МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра прикладної математики та моделювання складних систем

Допущено до захисту
Завідувач кафедри ПМ та МСС
_____ Коплик І.В.
(підпис)
«___» _____2021 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»
спеціальність 113 «Прикладна математика»
освітньо-професійна програма «Наука про дані та моделювання складних систем»

тема роботи: «**Порівняльний аналіз якості методів кластеризації: задача кластеризації італійських вин**»

Виконавець
студент факультету ЕлІТ
Протас Олексій Миколайович _____
(підпис)

Науковий керівник
док. фіз.-мат. наук, професор
Лисенко Олександр Володимирович _____
(підпис)

Суми – 2021

РЕФЕРАТ

Кваліфікаційна робота: 74 с., 25 рис, 12 джерел.

Мета роботи: провести порівняльний аналіз якості різних методів кластеризації на прикладі даних про хімічний склад італійських вин. Визначити фактори, які суттєво впливають на якість кластеризації. Визначити кращий алгоритм кластеризації для досліджуваних даних.

Об'єкт дослідження: методи та алгоритми кластеризації.

Предмет дослідження: характеристики методів та алгоритмів кластеризації, які визначають якість їх роботи.

Методи аналізу: стандартні методи та алгоритми кластеризації бібліотеки scikit-learn; методи та алгоритми оцінки якості кластеризації бібліотеки scikit-learn.

У роботі проведено порівняльний аналіз якості методів кластеризації на прикладі задачі про кластеризацію італійських вин за їх хімічним складом, за даними https://www.kaggle.com/harrywang/wine-dataset-for-clustering. Використовуючи стандартні методи визначено кількість кластерів в досліджуваному наборі даних, що дорівнює трьом. Для підвищення якості кластеризації було запропоновано провести попередню обробку даних, щоб середні значення усіх характеристик досліджуваних об'єктів дорівнювали нулю, а дисперсія – одиниці. Така попередня обробка даних дозволила підвищити точність (accuracy) розпізнавання кластерів з 71% до 97%. З'ясовано, що таке суттєве підвищення якості кластеризації пов'язано зі зміною масштабів ознак, що суттєво вплинуло на відстань між об'єктами. Запропоновано використовувати зміну масштабу ознак для підвищення якості кластеризації. Отримано, що найвища якість кластеризації на досліджуваних даних досягається за допомогою метода K-means (accuracy дорівнює 96,6%).

Ключові слова: КЛАСТЕРНИЙ АНАЛІЗ, ЯКІСТЬ КЛАСТЕРНОГО АНАЛІЗУ, K-MEANS, ACCURACY, SILHOUETTE

ЗМІСТ

ВСТУП

Кластерний аналіз [1-10] є сукупністю методів, які використовуються для групування об'єктів у відповідні категорії. На відміну від задачі класифікації в задачі кластеризації є невідомим ні кількість категорій, ні ознаки за якими потрібно групувати об’єкти. Мета кластерного аналізу – сортування різних об'єктів або точок даних в різні групи кластерів таким чином, щоб ступінь зв'язку між двома об'єктами була високою, якщо вони належать до однієї групи, і низькою, якщо вони належать до різних груп. У широкому сенсі, кластеризація може бути виражена як дослідження невідомого [5]. Кластеризація знаходить широке коло застосувань. Це і пошукові системи, соціальні мережі, візуальні задачі, такі як сегментація зображень, аналіз ДНК. Пошукові системи повинні групувати інформацію щоб мати можливість отримати відповідні дані під час запиту. Соціальні мережі за своєю суттю мають кластерний характер. Сегментація зображень – це візуальне застосування кластеризації. Молекулярна біологія останнім часом стає є багатообіцяючою областю для застосувань кластеризації. Кількість сфер застосувань кластерного аналізу в останній час стрімко збільшується, про що свідчить величезна кількість опублікованих за цією темою статей, оглядів, монографій. Декілька таких публікацій подані в [1-10]. Тому задачі, що пов’язані з кластерним аналізом є важливим і актуальними.

Мета роботи: провести порівняльний аналіз якості різних методів кластеризації на прикладі даних про хімічний склад італійських вин за даними https://www.kaggle.com/harrywang/wine-dataset-for-clustering. Визначити фактори, які суттєво впливають на якість кластеризації. Визначити кращий алгоритм кластеризації для досліджуваних даних.

Об’єкт дослідження: методи та алгоритми кластеризації.

Предмет дослідження: характеристики методів та алгоритмів кластеризації, які визначають якість їх роботи.

Методи аналізу: стандартні методи та алгоритми кластеризації бібліотеки scikit-learn; методи та алгоритми оцінки якості кластеризації бібліотеки scikit-learn.

Основні задачі роботи:

- провести огляд основних методів кластеризації;
- розглянути методи контролю якості алгоритмів кластеризації;
- провести кластерний аналіз даних про хімічний склад італійських вин різними методами;
- виконати порівняльний аналіз якості кожного з методів.

РОЗДІЛ 1
ЗАДАЧА КЛАСТЕРИЗАЦІЇ ТА МЕТОДИ ЇЇ ВИРІШЕННЯ (ОГЛЯД ЛІТЕРАТУРИ)

Кластеризація (англ. Cluster analysis) – задача угруповання множини об'єктів на підмножини (кластери) таким чином, щоб об'єкти з одного кластера були більш схожі один на одного, ніж на об'єкти з інших кластерів по якомусь критерію [1-10]. Кластеризацію можна визначити як задачу групування суб'єктів з точки зору міри подібності. Тут критичне питання полягає в тому, щоб зрозуміти, що означає «подібне». Подібність у певному сенсі є оберненою метрикою відстані між двома об’єктами. Чим коротша відстань, тим більше схожі сутності, і навпаки. Тому важливо зазначити, що результати аналізу будуть суттєво залежати від поняття подібності. Звичайною метрикою відстані є евклідова відстань між двома точками даних. Є багато інших мір подібності, наприклад, [1-10]. Методи кластеризації [1] зазвичай класифікують на чотири основні групи. Перша група заснована на формуванні кластерів методологією, що включає оптимізацію зверху вниз, знизу вгору та аналітичну оптимізацію. Друга група включає методи такі як ієрархічна [4], центроїд ний розподіл (K-середніх [5]), розподіли очікування максимізація [6], розподіли густини [7, 8]. По-третє, можна розглядати жорстку чи м’яку кластеризацію, що відноситься до бінарних або нечітких відношень відповідно. Остання група кластеризації, заснована на природі кластерних відносини, визначає відмінність між перекриттям проти не пересікаючи груп розділів загалом.

Слід зазначити, що проблема кластеризації не є тривіальною задачею, особливо у випадку даних високої розмірності, з яким необхідно працювати у в більшість реальних застосувань. Звичайні методи кластеризації як правило не працюють у таких випадках.

1.1. Постановка задачі кластеризації

Використаємо [1] для формулювання постановки задачі кластеризації. Нехай X - множина об'єктів, Y - множини ідентифікаторів (міток) кластерів. На множині X задана функція відстані між об'єктами ρ (x, x'). Дана кінцева навчальна вибірка об'єктів $X^m = \{x_1, ..., x_m\} \subset X$. Необхідно розбити вибірку на підмножини (кластери), тобто кожному об'єкту $x_i \in X^m$ зіставити мітку $y_i \in Y$, таким чином щоб об'єкти всередині кожного кластера були близькі щодо метрики ρ, а об'єкти з різних кластерів істотно розрізнялися.

Алгоритм кластеризації - функція a: X → Y, яка будь-якого об'єкта x∈X ставить у відповідність ідентифікатор кластера y∈Y.

Рішенням задачі кластерного аналізу є розбиття, що задовольняє деякому критерію оптимальності. Цей критерій може являти собою деякий функціонал, що виражає рівні бажаності різних розбиття і угруповань, який називають цільовою функцією.

Рішення завдання кластеризації принципово неоднозначно, і тому є кілька причин:

- не існує однозначно найкращого критерію якості кластеризації. Відомий цілий ряд евристичних критеріїв, а також ряд алгоритмів, які не мають чітко вираженого критерію, але здійснюють досить розумну кластеризації «з побудови». Всі вони можуть давати різні результати. Отже, для визначення якості кластеризації потрібно експерт предметної області, який би міг оцінити осмисленість виділення кластерів.
- число кластерів, як правило, невідомо заздалегідь і встановлюється відповідно до деякого суб'єктивним критерієм. Це справедливо тільки для методів дискримінації, так як в методах кластеризації виділення кластерів йде за рахунок формалізованого підходу на основі заходів близькості.

- результат кластеризації істотно залежить від метрики, вибір якої, як правило, також суб'єктивний і визначається експертом. Але є ряд рекомендацій щодо вибору заходів близькості для різних завдань.

1.2. Типологія задач кластеризації

Типи вхідних даних

- Опис об'єктів за ознаками. Кожен об'єкт описується набором своїх характеристик, які називаються ознаками (англ. Features). Ознаки можуть бути як числовими, так і категоріальним;
- Матриця відстаней між об'єктами. Кожен об'єкт описується відстанню до всіх об'єктів з навчальної вибірки.

Мета кластеризації:

- Класифікація об'єктів. Спроба зрозуміти залежності між об'єктами шляхом виявлення їх кластерної структури. Розбиття вибірки на групи схожих об'єктів спрощує подальшу обробку даних і прийняття рішень, дозволяє застосувати до кожного кластеру свій метод аналізу (стратегія «розділяй і володарюй»). В даному випадку прагнуть зменшити число кластерів для виявлення найбільш загальних закономірностей;
- Стиснення даних. Можна скоротити розмір вихідної вибірки, взявши один або кілька найбільш типових представників кожного кластера. Тут важливо найбільш точно окреслити межі кожного кластера, їх кількість не є важливим критерієм;
- Виявлення новизни (виявлення шуму). Виділення об'єктів, які не підходять за критеріями ні в один кластер. Виявлені об'єкти в подальшому обробляють окремо.

1.3. Методи кластеризації

1.3.1. Алгоритми ієрархічної кластеризації

Серед алгоритмів ієрархічної кластеризації виділяються два основних типи: висхідні і низхідні алгоритми. Низхідні алгоритми працюють за принципом «зверху-вниз»: на початку всі об'єкти поміщаються в один кластер, який потім розбивається на всі більш дрібні кластери. Більш поширені висхідні алгоритми, які на початку роботи поміщають кожен об'єкт в окремий кластер, а потім об'єднують кластери в більш великі, поки всі об'єкти вибірки не будуть міститися в одному кластері. Таким чином будується система вкладеного розбиття. Результати таких алгоритмів зазвичай представляють у вигляді дерева - дендрограми. До недоліку ієрархічних алгоритмів можна віднести систему повного розбиття, яка може бути зайвою в контексті розв'язуваної задачі.

1.3.2. Алгоритми квадратичної помилки

Завдання кластеризації можна розглядати як побудова оптимального розбиття об'єктів на групи. При цьому оптимальність може бути визначена як вимога мінімізації середньоквадратичної помилки розбиття:

$$e^2(X, L) = \sum_{j=1}^{K} \sum_{i=1}^{n_j} \left|\left| x_i^{(j)} - c_j \right|\right|$$

де c_j- «центр мас» кластера j (точка з середніми значеннями характеристик для даного кластера).

Алгоритми квадратичної помилки відносяться до типу плоских алгоритмів. Найпоширенішим алгоритмом цієї категорії є метод k-середніх. Цей алгоритм будує задане число кластерів, розташованих якнайдалі один від одного. Робота алгоритму ділиться на декілька етапів:

- Випадково вибрати k точок, які є початковими «центрами мас» кластерів.
- Віднести кожен об'єкт до кластеру з найближчим «центром мас».
- Перерахувати «центри мас» кластерів відповідно до їх поточним складом.
- Якщо критерій зупинки алгоритму не задоволений, повернутися до п. 2.

Як критерій зупинки роботи алгоритму зазвичай вибирають мінімальне зміна середньоквадратичної помилки. Так само можливо зупиняти роботу алгоритму, якщо на кроці 2 не було об'єктів, що перемістилися з кластера в кластер.

До недоліків даного алгоритму можна віднести необхідність задавати кількість кластерів для розбиття.

1.3.3. Нечіткі алгоритми

Найбільш популярним алгоритмом нечіткої кластеризації є алгоритм c-середніх (c-means). Він являє собою модифікацію методу k-середніх. Кроки роботи алгоритму:

- Вибрати початкове нечітке розбиття n об'єктів на k кластерів шляхом вибору матриці приналежності U розміру n x k.
- Використовуючи матрицю U, знайти значення критерію нечіткої помилки:

$$E^2(X,U) = \sum_{i=1}^{N}\sum_{k=1}^{K} U_{ik} \left|\left| x_i^{(j)} - c_j \right|\right|$$

де c_k- «центр мас» нечіткого кластера k:

$$c_k = \sum_{i=1}^{N} U_k\, x_i$$

- Перегрупувати об'єкти з метою зменшення цього значення критерію нечіткої помилки.

- Повертатися в п. 2 до тих пір, поки зміни матриці U не стануть незначними.

Цей алгоритм може не підійти, якщо заздалегідь невідомо число кластерів, або необхідно однозначно віднести кожен об'єкт до одного кластеру.

1.3.4. Алгоритми, засновані на теорії графів

Суть таких алгоритмів полягає в тому, що вибірка об'єктів представляється у вигляді графа G = (V, E), вершинам якого відповідають об'єкти, а ребра мають вагу, рівний «відстані» між об'єктами. Перевагою графових алгоритмів кластеризації є наочність, відносна простота реалізації і можливість внесення різних удосконалень, засновані на геометричних міркуваннях. Основними алгоритмами є алгоритм виділення зв'язкових компонент, алгоритм побудови мінімального покриваючого дерева.

1.3.5. Алгоритм виділення зв'язкових компонент

В алгоритмі виділення зв'язкових компонент задається вхідний параметр R і в графі видаляються всі ребра, для яких «відстані» більше R. Сполученими залишаються тільки найбільш близькі пари об'єктів. Сенс алгоритму полягає в тому, щоб підібрати таке значення R, що лежить в діапазон всіх «відстаней», при якому граф «розвалиться» на кілька зв'язкових компонент. Отримані компоненти і є кластери.

Для підбору параметра R зазвичай будується гістограма розподілів попарних відстаней. У завданнях з добре вираженою кластерної структурою даних на гістограмі буде два піки - один відповідає відстаням всередині кластерів, другий - меж кластерним відстаням. Параметр R підбирається із зони мінімуму між цими піками. При цьому управляти кількістю кластерів за допомогою порога відстані досить важко.

1.3.6. Алгоритм мінімального покриваючого дерева

Алгоритм мінімального остового дерева спочатку будує на графі мінімальне покриваюче дерево, а потім послідовно видаляє ребра з найбільшою вагою див.рис. 1.1

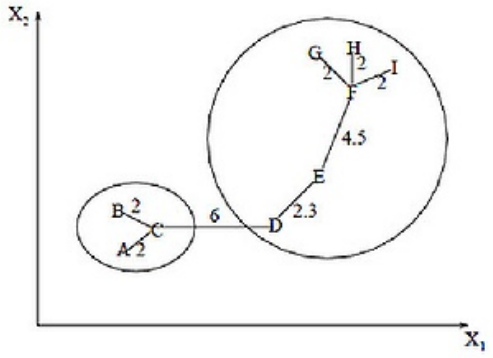


Рис. 1.1 – Мінімальне покриваюче дерево

Шляхом видалення зв'язку з позначкою CD, з довжиною рівною 6 одиницям (ребро з максимальною відстанню), отримуємо два кластери: {A, B, C} і {D, E, F, G, H, I}. Другий кластер в подальшому може бути розділений ще на два кластери шляхом видалення ребра EF, яке має довжину, рівну 4,5 одиницям.

1.4. Порівняння алгоритмів

Таблиця 1

Алгоритм кластеризації	**Обчислювальна складність**
Ієрархічний	$O(n^2)$
k-середніх	O(nkl),де к-число кластерів,l-число ітерацій
c-середніх	
Виділення зв'язкових компонент	Залежить від алгоритму
Мінімальне покриваюче дерево	$O(n^2 \log n)$

Таблиця 2

Алгоритм кластеризації	**Форма кластерів**	**Вхідні дані**	**Результати**
Ієрархічний	Довільна	Число кластерів або поріг відстані	Бінарне дерево кластерів
k-середніх	Гіперсфера	Число кластерів	Центри кластерів
c-середніх	Гіперсфера	Число кластерів, ступінь нечіткості	Центри кластерів, матриця приналежності
Виділення зв'язкових компонент	Довільна	Поріг відстані R	Деревоподібна структура кластерів
Мінімальне покриваюче дерево	Довільна	Число кластерів або поріг відстані для видалення ребер	Деревоподібна структура кластерів

1.5. Метрики якості кластеризації

Виділяють зовнішні і внутрішні метрики якості. Зовнішні використовують інформацію про справжній розбитті на кластери, в той час як внутрішні метрики не використовують ніякої зовнішньої інформації і оцінюють якість кластеризації, базуючись тільки на наборі даних. Оптимальне число кластерів зазвичай визначають з використанням внутрішніх метрик.

Всі зазначені нижче метрики реалізовані в sklearn.metrics.

1.5.1. Adjusted Rand Index (ARI)

Передбачається, що відомі істинні мітки об'єктів. Дана міра не залежить від самих значень міток, а тільки від розбиття вибірки на кластери. Нехай n - число об'єктів у вибірці. Позначимо через a - число пар об'єктів, що мають однакові мітки і знаходяться в одному кластері, через b - число пар об'єктів, що мають різні мітки і знаходяться в різних кластерах. Тоді Rand Index це

$$\mathrm{RI} = \frac{2(a+b)}{n(n-1)}$$

Тобто це частка об'єктів, для яких ці розбиття (вихідне і отримане в результаті кластеризації) "узгоджені". Rand Index (RI) висловлює схожість двох різних кластеризації однієї і тієї ж вибірки. Щоб цей індекс давав значення близькі до нуля для випадкових кластеризації при будь-якому n і числі кластерів, необхідно унормувати його. Так визначається Adjusted Rand Index:

$$ARI = \frac{\mathrm{RI} - E[RI]}{\max(RI) - E[RI]}$$

Цей захід симетричний, не залежить від значень і перестановок міток. Таким чином, даний індекс є мірою відстані між різними розбивками вибірки. ARI приймає значення в діапазоні [- 1, 1]. Негативні значення відповідають "незалежним" розбиття на кластери, значення, близькі до нуля, - випадковим розбиття, і позитивні значення свідчать про те, що два розбиття схожі (збігаються при ARI = 1).

Переваги:

- Інтерпретування : нескоригований індекс Ренда пропорційний кількості пар вибірок, мітки яких однакові в обох labels_predi / labels_true або розрізняються в обох.
- Випадкові (однакові) присвоєння міток мають скоригований бал індексу Rand, близький до 0,0 для будь-якого значення n_clustersi n_samples (що не відноситься, наприклад, до нескоректованому індексу Rand або V-міру).
- Обмежений діапазон: більш низькі значення вказують на різні маркування, аналогічні кластери мають високий (скоригований або нескоригований) індекс Rand, 1,0 - це оцінка ідеального відповідності. Діапазон оцінок становить [0, 1] для нескоректована індексу Rand і [-1, 1] для скоригованого індексу Rand.
- Не робиться ніяких припущень про структуру кластера: (скоригований або нескоригований) індекс Rand може використовуватися для

порівняння всіх видів алгоритмів кластеризації і може використовуватися для порівняння алгоритмів кластеризації, таких як k-середніх, який передбачає ізотропні форми краплі з результатами спектрального аналізу. алгоритми кластеризації, які можуть знайти кластер зі «складеними» формами.

Недоліки:

- Всупереч інерції, (скоригований або нескоригований) індекс Ренда вимагає знання основних класів істинності, що майже ніколи не є на практиці або вимагає ручного призначення аннотаторами -людьми (як в умовах контрольованого навчання). Однак (скоригований або нескоригований) індекс Rand також може бути корисний в чисто неконтрольованих налаштуваннях в якості будівельного блоку для консенсусного індексу, який може використовуватися для вибору моделі кластеризації (TODO).
- Нескоригований індекс Rand часто близько до 1,0, навіть якщо кластеризації істотно розрізняються. Це можна зрозуміти, інтерпретуючи індекс Ренда як точність маркування пар елементів, отриману в результаті кластеризації: на практиці часто існує більшість пар елементів, яким присвоюється different мітка пари як при прогнозованій, так і при базовій кластеризації істинності, що призводить до високої частка парних міток, які згодні, що згодом призводить до високої оцінки.

1.5.2. Гомогенність, повнота, V-міра

Формально ці заходи також визначаються з використанням функцій ентропії і умовної ентропії, розглядаючи розбиття вибірки як дискретні розподілу:

$$h = 1 - \frac{\mathrm{H}(\mathrm{C}\,|\,\mathrm{K})}{\mathrm{H}(\mathrm{C})}, c = 1 - \frac{\mathrm{H}(\mathrm{K}\,|\,\mathrm{C})}{\mathrm{H}(\mathrm{K})}$$

тут K - результат кластеризації, C - справжнє розбиття вибірки на класи. Таким чином, h вимірює, наскільки кожен кластер складається з об'єктів одного класу, а c - наскільки об'єкти одного класу відносяться до одного кластеру. Ці заходи не є симетричними. Обидві величини приймають значення в діапазоні [0, 1] , і великі значення відповідають більш точної кластеризації. Ці заходи не є нормалізованими, як ARI або AMI і тому залежать від числа кластерів. Випадкова кластеризація не даватиме нульові показники при великому числі класів і малому числі об'єктів. У цих випадках краще використовувати ARI. Однак при числі об'єктів більше 1000 і числі кластерів менше 10 дана проблема не так явно виражена і може бути проігнорована.

Для обліку обох величин h і c одночасно вводиться V -заходи, як їх середнє гармонійне:

$$v = 2\frac{hc}{h+c}$$

Переваги:

- Обмежені бали: 0,0 - це настільки погано, наскільки це можливо, 1,0 - ідеальний бал.
- Інтуїтивна інтерпретація: кластеризацію з поганою V-мірою можна якісно проаналізувати з точки зору однорідності та повноти, щоб краще зрозуміти, які «помилки» допускаються при завданні.
- Не робиться ніяких припущень про структуру кластера: може використовуватися для порівняння алгоритмів кластеризації, таких як k-середнє, яке передбачає ізотропні форми краплі, з результатами алгоритмів спектральної кластеризації, які можуть знаходити кластер зі «складеними» формами.

Недоліки:

- Раніше введені метрики НЕ нормалізовані щодо випадкової маркування: це означає, що в залежності від кількості вибірок, кластерів і основних класів істинності повністю випадкова маркування не завжди буде давати однакові значення для однорідності, повноти і, отже, v-заходи.

1.5.3. Силует

На відміну від описаних вище метрик, даний коефіцієнт не передбачає знання істинних міток об'єктів, і дозволяє оцінити якість кластеризації, використовуючи тільки саму (нерозмічену) вибірку і результат кластеризації. Спочатку силует визначається окремо для кожного об'єкта. Позначимо через a - середня відстань від даного об'єкта до об'єктів з того ж кластера, через b - середня відстань від даного об'єкта до об'єктів з найближчого кластера (відмінного від того, в якому лежить сам об'єкт). Тоді силуетом даного об'єкта називається величина:

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

Силуетом вибірки називається середня величина силуету об'єктів даної вибірки. Таким чином, силует показує, наскільки середня відстань до об'єктів свого кластера відрізняється від середньої відстані до об'єктів інших кластерів. Дана величина лежить в діапазоні [- 1, 1] . Значення, близькі до -1, відповідають поганим (розрізненим) кластеризації, значення, близькі до нуля, кажуть про те, що кластери перетинаються і накладаються один на одного, значення, близькі до 1, відповідають "щільним" чітко виділеним кластерам. Таким чином, чим більше силует, тим чіткіше виділені кластери, і вони є компактними, щільно згруповані хмари точок.

За допомогою силуету можна вибирати оптимальне число кластерів k (якщо воно заздалегідь невідомо) - вибирається число кластерів, максимізуючи значення силуету. На відміну від попередніх метрик, силует залежить від форми

кластерів, і досягає великих значень на більш опуклих кластерах, одержуваних за допомогою алгоритмів, заснованих на відновленні щільності розподілу.

Переваги:

- Оцінка обмежена від -1 за неправильну кластеризацію до +1 за високо щільну кластеризацію. Бали близько нуля вказують на перекриваючи кластери.
- Оцінка вище, коли кластери щільні і добре розділені, що відноситься до стандартної концепції кластера.
-

Недоліки:

- Коефіцієнт силуету зазвичай вище для опуклих кластерів, ніж для інших концепцій кластерів, таких як кластери на основі щільності, подібні до тих, які отримані за допомогою DBSCAN.

РОЗДІЛ 2
МЕТОДИ ТА АЛГОРИТМИ РОЗВ'ЯЗКУ ЗАДАЧІ КЛАСТЕРИЗАЦІЇ, ЩО ВИКОРИСТОВУЮТЬСЯ В РОБОТІ

Для вирішення задачі кластеризації у своїй роботі я використовую Python, бібліотеку scikit-learn.

Scikit-learn - одна з найбільш широко використовуваних бібліотек Python для Data Science і Machine Learning. Вона дозволяє виконувати безліч операцій і надає безліч алгоритмів. Scikit-learn також пропонує відмінну документацію про свої класи, методи і функції, а також опис використовуваних алгоритмів. Нижче ми розглянемо опис алгоритмів кластеризації які я використовую у своїй роботі [11-12]

2.1. K-means

Алгоритм K-Means групує дані, намагаючись розділити вибірки на n груп рівної дисперсії, мінімізуючи критерій, відомий як інерція або сума квадратів у межах кластера. Цей алгоритм вимагає вказати кількість кластерів.
Алгоритм k-середніх розділяє набір з N вибірок X на K непересічних кластерів, кожен з яких описується середнім значенням μ_j

зразків у кластері. Засоби зазвичай називають «центроїдами» кластера; зауважте, що вони взагалі не є точками від X, хоча вони живуть в одному просторі.

Мета алгоритму вибрати центроїди, які мінімізують інерцію, або критерій суми квадратів у кластері:

$$\sum_{i=0}^{n} \min_{\mu_j \in C}(||x_i - \mu_j||^2)$$

K-means часто називають алгоритмом Лойда. В основному алгоритм складається з трьох кроків. На першому кроці вибираються початкові центроїди, а найпростішим методом є вибір зразків із набору k даних X. Після ініціалізації K-засоби складаються з циклу між двома іншими кроками. Перший

крок призначає кожен зразок до найближчого центроїда. Другий крок створює нові центроїди, беручи середнє значення всіх зразків, призначених кожному попередньому центроїду. Обчислюється різниця між старим і новим центроїдами, і алгоритм повторює ці останні два кроки, поки це значення не стане меншим за порогове значення. Іншими словами, він повторюється до тих пір, поки центроїди не зрушяться суттєво.

Нижче наведено приклад реалізації алгоритму в бібліотеці.

class* sklearn.cluster.KMeans(*n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='auto')

детально про кожен параметр:

Таблиця 3

Параметри	**n_clusters:** ***int, за замовчуванням = 8*** Кількість кластерів, які необхідно сформувати, а також кількість центроїдів для створення. **init :*{'k-означає ++', 'випадковий'}, викличний або подібний до масиву форми (n_clusters, n_features), default = 'k-mean ++'*** Спосіб ініціалізації: 'k-mean ++': вибирає початкові центри кластерів для k-середнього кластеризації розумним способом для прискорення конвергенції. "випадковий": виберіть випадкові n_clusters спостереження (рядки) з даних для початкових центроїдів. Якщо передається масив, він повинен мати форму (n_clusters, n_features) і надавати початкові центри. Якщо передається виклик, він повинен прийняти аргументи X, n_clusters та випадковий стан та повернути ініціалізацію. **n_init:** ***int, за замовчуванням = 10*** Кількість разів, коли алгоритм k-mean буде виконуватися з різними насіння центроїдів. Кінцеві результати будуть найкращими результатами n_init послідовних прогонів з точки зору інерції. **max_iter:** ***int, за замовчуванням = 300*** Максимальна кількість ітерацій алгоритму k-mean за один пробіг. **tol :*float, за замовчуванням = 1e-4***

Параметри	Відносна толерантність щодо норми Фробеніуса різниці центрів кластера двох послідовних ітерацій для оголошення конвергенції. **Детальний:** ***int, за замовчуванням = 0*** Режим багатослівності. **random_state :*****int, екземпляр RandomState або None, за замовчуванням = немає*** Визначає генерацію випадкових чисел для ініціалізації центроїда. Використовуйте int, щоб зробити випадковість детермінованою. **copy_x :*****bool, за замовчуванням = True*** При попередньому обчисленні відстаней більш точним чисельним є спочатку центрування даних. Якщо copy_x має значення True (за замовчуванням), вихідні дані не змінюються. Якщо значення False, вихідні дані змінюються та повертаються до повернення функції, але невеликі числові відмінності можуть бути введені шляхом віднімання, а потім додавання середнього значення даних. Зауважте, що якщо вихідні дані не є C-суміжними, копію буде зроблено, навіть якщо copy_x має значення False. Якщо вихідні дані розріджені, але не у форматі CSR, копію буде зроблено, навіть якщо copy_x має значення False. **Алгоритм:** ***{"auto", "full", "elkan"}, default = "auto"*** Алгоритм K-означає. Класичний алгоритм у стилі EM "повний". Варіація "elkan" є більш ефективною для даних з чітко визначеними кластерами, використовуючи нерівність трикутника. Однак він потребує більше пам'яті через виділення додаткового масиву фігур (n_samples, n_clusters).
Атрибути	**cluster_centers_** ***ndray of shape (n_clusters, n_features)*** Координати кластерних центрів. Якщо алгоритм зупиниться до повного зближення. **labels_** ***ndarray of shape (n_samples,)*** Мітки кожної точки **інерція_** ***поплавок*** Сума квадратів відстаней зразків до їх найближчого центру кластеру, зважених за вагою вибірки, якщо така передбачена. **n_iter_** ***int*** Кількість запущених ітерацій. **n_особливості_в_** ***int*** Кількість особливостей, які можна побачити під час підгонки. **feature_names_in_** ***ndarray форми (n_features_in_,)*** Назви ознак, які можна побачити під час підгонки . Визначається лише тоді, коли X всі назви функцій є рядками.

Приклад коду:

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

2.2. Affinity propagation

Affinity Propagation створює кластери, надсилаючи повідомлення між парами зразків до конвергенції. Потім набір даних описується за допомогою невеликої кількості зразків, які ідентифікуються як найбільш репрезентативні для інших вибірок. Повідомлення, що надсилаються між парами, представляють придатність однієї вибірки бути прикладом іншої, яка оновлюється у відповідь на значення інших пар. Це оновлення відбувається ітеративно до конвергенції, після чого обираються остаточні зразки, а отже, подається остаточна кластеризація. На рис. 2.1 бачимо приклад кластеризації

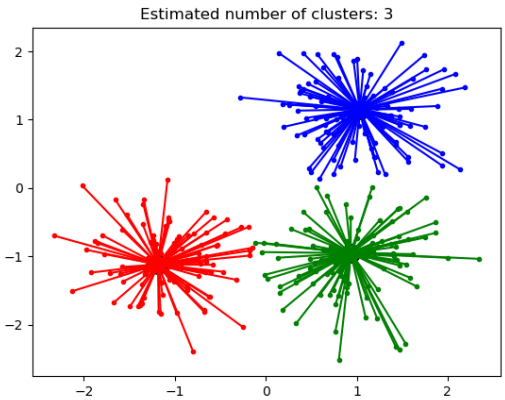


Рис 2.1-кластеризація методом Affinity Propagation

Affinity Propagation може бути цікавим, оскільки воно вибирає кількість кластерів на основі наданих даних. Для цієї мети два важливі параметри - це перевага, яка контролює кількість екземплярів, що використовуються, і

коефіцієнт демпфірування, який послаблює повідомлення про відповідальність та доступність, щоб уникнути числових коливань під час оновлення цих повідомлень.

Основним недоліком Affinity Propagation є його складність. Алгоритм має часову складність порядку O(N^2T), де N - кількість вибірок і T кількість ітерацій до збіжності. Крім того, складність пам'яті має порядок O(N^2), якщо використовується щільна матриця подібності, але скорочується, якщо використовується матриця розрідженої подібності. Це робить Affinity Propagation найбільш підходящим для малих і середніх наборів даних.
Опис алгоритму: Повідомлення, надіслані між точками, належать до однієї з двох категорій. По -перше, це відповідальність *r(i,k),* яка є накопиченим доказом того, що вибірка *k* має бути вибіркою для вибірки *i*. По -друге, це наявність *a(i,k),* яка є накопиченим доказом того, що вибірка *i* має обрати вибірку *k* для зразка і враховує значення всіх інших вибірок, які мають бути зразками. Таким чином, зразки відбираються за вибірками, якщо вони (1) досить подібні до багатьох вибірок і (2) відбираються багатьма вибірками, щоб бути репрезентативними для себе.
Це можна представити формулою:

$$r(i,k) \leftarrow s(i,k) - \max\left[a(i,k') + s(i,k')\ \forall k' \neq k\right]$$

Де *s(i,k)* подібність між вибірками *i ma k.* Можливість вибірки *k* бути зразком вибірки *i* представлено формулою:

$$a(i,k) \leftarrow min[0, r(k,k) + \sum_{i'\ s.t.\ i' \notin \{i,k\}} r(i',k)]$$

Почнемо з того, що всі значення для *r* та *a* встановлені до нуля, а обчислення кожного повторюється до збіжності. Як обговорювалося вище, щоб уникнути числових коливань під час оновлення повідомлень, коефіцієнт демпфірування λ вводиться до процесу ітерації:

$$r_{t+1}(i,k) = \lambda r_t(i,k) + (1-\lambda) r_{t+1}(i,k)$$

$$a_{t+1}(i,k) = \lambda a_t(i,k) + (1-\lambda) a_{t+1}(i,k)$$

Далі розглянемо приклад реалізації та опис параметрів і аргументів:

***class* sklearn.cluster.AffinityPropagation(*, *damping=0.5*, *max_iter=200*, *converg ence_iter=15*, *copy=True*, *preference=None*, *affinity='euclidean'*, *verbose=False*, *r andom_state=None*)**

Таблиця 4

Параметри	**damping:** ***float, за замовчуванням = 0,5*** Коефіцієнт демпфірування в діапазоні - це ступінь збереження поточного значення відносно вхідних значень (зважений 1 - демпфірування). Це для того, щоб уникнути числових коливань при оновленні цих значень (повідомлень).[0.5, 1.0) **max_iter :** ***int, за замовчуванням = 200*** Максимальна кількість ітерацій. **convergence_iter:** ***int, за замовчуванням = 15*** Кількість ітерацій без зміни кількості оцінених кластерів, що зупиняє конвергенцію. **copy :*****bool, default = True*** Зробіть копію вхідних даних. **Preference:** ***у формі масиву (n_samples,) або float, за замовчуванням = немає*** Переваги для кожної точки - точки з більшими значеннями уподобань, швидше за все, будуть обрані зразками. На кількість прикладів, тобто кластерів, впливає значення параметрів введення. Якщо параметри не передаються як аргументи, вони будуть встановлені як медіана вхідних подібностей. **Affinity:** ***{'евклідова', 'попередньо обчислена'}, за замовчуванням = 'евклідова'*** Яку спорідненість використовувати. На даний момент "попередньо обчислювані" і euclidean підтримуються. "евклідів" використовує негативну квадратну евклідову відстань між точками. **Verbose:** ***bool, за замовчуванням = False*** Чи варто бути багатослівним. **Random_state:** ***int, екземпляр RandomState або None, за замовчуванням = немає*** Генератор псевдовипадкових чисел для управління початковим станом.

Атрибути	**cluster_centers_indices_: n** ***масив форми (n_clusters,)*** Індекси кластерних центрів. **cluster_centers_ :*ndray of shape (n_clusters, n_features)*** Кластерні центри (якщо спорідненість! = precomputed). **labels_ :*ndarray of shape (n_samples,)*** Мітки кожної точки. **affinity_matrix_:масив** ***форми (n_зразки, n_samples)*** Зберігає матрицю спорідненості, що використовується в fit. **n_iter_ :*int*** Кількість ітерацій, необхідних для зближення. **n_features_in_:*int*** Кількість особливостей, які можна побачити під час підгонки . *Нове у версії 0.24.* **feature_names_in_ :*ndarray форми (n_features_in_,)*** Назви ознак, які можна побачити під час підгонки . Визначається лише тоді, коли X всі назви функцій є рядками.

Приклад коду:

```
>>> from sklearn.cluster import AffinityPropagation
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [4, 2], [4, 4], [4, 0]])
>>> clustering = AffinityPropagation(random_state=5).fit(X)
>>> clustering
AffinityPropagation(random_state=5)
>>> clustering.labels_
array([0, 0, 0, 1, 1, 1])
>>> clustering.predict([[0, 0], [4, 4]])
array([0, 1])
>>> clustering.cluster_centers_
array([[1, 2],
       [4, 2]])
```

2.3. Mean shift

Кластеризація MeanShift спрямована на виявлення плям у плавній щільності зразків. Це алгоритм на основі центроїдів, який працює, оновлюючи

кандидатів для центроїдів як середнє значення точок у певній області. Потім ці кандидати відфільтровуються на стадії пост-обробки, щоб усунути найближчі дублікати, щоб сформувати остаточний набір центроїдів.

Беручи до уваги кандидата x_i для ітерацій *t,* кандидат оновлюється за наступною формулою:

$$x_i^{t+1} = m(x_i^t)$$

Де N(x_i)- це околиці зразків на певній відстані навколо x_i та m – *це* середній вектор зсуву, який обчислюється для кожного центроїда, який вказує на область максимального збільшення щільності точок. Це обчислюється за допомогою наступного рівняння, ефективно оновлюючи центроїд як середнє значення зразків у його околицях:

$$m(x_i) = \frac{\sum_{x_j \in N(x_i)} K(x_j - x_i) x_j}{\sum_{x_j \in N(x_i)} K(x_j - x_i)}$$

Алгоритм автоматично встановлює кількість кластерів, замість того, щоб покладатися на параметр bandwidth, який визначає розмір області для пошуку. Цей параметр можна встановити вручну, але його можна оцінити за допомогою наданої estimate_bandwidth функції, яка викликається, якщо пропускна здатність не встановлена.

Алгоритм не дуже масштабований, оскільки вимагає кількох пошуків найближчих сусідів під час виконання алгоритму. Алгоритм гарантовано зблизиться, проте алгоритм припинить ітерацію, коли зміна центроїдів буде невеликою. На рис. 2.2 бачимо приклад

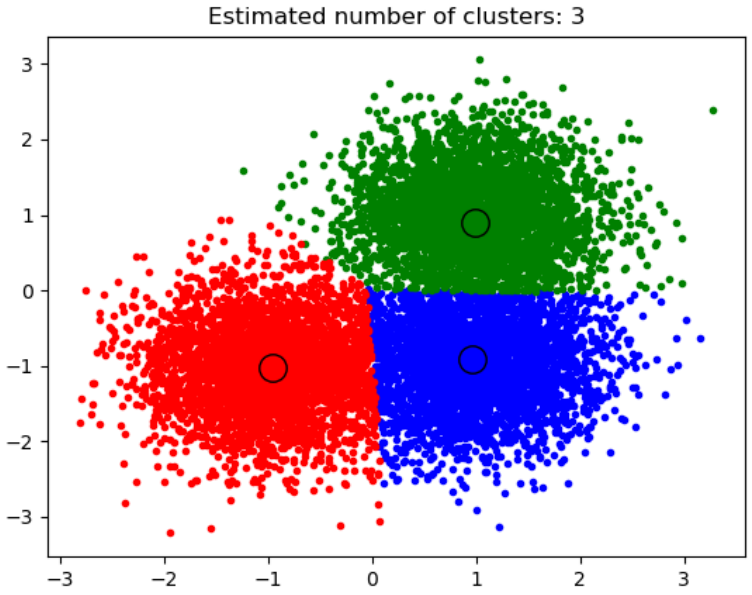


Рис. 2.2 – Приклад розбиття на кластери алгоритму MeanShift

Далі розглянемо приклад реалізації та опис параметрів і аргументів:

***class* sklearn.cluster.MeanShift(*, *bandwidth=None*, *seeds=None*, *bin_seeding=False*, *min_bin_freq=1*, *cluster_all=True*, *n_jobs=None*, *max_iter=300*)**

Таблиця 5

Параметри	**Bandwidth: float, *за замовчуванням = Немає*** Пропускна здатність, що використовується в ядрі RBF. Якщо не вказано, пропускна здатність оцінюється за допомогою sklearn.cluster.estimate_bandwidth; див. документацію до цієї функції для натяків на масштабованість (див. також примітки нижче). **Seeds: *подібні до масиву (n_samples, n_features), за замовчуванням = None*** Насіння, що використовуються для ініціалізації ядер. Якщо не встановлено, насіння обчислюється шляхом clustering.get_bin_seeds з пропускною здатністю як розмір сітки та значеннями за замовчуванням для інших параметрів. **bin_seeding: *bool, за замовчуванням = False*** Якщо це правда, початкові розташування ядра - це не розташування всіх точок, а скоріше місце розташування дискретної версії точок, де точки об'єднані в сітку, грубість якої відповідає пропускній здатності. Встановлення цієї опції на True прискорить роботу алгоритму, оскільки буде ініціалізовано менше насіння. Значення за замовчуванням - False. Ігнорується, якщо аргументом насіння не є None.

	min_bin_freq :*int, за замовчуванням = 1* Щоб прискорити роботу алгоритму, приймайте лише ті бункери, які мають принаймні точки min_bin_freq як насіння. **cluster_all : *bool, за замовчуванням = True*** Якщо це правда, то всі точки кластеризовані, навіть ті сироти, які не входять до ядра. Діти -сироти віднесені до найближчого ядра. Якщо значення false, то сиротам надається кластерна мітка -1. **n_jobs : *int, за замовчуванням = Немає*** Кількість завдань, які потрібно використовувати для обчислення. Це працює шляхом обчислення кожного з n_init запусків паралельно. None означає 1, якщо не в **joblib.parallel_backend** контексті. -1 означає використання всіх процесорів. **max_iter :*int, за замовчуванням = 300*** Максимальна кількість ітерацій на кожну точку початку до завершення операції кластеризації (для цієї точки виходу), якщо ще не зійшлася.
Атрибути	**cluster_centers_: *ndray of shape (n_clusters, n_features)*** Координати кластерних центрів. **labels_ :*ndarray of shape (n_samples,)*** Мітки кожної точки. **n_iter_: *int*** Максимальна кількість ітерацій, виконаних для кожного насіння. **feature_names_in_:*int*** Кількість особливостей, які можна побачити під час підгонки . **feature_names_in_ *ndarray форми (n_features_in_,)*** Назви ознак, які можна побачити під час підгонки . Визначається лише тоді, коли X всі назви функцій є рядками. Кількість ітерацій, необхідних для зближення. **n_features_in_:*int*** Кількість особливостей, які можна побачити під час підгонки . **feature_names_in_ :*ndarray форми (n_features_in_,)*** Назви ознак, які можна побачити під час підгонки . Визначається лише тоді, коли X всі назви функцій є рядками.

Приклад коду:

```
>>> from sklearn.cluster import MeanShift
>>> import numpy as np
>>> X = np.array([[1, 1], [2, 1], [1, 0],
...               [4, 7], [3, 5], [3, 6]])
>>> clustering = MeanShift(bandwidth=2).fit(X)
>>> clustering.labels_
array([1, 1, 1, 0, 0, 0])
>>> clustering.predict([[0, 0], [5, 5]])
array([1, 0])
>>> clustering
MeanShift(bandwidth=2)
```

2.4. Spectral clustering

Спектральна кластеризація виконує низько розмірне вбудовування матриці спорідненості між зразками з подальшим кластеризацією, наприклад, KMeans, компонентів власних векторів у просторі низьких розмірів. Це особливо обчислювально ефективно, якщо матриця спорідненості є розрідженою і розв'язувач amg використовується для вирішення проблеми власного значення (Зверніть увагу, що для розв'язування amg потрібно встановити модуль pyamg.)

Ця версія SpectralClustering вимагає, щоб кількість кластерів було вказано заздалегідь. Він добре працює для невеликої кількості кластерів, але не рекомендується для багатьох кластерів.

Для двох кластерів SpectralClustering вирішує опуклу релаксацію задачі нормалізованих розрізів на графіку подібності: розрізання графа надвоє, так що вага розрізаних ребер буде малим у порівнянні з вагою ребер усередині кожного кластера.

Далі розглянемо приклад реалізації та опис параметрів і аргументів:

***class* sklearn.cluster.SpectralClustering(*n_clusters=8, *, eigen_solver=None, n_components=None, random_state=None, n_init=10, gamma=1.0, affinity='rbf', n_neighbors=10, eigen_tol=0.0, assign_labels='kmeans', degree=3, coef0=1, kernel_params=None, n_jobs=None, verbose=False*)**

Таблиця 6

Параметри	**n_clusters:** ***int, за замовчуванням = 8*** Розмір підпростору проекції. **eigen_solver:** ***{'arpack', 'lobpcg', 'amg'}, за замовчуванням = Немає*** Використовувати стратегію розкладання власних значень. AMG вимагає встановлення pyamg. Це може бути швидше при дуже великих, рідкісних проблемах, але також може призвести до нестабільності. Якщо немає, то 'arpack'використовується. **n_components:*****int, за замовчуванням = n_кластерів*** Кількість власних векторів для використання для спектрального вбудовування **random_state:*****int,екземпляр RandomState, за замовчуванням = Немає*** Генератор псевдо випадкових чисел, що використовується для ініціалізації розкладання власних векторів lobpcg, коли та для ініціалізації K-Means. Використовуйте int, щоб зробити результати детермінованими для всіх викликів (див. Глосарій).eigen_solver == 'amg' **n_init :*****int, за замовчуванням = 10*** Кількість разів, коли алгоритм k-mean буде запускатися з різними насіння центроїдів. Кінцеві результати будуть найкращими результатами n_init послідовних прогонів з точки зору інерції. Використовується тільки якщо assign_labels='kmeans'. **Gamma :** ***float, за замовчуванням = 1,0*** Коефіцієнт ядра для ядер rbf, poly, sigmoid, laplacian і chi2. Проігноровано для affinity='nearest_neighbors'. **Affinity:*****str or callable, по замовчуванням = «RBF»*** **Як побудувати матрицю спорідненості.** • 'nearest_neighbors': побудуйте матрицю спорідненості, обчисливши графік найближчих сусідів. • 'rbf': побудуйте матрицю спорідненості, використовуючи ядро радіальної базисної функції (RBF). • "попередньо обчислене": інтерпретувати Хяк попередньо обчислену матрицю спорідненості, де більші значення вказують на більшу подібність між екземплярами. • 'precomputed_nearest_neighbors': інтерпретувати Хяк розріджений графік попередньо обчислених відстаней і побудувати двійкову матрицю спорідненості з n_neighborsнайближчих сусідів кожного екземпляра. • одне з ядер, підтримуваних pairwise_kernels.

Параметри	Слід використовувати лише ядра, які дають оцінки подібності (невід'ємні значення, які зростають із подібністю). Ця властивість не перевіряється алгоритмом кластеризації. **n_neighbors :*int, за замовчуванням = 10*** Кількість сусідів для використання при побудові матриці спорідненості за допомогою методу найближчих сусідів. Проігноровано для affinity='rbf'. **eigen_tol :*float, за замовчуванням = 0,0*** Критерій зупинки для власного розкладу лапласіанської матриці, коли eigen_solver='arpack'. **assign_labels :*{'kmeans', 'diskretize'}, default = 'kmeans'*** Стратегія призначення міток у просторі вбудовування. Існує два способи призначення міток після вкладання Лапласіана. k-means-популярний вибір, але він може бути чутливим до ініціалізації. Дискретизація - це ще один підхід, який менш чутливий до випадкової ініціалізації. **Degree:*float, за замовчуванням = 3*** Ступінь ядра полінома. Ігнорується іншими ядрами. **coef0 :*float, за замовчуванням = 1*** Нульовий коефіцієнт для поліноміальних і сигмоподібних ядер. Ігнорується іншими ядрами. **kernel_params: *dict від str до any, default = None*** Параметри (аргументи ключових слів) та значення ядра передаються як об'єкт, що викликається. Ігнорується іншими ядрами. **n_jobs: *int, за замовчуванням = Немає*** Кількість паралельних завдань, які потрібно виконати, коли affinity='nearest_neighbors' або affinity='precomputed_nearest_neighbors'. Паралельно проводитиметься пошук сусідів. Noneозначає 1, якщо не в **joblib.parallel_backend**контексті. -1означає використання всіх процесорів. Докладніше див. У глосарії . **Verbose: *bool, за замовчуванням = False*** Режим багатослівності.
Атрибути	**affinity_matrix_: *(n_samples, n_samples)*** Матриця спорідненості, яка використовується для кластеризації. Доступно тільки після дзвінка fit.

	labels_ ***ndarray of shape (n_samples,)*** Мітки кожної точки **n_features_in_: *int*** Кількість особливостей, які можна побачити під час підгонки . **feature_names_in_: *ndarray форми (n_features_in_,)*** Назви ознак, які можна побачити під час підгонки . Визначається лише тоді, коли X всі назви функцій є рядками.

Приклад коду:

```
>>> from sklearn.cluster import SpectralClustering
>>> import numpy as np
>>> X = np.array([[1, 1], [2, 1], [1, 0],
...               [4, 7], [3, 5], [3, 6]])
>>> clustering = SpectralClustering(n_clusters=2,
...         assign_labels='discretize',
...         random_state=0).fit(X)
>>> clustering.labels_
array([1, 1, 1, 0, 0, 0])
>>> clustering
SpectralClustering(assign_labels='discretize', n_clusters=2,
    random_state=0)
```

2.5. Hierarchical clustering

Ієрархічна кластеризація - це загальне сімейство алгоритмів кластеризації, яке створює вкладені кластери шляхом їх об’єднання або послідовного поділу. Ця ієрархія кластерів представлена у вигляді дерева (або дендрограми). Корінь дерева - це унікальний кластер, який збирає всі зразки, а листя - це кластери лише з одним зразком.

Агломеративна кластеризація напевно самий простий і зрозумілий алгоритм кластеризації без фіксованого числа кластерів. Інтуїція у алгоритму дуже проста:

1. Починаємо з того, що присвоюємо кожній точці свій кластер
2. Сортуємо попарні відстані між центрами кластерів по зростанню
3. Беремо пару найближчих кластерів, склеюємо їх в один і перераховуємо центр кластера
4. Повторюємо п. 2 і 3 до тих пір, поки всі дані не склеїти в один кластер

Сам процес пошуку найближчих кластерів може відбуватися з використанням різних методів об'єднання точок:

1. Single linkage - мінімум попарних відстаней між точками з двох кластерів
   $d(C_i, C_j) = min_{x_i \in C_i, x_j \in C_j} ||x_i - x_j||$
2. Complete linkage - максимум попарних відстаней між точками з двох кластерів
   $d(C_i, C_j) = max_{x_i \in C_i, x_j \in C_j} ||x_i - x_j||$
3. Average linkage - середнє попарних відстаней між точками з двох кластерів
   $d(C_i, C_j) = \frac{1}{n_i n_j} \sum_{x_i \in C_i} \sum_{x_j \in C_j} ||x_i - x_j||$
4. Centroid linkage - відстань між центроїдами двох кластерів
   $d(C_i, C_j) = ||\mu_i - \mu_j||$

Переваги перших трьох підходів у порівнянні з четвертим в тому, що для них не потрібно буде перераховувати відстані кожен раз після склеювання, що сильно знижує обчислювальну складність алгоритму.

За підсумками виконання такого алгоритму можна також побудувати чудове дерево склеювання кластерів і дивлячись на нього визначити, на якому етапі нам було б оптимальніше всього зупинити алгоритм. Або скористатися тим же правилом ліктя, що і в k-means.

Далі розглянемо приклад реалізації та опис параметрів і аргументів:

***class* sklearn.cluster.AgglomerativeClustering(*n_clusters=2, *, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None, compute_distances=False*)**

Таблиця 7

Параметри	**n_clusters :*int або None, за замовчуванням = 2*** Кількість кластерів для пошуку. Має бути, Noneякщо distance_thresholdні None. **afinity :*str або callable, default = 'euclidean'*** Метрика, що використовується для обчислення зв'язку. Може бути "євклідовим", "l1", "l2", "манхеттенським", "косинусом" або "попередньо обчисленим". Якщо зв'язок "палата", приймається лише "евклідова". Якщо "попередньо обчислюється", матриця відстані (замість матриці подібності) потрібна як вхід для методу fit.

Параметри	**Memory:*str або об'єкт* пам'яті *з інтерфейсом joblib.Memory, за замовчуванням = Немає*** Використовується для кешування результатів обчислень дерева. За замовчуванням кешування не виконується. Якщо задано рядок, це шлях до каталогу кешування. **Connectivity**: ***подібний до масиву або викликається, за замовчуванням = Немає*** Матриця зв'язності. Визначає для кожної вибірки сусідні вибірки відповідно до заданої структури даних. Це може бути сама матриця зв'язку або виклик, який перетворює дані в матрицю зв'язку, наприклад похідну від kneighbors_graph. За замовчуванням є None, тобто ієрархічний алгоритм кластеризації є неструктурованим. **compute_full_tree :*'auto' або bool, default = 'auto'*** Зупинити будівництво дерева на n_clusters. Це корисно для зменшення часу обчислень, якщо кількість кластерів не мала порівняно з кількістю вибірок. Ця опція є корисною лише при визначенні матриці підключення. Зауважте також, що при зміні кількості кластерів та використанні кешування може бути вигідним обчислити повне дерево. Має бути True якщо distance_thresholdні None. За замовчуванням compute_full_tree"авто", що еквівалентно True коли distance_threshold не None або що n_clusters поступається максимуму між 100 або . В іншому випадку "auto" еквівалентно .0.02 * n_samplesFalse **Linkage: *{'прихід', 'повний', 'середній', 'єдиний'}, за замовчуванням = 'прихід'*** Який критерій зв'язку використати. Критерій зв'язку визначає, яку відстань використовувати між наборами спостережень. Алгоритм об'єднає пари кластерів, які мінімізують цей критерій. • "прихід" мінімізує дисперсію об'єднаних кластерів. • "середнє" використовує середнє значення відстаней кожного спостереження двох наборів. • "повний" або "максимальний" зв'язок використовує максимальні відстані між усіма спостереженнями двох наборів. • 'single' використовує мінімальну відстань між усіма спостереженнями двох наборів. **distance_threshold :*float, за замовчуванням = немає*** Поріг відстані зв'язку, вище якого кластери не об'єднаються. Якщо ні None, то n_clustersмає бути Noneі compute_full_treeповинно бути True. **compute_distances :*bool, default = False*** Обчислює відстані між кластерами, навіть якщо distance_thresholdвони не використовуються. Це може бути використано для візуалізації дендрограми, але вводить обчислювальні та оперативні витрати на пам'ять.

	Продовження таблиці 7
Атрибути	**n_clusters_ :*int*** Кількість кластерів, знайдених алгоритмом. Якщо distance_threshold=None, то це буде дорівнює даному n_clusters. **labels_ *%ndarray of shape (n_samples)*** Кластерні мітки для кожної точки. **n_leaves_: *int*** Кількість листків в ієрархічному дереві. **n_connected_components_: *int*** Орієнтовна кількість підключених компонентів на графіку. **n_features_in_:*int*** Кількість особливостей, які можна побачити під час підгонки . **feature_names_in_ :*ndarray форми (n_features_in_,)*** Назви ознак, які можна побачити під час підгонки . Визначається лише тоді, коли X всі назви функцій є рядками. **children_:*масив, подібний до форми (n_samples-1, 2)*** Діти кожного нелистового вузла. Значення менше, ніж n_samples відповідають листям дерева, які є вихідними зразками. Вузол, ібільший або рівний, n_samplesє нелистовим і має дочірніх вузлів . Крім того, на i-й ітерації діти [i] [0] та діти [i] [1] об’єднуються, утворюючи вузол .children_[i - n_samples]n_samples + i **distances_:*подібні до масиву форми (n_nodes-1,)*** Відстані між вузлами у відповідному місці в children_. Обчислюється, лише якщо distance_threshold використовується або compute_distances встановлено значення True.

Приклад коду:

```
>>> from sklearn.cluster import AgglomerativeClustering
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [4, 2], [4, 4], [4, 0]])
>>> clustering = AgglomerativeClustering().fit(X)
>>> clustering
AgglomerativeClustering()
>>> clustering.labels_
array([1, 1, 1, 0, 0, 0])
```

2.6. DBSCAN

Алгоритм DBSCAN розглядає кластери як області високої щільності, розділені областями низької щільності. Завдяки цьому досить загальному вигляду кластери, знайдені DBSCAN, можуть мати будь-яку форму, на відміну від k-середніх, які передбачають, що кластери мають опуклу форму. Центральним компонентом DBSCAN є концепція основних зразків, які є зразками, що знаходяться в зонах високої щільності. Отже, кластер-це набір основних зразків, кожен з яких близький один до одного (вимірюється деякою мірою відстані), і набір непрофільних зразків, які близькі до вибірки ядра (але самі по собі не є зразками ядра). В алгоритмі є два параметри, min_samples і eps, які формально визначають, що ми маємо на увазі, коли говоримо щільним. Вищі min_sample або нижчі eps вказують на більшу щільність, необхідну для формування кластера.

Більш формально, ми визначаємо основну вибірку як вибірку в наборі даних, так що існують min_samples інших вибірок на відстані eps, які визначені як сусіди базової вибірки. Це говорить нам про те, що зразок ядра знаходиться в щільній зоні векторного простору. Кластер - це набір основних зразків, які можна побудувати шляхом рекурсивного відбору вибірки ядра, пошуку всіх його сусідів, які є вибірковими ядрами, пошуку всіх їх сусідів, які є вибірковими ядрами, тощо. Кластер також має набір неосновних вибірок, які є вибірками, які є сусідами основної вибірки в кластері, але самі по собі не є вибірками ядра. Інтуїтивно ці зразки знаходяться на межі кластера.
Будь -яка основна вибірка є частиною кластера, за визначенням. Алгоритм вважає будь -яку вибірку, яка не є зразком ядра, і знаходиться на відстані щонайменше eps від будь -якої вибірки ядра.

Хоча параметр min_samples насамперед контролює, наскільки алгоритм толерантний до шуму (на шумних і великих наборах даних може бути бажаним збільшити цей параметр), параметр eps має вирішальне значення для

правильного вибору набору даних та функції відстані, і зазвичай його не можна залишити за значенням за замовчуванням. Він контролює локальне сусідство точок. Якщо їх вибрати занадто мало, більшість даних взагалі не будуть кластеризовані (і позначені як -1 для "шуму"). Якщо його вибрати занадто великим, це призводить до об'єднання близьких кластерів в один кластер, і зрештою весь набір даних повертається як єдиний кластер. Деякі евристики для вибору цього параметра обговорювалися в літературі, наприклад, на основі коліна на графіку відстаней найближчого сусіда

Далі розглянемо приклад реалізації та опис параметрів і аргументів:

***class* sklearn.cluster.DBSCAN(*eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None*)**

Таблиця 8

Параметри	**eps** ***float, за замовчуванням = 0,5*** Максимальну відстань між двома зразками для одного вважатиметься як сусідство іншого. Це не максимальна межа на відстанях точок у кластері. Це найважливіший параметр DBSCAN для вибору відповідним чином для вашого набору даних та функції відстані. **min_samples** ***int, за замовчуванням = 5*** Кількість вибірок (або загальна вага) в околиці для точки, що вважається основною точкою. Сюди входить і сама точка. **metric** ***str або callable, default = 'euclidean'*** Метрика, яка використовується для обчислення відстані між екземплярами в масиві об'єктів. Якщо метрика є рядком або викликається, це має бути один із параметрів, дозволених **sklearn.metrics.pairwise_distances**для параметра метрики. Якщо метрика "попередньо обчислюється", X вважається матрицею відстані і має бути квадратною. X може бути Глосарієм , і в цьому випадку сусідами для DBSCAN можуть вважатися лише «ненульові» елементи. *Нове у версії 0.17: попередньо обчислена* метрика для прийняття попередньо обчисленої розрідженої матриці. **metric_params** ***dict, за замовчуванням = Немає*** Додаткові аргументи ключового слова для метричної функції. *Нове у версії 0.19.* **алгоритм** ***{'auto', 'ball_tree', 'kd_tree', 'brute'}, default = 'auto'*** Алгоритм, який буде використовуватися модулем NearestNeighbors для обчислення точкових відстаней та пошуку найближчих

Параметри	сусідів. Докладніше див. У документації до модуля NearestNeighbors. **leaf_size** ***int, за замовчуванням = 30*** Розмір листя передається BallTree або cKDTree. Це може вплинути на швидкість побудови та запиту, а також на пам'ять, необхідну для зберігання дерева. Оптимальне значення залежить від характеру проблеми. **p** ***float, за замовчуванням = Немає*** Потужність метрики Мінковського, яка буде використовуватися для обчислення відстані між точками. Якщо немає, то p=2(еквівалентно евклідовій відстані). **n_jobs** ***int, за замовчуванням = Немає*** Кількість паралельних завдань для виконання. Noneозначає 1, якщо не в **joblib.parallel_backend**контексті. -1означає використання всіх процесорів.
Атрибути	**core_sample_indices_** ***ndray of shape (n_core_samples,)*** Індекси зразків керна. **складові_набір** ***форми (n_яскраві_зразки, n_особливості)*** Копії кожного основного зразка, знайденого під час навчання. **labels_** ***ndarray of shape (n_samples)*** Кластерні мітки для кожної точки в наборі даних, надані fit (). Шумним зразкам надається мітка -1. **n_особливості_в_** ***int*** Кількість особливостей, які можна побачити під час підгонки . **feature_names_in_** ***ndarray форми (n_features_in_,)*** Назви ознак, які можна побачити під час підгонки . Визначається лише тоді, коли X всі назви функцій є рядками.

Приклад коду

```
>>> from sklearn.cluster import DBSCAN
>>> import numpy as np
>>> X = np.array([[1, 2], [2, 2], [2, 3],
...               [8, 7], [8, 8], [25, 80]])
>>> clustering = DBSCAN(eps=3, min_samples=2).fit(X)
>>> clustering.labels_
array([ 0,  0,  0,  1,  1, -1])
>>> clustering
DBSCAN(eps=3, min_samples=2)
```

2.7. OPTICS

Алгоритм OPTICS має багато спільного з алгоритмом DBSCAN, і його можна вважати узагальненням DBSCAN, яке послаблює вимогу eps від одного значення до діапазону значень. Ключова відмінність між DBSCAN та OPTICS полягає в тому, що алгоритм OPTICS будує графік досяжності, який призначає кожному зразку як відстань досяжності_, так і місце в атрибуті упорядкування кластера; ці два атрибути призначаються, коли модель встановлюється, і використовуються для визначення членства в кластері. Якщо OPTICS виконується зі значенням inf за замовчуванням, встановленим для max_eps, тоді вилучення кластера стилю DBSCAN можна виконувати повторно протягом лінійного часу для будь -якого заданого значення eps за допомогою методу cluster_optics_dbscan. Встановлення max_eps на нижче значення призведе до скорочення часу виконання, і його можна розглядати як максимальний радіус сусідства від кожної точки для пошуку інших потенційних точок досяжності див. рис. 2.3.

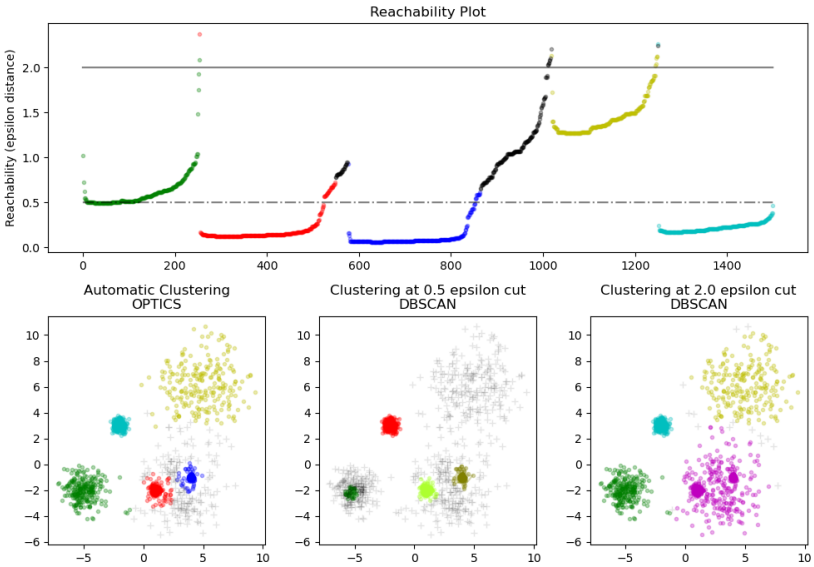


Рис. 2.3 – Приклад алгоритму OPTCS

Відстані досяжності, що генеруються OPTICS, дозволяють видобувати кластери зі змінною щільністю в межах одного набору даних. Як показано на наведеному вище графіку, поєднання відстаней досяжності та впорядкування набору даних_ створює графік досяжності, де щільність точок представлена на

осі Y, а точки впорядковані так, що сусідні точки розташовані поруч. "Зрізання" діаграми досяжності за одним значенням дає результати, подібні до DBSCAN; усі точки над "вирізом" класифікуються як шум, і кожен раз, коли відбувається перерва при читанні зліва направо, означає новий кластер. Вилучення кластерів за замовчуванням за допомогою OPTICS дивиться на круті схили всередині графіка, щоб знайти кластери, і користувач може визначити, що вважається крутим схилом, використовуючи параметр xi. Існують також інші можливості для аналізу на самому графіку, такі як створення ієрархічних уявлень даних за допомогою дендрограм графіку досяжності, а до ієрархії кластерів, виявлених алгоритмом, можна отримати доступ за допомогою параметра cluster_hierarchy_. Діаграма вище була позначена кольором, щоб кольори кластерів у площинному просторі відповідали кластерам лінійних сегментів на графіку досяжності. Зауважте, що синій та червоний кластери розташовані поруч із діаграмою досяжності та можуть бути представлені ієрархічно як дочірні елементи великого батьківського кластеру.

ОПТИКА (Точки впорядкування для ідентифікації кластерної структури), тісно пов'язана з DBSCAN, знаходить вибірку ядра високої щільності та розширює з них кластери . На відміну від DBSCAN, зберігає ієрархію кластерів для змінного радіусу сусідства. Краще підходить для використання у великих наборах даних, ніж поточна реалізація DBSCAN у склеарні.

Потім кластери виймаються за допомогою DBSCAN-подібного методу (cluster_method = 'dbscan') або автоматичної техніки, запропонованої в (cluster_method = 'xi').

Ця реалізація відхиляється від початкової OPTICS, спочатку виконуючи пошук k-найближчих околиць у всіх точках для визначення розмірів ядра, потім обчислюючи лише відстані до необроблених точок при побудові порядку кластера. Зауважте, що ми не використовуємо купу для управління кандидатами на розширення, тому часова складність складе O (n^2).

Далі розглянемо приклад реалізації та опис параметрів і аргументів:

***class* sklearn.cluster.OPTICS(*, *min_samples=5*, *max_eps=inf*, *metric='minkowski'*, *p=2*, *metric_params=None*, *cluster_method='xi'*, *eps=None*, *xi=0.05*, *predecessor_correction=True*, *min_cluster_size=None*, *algorithm='auto'*, *leaf_size=30*, *memory=None*, *n_jobs=None*)**

Таблиця 9

Параметри	**min_samples** ***int> 1 або float між 0 і 1, за замовчуванням = 5*** Кількість вибірок у околиці для точки, яка буде вважатися основною точкою. Крім того, круті регіони вгору і вниз не можуть мати більше ніж min_samples послідовно не круті точки. Виражається як абсолютне число або частка від кількості зразків (округляється до щонайменше 2). **max_eps** ***плаває, за замовчуванням = np.inf*** Максимальну відстань між двома зразками для одного вважатиметься як сусідство іншого. Значення за замовчуванням np.inf визначатиме кластери у всіх масштабах; скорочення max_eps призведе до скорочення часу роботи. **metric** ***str або викликається, за замовчуванням = 'minkowski'*** Метричні дані для обчислення відстані. Можна використовувати будь-які показники з scikit-learn або scipy.spatial.distance. Якщо метрика - це функція, яку можна викликати, вона викликається для кожної пари екземплярів (рядків), а отримане значення записується. Виклик повинен брати два масиви як вхідні дані і повертати одне значення, що вказує відстань між ними. Це працює для метрик Scipy, але менш ефективно, ніж передача імені метрики як рядка. Якщо метрика "попередньо обчислюється", X вважається матрицею відстані і має бути квадратною. Дійсні значення для показників: • з scikit-learn: ['cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan'] • з scipy.spatial.distance: ['braycurtis', 'canberra', 'chebyshev', 'correlation', 'dice', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'minkowski', 'rogerstanimoto ',' russellrao ',' seuclidean ',' sokalmichener ',' sokalsneath ',' sqeuclidean ',' yule '] Докладніше про ці показники див. У документації для scipy.spatial.distance. **p** ***int, за замовчуванням = 2***

Параметри	Параметр для метрики Мінковського з **pairwise_distances**. Якщо p = 1, це еквівалентно використанню manhattan_distance (l1) та euclidean_distance (l2) для p = 2. Для довільного p використовується minkowski_distance (l_p). **metric_params** ***dict, за замовчуванням = Немає*** Додаткові аргументи ключового слова для метричної функції. **cluster_method** ***str, default = 'xi'*** Метод вилучення, який використовується для вилучення кластерів з використанням розрахункової досяжності та впорядкування. Можливі значення "xi" та "dbscan". **eps** ***float, за замовчуванням = немає*** Максимальну відстань між двома зразками для одного вважатиметься як сусідство іншого. За замовчуванням він приймає те саме значення, що і max_eps. Використовується тільки тоді, коли cluster_method='dbscan'. **xi** ***float між 0 і 1, за замовчуванням = 0,05*** Визначає мінімальну крутизну на ділянці досяжності, що становить кордон кластера. Наприклад, точка вгору на графіку досяжності визначається співвідношенням від однієї точки до її наступника не більше 1-xi. Використовується тільки тоді, коли cluster_method='xi'. **predecessor_correction*****bool, за замовчуванням = Істина*** Правильні кластери відповідно до попередників, розрахованих OPTICS . Цей параметр надає мінімальний вплив на більшість наборів даних. Використовується тільки тоді, коли cluster_method='xi'. **min_cluster_size** ***int> 1 або плавати між 0 і 1, за замовчуванням = Немає*** Мінімальна кількість вибірок у кластері OPTICS, виражена як абсолютне число або частка від кількості зразків (округлена до щонайменше 2). Якщо None значення min_samples використовується замість цього. Використовується тільки тоді, коли cluster_method='xi'. **algorithm** ***{'auto', 'ball_tree', 'kd_tree', 'brute'}, default = 'auto'*** Алгоритм обчислення найближчих сусідів: • 'ball_tree' буде використовувати BallTree • 'kd_tree' буде використовувати KDTree • 'brute' буде використовувати грубий пошук. • 'auto' спробує визначити найбільш прийнятний алгоритм на основі значень, переданих **fit** методу. (за замовчуванням)

	Примітка: встановлення на розрідженому вході перекриє налаштування цього параметра, використовуючи грубу силу. **leaf_size** ***int, за замовчуванням = 30*** Розмір листка передано BallTreeабо KDTree. Це може вплинути на швидкість побудови та запиту, а також на пам'ять, необхідну для зберігання дерева. Оптимальне значення залежить від характеру проблеми. ***str або об'єкт*** **пам'яті** ***з інтерфейсом joblib.Memory, за замовчуванням = Немає*** Використовується для кешування результатів обчислень дерева. За замовчуванням кешування не виконується. Якщо задано рядок, це шлях до каталогу кешування. **n_jobs** ***int, за замовчуванням = Немає*** Кількість паралельних завдань для пошуку сусідів. Noneозначає 1, якщо не в **joblib.parallel_backend**контексті. -1означає використання всіх процесорів.
Атрибути	**labels_** ***ndarray of shape (n_samples,)*** Кластерні мітки для кожної точки в наборі даних, надані fit (). Шумні зразки та точки, які не входять до групи листя cluster_hierarchy_, позначаються як -1. **reachability_набір** ***форми (n_зразків,)*** Відстані досяжності для зразка, індексовані за порядком об’єктів. Використовуйте clust.reachability_[clust.ordering_]для доступу в порядку кластера. **ordering_набір** ***форми (n_зразків,)*** Кластер упорядкований список вибіркових індексів. **core_distances_** ***ndarray of shape (n_samples,)*** Відстань, на якій кожен зразок стає базовою точкою, індексований порядком об’єктів. Точки, які ніколи не будуть основними, мають відстань інф. Використовуйте clust.core_distances_[clust.ordering_]для доступу в порядку кластера. **predecessor_н** ***-масив форми (n_зразки,)*** Точка, з якої було отримано вибірку, індексована за порядком об’єктів. Посівні точки мають попередник -1. **cluster_hierarchy_** ***ndray of shape (n_clusters, 2)*** Список кластерів у вигляді у кожному рядку з усіма індексами включно. Кластери впорядковані відповідно (по зростанню) так, що

	більші кластери, що охоплюють менші кластери, йдуть за цими меншими. Оскільки не відображає ієрархію, зазвичай . Зверніть увагу також, що ці індекси належать до , тобто утворюють кластер. Доступно лише тоді, коли .[start, end](end, -start)labels_len(cluster_hierarchy_) > np.unique(optics.labels_)ordering_X[ordering_][start:end + 1]cluster_method='xi' **n_features_in_** ***int*** Кількість особливостей, які можна побачити під час підгонки . **feature_names_in_** ***ndarray форми (n_features_in_,)*** Назви ознак, які можна побачити під час підгонки . Визначається лише тоді, коли X всі назви функцій є рядками.

Приклад коду:

```
>>> from sklearn.cluster import OPTICS
>>> import numpy as np
>>> X = np.array([[1, 2], [2, 5], [3, 6],
...               [8, 7], [8, 8], [7, 3]])
>>> clustering = OPTICS(min_samples=2).fit(X)
>>> clustering.labels_
array([0, 0, 0, 1, 1, 1])
```

2.8. BIRCH

Це алгоритм ефективного запам'ятовування та навчання в режимі онлайн, наданий як альтернатива **MiniBatchKMeans**. Він будує деревоподібну структуру даних із зчитуванням центроїдів кластера з листа. Це можуть бути або кінцеві центроїди кластера, або можуть бути надані як вхідні дані для іншого алгоритму кластеризації, наприклад **AgglomerativeClustering**.

BIRCH створює дерево, яке називається Деревом кластеризації функцій (CFT) для даних. Дані по суті стискаються з втратами до набору вузлів функцій кластеризації (вузлів CF). У вузлах CF є ряд субкластерів, які називаються підкластерами функцій кластеризації (субкластери CF), і ці субкластери CF, розташовані в нетермінальних вузлах CF, можуть мати дочірні вузли CF.

Субкластери CF містять необхідну інформацію для кластеризації, що запобігає необхідності зберігати всі вхідні дані в пам'яті. Ця інформація включає:

- Кількість зразків у підгрупі.
- Лінійна сума - n -вимірний вектор, що містить суму всіх вибірок
- Квадратична сума - сума квадратичної норми L2 усіх зразків.
- Центроїди - щоб уникнути перерахунку лінійних сум / n_вибірок.
- Квадратна норма центроїдів.

Алгоритм BIRCH має два параметри: поріг та коефіцієнт розгалуження. Коефіцієнт розгалуження обмежує кількість субкластерів у вузлі, а поріг обмежує відстань між вхідною вибіркою та існуючими субкластерами.

Цей алгоритм можна розглядати як екземпляр або метод зменшення даних, оскільки він зменшує вхідні дані до набору субкластерів, які отримуються безпосередньо з листків CFT. Ці зменшені дані можуть бути додатково оброблені шляхом надсилання їх до глобального кластера. Цей глобальний кластеризатор може бути встановлений n_clusters. Якщо для n_кластерів встановлено значення None, субкластери з листків зчитуються безпосередньо, інакше на глобальному етапі кластеризації ці підкластери позначаються у глобальні кластери (мітки), а зразки відображаються на глобальну мітку найближчого підкластеру.

Опис алгоритму:

- Новий зразок вставляється в корінь дерева CF, який є вузлом CF. Потім він зливається з підгрупою кореня, який має найменший радіус після злиття, обмежений умовами порога та коефіцієнта розгалуження. Якщо у підкластері є якийсь дочірній вузол, це робиться неодноразово, поки він не досягне листа. Після знаходження найближчого підкластера в аркуші властивості цього підкластера та батьківського підкластера рекурсивно оновлюються.
- Якщо радіус підгрупи, отриманий шляхом об’єднання нової вибірки та найближчого підгрупи, більший за квадрат порога, і якщо кількість підкластерів більша за коефіцієнт розгалуження, то цій новій вибірці тимчасово виділяється простір. Беруться два найдальших підгрупи, а

підгрупи поділяються на дві групи на основі відстані між цими підгрупами.

- Якщо цей розділений вузол має батьківський підкластер і є місце для нового підкластера, то батьківський розділяється на два. Якщо місця немає, то цей вузол знову розбивається на дві частини, і процес продовжується рекурсивно, поки він не досягне кореня.

Далі розглянемо приклад реалізації та опис параметрів і аргументів:

***class* sklearn.cluster.Birch(*, *threshold=0.5*, *branching_factor=50*, *n_clusters=3*, *compute_labels=True*, *copy=True*)**

Таблиця 10

Параметри	**Threshold:** ***float, за замовчуванням = 0,5*** Радіус підгрупи, отриманий шляхом злиття нового зразка та найближчого підгрупи, повинен бути меншим за поріг. В іншому випадку запускається новий підкластер. Встановлення цього значення як дуже низького сприяє розщепленню і навпаки. **Branching_factor:** ***int, за замовчуванням = 50*** Максимальна кількість субкластерів CF у кожному вузлі. Якщо нові зразки вводяться так, що кількість субкластерів перевищує Branching_factor, тоді цей вузол розбивається на два вузли з перерозподілом субкластерів у кожному. Батьківський підкластер цього вузла видаляється, а два нові субкластери додаються як батьки 2 розділених вузлів. **n_clusters :*****int, екземпляр моделі sklearn.cluster, за замовчуванням = 3*** Кількість кластерів після останнього етапу кластеризації, який розглядає субкластери з листків як нові зразки. • None : останній крок кластеризації не виконується, і підкластери повертаються такими, якими вони є. • **sklearn.cluster** Оцінювач: Якщо надана модель, модель підходить для розгляду субкластерів як нових зразків, а вихідні дані відображаються на етикетці найближчого підкластеру. • int: Модель підгонка **AgglomerativeClustering**з n_clusters набором бути рівним межд. **compute_labels:** ***bool, за замовчуванням = True*** Обчислювати мітки для кожної відповідності чи ні. **copy :*****bool, default = True*** Копіювати ці дані чи ні. Якщо встановлено значення False, початкові дані будуть перезаписані.

Атрибути	**root_: _*CFNode*** Корінь CFTree. **dummy_leaf_: _*CFNode*** Почніть вказівник на всі листочки. **subcluster_centers_: *ndarray*** Центроїди всіх підгруп читаються безпосередньо з листя. **subcluster_labels_ :*ndarray*** Мітки, призначені центроїдам субкластерів після їх кластеризації у всьому світі. **labels_ :*ndarray of shape (n_samples,)*** Масив міток, призначених для вхідних даних. якщо замість fit використовується частковий_фіт, вони призначаються останній партії даних. **n_features_in_:*int*** Кількість особливостей, які можна побачити під час підгонки . **feature_names_in_ :*ndarray форми (n_features_in_,)*** Назви ознак, які можна побачити під час підгонки . Визначається лише тоді, коли X всі назви функцій є рядками.

Приклад коду:

```
>>> from sklearn.cluster import Birch
>>> X = [[0, 1], [0.3, 1], [-0.3, 1], [0, -1], [0.3, -1], [-0.3, -1]]
>>> brc = Birch(n_clusters=None)
>>> brc.fit(X)
Birch(n_clusters=None)
>>> brc.predict(X)
array([0, 0, 0, 1, 1, 1])
```

2.9. Порівняння методів бібліотеки scikit-learn

Приклади порівняння алгоритмів подано на рис. 2.4

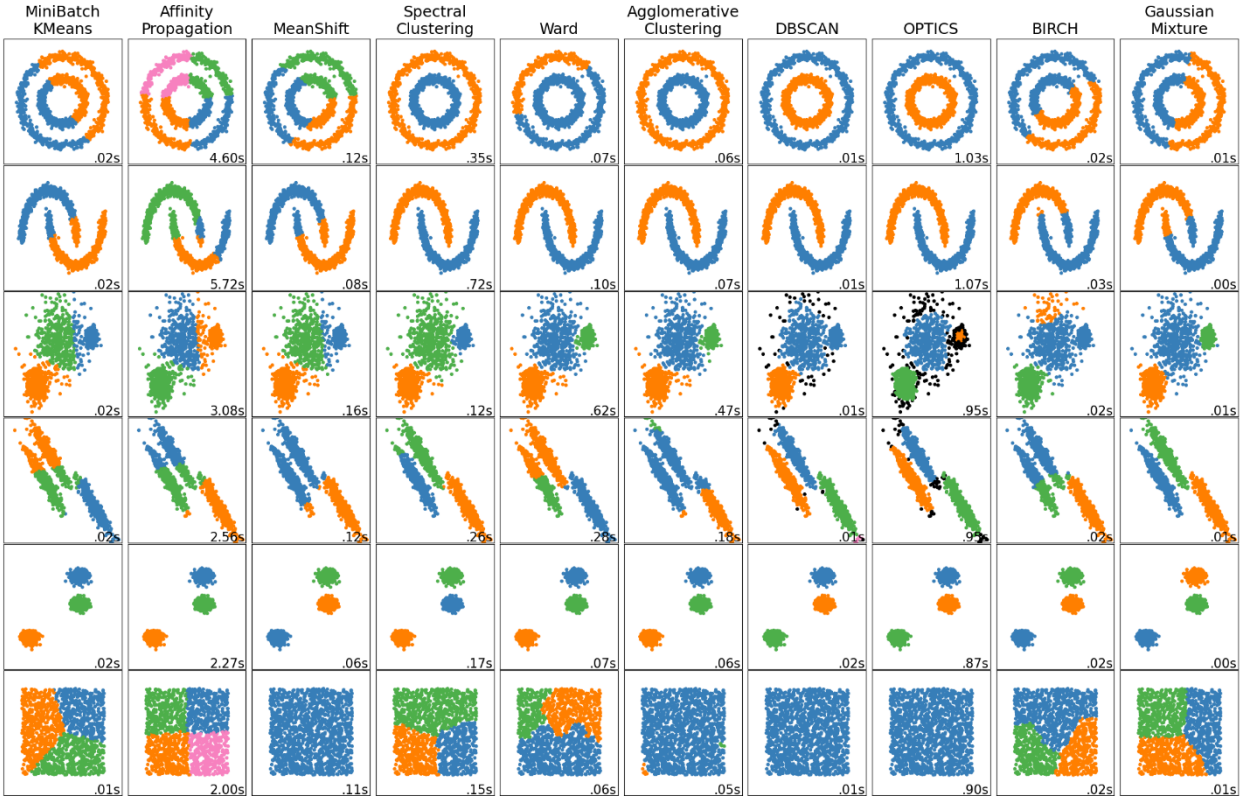


Рис. 2.4 – Методи кластеризації бібліотеки scikit learn

Також згідно опису алгоритмів можемо побачити порівняльну таблицю наведену нижче.

Таблиця 11

Назва методу	Параметри	Масштабованість	Спосіб використання	Геометрія
K-means	число кластерів	Дуже великий n_samples, середній n_clusters з кодом MiniBatch	Загальне призначення, рівний розмір кластерів, плоска геометрія, не надто багато кластерів, індуктивні	Відстані між точками
Mean-shift	пропускна здатність	Не масштабується з n_samples	Багато кластерів, нерівномірний розмір кластерів, нерівна геометрія, індуктивні	Відстані між точками
Spectral clustering	кількість кластерів	Середній n_samples, маленький n_clusters	Кілька кластерів, навіть розміри кластерів, нерівна геометрія	Відстань графіка (наприклад, графік найближчого сусіда)
Hierarchical clustering	кількість кластерів або поріг відстані	Великі n_samplesin_clusters	Багато кластерів, можливо обмеження з'єднання	Відстані між точками
Agglomerative clustering	кількість кластерів або поріг відстані, тип зв'язку, відстань	Великі n_samplesin_clusters	Багато кластерів, можливо обмеження зв'язків, неевклідові відстані	Будь -яка попарна відстань
OPTICS	мінімальне членство в кластері	Дуже великий n_samples, великий n_clusters	Не плоска геометрія, нерівномірні розміри кластерів, змінна щільність кластерів	Відстані між точками
BIRCH	коефіцієнт розгалуження, поріг, необов'язковий глобальний кластеризатор.	Великі n_clustersin_samples	Великий набір даних, видалення викидів, зменшення	Евклідова відстань між точками

РОЗДІЛ 3
РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОГО ЕКСПЕРИМЕНТУ: КЛАСТЕРИЗАЦІЯ ІТАЛІЙСЬКИХ ВИН

3.1. Опис набору даних

Постановка задачі

Задача полягає в тому, щоб на основі хімічного аналізу вина передбачити його сорт.

Опис даних

Дані були отримані з датасету представленому на каглі (https://www.kaggle.com/harrywang/wine-dataset-for-clustering) .Вони являють собою набір векторів, розмірністю 13. Всього 178 прикладів. Три класи вин. Опис ознак датасету:

- Alcohol(алкоголь)
- Malic_Acid(малеїнова кислота)
- Ash(зола)
- Ash_Alcanity (лужність золи)
- Magnesium (магній)
- Total_Phenols (загальний вміст фенолів)
- Flavanoids (флаваноіди)
- Nonflavanoid_Phenols (нефлаваноідние феноли)
- Proanthocyanins (проантоціаніни)
- Color_Intensity (інтенсивність кольору)
- Hue (відтінок)
- OD280 (розбавлених вин)
- Proline(пролин)

Підготовка даних

Імпортуємо необхідні бібліотеки та датасет

```
]: 1 import numpy as np
   2 import pandas as pd
   3 from sklearn.cluster import AffinityPropagation
   4 from sklearn import metrics
```

```
]: 1 df = pd.read_csv('wine.csv')
```

```
]: 1 df.head()
```

]:

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

```
]: 1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Wine                  178 non-null    int64
 1   Alcohol               178 non-null    float64
 2   Malic.acid            178 non-null    float64
 3   Ash                   178 non-null    float64
 4   Acl                   178 non-null    float64
 5   Mg                    178 non-null    int64
 6   Phenols               178 non-null    float64
 7   Flavanoids            178 non-null    float64
 8   Nonflavanoid.phenols  178 non-null    float64
 9   Proanth               178 non-null    float64
 10  Color.int             178 non-null    float64
 11  Hue                   178 non-null    float64
 12  OD                    178 non-null    float64
 13  Proline               178 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

Рис. 3.1

Форматуємо наш датасет

```
: 1 labels_true=df["Wine"].values
  2 labels_true=labels_true-1
  3 data=df.drop("Wine", axis=1)
  4 labels_true
```

```
: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2], dtype=int64)
```

```
: 1 data.head()
```

:

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

Рис. 3.2

Нормалізуємо наш датасет за допомогою функції StandardScaler

```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()
data_cluster=data.copy()
data_cluster[data_cluster.columns]=std_scaler.fit_transform(data_cluster)
data_cluster.describe()
```

	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Colo
count	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e
mean	-8.619821e-16	-8.357859e-17	-8.657245e-16	-1.160121e-16	-1.995907e-17	-2.972030e-16	-4.016762e-16	4.079134e-16	-1.699639e-16	-1.247442
std	1.002821e+00	1.002821e+00	1.002821e+00	1.002821e+00	1.002821e+00	1.002821e+00	1.002821e+00	1.002821e+00	1.002821e+00	1.002821e
min	-2.434235e+00	-1.432983e+00	-3.679162e+00	-2.671018e+00	-2.088255e+00	-2.107246e+00	-1.695971e+00	-1.868234e+00	-2.069034e+00	-1.634288e
25%	-7.882448e-01	-6.587486e-01	-5.721225e-01	-6.891372e-01	-8.244151e-01	-8.854682e-01	-8.275393e-01	-7.401412e-01	-5.972835e-01	-7.951025
50%	6.099988e-02	-4.231120e-01	-2.382132e-02	1.518295e-03	-1.222817e-01	9.595986e-02	1.061497e-01	-1.760948e-01	-6.289785e-02	-1.592246
75%	8.361286e-01	6.697929e-01	6.981085e-01	6.020883e-01	5.096384e-01	8.089974e-01	8.490851e-01	6.095413e-01	6.291754e-01	4.939560
max	2.259772e+00	3.109192e+00	3.156325e+00	3.154511e+00	4.371372e+00	2.539515e+00	3.062832e+00	2.402403e+00	3.485073e+00	3.435432e

Рис. 3.3

3.2. Розв’язок задачі на основі методу k-means

Визначимо оптимальну кількість кластерів

Для визначення оптимальної кількості кластерів використали 2 методи: оцінка силуету та інерція к-середніх(з аналізом ліктя)

Спочатку обчислимо всі інерції (чим нижче інерція тим краще) див. рис. 3.4

```
from sklearn.cluster import KMeans
from tqdm import tqdm
import matplotlib.pyplot as plt
inertia = []
for i in tqdm(range(2,10)):
    kmeans = KMeans(n_clusters=i,
                    init='k-means++',
                    n_init=15,
                    max_iter=500,
                    random_state=17)
    kmeans.fit(data_cluster)
    inertia.append(kmeans.inertia_)

.00%|██████████| 8/8 [00:00<00:00, 20.18it/s]
```

Рис. 3.4 – обчислення інерції к-середніх

Далі обчислюємо оцінку силуету(чим вище бал тим краще) див. рис. 3.5

```
from sklearn.metrics import silhouette_score

silhouette = {}
for i in tqdm(range(2,10)):
    kmeans = KMeans(n_clusters=i,
                    init='k-means++',
                    n_init=15,
                    max_iter=500,
                    random_state=17)
    kmeans.fit(data_cluster)
    silhouette[i] = silhouette_score(data_cluster, kmeans.labels_, metric='euclidean')
```

```
00%|██████████| 8/8 [00:00<00:00, 19.54it/s]
```

Рис. 3.5 - Обчислення коеф силуету

Візуалізуємо наші обчислення див. рис. 3.6

```
sns.set(style='white',font_scale=1.1, rc={'figure.figsize':(12,5)})

plt.subplot(1, 2, 1)

plt.plot(range(2,len(inertia)+2), inertia, marker='o',lw=2,ms=8,color='red')
plt.xlabel('Number of clusters')
plt.title('K-means Inertia',fontweight='bold')
plt.grid(True)

plt.subplot(1, 2, 2)

plt.bar(range(len(silhouette)), list(silhouette.values()), align='center',color= 'red',width=0.5)
plt.xticks(range(len(silhouette)), list(silhouette.keys()))
plt.grid()
plt.title('Silhouette Score',fontweight='bold')
plt.xlabel('Number of Clusters')
plt.show()
```

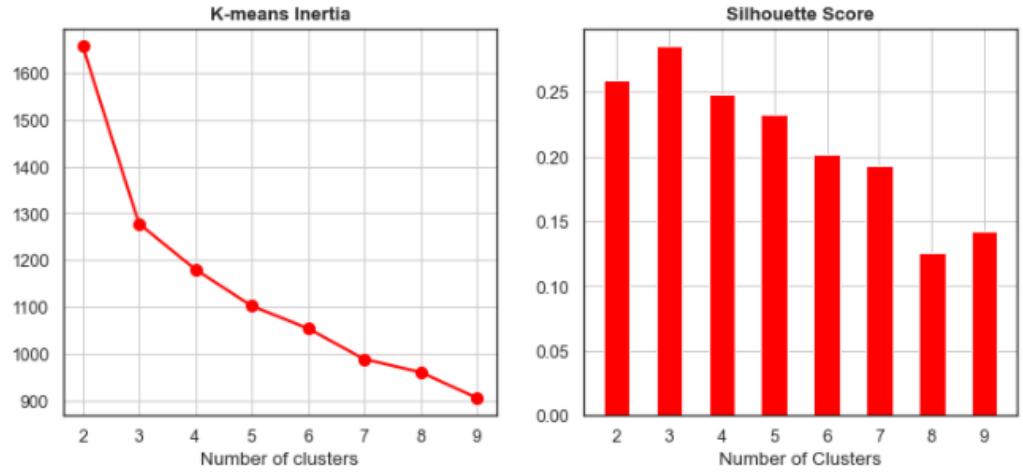


Рис. 3.6 – Графіки інерції та силуету

Як бачимо, оптимальна кількість кластерів 3

Для кращої візуалізації застосуємо метод головних компонент (PCA) див. рис. 3.7

```
from sklearn.decomposition import PCA
pca_2 = PCA(2)
pca_2_result = pca_2.fit_transform(data_cluster)

print ('Cumulative variance explained by 2 principal components: {:.2%}'.format(np.sum(pca_2.explained_variance_ratio_)))
```

```
Cumulative variance explained by 2 principal components: 55.41%
```

```
sns.set(style='white', rc={'figure.figsize':(9,6)},font_scale=1.1)

plt.scatter(x=pca_2_result[:, 0], y=pca_2_result[:, 1], color='red',lw=0.1)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Data represented by the 2 strongest principal components',fontweight='bold')
plt.show()
```

Рис. 3.7 – Реалізація методу PCA

Візуалізуємо розбиття наших кластерів див. рис. 3.8

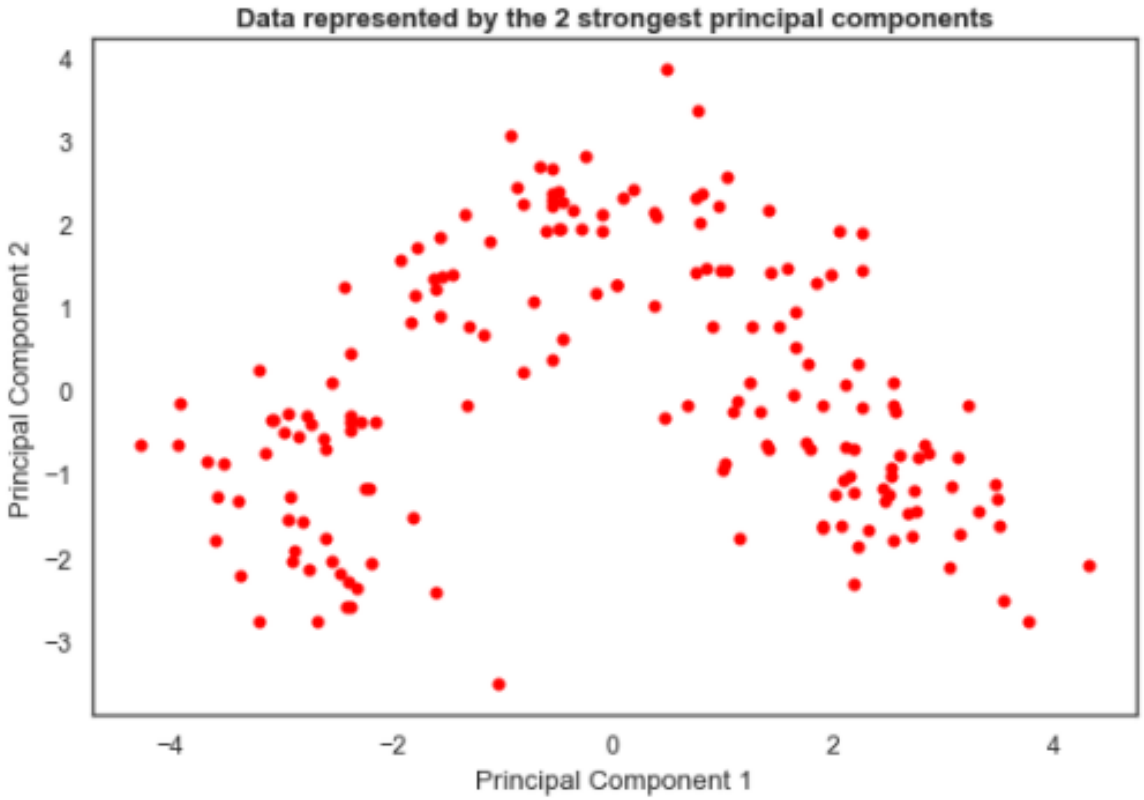


Рис. 3.8 – Візуалізація розбиття на кластери

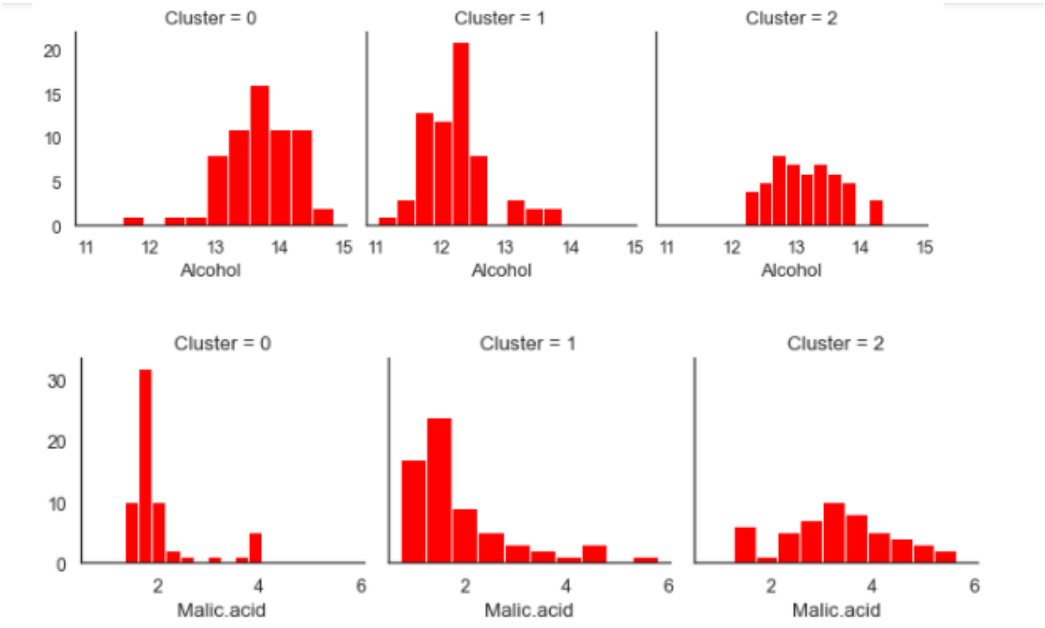
Знайдемо кількість об’єктів у кожному кластері див. рис. 3.9

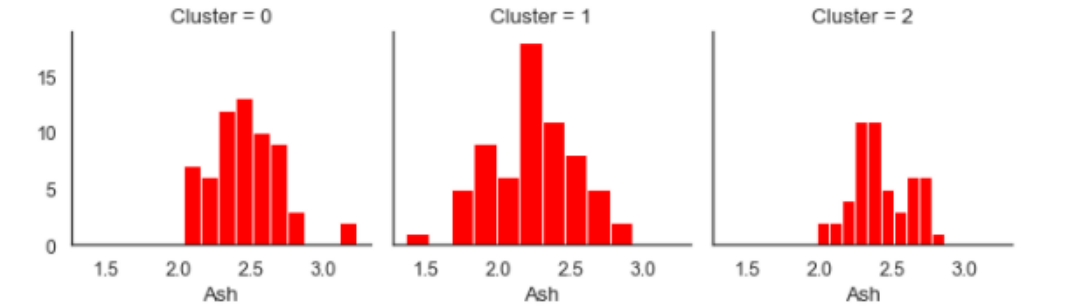
```
]: 1 centroids = kmeans.cluster_centers_
   2 centroids_pca = pca_2.transform(centroids)
   3
   4 pd.Series(labels_kmeans).value_counts()

]: 1    65
   0    62
   2    51
   dtype: int64
```

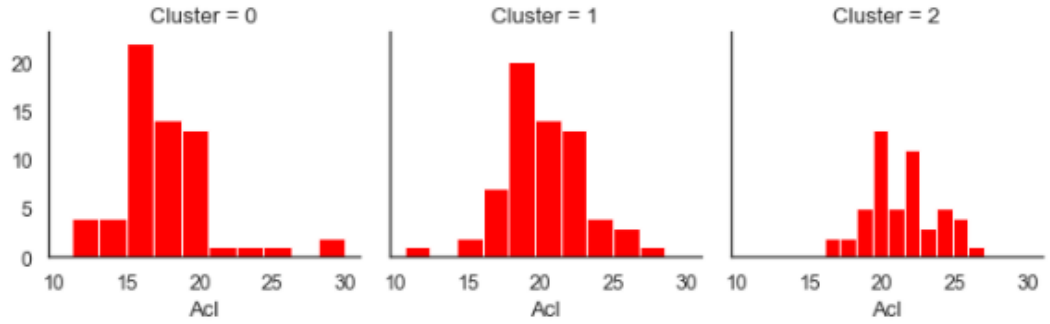
Рис. 3.9-кількість об’єктів у кожному кластері

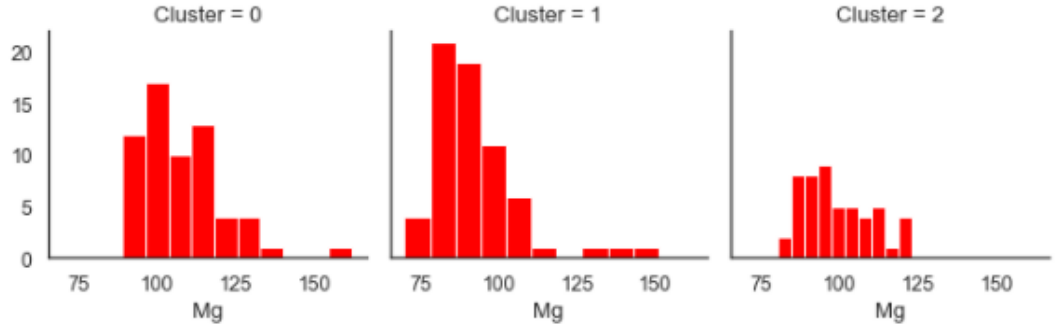
Далі візуалізуємо розподіл кожної функції відповідно до кожного кластера, на цьому кроці ми можемо визначити деякі характеристики для кожної групи див. рис. 3.10

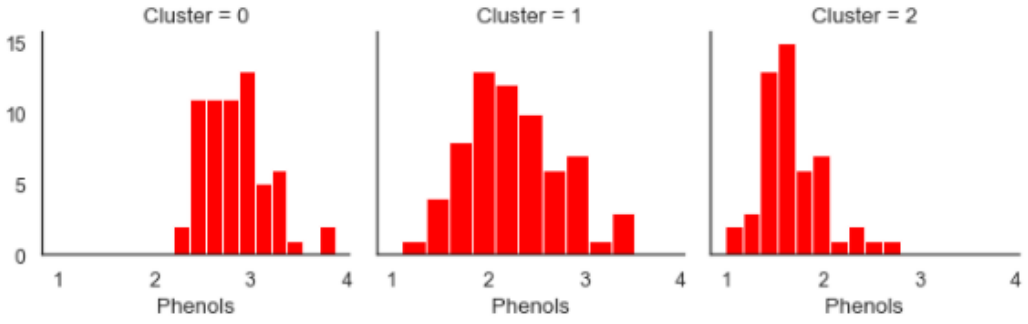
```
data2=data.copy()
data2['Cluster']=labels_kmeans

aux=data2.columns.tolist()
aux[0:len(aux)-1]

for cluster in aux[0:len(aux)-1]:
    grid= sns.FacetGrid(data2, col='Cluster')
    grid.map(plt.hist, cluster,color='red')
```

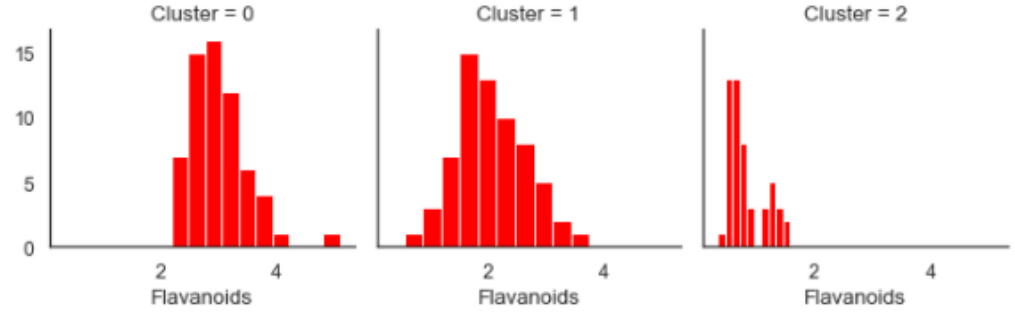


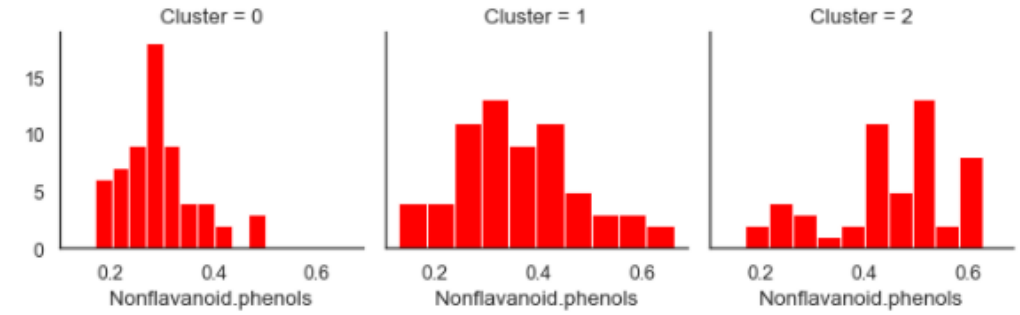

Cluster = 0
Cluster = 1
Cluster = 2
15
10
5
0
1.5
2.0
2.5
3.0
Ash

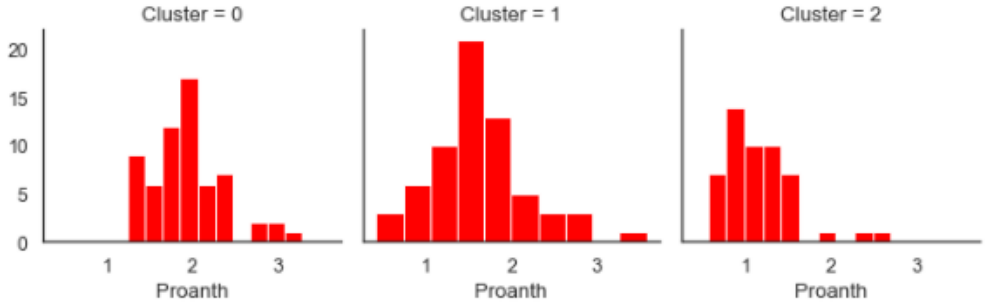

Cluster = 0
Cluster = 1
Cluster = 2
20
15
10
5
0
10
15
20
25
30
Acl

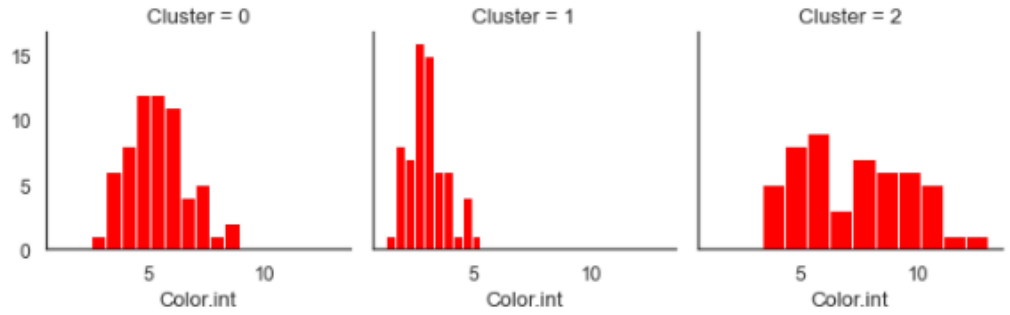

Cluster = 0
Cluster = 1
Cluster = 2
20
15
10
5
0
75
100
125
150
Mg

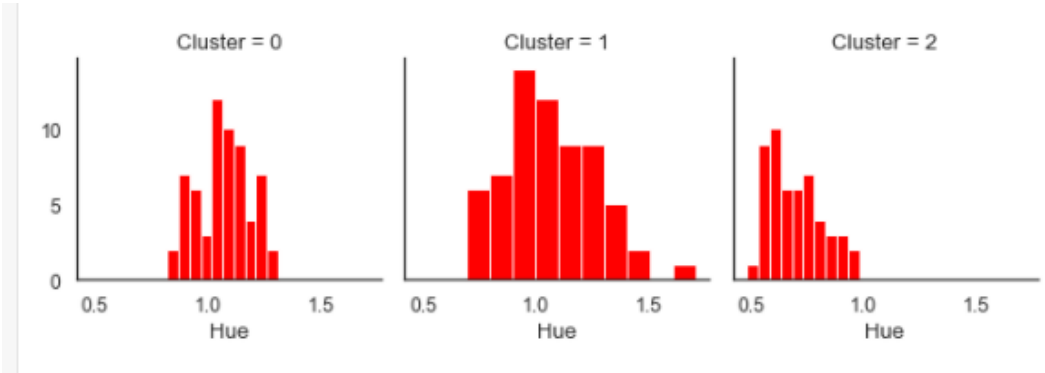

Cluster = 0
Cluster = 1
Cluster = 2
15
10
5
0
1
2
3
4
Phenols

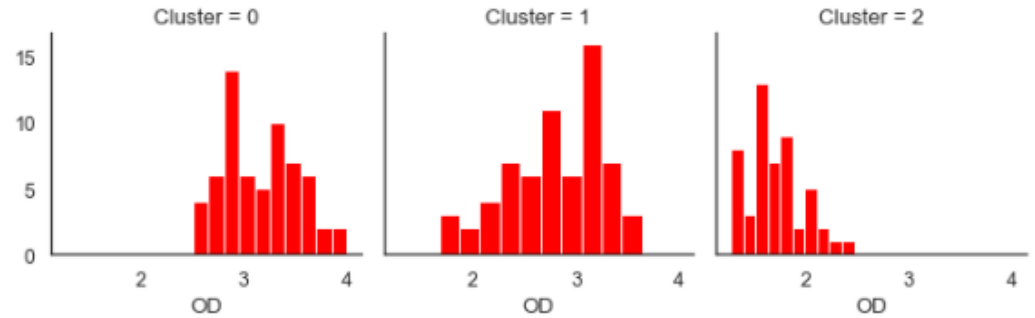

Cluster = 0
Cluster = 1
Cluster = 2
15
10
5
0
2
4
Flavanoids


Cluster = 0
Cluster = 1
Cluster = 2
15
10
5
0
0.2
0.4
0.6
Nonflavanoid.phenols


Cluster = 0
Cluster = 1
Cluster = 2
20
15
10
5
0
1
2
3
Proanth


Cluster = 0
Cluster = 1
Cluster = 2
15
10
5
0
5
10
Color.int


Cluster = 0
Cluster = 1
Cluster = 2
10
5
0
0.5
1.0
1.5
Hue


Cluster = 0
Cluster = 1
Cluster = 2
15
10
5
0
2
3
4
OD

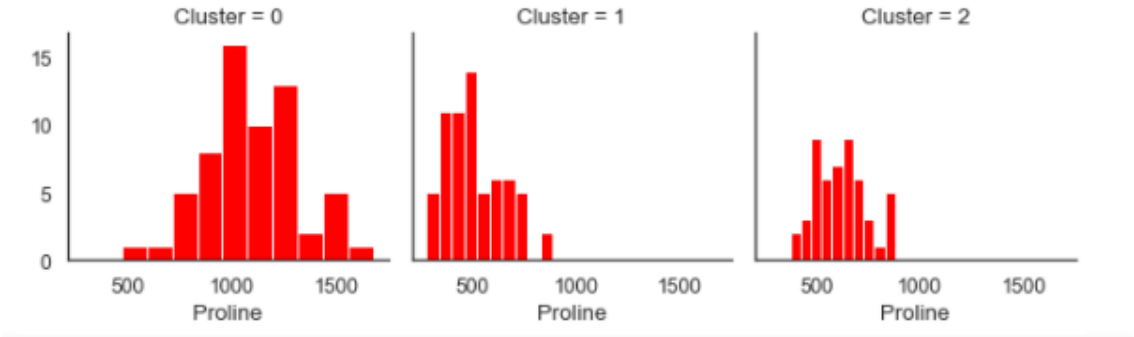


Рис. 3.10 – Розподіл функцій відповідно кластерам

Для більш чіткої візуалізації застосуємо різні кольори для різних груп кластерів Див. рис. 3.11

```
sns.set(style='white', rc={'figure.figsize':(9,6)},font_scale=1.1)

plt.scatter(x=pca_2_result[:, 0], y=pca_2_result[:, 1], c=labels_kmeans, cmap='autumn')
plt.scatter(centroids_pca[:, 0], centroids_pca[:, 1],
            marker='x', s=169, linewidths=3,
            color='black', zorder=10,lw=3)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Clustered Data (PCA visualization)',fontweight='bold')
plt.show()
```

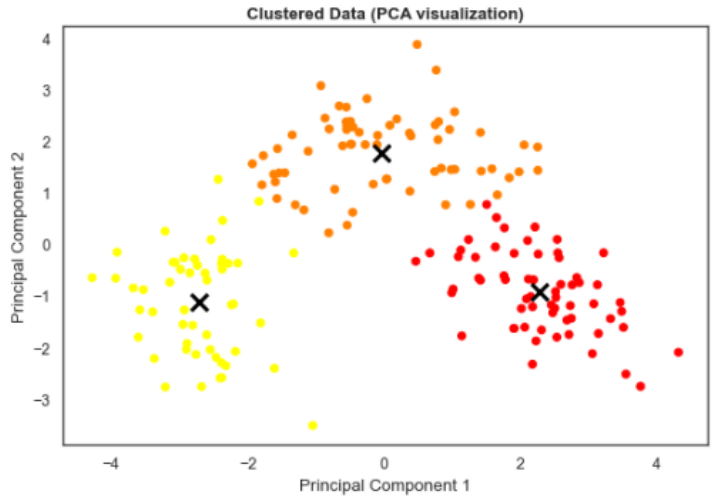


Рис. 3.11 – Кольорова візуалізація розбиття на кластери

Далі проведемо оцінку якості алгоритму див. рис. 3.12

```
from sklearn.cluster import KMeans
n_clusters=3
kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(data_cluster)
labels_kmeans = kmeans.labels_
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels_kmeans))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels_kmeans))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels_kmeans))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(labels_true, labels_kmeans))
print(
    "Adjusted Mutual Information: %0.3f"
    % metrics.adjusted_mutual_info_score(labels_true, labels_kmeans)
)
print(
    "Silhouette Coefficient: %0.3f"
    % metrics.silhouette_score(data_cluster, labels_kmeans, metric="euclidean")
)

max_acc, mapping_max_acc,mapped_predict_max, n_clusters=max_accuracy(labels_true, labels_kmeans)
print(n_clusters, '\n', max_acc, mapping_max_acc,'\n',mapped_predict_max)
print('\n',labels_true)
```

```
Homogeneity: 0.879
Completeness: 0.873
V-measure: 0.876
Adjusted Rand Index: 0.897
Adjusted Mutual Information: 0.875
Silhouette Coefficient: 0.285
3
 0.9662921348314607 {0: 0, 1: 1, 2: 2}
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 2 1 1 0 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Рис. 3.12 – Оцінка якості методу k-means

Як бачимо досить непогані показники

Візуалізуємо на графіках оцінку якості при іншій кількості кластерів

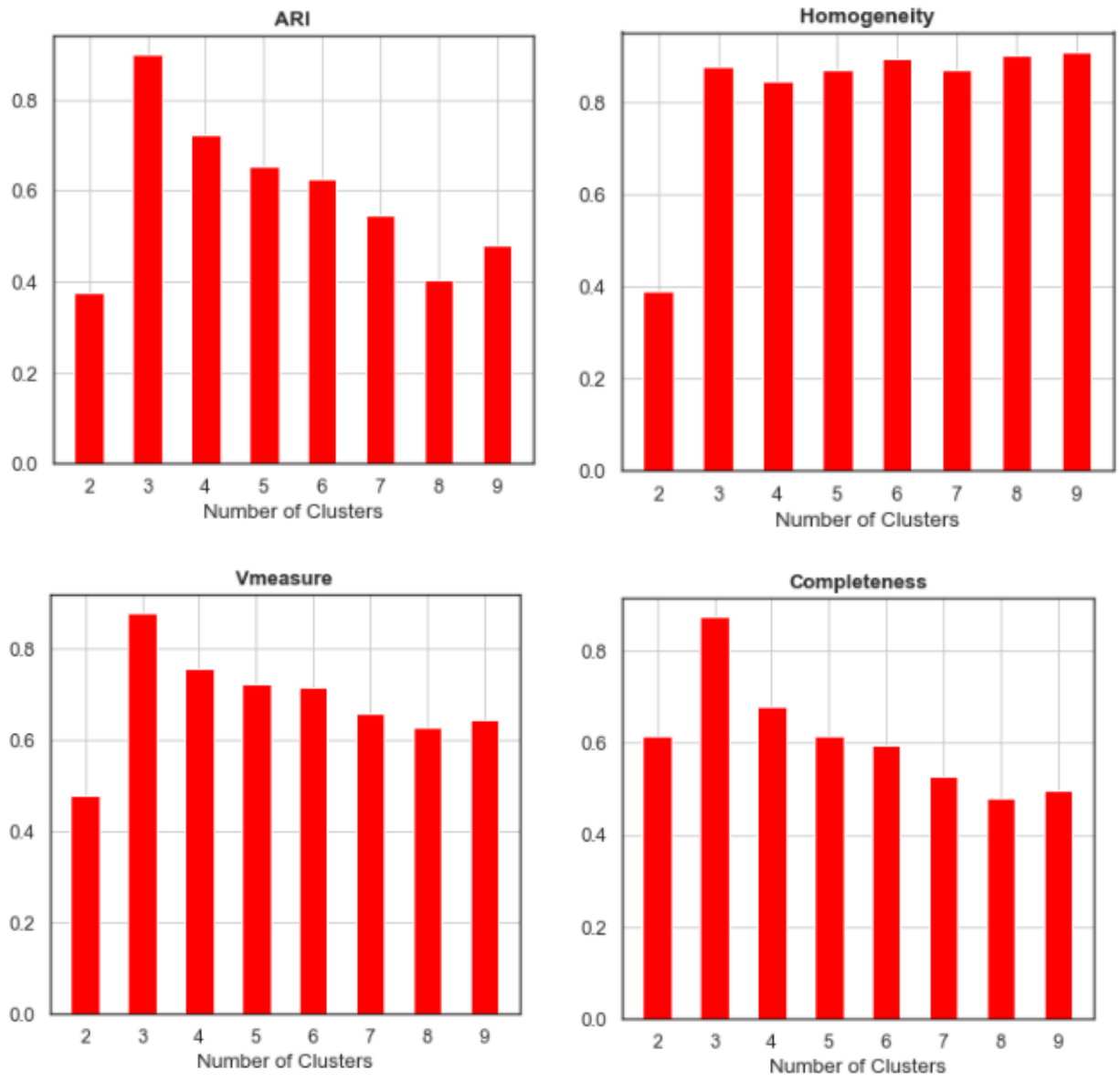


Рис. 3.13 – Порівняння параметрів оцінки якості на різній кількості кластерів

3.3. Розв'язок задачі на основі методу Meanshift

```
:
from sklearn.cluster import MeanShift
ms = MeanShift(bin_seeding=True)
ms.fit(data_cluster)
labels_ms = ms.labels_
cluster_centers = ms.cluster_centers_
labels_unique = np.unique(labels_ms)
n_clusters_ms = len(labels_unique)

print("Estimated number of clusters: %d" % n_clusters_ms)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels_ms))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels_ms))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels_ms))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(labels_true, labels_ms))
print(
    "Adjusted Mutual Information: %0.3f"
    % metrics.adjusted_mutual_info_score(labels_true, labels_ms)
)
print("metrics accuracy score: %0.3f" % metrics.accuracy_score(labels_true, labels_ms))
print(
    "Silhouette Coefficient: %0.3f"
    % metrics.silhouette_score(data_cluster, labels_ms, metric="sqeuclidean")
)
```

```
Estimated number of clusters: 2
Homogeneity: 0.019
Completeness: 0.196
V-measure: 0.035
Adjusted Rand Index: -0.006
Adjusted Mutual Information: 0.024
metrics accuracy score: 0.354
Silhouette Coefficient: 0.351
```

```
:
max_acc, mapping_max_acc,mapped_predict_max, n_clusters=max_accuracy(labels_true, labels_ms)
print(n_clusters, '\n', max_acc, mapping_max_acc,'\n',mapped_predict_max)
print('\n',labels_true)
```

```
2
 0.37640449438202245 {0: 1, 1: 0}
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Рис. 3.14 – Оцінка якості методу meanshift

Бачимо також погані результати оцінки та кількість кластерів 2

3.4. Розв’язок задачі на основі методу Spectral Clustering

```
from sklearn.cluster import SpectralClustering
sc=SpectralClustering(eigen_solver="arpack",
        affinity="nearest_neighbors",n_clusters=3,random_state=0).fit(data_cluster)
labels_sc = sc.labels_

print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels_sc))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels_sc))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels_sc))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(labels_true, labels_sc))
print(
    "Adjusted Mutual Information: %0.3f"
    % metrics.adjusted_mutual_info_score(labels_true, labels_sc)
)
print("metrics accuracy score: %0.3f" % metrics.accuracy_score(labels_true, labels_sc))
print(
    "Silhouette Coefficient: %0.3f"
    % metrics.silhouette_score(data_cluster, labels_sc, metric="sqeuclidean")
)
```

```
Homogeneity: 0.864
Completeness: 0.858
V-measure: 0.861
Adjusted Rand Index: 0.880
Adjusted Mutual Information: 0.859
metrics accuracy score: 0.961
Silhouette Coefficient: 0.448
```

```
max_acc, mapping_max_acc,mapped_predict_max, n_clusters=max_accuracy(labels_true, sc.labels_)
print(n_clusters, '\n', max_acc, mapping_max_acc,'\n',mapped_predict_max)
print('\n',labels_true)
```

```
3
 0.9606741573033708 {0: 0, 1: 1, 2: 2}
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0
 1 1 1 1 0 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 0 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Рис. 3.15 – Оцінка якості методу Spectral Clustering

Бачимо дуже гарні результати оцінки методу

3.5. Розв’язок задачі на основі методу Agglomerative Clustering

```
from sklearn.cluster import AgglomerativeClustering
ag=AgglomerativeClustering(linkage="ward", n_clusters=3).fit(data_cluster)
labels_ag = ag.labels_

print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels_ag))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels_ag))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels_ag))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(labels_true, labels_ag))
print(
    "Adjusted Mutual Information: %0.3f"
    % metrics.adjusted_mutual_info_score(labels_true, labels_ag)
)
print("metrics accuracy score: %0.3f" % metrics.accuracy_score(labels_true, labels_ag))
print(
    "Silhouette Coefficient: %0.3f"
    % metrics.silhouette_score(data_cluster, labels_ag, metric="sqeuclidean")
)
```

```
Homogeneity: 0.790
Completeness: 0.783
V-measure: 0.786
Adjusted Rand Index: 0.790
Adjusted Mutual Information: 0.784
metrics accuracy score: 0.045
Silhouette Coefficient: 0.438
```

```
max_acc, mapping_max_acc,mapped_predict_max, n_clusters=max_accuracy(labels_true, ag.labels_)
print(n_clusters, '\n', max_acc, mapping_max_acc,'\n',mapped_predict_max)
print('\n',labels_true)
```

```
3
 0.9269662921348315 {0: 1, 1: 2, 2: 0}
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2 1 1 1 0 0 1 2 1 2 0 1 0
 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 2 1 1 0 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Рис. 3.16 – Оцінка якості методу Agglomerative Clustering

Цей метод показав також досить непогані результати оцінки

3.6. Розв’язок задачі на основі методу DBSCAN

```python
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=40, min_samples=5).fit(data)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels_db = db.labels_
print(labels_db)
n_clusters_db = len(set(labels_db)) - (1 if -1 in labels_db else 0)
n_noise_db = list(labels_db).count(-1)


print("Estimated number of clusters: %d" % n_clusters_db)
print("Estimated number of noise points: %d" % n_noise_db)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels_db))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels_db))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels_db))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(labels_true, labels_db))
print(
    "Adjusted Mutual Information: %0.3f"
    % metrics.adjusted_mutual_info_score(labels_true, labels_db)
)
print("metrics accuracy score: %0.3f" % metrics.accuracy_score(labels_true, labels_db))
print(
    "Silhouette Coefficient: %0.3f"
    % metrics.silhouette_score(data, labels_db, metric="sqeuclidean")
)
```

```
[ 0  0  0 -1  1 -1  2  2  0  0 -1  2  2  0 -1  2  2  0 -1  1  1  1  0  0
  1  1  0  2  1  0  2 -1  0  2  0  1  1  0  0  1  1  0  0  1  1  0  0  0
  0  2  0  2  0 -1  0  0  0  2  2  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1]
Estimated number of clusters: 3
Estimated number of noise points: 8
Homogeneity: 0.334
Completeness: 0.429
V-measure: 0.375
Adjusted Rand Index: 0.241
Adjusted Mutual Information: 0.363
metrics accuracy score: 0.534
Silhouette Coefficient: 0.653
```

```python
max_acc, mapping_max_acc,mapped_predict_max, n_clusters=max_accuracy(labels_true, db.labels_)
print(n_clusters, '\n', max_acc, mapping_max_acc,'\n',mapped_predict_max)
print('\n',labels_true)
```

```
4
 0.5337078651685393 {-1: 2, 0: 0, 1: 1, 2: 3}
 [0 0 0 2 1 2 3 3 0 0 2 3 3 0 2 3 3 0 2 1 1 1 0 0 1 1 0 3 1 0 3 2 0 3 0 1 1
 0 0 1 1 0 0 1 1 0 0 0 0 3 0 3 0 2 0 0 0 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

Рис. 3.17 – Оцінка якості методу DBSCAN

Цей метод правильно визначив кількість кластерів але через специфіку своєї роботи загальні результати оцінки виявились не дуже гарні

3.7. Розв’язок задачі на основі методу Birch

Birch

```
4]:
from sklearn.cluster import Birch
brc = Birch(n_clusters=3)
brc.fit(data_cluster)
labels=brc.predict(data_cluster)

print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(labels_true, labels))
print(
    "Adjusted Mutual Information: %0.3f"
    % metrics.adjusted_mutual_info_score(labels_true, labels)
)
print("metrics accuracy score: %0.3f" % metrics.accuracy_score(labels_true, labels))
print(
    "Silhouette Coefficient: %0.3f"
    % metrics.silhouette_score(data_cluster, labels, metric="sqeuclidean")
)

max_acc, mapping_max_acc,mapped_predict_max, n_clusters=max_accuracy(labels_true, labels)
print(n_clusters, '\n', max_acc, mapping_max_acc,'\n',mapped_predict_max)
print('\n',labels_true)
```

```
Homogeneity: 0.790
Completeness: 0.783
V-measure: 0.786
Adjusted Rand Index: 0.790
Adjusted Mutual Information: 0.784
metrics accuracy score: 0.045
Silhouette Coefficient: 0.438
3
 0.9269662921348315 {0: 1, 1: 2, 2: 0}
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2 1 1 1 0 0 1 2 1 2 0 1 0
 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 2 1 1 0 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Рис. 3.18 – Оцінка якості методу BIRCH

Досить непогані результати оцінки

3.8. Розв’язок задачі на основі методу Gaussian Mixture

```python
from sklearn.mixture import GaussianMixture
gm = GaussianMixture(n_components=3, random_state=0)
labels_=gm.fit_predict(data_cluster)

print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels_))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels_))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels_))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(labels_true, labels_))
print(
    "Adjusted Mutual Information: %0.3f"
    % metrics.adjusted_mutual_info_score(labels_true, labels_)
)
print("metrics accuracy score: %0.3f" % metrics.accuracy_score(labels_true, labels_))
print(
    "Silhouette Coefficient: %0.3f"
    % metrics.silhouette_score(data_cluster, labels_, metric="sqeuclidean")
)

max_acc, mapping_max_acc,mapped_predict_max, n_clusters=max_accuracy(labels_true, labels_)
print(n_clusters, '\n', max_acc, mapping_max_acc,'\n',mapped_predict_max)
print('\n',labels_true)
```

```
Homogeneity: 0.864
Completeness: 0.858
V-measure: 0.861
Adjusted Rand Index: 0.880
Adjusted Mutual Information: 0.859
metrics accuracy score: 0.348
Silhouette Coefficient: 0.451
3
 0.9606741573033708 {0: 0, 1: 2, 2: 1}
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 0 1 1 1 0
 1 1 1 1 0 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Рис. 3.19 – Оцінка якості методу Gaussian Mixture

Метод показує гарні результати оцінки

3.9. Розв’язок задачі на основі методу OPTICS

```
from sklearn.cluster import OPTICS, cluster_optics_dbscan
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt

clust = OPTICS(min_samples=20, xi=0.05, min_cluster_size=0.05)
clust.fit(data_cluster)
labels_opt= clust.labels_

print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels_opt))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels_opt))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels_opt))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(labels_true, labels_opt))
print(
    "Adjusted Mutual Information: %0.3f"
    % metrics.adjusted_mutual_info_score(labels_true, labels_opt)
)
print("metrics accuracy score: %0.3f" % metrics.accuracy_score(labels_true, labels_opt))
print(
    "Silhouette Coefficient: %0.3f"
    % metrics.silhouette_score(data_cluster, labels_opt, metric="sqeuclidean")
)

max_acc, mapping_max_acc,mapped_predict_max, n_clusters=max_accuracy(labels_true, labels_opt)
print(n_clusters, '\n', max_acc, mapping_max_acc,'\n',mapped_predict_max)
print('\n',labels_true)
```

```
Homogeneity: 0.173
Completeness: 0.503
V-measure: 0.258
Adjusted Rand Index: 0.126
Adjusted Mutual Information: 0.252
metrics accuracy score: 0.000
Silhouette Coefficient: 0.219
2
 0.398876404494382 {-1: 1, 0: 0}
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 0 1 1 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Рис. 3.20 – Оцінка якості методу OPTICS

Цей метод показує погані результати оцінки

3.10.Порівняння результатів, що отримані різними методами

Наведемо таблицю для порівняння оцінки точності

Таблиця 12

Назва методу	ACU	H	C	VM	ARI	AMI	SI
K-means	0.966	0.879	0.873	0.876	0.897	0.875	0.285
Gaussian Mixture	0.961	0.864	0.858	0.861	0.880	0.859	0.451
Spectral Clustering	0.961	0.864	0.858	0.861	0.880	0.859	0.448
Agglomerative Clustering	0.927	0.790	0.783	0.786	0.790	0.784	0.438
BIRCH	0.927	0.790	0.783	0.786	0.790	0.784	0.438
OPTICS	0.399	0.173	0.503	0.258	0.126	0.252	0.219
Mean Shift	0.376	0.019	0.196	0.035	-0.006	0.024	0.351

У цій таблиці позначено: ACU – Accuracy, H – Homogeneity, C – Completeness, VM – V-measure, ARI – Adjusted Rand Index, AMI – Adjusted Mutual Information, SI – Silhouette Coefficient.

Як бачимо, найкращі показники оцінки якості має алгоритм k-means

Оцінка якості алгоритмів без нормалізації датасету за допомогою standard scaler наведена у таблиці 13

Таблиця 13

Назва методу	ACU	H	C	VM	ARI	AMI	SI
K-means	0.70	0.429	0.429	0.429	0.371	0.423	0.571
MeanShift	0.69	0.373	0.614	0.464	0.451	0.460	0.784
Spectral Clustering	0.71	0.419	0.420	0.420	0.359	0.414	0.704
Agglomerative Clustering	0.70	0.416	0.416	0.416	0.368	0.410	0.724
BIRCH	0.70	0.416	0.416	0.416	0.368	0.410	0.724
GaussianMixture	0.69	0.483	0.555	0.517	0.462	0.511	0.470
OPTICS	0.48	0.178	0.238	0.204	0.058	0.194	-0.255

1. ВИСНОВКИ

У роботі проведено порівняльний аналіз якості методів кластеризації на прикладі задачі про кластеризацію італійських вин за їх хімічним складом, за даними https://www.kaggle.com/harrywang/wine-dataset-for-clustering.

1. Використовуючи стандартні методи визначено кількість кластерів в досліджуваному наборі даних, що дорівнює трьом.
2. Використовуючи алгоритми бібліотеки scikit-learn, а саме k-means, Mean shift, Spectral Clustering, Agglomerative Clustering, DBSCAN, Birch, Gaussian Mixture, OPTICS проведено порівняння якості кластеризації за умови усталеного (by default) вибору параметрів алгоритмів. Отримано, що найвища досягнута точність (accuracy) не перевищує 71%, що відповідає низькій якості розпізнавання кластерів.
3. Для підвищення якості кластеризації було запропоновано провести попередню обробку даних, так щоб середні значення усіх характеристик досліджуваних об’єктів дорівнювала нулю, а дисперсія – одиниці. Така попередня обробка даних дозволила підвищити точність (accuracy) розпізнавання кластерів до 97%, на 26%!
4. Таке суттєве підвищення якості кластеризації пров’язано з наступним. Усі методи кластеризації використовують відстань між об’єктами, наприклад, скалярний добуток їх характеристик. У випадку, коли масштаб характеристик різний, вплив на таку відстань буде визначатись ознакою, яка має велике середнє значення за модулем та найбільшу дисперсію. Хоча вплив цієї ознаки на належність до відповідного кластеру може бути незначним. Тому нормалізація характеристик веде до рівності усіх характеристик у задачі належності об’єкта до відповідного модуля.
5. Це дозволяє визначити шлях покращення кластеризації наступним чином. Можна виходячи з моделі даних виділяти найважливіші ознаки і підсилювати їх. Наприклад, підсилення такої ознаки вин, як цукор

(дослідження фахівців стверджує, що це є одним з найголовніших характеристик вина) повинно дозволити підвищити якість кластеризації.

6. Таким чином, можна стверджувати, що попередня обробка даних перед використанням стандартних алгоритмів є принципово важливим етапом процесу кластеризації. В багатьох випадках ця обробка повинна враховувати структуру досліджуваної моделі.
7. Як випливає з таблиці 12, найвищу точність має метод кластеризації K-means (96,6%). Далі за спаданням точності слід назвати наступні методи: Gaussian Mixture, Spectral Clustering, Agglomerative Clustering, BIRCH, DBSCAN, OPTICS, Mean Shift.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wunsch D. Clustering / D. Wunsch, R. Xu. — John Wiley & Sons, 2008. — 400 p.
2. Aghabozorgi S. Time-series clustering – a decade review / S. Aghabozorgi, A. Seyed Shirkhorshidi, T. Ying Wah // Information Systems. — 2015. — Vol. 53. — P. 16–38.
3. Saxena A. A review of clustering techniques and developments / A. Saxena, M. Prasad, A. Gupta, [et al.] // Neurocomputing. — 2017. — Vol. 267. — P. 664–681.
4. Aggarwal C. C. Data clustering: algorithms and applications / C. C. Aggarwal, C. K. Reddy. — CRC Press, 2018. — 652 p.
5. Oktar Y. A review of sparsity-based clustering methods / Y. Oktar, M. Turkan // Signal Processing. — 2018. — Vol. 148. — P. 20–30.
6. Bouveyron C. Model-based clustering and classification for data science: with applications in R / C. Bouveyron, G. Celeux, T. B. Murphy, A. E. Raftery. — Cambridge University Press, 2019. — 444 p.
7. Mittal M. Clustering approaches for high-dimensional databases: a review / M. Mittal, L. M. Goyal, D. J. Hemanth, J. K. Sethi // WIREs Data Mining and Knowledge Discovery. — 2019. — Vol. 9, No. 3. — P. e1300.
8. Doreian P. Advances in network clustering and blockmodeling / P. Doreian, V. Batagelj, A. Ferligoj. — John Wiley & Sons, 2020. — 425 p.
9. Ghosal A. A short review on different clustering techniques and their applications / A. Ghosal, A. Nandy, A. K. Das, [et al.]. — Singapore : Springer, 2020. — 69–83 p.
10. Askari S. Fuzzy c-means clustering algorithm for data with unequal cluster sizes and contaminated with noise and outliers: review and development / S. Askari // Expert Systems with Applications. — 2021. — Vol. 165. — P. 113856.
11. Clustering — scikit-learn 1.0.1 documentation, — [ЕЛЕКТРОННИЙ РЕСУРС], — режим доступу: https://scikit-learn.org/stable/modules/clustering.html

12. Metrics and scoring: quantifying the quality of predictions — scikit-learn 1.0.1 documentation, — [ЕЛЕКТРОННИЙ РЕСУРС], — режим доступу: https://scikit-learn.org/stable/modules/model_evaluation.html