Розділ 5

Навчання без учителя: задачі кластеризації

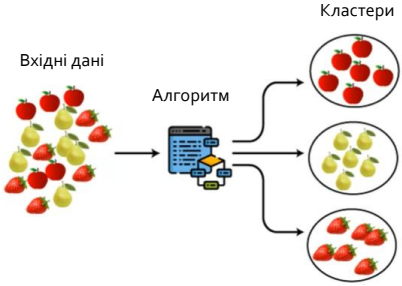
У цьому розділі буде надано основний теоретичний матеріал щодо методів кластеризації даних. Алгоритми кластеризації дозволяють розділити об'єкти у випадку, коли класи заздалегідь не зазначені, а кластери мають бути сформовані за схожістю елементів. Буде розглянуто два основних та найбільш часто використовуваних методів:

- Метод k-середніх (k-Means)
- Метод DBscan

Для кожного методу буде поставлено завдання та наведено приклад його розв'язання, а також приклад подання результатів.

5.1 Кластеризація

Кластеризація це класифікація, але без заздалегідь відомих класів. Машина сама шукає схожі об'єкти та об'єднує їх у кластери. Кількість кластерів можна задати заздалегідь або довірити машині. Схожість об'єктів машина визначає за тими ознаками, які ми їй розмітили – у кого багато схожих характеристик, тих поєднують в один кластер.



- Відмінний приклад кластеризації — маркери на картах в інтернеті. Коли ви шукаєте всі ресторани азіатської кухні у великому місті, машині доводиться групувати їх у кружечки з цифрою, інакше браузер зависне в потугах намалювати мільйон маркерів.
- Більш складні приклади кластеризації можна згадати у програмах iPhoto або Google Photos, які знаходять обличчя людей на фотографіях та групують їх у альбоми. Програма не знає як звуть ваших друзів, але може відрізнити їх за характерними рисами обличчя.

Кластеризація сьогодні використовують для:

- Сегментація ринку (типів покупців)
- Об'єднання близьких точок на карті
- Стиснення зображень
- Аналіз та розмітки нових даних
- Детектори аномальної поведінки

Популярні алгоритми: Метод K-середніх (K-means), Mean-Shift, DBSCAN

5.1.1 Постановка задачі

Дано:

- X – простір об'єктів
- $X^\ell = \{x_1, \dots x_\ell\}$ – навчальна вибірка
- $\rho:\ X \to [0, \infty)$ – функція відстані між об'єктами

Знайти:

- Y – множина кластерів
- $a:\ X \to Y$ – алгоритм кластеризації

Властивості кластерів:

- Кожен кластер складається з близьких за ознаками об'єктів
- Об'єкти різних кластерів суттєво різні

5.1.2 Некоректність задачі кластеризації

Розв'язок задачі кластеризації принципово неоднозначний

- Точної постановки задачі кластеризації як правило немає
- Існує багато критеріїв якості кластеризації: визначення оптимальної кількості кластерів
- Існує багато евристичних методів кластеризації
- Зазвичай кількість кластерів заздалегідь не відома
- Результат кластеризації сильно залежить від метрики ρ для розрахунку відстані між об'єктами

Приклад: скільки тут кластерів?

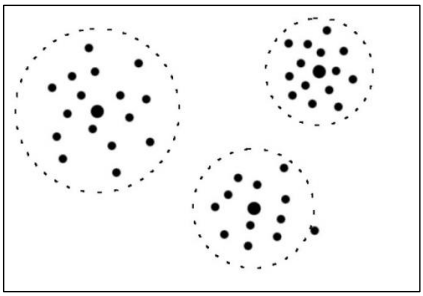
5.1.3 Завдання кластеризації

- Спростити подальшу обробку даних: розбити множину X^ℓ на групи схожих об'єктів з метою подальшого аналізу кожної групи окремо (задачі класифікації, регресії, прогнозування)
- Скоротити об'єм даних що зберігається: залишити по одному з типових представників кожного кластеру (центр мас) – задачі стиснення даних
- Виділити нетипові об'єкти (викиди), які не підходять до жодного з кластерів (задачі одно класової класифікації)
- Побудувати ієрархію множини об'єктів (класифікація рослин та тварин)

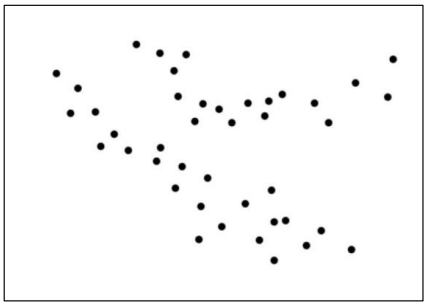
5.1.4 Типи структур кластерів

- Кожен метод кластеризації має свої обмеження і виділяє кластери лише декількох типів
- Поняття "тип кластерної структури" залежить від методу і також не має формального визначення

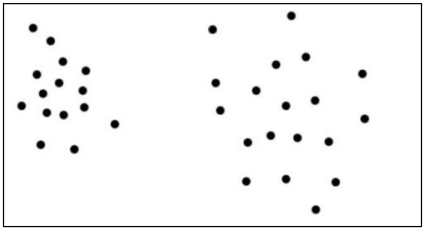
Кластери з центрами



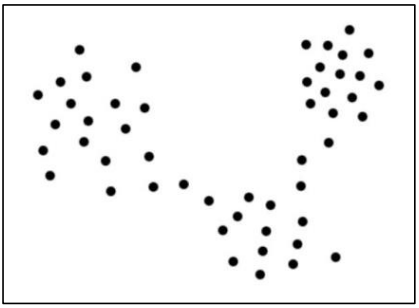
Стрічкові кластери



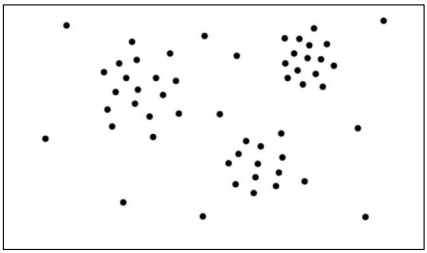
Внутрішньо-кластерні відстані менші за між-кластерні



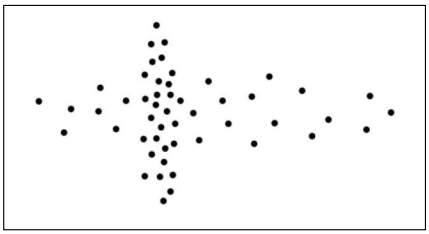
Існування перемичок між кластерами



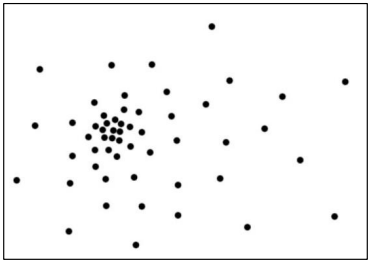
Розріджений набір нетипових об'єктів

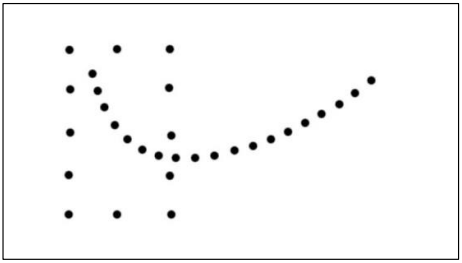


Кластери що перекриваються



Кластерів взагалі може не існувати





5.1.5 Методи кластеризації

"Кластеризацією" зазвичай вважають такий набір кластерів, які містять усі об'єкти набору даних. Додатково, можна розглянути відношення між кластерами. Наприклад, ієрархію вкладеності кластерів один у одного. Грубо можна виділити такі кластеризації:

- *Жорстка кластеризація*: кожен об'єкт або належить кластеру або ні.

- *М'яка кластеризація* (також нечітка кластеризація): кожен об'єкт належить кожному кластеру до певної міри. Наприклад, це ймовірність належності кластеру.

Серед них виділяють декілька доладних:

- *Жорстке розбиття на кластери*: кожен об'єкт належить рівно одному кластеру.
- *Жорстке розбиття на кластери з викидами*: об'єкт може не належати жодному кластеру і розглядається як викид.
- *Кластери з перетином*: об'єкт може належати більш ніж одному кластеру.
- *Ієрархічна кластеризація*: якщо об'єкт належить нащадку, то він також належить і предку.
- *Підпросторова кластеризація*: хоч кластери і можуть перетинатись, проте в межах визначеного підпростору кластери не перетинаються.

Типовими кластерними моделями є:

- *Моделі зв'язності*: наприклад, ієрархічна кластеризація або таксономія будуються на основі відстані між вузлами.
- *Центроїдні моделі*: наприклад, метод K-середніх (K-means) представляє кожен кластер єдиним усередненим вектором.
- *Статистичні моделі*: кластери будуються ґрунтуючись на статистичних розподілах.
- *Моделі засновані на щільності*: наприклад, в DBSCAN і в OPTICS кластери визначаються як зв'язані області відповідної щільності у просторі даних.
- *Групові моделі*: деякі алгоритми не забезпечують вдосконалену модель для своїх результатів, а просто описують групування об'єктів.

- *Графові моделі*: поняття клітинки (така підмножина вершин, в якій кожна пара вершин з'єднана ребром) у графі слугує прототипом кластеру.
- *Нейронні моделі*: найбільш відомою моделлю нейронної мережі з навчанням без учителя є нейронна мережа Кохонена.

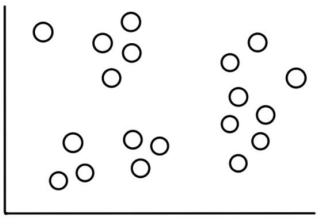
5.2 Метод k-середніх (k-means Method)

Даний метод належить до алгоритмів неієрахічного поділу (Partitioning algorithms), які здійснюють декомпозицію набору даних, що складається з n спостережень, на k груп (кластерів) із заздалегідь невідомими параметрами. При цьому виконується пошук центроїдів – максимально віддалених один від одного центрів згущень точок з мінімальним розкидом усередині кожного кластера.

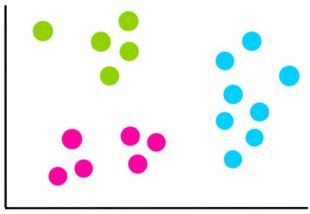
5.2.1 Покрокова реалізація методу

Постановка задачі та попередня обробка даних

Нехай існує набір даних з $n = 19$ значень, які виглядають приблизно так:



Тепер припустимо, що ми знаємо, що ці дані розбиваються на три, відносно очевидні категорії і виглядатиме це так:



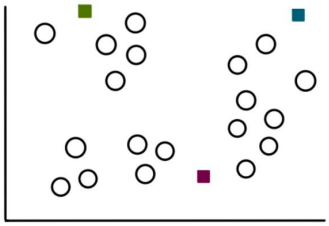
Наше завдання -– використовувати алгоритм кластеризації k-means, щоб зробити цю категоризацію.

Крок 1: Вибираємо кількість кластерів, k

Число кластерів, які ми хочемо розпізнати, це і є k у k-means. У нашому випадку, тому що ми припустили, що всього 3 кластери, $k = 3$.

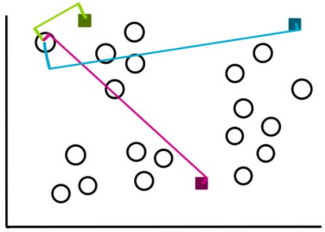
Крок 2: Вибираємо до випадкових значень

Процес виявлення кластерів ми починаємо з вибору трьох випадково обраних значень (не обов'язково, щоб вони були нашими даними). Ці точки зараз працюватимуть як центроїди або центри кластерів, які ми збираємося згрупувати:



Крок 3: Створення кластерів k

Щоб створити кластери, ми почнемо вимірювати відстань між кожним значенням наших даних до кожного з трьох центроїдів, а потім додаємо його до найближчого кластера. Для значення, взятого як приклад, відстані виглядатимуть приблизно так:



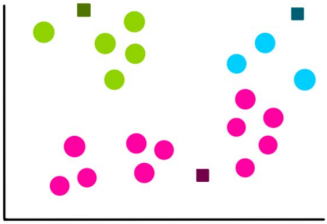
Для розрахунку відстані між об'єктами буде використовувати стандартне визначення декартової відстані:

$$d = \sqrt{\sum_{i=1}^{m}(x_i - c_{ij})^2},$$

де x_i – координата в m-вимірному просторі (значення кожної ознаки); c_{ij} – координата кожного центроїда. У 2-мірному просторі для знаходження відстані між двома точками маємо:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Використовуючи цю формулу, ми повторюємо процес зі значеннями, що залишилися, після цього кластери будуть виглядати наступним чином:

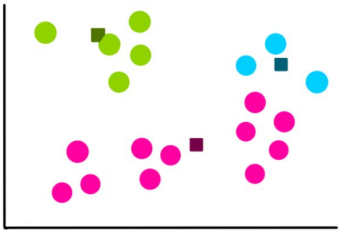


Крок 4: Обчислюємо новий центроїд кожного кластера

Тепер, коли ми маємо всі три кластери, ми знаходимо нові центроїди для кожного з них за формулою:

$$(c_{1j}, c_{2j}, \ldots, c_{mj}) = \left(\frac{1}{n_j} \sum_{i=1}^{n_j} x_{1i}, \frac{1}{n_j} \sum_{i=1}^{n_j} x_{2i}, \ldots, \frac{1}{n_j} \sum_{i=1}^{n_j} x_{mi} \right)$$

c_{kj} – k-та координата j-того кластера;
x_{ki} – k-та координата i-того об'єкта, що віднесений до j-того кластера;
n_j – кількість об'єктів у j-тому кластері.
Так, нові центроїди матимуть вигляд:



Крок 5: Знову віднесемо об'єкти до кожного з центроїдів

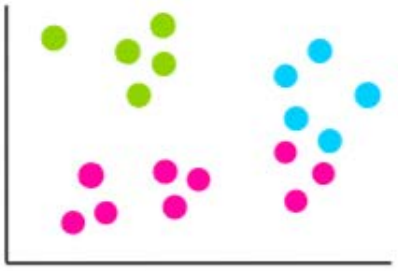
Деякі об'єкти віднесуться до іншого центроїду (кластеру). Розраховуємо суму квадратів внутрішньокластерних відстаней до центру кластера (within-cluster sum of squares, WCSS):

$$Q = \sum_{j=1}^{k} \sum_{i=1}^{n} \min \left(||x_i^j - c_j|| \right)^2 \tag{5.1}$$

Крок 6: повторюємо кроки 4-5

Кроки 4 і 5 повторюються до тих пір, поки алгоритм не стабілізується, тобто до тих пір, поки об'єкти не перестануть переходити від одного центроїду (кластера) до іншого. Говорячи формально, мета алгоритму – мінімізувати функція втрат (5.1).
Давайте припустимо, що наступні 4 ітерації виглядатимуть так:

Ітерація 1



Ітерація 3

Ітерація 2

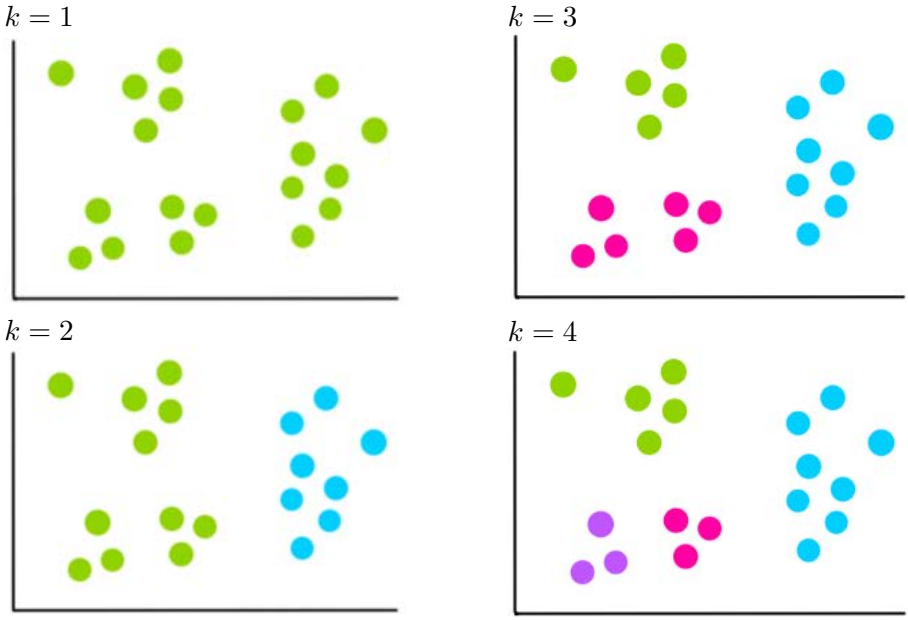
Ітерація 4

На двох останніх ітераціях бачимо, що кластери не змінюються. Це означає, що алгоритм зійшовся і ми зупиняємо процес. Потім ми вибираємо кластери із найменшим WCSS. Це будуть кластери на останніх 2 ітераціях, тому вони стають нашими фінальними кластерами.

5.2.2 Вибір кількості кластерів

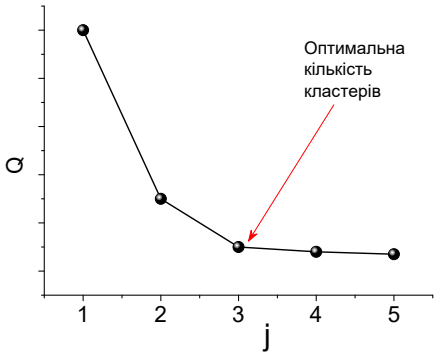
Один із способів вибору кількості кластерів є *експертний метод* – вибір кількості кластерів залежатиме від знання про предметну область (domain knowledge).

При кластеризації методом k-середніх кількість кластерів найчастіше оцінюють за допомогою "методу ліктя" (*elbow method*). Він передбачає багаторазове циклічне виконання алгоритму зі збільшенням кількості кластерів, що вибираються.



Для кожного вибору отримаємо мінімальне значення суму квадратів внутрішньокластерних відстаней за формулою (5.1). За результатами досліджень будуємо графік залежності $Q(j)$.

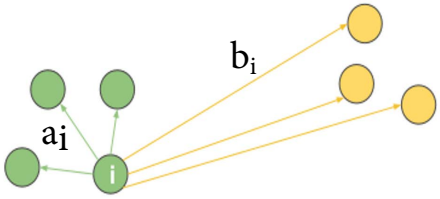
На рисунку наведено типову залежність $Q(j)$. Як бачимо, після того, як кількість кластерів досягає трьох, сума квадратів внутрішньокластерних відстаней перестає суттєво зменшуватися.



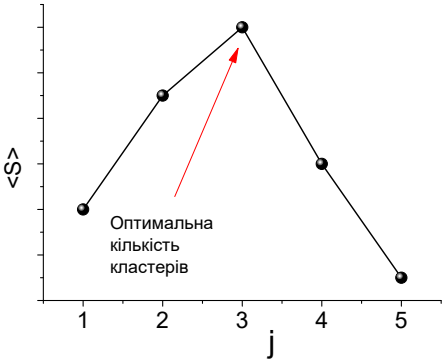
Метод “силуету” (*Silhouette Score*): коефіцієнт “силует” обчислюється за допомогою середньої внутрішньо-кластерної відстані a та середньої відстані до найближчого кластера b за кожним зразком:

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

a_i – середня відстань від об’єкта i до об’єктів свого кластеру;
b_i – середня відстань від об’єкта i до об’єктів іншого кластеру.



На рисунку наведено типову залежність $\langle S \rangle$. Як бачимо, Середнє значення силуету зростає до $k = 3$, та різко знижується при збільшенні значень k. Тобто ми отримуємо виражений пік при $k = 3$.



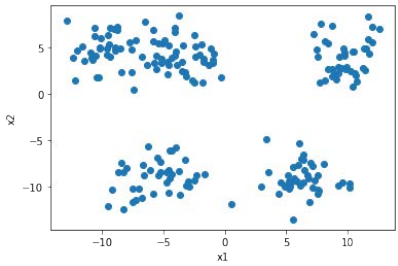
5.2.3 Постановка задачі

Провести кластеризацію даних з використанням методу k-середніх.
Етапи розв'язання

1. Імпортувати/згенерувати розмічені/нерозмічені дані
2. У випадку кількості ознак $n > 2$ звести до $n = 2$ з використанням SVD та PCA
3. Провести кластеризацію методом k-means при фіксованому значенні кількості кластерів
4. Візуалізувати отримані результати
5. Встановити оптимальну кількість кластерів з використанням методів "метод ліктя" та "метод силуету". Порівняти отримані результатами з розміченими даними (за наявності)
6. Порівняти результати з k-means з sklearn
7. Оформити результати у вигляді звіту.

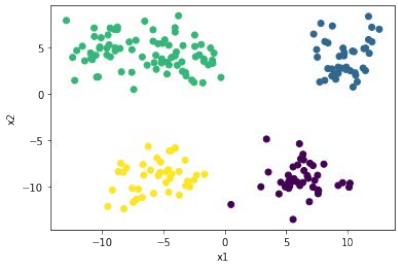
5.2.4 Приклад подання результатів

Імпортуємо/генеруємо вибірку з розміченими даними. За умови, якщо кількість ознак $n > 2$ зводимо до $n = 2$ (зменшуємо розмірність) за допомогою методів SVD та PCA.



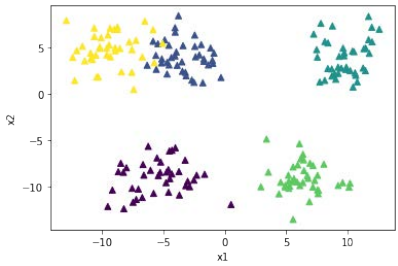
Візуалізуємо дані. Визначаємо (обираємо) кількість кластерів, які на нашу думку реалізуються у вибірці. Фіксуємо значення параметра k_0 що задає кількість кластерів.

Ділимо всю вибірку на навчальну та тестову. Будуємо алгоритм класифікації для $k = k_0$.



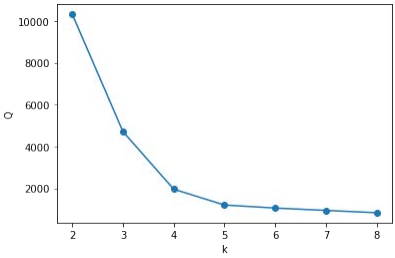
Візуалізуємо результати роботи алгоритму для $k_0 = 4$. Тут кружками різного кольору позначено різні кластери.

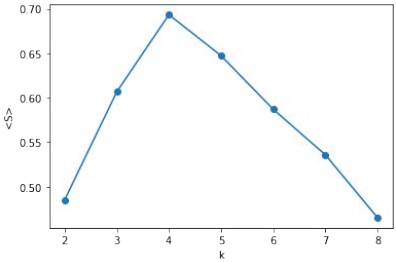
Візуалізуємо розмічені дані (за наявності) та рахуємо помилку класифікації



У розмічених даних існує 5 кластерів. З отриманих результатів робимо висновок про правильність кластеризації.

Проводимо оптимізацію алгоритму. Рахуємо внутрішньо-кластерні відстані Q за методом “ліктя” та коефіцієнт “силуету”





Робимо висновок щодо оптимальної кількості кластерів у вибірці. У даному випадку маємо $n = 4$.

Проводимо порівняння отриманих результатів з вбудованим класифікатором kNN бібліотеки sklearn

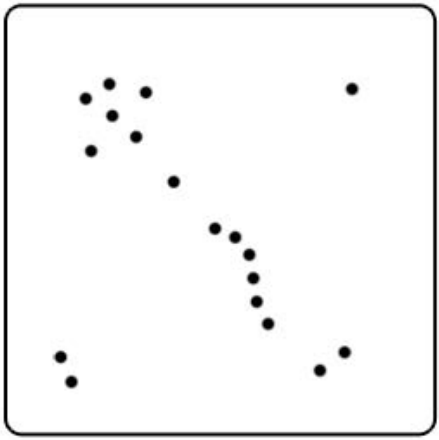
5.3 Метод DBSCAN

Маючи справу з просторовими кластерами різної щільності, розміру та форми, може бути складно виявити групу точок. Завдання може бути ще складнішим, якщо дані містять шум і викиди. Для роботи з великими просторовими базами даних був запропонований метод DBSCAN (Density-based spatial clustering of applications with noise, густинний алгоритм просторової кластеризації з присутністю шуму), який, як випливає з назви, оперує щільністю даних. На вхід він просить вже знайому матрицю близькості та два параметри – радіус *epsilon*-околиці та кількість сусідів. Так одразу і не зрозумієш, як їх вибрати, причому тут щільність, і чому саме DBSCAN добре розправляється з шумовими даними. Без цього складно визначити межі його застосування. Можна виділити три основні причини використання алгоритму:

1. Вимагає мінімальних знань предметної галузі.
2. Він може виявити кластери довільної форми.
3. Ефективний для великих баз даних, тобто коли розмір вибірки перевищує кілька тисяч.

5.3.1 Інтуїтивне пояснення

У гігантській залі натовп людей справляє чийсь день народження. Хтось тиняється один, але більшість – із товаришами. Деякі компанії просто юрмляться юрбою, деякі – водять хороводи або танцюють ламбаду.



Завдання: поділити людей у залі на гурти.

Але як виділити групи такої
різної форми, та ще й не забути про одинаків? Спробуємо оцінити щільність юрби навколо кожної людини. Напевно, якщо щільність натовпу між двома людьми вища за певний поріг, то вони належать одній компанії. Справді, буде дивно, якщо люди, які ведуть "потяг", будуть відноситися до різних груп, навіть якщо щільність ланцюжка між ними змінюється в деяких межах.

- Будемо говорити, що поруч із деякою людиною зібрався натовп, якщо близько до нього стоять кілька інших людей. Отже, відразу видно, що потрібно задати два параметри:

- Що означає "близько"? Візьмемо якусь інтуїтивно зрозумілу відстань. Скажімо, якщо люди можуть торкнутися голів один одного, то вони знаходяться близько. Близько метра.

- Тепер, скільки саме "кілька інших людей"? Припустимо, троє людей. Двоє можуть гуляти і просто так, але третій – безумовно зайвий.

- Нехай кожен підрахує, скільки людей знаходиться в радіусі метра від нього. Усі, хто має хоча б трьох сусідів, беруть у руки зелені прапорці. Тепер вони є корінними елементами, саме вони формують групи.

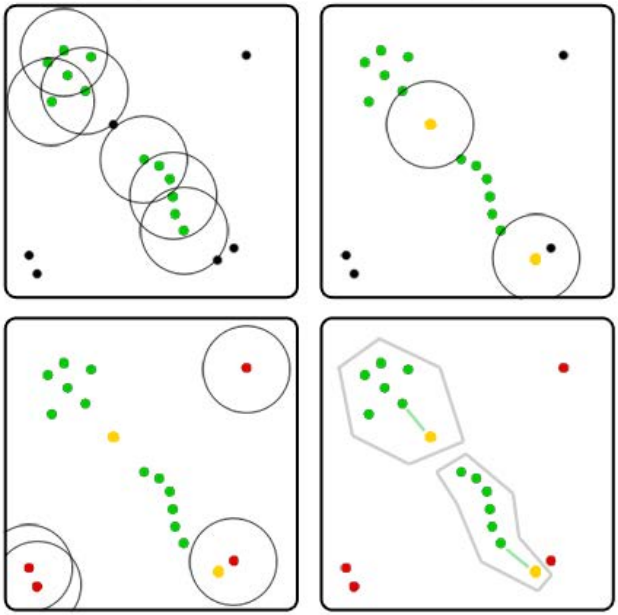
- Звернемося до людей, які мають менше трьох сусідів. Виберемо тих, у яких принаймні один з сусідів має зелений прапор, і вручимо їм жовті прапори. Скажімо, вони знаходяться на межі груп.

- Залишилися одинаки, у яких мало того, що немає трьох сусідів, так ще й жоден з них не тримає зелений прапор. Роздамо їм червоні прапори. Вважатимемо, що вони не належать жодній групі.

Таким чином, якщо від однієї людини до іншої можна створити ланцюжок людей з зеленими прапорцями, то ці дві людини належать до однієї групи. Вочевидь, що це скупчення розділені або порожнім простором чи людьми з жовтими прапорами. Можна їх

пронумерувати: кожен у групі №1 може досягти по ланцюжку рук кожного іншого групи №1, але нікого в №2, №3 тощо. Те саме для інших груп.

Якщо поруч із людиною з жовтим прапорцем є лише один сусід з зеленим прапорцем, то він буде належати тій групі, до якої належить його сусід. Якщо таких сусідів кілька, і в них різні групи, то доведеться вибирати. Тут можна скористатися різними методами – подивитися, хто із сусідів найближчий, наприклад. Прийдеться якось обминати крайові випадки, але нічого страшного.



Як правило, немає сенсу помічати всіх корінних елементів натовпу відразу. Оскільки від кожного корінного елемента групи можна провести ланцюжок до кожного іншого, то все одно з якого починати обхід рано чи пізно знайдеш усіх. Тут краще підходить ітеративний варіант:

1. Підходимо до випадкової людини з натовпу.
2. Якщо поряд з ним менше трьох осіб, переносимо його до списку можливих одинаків (викидів, шумів) і вибираємо когось

іншого.

3. Інакше:

   (а) Виключаємо його зі списку людей, яких треба оминути.

   (б) Вручаємо цій людині зелений прапорець і створюємо нову групу, в якій він поки що єдиний мешканець.

   (в) Обходимо всіх його сусідів. Якщо його сусід уже у списку потенційних одинаків чи поруч із ним мало інших людей, то перед нами край натовпу. Для простоти можна відразу помітити його жовтим прапором, приєднати до групи та продовжити обхід. Якщо сусід теж Має зелений прапорець, то він не починає нову групу, а приєднується до вже створеної; крім того, ми додаємо до списку обходу сусідів сусіда. Повторюємо цей пункт, доки список обходу не виявиться порожнім.

4. Повторюємо кроки 1–3, поки що так чи інакше не обійдемо всіх людей.

5. Розбираємось зі списком викидів. Якщо на кроці 3 ми вже розкидали всіх крайових, то в ньому залишилися тільки викиди-одиначки – можна відразу закінчити. Якщо ні, то треба якось розподілити людей, що залишилися в списку.

5.3.2 Формальний підхід

Уведемо кілька визначень.
Нехай задана деяка симетрична функція відстані $\rho(x, y)$ та константи ϵ і m. Тоді:

1. Назвемо область $E(x)$, для якої $\forall y : \rho(x, y) \leq \epsilon$, ϵ-околиця об'єкта x.

2. Кореневим об'єктом (core point) ступеня m називається об'єкт, ϵ-околиця якого містить щонайменше m об'єктів: $|E(x)| \geq m$.

3. Об'єкт p безпосередньо щільно досягається з об'єкта q, якщо $p \in E(q)$ і q – кореневий об'єкт.

4. Об'єкт p щільно-досяжний з об'єкта q, якщо $\exists p_1, p_2 \dots p_n, p_1 = q, p_n = p$, такі що $\forall i \in 1 \dots n-1 : p_{i+1}$ безпосередньо щільно-досяжний з p_i

Оберемо якийсь кореневий об'єкт p з датасету, позначимо його і помістимо всіх його безпосередньо щільно-досяжних сусідів у список обходу. Тепер для кожного об'єкту q зі списку: позначимо його, і якщо він теж є кореневим, додамо всіх його сусідів до списку обходу. Тривіально доводиться, що кластери помічених точок, сформовані в ході цього алгоритму максимальні (тобто їх не можна розширити ще однією точкою, щоб задовольнялися умови) і зв'язні в сенсі досяжності. Звідси випливає, що якщо ми обійшли не всі точки, можна перезапустити обхід з якогось іншого кореневого об'єкта, і новий кластер не поглине попередній.

5.3.3 Нюанси застосування

В ідеальному випадку DBSCAN може досягти складності $O(N)$, але не варто на це розраховувати. Якщо не перераховувати щоразу $E(x)$ точок, то очікувана складність – $O(N \log N)$. Найгірший випадок (погані дані або брутфорс-реалізація) – $O(N^2)$. Наївні реалізації DBSCAN виділяють $O(N^2)$ пам'яті під матрицю відстаней – це явно надмірно. Багато версій алгоритму вміють працювати з більш щадними структурами даних: sklearn реалізацію можна оптимізувати за допомогою KD-tree.

DBSCAN з невипадковим правилом обробки крайових точок детермінований. Однак більшість реалізацій для прискорення роботи та зменшення кількості параметрів віддають крайові точки першим кластерам, які до них дотяглися. Наприклад, центральна жовта точка на зображенні вище в різних запусках може належати як нижньому, так і верхньому кластеру. Як правило, це не дуже впливає на якість роботи алгоритму, адже через граничні точки кластер все одно не поширюється далі – ситуація, коли точка перескакує з кластера в кластер і "відкриває дорогу" до інших точок, неможлива.

DBSCAN не обчислює самостійно центри кластерів, проте навряд це проблема, особливо враховуючи довільну форму кластерів. Проте DBSCAN автоматично визначає викиди, що досить здорово.

Співвідношення $\frac{m}{\epsilon^n}$, де n – розмірність простору, можна інтуїтивно розглядати як граничну щільність точок даних у сфері простору. Очікується, що при однаковому співвідношенні $\frac{m}{\epsilon^n}$, і результати будуть приблизно однакові. Іноді це справді так, але є причина, чому алгоритму потрібно задати два параметри, а не один. По-перше, типова відстань між точками в різних датасетах – різна: явно задавати радіус доводиться завжди. По-друге, важливу роль відіграють неоднорідності датасету. Що більше m і ϵ, то більше алгоритм схильний "пробачати" варіації щільності в кластерах. З одного боку, це може бути корисно: неприємно побачити у кластері "дірки", де просто не вистачило даних. З іншого боку, це шкідливо, коли між кластерами немає чіткої межі чи шум створює "міст" між скупченнями. Тоді DBSCAN запросто поєднає дві різні групи. У балансі цих параметрів і полягає складність застосування DBSCAN: реальні набори даних містять кластери різної щільності з межами різного ступеня розмитості. В умовах, коли щільність деяких меж між кластерами більша або дорівнює щільності якихось відокремлених кластерів, доводиться чимось жертвувати.

5.3.4 Налаштування параметрів

Існують варіанти DBSCAN, які здатні пом'якшити цю проблему. Ідея полягає у підстроюванні ϵ у різних галузях по ходу роботи алгоритму. На жаль, зростає кількість параметрів алгоритму.

Існують евристики для вибору m та ϵ. Найчастіше застосовується такий метод та його варіації:

1. Обираємо m. Зазвичай використовуються значення від 3 до 9, чим неоднорідніший очікується датасет, і чим більший рівень шуму, тим більшим слід взяти m.

2. Обчислюємо середню відстань по m найближчим сусідам кожної точки. Тобто, якщо $m = 3$, потрібно вибрати трьох найближчих сусідів, скласти відстані й поділити втричі.

3. Сортуємо отримані значення за зростанням та виводимо на екран.

4. Отримуємо зростаючу залежність. Обираємо ϵ десь у смузі, де відбувається найсильніший перегин. Чим більше ϵ, тим більшими вийдуть кластери, і тим менше їх буде.

У будь-якому випадку, головні недоліки DBSCAN – нездатність з'єднувати кластери через прорізи, і, навпаки, здатність пов'язувати різні кластери через щільно населені перемички. Частково тому зі збільшенням розмірності даних n підступний "удар у спину" завдає "прокляття" розмірності: що більше n, то більше місць, де можуть випадково виникнути отвори чи мости. Адекватна кількість точок даних N зростає експоненційно зі збільшенням n.

DBSCAN добре піддається модифікації, одним я яких є схрещення DBSCAN із k-means для прискорення. DBSCAN розпаралелюється, але це нетривіально зробити ефективно. Якщо процес №1 виявив, що множина A – субкластер, а процес №2, що B – субкластер, і $A \cap B \neq \emptyset$, то A і B можна об'єднати. Проблема полягає в тому, щоб вчасно помітити, що перетин множин не порожній, адже постійна синхронізація списків чудово підриває продуктивність.

5.3.5 Підсумок

DBSCAN слід використовувати, коли:

- Датасет є в міру великий ($N \; approx 10^6$). Навіть $N \approx 10^7 - 10^8$ якщо під рукою оптимізована та розпаралена реалізація.
- Наперед відома функція близькості, симетрична, бажано, не дуже складна. KD-Tree оптимізація часто працює тільки з евклідовою відстанню.
- Очікувані кластери складної форми: вкладені та аномальні кластери, згустки даних малої розмірності.
- Щільність меж між згустками менше щільності найменш щільного кластера. Краще якщо кластери зовсім відокремлені один від одного.
- Складність елементів датасета значення не має. Однак їх має бути достатньо, щоб не виникало сильних розривів у густині.

- Кількість елементів у кластері може змінюватись як завгодно.
- Кількість викидів значення не має (в розумних межах), якщо вони розсіяні у великому об'ємі.
- Кількість кластерів значення немає.

5.3.6 Сфери застосування

DBSCAN має історію успішних застосувань. Наприклад, можна відзначити його використання у завданнях:

- Виявлення соціальних гуртків
- Сегментування зображень
- Моделювання поведінки користувачів веб-сайтів
- Попередня обробка в задачі прогнозування погоди

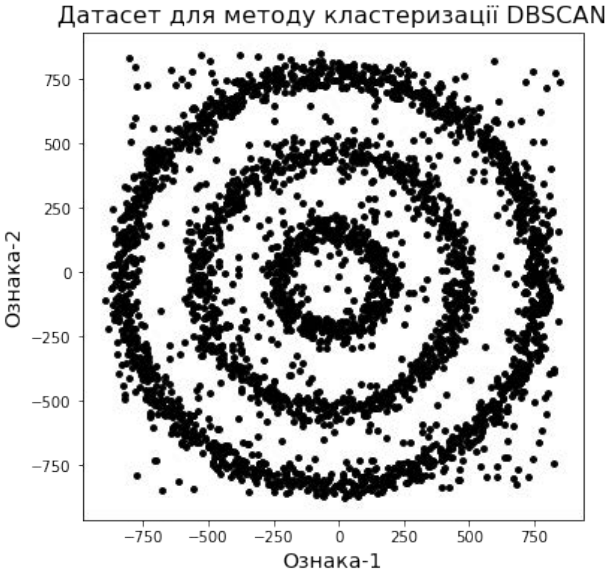
5.3.7 Постановка задачі

Провести кластеризацію даних з використанням методу DBSCAN.
Етапи розв'язання

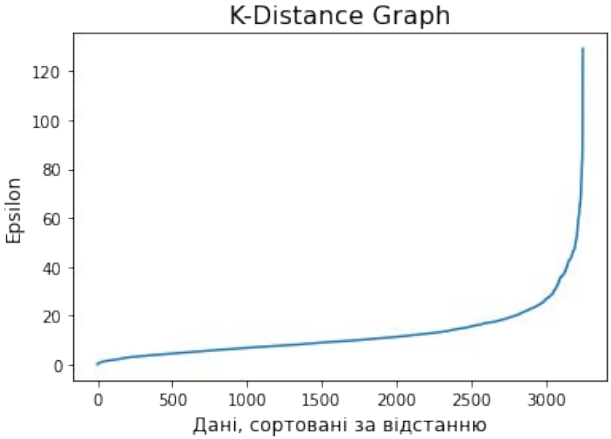
1. Імпортувати/згенерувати розмічені/нерозмічені дані
2. У випадку кількості ознак $n > 2$ звести до $n = 2$ з використанням SVD та PCA
3. Зафіксувати кількість сусідів m, та побудувати K-Distance Graph та визначити оптимальне значення щільності ϵ
4. Провести кластеризацію методом DBSCAN при обраних параметрах m та ϵ
5. Візуалізувати отримані результати
6. Порівняти результати з DBSCAN з sklearn
7. Оформити результати у вигляді звіту.

5.3.8 Приклад подання результатів

Імпортуємо/генеруємо вибірку з розміченими даними. За умови, якщо кількість ознак $n > 2$ зводимо до $n = 2$ (зменшуємо розмірність) за допомогою методів SVD та PCA.

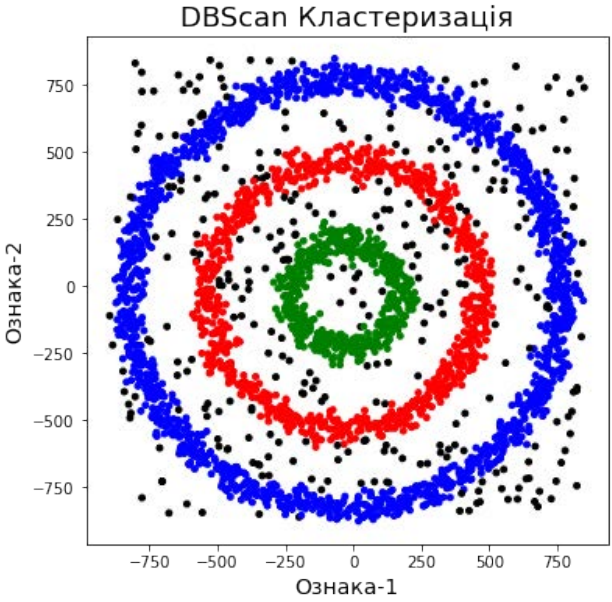


Фіксуємо кількість сусідів $m = 3$, та будуємо K-Distance Graph.



Визначаємо оптимальне значення щільності $\epsilon = 30$

Проводимо кластеризацію даних методом DBSCAN
Візуалізуємо результати



Проводимо порівняння отриманих результатів з вбудованим класифікатором DSSCAN бібліотеки sklearn