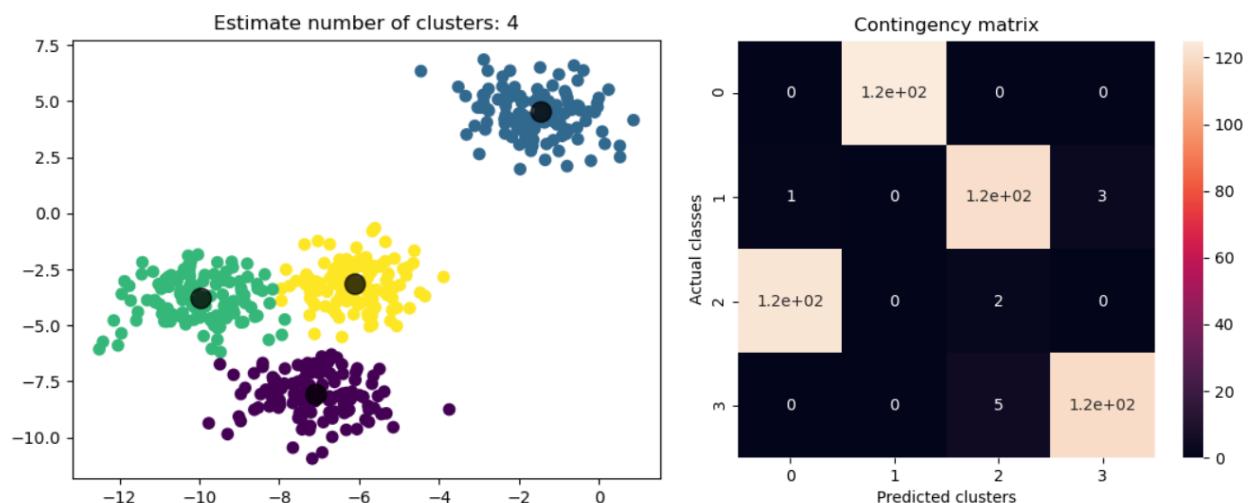


Calinski-harabasz index: 855.8723764188213

Davies-bouldin index: 0.7293172229026587

Estimated number of clusters: 9

* The second dataset with changed preference *



Calinski-harabasz index: 2704.4858735121097

Davies-bouldin index: 0.4776988389567719

Estimated number of clusters: 4

9) Зробити висновки про якість роботи моделей на досліджених даних. До слідити різні значення параметрів основної моделі, різні функції відстані та різну кількість кластерів в алгоритмах, де кількість кластерів слугує параметром.

```

def get_scores(model, X, y_true):
    y_predict = model.predict(X)
    ch_index = calinski_harabasz_score(X, y_predict)
    db_index = davies_bouldin_score(X, y_predict)
    v_score = v_measure_score(y_true, y_predict)
    num_clusters = len(model.cluster_centers_indices_)
    return y_predict, ch_index, db_index, v_score, num_clusters

def estimate_models(X, y_true, affinity, damping, preference):
    affinity_params = []
    v_scores = []
    ch_indices = []
    db_indices = []
    clusters_num = []

    for aff in affinity:
        for damp in damping:
            for pref in preference:
                try:
                    AP = AffinityPropagation(affinity=aff, damping=damp,
preference=pref).fit(X)
                    get_scores(AP, X, y_true),
                    ch_index, db_index, v_score, num_clusters =
affinity_params.append([aff, damp, pref])
                    ch_indices.append(ch_index)
                    db_indices.append(db_index)
                    v_scores.append(v_score)
                    clusters_num.append(num_clusters)

                except Exception as exc:
                    continue

    return pd.DataFrame({"params[affinity, damping, preference]":
affinity_params,
                      "V_scores": v_scores,
                      "cluster_number": clusters_num,
                      "CH_scores": ch_indices,
                      "DB_scores": db_indices})

```

```

# estimate models
affinity_check = ['euclidean', 'precomputed']
damping_check = [i / 10 for i in range(5, 10)]
preferences1 = [i for i in range(0, -14, -1)]
preferences2 = [i for i in range(-200, -300, -1)]

print('\n-----search for optimal parameters-----\n')

# first dataset
data_frame_moons = estimate_models(X_moons, y_moons, affinity_check,
damping_check, preferences1).\n
    set_index("params[affinity, damping, preference]")

```

```

data_frame_moons = data_frame_moons[data_frame_moons["cluster_number"] != 0]
print('\nMetrics of the first dataset', data_frame_moons)

data_frame_moons_ch = data_frame_moons.sort_values(['CH_scores'],
ascending=False)
param = data_frame_moons_ch.index.values[0]
print('\nThe best model on ch-score:\n', param)
best = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_moons)
print(metrics_plot(best, X_moons, y_moons))

data_frame_moons_db = data_frame_moons.sort_values(['DB_scores'])
param = data_frame_moons_db.index.values[0]
print('\nThe best model on db-score:\n', param)
best_db = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_moons)
print(metrics_plot(best_db, X_moons, y_moons))

data_frame_moons_v = data_frame_moons.sort_values(['V_scores'],
ascending=False)
param = data_frame_moons_v.index.values[0]
print('\nThe best model on v-score:\n', param)
best = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_moons)
print(metrics_plot(best, X_moons, y_moons))

# second dataset
data_frame_blobs = estimate_models(X_blobs, y_blobs, affinity_check,
damping_check, preferences2).\
    set_index("params[affinity, damping, preference]")
data_frame_blobs = data_frame_blobs[data_frame_blobs["cluster_number"] != 0]
print('\nMetrics of the second dataset', data_frame_blobs)

data_frame_blobs_ch = data_frame_blobs.sort_values(['CH_scores'],
ascending=False)
param = data_frame_blobs_ch.index[0]
print('\nThe best model on ch-score:\n', param)
best = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_blobs)
print(metrics_plot(best, X_blobs, y_blobs))

data_frame_blobs_db = data_frame_blobs.sort_values(['DB_scores'])
param = data_frame_blobs_db.index[0]
print('\nThe best model on db-score:\n', param)
best_db = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_blobs)
print(metrics_plot(best_db, X_blobs, y_blobs))

data_frame_blobs_v = data_frame_blobs.sort_values(['V_scores'],
ascending=False)
param = data_frame_blobs_v.head().index[0]
print('\nThe best model on v-score:\n', param)
best = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_blobs)
print(metrics_plot(best, X_blobs, y_blobs))

```

-----search for optimal parameters-----

Коефіцієнт демпфування (в діапазоні від 0,5 до 1) - це ступінь, в якій поточне значення підтримується щодо значень, що надходять (зважене 1-демпфування). Це зроблено для

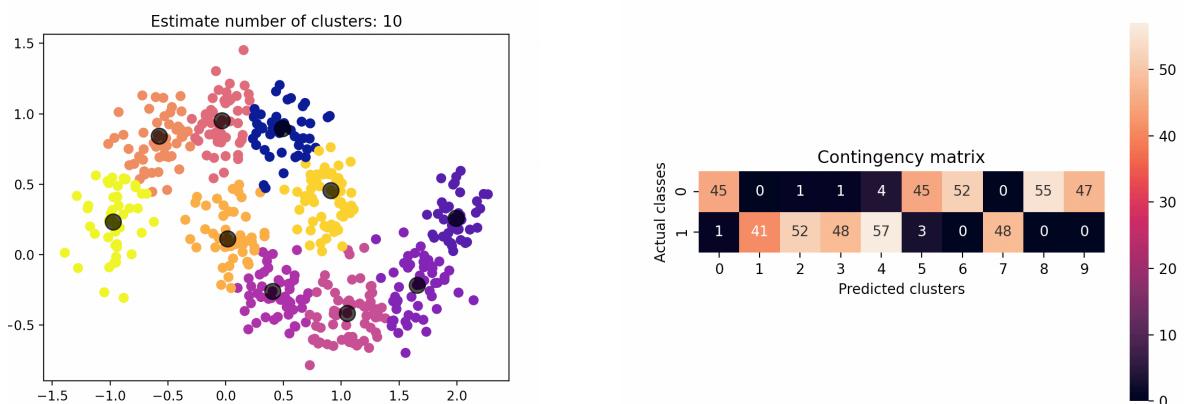
того, щоб уникнути числових коливань при оновленні цих значень (повідомлень).

Привілеї для кожної точки-точки з більшими значеннями привілеїв з більшою ймовірністю будуть обрані в якості прикладу. Кількість прикладів, тобто. кластерів, залежить від значення вхідних переваг.

First dataset:

The best model on ch-score= 939.9841243239389:

['euclidean', 0.8, -5]



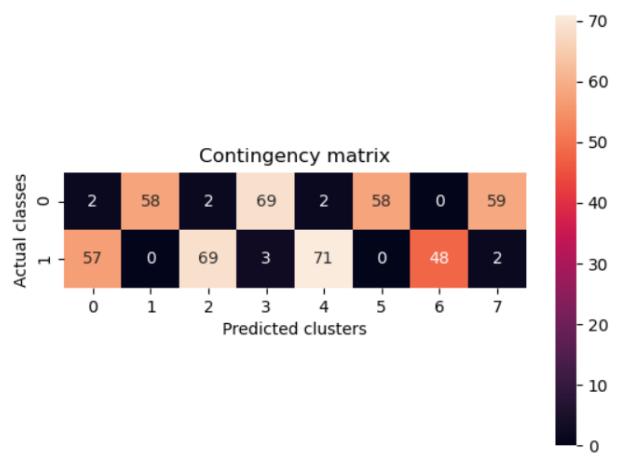
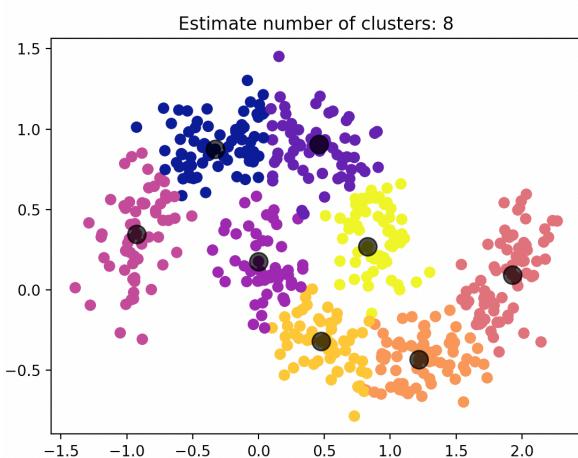
Calinski-harabasz index: 939.9841243239389

Davies-bouldin index: 0.7230999077735198

Estimated number of clusters: 10

The best model on db-score:

['euclidean', 0.6, -7]



Calinski-harabasz index: 866.8548506148842

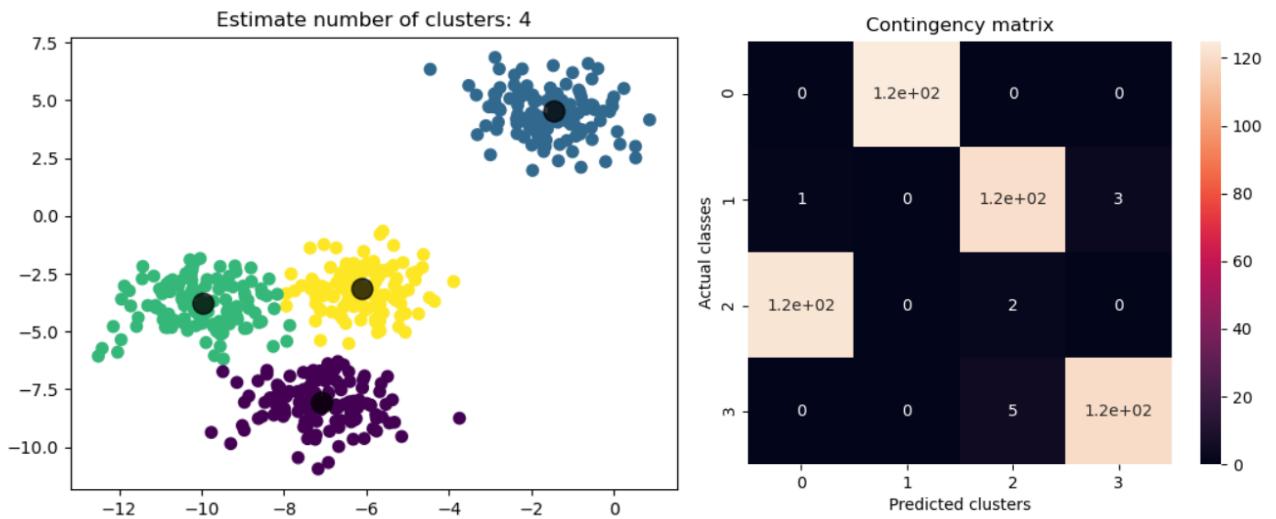
Davies-bouldin index: 0.6656809109335695

Estimated number of clusters: 8

Second dataset:

The best model on ch-score:

['euclidean', 0.7, -285]

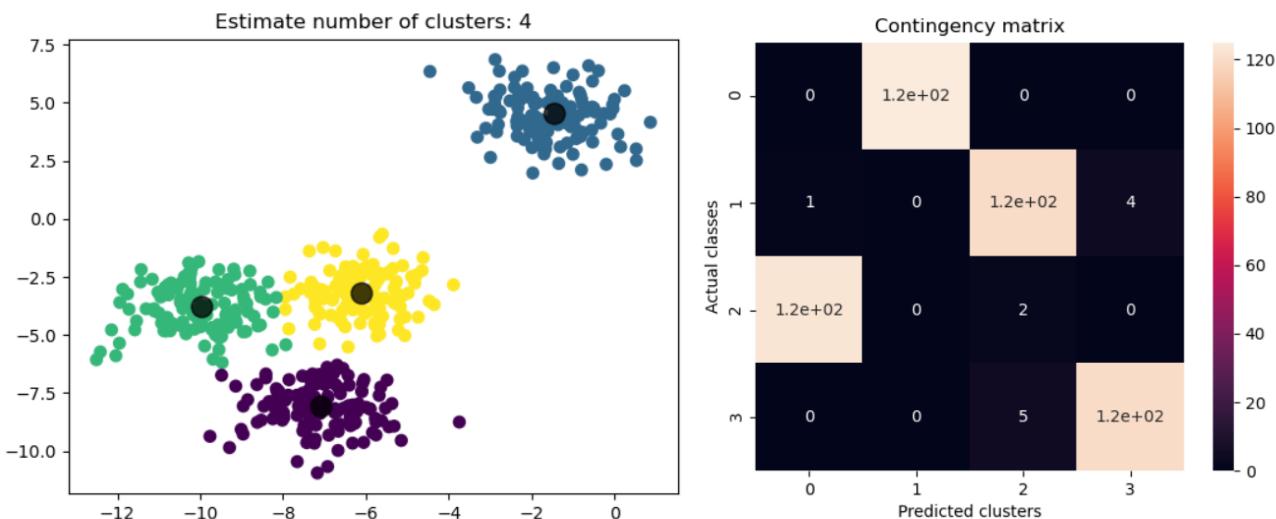


Calinski-harabasz index: 2704.4858735121097

Davies-bouldin index: 0.4776988389567719

The best model on ch-score:

['euclidean', 0.7, -285]



Calinski-harabasz index: 2704.4858735121097

Davies-bouldin index: 0.4776988389567719

8) Виконати аналіз результатів кластеризації одним з неформальних методів (тільки методом згідно з варіантом): чи є розбиття стабільним після видалення окремих об'єктів.

```
def estimate_size_quality(model, X, y_true):
    samples = [i/10 for i in range(1, 6, 1)]
    ch_scores = []
    bd_scores = []
    v_scores = []

    for i in samples:
        x_samples, _, y_samples, _ = train_test_split(X, y_true, test_size=i)
        _, ch, bd, v, _ = get_scores(model, x_samples, y_samples)
        ch_scores.append(ch)
        bd_scores.append(bd)
        v_scores.append(v)
        print('Metrics with ', i*100, '% data remove')
        metrics_plot(model, x_samples, y_samples)

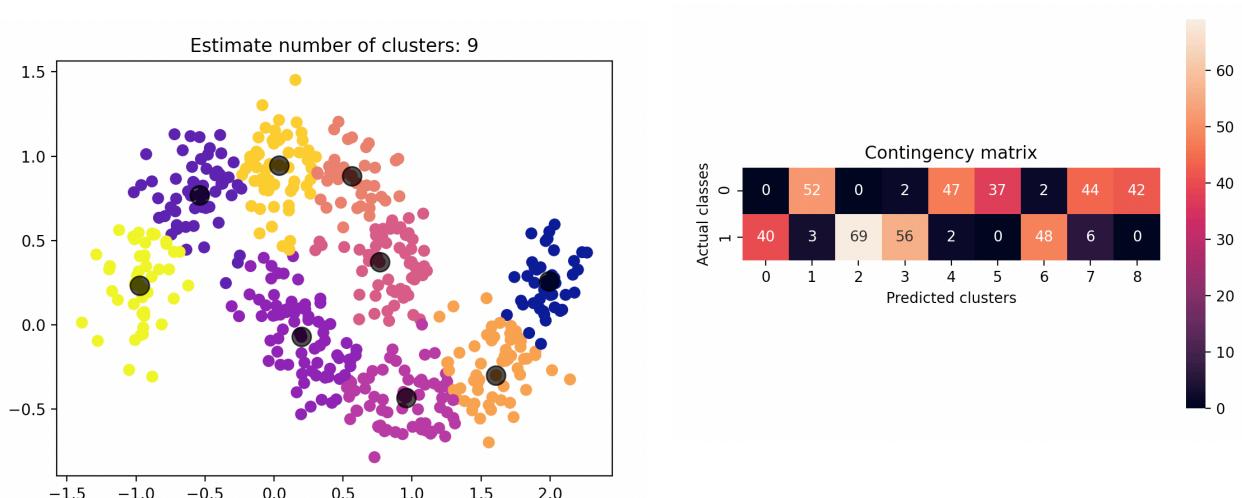
    samples = [i*10 for i in range(5, 10)]
    plt.plot(samples, ch_scores)
    plt.legend(['Calinsky_Harabasz_score'])
    plt.xlabel("% of data used for training")
    plt.show()

    plt.plot(samples, bd_scores)
    plt.plot(samples, v_scores)
    plt.legend(['Davies_Bouldin score', 'V score'])
    plt.xlabel("% of data used for training")
    plt.show()
```

```
estimate_size_quality(model_moons_change_pref, X_moons, y_moons)
estimate_size_quality(model_blobs_change_pref, X_blobs, y_blobs)
```

First dataset (для прикладу обрано найкращу за DB-score модель):

Metrics with 10.0 % data remove

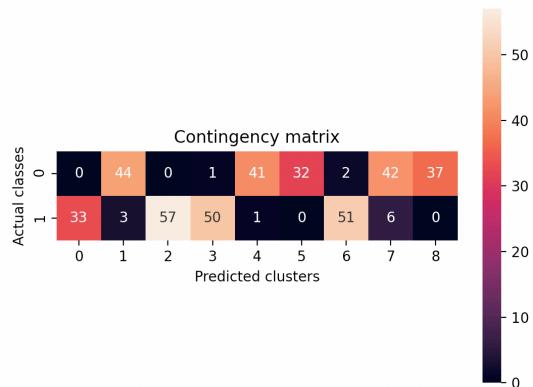
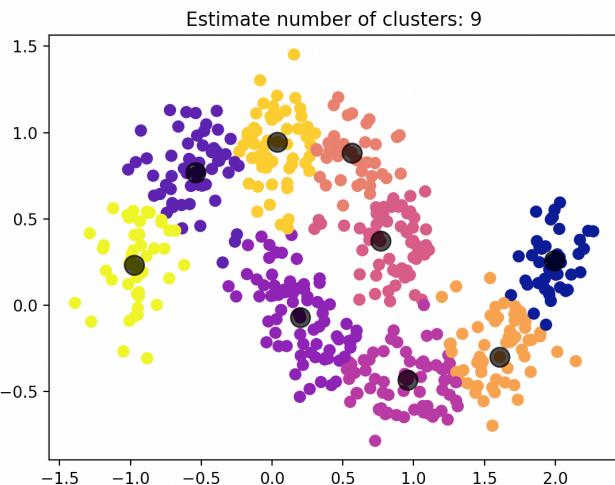


Calinski-harabasz index: 763.5622365755597

Davies-bouldin index: 0.7350080751424682

Estimated number of clusters: 9

Metrics with 20.0 % data remove

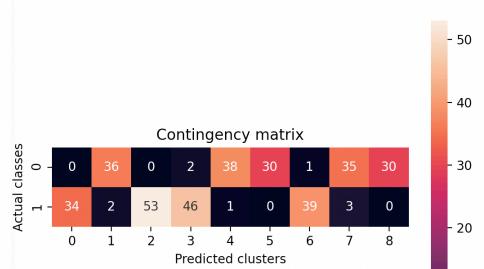
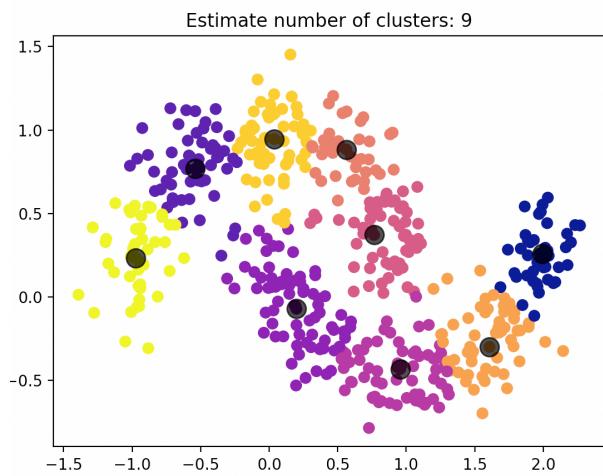


Calinski-harabasz index: 689.1146063161306

Davies-bouldin index: 0.7374682091773964

Estimated number of clusters: 9

Metrics with 30.0 % data remove

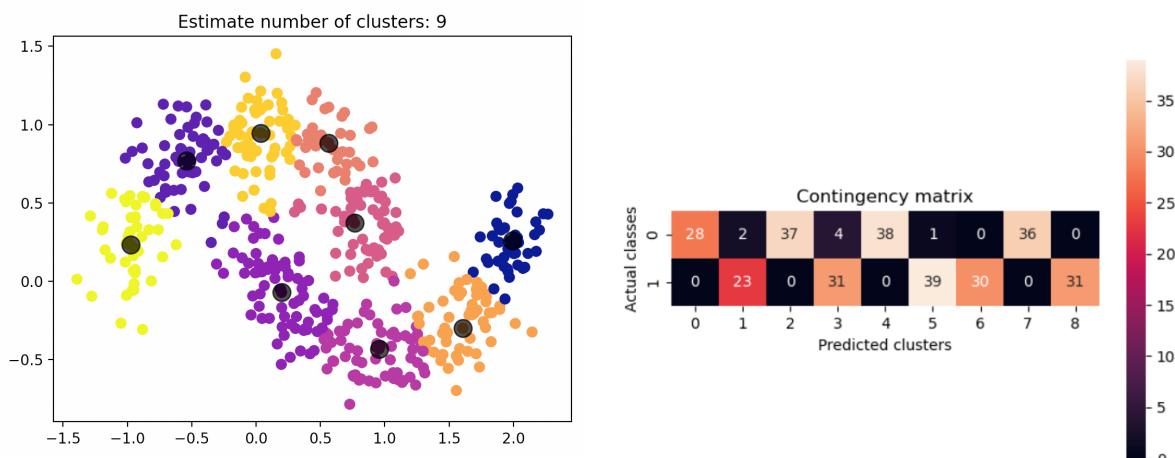


Calinski-harabasz index: 593.0728566070494

Davies-bouldin index: 0.7286024261232839

Estimated number of clusters: 9

Metrics with 40.0 % data remove

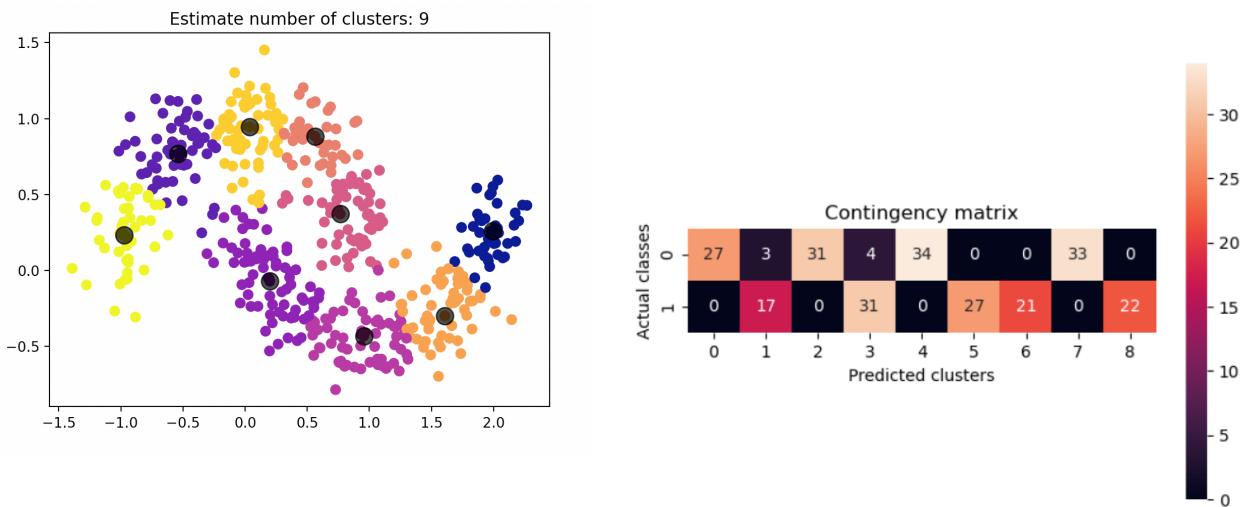


Calinski-harabasz index: 492.6076607222549

Davies-bouldin index: 0.7635705812825646

Estimated number of clusters: 9

Metrics with 50.0 % data remove

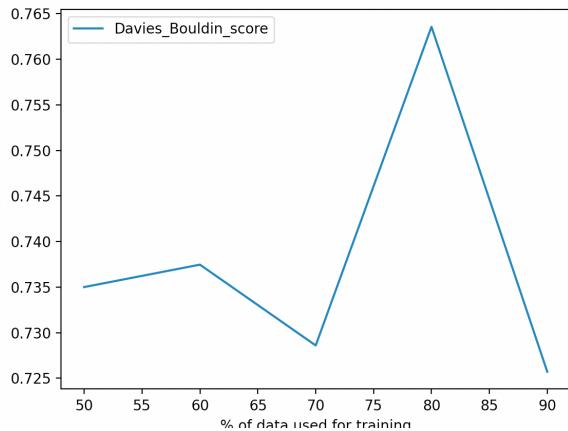
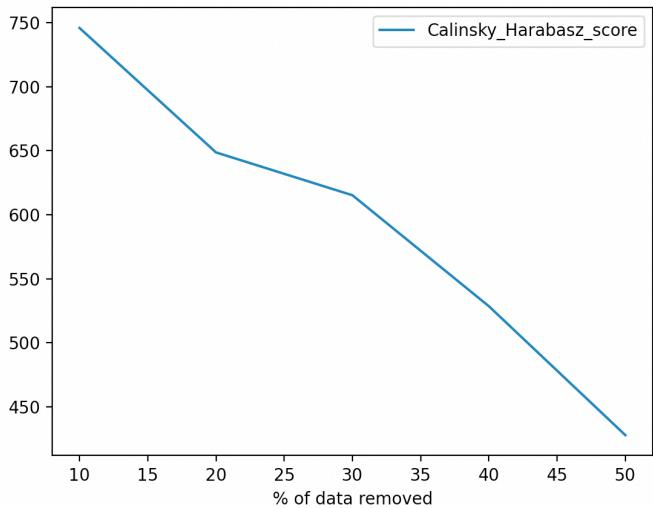


Calinski-harabasz index: 428.4710059936666

Davies-bouldin index: 0.7257135512101227

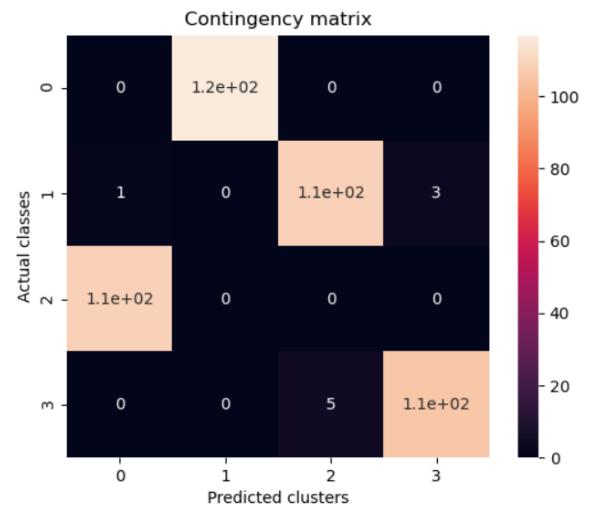
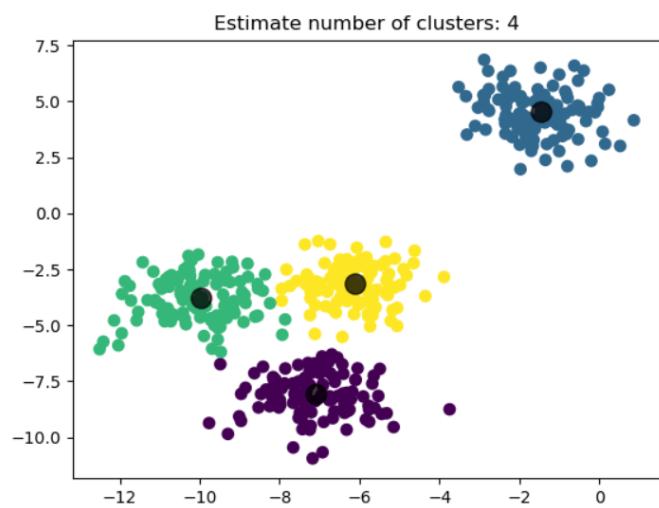
Estimated number of clusters: 9

Графіки метрик для першого датасету:



First dataset (для прикладу обрано найкращу за DB-score модель):

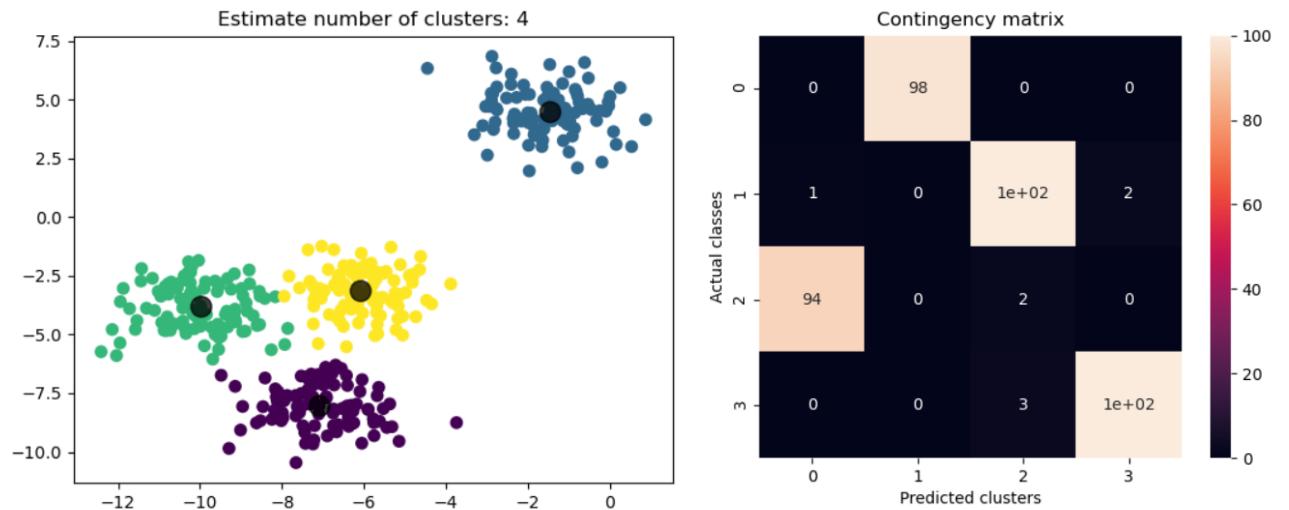
Metrics with 10.0 % data remove



Calinski-harabasz index: 2597.4645587941573

Davies-bouldin index: 0.4692451628043596

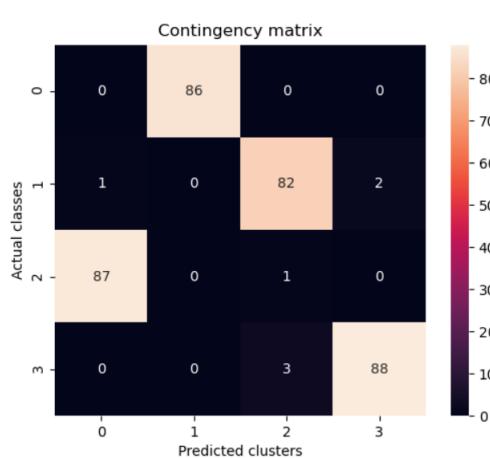
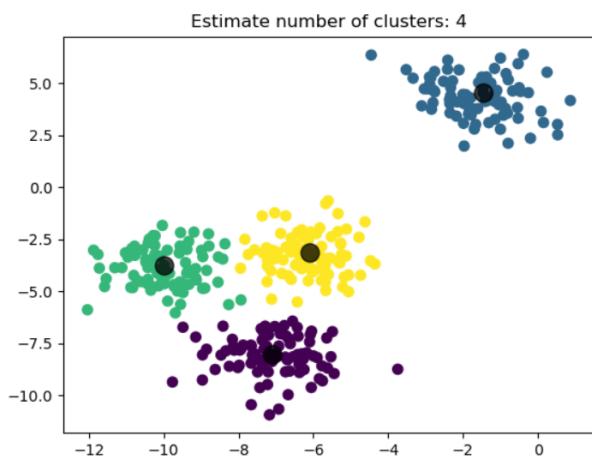
Metrics with 20.0 % data remove



Calinski-harabasz index: 2180.7321564942004

Davies-bouldin index: 0.4719433684719463

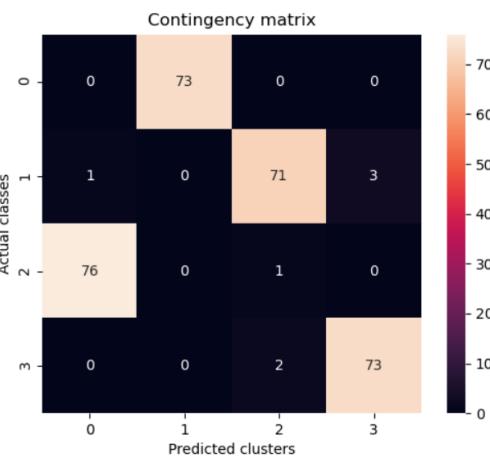
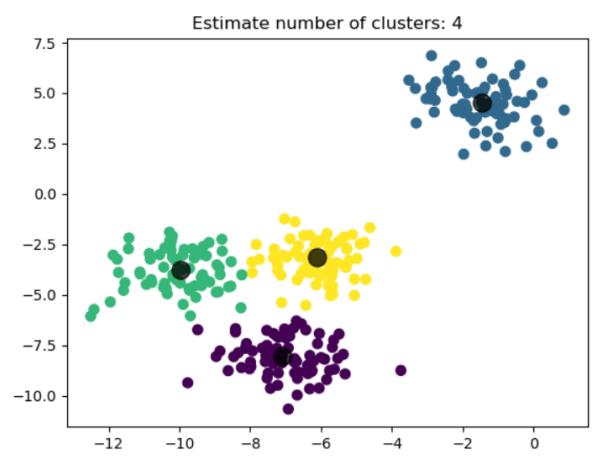
Metrics with 30.0 % data remove



Calinski-harabasz index: 1895.9051399200837

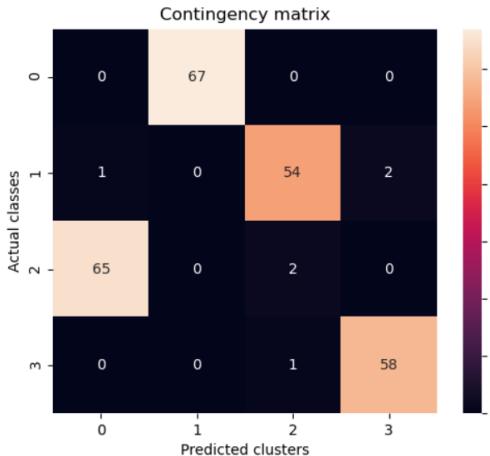
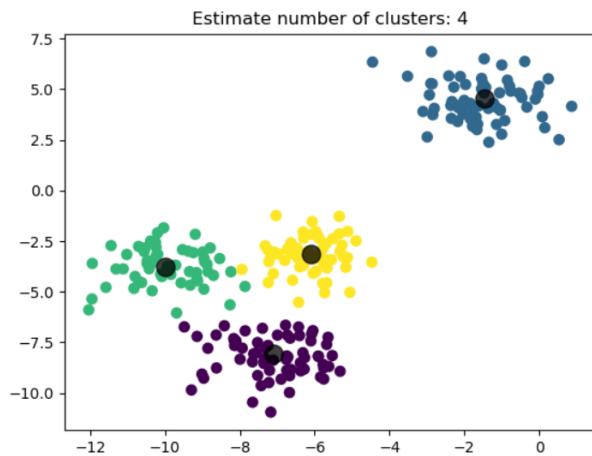
Davies-bouldin index: 0.47155913771006897

Metrics with 40.0 % data remove

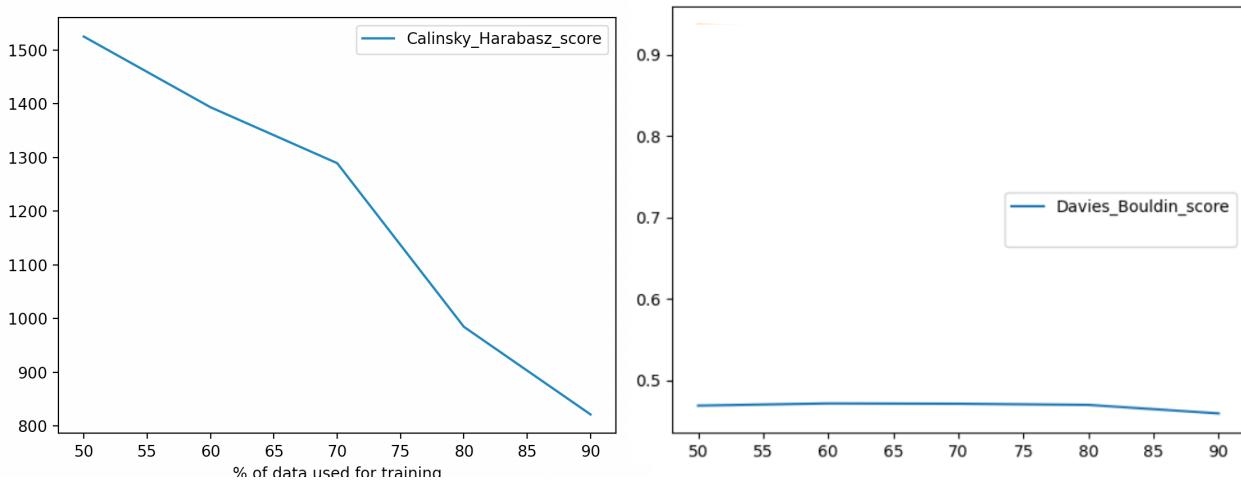


Calinski-harabasz index: 1600.4558610989345

Davies-bouldin index: 0.470134615517163



Графіки метрик другого датасету:



Висновки:

1. При підборі параметрів було помічено, що за кластеризації методом Affinity propagation отримується ненульова кількість кластерів для заданих датасетів лише за значення параметра `affition = 'euclidean'`, що і не дивно, оскільки матриці подібності одразу задано не було. Серед оптимальних моделей за метриками значення параметра `damping` змінювалося з 0.6 на 0.7 і навпаки. Головні зміни а якості роботи модель відбувалися внаслідок зміни значення параметра `preferens` – чим нижчим було це значення, тим точніше початковий набір даних розбивався на кластери, проміжок досліджуваних варіантів параметра `preferens`.
2. Під час видалення деяких даних із початкового набору змін у розподілі набору даних на кластери не спостерігалося, значення Davies – Bouldin майже не змінювалося, проте значення Calinski – Harabasz різко спадало. Пояснення цьому є інтуїтивно зрозумілим. Оскільки кількість та положення кластерів, а отже і їх розкид(дисперсія) не змінювалися, в той час із кластерів видалялися елементи, що призводило до збільшення внутрішньокласової дисперсії, і відповідно значення Calinski – Harabasz.
3. Відповідно до виконання пункту 9 лабораторної роботи, найкраще на заданих наборах даних проявили себе моделі з параметрами `damping = {0.6, 0.7}`

для обох наборів та значеннями -5;-7 для першого і -285 для другого, які наближаються до квадратів максимальних відстаней Хемінга, взятих зі знаком мінус.

Алгоритм стійкий до видалення даних

Код програми

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.datasets import make_blobs
from sklearn.exceptions import ConvergenceWarning
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AffinityPropagation
from sklearn.metrics import calinski_harabasz_score
from sklearn.metrics import davies_bouldin_score
from sklearn.metrics import v_measure_score
from sklearn.metrics.cluster import contingency_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics.pairwise import euclidean_distances
import pandas as pd
from sklearn.model_selection import train_test_split

from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)

def show_clusters(model, X, y_pred):
    plt.scatter(X[:, 0], X[:, 1], c=y_pred, s=45, cmap='viridis')
    centers = model.cluster_centers_
    plt.scatter(centers[:, 0], centers[:, 1], c='black', s=150, alpha=0.75)
    plt.title(f'Estimate number of clusters: {len(centers)}')
    plt.show()

def create_matrix(y_true, y_predict):
    cm = contingency_matrix(y_true, y_predict)
    sns.heatmap(cm, annot=True, square=True)
    plt.ylabel("Actual classes")
    plt.xlabel("Predicted clusters")
    plt.title("Contingency matrix")
    plt.show()

def get_scores(model, X, y_true):
    y_predict = model.predict(X)
    ch_index = calinski_harabasz_score(X, y_predict)
    db_index = davies_bouldin_score(X, y_predict)
    v_score = v_measure_score(y_true, y_predict)
    num_clusters = len(model.cluster_centers_indices_)
    return y_predict, ch_index, db_index, v_score, num_clusters

def metrics_plot(model, X, y_true):
    y_predict, ch_index, db_index, _, _ = get_scores(model, X, y_true)
    show_clusters(model, X, y_predict)
    print(f'Calinski-harabasz index: {ch_index}')
    print(f'Davies-bouldin index: {db_index}')
    create_matrix(y_true, y_predict)

def estimate_models(X, y_true, affinity, damping, preference):
    affinity_params = []
    v_scores = []
    ch_indices = []
    db_indices = []
    clusters_num = []

    for aff in affinity:
        for damp in damping:
            for pref in preference:
```

```

        try:
            AP = AffinityPropagation(affinity=aff, damping=damp,
preference=pref).fit(X)
            ch_index, db_index, v_score, num_clusters =
get_scores(AP, X, y_true)
            affinity_params.append([aff, damp, pref])
            ch_indices.append(ch_index)
            db_indices.append(db_index)
            v_scores.append(v_score)
            clusters_num.append(num_clusters)

        except Exception as exc:
            continue

    return pd.DataFrame({"params[affinity, damping, preference]":
affinity_params,
                      "V_scores": v_scores,
                      "cluster_number": clusters_num,
                      "CH_scores": ch_indices,
                      "DB_scores": db_indices})
}

def estimate_size_quality(model, X, y_true):
    samples = [i/10 for i in range(1, 6, 1)]
    ch_scores = []
    bd_scores = []
    v_scores = []

    for i in samples:
        x_samples, _, y_samples, _ = train_test_split(X, y_true, test_size=i)
        ch_scores.append(ch)
        bd_scores.append(bd)
        v_scores.append(v)
        print('Metrics with ', i*100, '% data remove')
        metrics_plot(model, x_samples, y_samples)

    samples = [i*10 for i in range(5, 10)]
    plt.plot(samples, ch_scores)
    plt.legend(['Calinsky_Harabasz_score'])
    plt.xlabel("% of data used for training")
    plt.show()

    plt.plot(samples, bd_scores)
    plt.plot(samples, v_scores)
    plt.legend(['Davies_Bouldin_score', 'V_score'])
    plt.show()

if __name__ == '__main__':
    # create datasets
    X_moons, y_moons = make_moons(n_samples=500, noise=0.15)
    X_blobs, y_blobs = make_blobs(n_samples=500,
                                  n_features=2,
                                  centers=4,
                                  cluster_std=1,
                                  center_box=(-10.0, 10.0),
                                  shuffle=True,
                                  random_state=1)

    # show datasets
    plt.scatter(X_moons[:, 0], X_moons[:, 1], c=y_moons)
    plt.show()

    plt.scatter(X_blobs[:, 0], X_blobs[:, 1], c=y_blobs)
    plt.show()

```

```

# create models:
model_moons = AffinityPropagation().fit(X_moons)
model_blobs = AffinityPropagation().fit(X_blobs)

# show clusters
print('\n* The first dataset *')
metrics_plot(model_moons, X_moons, y_moons)
print('\n* The second dataset *')
metrics_plot(model_blobs, X_blobs, y_blobs)

# change distances
new_preference1 = -1*np.amax(euclidean_distances(X_moons))**1.2
new_preference2 = -1*np.amax(euclidean_distances(X_blobs))**1.8

model_moons_change_pref =
AffinityPropagation(preference=new_preference1).fit(X_moons)
model_blobs_change_pref =
AffinityPropagation(preference=new_preference2).fit(X_blobs)

print('\n* The first dataset with changed preference *')
metrics_plot(model_moons_change_pref, X_moons, y_moons)
print('\n* The second dataset with changed preference *')
metrics_plot(model_blobs_change_pref, X_blobs, y_blobs)

# estimate models
affinity_check = ['euclidean', 'precomputed']
damping_check = [i / 10 for i in range(5, 10)]
preferences1 = [i for i in range(0, -14, -1)]
preferences2 = [i for i in range(-200, -300, -1)]

print('\n-----search for optimal parameters-----\n')

# first dataset
data_frame_moons = estimate_models(X_moons, y_moons, affinity_check,
damping_check, preferences1).\
    set_index("params[affinity, damping, preference]")
data_frame_moons = data_frame_moons[data_frame_moons["cluster_number"] != 0]
print('\nMetrics of the first dataset', data_frame_moons)

data_frame_moons_ch = data_frame_moons.sort_values(['CH_scores'],
ascending=False)
param = data_frame_moons_ch.index.values[0]
print('\nThe best model on ch-score:\n', param)
best = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_moons)
print(metrics_plot(best, X_moons, y_moons))

data_frame_moons_db = data_frame_moons.sort_values(['DB_scores'])
param = data_frame_moons_db.index.values[0]
print('\nThe best model on db-score:\n', param)
best_db = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_moons)
print(metrics_plot(best_db, X_moons, y_moons))

data_frame_moons_v = data_frame_moons.sort_values(['V_scores'],
ascending=False)
param = data_frame_moons_v.index.values[0]
print('\nThe best model on v-score:\n', param)
best_v = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_moons)
print(metrics_plot(best_v, X_moons, y_moons))

# second dataset
data_frame_blobs = estimate_models(X_blobs, y_blobs, affinity_check,
damping_check, preferences2).\
    set_index("params[affinity, damping, preference]")
data_frame_blobs = data_frame_blobs[data_frame_blobs["cluster_number"] != 0]
print('\nMetrics of the second dataset', data_frame_blobs)

```

```

    data_frame_blobs_ch = data_frame_blobs.sort_values(['CH_scores'],
ascending=False)
    param = data_frame_blobs_ch.index[0]
    print('\nThe best model on ch-score:\n', param)
    best = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_blobs)
    print(metrics_plot(best, X_blobs, y_blobs))

    data_frame_blobs_db = data_frame_blobs.sort_values(['DB_scores'])
    param = data_frame_blobs_db.index[0]
    print('\nThe best model on db-score:\n', param)
    best_db = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_blobs)
    print(metrics_plot(best_db, X_blobs, y_blobs))

    data_frame_blobs_v = data_frame_blobs.sort_values(['V_scores'],
ascending=False)
    param = data_frame_blobs_v.head().index[0]
    print('\nThe best model on v-score:\n', param)
    best = AffinityPropagation(affinity=param[0], damping=param[1],
preference=param[2]).fit(X_blobs)
    print(metrics_plot(best, X_blobs, y_blobs))

estimate_size_quality(model_moons_change_pref, X_moons, y_moons)
estimate_size_quality(model_blobs_change_pref, X_blobs, y_blobs)

```