Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра
(бакалавра, спеціаліста, магістра)

Студента Пачевський Михайло Володимирович
(ПІБ)
академічної групи 126м-22-1
(шифр)
спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)
за освітньо-професійною програмою «Інформаційні системи та технології»
(офіційна назва)
на тему Комплексна кваліфікаційна робота: Розробка інформаційної технології визначення об'єктів у системах машинного зору в умовах обмежених обчислювальних потужностей
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Коротенко Г.М.			
розділів:				

Рецензент	доц. Ширін А.Л.			

Нормоконтролер	проф. Коротенко Г.М.			

Дніпро
2023

ЗАТВЕРДЖЕНО:
завідувач кафедри
___інформаційних технологій___
_____та комп'ютерної інженерії___
(повна назва)
_____ _Гнатушенко В.В._
(підпис) (прізвище, ініціали)
«_____»_____ 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня _____магістр _____
(бакалавра, спеціаліста, магістра)

студенту _____Пачевський М.В._____ **академічної групи** ___126м-22-1_____
(прізвище та ініціали) (шифр)

спеціальності _____126 «Інформаційні системи та технології»_____ _____

за освітньою-професійною програмою _____

_____«Інформаційні системи та технології»_____

на тему _____Комплексна кваліфікаційна робота: Розробка інформаційної технології визначення об'єктів у системах машинного зору в умовах обмежених обчислювальних потужностей_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 09.10.2021 р. № 1227-с

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз стану області рішення задачі	09.10.2023 – 20.10.2023
Розділ 2	Метрики для визначення якості кластеризації та оцінка просторової і часової складності описаних алгоритмів	21.10.2023 – 30.11.2023
Розділ 3	Порівняння методів кластеризації зображень в умовах обмеженої обчислювальної потужності	1.12.2023 – 18.12.2023

Завдання видано _____ _____Коротенко Г.М._____
(підпис керівника) (прізвище, ініціали)

Дата видачі _____1.10.2023 р._____

Дата подання до екзаменаційної комісії ____18.12.2023 р.____

Прийнято до виконання _____ _____Пачевський М.В._____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 98 стор., 21 рис., 4 додатки, 32 джерела.

Об'єкт дослідження: інформаційна технологія визначення об’єктів на зображеннях в системах машинного зору за умов обмеженої продуктивності.

Предмет дослідження: методи визначення об’єктів на зображеннях в системах машинного зору.

Мета кваліфікаційної роботи: підвищення швидкості визначення об’єкту на зображеннях у системах машинного зору за умови обмеженої продуктивності.

Новизна отриманих результатів полягає у розробці інформаційної технології визначення прямокутника для об’єкта на зображенні за умови обмеженої продуктивності системи машинного зору.

Пояснювальна записка містить опис аналізу теоретичних відомостей про класичні алгоритми кластеризації, аналіз часової та просторової складності цих алгоритмів, опис метрик якості кластеризації та реалізацію методів кластеризації з результатами їх тестування в умовах обмеженої продуктивності, а також опис та реалізацію алгоритму виділення прямокутників, що містять кластери певної площі.

Список ключових слів: АЛГОРИТМ КЛАСТЕРИЗАЦІЇ, ЦЕНТРОЇД, МІРА ДОСТУПНОСТІ, МІРА ПОДІБНІСТЬ, МЕТРИКА КЛАСТЕРИЗАЦІЇ, МОДА, АРТЕФАКТ, K-MEANS, AFFINITY PROPAGATION, DBSCAN, OPTICS, HDBSCAN, MEANSHIFT, SPECTRAL CLUSTERING.

ABSTRACT

Explanatory note: pages 98, figures 21, applications 4, sources 32.

Object of research: information technology for detecting objects in images in machine vision systems under conditions of limited performance.

Subject of research: methods for detecting objects in images in machine vision systems.

Purpose of research: increase the speed of object detection in images in machine vision systems with limited performance.

The novelty of the obtained results is the development of information technology for determining the rectangle for an object in an image under the condition of limited performance of a machine vision system.

The explanatory note contains a description of the analysis of theoretical information about classical clustering algorithms, an analysis of the time and space complexity of these algorithms, a description of clustering quality metrics and the implementation of clustering methods with the results of their testing under limited performance, as well as a description and implementation of an algorithm for selecting rectangles containing clusters of a certain area.

List of keywords: CLUSTERING ALGORITHM, CENTROID, ACCESSIBILITY MEASURE, SIMILARITY MEASURE, CLUSTERING METRIC, FASHION, ARTIFACT, K-MEANS, AFFINITY PROPAGATION, DBSCAN, OPTICS, HDBSCAN, MEANSHIFT, SPECTRAL CLUSTERING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Алгоритм кластеризації - Метод, який групує набір даних на підмножини, звані кластерами, на основі подібності між його елементами.

Центроїд - Центр мас кластера, який зазвичай являє собою середнє значення всіх точок у кластері.

Міра доступності - Міра, що описує, наскільки одна точка "доступна" для іншої точки в термінах подібності.

Міра подібність - Міра схожості між двома об'єктами або кластерами, часто використовується для визначення близькості точок у просторі даних.

Метрика кластеризації - Критерій для оцінювання якості кластеризації. Приклади включають індекси силуету, кореляції Ренда та інші.

Мода - У контексті статистики, це значення, яке зустрічається найчастіше в наборі даних.

Артефакт - Небажане штучне явище або об'єкт, що виникає в результаті роботи алгоритму або процесу.

K-means - Алгоритм кластеризації, який розбиває дані на K кластерів, мінімізуючи середньоквадратичне відхилення між точками і центроїдами кластерів.

Affinity Propagation - Алгоритм кластеризації, заснований на передачі повідомлень між точками для вибору центрів кластерів.

DBSCAN - Алгоритм кластеризації, який формує кластери на основі щільності точок у просторі даних, виділяючи шум.

HDBSCAN - Розширення DBSCAN, яке будує ієрархію кластерів і автоматично визначає оптимальну кількість кластерів.

OPTICS - Алгоритм кластеризації, який упорядковує точки даних у порядку, що відображає структуру кластерів.

MeanShift - Алгоритм, який переміщує центри кластерів у напрямку найбільшого збільшення щільності даних.

Spectral Clustering - Алгоритм кластеризації, заснований на аналізі власних векторів графа схожості між точками даних.

ВСТУП

Актуальність роботи. Сучасний стан розвитку роботизованих систем штучного інтелекту вимагає обробки даних, що надходить каналами машинного зору та машинного слуху у реальному масштабі часу. Це обумовлено тим, що роботизованні системи мусять встигнути відреагувати на швидкоплинні зміни навколишнього середовища та на зміни власного стану включно із зсувом центру ваг просторового положення окремих елементів роботизованої системи тощо. Однією з важливих задач під час аналізу інформації, що надходить від системи машинного зору, є ідентифікація та класифікація об'єктів навколишнього світу, враховуючи те, що роботизовані системи часто мають обмежений обчислювальний ресурс. Використання навіть традиційних алгоритмів іноді може бути недоречним через погану часову складність цих алгоритмів. Дані, що надходять у реальному масштабі часу, можуть бути обробленні невчасно або взагалі загубленні. Таким чином, постає питання: який з алгоритмів традиційного виділення точок, що належать зображенню певного об'єкта обрати. Добір має давати достатню точність за умови кращої часової і бажано просторової складності алгоритму.

Об'єктом досліджень є алгоритми кластеризації точок на зображенні.

Предметом досліджень є часова та просторова складність традиційних алгоритмів кластеризації під час їх використання на пристроях з обмеженою обчислювальною потужністю.

Мета роботи – підвищення швидкості кластеризації точок на зображені за умови збереження високої точності алгоритму під час використання на пристроях з обмеженою обчислювальною потужністю, а також у визначенні кращого способу використання традиційних алгоритмів на пристроях з обмеженою обчислювальною потужністю.

Пояснювальна записка містить змістовний опис алгоритмів кластеризації теоретичних оцінок їх просторової та обчислювальної

складності та практичних результатів тестування означених алгоритмів на пристрої з обмеженою обчислювальною потужністю.

Постановка задачі

За результатами аналізу традиційних алгоритмів кластеризації, їхньої теоретичної та часової та просторової складності провести випробування практичного застосування цих алгоритмів на пристрої з обмеженою обчислювальною потужністю у різних режимах експлуатації. За результатами практичних випробувань обґрунтувати використання традиційного алгоритму кластеризації та певного режиму його використання.

Висновок. У цій атестаційній роботі будуть розглянуті традиційні алгоритми кластеризації їхня математична сутність теоретичні оцінки часової та просторової складності будуть проведенні випробування алгоритмів в одно поточному та многопоточних режимах та буде обґрунтовано використання одного з цих алгоритмів та певного режиму його використання.

РОЗДІЛ 1
АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧІ

1. Стан проблема

Сучасні роботизовані системи досить часто мають у своєму складі підсистеми машинного зору. Разом із засобами акустичної та радіолокації система машинного зору можуть допомогти автономному роботу орієнтуватись у просторі та використовувати власні маніпулятори задля взаємодії з навколишнім середовищем. Такі роботизовані системи залежно від характеру задач задля яких вони були створені можуть мати потужний обчислювальний компонент віддалений у просторі (окремий обчислювальний блок чи блоки, під'єднані засобами комп'ютерних мереж). Однак, деякі роботизовані системи мусять самостійно приймати рішення спираючись на результати власної автономної підсистеми машинного зору. Наразі для таких задач були створені спеціалізовані машинні чіпи, однак їхня продуктивність хоча й перевищує продуктивність мобільних процесорів загального призначення, все одно залишається невеликою. Таким чином в автономних підсистемах машинного зору можуть виникати проблеми зі швидкістю прийняття рішень через затримки в процесі обробки графічної інформації.

До задач обробки належать: пошук об'єктів на зображеннях, їхнє розпізнавання, оцінка відстані до об'єктів, оцінка руху об'єктів, стеження за об'єктами тощо. Переважна більшість складних задач таких як стеження, розпізнавання та інше, ефективно розв'язується тільки методами застосування штучних нейронних мереж. Але використанню цих мереж передує визначення об'єктів на зображенні. Це завдання можна розв'язувати як більш традиційними методами, так і застосовуючи нейронні мережі. У цій роботі буде розглянуто традиційні методи та за результатами аналізу їх застосування в умовах обмеженої продуктивності буде запропоновано

використання методу, який за умови достатньої точності має кращу часову оцінку.

Розглядаючи задачу визначення об’єктів на зображенні можна припустити, що вона складається з двох частин. По-перше, за результатами аналізу кольору об’єктів необхідно визначити точки зображення, які належать тим чи іншим об’єктам. Далі, використовуючи результати аналізу точок, можна застосувати алгоритми пошуку меж об’єктів, які нададуть можливість окреслити геометричні форми тої чи іншої речі. Однак враховуючи той факт, що не завжди потрібно чітко визначати геометричну форму об’єкту, а достатньо зрозуміти, який це об’єкт, поставлена задача може бути спрощена до використання того чи іншого методу кластеризації точок на зображенні з подальшим пошуком крайніх точок кожного об’єкту за час O(n^2). Таким чином буде отримано прямокутник в якому міститься об'єкт і це може бути використано для роботи класифікаційної нейронної мережі. Тож, фактично, необхідно дослідити методи кластеризації точок на зображенні та їхню реальну часову оцінку та якість отриманих результатів.

1.1. Що таке кластеризація? І коли її варто застосовувати?

Кластерний аналіз[5] – це об’єднання в групи більш схожих між собою об’єктів. Класифікація на кластери здійснюється за допомогою таких критеріїв, як найменша відстань, щільність точок даних, графіки або різні статистичні розподіли за допомогою яких треба створити набори кластерів які якомога більше будуть відрізнятись один від одного при мінімальній відмінності об’єктів в середині кожного кластеру. Кластерний аналіз має широке застосування, в тому числі в некерованому машинному навчанні, інтелектуальному аналізі даних, статистиці, графічній аналітиці, обробці зображень, а також у численних фізичних і соціальних науках.

Кластеризацію варто застосовувати коли треба розпочати з великого, неструктурованого набору даних

Кластеризація великих наборів даних є, мабуть, найціннішим застосуванням цього інструменту аналізу завдяки великій кількості роботи, якої він дозволяє уникнути. Як і у випадку з іншими інструментами навчання без нагляду, кластеризація може брати великі набори даних і без інструкцій швидко організовувати їх у щось більш придатне для використання. Найкраще те, що якщо немає мети виконати масивний аналіз, кластеризація може дати швидкі відповіді про подані дані.

Якщо невідомо, на скільки або на які класи поділяються дані.

Навіть якщо почати з більш структурованого і добре маркованого набору даних, він може не мати тієї глибини і стратифікації, яку потрібно знайти. Кластеризація - це чудовий перший крок у підготовці даних, оскільки вона починає давати відповіді на ключові питання про набір даних. Наприклад, можна виявити, що те, що вважалось двома основними підмножинами, насправді є чотирма, або що категорії, про які не було відомо, є окремими класами.

Коли ручне розділення та анотування даних забирає надто багато ресурсів.

Для невеликих наборів даних ручна анотація та організація можлива, якщо не ідеальна. Однак, коли дані починають масштабуватися, анотування, класифікація та категоризація стають експоненціально складнішими[8]. Кластеризація - залежно від алгоритму, який використовується, - може скоротити час на анотування та класифікацію, оскільки вона менш зацікавлена в конкретних результатах і більше зосереджена на самій категоризації. Наприклад, алгоритми розпізнавання мови створюють мільйони точок даних, на повну анотацію яких знадобляться сотні годин. Алгоритми кластеризації можуть скоротити загальний час роботи і дати відповіді швидше.

Коли треба знайти аномалії у даних.

Одне з найцінніших застосувань кластеризації полягає в тому, що завдяки чутливості багатьох алгоритмів до точок, що виділяються, вони

можуть слугувати ідентифікаторами аномалій у даних. Дійсно, алгоритми кластерного аналізу, такі як DBSCAN призначені для пошуку окремих кластерів, які розташовані близько один до одного, і позначення викидів у наборах даних. Розуміння аномальних даних може допомогти оптимізувати існуючі інструменти збору даних і привести до більш точних результатів у довгостроковій перспективі.

Варіанти використання кластеризації.

Зі зростанням кількості доступних алгоритмів кластеризації не дивно, що кластеризація стала основною методологією в різних видах бізнесу та організацій з різними варіантами використання. Серед прикладів використання кластеризації - аналіз біологічних послідовностей, генетична кластеризація людини, кластеризація тканин медичних зображень, сегментація ринку або клієнтів, групування соціальних мереж або результатів пошуку для рекомендацій, виявлення аномалій у комп'ютерних мережах, обробка природної мови для групування текстів, кластерний аналіз злочинів і кліматичний кластерний аналіз. Нижче наведено опис деяких прикладів.

Класифікація мережевого трафіку. Організації шукають різні способи зрозуміти різні типи трафіку, що надходить на їхні веб-сайти, зокрема, що є спамом, а що - ботами. Кластеризація використовується для групування спільних характеристик джерел трафіку, а потім створення кластерів для класифікації та диференціації типів трафіку[7]. Це дозволяє надійніше блокувати трафік, а також дає змогу краще зрозуміти, як стимулювати зростання трафіку з бажаних джерел.

Маркетинг і продажі. Успіх у маркетингу означає націленість на потрібних людей або потенційних клієнтів у правильний спосіб. Алгоритми кластеризації об'єднують людей зі схожими рисами, можливо, на основі їхньої ймовірності покупки.

Аналіз документів. Будь-яка організація, що має справу з великими обсягами документів, виграє, якщо зможе ефективно і швидко впорядковувати їх по мірі створення. Це означає, що буде можливо зрозуміти

основні теми в документах, а потім порівняти їх з іншими документами. Алгоритми кластеризації досліджують текст у документах, а потім групують їх у кластери за різними темами[32]. Таким чином їх можна швидко впорядкувати відповідно до фактичного змісту.

Приклади областей використання кластеризації:

- Маркетинг: Може бути використана для характеристики та виявлення сегментів клієнтів для маркетингових цілей.
- Біологія: Може використовуватися для класифікації серед різних видів рослин і тварин.
- Бібліотеки: Використовується для кластеризації різних книг на основі тем та інформації.
- Страхування: Використовується для розпізнавання клієнтів, їхніх полісів та виявлення шахрайства.
- Міське планування: Використовується для створення груп будинків і вивчення їхньої вартості на основі географічного розташування та інших факторів.
- Вивчення землетрусів: Вивчаючи райони, що постраждали від землетрусів, ми можемо визначити небезпечні зони.
- Обробка зображень: Кластеризація може бути використана для групування схожих зображень, класифікації зображень на основі вмісту та виявлення закономірностей у даних зображень.
- Генетика: Кластеризація використовується для групування генів, які мають схожі моделі експресії, та виявлення генних мереж, які працюють разом у біологічних процесах.
- Фінанси: Кластеризація використовується для визначення сегментів ринку на основі поведінки клієнтів, виявлення закономірностей у даних фондового ринку та аналізу ризиків в інвестиційних портфелях.

- Обслуговування клієнтів: Кластеризація використовується для групування запитів та скарг клієнтів за категоріями, виявлення спільних проблем та розробки цільових рішень.
- Виробництво: Кластеризація використовується для групування схожих продуктів, оптимізації виробничих процесів та виявлення дефектів у виробничих процесах.
- Медична діагностика: Кластеризація використовується для групування пацієнтів зі схожими симптомами або захворюваннями, що допомагає ставити точні діагнози та визначати ефективні методи лікування.
- Виявлення шахрайства: Кластеризація використовується для виявлення підозрілих шаблонів або аномалій у фінансових операціях, що може допомогти у виявленні шахрайства або інших фінансових злочинів.
- Аналіз трафіку: Кластеризація використовується для групування подібних даних про транспортні потоки, таких як години пік, маршрути та швидкість, що може допомогти в покращенні транспортного планування та інфраструктури.
- Кібербезпека: Кластеризація використовується для групування схожих моделей мережевого трафіку або поведінки системи, що може допомогти у виявленні та запобіганні кібератак.
- Аналіз клімату: Кластеризація використовується для групування подібних моделей кліматичних даних, таких як температура, опади і вітер, що може допомогти в розумінні зміни клімату та його впливу на навколишнє середовище.
- Аналіз спорту: Кластеризація використовується для групування схожих моделей даних про продуктивність гравців або команд, що може допомогти в аналізі сильних і слабких сторін гравців або команд і прийнятті стратегічних рішень.
- Кримінальний аналіз: Кластеризація використовується для групування схожих даних про злочини, таких як місце, час і тип, що може

допомогти у виявленні гарячих точок злочинності, прогнозуванні майбутніх тенденцій злочинності та вдосконаленні стратегій запобігання злочинам.

Список не є вичерпним.

1.2. Як проводиться кластерний аналіз? Що таке алгоритми кластеризації?

Алгоритми кластеризації використовуються для групування точок даних на основі певної схожості. Не існує критерію хорошої кластеризації[6]. Кластеризація визначає групування немічених даних. Це в основному залежить від конкретного користувача і сценарію.

Типові моделі кластерів включають:

- Моделі зв'язності - як ієрархічна кластеризація, яка будує моделі на основі зв'язності за відстанні.
- Центроїдні моделі - подібні до кластеризації за методом K-means, який формує набір середніх значень для кожного з кластерів.
- Моделі розподілу – моделі розподілу мають наступну ідею таку як кластери даних мають схожі розподіли.
- Моделі щільності - такі як DBSCAN, HDBSCAN, OPTICS(+) які визначають кластеризацію як пов'язану щільну область у просторі даних.
- Групові моделі - ці моделі не дають точних результатів. Вони лише пропонують інформацію про групування.
- Графові моделі - підмножина вузлів у графі, де ребро з'єднує кожні два вузли в підмножині, може розглядатися як прототипна форма кластера.
- Нейронні моделі - карти, що самоорганізуються, є одними з найвідоміших некерованих нейронних мереж (НМ), і вони характеризуються як подібні до однієї або декількох моделей, описаних вище.

Треба звернути увагу, що існують різні типи кластеризації:

- Жорстка кластеризація - точка даних або повністю належить кластеру, або ні. Наприклад, сегментація клієнтів на чотири групи. Кожен клієнт може належати до однієї з чотирьох груп.
- М'яка кластеризація - точкам даних присвоюється оцінка ймовірності потрапляння в ці кластери.

Важливо зазначити, що аналіз кластерів не є роботою одного алгоритму. Навпаки, різні алгоритми зазвичай беруть на себе ширше завдання аналізу, кожен з яких часто суттєво відрізняється від інших. В ідеалі, алгоритм кластеризації створює кластери, де внутрішньокластерна схожість дуже висока, тобто дані всередині кластера дуже схожі один на одного. Також алгоритм повинен створювати кластери, де міжкластерна схожість[19] набагато менша, тобто кожен кластер містить інформацію, яка якомога менше відрізняється від інших кластерів.

Існує багато алгоритмів кластеризації, просто тому, що існує багато уявлень про те, яким має бути кластер або як його визначати. Насправді, на сьогоднішній день існує понад 100 алгоритмів кластеризації, які були опубліковані. Вони являють собою потужну техніку машинного навчання на неконтрольованих даних. Алгоритм, побудований і розроблений для певного типу кластерної моделі, як правило, не спрацьовує, якщо його застосувати до набору даних, що містить зовсім інший тип кластерної моделі.

Спільною рисою всіх алгоритмів кластеризації є група об'єктів даних. Але аналітики даних і програмісти використовують різні моделі кластерів, причому кожна модель вимагає свого алгоритму. Кластери або набори кластерів часто розрізняють як жорстку кластеризацію, коли кожен об'єкт належить до кластера або ні, або м'яку кластеризацію, коли кожен об'єкт належить до кожного кластера певною мірою.

1.2.1. Алгоритм Kmeans

Алгоритм K-means - це алгоритм[10] кластеризації, який намагається з поданого набору даних утворити K кластерів підгруп (кластерів), які не

перетинаються, в яких кожна точка даних належить тільки одній групі. При цьому внутрішньокластерні точки даних намагаються зробити якомога більш схожими, а кластери – якомога більш відмінними один від одного «далекими по відстані». Мінімізуючи суму квадратів відстаней від точок до центроїдів кластерів точки розподіляють по кластерах. Що менший розкид усередині кластерів, то більш однорідними (схожими) є точки даних в одному кластері.

Робота алгоритму K-means виглядає наступним чином:

1. Алгоритм потребує попередього визначення кількості кластерів K;
2. Ініціалізація центроїдів відбувається шляхом випадкового перемішування вхідних даних та подальшої випадкової вибірки з нього K точок.
3. Треба продовжити ітерації доти, доки центроїди не зміняться, тобто не зміниться розподіл точок даних за кластерами;
4. Обчислити суму квадратів відстаней між точками даних і всіма центроїдами;
5. Присвоїти кожній точці даних найближчий кластер (центроїд);
6. Обчислити центроїди обчисливши середнє значення між всіма точками в кластері.

Підхід, який використовує k-means для вирішення цього завдання, називається Expectation-Maximization. Крок Е полягає у віднесенні точок даних до найближчого кластера. М-крок полягає в обчисленні центроїда кожного кластера. Нижче наведено математичний опис розв'язання цієї задачі. Цільова функція має вигляд:

$$J = \sum_{i=1}^{m} \sum_{k=1}^{k} w_{ik} \|x^i - \mu_k\|^2 \quad (1.1)$$

де w_{ik}=1 для точки даних x_i, якщо вона належить до кластеру k, інакше $w_{ik} = 0$. Також, μk - це центроїд кластеру x_i.

Задача мінімізації складається з двох частин[18]. Спочатку треба мінімізувати J w_{rt} w_{ik} і вважається що μk фіксоване. Потім треба

мінімізувати $J\ w_{rt}.\ \mu k$ і вважається що w_{ik} фіксоване. Технічно кажучи, спочатку відбувається диференціація J w.r.t. w_{ik} і оновлюється призначення кластерів (Е-крок). Потім диференціація $J\ w_{rt}.\ \mu k$ і переобчислюються центроїди після кластерних розподілів з попереднього кроку (М-крок). Таким чином, Е-крок дорівнює:

$$\frac{\partial J}{\partial w_{ik}} = \sum_{i=1}^{m}\sum_{k=1}^{k} \|x^i - \mu_k\|^2$$

$$\Rightarrow w_{ik} = \{1\ if\ k = argmin_j \|x^i - \mu_k\|^2\ 0\ otherwise. \quad (1.2)$$

Іншими словами, віднести точку даних x_i до найближчого кластера за сумою квадратів відстаней від центроїда кластера.

І М-крок:

$$\frac{\partial J}{\partial \mu_k} = 2\sum_{i=1}^{m} \omega_{ik}\left(x^i - \mu_k\right) = 0$$

$$\Rightarrow \mu_k = \frac{\sum_{i=1}^{m} \omega_{ik} x^i}{\sum_{i=1}^{m} \omega_{ik}} \quad (1.3)$$

Це означає переобчислення центроїда кожного кластера, щоб відобразити нові призначення.

Тут слід зазначити кілька речей:

– Оскількі різні алгоритми кластеризації включно з K-means використовують міру відстані для визначення приналежності точок до кластера бажано нормалізувати вхідний набор даних з математичним очікуванням 0 та дисперсією 1 [13].

– З урахуванням ітеративної природи k-means його варто запускати кілька разів з різними даними початкової ініціалізації і обрати той результат сума квадратів відстаней якого менша.

Розподіл прикладів не змінюється - це те саме, що не змінюється варіація всередині кластера:

$$\frac{1}{m_k}\sum_{i=1}^{m_k} \|x^i - \mu_{c^k}\|^2 \quad (1.4)$$

1.2.2. Affinity Propagation

Метод Affinity Propagation (AP) був опублікований Фреєм і Дукеком у 2007 році і стає дедалі популярнішим завдяки своїй простоті, загальній застосовності та ефективності.

Основними недоліками K-means та подібних алгоритмів є необхідність вибору кількості кластерів та вибору початкового набору точок. Натомість Affinity Propagation бере на вході міру схожості між парами точок даних і одночасно розглядає всі точки даних як потенційні зразки. Між точками даних відбувається обмін повідомленнями з реальним значенням, доки поступово не сформується високоякісний набір зразків і відповідних кластерів[15].

На вході алгоритм вимагає два набори даних:

Схожість між точками даних, яка показує, наскільки добре одна точка підходить для того, щоб бути зразком для іншої. Якщо між двома точками немає подібності, тобто вони не можуть належати до одного кластеру, цю схожість можна опустити або встановити в -Нескінченність, залежно від реалізації.

Уподобання, що відображають придатність кожної точки даних бути зразком. Можна мати апріорну інформацію про те, які точки можуть бути кращими для цієї ролі, і тому можна представити її за допомогою преференцій.

Як схожість так і преференції часто подають за допомогою однієї матриці, де значення на головній діагоналі формують схожість. Матричне представлення добре підходить для щільних наборів даних. Там, де зв'язки між точками розрідженні, практичніше не зберігати всю матрицю n x n в пам'яті, а замість цього зберігати список схожостей для з'єднаних точок. За лаштунками "обмін повідомленнями між точками" - це те ж саме, що і маніпулювання матрицями, і це лише питання точки зору та реалізація.

Потім алгоритм виконує декілька ітерацій, доки не зійдеться. Кожна ітерація складається з двох кроків передачі повідомлень:

- Обчислення відповідальності: Відповідальність $r(i,k)$ відображає накопичені докази того, наскільки добре точка k підходить в якості зразка для точки i, беручи до уваги інші потенційні зразки для точки i. Відповідальність надсилається від точки даних і до точки-кандидата на роль зразка k.
- Обчислення доступності: Доступність $a(i,k)$ відображає накопичені докази того, наскільки доречно для точки i було б обрати точку k в якості зразка, беручи до уваги підтримку інших точок, що точка k має бути зразком. Доступність надсилається від точки-кандидата на роль зразка k до точки i.

Для обчислення відповідності алгоритм використовує вихідні подібності та доступності, обчислені на попередній ітерації (спочатку всі доступності дорівнюють нулю). Відповідність встановлюється на основі вхідної схожості між точкою i та точкою k як її зразком, мінус найбільша з сум схожості та доступності між точкою і та іншими зразками-кандидатами. Логіка розрахунку придатності точки для зразка полягає в тому, що їй надається більша перевага, якщо початкова апріорна перевага була вищою, але відповідальність зменшується, коли є схожа точка, яка вважає себе хорошим кандидатом, тому між ними відбувається "конкуренція", доки одна з них не буде обрана на певній ітерації.

Обчислення доступності, таким чином, використовує обчислені обов'язки як доказ того, що кожен кандидат може бути хорошим прикладом. Доступність $a(i,k)$ дорівнює власній відповідальності $r(k,k)$ плюс сума позитивних оцінок, які кандидат-зразок k отримав від інших точок.

Нарешті, можна отримати різні критерії зупинки для завершення процедури, наприклад, коли зміни значень падають нижче певного порогу, або коли досягається максимальна кількість ітерацій. На будь-якому етапі

процедури поширення спорідненості підсумовування матриць відповідальності (r) та доступності (a) дає необхідну інформацію про кластеризацію: для точки i, k з максимальним значенням $r(i,k) + a(i,k)$ представляє зразок точки i. Або, якщо потрібен лише набір зразків, можна просканувати головну діагональ. Якщо $r(i,i) + a(i,i) > 0$, то точка i є зразком.

Треба зазначити, що за допомогою K-means та подібних алгоритмів визначення кількості кластерів може бути складним завданням. З Affinity Propagation не потрібно явно вказувати кількість кластерів, але все одно може знадобитися певне налаштування, якщо буде отримано більше або менше кластерів, ніж вважається оптимальним. На щастя, Affinity Propagation дає можливість зменшити або збільшити кількість кластерів, просто змінивши налаштування. Встановлення більшого значення параметрів призведе до більшої кількості кластерів, оскільки кожна точка є "більш впевненою" у своїй придатності бути зразком, і тому її важче "обійти" і включити до якогось іншого кластеру "домінування". І навпаки, встановлення нижчих параметрів призведе до меншої кількості кластерів. Як правило, можна встановити всі уподобання на середню схожість для середньої та великої кількості кластерів, або на найменшу схожість для помірної кількості кластерів. Однак, може знадобитися кілька прогонів з коригуванням параметрів, щоб отримати результат, який точно відповідає вказаним потребам.

Варто також згадати ієрархічне поширення спорідненості, як варіант алгоритму, який має справу з квадратичною складністю, розбиваючи набір даних на кілька підмножин, кластеризуючи їх окремо, а потім виконуючи другий рівень кластеризації.

1.2.3. DBSCAN

Кластери це щільні простори у наборі даних розділені розрідженими областями. В наборі даних можуть існувати значення які не можна віднести

до жодного кластеру та які самі не можуть утворити кластери ці дані утворюють шум у поданому наборі даних. Алгоритм DBSCAN базується саме на поняттях щільності та шуму. Ідея алгоритму полягає в тому, що кластери утворюються з точок які є щільно розташованими в просторі ознак[11].

Параметри, необхідні для алгоритму DBSCAN:

eps: Визначає околиці навколо точки даних, тобто точки є сусідами якщо відстань між ними менша за eps. Якщо число eps замале то більша частина поданих даних буде вважатись шумом. А якщо eps буде завелике вхідні дані зливатимуться й більшість даних потрапить не до своїх кластерів.

Один із способів знайти значення eps базується на графіку k-відстаней.

MinPts – параметр описаного алгоритму який визначає скільки мінімально має бути сусідів аби вона не вважалася шумом. Найпоширенішим способом визначення цього параметру є обрання значення яке більше або дорівнює кількості вимирів D $MinPts >= D + 1$.

MinPts: Мінімальна кількість сусідів (точок даних) в радіусі eps. Чим більший набір даних, тим більше значення MinPts[27] має бути вибрано. Як правило, мінімальне значення MinPts може бути отримане з кількості вимірів D у наборі даних, тобто $MinPts >= D + 1$. Мінімальне значення MinPts повинно бути не менше 3.

У цьому алгоритмі розглядається 3 типи точок даних.

Основна точка: Точка є опорною, якщо вона має більше ніж MinPts точок в межах eps.

Гранична точка: Точка, яка має менше ніж MinPts в межах eps, але знаходиться по сусідству з основною точкою.

Шум або викид: Точка, яка не є основною або граничною точкою.

Кроки, що використовуються в алгоритмі DBSCAN:

1. Знаходяться всі сусідні точки в межах eps та визначаються основні точки або точки, які відвідуються з більш ніж MinPts сусідами.

2. Для кожної центральної точки, якщо вона ще не віднесена до кластера, створюється новий кластер.

3. Знаходяться рекурсивно всі точки, пов'язані щільністю, і відносяться до того ж кластеру, що і центральна точка.

4. Точки a та b вважаються щільно зв'язаними, якщо існує точка c, яка має достатню кількість точок серед своїх сусідів, і обидві точки a та b знаходяться на відстані eps. Це ланцюговий процес. Отже, якщо b є сусідом c, то c є сусідом d, a d є сусідом e, який в свою чергу є сусідом a, що означає, що b є сусідом a.

Далі треба пройтись по усім невідвіданим точкам у наборі даних. Ті точки, які не належать до жодного кластеру, є шумом.

Якщо порівнювати кластеризацію за методом K-means та методом DBSCAN, то можно виділити наступне:

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) і K-Means - це алгоритми кластеризації, які групують дані, що мають однакову характеристику. Однак вони працюють на різних принципах і підходять для різних типів даних. Вважається краще використовувати DBSCAN, коли дані не мають сферичної форми або кількість класів не відома заздалегідь[28].

У нижче приведеній таблиці буде порівняно алгоритм DBSCAN з вище згаданим алгоритмом K-means.

Таблиця 1.1

Порівняння алгоритмів DBSCAN та K-means

DBSCAN	K-Means
У DBSCAN не потрібно вказувати кількість кластерів.	K-Means дуже чутливий до кількості кластерів, тому їх кількість потрібно вказувати
Кластери, сформовані в DBSCAN, можуть мати довільну форму.	Кластери, сформовані в K-Means, мають сферичну або опуклі за формою
DBSCAN може добре працювати з наборами даних, що містять шум та викиди	K-Means погано працює з даними з викидами. Пропуски можуть дуже сильно викривляти кластери в K-Means.
У DBSCAN для навчання моделі потрібні два параметри	У K-Means для навчання моделі потрібен лише один параметр

1.2.4. HDBSCAN*

HDBSCAN* розшифровується як Hierarchical DBSCAN. Суфікс зірочка вказує на покращення, яке ті ж самі автори зробили в DBSCAN. Ці автори розширили свою роботу, представивши повний фреймворк для кластерного аналізу та виявлення викидів, який включає розширене пояснення алгоритму HDBSCAN. HDBSCAN покращує DBSCAN шляхом створення ієрархічного представлення кластерів. Ієрархія, отримана в результаті виконання алгоритму, може бути дуже ефективно використана для виділення кластерів і виявлення викидів. HDBSCAN долає обмеження DBSCAN, ідентифікуючи кластери будь-якої щільності. Хоча алгоритм HDBSCAN має ряд переваг, він має один недолік, який слід враховувати.

Алгоритм[1] має загальну асимптотичну складність $O(n ** 2)$.

Крім того, алгоритм має багато підкроків, які самі по собі мають складність $O(n ** 2)$.

Ця складність може мати значний вплив на час виконання алгоритму для великих наборів даних. Існують реалізації згаданого вище алгоритму K-means, які мають складність $O(n)$

Крім набору даних для кластеризації, цей алгоритм має два обов'язкових вхідних параметри. Перший параметр - це мінімальна кількість точок s, що використовується для обчислення відстані,

Другим необхідним параметром є мінімальний розмір кластера p, який визначає нижню межу кількості точок даних, необхідних для формування кластера. Набір даних - це набір однорідних точок, де точка є записом числових ознак. На рис. 1.1 показано простий набір даних з 39 двовимірними точками, який буде використовуватися як приклад.

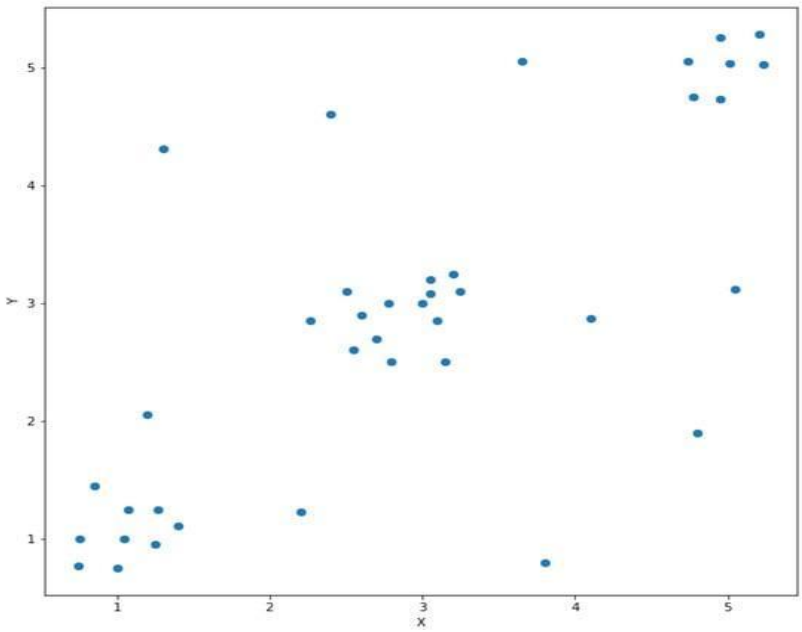


Рис. 1.1 Результат кластеризації HDBSCAN

Першим кроком алгоритму є обчислення базових відстаней для кожної точки в наборі даних. Базова відстань точки - це відстань до k-го найближчого сусіда. Значення k включає саму точку. Це метод для отримання приблизної щільності для кожної точки. На рис. 1.2 показано базові відстані для точок (2.7,2.7) та (3.0,3.0) для $k = 5$ простого набору даних. Потрібно звернути увагу, що 5-й найближчий сусід обох точок є спільним. Точка (3,0,3,0) має менше значення відстані до ядра, ніж точка

(2.7,2.7) тому що її сусідні точки ближчі, що означає, що вона має вищу апроксимаційну щільність.

Використовуючи основні відстані, обчислюється нова метрика відстані, яка називається відстанню взаємної досяжності. Відстань взаємної досяжності між двома точками x та y - це максимальне значення: центральної відстані x, центральної відстані y або відстані між x та y. Точки (2.7,2.7) та (3.0,3.0)

показані на рис. 1.2, мають відстань взаємної досяжності, яка дорівнює відстані між двома точками. Зокрема, для ????=5 відстань між точками (2.7,2.7) дорівнює 0.22; відстань між точками (3.0,3.0) дорівнює 0.31, а відстань між двома точками дорівнює 0.42

Отже, відстань взаємної досяжності між цими точками дорівнює 0.42

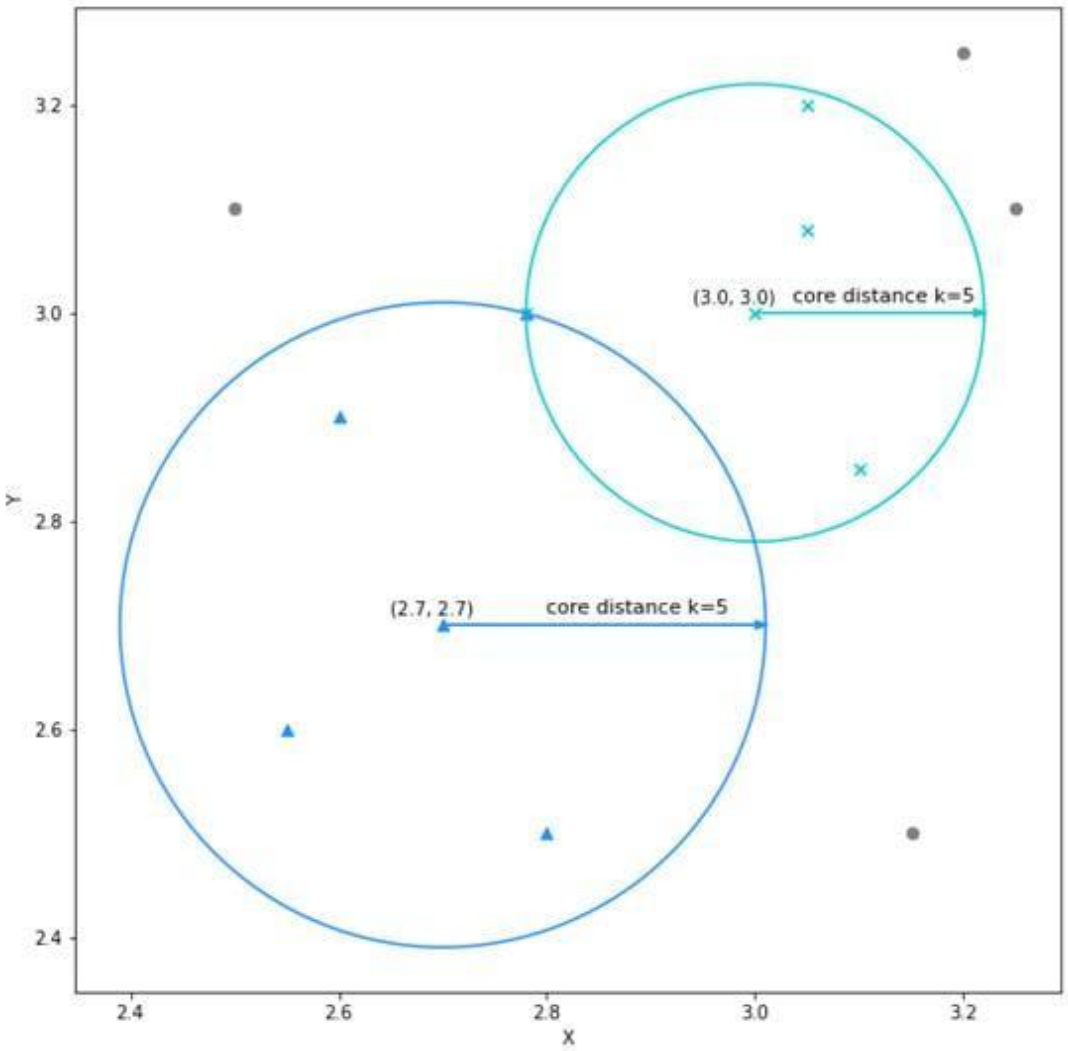


Рис 1.2. Взаємна досяжність точок HDBSCAN

Далі, відстані взаємної досяжності між точками можуть бути використані для побудови зваженого графа, де точки даних є вершинами, а ребро між будь-якими двома точками має вагу, рівну відстані взаємної досяжності між цими точками. Цей повний граф є лише концептуальним артефактом в алгоритмі, оскільки потрібно обчислити мінімальне остовне дерево графа. Мінімальне остовне дерево - це остовне дерево, сума ваг ребер якого є найменшою. Тобто всі його вершини з'єднані з підмножиною ребер з повного графа, без циклів і з мінімально можливою сумарною вагою ребер. Отримане мінімальне остовне дерево модифікується шляхом додавання до кожної вершини самого ребра з відстанню до ядра точки як вагою.

Це дає граф, який називається розширеним мінімальним остовним деревом. На рис. 1.3 показано мінімальне остовне дерево для простого набору даних.

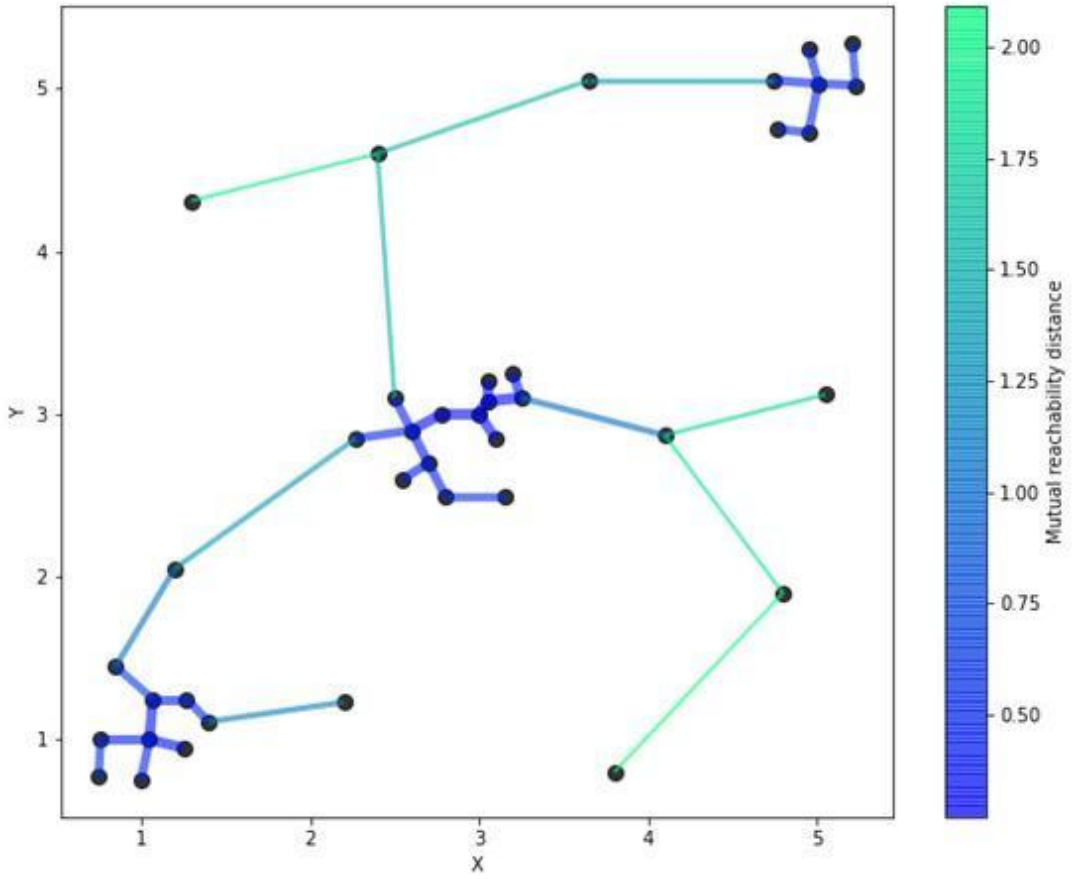


Рис 1.3. Мінімальне остовне дерево для простого набору даних

Тепер граф можна використовувати для побудови ієрархії HDBSCAN*. Для початку створюється одна мітка кластера, і всі точки відносяться до цього кластера. Цей кластер додається до списку кластерів. Потім граф сортується за вагою ребер у порядку зростання. Починаючи з нижньої частини графа, ребра ітеративно видаляються з розширеного мінімального остовного дерева. Ребра з однаковими вагами повинні видалятися одночасно. Значення ваги ребра (ребер), що видаляється, використовується для позначення поточного ієрархічного рівня. При видаленні ребра досліджується кластер, що містить видалене ребро. Кластерам, які стали роз'єднаними і містять менше точок, ніж s (мінімальний розмір кластера), присвоюється мітка шуму. Кластеру, який роз'єднався, але містить більше ніж s точок, присвоюється нова мітка кластера.

Це називається розбиттям кластера. Крім того, новий кластер додається до списку кластерів. Новий рівень ієрархії створюється, коли видалення ребра призвело до появи нових кластерів внаслідок розбиття кластерів. Наприкінці процесу, на останньому рівні ієрархії, всі точки в наборі даних будуть віднесені до шуму. Ієрархія, отримана в результаті цього процесу, є ієрархією HDBSCAN*.

На рис. 1.4 показано табличне представлення ієрархії HDBSCAN* для простого набору даних. Кожен стовпчик таблиці представляє точку даних, а рядки містять призначення кластерів під час роботи алгоритму. У верхньому рядку всі точки віднесено до одного кластера[31]. У другому рядку деякі точки дорівнюють нулю, оскільки вони були відокремлені і позначені як викиди. У третьому рядку тепер є два окремі кластери і один додатковий викид. В останньому рядку всі точки позначені як викиди або точки шуму. Під час побудови ієрархії HDBSCAN* також ведеться список кластерів, де кожен кластер має посилання на свого батька.

Edge Weight	Cluster Labels: Each Column is a Data Point from the Dataset
1.97230829	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1.5435349	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 0 1 1 1 0 1 1 1 0 1 1
1.3360015	2 2 2 2 2 2 2 2 2 2 0 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 3 0 2 3 3 0 3 3 3 0 3 3
1.32853303	4 4 4 4 4 4 4 4 4 4 0 4 4 5 0 5 5 5 5 5 5 5 5 5 5 5 5 5 3 0 5 3 3 0 3 3 3 0 3 3
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Рис 1.4. Табличне представлення ієрархії HDBSCAN* для простого набору даних

Після побудови ієрархії HDBSCAN* та складання списку кластерів, наступним кроком є визначення видатних кластерів з цієї ієрархії. Для цього необхідно встановити нову міру, яку можна використовувати для визначення стабільності кластера. Вона називається лямбда, тобто $\lambda = 1/\text{вага ребра}$.

Далі, для кластера також визначається значення $\lambda birth$ та $\lambda death$ значення лямбда, коли новий кластер було створено в результаті поділу кластера, та значення лямбда, коли той самий кластер було поділено, відповідно. Для кожної точки кластера можливо визначити значення λp як значення лямбда, при якому ця точка випала з кластера. Стабільність для кластера можна обчислити, як показано в формулі 1.5.

$$stability = \sum_{P \epsilon Cluster}(\lambda_P - \lambda_{birth}) \quad (1.5)$$

Стабільність потрібно поширювати через кластери. Листяні кластери - це кластери без дочірніх кластерів, і їх можна знайти у списку кластерів. Починаючи з цих кластерів, потрібно рухатися вгору, використовуючи посилання на батьківський кластер. Листяні кластери завжди передають свою стабільність батькам і додають себе до батьківського кластера як розмноженого нащадка. Для нелистяних кластерів відбудеться одна з двох речей. Якщо кластер, що обробляється, має вищу стабільність, ніж кумулятивна стабільність його нащадків, то лише вона буде передана батьківському кластеру. В іншому випадку до батьківського кластера буде передано кумулятивну стабільність усіх нащадків поточного кластера. Очевидно, що для кореневого кластера не відбувається ніякого поширення, оскільки він не має батька. Коли цей процес буде завершено, кореневий кластер міститиме посилання на нащадкові кластери з найвищою

стабільністю. Кластери з найвищою стабільністю є найбільш помітними кластерами. Використовуючи ієрархію HDBSCAN* разом з деталями найбільш помітних кластерів, можна швидко згенерувати список кластерних призначень для кожної точки даних. Це найбільш значущий артефакт, що генерується як результат роботи алгоритму HDBSCAN*. Основні кластери, виявлені за допомогою простого набору даних, показані на рис. 1.5. Виявлено три кластери, а п'ять жовтих точок визначено як викиди.

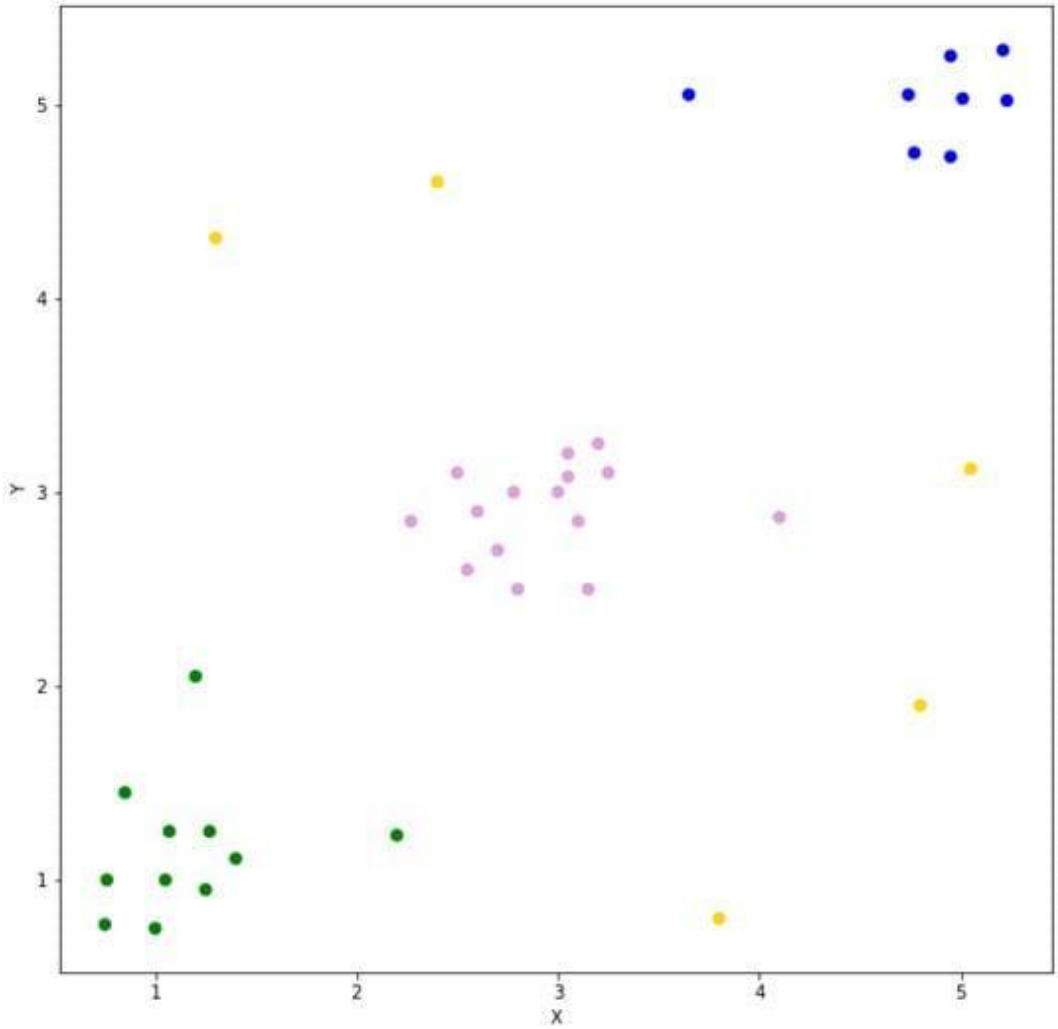


Рис 1.5. Розподіл по класах за результатами HDBSCAN*

Ще одним важливим артефактом, який можна отримати в результаті виконання алгоритму HDBSCAN*, є оцінка викидів для кожної точки даних. У літературі це називається GLOSH точки, що розшифровується як Global-Local Outlier Score from Hierarchies (глобально-локальна оцінка викидів з ієрархій). На щастя, обчислення GLOSH точки даних не є надто складним,

але вимагає, щоб під час побудови ієрархії HDBSCAN* були зроблені деякі додаткові розрахунки. Раніше згадувалося, що значення ваги ребра, яке видаляється, використовується для позначення поточного рівня ієрархії. Це значення ваги, а також остання мітка кластера повинні бути записані для кожної точки, в момент, коли точка позначається як шум, або їй присвоюється мітка шуму, щоб використовувати ті ж самі формулювання, що і раніше. Точка даних x має значення ϵ та $\epsilon max,$ які є ваговим значенням точки безпосередньо перед тим, як її було позначено як шумову, та найнижчим поширеним рівнем зникнення нащадків з її останнього позначеного кластера, відповідно. Значення показника викиду для точки x можна отримати за допомогою формули 1.6.

$$GLOSH(x) = 1 - \frac{\epsilon_{max}(x)}{\epsilon(x)} \quad (1.6)$$

HDBSCAN - популярний метод групування об'єктів. Він створює групи таким чином, що об'єкти всередині групи схожі один на одного і відрізняються від об'єктів в інших групах. Кластери візуально представлені у вигляді ієрархічного дерева, яке називається дендрограмою.

HDBSCAN має кілька ключових переваг:

Немає необхідності попередньо визначати кількість кластерів. Замість цього дендрограму можна розрізати на відповідному рівні, щоб отримати потрібну кількість кластерів.

За допомогою дендрограми дані легко узагальнюються/організовуються в ієрархію. Дендрограми полегшують вивчення та інтерпретацію кластерів.

Існує два основних типи HDBSCAN:

Агломеративний: Спочатку кожен об'єкт розглядається як окремий кластер. Відповідно до певної процедури, кластери об'єднуються крок за кроком, поки не залишиться один кластер. Наприкінці процесу об'єднання кластерів утворюється кластер, що містить усі елементи.

Роздільний: Метод поділу є протилежним до агломеративного методу. Спочатку всі об'єкти розглядаються в одному кластері. Потім процес поділу виконується крок за кроком, поки кожен об'єкт не сформує окремий кластер. Процедура поділу або розбиття кластерів здійснюється відповідно до певних принципів, які максимізують відстань між сусідніми об'єктами в кластері.

Між агломеративною та роздільною кластеризацією, як правило, перевага надається агломеративній кластеризації.

HDBSCAN використовує міру відстані/подібності для створення нових кластерів. Кроки для агломеративної кластеризації можна узагальнити наступним чином:

Крок 1: Обчислення матриці близькості з використанням певної метрики відстані;

Крок 2: Кожній точці даних присвоюється кластер;

Крок 3: Об'єднання кластерів на основі метрики подібності між кластерами;

Крок 4: Оновлення матриці відстаней;

Крок 5: Треба повторювати кроки 3 і 4, поки не залишиться лише один кластер.

- Обчислення матриці близькості

Першим кроком алгоритму є створення матриці відстаней. Значення матриці обчислюються шляхом застосування функції відстані між кожною парою об'єктів. Для цієї операції зазвичай використовується евклідова функція відстані. Структура матриці близькості буде наступною для набору даних з n елементів. Тут $d(p_i, p_j)$ представляють значення відстаней між p_i та p_j.

Таблиця 1.2

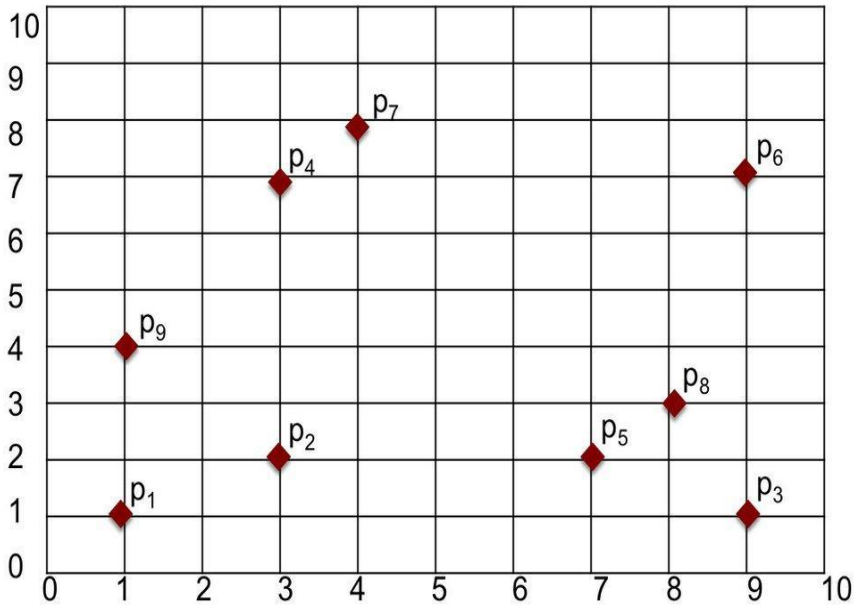
Обчислення матриці близькості

	p_1	p_2	p_3	...	p_n
p_1	$d(p_1,p_1)$	$d(p_1,p_2)$	$d(p_1,p_3)$	...	$d(p_1,p_n)$
p_2	$d(p_2,p_1)$	$d(p_2,p_2)$	$d(p_2,p_3)$	...	$d(p_2,p_n)$
p_3	$d(p_3,p_1)$	$d(p_3,p_2)$	$d(p_3,p_3)$	...	$d(p_3,p_n)$
...	...	...	...	...	...
p_n	$d(p_n,p_1)$	$d(p_n,p_2)$	$d(p_n,p_3)$	...	$d(p_n,p_n)$

- Подібність між кластерами

Основне питання в HDBSCAN полягає в тому, як обчислити відстань між кластерами та оновити матрицю близькості. Існує багато різних підходів для відповіді на це питання. Кожен підхід має свої переваги та недоліки. Вибір буде залежати від того, чи є шум у наборі даних, чи є форма кластерів круглою чи ні, а також від щільності точок даних.

Числовий приклад допоможе проілюструвати методи і вибір. Буде використана невелика вибірка даних, що містить лише дев'ять двовимірних точок, як показано рис 1.6.



Point	x	y
p_1	1	1
p_2	3	2
p_3	9	1
p_4	3	7
p_5	7	2
p_6	9	7
p_7	4	8
p_8	8	3
p_9	1	4

Рис. 1.6. Подібність між кластерами у результатах HDBSCAN

Нехай у вибірці даних є два кластери, як показано рис. 1.7. Існують різні підходи до обчислення відстані між кластерами. Нижче наведено найпопулярніші з них.

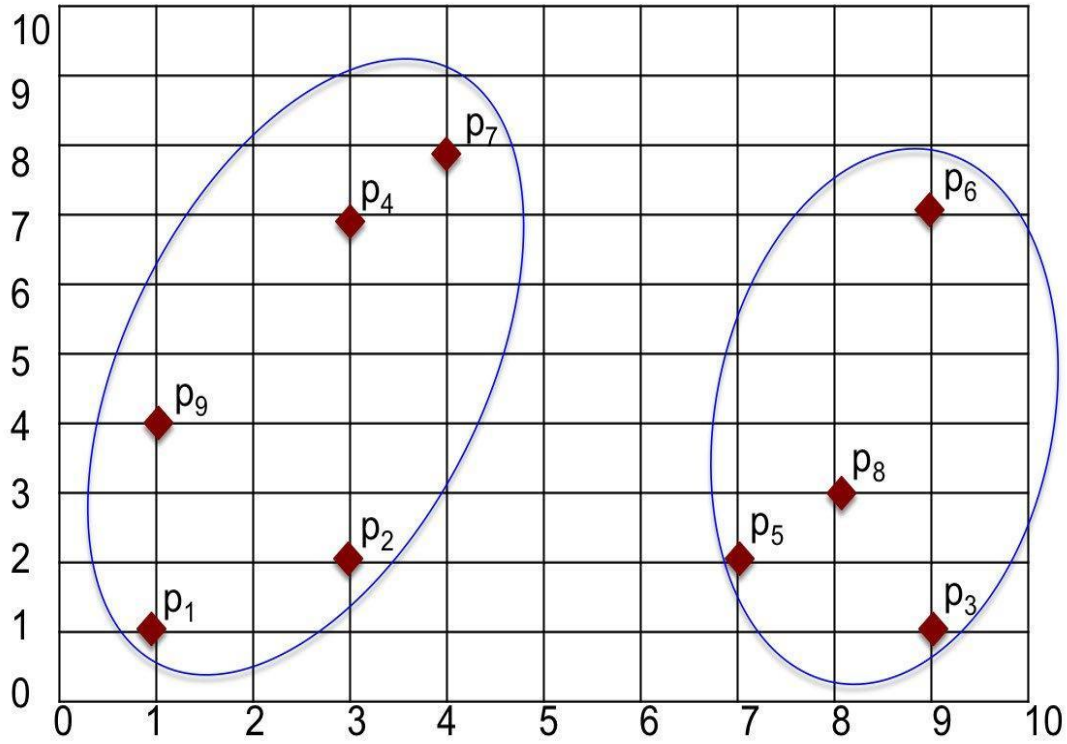


Рис. 1.7. Обчислення відстані між кластерами

- Мінімальний (одиничний) зв'язок (Min)

Один із способів виміряти відстань між кластерами - це знайти мінімальну відстань між точками в цих кластерах. Тобто, можна знайти точку в першому кластері, найближчу до точки в іншому кластері, і обчислити відстань між цими точками. На рис. 1.8 найближчими точками є $p2$ в одному кластері та $p5$ в іншому. Відстань між цими точками, а отже і відстань між кластерами, знаходиться як $d(p2, p5) = 4$

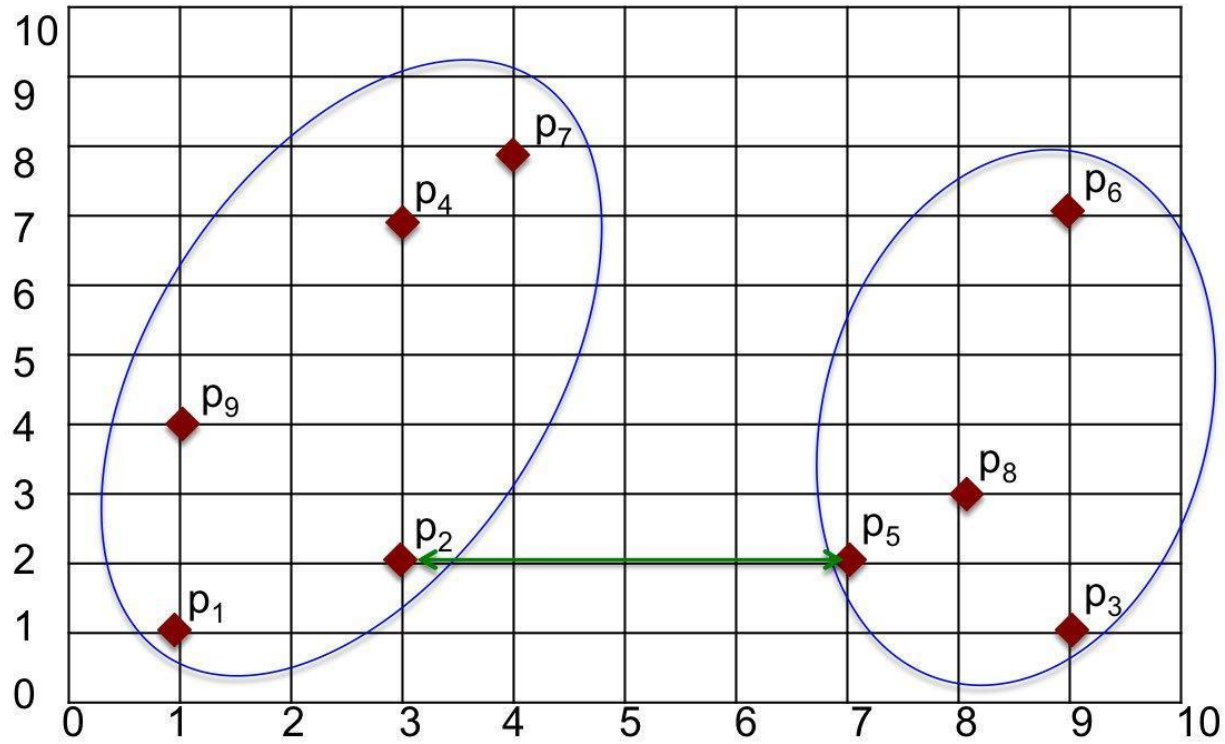


Рис 1.8. Пошук мінімальної відстані між кластерами

Перевага методу Min полягає в тому, що він може точно обробляти нееліптичні форми. Недоліком є чутливість до шуму та викидів.

- Максимальний (повний) зв'язок (Max)

Інший спосіб виміряти відстань - знайти максимальну відстань між точками в двох кластерах. Можна знайти точки в кожному кластері, які є найвіддаленішими одна від одної, і обчислити відстань між ними. На рисунку 1.9 максимальна відстань знаходиться між $p1$ та $p6$. Відстань між цими двома точками, а отже і відстань між кластерами, знаходиться як $d(p1, p6) = 10$.

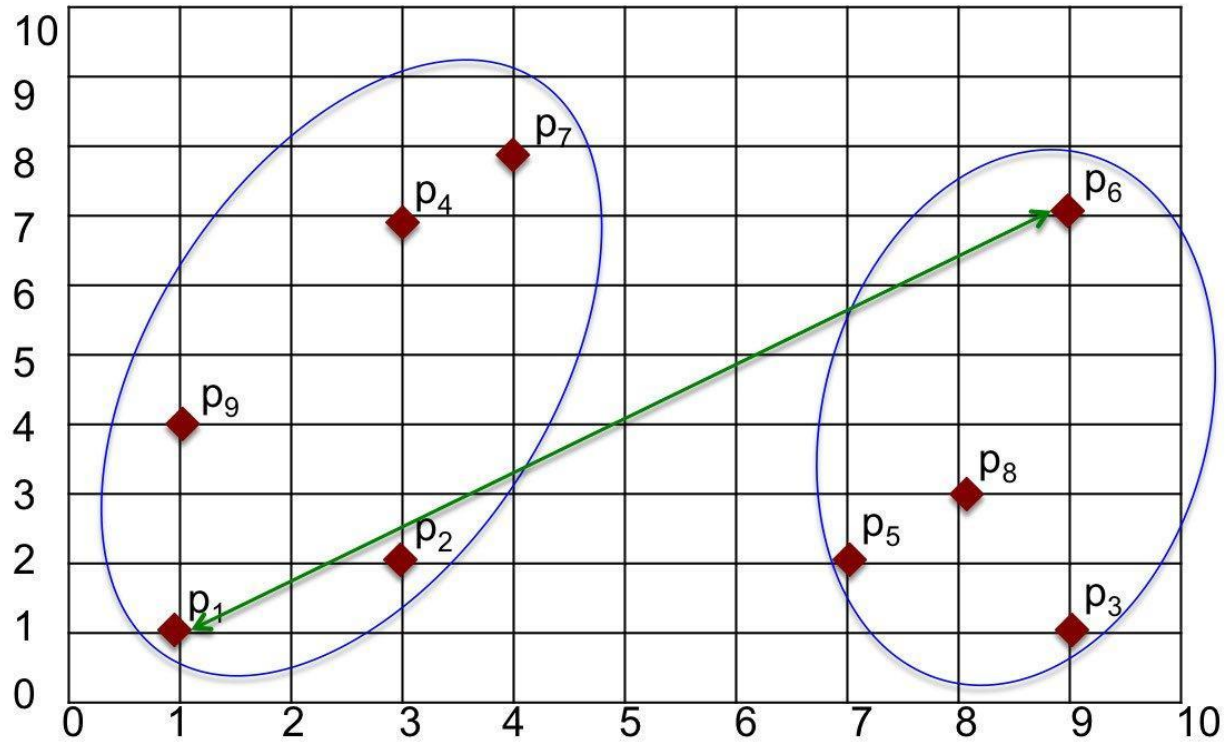


Рис 1.9. Пошук максимальної відстані між кластерами

Метод MAX менш чутливий до шуму та викидів порівняно з методом MIN. Однак MAX може розбивати великі кластери і має тенденцію до упередженості до глобулярних кластерів.

- Зв'язок з центроїдом

Метод центроїдів визначає відстань між кластерами як відстань між їхніми центрами/центроїдами[16]. Після обчислення центроїда для кожного кластера обчислюється відстань між цими центроїдами за допомогою функції відстані (рис. 10).

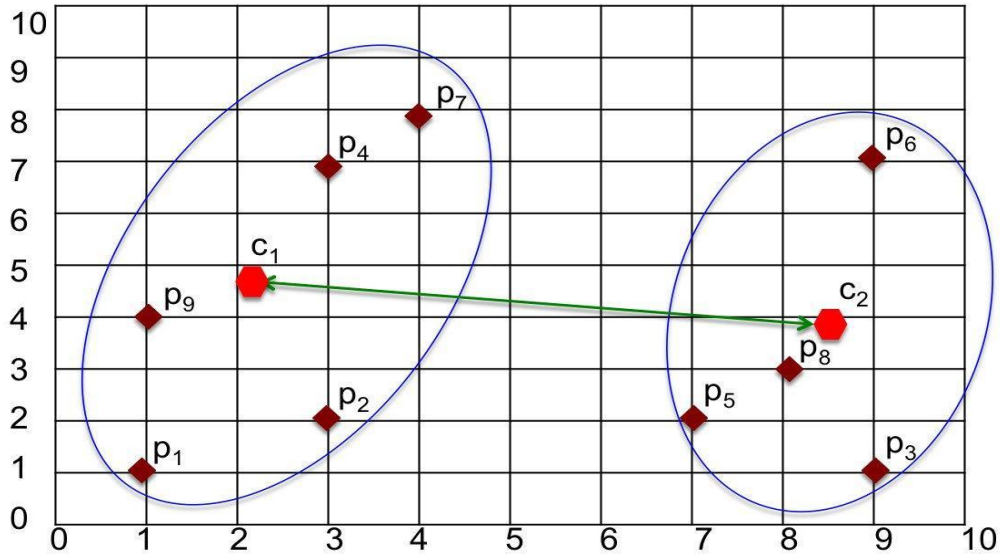


Рис 1.10. Пошук відстані між центроїдами кластерів

- Середній зв'язок

Метод Average визначає відстань між кластерами як середню попарну відстань між усіма парами точок у кластерах. Для простоти на рисунку 1.11 показано лише деякі лінії, що з'єднують пари точок.

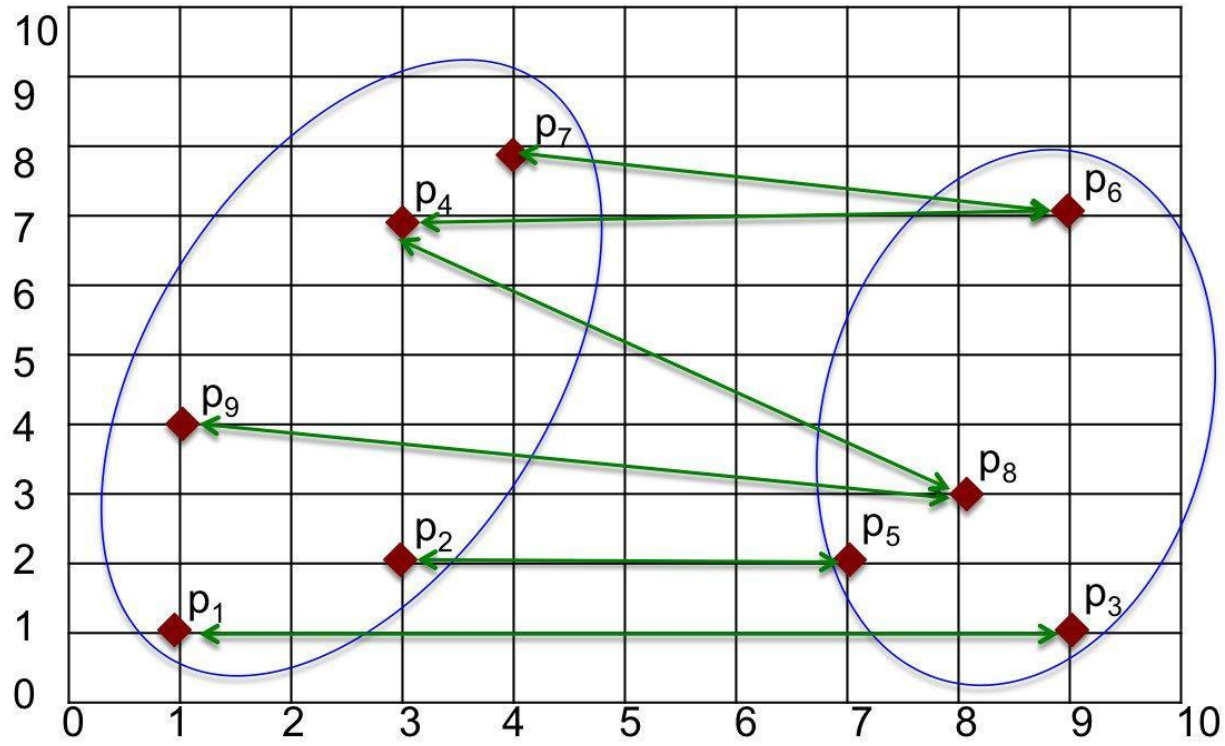


Рис 1.11. Метод Average для визначення відстані між кластерами

- Зв'язок Уорда

Підхід Уорда аналізує дисперсію кластерів, а не вимірює відстані безпосередньо, мінімізуючи дисперсію між кластерами.

За методом Уорда відстань між двома кластерами залежить від того, наскільки збільшиться значення суми квадратів при їх об'єднанні.

Іншими словами, метод Уорда намагається мінімізувати суму квадратів відстаней точок від центрів кластерів. Порівняно з описаними вище відстанями, метод Уорда менш чутливий до шуму та викидів. Тому методу Уорда надають перевагу при кластеризації більше, ніж іншим.

1.2.5. OPTICS

Кластеризація OPTICS розшифровується як "Впорядкування точок для визначення кластерної структури". Він є різновидом алгоритму кластеризації DBSCAN. Він додає ще два терміни до концепції кластеризації DBSCAN.

OPTICS - алгоритм на основі щільності, подібний до DBSCAN (Density-Based Spatial Clustering of Applications with Noise), але він може виділити кластери різної щільності та форми. Він корисний для виявлення кластерів різної щільності у великих наборах даних високої розмірності.

Основна ідея OPTICS в тому, щоб виділити кластерну структуру набору даних шляхом ідентифікації точок, пов'язаних з щільністю. Алгоритм будує представлення даних на основі щільності, створюючи впорядкований список точок, який називається діаграмою досяжності[30]. Кожна точка в списку пов'язана з відстанню досяжності, яка є мірою того, наскільки легко дістатися до цієї точки з інших точок набору даних. Точки з однаковою відстанню досяжності, швидше за все, знаходяться в одному кластері.

Алгоритм OPTICS виконує такі основні кроки:

- Визначити пороговий параметр щільності, Eps, який контролює мінімальну щільність кластерів.
- Для кожної точки в наборі даних обчислити відстань до її k найближчих сусідів.

- Починаючи з довільної точки, обчислюється відстань досяжності кожної точки набору даних, виходячи з щільності її сусідів.
- Впорядковуються точки за відстанню до них і створюється графік досяжності.
- Виділяються кластери з діаграми досяжності, згрупувавши точки, які знаходяться близько одна до одної і мають однакову відстань до них.

Однією з головних переваг OPTICS над DBSCAN є те, що він не вимагає заздалегідь задавати кількість кластерів, натомість він витягує кластерну структуру даних і будує графік досяжності. Це дозволяє користувачеві мати більшу гнучкість у виборі кількості кластерів, обрізаючи графік досяжності в певній точці.

Крім того, на відміну від інших алгоритмів кластеризації на основі щільності, таких як DBSCAN, він може працювати з кластерами різної щільності та форми, а також ідентифікувати ієрархічну структуру (рис. 1.12).

Це:

- **Відстань до ядра**: Це мінімальне значення радіуса, необхідне для того, щоб класифікувати задану точку як опорну. Якщо дана точка не є опорною, то відстань до неї не визначена.
- **Відстань досяжності**: Визначається відносно іншої точки даних q(Let). Відстань досяжності між точками p і q - це максимальне значення між основною відстанню p і евклідовою відстанню (або іншою метрикою відстані) між p і q. Треба зауважити, що відстань досяжності не визначена, якщо q не є основною точкою.

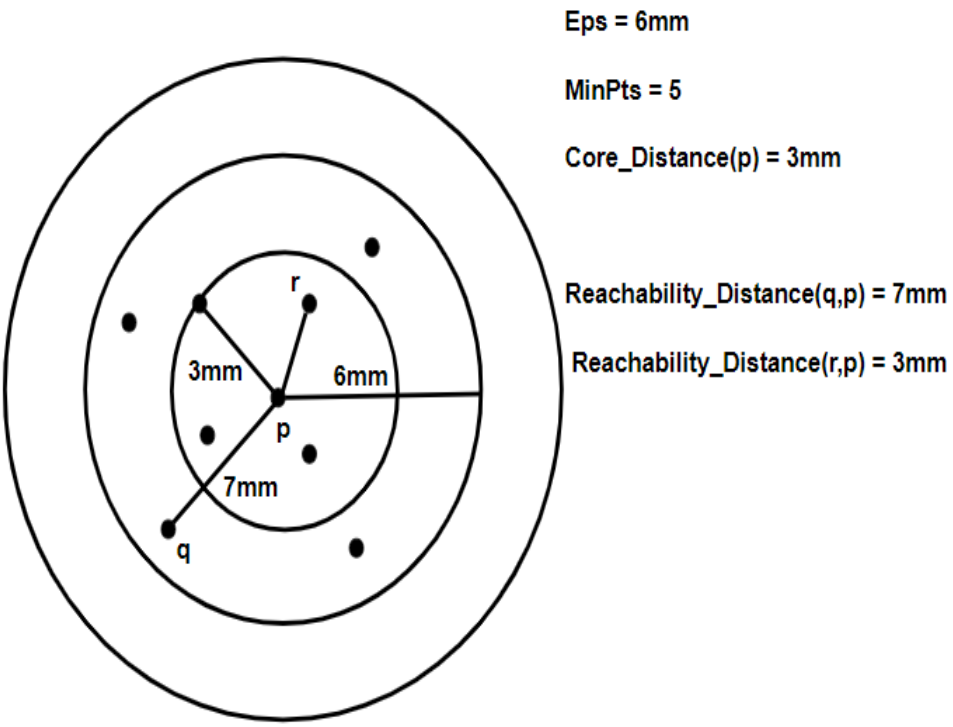


Рис 1.12. Визначення відстаней досяжності в алгоритмі OPTICS

Цей метод кластеризації[2] відрізняється від інших методів кластеризації в тому сенсі, що він не сегментує дані в явному вигляді на кластери. Замість цього він створює візуалізацію відстаней досяжності і використовує цю візуалізацію для кластеризації даних.

Якщо порівнювати алгоритм кластеризації OPTICS з DBSCAN то можна зробити наступні висновки:

- **Витрати пам'яті**: Метод кластеризації OPTICS вимагає більше пам'яті, оскільки він підтримує пріоритетну чергу (Min Heap) для визначення наступної точки даних, яка є найближчою до точки, що обробляється в даний момент, з точки зору відстані досяжності (Reachability Distance). Він також вимагає більшої обчислювальної потужності, оскільки запити до найближчого сусіда є складнішими, ніж радіусні запити в DBSCAN.
- **Менша кількість параметрів**: Метод кластеризації OPTICS не потребує збереження параметра. Це призводить до скорочення аналітичного процесу налаштування параметрів.

Ця методика не розділяє дані на кластери. Вона просто створює графік відстані досяжності, а інтерпретація точок у кластери залежить від інтерпретації програміста.

Робота з різною щільністю: Кластеризація DBSCAN може мати проблеми з обробкою наборів даних з різною щільністю, оскільки вона вимагає єдиного значення епсилон для визначення розміру околиці для всіх точок. На противагу цьому, OPTICS може працювати з різною щільністю, використовуючи концепцію відстані досяжності, яка адаптується до локальної щільності даних[23]. Це означає, що OPTICS може ідентифікувати кластери різних розмірів і форм ефективніше, ніж DBSCAN, у наборах даних з різною щільністю.

- Виділення кластерів: Хоча і OPTICS, і DBSCAN можуть ідентифікувати кластери, OPTICS створює графік відстані досяжності, який можна використовувати для виокремлення кластерів на різних рівнях деталізації. Це дозволяє гнучкіше проводити кластеризацію і може виявити кластери, які можуть бути неочевидними при фіксованому значенні епсилон в DBSCAN. Однак, це також вимагає більше ручної інтерпретації та прийняття рішень з боку програміста.
- Обробка шумів: DBSCAN явно розрізняє основні точки, граничні точки і точки шуму, в той час як OPTICS не ідентифікує точки шуму явно. Замість цього, точки з великою відстанню досяжності можна розглядати як потенційні точки шуму. Однак це також означає, що OPTICS може бути менш ефективним у виявленні невеликих кластерів, оточених шумовими точками, оскільки ці кластери можуть бути об'єднані з шумовими точками на графіку відстані досяжності.
- Складність виконання: Складність виконання OPTICS, як правило, вища, ніж у DBSCAN, через використання черги пріоритетів для підтримки відстаней досяжності. Однак,

нещодавні дослідження запропонували оптимізацію для зменшення обчислювальної складності OPTICS, що робить його більш масштабованим для великих наборів даних.

1.2.6. Спектральна кластеризація

Спектральна кластеризація - це метод інтелектуального аналізу даних, який зводить складні багатовимірні набори даних до кластерів подібних даних у більш рідкісних вимірах. Основна ідея полягає в тому, щоб об'єднати весь спектр неорганізованих точок даних у кілька груп на основі їхньої унікальності "Спектральна кластеризація є однією з найпопулярніших форм багатовимірного статистичного аналізу “Спектральна кластеризація використовує підхід до кластеризації на основі зв'язності”, де на графіку визначаються спільноти вузлів (тобто точок даних), які з'єднані або знаходяться безпосередньо поруч один з одним. Потім вузли відображаються в низьковимірний простір, який можна легко розділити, щоб сформувати кластери. Спектральна кластеризація використовує інформацію з власних значень (спектру) спеціальних матриць (наприклад, матриці спорідненості, матриці ступенів і матриці лапласа), отриманих з графа або набору даних.

Методи спектральної кластеризації привабливі, прості в реалізації, досить швидкі, особливо для розріджених наборів даних до декількох тисяч. Спектральна кластеризація[3] розглядає кластеризацію даних як задачу розбиття графа, не роблячи жодних припущень щодо форми кластерів даних.

Відмінність спектральної кластеризації від звичайних методів кластеризації

Спектральна кластеризація є гнучкою і дозволяє кластеризувати неграфічні дані. Вона не робить жодних припущень щодо форми кластерів. Методи кластеризації, такі як K-means, припускають, що точки, віднесені до кластера, мають сферичну форму відносно центру кластера. Це припущення не завжди може бути релевантним. У таких випадках спектральна кластеризація допомагає створити більш точні кластери. Вона може

правильно кластеризувати спостереження, які насправді належать до одного кластера, але знаходяться далі, ніж спостереження в інших кластерах, завдяки зменшенню розмірності.

Точки даних у спектральній кластеризації повинні бути пов'язані, але не обов'язково мати опуклі межі, на відміну від звичайних методів кластеризації, де кластеризація ґрунтується на компактності точок даних. Хоча для великих наборів даних цей метод є обчислювально коштовним, оскільки потрібно обчислювати власні значення та власні вектори, а кластеризація виконується на цих векторах. Крім того, для великих наборів даних складність збільшується, а точність значно знижується.

- Графи подібності

Маючи набір точок даних $x1, \dots xn$ і деяке поняття подібності $sij \geq 0$ між усіма парами точок даних x_i і x_j, інтуїтивно зрозумілою метою кластеризації є розбиття точок даних на кілька груп таким чином, щоб точки в одній групі були схожими, а точки в різних групах відрізнялися одна від одної. Якщо ми не маємо більше інформації, ніж схожість між точками даних, то гарним способом представлення даних є граф схожості $G = (V, E)$. Кожна вершина v_i на цьому графі представляє точку даних x_i. Дві вершини з'єднуються, якщо подібність s_{ij} між відповідними точками даних хі та x_j є додатною або більшою за певний поріг, а ребро має вагу s_{ij} . Тепер проблему кластеризації можна переформулювати за допомогою графа подібності: ми хочемо знайти таке розбиття графа, щоб ребра між різними групами мали дуже низьку вагу (що означає, що точки в різних кластерах відрізняються одна від одної), а ребра всередині групи мали високу вагу (що означає, що точки в одному кластері схожі одна на одну).

- Різні графи подібності

Існує декілька популярних способів перетворення заданої множини $x1, \ldots, xn$ точок даних з попарною схожістю s_{ij} або попарною відстанню d_{ij} у граф. При побудові графів подібності метою є моделювання локальних сусідських відносин між точками даних.

Граф ε-сусідства: В ньому поєднуються всі точки, парні відстані між якими менші за ε. Оскільки відстані між усіма з'єднаними точками приблизно однакового масштабу (не більше ε), зважування ребер не додасть графу більше інформації про дані. Тому граф ε-сусідства зазвичай розглядається як незважений граф.

На рисунку нижче точки, що знаходяться на відстані, меншій за $\varepsilon = 0{,}28$, з'єднані ребрами. Він показує, що середні точки на кожному кластері сильно пов'язані між собою, тоді як точки на крайніх кластерах менш зв'язані. Взагалі, важко визначити параметр ε, який є надійним, особливо коли точки розподілені з різними відстанями між ними залежно від простору. Якщо ми виберемо хороший параметр ε, то на виході алгоритму отримаємо чітко окреслені кластери. Як показано на рисунку 1.13, дані згруповані у вісім кластерів, позначених різними кольорами.

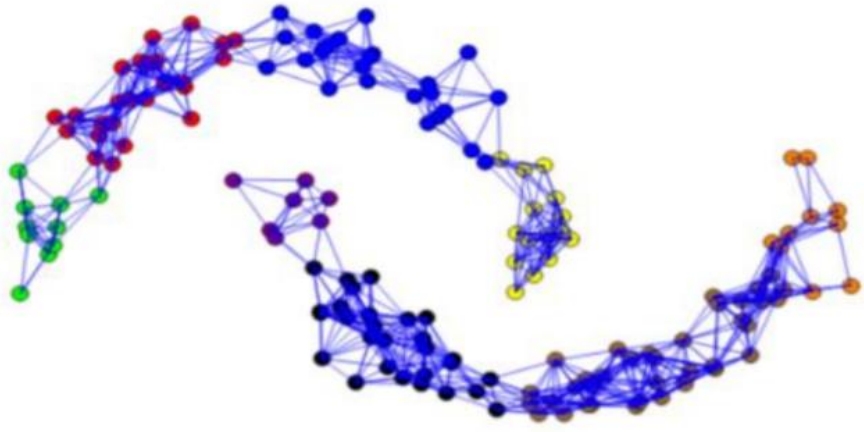


Рис 1.13. Граф сусідства алгоритму OPTICS

Граф k-найближчих сусідів: Тут мета полягає в тому, щоб з'єднати вершину v_i з вершиною v_j, якщо v_jє серед k найближчих сусідів v_i. Однак,

це визначення призводить до орієнтованого графа, оскільки відношення сусідства не є симетричним. Існує два способи зробити цей граф неорієнтованим. Перший спосіб - просто ігнорувати напрямок ребер, тобто з'єднати v_i та v_j неорієнтованим ребром, якщо v_i є одним з k найближчих сусідів v_j або якщо v_j є одним з k найближчих сусідів v_i. Отриманий граф є тим, що зазвичай називають графом k-найближчих сусідів. Другий варіант - з'єднати вершини v_i і v_j, якщо і v_i є серед k-найближчих сусідів v_j, і v_j є серед k-найближчих сусідів vi. Отриманий граф називається графом взаємних k-найближчих сусідів. В обох випадках після з'єднання відповідних вершин ми зважуємо ребра за схожістю їхніх кінцевих точок (рис. 1.14).

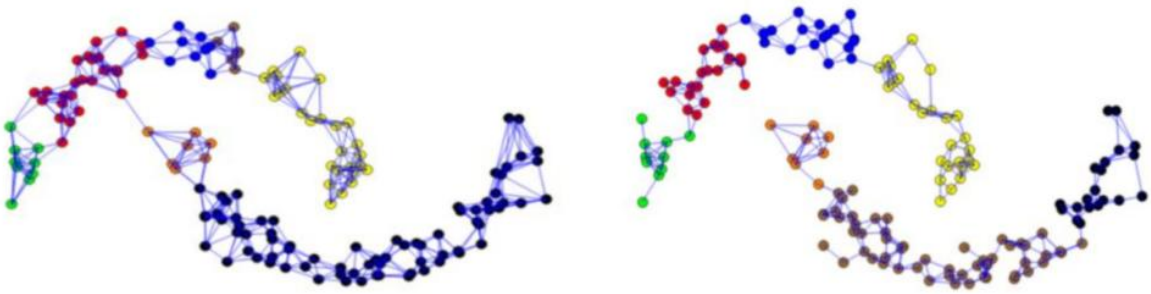


Рис 1.14. Граф k-найближчих сусідів алгоритму OPTICS

Повністю зв'язний граф: Тут ми просто з'єднуємо всі точки з позитивною схожістю між собою, а всі ребра зважуємо за допомогою s_{ij} . Оскільки граф має відображати локальні сусідські зв'язки, ця конструкція є корисною лише тоді, коли сама функція подібності моделює локальні сусідства. Прикладом такої функції подібності є гаусова функція подібності

$$s(x_i, x_j) = exp\,(-(x_i - x_j\ (2/(2\sigma 2))), \qquad (1.7)$$

де параметр σ контролює ширину околиць. Цей параметр відіграє таку ж роль, як і параметр ε у випадку графа ε-сусідів.

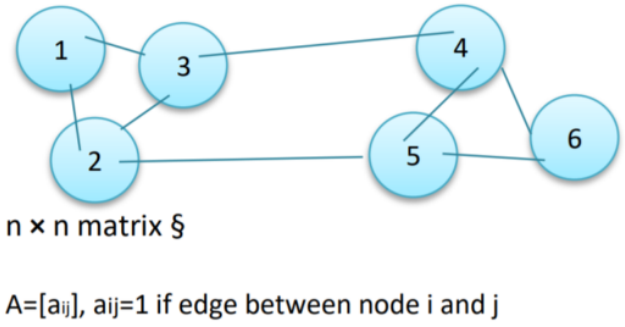
Матричне представлення спектральної кластеризації

- Матриця суміжності та спорідненості (A)

Граф (або набір точок даних) можна представити у вигляді матриці суміжності, де індекси рядків і стовпців представляють вузли, а записи -

відсутність або наявність ребра між вузлами (тобто, якщо запис у рядку 0 і стовпці 1 дорівнює 1, це означає, що вузол 0 з'єднаний з вузлом 1).

Матриця спорідненості схожа на матрицю суміжності, за винятком того, що значення для пари точок виражає, наскільки ці точки схожі одна на одну. Якщо пари точок дуже відрізняються, то афінність дорівнює 0. Якщо точки ідентичні, то афінність може дорівнювати 1. Таким чином, спорідненість діє як вага ребер на нашому графі (рис.1.15).

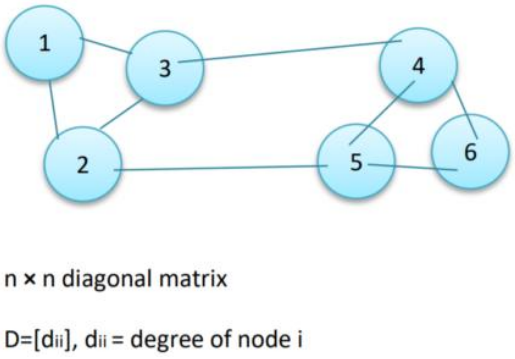


	1	2	3	4	5	6
1	0	1	0	0	1	1
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	1	0	1	0	1	1
5	0	1	0	1	0	1
6	0	1	1	0	1	0

Рис 1.15. Матриця суміжності та спорідненості (A)

- Степенева матриця (D)

Матриця степенів - це діагональна матриця, де степінь вузла (тобто значення) діагоналі задається кількістю ребер, з'єднаних з ним. Ми також можемо отримати степінь вершин, взявши суму кожного рядка в матриці суміжності (рис.1.16).

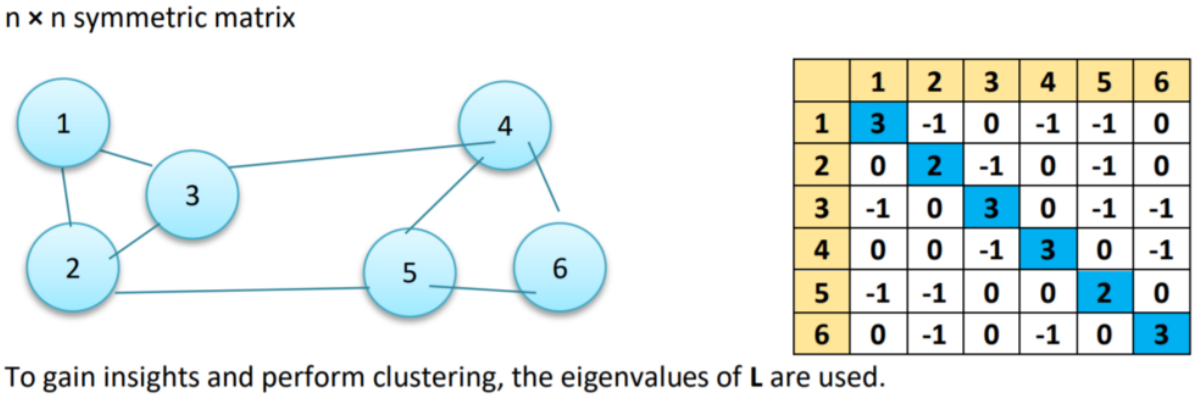


	1	2	3	4	5	6
1	3	0	0	0	0	0
2	0	2	0	0	0	0
3	0	0	3	0	0	0
4	0	0	0	3	0	0
5	0	0	0	0	2	0
6	0	0	0	0	0	3

Рис 1.16. Степенева матриця (D)

- Матриця Лапласа (L)

Це ще одне представлення графа/точок даних, яке має гарні властивості, що використовуються спектральною кластеризацією. Одне з таких представлень отримується шляхом віднімання матриці суміжності від матриці степенів (рис. 1.17) (тобто $L = D - A$) [9].



	1	2	3	4	5	6
1	3	-1	0	-1	-1	0
2	0	2	-1	0	-1	0
3	-1	0	3	0	-1	-1
4	0	0	-1	3	0	-1
5	-1	-1	0	0	2	0
6	0	-1	0	-1	0	3

Рис 1.17. Матриця Лапласа (L)

Спектральна щільність: Перше значення яке не дорівнює нулю називається спектральною щільністю. Спектральна щільність дає нам деяке уявлення про щільність графа.

Значення Фідлера: Друге значення має назву значення Фідлера, а відповідно його вектор називають - вектором Фідлера. Кожне значення у векторі Фідлера дає нам інформацію про те, до якої сторони границі рішення належить певна вершина.

Використовуючи L, ми знаходимо перший великий розрив між власними значеннями, який зазвичай вказує на те, що кількість власних значень до цього розриву дорівнює кількості кластерів.

- Лапласіани графів

  Ненормований лапласіан графа $L = D - W$

  Нормалізовані лапласіани графів - Існують дві матриці, які в літературі називають нормалізованими лапласіанами графів. Обидві матриці тісно пов'язані між собою і визначаються як

$$Lsym := D - \frac{1}{2} * LD - \frac{1}{2} = I - D - \frac{1}{2} * WD - \frac{1}{2} \quad (1.8)$$

$$Lrw := D - 1L = I - D - 1W \quad (1.9)$$

Позначимо першу матрицю через $Lsym$, оскільки вона є симетричною, а другу через Lrw, оскільки вона тісно пов'язана з власними векторами та власними значеннями випадкового блукання.

.Для матриці A, якщо існує вектор x, який не містить всіх 0, і скаляр λ такий, що $Ax = \lambda x$, то кажуть, що x є власним вектором A з відповідним власним значенням λ. Можливо розглядати матрицю A як функцію, яка відображає вектори на нові вектори. Власні вектори матриці A - це особливі вектори, які не змінюють свого напрямку при множенні на матрицю. Це означає, що якщо провести пряму через початок координат і власний вектор, то після множення на матрицю A власний вектор все одно потрапить на цю пряму. Власні вектори є важливими в лінійній алгебрі, оскільки вони допомагають описувати динаміку систем, представлених матрицями.

"Власні значення є невід'ємними дійсними числами, а власні вектори є дійсними та ортогональними".

Алгоритм спектральної кластеризації:

1. Побудова матриці схожості (Affinity Matrix):
   - Визначення міри схожості між парами даних. Це може бути евклідова відстань, гаусівське ядро та інше.
   - Матриця схожості є квадратною матрицею розміром NxN, де N - кількість даних.
2. Побудова матриці Лапласіана (Laplacian Matrix):
   - Обчислення ступенів вершин (діагональна матриця ступенів), де кожен елемент на діагоналі - сума ваг ребер, що пов'язані з відповідною вершиною.
   - Віднімання матриці схожості від матриці ступенів, щоб отримати матрицю Лапласіана.
3. Обчислення власних значень і векторів матриці Лапласіана:

- Розв'язання задачі знаходження власних значень і векторів для матриці Лапласіана. Зазвичай використовується метод розкладу на власні вектори чи метод степеневої ітерації.

4. Вибір к-власних векторів:
    - Вибір перших k власних векторів, що відповідають найменшим k власним значенням. Значення k вибирається в залежності від кількості кластерів, які потрібно виділити.
5. Створення нового простору ознак:
    - Сформувати нову матрицю даних, використовуючи вибрані власні вектори як нові ознаки (зазвичай k стовбців).
6. Кластеризація в новому просторі ознак:
    - Застосування обраного алгоритму кластеризації (наприклад, k-середні) до даних в новому просторі ознак для остаточного формування кластерів.

1.2.7. Mean-Shift Clustering

Meanshift підпадає під категорію алгоритму кластеризації, на відміну від навчання без учителя, який розподіляє точки даних по кластерах ітеративно, зміщуючи точки до моди (мода - це найвища щільність точок даних в регіоні, в контексті Meanshift). Таким чином, він також відомий як алгоритм пошуку моди. Meanshift знаходить застосування в області обробки зображень і комп'ютерного зору.

Маючи набір точок даних, алгоритм ітеративно призначає кожній точці даних напрямок до найближчого центроїда кластера, а напрямок до найближчого центроїда кластера визначається тим, де знаходиться більшість точок, розташованих поруч[29]. Таким чином, з кожною ітерацією кожна точка даних переміщується ближче до того місця, де знаходиться більшість точок, що веде або буде вести до центру кластера. Коли алгоритм зупиняється, кожна точка відноситься до кластера.

На відміну від популярного кластерного алгоритму K-Means, Meanshift не вимагає попереднього визначення кількості кластерів. Кількість кластерів визначається алгоритмом відповідно до даних.

Кластеризація за Meanshift[4] - це непараметричний алгоритм кластеризації на основі щільності, який можна використовувати для виявлення кластерів у наборі даних. Він особливо корисний для наборів даних, де кластери мають довільну форму і погано розділені лінійними межами.

Основна ідея кластеризації зі Meanshift полягає в тому, щоб змістити кожну точку даних у бік моди (тобто найвищої щільності) розподілу точок у певному радіусі. Алгоритм ітеративно виконує ці зсуви, поки точки не сходяться до локального максимуму функції щільності. Ці локальні максимуми представляють кластери в даних.

Процес роботи алгоритму кластеризації Meanshift можна підсумувати наступним чином:

Потрібно ініціалізувати точки даних як центроїди кластерів.

Треба повторювати наступні кроки до збіжності або до досягнення максимальної кількості ітерацій:

1. Для кожної точки даних обчисліть середнє значення всіх точок в межах певного радіусу (тобто "ядра") з центром у точці даних.
2. Треба зсунути точку даних до середнього значення.
3. Визначити центроїди кластерів як точки, що не змістилися після збіжності.
4. Далі повернути остаточні центроїди кластерів і розподіл точок даних за кластерами.

Однією з головних переваг кластеризації Meanshift є те, що вона не вимагає попереднього визначення кількості кластерів. Вона також не робить жодних припущень щодо розподілу даних і може працювати з довільними формами та розмірами кластерів. Однак він може бути чутливим до вибору ядра і радіуса ядра.

Кластеризація Meanshift може застосовуватися до різних типів даних, включаючи обробку зображень і відео, відстеження об'єктів і біоінформатику.

Оцінка щільності ядра

Першим кроком при застосуванні алгоритмів кластеризації Meanshift є математичне представлення даних, тобто представлення даних у вигляді точок, як показано на рисунку 1.18.

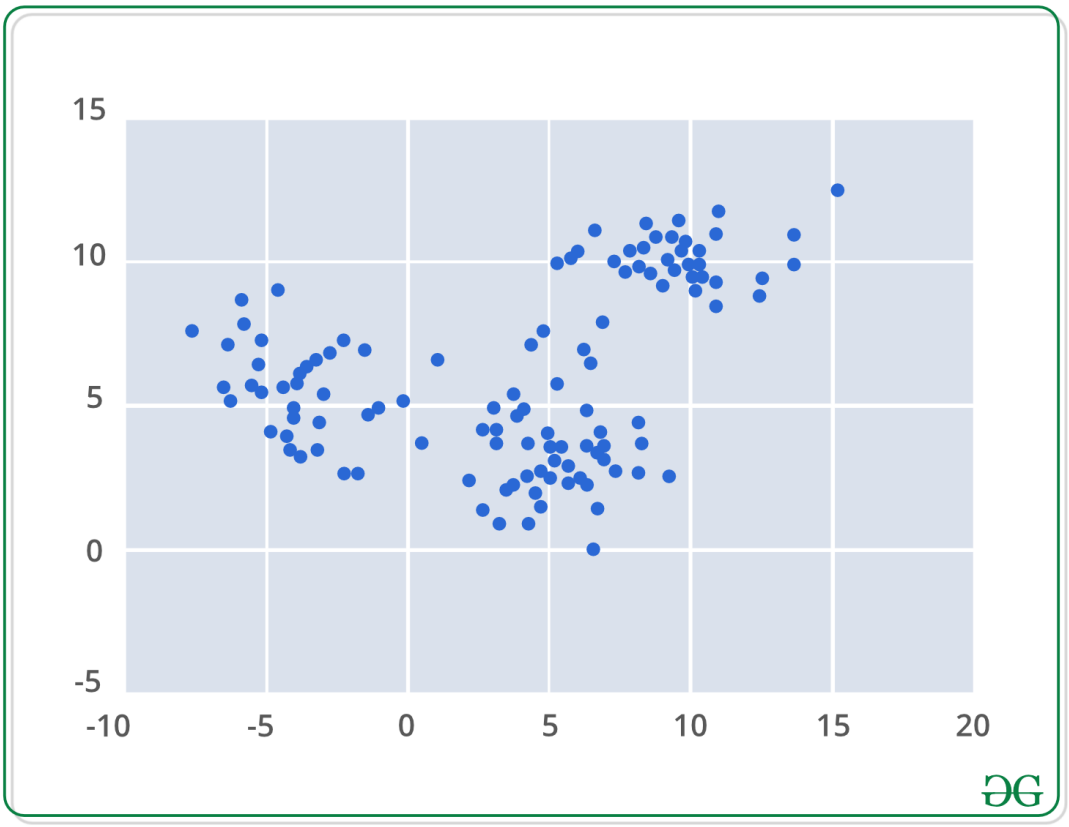
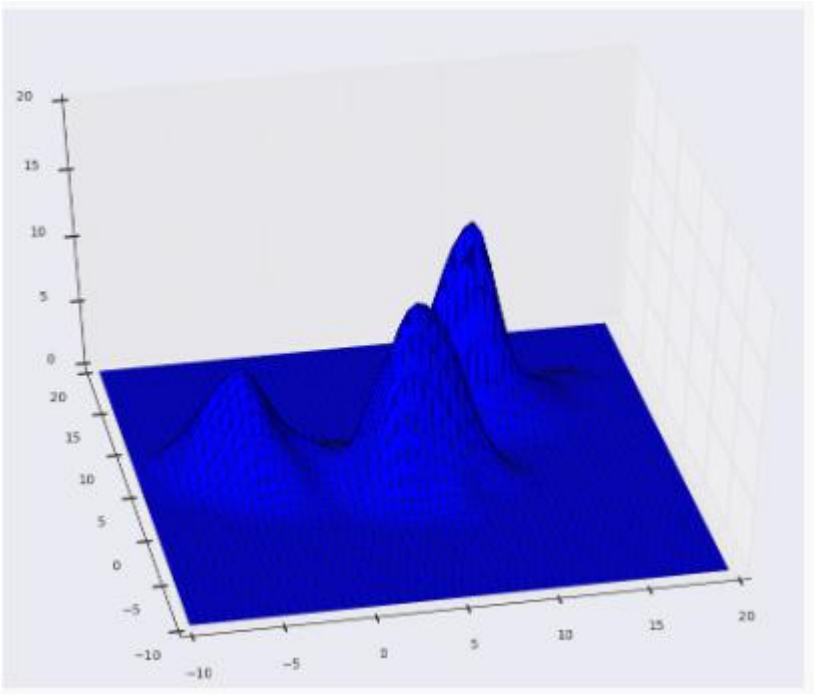


Рис 1.18. представлення даних у вигляді точок алгоритму Mean Shift

Meanshift ґрунтується на концепції оцінки щільності ядра, скорочено KDE. Можна уявити, що наведені вище дані було взято з розподілу ймовірностей. KDE - це метод оцінки основного розподілу, який також називають функцією щільності ймовірності для набору даних. Він працює шляхом розміщення ядра у кожній точці набору даних. Ядро - це вигадана математична назва вагової функції, яку зазвичай використовують при згортуванні. Існує багато різних типів ядер, але найпопулярнішим є ядро Гауса. Додавання всіх окремих ядер генерує функцію щільності поверхні

ймовірностей, як показано на рисунку 1.19. Залежно від параметра пропускної здатності ядра, який використовується, результуюча функція густини буде різною. Нижче наведено поверхню KDE для наведених вище точок (рис 1.20) за допомогою гауссового ядра з шириною смуги пропускання ядра 2.



Поверхнева ділянка Рис 1.19

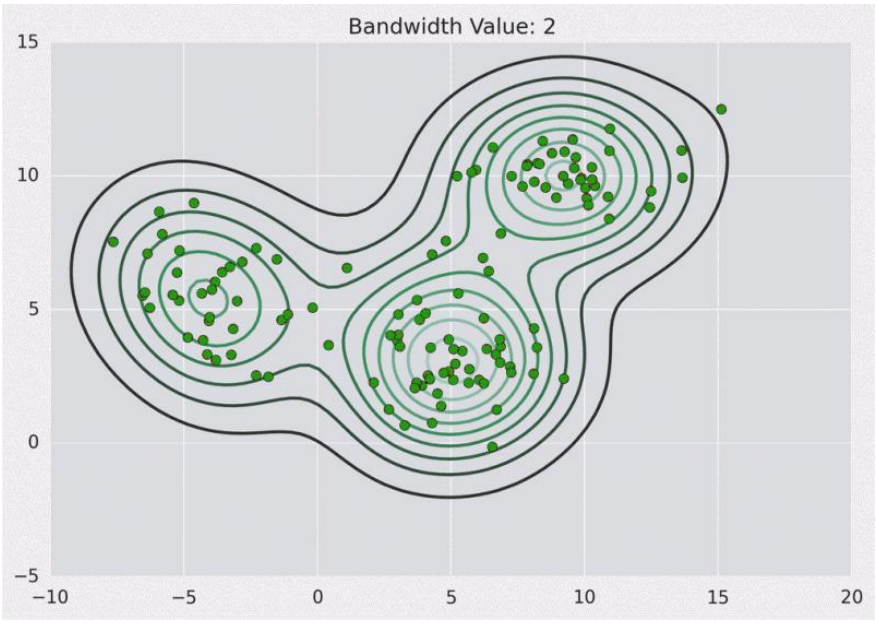


Рис 1.20. Контурний малюнок

Плюси:

- Знаходить змінну кількість мод

- Стійкий до викидів
- Загальний, незалежний від програми інструмент

Без моделей, не приймає жодної попередньої форми, наприклад, сферичної, еліптичної і т.д. на кластерах даних

Лише один параметр (розмір вікна h), де h має фізичне значення (на відміну від k-середніх)

Мінуси:

- Вихід залежить від розміру вікна
- Вибір розміру вікна (пропускної здатності) не є тривіальним
- Обчислювально (відносно) дорогий (приблизно 2 с/зображення)
- Погано масштабується з розмірністю простору ознак.

1.3. Висновок до розділу

В цьому розділі були проаналізовані класичні методи кластеризації, що мають стати основою методу визначення об’єктів на зображенні. Переважно були проаналізовані ті методи кластеризації, які не потребують попереднього задання кількості кластерів, оскільки не завжди можна визначити точно, скільки об’єктів є на зображенні. Виняток становлять k-means та спектральна кластеризація. Очевидно, що деякі алгоритми мають працювати швидше за інші, але не ясно, наскільки вони будуть точними і чи достатньо такої їхньої точності для визначення об’єктів. Тож, фактично, необхідно дослідити теоретичну та практичну швидкодію алгоритмів в умовах обмеженої продуктивності та визначити, який або які з них краще використовувати для розв’язання цієї задачі.

РОЗДІЛ 2
МЕТРИКИ ДЛЯ ВИЗНАЧЕННЯ ЯКОСТІ КЛАСТЕРИЗАЦІЇ ТА ОЦІНКА ПРОСТОРОВОЇ І ЧАСОВОЇ СКЛАДНОСТІ ОПИСАНИХ АЛГОРИТМІВ

2.1. Результати кластеризації та їхня оцінка

Результатами кластеризації для алгоритмів із заздалегідь визначеною кількістю кластерів є розподіл точок на саме цю кількість кластерів, а для алгоритмів без попереднього визначення кількості кластерів – точки згруповані у знайдені у даних кластери. При цьому треба враховувати що вартість обчислень для різних методів може достатньо сильно відрізнятися тож на певному етапі можливо доведеться шукати компроміс між точністю та вартістю обчислень хоча повторні спроби кластеризації мають уточнити отриманні результати тож для розуміння можливостей компромісу треба чітко уявляти характеристики за якими можна визначити точність а також межі часу для роботи кожного з алгоритмів. Характеристики за якими визначається точність кластеризації називаються метрики кластеризації.

2.2. Показники якості кластеризації

Якщо всі об'єкти даних у кластері дуже схожі, то кластер має високу якість. У більшості ситуацій можливо виміряти якість кластеризації, використовуючи метрику розбіжності/подібності[17]. Але є й інші методи для вимірювання якостей хорошої кластеризації, якщо кластери схожі між собою.

Метрика відмінності/подібності: Подібність між кластерами може бути виражена через функцію відстані, яка представлена d(i, j). Функції відстані відрізняються для різних типів даних і змінних даних. Міра функції

відстані відрізняється для неперервних змінних, категоріальних змінних та векторних змінних.

Евклідова відстань:

$$d(x,y) = \sqrt{(x-y)`(x-y)} \quad (2.1)$$

Статистична відстань:

$$d(x,y) = \sqrt{(x-y)`A(x-y)} \quad (2.2)$$

Звичайно, A = S -1, де S містить вибіркові дисперсії та коваріації. Однак без попереднього знання про окремі групи ці вибіркові величини неможливо обчислити. З цієї

причини для кластеризації часто надають перевагу евклідовій відстані. Метрика Мінковського:

$$d(x,y) = (\sum_{i=1}^{p} |x_i - y_i|^m)^{1/m} \quad (2.3)$$

Для m = 1, d(x, y) вимірює відстань між двома точками у p вимірах.

Для m = 2, d(x, y) стає евклідовою відстанню. Загалом, зміна m змінює вагу, що надається у бік збільшення або зменшення різниці.

Повнота кластера: Повнота кластера є важливим параметром для хорошої кластеризації, якщо будь-які два об'єкти даних мають схожі характеристики, то вони будуть віднесені до однієї категорії кластера відповідно до базової істини. Повнота кластера є високою, якщо об'єкти належать до однієї категорії.

Розглянемо кластеризацію C1, яка містить підкластери s1 та s2, де члени кластерів s1 та s2 належать до однієї категорії згідно з базовою істиною. Розглянемо іншу кластеризацію C2, яка ідентична C1, але тепер s1 і s2 об'єднані в один кластер. Тоді буде визначена міра якості кластеризації, Q, і відповідно до повноти кластера C2 матиме більшу якість кластера порівняно з C1, тобто Q(C2, Cg) > Q(C1, Cg).

Метод «Ганчір'яного мішка»: У деяких ситуаціях може бути декілька категорій, в яких об'єкти цих категорій не можуть бути об'єднані з іншими об'єктами. Тоді якість цих кластерних категорій вимірюється методом

ганчір'яного мішка. Відповідно до методу ганчір'яного мішка[22], потрібно помістити гетерогенний об'єкт в категорію ганчір'яного мішка.

Розглянемо кластеризацію C1 і кластер C ∈ C1 так, що всі об'єкти в C належать до однієї категорії кластера C1, окрім об'єкта o, згідно з базовою істиною. Розглянемо кластеризацію C2, яка ідентична C1, за винятком того, що об'єкт o віднесено до кластера D, який містить об'єкти різних категорій. Відповідно до базової істини, ця ситуація є зашумленою, і якість кластеризації вимірюється за допомогою критерію ганчіркового мішка. ми визначаємо міру якості кластеризації, Q, і відповідно до критеріїв методу ганчіркового мішка, кластер C2 матиме більшу якість кластеризації порівняно з кластером C1, тобто Q(C2, Cg)>Q(C1, Cg).

Збереження малих кластерів: Якщо невелика категорія кластеризації розбивається на дрібні частини, то ці дрібні частини кластера стають шумом для всієї кластеризації і, таким чином, стає важко ідентифікувати цю невелику категорію з кластеризації. Критерій збереження малих кластерів стверджує, що поділ невеликої категорії на частини не є доцільним і ще більше знижує якість кластерів, оскільки частини кластерів є відмінними один від одного. Припустимо, що кластеризація C1 розпалася на три кластери, C11 = {d1, . . . , dn}, C12 = {dn+1} і C13 = {dn+2}.

Нехай кластеризація C2 також розбивається на три кластери, а саме C1 = {d1, . . . , dn-1}, C2 = {dn} і C3 = {dn+1,dn+2}. Оскільки C1 розбиває малу категорію об'єктів, а C2 - велику категорію, якій надається перевага згідно з правилом, згаданим вище, то міра якості кластеризації Q повинна давати вищу оцінку C2, тобто Q(C2, Cg) > Q(C1, Cg).

Силует (Silhouette Score): Це метрика, яка оцінює, наскільки об'єкти всередині одного кластера ближчі один до одного, ніж до об'єктів у сусідніх кластерах. Коефіцієнт силуету або оцінка силуету[20] - це метрика, яка використовується для обчислення ефективності метода кластеризації. Коефіцієнт силуету або оцінка силуету - це метрика, яка використовується

для обчислення якості методу кластеризації. Його значення коливається від -1 до 1.

1: Означає, що кластери добре відокремлені один від одного і чітко розрізняються.

0: Означає, що кластери індиферентні, або можна сказати, що відстань між кластерами не є значущою.

-1: Кластери розподілені неправильно.

$$\text{Оцінка силуету} = (b - a)/max(a, b) \quad (2.4)$$

Де:

a= середня внутрішньокластерна відстань, тобто середня відстань між кожною точкою всередині кластера[25];

b= середня міжкластерна відстань, тобто середня відстань між усіма кластерами.

Індекс Девіса-Болдуіна (Davies-Bouldin Index): Це метрика, що вимірює середню схожість кожного кластера з його найбільш схожим кластером (сусіднім кластером). Менші значення індексу свідчать про кращу кластеризацію.

Індекс силового поля (Dunn Index): Індекс Данна (DI) (введений Дж. К. Данном у 1974 році), метрика для оцінки алгоритмів кластеризації, є внутрішньою схемою оцінки, де результат ґрунтується на самих кластеризованих даних. Як і всі інші подібні індекси, індекс Данна має на меті визначити набори кластерів, які є компактними, з невеликою дисперсією між членами кластера, і добре відокремленими, де середні значення різних кластерів досить далекі один від одного, порівняно з дисперсією всередині кластера.

Чим вище значення індексу Данна, тим кращою є кластеризація. Кількість кластерів, яка максимізує індекс Данна, приймається за оптимальну кількість кластерів k. Вона також має певні недоліки. Зі збільшенням кількості кластерів та розмірності даних зростають і обчислювальні витрати.

Індекс Данна для c кількості кластерів визначається як :

$$Dunn\ Index(U) = min_{1\le i\le c}\ \{\left\{\frac{\delta(x_i,x_j)}{min_{1\le k\le c}\{\Delta(x_k)\}}\right\}\} \quad (2.5)$$

$\delta(x_i,\ x_j)$ між кластерна відстань, тобто відстань між кластером $x_i,$ і x_j

$\Delta(x_k)$ внутрішньокластерна відстань кластера x_k, тобто відстань всередині кластера x_k

DB-index: Індекс Девіса-Болдіна (DBI) (введений Девідом Л. Девісом і Дональдом В. Боулдіном у 1979 році), метрика для оцінки алгоритмів кластеризації, є внутрішньою схемою оцінки, де перевірка того, наскільки добре була виконана кластеризація, здійснюється за допомогою кількісних показників і характеристик, притаманних набору даних.

Чим нижче значення індексу БД, тим кращою є кластеризація. У цього методу також є недолік. Хороше значення, отримане за допомогою цього методу, не означає найкращого пошуку інформації.

Індекс БД для k кластерів визначається як :

$$DB\ index(U) = \frac{1}{k}\sum_{i=1}^{k}\Delta(x_i) + \Delta(x_j)/\delta(x_i,x_j) \quad (2.6)$$

$\delta(x_i,\ x_j)$ між кластерна відстань, тобто відстань між кластером $x_i,$ і x_j

$\Delta(x_k)$ внутрішньокластерна відстань кластера x_k, тобто відстань всередині кластера x_k

ARI (Adjusted Rand Index): Це метрика, яка порівнює схожість між кластеризаціями та істинними мітками даних (якщо вони доступні)[26]. ARI видає значення між -1 і 1, де високі значення вказують на хорошу схожість між істинною структурою і кластеризацією.

NMI (Normalized Mutual Information): Це метрика, яка також порівнює кластеризацію з істинними мітками даних, але враховує збіг інформації між кластерами і мітками. NMI теж видає значення між 0 і 1, де 1 вказує на ідеальний збіг.

Формула розрахунку NMI:

$$NMI\ (\Omega, C) = \frac{I(\Omega;C)}{[H(\Omega)+H(C)]/2} \quad (2.7)$$

Purity: Purity є простим і прозорим мірилом оцінки. Нормалізована взаємна інформація може бути теоретично інтерпретована. Індекс Ренда карає як хибнопозитивні, так і хибнонегативні рішення під час кластеризації. Міра F додатково підтримує диференційоване зважування цих двох типів помилок.

Для обчислення чистоти кожному кластеру присвоюється клас, який найчастіше зустрічається в кластері, а потім вимірюється точність цього присвоєння шляхом підрахунку кількості правильно віднесених данних і ділення на N. Формально:

$$purity(\Omega, C) = \frac{1}{N}\sum_k max_j|\omega_k \cap C_j| \quad (2.8)$$

Де $\Omega = \{w_1, w_2, w_3 \dots w_k\}$ множина кластерів і $C = \{c_1, c_2, c_3 \dots c_j\}$ множина класів. Високої чистоти легко досягти, коли кількість кластерів велика - зокрема, чистота дорівнює 1, якщо кожна точка даних отримує власний кластер. Таким чином, ми не можемо використовувати чистоту як компроміс між якістю кластеризації та кількістю кластерів.

Мірою, яка дозволяє нам зробити цей компроміс, є нормалізована взаємна інформація або NMI.

2.3. Часова та просторова складність алгоритмів кластеризації

2.3.1. **Часова та просторова складність алгоритму K-means**

Алгоритм K-means знаходить K кластерів за певним стандартом, і, як правило, використовує стандарт квадратичної похибки. Часова складність алгоритму k-means становить O (NTK), де N - загальна кількість наборів даних, K - загальна кількість розділів, а T – це кількість ітерацій у процесі кластеризації. Просторова складність для алгоритму k-means становить $\mathcal{O}((n+k)d)$ де d - кількість ознак у наборі даних. Ефективність реалізації алгоритму k-середніх є відносно висока, але кластеризацію слід проводити у

випадку, коли K відомо, а початкова центральна вибирається випадковим чином, так що результат кластеризації має високу флуктуацію[12].

2.3.2. **Часова та просторова складність алгоритму Affinity propagation**

Основним недоліком методу Affinity propagation є велика часова та просторова складність. Часова складність складає $O(N^2 * T)$ де N – кількість елементів а своєю чергою T – це кількість ітерацій до збіжності алгоритма.

Крім того просторова складність описаного алгоритму складає $O(N^2)$ у випадку якщо буде використана щільна матриця подібності але можна досягнути зменьшення просторової складності за умови використання розрідженої матриці подібності. Саме цим обумовлюється ефективність застосування Affinity propagation до не великих наборів даних.

Affinity Propagation - це алгоритм кластеризації, який явно не вимагає зберігання матриці відстаней або матриці подібності. Замість цього він використовує набір повідомлень з реальними значеннями між точками даних, щоб визначити, які точки є зразками. Просторова складність Affinity Propagation зазвичай вважається O(N^2), де N - кількість точок даних. Основний внесок у просторову складність Affinity Propagation вносить необхідність зберігати матрицю подібності, яка має розмір N x N. Крім того, існують матриці повідомлень, які потрібно зберігати під час ітерацій алгоритму[24]. Важливо зазначити, що точна складність простору може змінюватися залежно від деталей реалізації та оптимізацій. Деякі реалізації можуть використовувати розріджені матриці або інші методи для зменшення вимог до пам'яті, але загальна ідея полягає в тому, що складність простору масштабується з квадратом кількості точок даних.

2.3.3. **Часова та просторова складність алгоритму DBSCAN**

DBSCAN - це алгоритм кластеризації на основі щільності[14]. Ось загальний огляд його часової та просторової складності:

Часова складність:

- Пошук по сусідству: Для кожної точки даних DBSCAN повинен визначити кількість точок в межах заданої околиці. Залежно від структури даних, що використовується для цього пошуку (наприклад, структури просторової індексації, такі як KD-дерева, або перебір), часова складність часто становить O(n log n) або O(n^2), де n - кількість точок даних.
- Ідентифікація опорних точок: Визначення основних точок (точок з достатньою кількістю сусідів в межах околиці) має часову складність O(n).
- Розширення кластерів: Часова складність розширення кластерів зазвичай становить O(n), оскільки кожна точка відвідується не більше одного разу протягом процесу.

Просторова складність:

- Зберігання точок ядра: Просторова складність зберігання опорних точок становить O(n).
- Кластерні призначення: Об'ємна складність зберігання кластерних призначень для кожної точки даних становить O(n).
- Структури даних: Додаткові структури даних, що використовуються для ефективного пошуку околиць, збільшують складність простору. Вибір структур даних може впливати на ефективність алгоритму.

Практичні міркування:

- DBSCAN ефективний для наборів даних з нерівномірною щільністю і може виявляти кластери довільної форми.
- Вибір метрики відстані та параметра розміру околиці (ε) може впливати на результати.

- DBSCAN чутливий до порядку точок даних, проте остаточна кластеризація не повинна бути чутливою до порядку обробки точок.
- Продуктивність алгоритму може змінюватися залежно від обраних структур даних для пошуку околиць.

Таким чином, DBSCAN має часову складність, яка часто є лінійною або трохи суперлінійною в залежності від кількості точок даних, що робить його ефективним для багатьох практичних застосувань. Однак, вибір метрики відстані та параметрів розміру околиці, а також структури даних, що використовуються для пошуку околиць, може впливати на поведінку та продуктивність алгоритму.

2.3.4. **Часова та просторова складність алгоритму HDBSCAN**

Часова та просторова складність HDBSCAN залежить від різних факторів, включаючи розмір набору даних, структуру даних та параметри, обрані для алгоритму. Ось загальний огляд:

Часова складність:

- Найгірша часова складність: HDBSCAN має найгіршу часову складність O(n log n), де n - кількість точок у наборі даних. Ця складність пов'язана з побудовою графа взаємної досяжності та обчисленням мінімального остовного дерева.
- Найкраща часова складність: У найкращому випадку, коли набір даних має чітку і просту структуру, часова складність може бути близькою до лінійної, O(n).

Просторова складність:

- Основна пам'ять: Просторова складність HDBSCAN в першу чергу визначається вимогами до пам'яті для зберігання графа взаємної досяжності та мінімального остовного дерева. У найгіршому випадку вона становить O(n).

- Додаткова пам'ять: Залежно від реалізації, може знадобитися додаткова пам'ять для зберігання облікових і проміжних структур даних. Наприклад, підтримка черги пріоритетів для ефективного обчислення мінімального остовного дерева може призвести до додаткової складності простору.

Практичні міркування:

- На практиці на ефективність HDBSCAN можуть впливати такі фактори, як вибір метрики відстані, стратегії оптимізації, що використовуються при реалізації, та характеристики набору даних. Великі та складні набори даних можуть призвести до більшого часу виконання та більшого використання пам'яті.

Варто зазначити, що ці складнощі є теоретичними і можуть відрізнятися в різних реалізаціях. Ефективність HDBSCAN часто робить його гарним вибором для завдань кластеризації, особливо при роботі з наборами даних різної щільності та форми.

2.3.5. **Часова та просторова складність алгоритму OPTICS**

OPTICS - ще один алгоритм кластеризації на основі щільності, призначений для виявлення кластерів різної форми та розміру в просторових даних[21]. Ось короткий огляд часової та просторової складності OPTICS:

Часова складність:

- Найгірший випадок часової складності: Найгірша часова складність OPTICS зазвичай становить O(n^2 * log(n)), де n - кількість точок даних. Це пов'язано з необхідністю обчислення попарних відстаней між усіма точками даних, що займає O(n^2) часу, та використанням черги пріоритетів для ефективної обробки точок даних.
- Найкраща часова складність: У найкращому випадку, коли набір даних має чітку і просту структуру, часова складність може бути ближчою до лінійної, O(n).

Просторова складність:

- Оперативна пам'ять: Вимоги до оперативної пам'яті OPTICS залежать від зберігання матриці відстаней, яка в найгіршому випадку становить O(n^2).
- Додаткова пам'ять: Додаткова пам'ять може знадобитися для підтримки черги пріоритетів під час виконання алгоритму. Просторова складність додаткових структур даних може змінюватися залежно від реалізації.

Практичні міркування:

На фактичну продуктивність OPTICS можуть впливати такі фактори, як вибір метрики відстані, стратегії оптимізації, що використовуються в реалізації, та характеристики набору даних. Здатність алгоритму ідентифікувати кластери різної форми та розміру робить його придатним для певних типів наборів даних.

Важливо зазначити, що хоча OPTICS має деякі переваги, наприклад, не вимагає попереднього визначення кількості кластерів, він може бути дороговартісним в обчисленнях для великих наборів даних. Крім того, різні реалізації можуть демонструвати різну продуктивність. Як і з будь-яким іншим алгоритмом кластеризації, рекомендується поекспериментувати з різними налаштуваннями та оцінити продуктивність алгоритму на конкретних наборах даних.

2.3.6. **Часова та просторова складність алгоритму Spectral Clustering**

Часова та просторова складність спектральної кластеризації залежить від конкретної реалізації та методу, який використовується для обчислення власного розкладу матриці афінності. Тут я надам загальний огляд:

Часова складність:

- Власна декомпозиція: Найвитратніший в обчислювальному плані крок спектральної кластеризації пов'язаний з власним розкладанням матриці афінності, яке зазвичай має часову складність O(n^3) для щільної

матриці, де n - кількість точок даних. Ця складність виникає через такі методи, як QR-алгоритм або алгоритм Ланчоса, що використовуються для обчислення власних значень та власних векторів.

- Побудова матриці афінності: Побудова матриці афінності може мати часову складність O(n^2) або O(n^2 log(n)), залежно від методу, що використовується. Для розріджених матриць афінності складність може бути меншою.

Кластеризація за методом K-середніх: Після отримання власних векторів спектральна кластеризація часто використовує кластеризацію за методом k-середніх на низьковимірному представленні. Часова складність k-середніх зазвичай становить O(k * n * d * i * t), де k - кількість кластерів, n - кількість точок даних, d - кількість вимірів, i - кількість ітерацій та t - кількість балів, віднесених до кожного кластера. На практиці метод k-середніх часто є швидким і швидко збігається.

Просторова складність:

- Власна декомпозиція: Просторова складність алгоритмів власної декомпозиції може бути значною. Зазвичай вона становить O(n^2) для зберігання матриці афінності та O(n^2) для зберігання власних векторів і власних значень.
- Матриця афінності: Складність простору для зберігання матриці афінності залежить від її розрідженості. Для щільних матриць вона становить O(n^2), тоді як розріджені матриці можуть мати меншу просторову складність.
- Кластеризація за k-середніми: Просторова складність k-середніх залежить від кількості кластерів, k, та розмірності даних.

Практичні міркування:

- Вибір методу власної декомпозиції може суттєво вплинути як на часову, так і на просторову складність. Деякі методи, такі як алгоритм Ланчоса, є більш ефективними для розріджених матриць.
- Кількість використовуваних власних векторів (зменшення розмірності) і кількість заданих кластерів також впливають на обчислювальні вимоги.
- Розріджені матриці афінності можуть бути кращими з точки зору як часової, так і просторової складності.

Важливо зазначити, що хоча спектральна кластеризація є ефективною для певних типів даних, обчислювальні витрати можуть бути високими для великих наборів даних, особливо при використанні щільних матриць афінності та методів власної декомпозиції з кубічною часовою складністю. Для дуже великих наборів даних можна розглянути апроксимації або масштабовані варіанти спектральної кластеризації.

2.3.7. **Часова та просторова складність алгоритму Meanshift**

Кластеризація Meanshift - це непараметричний алгоритм кластеризації, який ідентифікує кластери, знаходячи моди в розподілі даних. Ось загальний огляд часової та просторової складності кластеризації Meanshift:

Часова складність:

- Ініціалізація: Часова складність кроку ініціалізації часто становить O(n), де n - кількість точок даних. На цьому кроці призначаються початкові центроїди кластерів.
- Ітерації зсуву середнього: Основні обчислювальні витрати пов'язані з ітераційною процедурою зсуву середнього, коли кожна точка даних зсувається в бік її локального розподілу. Часова складність цього кроку зазвичай становить O(T * n * d), де T - кількість ітерацій, n - кількість точок даних, та d - кількість вимірів.
- Збіжність: Кількість ітерацій T, необхідна для збіжності, може змінюватися залежно від набору даних та обраних критеріїв збіжності.

Просторова складність:

- Зберігання даних: Просторова складність для зберігання точок даних становить O(n * d), де n - кількість точок даних, а d - кількість вимірів.
- Кластерні призначення: Складність простору для зберігання кластерних призначень для кожної точки даних становить O(n).
- Оцінка щільності ядра: Обчислювальна складність для оцінки щільності ядра, що використовується в обчисленні середнього зсуву, залежить від методу та представлення даних. Вона часто є функцією кількості точок даних та розмірності.

Практичні міркування:

- Метод зсуву середнього чутливий до вибору параметра смуги пропускання або радіуса, який впливає на розмір локальної околиці.
- Збіжність Meanshift залежить від обраного критерію зупинки. Загальний підхід полягає в тому, щоб зупинитися, коли вектор середнього зсуву стає малим.
- Хоча Meanshift ефективний для певних типів даних, його обчислювальна складність може бути високою для великих наборів даних.
- Зсув середнього значення, як правило, добре працює, коли кластери мають схожу форму і розмір.

На практиці на продуктивність Meanshift можуть впливати такі фактори, як характеристики набору даних, вибір параметрів і деталі реалізації. Він може бути не таким масштабованим, як деякі інші алгоритми кластеризації, особливо для великих наборів даних високої розмірності.

2.4. **Висновок до розділу**

Виходячи з теоретичних відомостей, виглядає так, щонайменшу обчислювальну складність матимуть алгоритми Affinity propagation та Dbscan, однак часова та просторова складність майже всіх перелічених алгоритмів залежить від даних, а значить треба провести випробування. В

якості метрики якості варто обрати набір кількох метрик. В рамках цієї атестаційної роботи такими метриками будуть силует та індекс Ренда.

РОЗДІЛ 3 ПОРІВНЯННЯ МЕТОДІВ КЛАСТЕРИЗАЦІЇ ЗОБРАЖЕНЬ В УМОВАХ ОБМЕЖЕНИХ ОБЧИСЛЮВАЛЬНИХ ПОТУЖНОСТЕЙ З МЕТОЮ ПОБУДОВИ ВІДПОВІДНОЇ ІТ

3.1. Умови тестування

Цілком зрозуміло, що у випадку наявності достатніх обчислювальних потужностей, як у сенсі можливостей процесора, так і за умови наявності великої кількості вільної швидкої пам’яті, задача кластеризації класичними методами не займе багато часу і різниця між алгоритмами полягатиме тільки в тому, наскільки якісно той чи інший алгоритм опрацьовуватиме ті чи інші дані. Але ситуація, суттєво змінюється, коли обчислювальні можливості процесора обмежені, обсяг пам’яті не надто великий і дані надходять надзвичайно швидко. Прикладом такої ситуації можуть бути речі IoT, які мусять реагувати на швидку зміну зовнішнього середовища. Тож постає питання: який алгоритм обрати в тому чи іншому випадку, спираючись, як на теоретичні відомості, так і на дослідницькі дані реальних випробувань. Для того, щоб відповісти на це питання, оберімо у якості застосунку систему монокулярного машинного зору, яка мусить обробляти відеопотік зі швидкістю не менше 20 кадрів на секунду, роздільна здатність зображення становить 1920 на 1080 точок (Full HD). В якості основи для цієї системи машинного зору використовувався RaspberryPi model 4B. Врешті алгоритм має доволі точно визначати всі точки, які належать до того чи іншого об’єкта, який можна розрізнити людським оком. З-поміж всіх методів кластеризації, перевагу слід віддати тим, які не вимагають попереднього передбачення кількості предметів на зображенні. Бо в цьому випадку доведеться робити кілька ітерацій кластеризації з метою мінімізації помилок.

3.2. Реалізація affinity propagation

Згідно розділу 1.2.2, код алгоритму affinity propagation у найпростішій реалізації може виглядати так:

Лістинг 3.1

```
import numpy as np
from utilities import *

def affinity_propagation(X, max_iter=200, convergence_iter=15, damping=0.9):
    n_samples = X.shape[0]
    preferences = np.zeros(n_samples)
    similarity_matrix = np.zeros((n_samples, n_samples))

    for i in range(n_samples):
        for j in range(n_samples):
            similarity_matrix[i, j] = -np.linalg.norm(X[i] - X[j]) ** 2

    np.fill_diagonal(similarity_matrix, np.median(similarity_matrix))
    exemplars = np.zeros(n_samples, dtype=int)

    for _ in range(max_iter):
        tmp_preferences = similarity_matrix + np.diag(preferences)
        tmp_exemplars = np.argmax(tmp_preferences, axis=1)

        for i in range(n_samples):
            exemplars[i] = tmp_exemplars[tmp_exemplars[i]]

        for i in range(n_samples):
            S = similarity_matrix[i, :] - np.max(similarity_matrix[i, :])
            S[i] = similarity_matrix[i, i] - np.max(S)
            preferences[i] = damping * preferences[i] + (1 - damping) * S.sum()

        if np.all(exemplars == tmp_exemplars):
            break
```

```
    return exemplars
```

3.3. Dbscan

Згідно розділу 1.2.3, код алгоритму dbscan у найпростішій реалізації може виглядати так:

Лістинг 3.2

```
import numpy as np
from utilities import *

def dbscan(data, eps, min_samples):
    """
    Реалізація методу DBSCAN для кластеризації зображень.

    Параметри:
    - data: набір даних (2D масив, представляє зображення)
    - eps: радіус сусідства для визначення сусідів
    - min_samples: мінімальна кількість сусідів, необхідних для утворення кластера

    Повертає:
    - labels: масив, що містить мітки кластерів для кожної точки
    """
    num_points = data.shape[0]
    labels = np.zeros(num_points)
    current_label = 0

    for i in range(num_points):
        if labels[i] != 0:
            continue

        neighbors = find_neighbors(data, i, eps)

        if len(neighbors) < min_samples:
            labels[i] = -1
```

```
        else:
            current_label += 1
            expand_cluster(data, labels, i, neighbors, current_label, eps, min_samples)

    return labels
```

3.4. Hdbscan

Згідно розділу 1.2.4, код алгоритму hdbscan у найпростішій реалізації може виглядати так:

Лістинг 3.3

```
import numpy as np

def hdbscan(data, min_cluster_size, min_samples, metric="euclidean"):
    tree = AgglomerativeClustering(n_clusters=None, affinity=metric, linkage="ward")
    tree.fit(data)
    clusters = []
    for i, node in enumerate(tree.children_):
        if node == -1:
            clusters.append([])
        else:
            clusters[node].append(i)
    for i, cluster in enumerate(clusters):
        if len(cluster) < min_cluster_size:
            for point in cluster:
                clusters[point].append(i)
            clusters.remove(cluster)
    for i, cluster in enumerate(clusters):
        if len(cluster) < min_samples:
            for point in cluster:
                clusters[point].append(i)
            clusters.remove(cluster)
    return clusters
```

3.5. OPTICS

Згідно розділу 1.2.5, код алгоритму OPTICS у найпростішій реалізації може виглядати так:

Лістинг 3.4

```
import numpy as np
from utilities import *

def optics(data, epsilon, min_samples):
    core_distances = np.zeros(len(data))
    reachability_distances = np.full(len(data), np.inf)
    processed = np.zeros(len(data), dtype=bool)
    ordering = []

    for i in range(len(data)):
        if not processed[i]:
            neighbors = find_neighbors(data, i, epsilon)
            processed[i] = True
            ordering.append(i)

            if len(neighbors) >= min_samples:
                core_distances[i] = np.sort([euclidean_distance(data[i], data[j]) for j in
neighbors])[min_samples - 1]

                seeds = []
                seeds = update_seeds(neighbors, i, seeds, core_distances, reachability_distances)

                while seeds:
                    current_p = seeds.pop(0)
                    processed[current_p] = True
                    ordering.append(current_p)

                    current_neighbors = find_neighbors(data, current_p, epsilon)

                    if len(current_neighbors) >= min_samples:
```

```
                core_distances[current_p] = np.sort([euclidean_distance(data[current_p], data[j])
for j in current_neighbors])[min_samples - 1]
                seeds = update_seeds(current_neighbors, current_p, seeds, core_distances,
reachability_distances)

    return ordering, core_distances, reachability_distances

def update_seeds(neighbors, seed, seeds, core_distances, reachability_distances):
    for neighbor in neighbors:
        if not seeds.__contains__(neighbor):
            reachability_distance = max(core_distances[seed], euclidean_distance(data[seed],
data[neighbor]))
            if reachability_distance < reachability_distances[neighbor]:
                reachability_distances[neighbor] = reachability_distance
                seeds.append(neighbor)
    seeds.sort(key=lambda x: reachability_distances[x])
    return seeds
```

3.6. Спектральна кластеризація

Згідно розділу 1.2.6, код алгоритму спектральної кластеризації у найпростішій реалізації може виглядати так:

Лістинг 3.5

```
import numpy as np
from utilities import *

def affinity_matrix(data, sigma):
    n = len(data)
    A = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            A[i, j] = np.exp(-euclidean_distance(data[i], data[j]) / (2 * sigma**2))
    return A

def spectral_clustering(data, num_clusters, sigma):
```

```
    A = affinity_matrix(data, sigma)
    D_inv_sqrt = np.diag(1 / np.sqrt(np.sum(A, axis=1)))
    L = np.identity(len(data)) - D_inv_sqrt @ A @ D_inv_sqrt

    _, eigenvectors = np.linalg.eigh(L)
    selected_eigenvectors = eigenvectors[:, :num_clusters]
    centroids, labels = k_means(selected_eigenvectors, num_clusters)
    return labels

def k_means(data, num_clusters, max_iters=100):
    n, m = data.shape
    centroids = data[np.random.choice(n, num_clusters, replace=False)]
    labels = np.zeros(n, dtype=int)
    for _ in range(max_iters):
        distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)
        new_centroids = np.array([data[labels == k].mean(axis=0) for k in range(num_clusters)])
        if np.all(new_centroids == centroids):
            break
        centroids = new_centroids
    return centroids, labels
```

3.7. Meanshift

Згідно розділу 1.2.7, код алгоритму MeanShift у найпростішій реалізації може виглядати так:

Лістинг 3.6

```
import numpy as np
from utilities import *

def mean_shift(data, bandwidth=0.01, min_distance=1e-4):
    shifted_points = data.copy().reshape(-1)

    while True:
        new_shifted_points = np.zeros_like(shifted_points)
```

```
    for i, point in enumerate(shifted_points):
        weights = np.exp(-0.5 * ((point - shifted_points) / bandwidth) ** 2)
        shifted_point = np.sum(weights * shifted_points, axis=0) / np.sum(weights)
        new_shifted_points[i] = shifted_point
    if np.max(np.abs(new_shifted_points - shifted_points)) < min_distance:
        break
    shifted_points = new_shifted_points
labels, new_data = np.unique(shifted_points, axis=0, return_inverse=True)
return labels, new_data
```

3.8. Результати випробувань

Таблиця 3.1

Часова оцінка алгоритмів кластеризації у результаті випробувань

	Affinity propagation	Dbscan	Hdbscan	OPTICS	Спектральна кластеризація	Meanshift
Розмір Зображення	1920x1080	1920x 1080	1920x 1080	1920x 1080	1920x1080	1920x1080
Кількість потоків	1	1	1	1	1	1
Кількість випробувань	100	100	100	100	100	100
Середній час виконання однієї спроби	1,85 с.	1,94 с.	2,28 с.	2,88 с.	2,41 с.	2,625 с.
Середня очікувана кількість кластерів	900	900	900	900	900	900
Фактична кількість кластерів	831	836	842	890	847	863
Найкраща оцінка силуету	0,8	0,75	0,9	0,9	0,75	0,85
Найкраще значення індексу Ренда	0,8	0,66	0,7	0,73	0,7	0,79

3.9. Виявлення об’єктів

Як видно з таблиці 3.1, найшвидшим в умовах експерименту виявив себе Affinity propagation. Навіть попри порівняно нижчу точність він є кращим варіантом, оскільки швидкість кластеризації під час обробки відеопотоку є вкрай важливою. Нестачу точності можна компенсувати за допомогою «мудрості натовпу», усереднивши результати кластеризації з кількох сусідніх кадрів.

Наступним кроком у побудові інформаційної технології виявлення об’єктів у відеопотоці є виділення прямокутників, що містять кластери певного розміру. Це можна зробити за допомогою функції наведеної нижче:

Лістинг 3.7

```
def get_objects(clusters_map, min_area=100):
    objects = []
    cm = np.copy(clusters_map)
    side = int(np.sqrt(min_area)) + 1
    for i in range(len(cm)-side):
        for j in range(len(cm[0])-side):
            found = False
            for object in objects:
                if object[0] <= i and object[2] >= i and object[1] <= j and object[3] >= j:
                    found = True


            if found:
                continue
            box = np.array([cm[i][j], cm[i][j+side-1], cm[j+side-1][i], cm[i+side-1][j+side-1]])
            if len(set(box)) == 4:
                continue
            solid_rect = 0
            cluster = box[0]
            new_side = side
```

```
        box_copy = np.copy(box)
        for _ in range(3):
            solid_rect += np.sum([0.1 if box_copy[k] == cluster else 0 for k in
range(len(box_copy))])
            new_side = int(2*new_side/3)
            box_copy = np.array([cm[i][j+new_side-1], cm[i+new_side-1][j], cm[i+new_side-
1][j+new_side-1]])
        if solid_rect <= 0.6:
            continue
        diff_x = side
        diff_y = side
        bound_left = j - diff_x if j - diff_x >= 0 else 0
        bound_top  = i - diff_y if i - diff_y >= 0 else 0
        bound_right = j + diff_x if j + diff_x < len(cm[0]) - 1 else len(cm[0]) - 1
        bound_bottom = i + diff_y if i + diff_y < len(cm) - 1 else len(cm) - 1
        expand_left = True if bound_left > 0 else False
        expand_top = True if bound_top > 0 else False
        expand_right = True if bound_right < len(cm[0]) -1 else False
        expand_bottom = True if bound_bottom < len(cm) - 1 else False
        while (expand_left and bound_left > 0) or (expand_top and bound_top > 0) or \
            (expand_right and bound_right < len(cm[0]) - 1) or (expand_bottom and
bound_bottom < len(cm[0]) - 1):
            if expand_top:
                bound_top  = int(bound_top - diff_y) if (bound_top - diff_y) >= 0 else 0
                solid_line = 0
                diff_x = (bound_right - bound_left) / 10 if (bound_right - bound_left) / 10 > 1 else 1
                x = int(bound_left + diff_x)
                for _ in range(10):
                    solid_line += 0.1 if cm[bound_top][x] == cluster else 0
                    x = int(x + diff_x)
                if solid_line < 0.3:
                    expand_top = False
            if expand_left:
```

```
            bound_left = int(bound_left - diff_x) if int(bound_left - diff_x) >= 0 else 0
            solid_line = 0
            diff_y = (bound_bottom - bound_top) / 10 if (bound_bottom - bound_top) / 10 > 1
else 1
            y = int(bound_top + diff_y)
            for _ in range(10):
                solid_line = 0.1 if cm[y][bound_left] == cluster else 0
                y = int(y + diff_y)
            if solid_line < 0.3:
                expand_left = False
        if expand_bottom:
            bound_bottom  = int(bound_bottom + diff_y) if int(bound_bottom - diff_y) <=
len(cm) - 1 else len(cm) - 1
            solid_line = 0
            diff_x = (bound_right - bound_left) / 10 if (bound_right - bound_left) / 10 > 1 else 1
            x = int(bound_left + diff_x)
            for _ in range(10):
                solid_line += 0.1 if cm[bound_bottom][x] == cluster else 0
                x = int(x + diff_x)
            if solid_line < 0.3:
                expand_bottom = False
        if expand_right:
            bound_right = int(bound_right + diff_x) if int(bound_right + diff_x) <= len(cm[0]) -
1 else len(cm[0]) - 1
            solid_line = 0
            diff_y = (bound_bottom - bound_top) / 10 if (bound_bottom - bound_top) / 10 > 1
else 1
            y = int(bound_top + diff_y)
            for _ in range(10):
                solid_line = 0.1 if cm[y][bound_right] == cluster else 0
                y = int(y + diff_y)
            if solid_line < 0.3:
                expand_right = False
```

```
        objects.append([bound_top, bound_left, bound_bottom, bound_right])
    return objects
```

Ця функція повертає список прямокутників на зображенні, що містять кластери площею не менш за min_area. Після виклику функції, слід знайти всі прямокутники з однаковими центроїдами (середнім значенням кольору у межах прямокутника) та з ті з них, які межують один з одним, треба спробувати об’єднати у прямокутник більшого розміру. Прямокутник найбільшого розміру утворюватиме тло. Залежно від задачі, його можна аналізувати, але найчастіше шуканий об’єкт не є тлом. На рисунку нижче можна побачити результат роботи описаної вище технології. Програма повинна знайти на зображенні об’єкти кольору відмінного від кольору тла та обвести їх прямокутником.

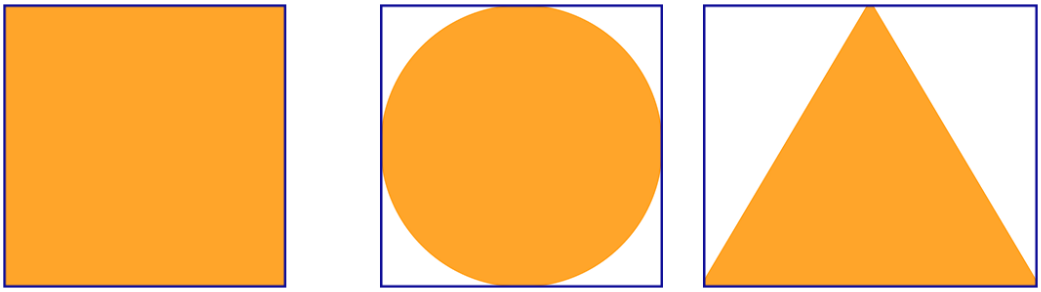


Рис. 3.1 Результат застосування інформаційної технології виявлення об’єкту на зображенні

3.10. Висновок до розділу

Результати кластеризації показують, що найточнішим, але найдовшим у часі є алгоритм OPTICS, але алгоритм affinity propagation є найшвидшим та достатньо точним для практичного використання. Розроблена технологія за результатами кластеризації передбачає пошук прямокутників на зображенні,

що містять кластери певної площі, які потім вклеюються у разі сусідства задля утворення тла. За результатами роботи алгоритму можна дізнатися в яких прямокутниках знаходяться кластери площі не меншої за потрібну й надалі ці дані можуть бути використані у системах класифікації (випадкові ліси чи нейронні мережі) або у рекурентних нейронних мережах для стеження за об’єктами. Треба зважати, що цей висновок дійсний, технічно та функціонально подібних до системи, що використовувалася під час розробки.

ВИСНОВОК

В ході виконання атестаційної роботи був виконаний аналіз способу визначення об’єктів на зображенні в системах машинного зору, досліджені традиційні методи кластеризації точок на зображенні, метрики кластеризації, як спосіб оцінки результатів її роботи, були проаналізовані теоретичні часові та просторові оцінки різних алгоритмів кластеризації та виконано практичну перевірку різних алгоритмів кластеризації у системі машинного зору, що має обмежену обчислювальну продуктивність. Були розроблені та реалізовані алгоритми обчислення метрик кластеризації для оцінки точності отримання результатів при застосуванні обраних методів в умовах обмежених обчислювальних потужностей. Також був запропонований алгоритм виділення прямокутників на зображенні, що містять кластери, які відповідають об’єктам певного розміру. За результатами виконання роботи запропонована інформаційна технологія має включати в себе кластеризацію методом affinity propagation з подальшим виділенням прямокутників, що містять об'єкти та визначення кольору тла. Результати застосування розробленої технології можуть бути використані у класифікаційних алгоритмах та рекурентних нейронних мережах для стеження за об’єктами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. https://www.mdpi.com/2076-3417/12/5/2405
2. https://bobrupakroy.medium.com/optics-clustering-intro-76dcdaf94bde
3. https://www.mygreatlearning.com/blog/introduction-to-spectral-clustering/
4. https://www.geeksforgeeks.org/ml-mean-shift-clustering/
5. https://www.nvidia.com/en-us/glossary/data-science/clustering/
6. https://www.nature.com/articles/nmeth.4299#citeas
7. https://www.geeksforgeeks.org/clustering-in-machine-learning/
8. https://www.explorium.ai/blog/machine-learning/clustering-when-you-should-use-it-and-avoid-it/
9. https://developer.nvidia.com/discover/cluster-analysis
10. https://www.toptal.com/machine-learning/clustering-algorithms
11. https://neptune.ai/blog/clustering-algorithms
12. https://www.researchgate.net/profile/Mohammad-Kamrul-Hasan-5/publication/353839789_Performances_of_K-Means_Clustering_Algorithm_with_Different_Distance_Metrics/links/6117623b1ca20f6f861e738f/Performances-of-K-Means-Clustering-Algorithm-with-Different-Distance-Metrics.pdf
13. https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1
14. https://arxiv.org/pdf/2004.11530.pdf
15. https://towardsdatascience.com/clustering-concepts-algorithms-and-applications-f512a949549a
16. https://support.minitab.com/en-us/minitab/21/help-and-how-to/statistical-modeling/multivariate/how-to/cluster-k-means/interpret-the-results/all-statistics-and-graphs/#:~:text=the%20cluster%20centroid.-,Cluster%20centroid,dimensional%20average%20of%20the%20cluster.
17. https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html

18. https://timroughgarden.org/papers/kmeans.pdf
19. "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar
20. "Cluster Analysis" by Brian S. Everitt, Sabine Landau, Morven Leese, and Daniel Stahl
21. "Data Clustering: Algorithms and Applications" by Charu C. Aggarwal
22. "Evaluation of Clustering Algorithms" by Douglas A. Reynolds
23. "On Comparing Clusterings: An Objective Function Approach" by Marina Meilă
24. "A Comparison of External Clustering Evaluation Metrics Based on Formal Constraints" by Amri Napolitano, Claire Franzon, and Aline Paes
25. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis" by Peter J. Rousseeuw
26. "Adjust Rand Index as a Measure of Clustering Agreement Between Data Partitions" by Lawrence Hubert and Phipps Arabie
27. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise" by Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu.
28. "Hierarchical Density-Based Clustering" by Rui Xia, Yan Huang, Jian Pei.
29. "Mean Shift: A Robust Approach toward Feature Space Analysis" by Dorin Comaniciu and Peter Meer.
30. "OPTICS: Ordering Points To Identify the Clustering Structure" by Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander.
31. "Clustering by Passing Messages Between Data Points" by Brendan J. Frey and Delbert Dueck.
32. "Data Science for Business" by Foster Provost and Tom Fawcett.

ДОДАТОК А

Відомість матеріалів кваліфікаційної роботи

		Позначення	Найменування	Кільк. аркушів	Примітка
	1				
	2		Документація		
	3				
	4	**ІТКІ.ДП 18.01.ДА.ПЗ**	Пояснювальна записка	98	
	5				
	6		Диск CD-R з презентацією	1	

Зм	Лист	№ докум.	Підпис	Дата	**ІТКІ.ДП 18.01.ДА.ПЗ**			
Розроб.		Пачевський			**Матеріали кваліфі-каційної роботи**	Літ.	Аркуш	Аркушів
Керівник		Коротенко					1	1
Рецензент		Ширін				НТУ «ДП» 8; 126м-22-1		
Н.контр.		Коротенко						
Зав. каф.		Гнатушенко						

ДОДАТОК Б

КОД ПРОГРАМИ

bounder.py

```
import numpy as np


def get_objects(clusters_map, min_area=100):
    objects = []
    cm = np.copy(clusters_map)
    side = int(np.sqrt(min_area)) + 1
    for i in range(len(cm)-side):
        for j in range(len(cm[0])-side):
            found = False
            for object in objects:
                if object[0] <= i and object[2] >= i and object[1] <= j and object[3] >= j:
                    found = True


            if found:
                continue
            box = np.array([cm[i][j], cm[i][j+side-1], cm[j+side-1][i], cm[i+side-1][j+side-1]])
            if len(set(box)) == 4:
                continue
            solid_rect = 0
            cluster = box[0]
            new_side = side
            box_copy = np.copy(box)
            for _ in range(3):
                solid_rect += np.sum([0.1 if box_copy[k] == cluster else 0 for k in range(len(box_copy))])
                new_side = int(2*new_side/3)
                box_copy = np.array([cm[i][j+new_side-1], cm[i+new_side-1][j], cm[i+new_side-1][j+new_side-1]])
            if solid_rect <= 0.6:
                continue
            diff_x = side
            diff_y = side
            bound_left = j - diff_x if j - diff_x >= 0 else 0
            bound_top  = i - diff_y if i - diff_y >= 0 else 0
            bound_right = j + diff_x if j + diff_x < len(cm[0]) - 1 else len(cm[0]) - 1
            bound_bottom = i + diff_y if i + diff_y < len(cm) - 1 else len(cm) - 1
            expand_left = True if bound_left > 0 else False
            expand_top = True if bound_top > 0 else False
            expand_right = True if bound_right < len(cm[0]) -1 else False
```

```
expand_bottom = True if bound_bottom < len(cm) - 1 else False
while (expand_left and bound_left > 0) or (expand_top and bound_top > 0) or \
    (expand_right and bound_right < len(cm[0]) - 1) or (expand_bottom and bound_bottom < len(cm[0]) - 1):
  if expand_top:
    bound_top  = int(bound_top - diff_y) if (bound_top - diff_y) >= 0 else 0
    solid_line = 0
    diff_x = (bound_right - bound_left) / 10 if (bound_right - bound_left) / 10 > 1 else 1
    x = int(bound_left + diff_x)
    for _ in range(10):
      solid_line += 0.1 if cm[bound_top][x] == cluster else 0
      x = int(x + diff_x)
    if solid_line < 0.3:
      expand_top = False
  if expand_left:
    bound_left = int(bound_left - diff_x) if int(bound_left - diff_x) >= 0 else 0
    solid_line = 0
    diff_y = (bound_bottom - bound_top) / 10 if (bound_bottom - bound_top) / 10 > 1 else 1
    y = int(bound_top + diff_y)
    for _ in range(10):
      solid_line = 0.1 if cm[y][bound_left] == cluster else 0
      y = int(y + diff_y)
    if solid_line < 0.3:
      expand_left = False
  if expand_bottom:
    bound_bottom  = int(bound_bottom + diff_y) if int(bound_bottom - diff_y) <= len(cm) - 1 else len(cm) - 1
    solid_line = 0
    diff_x = (bound_right - bound_left) / 10 if (bound_right - bound_left) / 10 > 1 else 1
    x = int(bound_left + diff_x)
    for _ in range(10):
      solid_line += 0.1 if cm[bound_bottom][x] == cluster else 0
      x = int(x + diff_x)
    if solid_line < 0.3:
      expand_bottom = False
  if expand_right:
    bound_right = int(bound_right + diff_x) if int(bound_right + diff_x) <= len(cm[0]) - 1 else len(cm[0]) - 1
    solid_line = 0
    diff_y = (bound_bottom - bound_top) / 10 if (bound_bottom - bound_top) / 10 > 1 else 1
    y = int(bound_top + diff_y)
    for _ in range(10):
      solid_line = 0.1 if cm[y][bound_right] == cluster else 0
```

```
            y = int(y + diff_y)
        if solid_line < 0.3:
            expand_right = False
    objects.append([bound_top, bound_left, bound_bottom, bound_right])
return objects
```

dbscan.py

```
import numpy as np
from utilities import *

def dbscan(data, eps, min_samples):
    """
    Реалізація методу DBSCAN для кластеризації зображень.

    Параметри:
    - data: набір даних (2D масив, представляє зображення)
    - eps: радіус сусідства для визначення сусідів
    - min_samples: мінімальна кількість сусідів, необхідних для утворення кластера

    Повертає:
    - labels: масив, що містить мітки кластерів для кожної точки
    """
    num_points = data.shape[0]
    labels = np.zeros(num_points)
    current_label = 0

    for i in range(num_points):
        if labels[i] != 0:
            continue

        neighbors = find_neighbors(data, i, eps)

        if len(neighbors) < min_samples:
            labels[i] = -1
        else:
            current_label += 1
            expand_cluster(data, labels, i, neighbors, current_label, eps, min_samples)

    return labels

if __name__ == '__main__':
    dataset = np.array([[1, 2], [5, 6], [1.5, 1.8], [8, 8], [1, 0.6], [9, 11]])
    classes = dbscan(dataset, 2, 2)
 print("Мітки кластерів:", classes)
```

hdbscan.py

```
import numpy as np
from utilities import *
```

```
def hdbscan(data, eps, min_samples):
    """
    Проста реалізація алгоритму HDBSCAN для кластеризації зображень.

    Параметри:
    - data: набір даних (2D масив, представляє зображення)
    - eps: радіус сусідства для визначення сусідів
    - min_samples: мінімальна кількість сусідів, необхідних для утворення кластера

    Повертає:
    - labels: масив, що містить мітки кластерів для кожної точки
    """
    num_points = data.shape[0]
    labels = np.zeros(num_points)  # Ініціалізація міток кластерів
    current_label = 0  # Початкова мітка кластера

    for i in range(num_points):
        if labels[i] != 0:
            continue

        neighbors = find_neighbors(data, i, eps)

        if len(neighbors) < min_samples:
            labels[i] = -1  # Позначте як шум (-1)
        else:
            current_label += 1
            expand_cluster(data, labels, i, neighbors, current_label, eps, min_samples)

    return labels

if __name__ == '__main__':
    # Приклад використання:
    # Замініть dataset на свій набір даних
    dataset = np.array([[1, 2], [5, 6], [1.5, 1.8], [8, 8], [1, 0.6], [9, 11]])
    classes = hdbscan(dataset, 2, 2)
 print("Мітки кластерів:", classes)
```

kmeans.py

```
import numpy as np
from utilities import *

def k_means(data, num_clusters, max_iters=100):
    n, m = data.shape
    centroids = data[np.random.choice(n, num_clusters, replace=False)]
    labels = np.zeros(n, dtype=int)

    for _ in range(max_iters):
        # Етап призначення кластерів
        distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)
```

```
        # Етап оновлення центроїдів
        new_centroids = np.array([data[labels == k].mean(axis=0) for k in range(num_clusters)])

        # Перевірка збіжності
        if np.all(new_centroids == centroids):
            break

        centroids = new_centroids

    return centroids, labels

if __name__ == '__main__':
    # Генеруємо випадкові дані для прикладу
    np.random.seed(42)
    data = np.random.randn(300, 2)
    # Проводимо кластеризацію
    classes = spectral_clustering(data, 3, 0.5)
print("Cluster Labels:", classes)
```

mean_shift.py

```
import numpy as np
from utilities import *

def mean_shift(data, bandwidth=0.01, min_distance=1e-4):
    shifted_points = data.copy().reshape(-1)

    while True:
        new_shifted_points = np.zeros_like(shifted_points)
        for i, point in enumerate(shifted_points):
            weights = np.exp(-0.5 * ((point - shifted_points) / bandwidth) ** 2)
            shifted_point = np.sum(weights * shifted_points, axis=0) / np.sum(weights)
            new_shifted_points[i] = shifted_point
        if np.max(np.abs(new_shifted_points - shifted_points)) < min_distance:
            break
        shifted_points = new_shifted_points
    labels, new_data = np.unique(shifted_points, axis=0, return_inverse=True)
    return labels, new_data

if __name__ == '__main__':
    np.random.seed(42)
    data = np.random.randn(300, 1) #* 2 + np.array([5, 5])
    bandwidth = 0.1
    cluster_labels = mean_shift(data, bandwidth)
 print("Cluster Labels:", cluster_labels)
```

optics.py

```
import numpy as np
from utilities import *
```

```
def optics(data, epsilon, min_samples):
    core_distances = np.zeros(len(data))
    reachability_distances = np.full(len(data), np.inf)
    processed = np.zeros(len(data), dtype=bool)
    ordering = []

    for i in range(len(data)):
        if not processed[i]:
            neighbors = find_neighbors(data, i, epsilon)
            processed[i] = True
            ordering.append(i)

            if len(neighbors) >= min_samples:
                core_distances[i] = np.sort([euclidean_distance(data[i], data[j]) for j in neighbors])[min_samples - 1]

                seeds = []
                seeds = update_seeds(neighbors, i, seeds, core_distances, reachability_distances)

                while seeds:
                    current_p = seeds.pop(0)
                    processed[current_p] = True
                    ordering.append(current_p)

                    current_neighbors = find_neighbors(data, current_p, epsilon)

                    if len(current_neighbors) >= min_samples:
                        core_distances[current_p] = np.sort([euclidean_distance(data[current_p], data[j]) for j in current_neighbors])[min_samples - 1]
                        seeds = update_seeds(current_neighbors, current_p, seeds, core_distances, reachability_distances)

    return ordering, core_distances, reachability_distances

def update_seeds(neighbors, seed, seeds, core_distances, reachability_distances):
    for neighbor in neighbors:
        if not seeds.__contains__(neighbor):
            reachability_distance = max(core_distances[seed], euclidean_distance(data[seed], data[neighbor]))
            if reachability_distance < reachability_distances[neighbor]:
                reachability_distances[neighbor] = reachability_distance
                seeds.append(neighbor)
    seeds.sort(key=lambda x: reachability_distances[x])
    return seeds

if __name__ == '__main__':
    np.random.seed(42)
    data = np.random.randn(300, 2)
    epsilon = 0.5
    min_samples = 5
    ordering, core_distances, reachability_distances = optics(data, epsilon, min_samples)
    print("Ordering:", ordering)
    print("Core Distances:", core_distances)
print("Reachability Distances:", reachability_distances)
```

rand.py

```
import numpy as np

def rand_index(labels_true, labels_pred):
    intersection = np.sum(labels_true == labels_pred)
    sum_true = np.sum(labels_true)
    sum_pred = np.sum(labels_pred)
    sum_diag = np.sum(labels_true == labels_true)
    sum_row = np.sum(labels_true, axis=1)
    sum_col = np.sum(labels_true, axis=0)
    rand_index = (
        (intersection + sum_diag - sum_row - sum_col) / (sum_true * sum_pred)
    )
return rand_index
```

silhouette.py

```
import numpy as np

def silhouette_score(data, labels):
    distances_to_nearest_cluster = np.min(
        np.linalg.norm(data - data[labels == label], axis=1), axis=1
    )
    distances_to_all_clusters = np.mean(
        np.linalg.norm(data - data[labels == label], axis=1), axis=0
    )
    silhouette_scores = (distances_to_nearest_cluster - distances_to_all_clusters) / np.maximum(
        distances_to_nearest_cluster, distances_to_all_clusters
    )
 return silhouette_scores.mean()
```

spectral_clustering.py

```
import numpy as np
from utilities import *

def affinity_matrix(data, sigma):
    n = len(data)
    A = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            A[i, j] = np.exp(-euclidean_distance(data[i], data[j]) / (2 * sigma**2))
    return A

def spectral_clustering(data, num_clusters, sigma):
    A = affinity_matrix(data, sigma)
    D_inv_sqrt = np.diag(1 / np.sqrt(np.sum(A, axis=1)))
```

```
    L = np.identity(len(data)) - D_inv_sqrt @ A @ D_inv_sqrt

    _, eigenvectors = np.linalg.eigh(L)
    selected_eigenvectors = eigenvectors[:, :num_clusters]
    centroids, labels = k_means(selected_eigenvectors, num_clusters)
    return labels

def k_means(data, num_clusters, max_iters=100):
    n, m = data.shape
    centroids = data[np.random.choice(n, num_clusters, replace=False)]
    labels = np.zeros(n, dtype=int)
    for _ in range(max_iters):
        distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)
        new_centroids = np.array([data[labels == k].mean(axis=0) for k in range(num_clusters)])
        if np.all(new_centroids == centroids):
            break
        centroids = new_centroids
    return centroids, labels

if __name__ == '__main__':
    np.random.seed(42)
    data = np.random.randn(300, 2)
    classes = spectral_clustering(data, 3, 0.5)
print("Cluster Labels:", classes)
```

utilities.py

```
import numpy as np

def euclidean_distance(point1, point2):
    """
    Обчислення евклідової відстані між двома точками.

    Параметри:
    - point1: координати першої точки
    - point2: координати другої точки

    Повертає:
    - distance: евклідова відстань між точками
    """
    return np.sqrt(np.sum((point1 - point2) ** 2))

def find_neighbors(data, point_index, eps):
    """
    Знаходження сусідів точки в радіусі eps.

    Параметри:
    - data: набір даних (масив точок)
    - point_index: індекс поточної точки
    - eps: радіус сусідства
```

```
    Повертає:
    - neighbors: список індексів сусідніх точок
    """
    neighbors = []
    for i, point in enumerate(data):
        if euclidean_distance(data[point_index], point) <= eps and i != point_index:
            neighbors.append(i)
    return neighbors

def expand_cluster(data, labels, point_index, neighbors, cluster_label, eps, min_samples):
    """
    Розширення кластера для заданої точки.

    Параметри:
    - data: набір даних (масив точок)
    - labels: масив міток кластерів
    - point_index: індекс поточної точки
    - neighbors: список індексів сусідніх точок
    - cluster_label: поточна мітка кластера
    - eps: радіус сусідства
    - min_samples: мінімальна кількість сусідів
    """
    labels[point_index] = cluster_label  # Позначте поточну точку міткою кластера

    i = 0
    while i < len(neighbors):
        current_neighbor = neighbors[i]

        # Якщо сусід ще не має мітки, призначте йому поточну мітку кластера
        if labels[current_neighbor] == 0:
            labels[current_neighbor] = cluster_label

            # Знайдіть сусідів нового сусіда
            current_neighbors = find_neighbors(data, current_neighbor, eps)

            # Додайте нових сусідів до загального списку сусідів
            if len(current_neighbors) >= min_samples:
                neighbors.extend(current_neighbors)

        i += 1
```

ДОДАТОК В

ВІДГУК
на комплексну кваліфікаційну роботу рівня магістра «Розробка інформаційної технології визначення об’єктів у системах машинного зору в умовах обмежених обчислювальних потужностей» студента групи 126м-22-1 Пачевського Михайла Володимировича

1 Мета даної кваліфікаційної роботи – підвищення точності визначення об’єкту на зображеннях у системах машинного зору за умови обмеженої продуктивності.

2 Обрана тема актуальна тому, що під час аналізу інформації що надходить від системи машинного зору є ідентифікація та класифікація об’єктів навколишнього світу, враховуючи те що роботизовані системи часто мають обмежений обчислювальний ресурс і використання навіть традиційних алгоритмів іноді може бути недоречним через погану часову складність цих алгоритмів. Дані, що надходять у реальному масштабі часу можуть бути оброблені невчасно або взагалі загублені. Таким чином постає питання: який з алгоритмів традиційного виділення точок, що належать зображенню певного об’єкта обрати. Добір має давати достатню точність за умови кращої часової і бажано просторової складності алгоритму.

3 Тема кваліфікаційної роботи відповідного рівня безпосередньо пов'язана з об'єктом діяльності магістра спеціальності 126 «Інформаційні системи та технології» галузі знань 12 «Інформаційні технології» – створення інформаційних технологій різного призначення і застосування.

4 Явища і процеси, що досліджуються в даній кваліфікаційній роботі і обрані для моделювання, оцінювання та реалізації – віднесені в освітньо-кваліфікаційній характеристиці магістрів до класу дослідних та евристичних, рішення яких заснована на знаково-понятійних уміннях.

5 Робота складається з трьох розділів. Перший розділ присвячений аналізу теми дослідження та постановці задачі. У другому розділі наведено проектну складову вирішення завдання. Третій розділ присвячено порівнянню методів кластерізації зображень в умовах обмежених обчислювальних потужностей. Оригінальність отриманих в роботі наукових результатів та їх наукова новизна полягають у наступному:

- досліджено і обгрунтовано застосування відповідних методів кластерізації складних даних;
- детально прорoблено алгоритмічні компоненти методів кластерізації;
- порівняно обрані алгоритми кластерізації;
- обрано два алгоритми (OPTICS і affinity propagation), які відповідають вимогам точності та швидкості.

6 Практичне значення результатів роботи полягає в обранні потрібних для вирішення поставленої задачі алгоритмів на основі порівняння групи найбільш відомих.

7 Практичні результати кваліфікаційної роботи отримані із застосуванням відповідних технічних і програмних засобів, а також програмних продуктів MS Word і MS PowerPoint на інформаційно-технологічній платформі Windows.

8 Оформлення графічних матеріалів до кваліфікаційної роботи рівня магістр виконано на сучасному рівні і відповідає вимогам, що пред'являються до рівня виконання робіт даної кваліфікації.

9 Ступінь самостійності виконання кваліфікаційної роботи достатньо висока.

10 Деякі дискусійні положення та недоліки, які мають місце в роботі:

а) недостатньо повно визначено кінцеву функціональну придатність даної розробки;

б) в цілому, тема розкрита не повністю, бо не зовсім зрозуміло, яким чином будуть реалізовані обрані алгоритми;

в) у роботі мають місце поодинокі орфографічні та стилістичні помилки.

Незважаючи на вищевказані зауваження, кваліфікаційна робота в цілому заслуговує оцінки «_____90_(відмінно)_____» та присвоєння здобувачу відповідної кваліфікації.

Керівник кваліфікаційної роботи,
проф. кафедри ІТКІ, д.т.н. Г.М. Коротенко

ДОДАТОК Г

РЕЦЕНЗІЯ

на комплексну кваліфікаційну роботу рівня магістра
«Розробка інформаційної технології визначення об’єктів у системах машинного зору в умовах обмежених обчислювальних потужностей»
студента групи 126м-22-1 Пачевського Михайла Володимировича

Розглянута робота присвячена підвищення точності визначення об’єкту на зображеннях у системах машинного зору за умови обмеженої продуктивності.

Завдання і зміст кваліфікаційної роботи відповідає головній цілі - перевірці знань і ступеня підготовленості студента за фахом 126 «Інформаційні системи та технології» галузі знань 12 «Інформаційні технології».

Зміст пояснювальної записки кваліфікаційної роботи відповідає необхідним критеріям та затвердженій темі.

Актуальність обраної теми обумовлена тим, що дослідження ефективності застосування інформаційних технологій і відповідних програмних засобів продовжують розвиватися на базі розширення їхнього спектру.

Повнота і глибина вирішення задач, поставлених в завданні на кваліфікаційну роботу є достатньою.

Оформлення пояснювальної записки кваліфікаційної роботи виконано в повній відповідності з діючими стандартами і нормативними вимогами.

Наукова новизна результатів кваліфікаційної роботи роботи визначається тим, що підведено базу під розробку інформаційної технології, що призначена для визначення об’єктів у системах машинного зору в умовах обмежених обчислювальних потужностей.

Практичне значення результатів роботи полягає у порівнянні та обранні найбільш дійових алгоритмів для визначення об’єктів у системах машинного зору в умовах обмежених обчислювальних потужностей.

До числа загальних зауважень і недоліків роботи слід віднести:

1) відсутність реалізації та результатів практичного застосування розробленої технології;

2) дещо спрощений опис кінцевих результатів роботи.

Однак, зазначені зауваження не здійснюють істотного впливу на підсумкові результати кваліфікаційної роботи і не знижують її безумовну практичну та наукову цінність.

Таким чином, слід зробити висновок, що кваліфікаційна робота в цілому

заслуговує оцінки «_____», а її виконавець присвоєння відповідної кваліфікації.

Рецензент, доцент кафедри програмного
забезпечення комп’ютерних систем НТУ
«ДП», к.т.н. А.Л. Ширін