Лабораторна робота №1
“Кластеризація: порівняльний аналіз кількох алгоритмів кластеризації наборів даних”

Група ТК-41
Соловей Євгеній

2025

Зміст

Github: https://github.com/TheYev/KNU/tree/main/ROTAS/lab1

K-Means

K-means - це один із найпопулярніших алгоритмів кластеризації, який використовується для поділу даних на k груп . Він працює за методом розбиття даних таким чином, щоб кожен об'єкт належав до найближчого кластера.

Розглянемо реалізований алгоритм

Спочатку напишемо функцію яка створює 4 групи точок, які будуть використовуватися для кластеризації. Кожна група має свої межі координат, що імітує реальні дані, які можуть природно розбиватися на групи.

Визначення оптимального числа кластерів. Алгоритм KMeans пробує різні значення k (від 2 до 6). kmeans.fit(dataset) знаходить кластери. kmeans.score(dataset) повертає внутрішньо кластерну дисперсію (якість кластеризації). Графік plt.plot(range(2, 7), scores) допомагає знайти точку "згину", де оптимально вибрати k.

```
scores = []
for i in range(2, 7):
    kmeans = KMeans(n_clusters=i, n_init=10)
    kmeans.fit(dataset)
    y_kmeans = kmeans.predict(dataset)
    scores.append(kmeans.score(dataset))

plt.plot(range(2, 7), scores)
plt.xticks(range(2, 7))
plt.show()
```

Запуск кластеризації для k=4. Після вибору k=4 виконується кластеризація. y_kmeans містить номери кластерів для кожної точки.

```
kmeans = KMeans(n_clusters=4, n_init=10)
kmeans.fit(dataset)
y_kmeans = kmeans.predict(dataset)
```

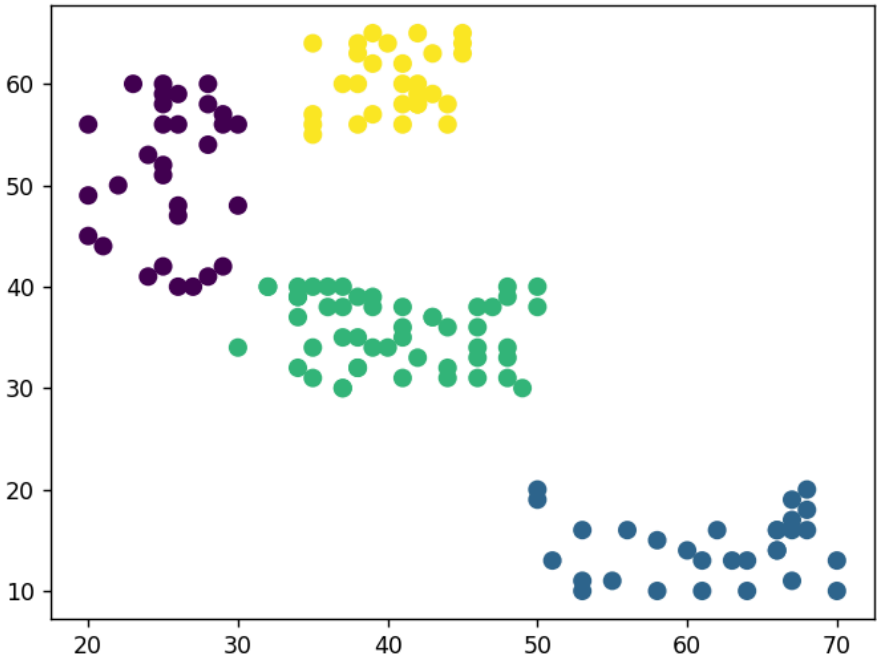
Оскільки наш набір даних містить чотири розподіли, логічно обрати саме таку кількість кластерів. Проте в загальному випадку дані можуть групуватися по-різному, і зі збільшенням кількості центрів середня відстань точок до них зменшується. Це може створювати ілюзію кращої кластеризації, навіть якщо оптимальним вибором є менша кількість кластерів. Для визначення правильної кількості центрів варто використовувати попередні знання або методи, як-от "elbow rule" чи "silhouette score". Оскільки K-means є випадковим алгоритмом, його слід запускати кілька разів і обирати найкращий розв’язок, порівнюючи середні відстані точок до центрів.

Використаємо реалізацію моделі K-Means з бібліотеки sklearn. Дана реалізація є дуже популярним вибором у реальному житті і розробники часто її використовують. Застосування методу K-Means за допомогою sklearn є дуже простим:

```
from sklearn.cluster import KMeans
```

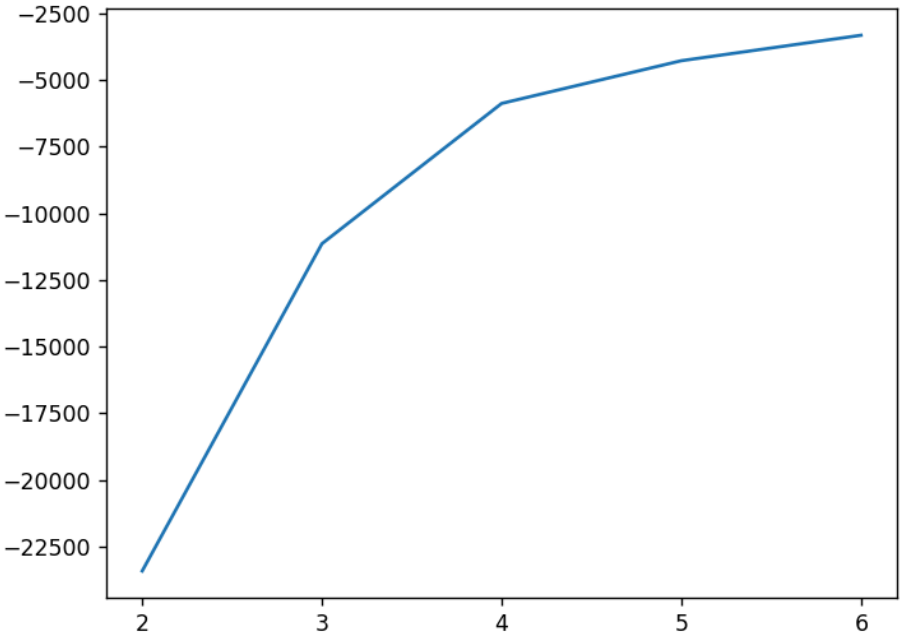
```
kmeans = KMeans(n_clusters=4, n_init=10)
kmeans.fit(dataset)
y_kmeans = kmeans.predict(dataset)
plt.scatter([data[0] for data in dataset], [data[1] for data in dataset], c=y_kmeans, s=50, cmap='viridis')
plt.show()
```

Було використано такі параметри як n_clusters=4 та n_init=10, тобто ми вирішили знайти саме чотири центри кластерів.



Якщо орати іншу кількість кластерів, то результат буде виглядати інакше.

Демонстрація метод ліктя (“elbow rule”) для вибору правильної кількості кластерів.



Як ми бачимо, зі збільшенням кількості центрів, відстань точок до них тільки зменшується, навіть коли ми збільшуємо кількість центрів до неоптримального числа (п’ять, шість). Але також ми можемо побачити на графіку, що після чотирьох центрів покращення стає набагато менш суттєвим. Тому, навіть якби ми не знали, що дані отримані з чотирьох розподілів, ми могли б використати правило ліктя та обрати K-Means алгоритм саме із чотирма центрами.

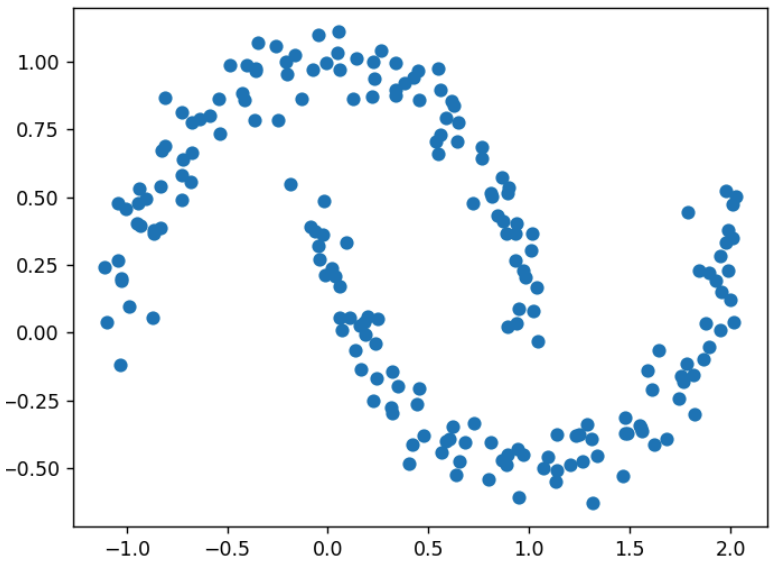
DBSCAN

DBSCAN - це один методів кластеризації, який групує точки на основі їхньої щільності. Він добре працює з наборами даних довільної форми та здатний виявляти шумові точки, які не належать жодному кластеру.

Розглянемо реалізований алгоритм

Генерація даних. Функція make_moons створює 200 точок у вигляді двох півмісяців. noise=0.07 додає невеликий шум, щоб дані не були ідеально розділеними. Після генерації ми відображаємо дані на графіку, щоб побачити початковий розподіл точок.

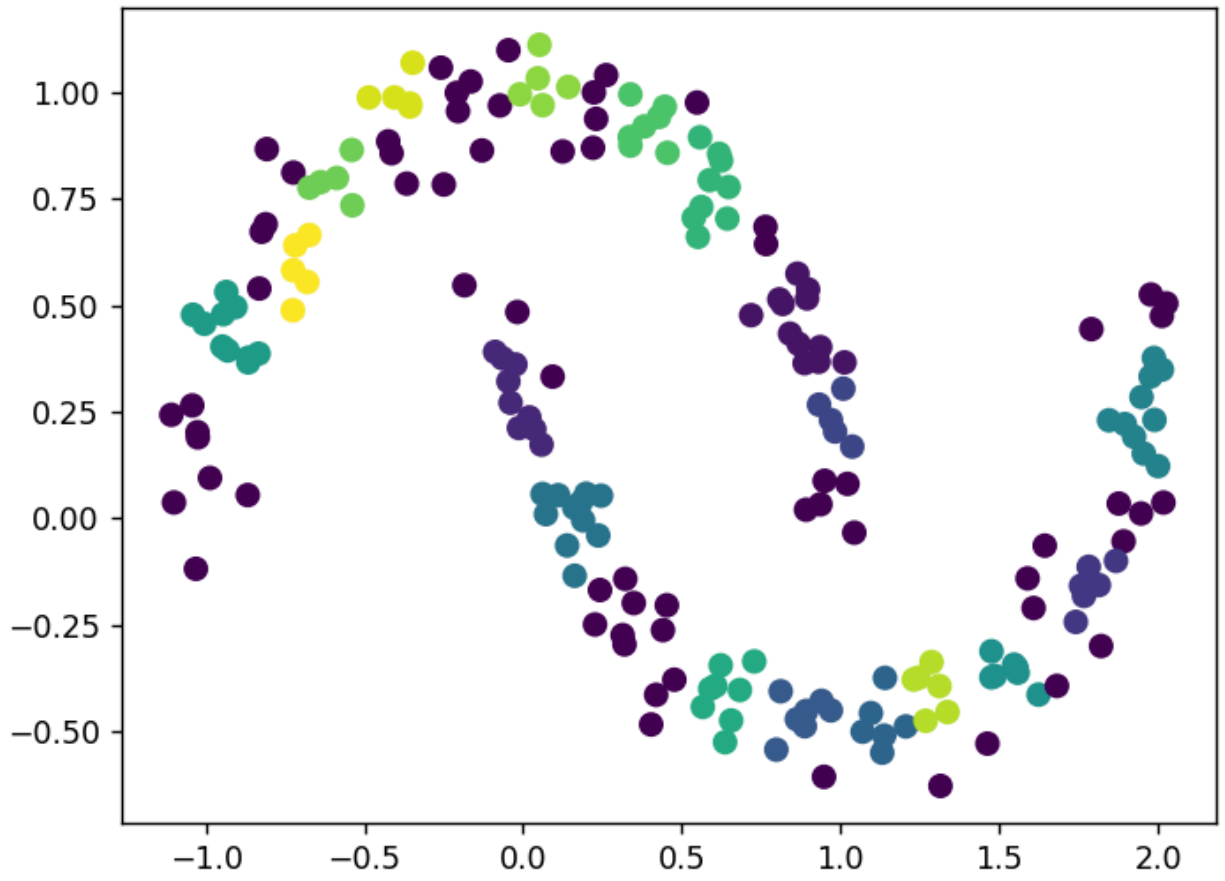
```
dataset = make_moons(n_samples=200, noise=0.07, random_state=0)
plt.scatter([data[0] for data in dataset[0]], [data[1] for data in dataset[0]])
plt.show()
```



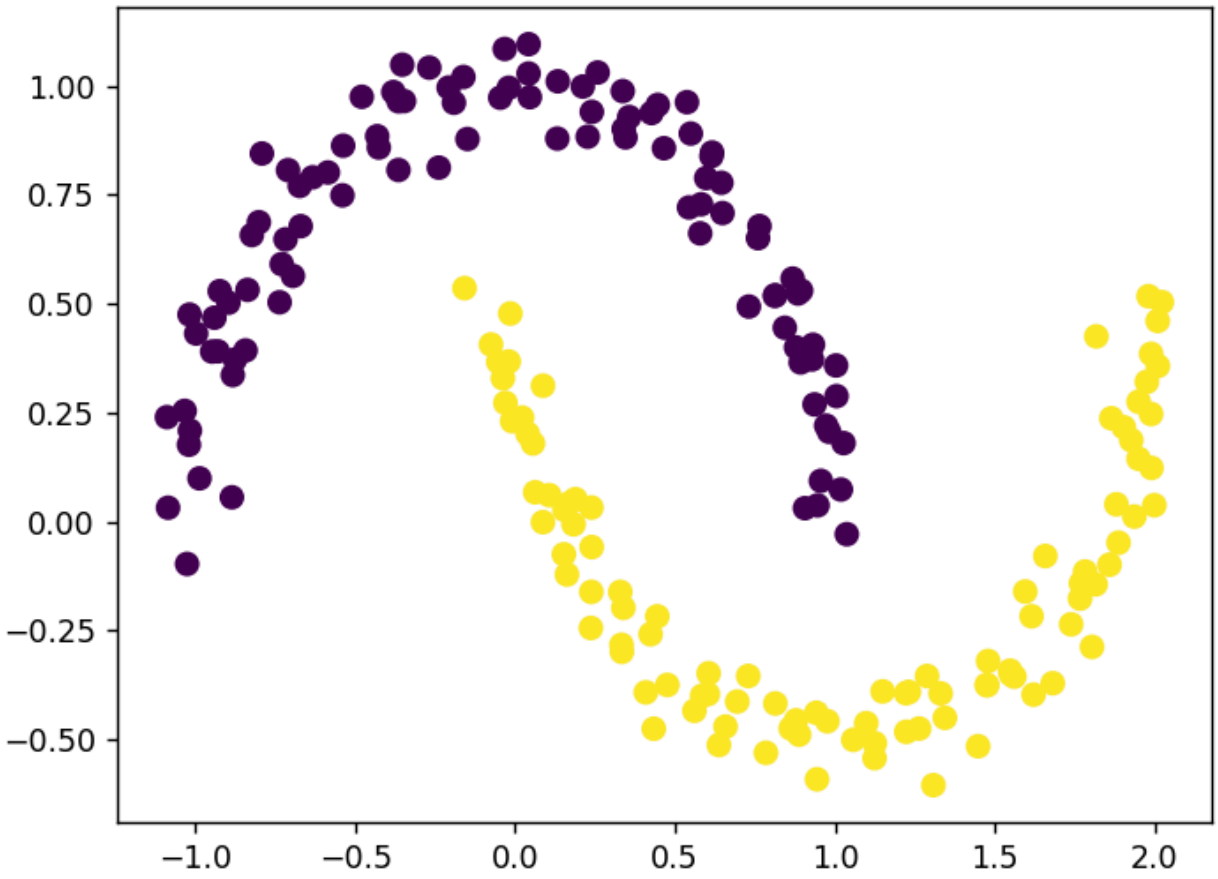
Запуск DBSCAN з eps=0.1.

eps=0.1 означає, що точка буде вважатися частиною кластера, якщо вона має достатньо сусідів у радіусі 0.1. min_samples=5 задає мінімальну кількість точок для формування кластера. dbscan.fit(dataset[0]) запускає кластеризацію, а dbscan.labels_ отримує мітки кластерів для кожної точки. Точки одного кластеру матимуть однаковий колір. Точки, які DBSCAN вважає шумовими (outliers), будуть позначені як -1 (зазвичай чорного кольору). Маленьке значення eps (0.2) означає, що алгоритм буде шукати тільки найближчі точки для об'єднання у кластери, тому може утворитися багато дрібних кластерів, а деякі точки залишаться шумовими.

```
dbscan = DBSCAN(eps=0.1, min_samples=5)
dbscan.fit(dataset[0])
y_dbscan = dbscan.labels_
plt.scatter([data[0] for data in dataset[0]], [data[1] for data in dataset[0]], c=y_dbscan, s=50, cmap='viridis')
plt.show()
```



Бачимо що утворилось багато кластерів, тому зробимо наш eps=0.3 та отримає результат.



Як бачимо відбулось розділення на кластори. Змінючи наш параметер eps можемо зручно керувати моделю.

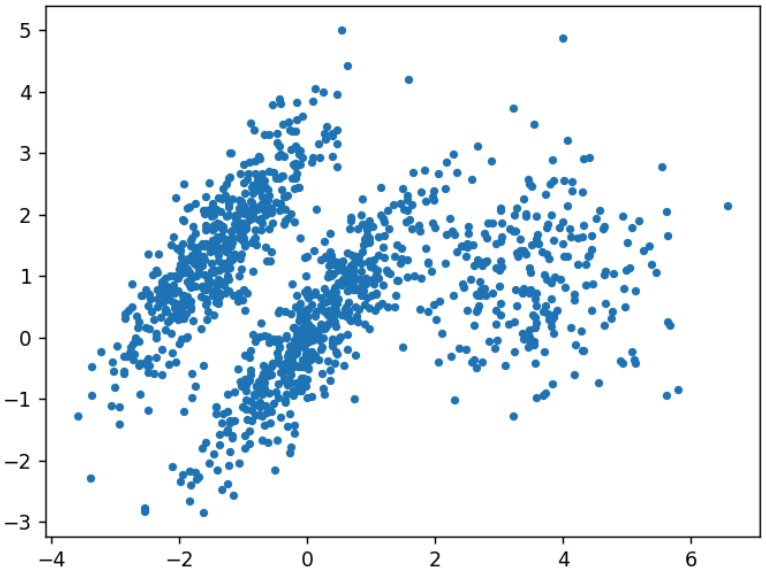
Gaussian Mixtures

Gaussian Mixture Model - це метод кластеризації, який моделює дані як суміш кількох гаусових розподілів. На відміну від k-means, який жорстко прив’язує кожну точку до певного кластера, GMM використовує ймовірності: кожна точка може частково належати до кількох кластерів.

Розглянемо реалізований алгоритм

Генерація даних. Створюємо два набори даних (X1 та X2) за допомогою make_blobs. X1 містить 1000 точок, центрованих у (4, -4) і (0, 0), після чого піддається лінійному перетворенню, щоб зробити кластери еліптичними. X2 містить 250 точок біля (6, -8). Об'єднуємо всі точки в один набір X. Через неоднакову форму кластерів, точки, що належать одному з них можуть бути невірно присвоєні іншому кластеру.

```
X1, y1 = make_blobs(n_samples=1000, centers=((4, -4), (0, 0)), random_state=42)
X1 = X1.dot(np.array([[0.374, 0.95], [0.732, 0.598]]))
X2, y2 = make_blobs(n_samples=250, centers=1, random_state=42)
X2 = X2 + [6, -8]
X = np.r_[X1, X2]
```



Щоб побороти цю проблему, було запропоновано розробити модель для кластеризації, що бере до уваги не тільки координати центру кластеру, а й

його форму. Форма кластера визначається варіативністю розподілу в різних напрямках. Якщо горизонтальна варіативність висока, а вертикальна низька, кластер буде витягнутим по горизонталі. Центр визначається не просто як координати, а як середнє значення всього розподілу. Оскільки ми враховуємо і середнє, і варіативність, розподіл природно описується Гаусівським розподілом із середнім значенням μ та матрицею варіативності Σ. Як у K-Means, алгоритм починає з випадкових μ та Σ, потім визначає приналежність точок до розподілів ітеративно оновлюючи параметри. Відмінність у тому, що тут оптимізується більше параметрів, що дозволяє точніше описати кластери.

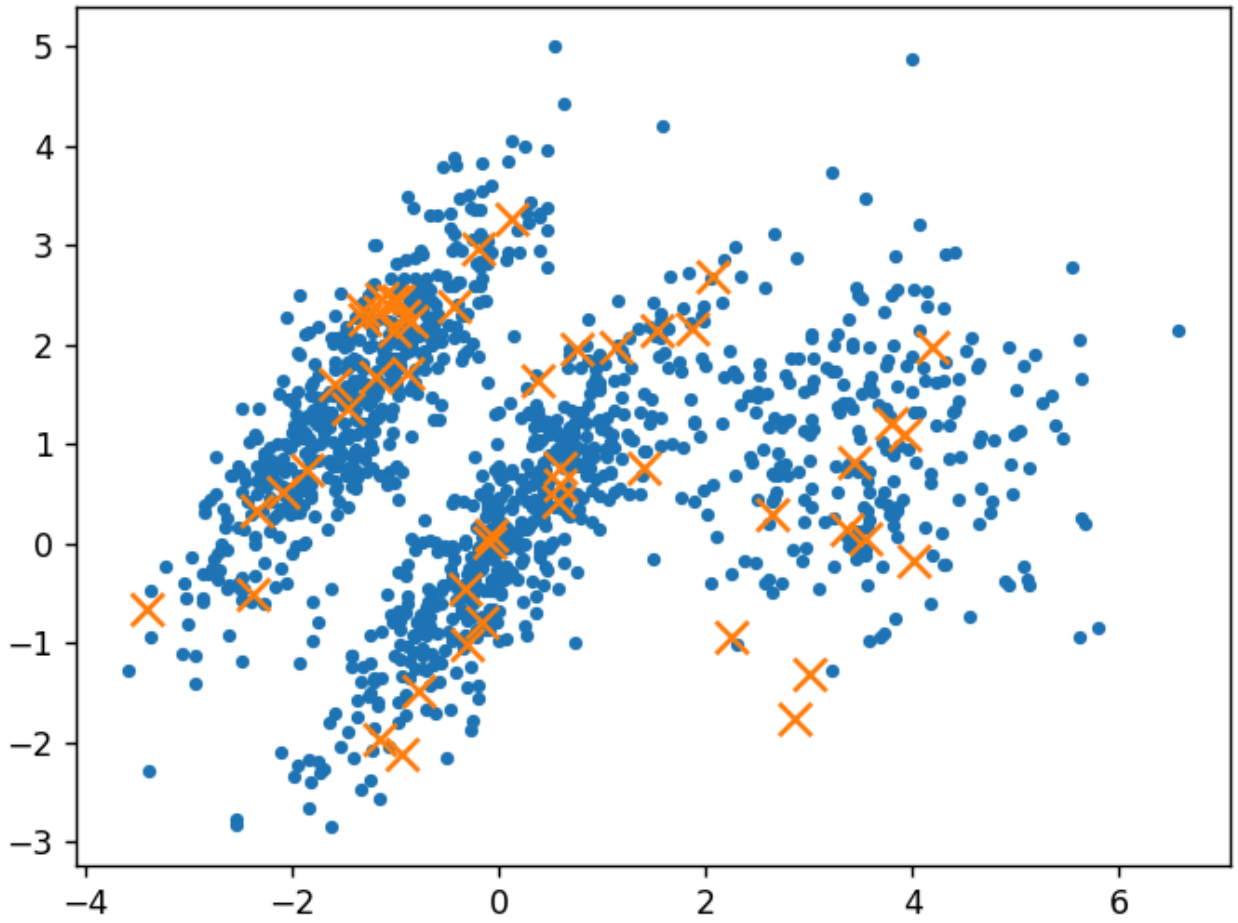
```
X1, y1 = make_blobs(n_samples=1000, centers=((4, -4), (0, 0)), random_state=42)
X1 = X1.dot(np.array([[0.374, 0.95], [0.732, 0.598]]))
X2, y2 = make_blobs(n_samples=250, centers=1, random_state=42)
X2 = X2 + [6, -8]
X = np.r_[X1, X2]
```

```
gm = GaussianMixture(n_components=3, n_init=10)
gm.fit(X)
print(gm.weights_)
print(gm.means_)

res = gm.sample(100)

plt.scatter(X[:, 0], X[:, 1], s=10)
plt.scatter(res[0][:, 0], res[0][:, 1], s=100, marker='x')
plt.show()
```

GaussianMixture(n_components=3, n_init=10) створює GMM з 3 кластерами. gm.fit(X) навчає модель. gm.sample(100) генерує 50 нових точок відповідно до ймовірнісного розподілу кластерів.



Будується графік із початковими точками. Додатково малюються згенеровані точки, які модель вважає такими, що відповідають розподілу вихідних кластерів.

Висновок

K-Means, DBSCAN та Gaussian Mixtures - три популярні алгоритми кластеризації, кожен із яких має свої переваги та недоліки залежно від типу даних.

1. K-Means - простий і швидкий алгоритм, що добре працює для сферичних кластерів. Однак він вимагає попереднього визначення кількості кластерів і чутливий до вибору початкових центрів.
2. DBSCAN - добре підходить для кластерів довільної форми та може автоматично визначати їх кількість. Він стійкий до шуму, але потребує ретельного підбору параметрів eps та min_samples.
3. Gaussian Mixtures - більш гнучкий підхід, що моделює кластери як Гаусівські розподіли. Він може виявляти перекриваючі кластери, але складніший у налаштуванні та вимагає більше обчислень.

Вибір алгоритму залежить від структури даних:

1. K-Means - для добре розділених, сферичних кластерів.
2. DBSCAN - для кластерів довільної форми та виявлення шуму
3. Gaussian Mixtures- для складніших розподілів із перекриттям кластерів.

Використані джерела

1. https://www.w3schools.com/python/python_ml_k-means.asp
2. https://www.datacamp.com/tutorial/dbscan-clustering-algorithm
3. https://scikit-learn.org/stable/modules/mixture.html
4. https://medium.com/@juanc.olamendy/understanding-gaussian-mixture-models-a-comprehensive-guide-df30af59ced7