МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра прикладної математики та моделювання складних систем

«До захисту допущено»
Завідувач кафедри
_____ _Ігор КОПЛИК_
(підпис) (Ім'я та ПРІЗВИЩЕ)
___ _____20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 113 «Прикладна математика», освітньо-професійної програми «Наука про дані та моделювання складних систем» на тему:

«Виявлення аномалій в комп'ютерних мережах за допомогою кластеризації»

Здобувача групи ПМ.м–21 Димова Максима Вадимовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Максим ДИМОВ
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник: док. фіз.-мат. наук, професор Олександр ЛИСЕНКО _____
(підпис)

Суми – 2023

АНОТАЦІЯ

Кваліфікаційна робота: 96 сторінок, 14 рисунків, 31 джерело.

Мета роботи: провести аналіз та оцінку ефективності різних методів кластеризації для виявлення аномалій у мережевій активності та кібератак; дослідити вплив параметрів кластеризації на точність виявлення аномалій та кібератак різного типу.

Об’єкт дослідження: датасет «NSL-KDD», що характеризує активність комп'ютерних мереж, процеси кластеризації.

Предмет дослідження: параметри якості кластеризації досліджуваного набору даних, параметри попередньої обробки даних.

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

В кваліфікаційній роботі було проведено аналіз активності комп’ютерних мереж з метою виявлення аномалій та кібератак з використанням даних «NSL-KDD», Canadian Institute for Cybersecurity, а також проведено порівняльний аналіз якості роботи різних методів кластеризації. Особливістю досліджуваних даних було те, що вони містили досить широкий набір аномальних активностей різного типу.

Були використані такі методи: k-means, MeanShift, Spectral Clustering, Agglomerative Clustering, DBSCAN, BIRCH, OPTICS та Gaussian Mixture. Для підвищення якості кластеризації було проведено попередню обробку даних, а саме – проведена мінімаксна нормалізація даних. Така попередня обробка даних дозволяє суттєво підвищити точність (accuracy) розпізнавання кластерів усіх методів.

В результаті, дослідження показали, що найбільш ефективними для виявлення аномалій є такі методи: DBSCAN , Meanshift та k-means. Accuracy таких моделей дорівнювала 91%, 85% та 83% відповідно.

Ключові слова: КЛАСТЕРНИЙ АНАЛІЗ, ЯКІСТЬ КЛАСТЕРНОГО АНАЛІЗУ, K-MEANS, DBSCAN, КІБЕРБЕЗПЕКА.

ЗМІСТ

ВСТУП

Все більше та більше аспектів нашого життя стає залежним від сучасних інформаційних технологій. Зростаюча комп'ютеризація і підключеність до мережі Інтернет приводять до безпрецедентного обсягу обміну даними та інформацією. Однак це також призводить до збільшення загроз кібербезпеці, зокрема, зростає ризик кібератак та інших небажаних подій в комп'ютерних мережах.

Забезпечення безпеки мережі є надзвичайно важливим завданням для забезпечення нормального функціонування сучасного інформаційного середовища. Один із ключових аспектів кібербезпеки – це виявлення аномалій або незвичних активностей, які можуть бути зв'язані з кібератаками, шпигунством, зловмисним програмним забезпеченням чи іншими загрозами.

Ця магістерська робота присвячена розробці та дослідженню алгоритмів для виявлення аномальної активності та кібератак в комп'ютерних мережах за допомогою методів кластеризації. Кластеризація вже доведена своєю ефективністю в аналізі даних та виявленні закономірностей, адже вона дозволяє групувати подібні об'єкти разом. Використання цього підходу для виявлення аномалій може надати значний внесок у розв'язання актуальних проблем кібербезпеки.

У нашій роботі ми розглядаємо важливі аспекти виявлення аномалій, встановлюємо математичні основи методів кластеризації для цієї задачі та розробляємо комп'ютерну модель, яка дозволить провести експерименти та аналіз результатів. Ми детально розглядаємо різні алгоритми кластеризації, їхню ефективність та можливості виявлення аномалій в реальних умовах.

В даній роботі ми прагнемо внести свій власний внесок у розвиток методів кібербезпеки та покращити здатність комп'ютерних мереж виявляти та запобігати небажаній активності. Результати наших досліджень можуть знайти практичне застосування в реальних системах забезпечення кібербезпеки та сприяти створенню більш безпечного інформаційного оточення.

Актуальність дослідження полягає в насущності забезпечення кібербезпеки в умовах зростаючого використання комп'ютерних мереж та інформаційних технологій у всіх сферах сучасного суспільства. З кожним днем значення інформаційних систем та мережевих технологій у нашому житті лише зростає, що створює сприятливі умови для небажаної та зловмисної активності.

Сучасні кіберзагрози стають все більш складними та вдосконаленими, і стандартні методи захисту вже не завжди ефективні. Кібератаки, шпигунство, витоки конфіденційної інформації, а також інші види аномальної активності в мережах можуть спричинити серйозні матеріальні та моральні збитки для користувачів, компаній, установ та навіть держав.

Виявлення аномалій є ключовою складовою сучасних систем кібербезпеки. Застосування методів кластеризації для виявлення аномалій в комп'ютерних мережах дозволить вчасно розпізнавати незвичайну активність та реагувати на потенційні загрози. Це відкриває шлях до покращення надійності та стабільності інформаційних систем, а також захисту конфіденційної інформації користувачів.

Об'єкт дослідження: Методи та алгоритми кластеризації, їх застосування для забезпечення кібербезпеки в комп’ютерних мережах.

Предмет дослідження: Виявлення аномалій в комп'ютерних мережах з використанням методів кластеризації.

Мета дослідження: Метою даної магістерської роботи є розробка та аналіз алгоритмів для виявлення аномальної активності та кібератак в комп'ютерних мережах з використанням методів кластеризації. Робота спрямована на покращення ефективності виявлення аномалій та підвищення рівня кібербезпеки, сприяючи безпечному функціонуванню інформаційних систем.

Завдання дослідження:

1. Аналіз літератури: Провести аналіз наявних методів та підходів до виявлення аномалій в комп'ютерних мережах та визначити їхні переваги та недоліки.
2. Розробка математичної моделі: Розробити математичні основи для методів кластеризації, враховуючи специфіку виявлення аномалій в мережах.
3. Розробка комп'ютерної моделі: Реалізувати алгоритми виявлення аномалій на комп'ютерній платформі, створити програмний інструмент для їх тестування та аналізу.
4. Експерименти та аналіз результатів: Провести експерименти з використанням розроблених алгоритмів на реальних або симульованих даних, оцінити їхню ефективність, точність та здатність виявляти аномалії.
5. Висновки та рекомендації: Сформулювати висновки з результатів дослідження, підкреслити практичне значення розроблених алгоритмів для підвищення рівня кібербезпеки та запропонувати можливі напрямки подальших досліджень.

Ці завдання визначають основні кроки, які планується здійснити під час виконання магістерської роботи з метою досягнення визначеної мети.

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

1.1 Опис предметної галузі

Сучасний світ інформаційних технологій невіддільний від використання комп'ютерних мереж, які є нервовою системою організацій, державних структур, а також приватного сектору. Комп'ютерні мережі є не тільки основою для обміну даними, але й потенційними цілями для кібератак та несанкціонованого проникнення. Виявлення аномалій у комп'ютерних мережах є ключовою задачею в області кібербезпеки, оскільки аномалії можуть бути ознаками збоїв в системі, помилок конфігурації, а також симптомами зловмисних дій.

Задача виявлення аномалій є особливо актуальною в умовах постійного росту обсягів даних та їхньої складності. Традиційні підходи, як-от системи виявлення вторгнень (IDS) та системи виявлення аномалій (ADS), часто вимагають значних ресурсів для аналізу трафіку в реальному часі та мають високий рівень помилкових тривог. Тому розвиток і застосування методів машинного навчання, зокрема алгоритмів кластеризації, стає перспективним напрямком у сфері кібербезпеки.

Кластеризація дозволяє групувати дані на основі їхньої схожості без необхідності попереднього маркування, що робить цей метод особливо вигідним для великих нерозмічених наборів даних. Використання алгоритмів кластеризації, таких як K-means, DBSCAN, OPTICS, та інших, дозволяє виявляти нетипові патерни, що можуть вказувати на аномальну поведінку в мережі.

Ця робота фокусується на дослідженні та застосуванні різноманітних методів кластеризації для виявлення аномалій та кібератак у комп'ютерних мережах. Це дослідження має на меті не тільки виявити ефективні алгоритми для конкретних типів аномалій, але й порівняти їх ефективність, швидкодію, та практичність використання в реальних умовах. Ми розглянемо особливості

кожного з алгоритмів, їх сильні та слабкі сторони, а також специфіку їх застосування до даних різної природи та обсягу.

Таким чином, предметна область даної роботи охоплює теоретичні основи алгоритмів кластеризації, аналіз сучасного стану інструментів виявлення аномалій в мережах та розробку практичних рекомендацій для впровадження цих методів в інфраструктуру кібербезпеки.

Практичне значення:

- Покращення кібербезпеки: результат роботи може допомогти підвищити ефективність виявлення та протидії кібератакам, забезпечуючи більш безпечне цифрове середовище для організацій та індивідуальних користувачів.
- Зниження витрат: автоматизація процесу виявлення аномалій може знизити витрати на моніторинг і аналіз трафіку, оскільки ресурсомісткі традиційні методи потребують значного обсягу ручної роботи і аналізу.
- Підвищення точності реагування: точно виявлення аномалій знижує кількість помилкових тривог, що підвищує довіру до систем безпеки та оптимізує час реакції на істинно небезпечні події.
- Оптимізація ресурсів: алгоритми кластеризації можуть допомогти оптимізувати використання обчислювальних ресурсів шляхом виявлення і відсікання несуттєвих даних та подальшого фокусування на аномальних патернах.
- Зміцнення обороноздатності: впровадження відкритих алгоритмів для виявлення кібератак допомагає зміцнити національну безпеку, особливо в контексті зростаючих кіберзагроз на державному рівні.
- Індустріальна адаптація: результати дослідження можуть сприяти розвитку нових продуктів та рішень для ринку кібербезпеки, що може мати комерційний потенціал.

Враховуючи ці аспекти, можна стверджувати, робота має значний потенціал для внеску у сферу кібербезпеки та захисту інформаційної

інфраструктури, що є актуальним і важливим не тільки на теоретичному, а й на практичному рівні.

1.1.1 Опис типових кібератак

У світі існує велика кількість групувань, що займаються «хакінгом», серед яких є державні актори, які підтримуються урядами, незалежні хакерські колективи, які діють зі своїх власних причин, і злочинні синдикати, які використовують хакінг для отримання прибутку через крадіжку даних, шантаж або інші шахрайські схеми. Важливо зауважити, що деякі з цих груп можуть перетинатися з так званими «хактивістами», які використовують свої навички для просування соціальних або політичних цілей, часто через незаконні або неетичні дії.

На рис.1.1 представлені основні типи атак:

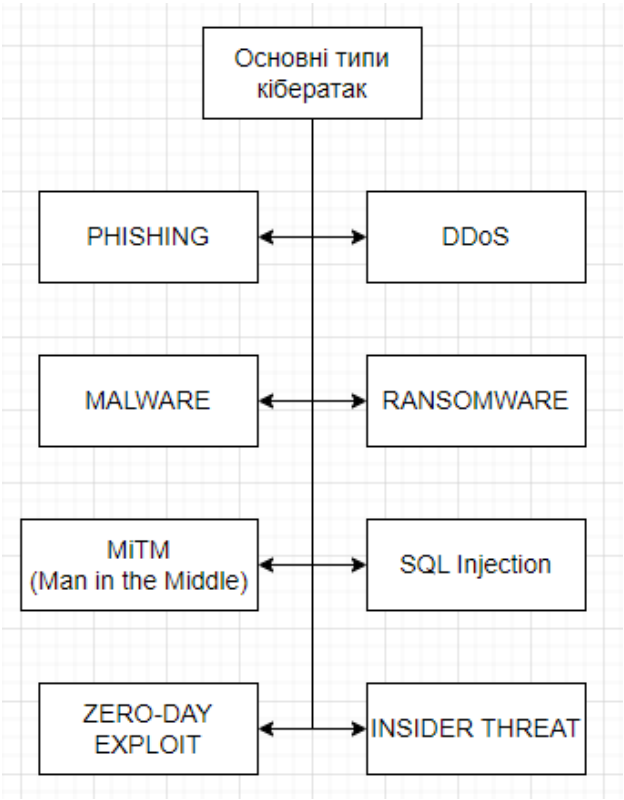

Рисунок 1.1 – Основні типи кібератак

Більш детально розглянемо їх механізми та відповідні засоби протидії:

1. Phishing: зловмисники надсилають електронні листи або повідомлення, які імітують легітимні запити від відомих компаній чи осіб з метою змусити жертв викрити особисту інформацію, таку як паролі та банківські дані. Для захисту застосовують двофакторну аутентифікацію, навчання працівників розпізнавати підозрілі листи, використання антифішингових фільтрів.
2. DDoS (Distributed Denial of Service): зловмисники "засипають" цільову систему, таку як сервери веб-сайтів, несподівано великою кількістю запитів, що призводить до перевантаження системи і її нездатності обслуговувати легітимний трафік. Для захисту застосовують спеціалізовані рішення від DDoS-атак, такі як міжмережеві екрани та системи зміни трафіку.
3. Malware: включає в себе різноманітні шкідливі програми (віруси, троянські коні, шпигунське ПЗ), які можуть бути передані через інфіковані веб-сайти, електронну пошту або носії інформації для крадіжки даних, шпигунства або завдання шкоди. Антивірусне ПЗ може бути захистом, також регулярні оновлення систем та програм, виховання кібергігієни серед користувачів.
4. Ransomware (Програми-вимагачі): Окремий вид малвару, який шифрує файли на зараженому комп'ютері та вимагає викуп за їх розшифровку. Для безпеки застосовують резервне копіювання даних, остережне відкриття електронної пошти та вкладень, використання захисту від вимагань.
5. Man-in-the-Middle (MitM) (Атака "людина посередині"): зловмисник перехоплює комунікації між двома сторонами, наприклад, між користувачем та веб-сайтом, для крадіжки даних або введення фальсифікованої інформації. Для захисту використовують шифроване з'єднання через HTTPS, VPN та використання сертифікатів SSL/TLS.
6. SQL Injection: це вставка шкідливих SQL запитів через інтерфейси веб-сайтів, що дозволяє зловмисникам отримувати несанкціонований

доступ до БД/СУБД. Як захист, впроваджують використання параметризованих запитів, валідацію введення користувачів, регулярне оновлення та аудит баз даних.

7. Zero-Day Exploit: це використання невідомих раніше вразливостей у програмному забезпеченні до того, як розробники випустять виправлення. Запобіжником лише є встановлення оновлень програмного забезпечення якомога швидше та використання систем виявлення вторгнень та превентивних заходів.
8. Insider Threat (Внутрішня загроза): несанкціонований доступ, використання або розкриття інформації компанії працівниками або колишніми співробітниками. Контроль доступу до даних, моніторинг активності користувачів, політика безпеки внутрішньої інформації.

1.1.2 Основні характеристики та параметри мережевої активності

У сфері кібербезпеки, аналіз мережевих даних грає критичну роль у виявленні та протидії кібератакам. Мережі та їх активність можна детально охарактеризувати за допомогою різноманітних параметрів, які дозволяють ідентифікувати нормальну поведінку та виявляти відхилення, що можуть сигналізувати про аномальну активність або потенційні кібератаки. Ось ключові параметри, що використовуються для оцінки мережевої активності:

1. Трафік мережі: кількість даних, що передаються через мережу за певний період. Вимірюється в бітах за секунду (bps), кілобітах за секунду (kbps), мегабітах за секунду (Mbps) та ін.
2. Кількість та типи пакетів, що проходять через мережу. Можуть бути виміряні як кількість пакетів за часовий проміжок (пакетів/сек) або як відсоток певних типів пакетів (наприклад, TCP, UDP, ICMP).
3. Протоколи, що використовуються для комунікації. Деякі протоколи можуть бути асоційовані з легітимною активністю, тоді як інші можуть вказувати на можливі атаки.

4. Використання мережевих портів і статистика з'єднань можуть вказувати на типи служб, які працюють на мережі, і їх активність.
5. Аналіз IP-адрес, з яких походить трафік, може вказувати на джерело трафіку та його легітимність.
6. Час затримки та втрати пакетів: висока затримка або втрати пакетів можуть вказувати на проблеми у мережі або на атаки типу DoS/DDoS.

Характеристики даних, які використовуються для виявлення аномалій та кібератак, можуть включати:

1. Інтенсивність трафіку – раптове збільшення кількості трафіку може бути ознакою атаки типу DoS.
2. Непередбачуване використання портів – незвичайна активність на певних портах може вказувати на сканування портів або спроби вторгнення.
3. Нестандартні протоколи – використання рідко вживаних або небезпечних протоколів може бути ознакою атаки.
4. Аномалії в змісті пакетів – неочікувані зміни в навантаженні пакетів можуть вказувати на вміст шкідливого коду або спробу експлойта.

Щоб зрозуміти, що діяльність може бути аномальною або є кібератакою, можна застосовувати різні аналітичні та виявлювальні системи і аналітичні інструменти, які дозволяють оцінити трафік та поведінку мережі в реальному часі.

Тому для визначення аномальної активності або кібератаки застосовують:

1. Базові статистичні аналізи – мається на увазі порівняння поточної активності з історичними даними для виявлення значних відхилень.
2. Евристичний аналіз – застосування попередньо визначених правил або патернів, щоб ідентифікувати потенційно шкідливу активність.
3. Машинне навчання та аналітика поведінки – застосування алгоритмів для виявлення аномальних патернів, які не відповідають нормальній поведінці мережі.

4. Сигнатурний аналіз – порівняння мережевих подій з базою даних вже відомих підписів атак. [1]

1.2 Постановка задачі кластеризації

Постановка задачі кластеризації є важливим етапом в аналізі даних, який визначає подальший хід дослідження та вибір методів. Одним із ключових аспектів є визначення кількості кластерів у вхідних даних. Згідно з дослідженням Jain A. K. [2], вибір оптимальної кількості кластерів може бути складною задачею і вимагає врахування специфіки даних та мети кластеризації. У реальних задачах, вибір кількості кластерів може відбуватися на основі апріорної інформації або вимагати використання евристичних підходів.

Крім того, важливо враховувати типи ознак у вхідних даних. Наприклад, при аналізі текстових даних, важливо вибрати відповідний підхід до обробки текстових репрезентацій та метрик схожості. В роботі Aggarwal C. C. та Reddy C. K. [3] розглядається проблема обробки та кластеризації текстових даних, в тому числі використання методів з різними метриками схожості та підходами до векторизації текстів.

Відповідно до вищезазначеного, постановка задачі кластеризації вимагає уважного аналізу характеристик даних та вибору оптимальних параметрів для успішної реалізації методів кластеризації.

Для належного розуміння постановки задачі кластеризації необхідно визначити основні поняття, що лежать в основі цього підходу до аналізу даних.

Об'єкт дослідження: Об'єкт дослідження є множина X, яка складається з n об'єктів, представлених у вигляді векторів ознак. Кожен об'єкт $X = \{x_n\}$ може бути, наприклад, спостереженням у комп'ютерній мережі чи даними, які характеризують інший вид активності.

Множина даних: Множина даних X представляє собою збірку об'єктів, що вивчаються. Кожен об'єкт може бути описаний різними ознаками, які характеризують його властивості.

Кластер: Кластер є підмножиною об'єктів, які є схожими один на одного в рамках певних критеріїв схожості. Ідея полягає в тому, щоб об'єднати об'єкти, які мають близькі значення ознак, у відповідні групи.

Міра схожості: Міра схожості визначає ступінь близькості між двома об'єктами. Вона може бути розрахована за допомогою різних методів та метрик, таких як Евклідова відстань, косинусна схожість, кореляція та інші.

Функція відстані: Функція відстані обчислює відстань або схожість між двома об'єктами. Вона є основною складовою методів кластеризації та визначає спосіб порівняння об'єктів у просторі ознак.

Відомості про ці основні поняття є необхідними для належного розуміння завдання кластеризації та методів, які використовуються для його розв'язання.

Формалізація задачі кластеризації передбачає визначення конкретних складових елементів, що входять у її структуру, та параметрів, які визначають її характер.

Нехай задана множина об'єктів дослідження $X = \{x_1, x_2, \dots, x_n\}$, де x_i – це m-вимірний вектор ознак, що характеризує і-й об'єкт. Метою задачі є розділення цих об'єктів на K неперетинаючих підмножин $C = \{C_1, C_2, \dots, C_K\}$, де кожний кластер C_K містить об'єкти, що є схожими між собою.

Формально, задача кластеризації може бути визначена наступним чином.

Задано:

1. Множина об'єктів дослідження, що належать до множини дійсних чисел R^m:

$$X = \{x_1, x_2, \dots, x_n\}$$

де n – кількість об’єктів,

2. Кількість кластерів K.
3. Потрібно знайти:

Множину підмножин

$$C = \{C_1, C_2, \dots, C_K\}$$

де C_K – підмножина множини X для $k = 1, 2, \dots, K$

Відображення f: $X \rightarrow \{1, 2, \dots, K\}$:

Це функція, яка відображає об’єкти X у відповідні кластери, де $f(x_i) = k$ означає, що об’єкт x_i належить до кластера C_k.

Ця формалізація визначає основну структуру задачі кластеризації, де кожен кластер містить групу об'єктів, а метою є розділення об'єктів на такі групи, щоб об'єкти всередині кожного кластера були максимально схожі, а об'єкти різних кластерів – якомога відмінні.

Задача кластеризації має різні варіації та модифікації, які виникають в залежності від конкретних умов та вимог дослідження. Деякі з цих варіацій враховують додаткові обмеження, особливості даних чи специфічні завдання аналізу [4].

Обмеження розмірів кластерів: У деяких випадках, важливо встановити обмеження на кількість об'єктів у кожному кластері. Наприклад, може виникнути необхідність визначити кластери з певним мінімальним або максимальним числом об'єктів.

Ієрархічна кластеризація: В цьому варіанті задачі об'єкти групуються в ієрархічну структуру кластерів, де кожен кластер може включати в себе підкластери. Цей підхід дозволяє отримати докладний огляд структури даних на різних рівнях деталей.

Кластеризація з врахуванням вагових коефіцієнтів: В деяких випадках, ознаки можуть мати різну важливість або значущість. Така варіація дозволяє враховувати вагові коефіцієнти для ознак при обчисленні схожості та побудові кластерів.

М'яка кластеризація: У цьому підході кожен об'єкт може належати до декількох кластерів з різними ступенями належності. Ця варіація може бути корисною, коли об'єкти можуть мати багатозначну належність до різних груп.

Кластеризація відкритих даних: У випадках, коли кількість кластерів невідома заздалегідь, може виникнути необхідність у розробці методів для автоматичного виявлення оптимальної кількості кластерів, таких як методи засновані на аналізі зміни міри схожості.

Ці варіації задачі кластеризації відображають різноманітність підходів до аналізу даних та надають можливість вибору найбільш підходящого методу для конкретних завдань дослідження.

1.3. Типологія задач кластеризації

У дослідженні і застосуванні методів кластеризації виникає необхідність вибору підходів відповідно до характеристик даних та завдань аналізу. Методи кластеризації можуть бути розділені на різні категорії, такі як ієрархічні, центроїдні, основані на графах, ймовірнісні тощо. У роботі Shaojun Huang et al. [5] досліджуються методи кластеризації з використанням центроїдних підходів, таких як метод K-means та метод Gaussian Mixture Model (GMM). Дослідження показали, що метод GMM може бути більш ефективним для даних з комплексними структурами.

Порівняння різних методів кластеризації на реальних даних може надати важливу інформацію про їх ефективність та властивості. У роботі Sabzi, S. et al. [6] проведено порівняння різних методів кластеризації на основі метрик схожості та структури даних. Результати дослідження показали, що деякі методи можуть бути більш адаптованими до конкретних типів даних.

Також слід зазначити, що залежно від властивостей даних, один метод кластеризації може бути більш ефективним за інший. Тому, вибір методу кластеризації та його параметрів є важливим завданням при аналізі даних.

Залежно від того, чи маємо ми апріорну інформацію про кількість та характер кластерів у даних, задачі кластеризації можуть бути розділені на декілька типів.

Фіксована кількість кластерів: У цьому випадку передбачається, що кількість кластерів K відома заздалегідь. Метою є розділення об'єктів на K

груп, кожна з яких відображає конкретний кластер. Такий підхід часто використовується в випадках, коли існують чіткі знання про кількість категорій або класів, на які можна поділити дані [7].

Невідома кількість кластерів: В цьому варіанті кластеризації кількість кластерів K є невідомою та підлягає визначенню шляхом аналізу даних. Методи такої кластеризації намагаються знайти оптимальну кількість кластерів, враховуючи структуру даних та взаємодію об'єктів.

Кластеризація з використанням апріорної інформації: У деяких випадках, наявність додаткових даних або знань про кластери може сприяти покращенню якості кластеризації. Наприклад, використання міток класів або експертної інформації може допомогти зробити кластеризацію більш точною та структурованою [8].

Кожен з цих типів кластеризації має свої особливості та підходи до розв'язання. Вибір конкретного типу залежить від природи даних та мети аналізу, і від нього залежатиме вибір методів та алгоритмів для вирішення задачі кластеризації.

Кластеризація з урахуванням природи даних відображає підхід до аналізу, де акцент робиться на особливостях та властивостях даних, які можуть впливати на вибір методів та підходів до кластеризації. В залежності від природи даних та їх розподілу, можуть виникати різні типи задач кластеризації.

Особливості вибірки та розподілу: В деяких випадках дані можуть мати нерівномірний розподіл, або можуть бути відомі апріорні властивості розподілу. Такі особливості можуть вплинути на вибір метрик схожості, відстаней між об'єктами, а також на підхід до побудови кластерів [5].

Випадковість та шум: Деякі набори даних можуть містити шум, тобто об'єкти, які не належать жодному кластеру, або додаткові варіації у даних. Кластеризація з урахуванням природи даних може включати в себе методи, які допомагають ідентифікувати та обробляти шумові об'єкти.

Асиметрія та важливість ознак: В деяких випадках, окремі ознаки можуть мати більшу або меншу вагу при кластеризації. Може виникнути

необхідність нормалізувати дані або використовувати вагові коефіцієнти для підрахунку схожості.

Неоднорідність та внутрішня структура: Деякі набори даних можуть мати внутрішню структуру, де кластери можуть бути розташовані на різних віддаленостях або мати неоднорідність у розміщенні. Кластеризація з урахуванням природи даних може включати методи, які допомагають виявляти ці внутрішні зв'язки [3].

Ці аспекти показують, що при вирішенні задачі кластеризації важливо аналізувати природу даних, їх розподіл та властивості, щоб вибрати відповідний метод та підхід до розв'язання задачі.

Кластеризація з використанням додаткової інформації є підхід до аналізу даних, де додаткові знання або інформація про кластери додаються до процесу кластеризації з метою покращити якість та релевантність отриманих результатів. Цей підхід може збільшити точність та інтерпретованість кластерів, а також допомогти уникнути невірних або неочікуваних результатів.

Використання міток класів: У деяких випадках можуть бути доступні мітки класів для об'єктів, що вказують на їх приналежність до певних категорій або класів. Ця додаткова інформація може використовуватися під час кластеризації для підтримки та підвищення точності результатів [9].

Експертна інформація: В деяких випадках експертна інформація про взаємозв'язки між об'єктами може бути використана для побудови кластерів. Експертні знання можуть включати в себе інсайти про природу даних, схожість об'єктів, або інші параметри, які можуть бути важливими для кластеризації.

Додаткові ознаки та відносини: У деяких випадках можуть бути доступні додаткові ознаки або дані, які можуть допомогти виявити внутрішні зв'язки між об'єктами. Наприклад, соціальні мережі можуть надавати інформацію про зв'язки між користувачами, яка може бути використана для кращої ідентифікації кластерів [10].

Інші джерела інформації: Додаткова інформація може надходити з різних джерел, таких як географічні дані, часові ряди, додаткові атрибути тощо.

Використання цих даних під час кластеризації може збагатити аналіз та забезпечити більш деталізовану картину структури даних.

Застосування додаткової інформації в процесі кластеризації може сприяти покращенню якості та зрозумілості результатів, а також допомогти виявити внутрішні зв'язки та закономірності, які можуть бути приховані в даних [11].

1.4 Методи кластеризації

Оцінка якості кластеризації є важливим етапом у виборі найкращого методу для конкретних даних. Для цього використовуються різні метрики, які дозволяють кількісно оцінити рівень роздільності та структуру сформованих кластерів. У роботі Rokach, L. et al. [9] досліджено різні метрики оцінки якості кластеризації та їх вплив на вибір оптимальних методів.

Однією з найбільш використовуваних метрик є внутрішня індексна метрика, така як коефіцієнт силуету. Ця метрика дозволяє оцінити, наскільки об'єкти в одному кластері схожі між собою, порівняно з об'єктами з інших кластерів. У роботі Awate, S. P. et al. [11] проведено дослідження використання різних метрик для оцінки якості кластеризації на прикладі медичних зображень.

Порівняльний аналіз різних методів оцінки якості кластеризації та їх вплив на вибір оптимальних методів дозволяє зробити обґрунтований вибір методу для конкретного завдання [5].

Алгоритми ієрархічної кластеризації є одним із способів побудови кластерів, використовуючи деревоподібну структуру. Ці алгоритми поділяються на два основних типи: агломеративні та дивізивні. Обидва типи спрямовані на ітеративне об'єднання або розбиття кластерів залежно від схожості між об'єктами [2].

Агломеративні алгоритми: Агломеративні методи починають з кожного об'єкта у власному кластері та послідовно об'єднують найбільш схожі кластери, поки не буде досягнута одна велика група, що містить всі об'єкти.

Під час об'єднання використовуються різні метрики схожості, такі як відстань між середніми, одиночна відстань та інші. Агломеративні методи формують ієрархічну структуру, яку можна подати у вигляді дерева, відомого як дендрограма.

Дивізивні алгоритми: Дивізивні методи діють навпаки, починаючи з одного великого кластеру, який поділяється на менші підкластери на кожному кроці. Ці методи допомагають з'ясувати, які об'єкти найкраще розділити, щоб утворити окремі кластери. Хоча дивізивні методи менш поширені, вони можуть бути корисними в деяких випадках.

Перевагою алгоритмів ієрархічної кластеризації є їхня здатність відображати структуру даних у вигляді дерева, що дозволяє візуалізувати та інтерпретувати кластери. Однак, велика обчислювальна складність та висока вимога до пам'яті можуть бути обмеженнями при роботі з великими наборами даних.

1.4.1 Алгоритми ієрархічної кластеризації

Алгоритми ієрархічної кластеризації зазвичай використовуються, коли важливим є виявлення структури в даних та її візуалізація, а також коли кількість кластерів не обмежена жорсткою границею.

Алгоритми кластеризації на основі квадратичної помилки є одними з найбільш популярних та широко використовуваних підходів. Ці методи ставлять за мету мінімізувати суму квадратів відстаней між об'єктами та центрами їхніх кластерів. Основним прикладом такого підходу є метод K-means [12], який широко використовується для різних завдань кластеризації.

Метод K-means: Метод K-means спробує розділити дані на K кластерів, де K - попередньо визначена кількість. Цей алгоритм виконується черговою мінімізацією функції внутрішньокластерної дисперсії, тобто суми квадратів відстаней між об'єктами та центрами їхніх відповідних кластерів. Кожен об'єкт призначається до того кластера, центр якого знаходиться найближче до нього.

Метод K-medoids: Відмінністю методу K-medoids є те, що замість центрів кластерів використовуються фактичні об'єкти з набору даних. Ці

об'єкти називаються медоїдами, і вони представляють центр кластера, який мінімізує відстані між собою та іншими об'єктами у своєму кластері [13].

1.4.2 Алгоритми квадратичної помилки

Алгоритми кластеризації на основі квадратичної помилки є важливими інструментами для групування даних у кластери, де основною метою є мінімізація суми квадратів відстаней між об'єктами та їхніми центрами кластерів. Ці алгоритми засновані на ідеї знаходження оптимальних центрів (центроїдів) для кожного кластера, які найкраще відображають середню характеристику об'єктів у кластері.

Дану функцію можна відобразити наступним чином:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y_i})^2$$

де L – значення квадратчиної помилки,

y_i – спостережуване значення для i-го спостереження,

N – загальна кількість спостережень.

Метод K-means: Один із найбільш відомих алгоритмів кластеризації на основі квадратичної помилки - метод K-means. Цей алгоритм включає в себе кілька ітераційних кроків, під час яких об'єкти призначаються до найближчого центру кластера, а потім центри перераховуються як середнє значення об'єктів у кожному кластері. Процес повторюється до збіжності, коли зміни між ітераціями стають мінімальними [14].

Метод K-medoids: Метод K-medoids є варіантом K-means, де замість центрів використовуються фактичні об'єкти з набору даних як центри кластерів. Це зроблено для зменшення впливу викидів та аномалій на кінцевий результат кластеризації. Кожен об'єкт, який слугує центроїдом, називається медоїдом [15].

Алгоритми кластеризації на основі квадратичної помилки допомагають згуртувати схожі об'єкти в кластери, але вони вимагають попередньої вказівки кількості кластерів, а також можуть залежати від початкового розташування

центрів кластерів. Вони залишаються популярними інструментами для виявлення структури в даних, побудови профілів кластерів та знаходження характеристик груп об'єктів.

1.4.3 Нечіткі алгоритми

Нечіткі алгоритми кластеризації є потужними інструментами для розв'язання завдань групування, де об'єкти можуть належати до декількох кластерів одночасно з різними ступенями належності. Ці алгоритми використовують теорію нечітких множин для врахування неоднозначності та невизначеності при призначенні об'єктів до кластерів.

Метод нечіткої с-середньої: Цей метод є розширенням методу K-means для нечітких даних. Він призначає кожному об'єкту ваговий вектор членства в кластерах, який показує, наскільки даний об'єкт належить до кожного кластера. Після цього центри кластерів перераховуються з урахуванням вагових членств об'єктів. Виглядає це таким чином:

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\|x_i - v_j\|}{\|x_i - v_k\|}\right)^{\frac{2}{m-1}}}$$

де c – кількість кластерів,

v_j – центр кластера c_j,

m – параметр «розмитості» (зазвичай значення > 1),

$\|x_i - v_j\|$ – відстань між об’єктом x_i та центром кластера c_j,

$\|x_i - v_k\|$ – відстань між об’єктом x_i та центром кластера c_k.

Метод нечіткої агломеративної кластеризації: Цей метод базується на агломеративній кластеризації, але враховує нечітку природу даних. Він об'єднує кластери на основі нечіткої схожості, яка враховує ступінь подібності між кластерами [16].

Нечіткі алгоритми кластеризації можуть бути корисними, коли об'єкти можуть мати багатозначну належність до різних кластерів, або коли потрібно враховувати різні ступені схожості між об'єктами. Вони знайшли застосування

в областях, де важливо враховувати неоднозначність та невизначеність в даних, таких як аналіз зображень, обробка природних мов, або оцінка експертних думок.

1.4.4 Алгоритми, засновані на теорії графів

Алгоритми, засновані на теорії графів, є інноваційним і важливим підходом до кластеризації, який використовує структуру та взаємозв'язки між об'єктами для групування даних. Для кожного методу формули можуть значно відрізнятися в залежності від завдання, проте основною для алгоритму вважаться:

$$G = (V, E)$$

де V – множина вершин, E – мжожина ребер, в якому кожне ребро має вагу.

У цих методах об'єкти розглядаються як вузли графа, а взаємозв'язки між ними - як ребра, що дозволяє використовувати поняття графів для виявлення кластерної структури.

Спектральна кластеризація: Спектральна кластеризація використовує спектральні властивості графа для групування об'єктів. Граф побудований так, що вузли відповідають об'єктам, а ребра відображають ступінь схожості між об'єктами. Застосовуючи методи аналізу спектральних властивостей матриці схожості графа, можна отримати кластери.

Метод знаходження спільнот у графі: Цей метод визначає групи об'єктів, які взаємодіють або співпрацюють один з одним у складі графа. Він може бути застосований до соціальних мереж, де вузли представляють користувачів, а ребра - взаємодії між ними [17].

Алгоритми, засновані на теорії графів, дозволяють виявити складні взаємозв'язки та залежності між об'єктами, що може бути корисним для виявлення кластерної структури в даних. Вони також можуть бути використані для виявлення груп об'єктів з подібними характеристиками або поведінкою в графових структурах.

1.4.5 Алгоритм виділення зв'язкових компонент

Алгоритм виділення зв'язкових компонент є спеціалізованим методом для виявлення груп об'єктів у графах, де об'єкти утворюють підмножини, де кожен об'єкт має принаймні одне з'єднання з іншими об'єктами у кластері.

Алгоритм намагається знайти всі "зв'язкові компоненти" – підмножини вузлів графа, які взаємодіють між собою, але мають обмежені зв'язки з іншими компонентами. Ці компоненти можуть бути представлені як групи об'єктів, що мають сильні внутрішні зв'язки, але слабкі або обмежені зв'язки з іншими групами [10].

Алгоритм виділення зв'язкових компонент може бути особливо корисним у великих графах, таких як соціальні мережі або мережі співпраці, де важливо виділити підмножини вузлів зі спільними зв'язками. Він також може знайти застосування у задачах групування даних за їхніми зв'язками та взаємодіями.

1.4.6 Алгоритм мінімального покриваючого дерева

Алгоритм мінімального покриваючого дерева є ефективним методом для виявлення структури в графі, де об'єкти взаємодіють між собою через зв'язки, які можна представити у вигляді дерева. Метою цього алгоритму є знайти дерево, яке об'єднує всі вузли графа і має найменшу суму ваг ребер.

Алгоритм вибирає ребра з графа в порядку зростання їхніх ваг, при цьому уникнення циклів. Кожен наступний вибір ребра додається до дерева, яке поступово росте, доки всі вузли не будуть покриті. Це призводить до створення дерева з найменшою сумою ваг ребер, що є оптимальним способом виявлення зв'язків та структури в графі [18].

Алгоритм мінімального покриваючого дерева знайшов широке застосування в різних областях, таких як транспортне планування, мережі співпраці, аналіз даних та інші. Він допомагає виявити ключові зв'язки та

структуру в даних, де важливо знайти найоптимальніший шлях між вузлами або об'єднати їх у мінімальну ієрархію.

Також він використовує алгоритм Прима та бібліотеку heapq для ефективного вибору ребер з найменшою вагою. Програма виводить мінімальне покриваюче дерево, де кожне ребро показує з'єднання між вершинами та вагу ребра [5].

1.5 Порівняння алгоритмів

Порівняння різних алгоритмів кластеризації є важливим кроком для визначення найефективнішого підходу до обробки конкретних даних. Це допомагає зрозуміти переваги та недоліки кожного методу і вибрати найкращий для вирішення певної задачі.

У роботі Jain, A. K. [18] проведено обширний аналіз різних методів кластеризації, включаючи ієрархічну кластеризацію, метод k-середніх, DBSCAN та інші. Автори використовували різні набори даних, включаючи винні дані, медичні зображення та тексти, для оцінки продуктивності алгоритмів у різних сценаріях.

Дослідження показали, що найефективніший метод може залежати від характеристик даних, таких як розмір вибірки, кількість кластерів та їх геометрична структура. Наприклад, для даних з нелінійною структурою найкраще може підійти метод DBSCAN, тоді як для даних з гаусівськими розподілами краще підходить метод k-середніх.

Важливим аспектом порівняння алгоритмів є обрання правильних метрик для оцінки якості кластеризації, таких як коефіцієнт силуету, індекс Данна та інші. Аналіз різних метрик та їх вплив на результати порівняння дозволяє зробити об'єктивний висновок про ефективність кожного методу.

Метрики якості кластеризації відіграють важливу роль у порівнянні різних алгоритмів і визначенні їхньої ефективності у відокремленні та групуванні даних. Ці метрики допомагають оцінити, наскільки добре кластери відповідають структурі даних та реальним зв'язкам між об'єктами. Наприклад:

1. Adjusted Rand Index (ARI): ARI є однією з найпоширеніших метрик для порівняння кластерів. Він враховує як збіги, так і відмінності між справжніми мітками кластерів та мітками, отриманими від алгоритму. ARI надає значення в діапазоні від -1 (незадовільний результат) до 1 (ідеальний результат), де значення навколо нуля вказують на випадковий вибір кластерів [13].
2. Гомогенність, повнота, V-міра: метрики, які дозволяють оцінити ступінь однорідності та повноти кластерів. Гомогенність вимірює, наскільки об'єкти в одному кластері подібні між собою, тоді як повнота вказує, наскільки об'єкти з одного справжнього кластера об'єднані в одному прогнозованому кластері. V-міра об'єднує ці дві метрики, надаючи комплексну оцінку якості кластеризації.
3. Силует: метрика, яка оцінює якість розділення кластерів та відстань між ними. Вона враховує, наскільки об'єкти в одному кластері подібні між собою порівняно з іншими кластерами. Вищі значення силуету вказують на кращу якість кластеризації.

Метрики якості кластеризації допомагають об'єктивно оцінити рівень подібності та відмінності між кластерами, дозволяючи вибрати найкращий алгоритм для конкретної задачі.

Для оцінки та порівняння ефективності різних алгоритмів кластеризації був проведений комп'ютерний експеримент, в рамках якого використовувалися як штучно створені набори даних, так і реальні дані.

Штучно створені набори даних: Для перевірки алгоритмів на відповідність та стійкість були створені випадкові набори даних з різними структурами та характеристиками. Це дозволило дослідникам перевірити, як кожен алгоритм впорядковує об'єкти у кластери на різних типах даних.

В процесі експерименту для кожного алгоритму були отримані результати кластеризації для кожного набору даних. Ці результати включали в себе набір міток кластерів для кожного об'єкта та оцінки метрик якості кластеризації, такі як ARI, гомогенність, повнота, V-міра та силует.

Зібрані результати порівняння різних алгоритмів кластеризації були піддані детальному аналізу з метою визначення найефективніших методів для групування даних. Цей аналіз проводився з урахуванням різних метрик якості кластеризації, які були описані у підпункті 1.4.1, а також інших показників, що відображають структуру та зв'язки в кластерах.

В рамках аналізу було оцінено якість кластерів, їхню однорідність та повноту. Зокрема, метрики гомогенності, повноти та V-міри були використані для визначення ступеня схожості та повноти кластерів. Крім того, було розглянуто метрику силуету, яка оцінює внутрішню та зовнішню віддаль між кластерами.

1.6. Метрики якості кластеризації

1.6.1 Adjusted Rand Index (ARI)

Adjusted Rand Index (ARI) є однією з ключових метрик якості кластеризації, що використовується для вимірювання подібності між справжніми мітками кластерів та мітками, отриманими в результаті кластеризації алгоритмом. Ця метрика надає можливість оцінити, наскільки добре алгоритм зміг відобразити структуру даних у вигляді кластерів.

ARI підтримується в діапазоні від -1 до 1, де значення -1 вказує на абсолютно випадковий вибір кластерів, а значення 1 показує ідеальний збіг міток кластерів. Значення навколо нуля свідчать про те, що кластеризація відбулась випадковим чином.

Формула для обчислення ARI базується на кількості пар об'єктів, які знаходяться в одному кластері в справжньому розподілі та в алгоритмі:

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

Також враховується кількість пар об'єктів, які знаходяться в різних кластерах в обох розподілах. Використовуючи ці значення, ARI розраховується для визначення ступеня узгодженості міток кластерів [6].

ARI полягає в тому, що вона враховує випадковий вибір міток, що може відбутися при розрахунку інших метрик. Однак варто зазначити, що ARI може бути чутливий до розмірів кластерів та може давати низькі значення для великих кластерів, незалежно від їхньої справжньої якості.

1.6.2 Гомогенність, повнота, V-міра

Гомогенність, повнота та V-міра є іншими важливими метриками якості кластеризації, які допомагають оцінити ступінь однорідності та повноти кластерів. Ці метрики надають більш детальний розгляд того, наскільки добре кластери відповідають справжнім структурам даних.

Гомогенність (Homogeneity): Гомогенність вимірює ступінь, до якої об'єкти в одному кластері подібні між собою з точки зору справжнього розподілу. Висока гомогенність вказує на те, що кожен кластер включає об'єкти з одного справжнього класу. Гомогенність може бути обчислена за допомогою формули, яка враховує внутрішньокластерні пари об'єктів з одного справжнього класу.

Повнота (Completeness): Повнота вимірює ступінь, до якої всі об'єкти з одного справжнього класу входять в один кластер. Висока повнота вказує на те, що жодний об'єкт з одного справжнього класу не був розбитий між кластерами. Повнота може бути обчислена за допомогою формули, яка враховує внутрішньокластерні пари об'єктів з одного справжнього класу [19].

V-міра (V-Measure): V-міра об'єднує гомогенність та повноту для надання комплексної оцінки якості кластеризації. Вона підраховується як гармонічне середнє між гомогенністю та повнотою. V-міра враховує як якість кластерів, так і зв'язки між ними, і надає високе значення там, де і гомогенність, і повнота високі.

Наступна формула дозволить оцінити структуру даних:

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

де C – кластери, що створені алгоритмом кластеризації,

K – відомі класи або інші інфоджерела.

Ці метрики дозволяють краще зрозуміти, наскільки кластери відображають справжні структури даних, і допомагають оцінити їхню якість в контексті конкретної задачі. Вони доповнюють метрику ARI, надаючи більш детальний погляд на результати кластеризації [20].

1.6.3 Силует

Силует є ще однією важливою метрикою якості кластеризації, яка використовується для оцінки внутрішньої якості кластерів та розміщення об'єктів всередині кластерів. Вона надає можливість кількісно виміряти те, наскільки кожен об'єкт знаходиться ближче до інших об'єктів у своєму кластері, порівняно з об'єктами інших кластерів.

Для кожного об'єкта визначається два параметри: a - середній відстані між об'єктом та іншими об'єктами у тому ж кластері, та b - середній відстані між об'єктом та об'єктами іншого найближчого кластера (якщо об'єкт належить до кластера, відмінного від свого). Силует обчислюється за допомогою формули:

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

де a – середня відстань від поточного об’єкта до інших з того самого кластера,

b – середня відстань від поточного об’єкта до інших з найближчих кластерів,

S – варіюється від -1 до 1. Високе значення силуету вказує на те, що об'єкт розташований далеко від інших кластерів та близько до інших об'єктів свого кластера.

Силует може бути використаний для вибору оптимальної кількості кластерів, оскільки вона допомагає виявити найкращу структуру даних. Велике значення силуету вказує на добре розділені та однорідні кластери.

Однак важливо зазначити, що силует не завжди підходить для всіх типів даних та структур кластерів і може давати недостатньо інформації у випадках, коли кластери мають нетривіальну форму або зсунуті структури [21].

У нашому дослідженні силует допоможе визначити, як добре об'єкти розміщені всередині кластерів та наскільки кластери є однорідними. Ця метрика підтримає аналіз та порівняння ефективності різних алгоритмів кластеризації на основі їхньої внутрішньої якості.

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

2.1. K-means

K-means є алгоритмом кластеризації, який базується на центральних методах і використовує поняття центроїда для групування об'єктів у кластери. Основна ідея алгоритму полягає в послідовному виборі центроїдів, призначенні об'єктів до кластерів, перерахунку центроїдів та ітераціях до досягнення збіжності.

Ініціалізація центроїдів: Перший крок алгоритму – це ініціалізація центроїдів для кожного з k кластерів. Це може бути випадковий вибір k об'єктів з даних або використання інших методів, наприклад, методу k-means++.

Призначення до кластерів: На наступному етапі кожен об'єкт даних призначається до найближчого центроїда. Відстань може обчислюватися, наприклад, за допомогою Евклідової відстані або інших метрик схожості.

Перерахування центроїдів: Після призначення об'єктів до кластерів, центроїди кожного кластера перераховуються як середнє значення координат об'єктів, що входять до цього кластера.

Ітерації: Процес призначення та перерахунку центроїдів повторюється у вигляді ітерацій до досягнення збіжності. Збіжність може бути визначена за певною умовою, наприклад, коли зміни у центроїдах між ітераціями стають незначними [17].

Алгоритм K-means завершує свою роботу, коли досягнута збіжність, і кожен об'єкт призначений до одного з k кластерів. Цей алгоритм може бути ефективним для великих об'ємів даних та коли кластери мають виразні центроїди, але він також може бути чутливим до початкового розташування центроїдів і не завжди добре справляється з кластеризацією даних із складними структурами.

K-means є одним із найпоширеніших алгоритмів кластеризації, і він має як свої переваги, так і недоліки.

Переваги K-means:

1. Швидкість та простота реалізації: Однією з основних переваг K-means є його швидкість та відносно проста реалізація. Він добре масштабується на великі обсяги даних, що робить його відмінним вибором для великих наборів даних.
2. Ефективність на роздільних кластерах: K-means працює добре, коли кластери мають виразно виокремлені центроїди та добре відокремлені від інших кластерів.

Недоліки K-means:

1. Чутливість до початкових значень центроїдів: Вибір початкових значень центроїдів може вплинути на кінцевий результат. Якщо початкові значення вибрані неоптимально, алгоритм може збігтися до підоптимального розв'язку.
2. Неспроможність вирішити задачу з неоднаковими розмірами та густотами кластерів: K-means передбачає, що кластери мають однакову густоту та розмір. У випадках, коли кластери мають різні розміри та густоти, K-means може давати незадовільні результати.

Незважаючи на ці недоліки, K-means залишається потужним інструментом для багатьох завдань кластеризації, особливо для даних з досить чіткими та окремими кластерами. Однак, для більш складних завдань та даних з неоднорідною структурою, можуть бути використані більш розширені алгоритми, які будуть розглянуті в подальших розділах.

Алгоритм K-means має великий потенціал у сфері аналізу даних та виявлення прихованих структур. У нашому дослідженні ми плануємо застосувати K-means для кластеризації даних, зокрема для наборів даних, та штучно створених наборів даних [10].

Метою нашого дослідження є виявлення прихованих залежностей та групування об'єктів за схожістю. Алгоритм K-means, завдяки своїм характеристикам швидкості та простоти, є привабливим вибором для цього

завдання. Його можна легко застосовувати до великих обсягів даних, що дозволить нам ефективно аналізувати.

Приклад коду для використання алгоритму K-means на мові програмування Python з використанням бібліотеки scikit-learn:

```
# Імпортуємо необхідні бібліотеки
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Створюємо випадкові дані для прикладу
np.random.seed(0)
X = np.array([[1, 2], [1.5, 1.8], [5, 8], [8, 8], [1, 0.6], [9, 11]])

# Визначаємо кількість кластерів
num_clusters = 2

# Ініціалізуємо K-means алгоритм
kmeans = KMeans(n_clusters=num_clusters)

# Застосовуємо K-means до даних
kmeans.fit(X)

# Отримуємо координати центроїдів та призначення кластерів для кожного об'єкта
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

# Виводимо результати
print("Центроїди кластерів:", centroids)
print("Призначення кластерів:", labels)

# Візуалізуємо результати
colors = ["g.", "r.", "c.", "y."]

for i in range(len(X)):
    plt.plot(X[i][0], X[i][1], colors[labels[i]], markersize=10)

plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=150, linewidths=5,
zorder=10)
plt.show()
```

Цей код створює випадкові дані, застосовує алгоритм K-means до цих даних та візуалізує результати, показуючи призначення кожного об'єкта до кластерів та розташування центроїдів. Будь ласка, зауважте, що це лише простий приклад, і реальна робота з даними може вимагати додаткового налаштування та обробки [6].

2.2. Affinity propagation

Affinity propagation (AP) – це захоплюючий підхід до кластеризації даних, який відрізняється своєю неповторною концепцією. Незвичайність AP полягає в тому, що він не зводиться до класичного призначення об'єктів до кластерів, а заснований на визначенні взаємодії між об'єктами, що називаються "епіцентрами". Ця ідея дозволяє “Affinity propagation” виявляти структуру даних за допомогою внутрішньокластерної взаємодії та вибору представників кожного кластера.

Основні концепції “Affinity propagation”:

Епіцентри кластерів: Замість призначення кожного об'єкта до конкретного кластера, як це робиться в інших алгоритмах, AP визначає обмежену кількість "епіцентрів" або "схильностей" – об'єктів, які найкраще представляють кожен кластер.

Подібність та доступність: AP використовує два типи значень – схожість (affinity) та доступність (availability) – для кожної пари об'єктів. Схожість відображає міру подібності між об'єктами, тоді як доступність відображає готовність об'єкта стати епіцентром.

Взаємодія та визначення епіцентрів: AP використовує ітеративний підхід, під час якого об'єкти взаємодіють, обчислюючи схожість та доступність один для одного. В результаті цієї взаємодії об'єкти з високою схожістю та доступністю можуть бути обрані в якості епіцентрів, представляючи кластери.

Формування кластерів: Після завершення ітераційного процесу об'єкти будуть автоматично призначені до кластерів на основі їхньої взаємодії та подібності до епіцентрів [8].

Принципи “Affinity propagation” відображають абсолютно новий підхід до визначення структури даних через взаємодію об'єктів та вибір об'єктів-епіцентрів для кластерів. Цей унікальний метод кластеризації виявляє потенційні групи об'єктів та може бути особливо корисним у випадках, коли

кількість кластерів не відома наперед чи коли кластери мають складні форми та розміри.

Алгоритм “Affinity propagation” (AP) демонструє інноваційний спосіб кластеризації, який ґрунтується на взаємодії об'єктів та обчисленні міри їхньої подібності. Цей алгоритм може бути описаний наступними етапами:

Обчислення схожості: Перший крок включає обчислення міри подібності (affinity) між кожною парою об'єктів у вихідних даних. Ця міра може бути визначена різними способами, такими як Евклідова відстань, косинусна схожість або інші метрики залежно від характеру даних.

Оновлення схильності та доступності: Об'єкти взаємодіють між собою, обчислюючи дві величини - схильність (responsibility) та доступність (availability). Схильність відображає, наскільки добре об'єкт підходить для статусу епіцентру для іншого об'єкта, враховуючи їхню подібність. Доступність відображає, наскільки доступно є статус епіцентру для об'єкта, засновуючись на внутрішньокластерній схожості.

Формування кластерів: Алгоритм проводить кілька ітерацій оновлення схильності та доступності. Після закінчення ітерацій об'єкти з високою схильністю та доступністю можуть бути обрані як епіцентри кластерів. Кожен об'єкт буде призначений до того кластера, для якого його схильність та доступність є найвищими [22].

Алгоритм “Affinity propagation” продемонстрував свою ефективність у виявленні структури в даних, особливо в складних та невизначених випадках. Він здатний виявити незрозумілі зв'язки між об'єктами та допомагає визначити представників кожного кластера на основі їхньої взаємодії.

Переваги “Affinity propagation”:

1. Автоматичний вибір кількості кластерів: Однією з головних переваг “Affinity propagation” є його здатність автоматично визначати оптимальну кількість кластерів у даних. Це дозволяє уникнути необхідності передбачати або встановлювати кількість кластерів наперед.

2. Адаптація до різних форм та розмірів кластерів: AP проявляє ефективність навіть у випадках, коли кластери мають різні форми та розміри. Він допомагає виявити складні структури даних, що здатні ускладнити роботу інших алгоритмів.
3. Гнучкість у виборі схожості: Ви можете вибрати або налаштувати функцію схожості для об'єктів, що враховує особливості даних. Це дозволяє вам точніше враховувати внутрішні зв'язки та властивості кластерів.

Недоліки Affinity propagation:

1. Час виконання: У великих наборах даних час виконання Affinity propagation може бути значною проблемою через квадратичну складність алгоритму. Він може стати обчасно вимогливим обчислювально.
2. Чутливість до початкових значень: Вибір початкових "епіцентрів" може суттєво вплинути на результат кластеризації. На великих даних важливо правильно вибрати ці початкові точки для досягнення задовільних результатів.
3. Вимоги до пам'яті: Алгоритм вимагає пам'яті для зберігання матриці подібності та інших даних. Зрозуміло, що це може стати обмеженням у випадку дуже великих об'ємів даних.

Інакше кажучи, “Affinity propagation” є потужним і гнучким інструментом для кластеризації даних, але варто враховувати його обмеження щодо часу виконання та обчислювальних ресурсів.

Приклад коду для застосування алгоритму Affinity Propagation з використанням бібліотеки scikit-learn в мові програмування Python:

```
from sklearn.cluster import AffinityPropagation
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Створення випадкових даних для прикладу
X, labels = make_blobs(n_samples=300, centers=4, random_state=0, cluster_std=1.0)

# Ініціалізація та навчання моделі Affinity Propagation
```

```
model = AffinityPropagation(damping=0.9, preference=-200)
model.fit(X)

# Отримання міток кластерів та координат епіцентрів
cluster_centers = model.cluster_centers_
cluster_labels = model.labels_
num_clusters = len(cluster_centers)

# Відображення результатів
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels, cmap='viridis')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], marker='x', s=100,
color='red')
plt.title(f'Affinity Propagation Clustering\nNumber of clusters: {num_clusters}')
plt.show()
```

2.3. Meanshift

MeanShift – це нелінійний алгоритм кластеризації, який використовує поняття зсуву середнього значення для визначення кластерів в даних. Цей алгоритм базується на ідеї знаходження локальних максимумів густини даних та використанні їх як епіцентрів кластерів.

Основні кроки алгоритму такі:

1. Ініціалізація епіцентрів: Алгоритм починається з вибору початкових точок в просторі даних, які слугують початковими епіцентрами для кластерів.

2. Обчислення зсуву середнього значення: Для кожного епіцентру обчислюється зсув середнього значення. Цей зсув визначається як середнє значення векторів даних, які потрапляють в межі певного радіусу навколо епіцентру.

3. Оновлення епіцентрів: Після обчислення зсуву середнього значення епіцентри переміщуються в напрямку зсуву. Цей процес піднімає епіцентри в напрямку, де густина даних є вищою.

4. Призначення до кластерів: Об'єкти даних призначаються до кластерів, вибираючи той епіцентр, до якого вони мають найбільший зсув. Ітерації повторюються, поки епіцентри не стабілізуються.

Mean Shift дозволяє знаходити кластери з різними формами та розмірами та не вимагає заздалегідь встановленої кількості кластерів. Цей алгоритм може бути особливо корисним для виявлення нелінійних структур в даних, але також може бути вимогливим до обчислювальних ресурсів через повторні обчислення та оновлення епіцентрів.

Обчислення густини даних за допомогою ядерної функції: Спочатку, для кожного об'єкта даних обчислюється густина на основі певної ядерної функції. Одним із часто використовуваних варіантів ядерної функції є гаусівське ядро. Це допомагає визначити, наскільки щільно розташовані дані навколо кожного об'єкта [23].

Ініціалізація початкових епіцентрів кластерів: Початкові епіцентри кластерів вибираються випадково або іншим способом, наприклад, за допомогою певної підвибірки даних.

Обчислення зсуву середнього значення: Для кожного епіцентру обчислюється зсув середнього значення, використовуючи ваговану суму векторів даних, де ваги визначаються гаусівською ядерною функцією. Цей зсув вказує напрямок, в якому краще рухатися для знаходження більшої густини даних.

Оновлення епіцентрів: Епіцентри кластерів оновлюються, переміщаючи їх у напрямку зсуву середнього значення. Цей крок вказує алгоритму, як краще згруповувати дані, тягнучи епіцентри до областей більшої густини.

Повторення ітерацій: Кроки 3 і 4 повторюються доки епіцентри не збігнуться, тобто досягнеться збіжність.

Призначення до кластерів: На завершальному етапі, кожний об'єкт призначається до найближчого епіцентру, утворюючи кінцеві кластери.

Алгоритм Mean Shift дозволяє виявляти кластери з різною формою та розмірами, а також автоматично визначає кількість кластерів. Однак він може бути вимогливим до обчислювальних ресурсів, особливо в великих наборах даних, через необхідність повторних обчислень та оновлень епіцентрів.

Переваги Mean Shift:

1. Автоматичний вибір кількості кластерів: Однією з основних переваг Mean Shift є його здатність автоматично визначати кількість кластерів на основі густин даних. Це дозволяє уникнути необхідності вручну задавати цей параметр.
2. Спроможність виявляти нелінійні структури: Алгоритм може успішно виявляти складні нелінійні структури даних, що робить його корисним для обробки різноманітних типів даних.
3. Здатність робити кластеризацію без сферичності: Mean Shift може кластеризувати дані з різними розмірами, густинами та формами, що робить його більш універсальним для різноманітних сценаріїв.

Недоліки Mean Shift:

1. Час виконання: Одним з головних недоліків алгоритму є його обчислювальна складність, особливо на великих наборах даних. Обчислення густини та зсуву для кожної точки може вимагати значних обчислювальних ресурсів.
2. Чутливість до параметрів: Ефективність алгоритму може значно залежати від правильного вибору параметрів ядра та радіусу, що може бути нетривіальним завданням у практиці.
3. Може мати проблеми з шумом: Mean Shift може вказувати на помилкові епіцентри для шумових або малозначущих даних, що може призвести до неточностей у кластеризації.
4. Не завжди збігається до оптимуму: Залежно від початкових точок, алгоритм може зігнути в сторону локального оптимуму, що може призвести до різних результатів кластеризації.

Незважаючи на свої недоліки, Mean Shift є потужним інструментом для кластеризації даних з нелінійними структурами, якщо правильно підібрати параметри та забезпечити достатні обчислювальні ресурси.

Приклад простого коду для реалізації алгоритму Mean Shift за допомогою бібліотеки scikit-learn у Python:

```
import numpy as np
from sklearn.cluster import MeanShift
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Створення випадкових даних для прикладу
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Створення об'єкту MeanShift і навчання на даних
bandwidth = 0.8  # Параметр радіусу для ядерної функції
model = MeanShift(bandwidth=bandwidth)
model.fit(X)

# Отримання епіцентрів та кількості кластерів
centroids = model.cluster_centers_
num_clusters = len(centroids)

# Відображення результатів
plt.scatter(X[:, 0], X[:, 1], c=model.labels_)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', color='red', s=200)
plt.title(f'Mean Shift Clustering\nNumber of clusters: {num_clusters}')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Цей код створює випадкові дані за допомогою make_blobs, використовує бібліотеку scikit-learn для навчання моделі Mean Shift на даних та відображає результати кластеризації та визначені епіцентри.

2.4. Spectral clustering

Spectral clustering є потужним алгоритмом кластеризації, який використовує спектральний аналіз графів для виявлення структури в даних. Основна ідея полягає в тому, що об'єкти даних можна розглядати як вершини графу, а відношення між ними – як ребра. Завдяки цьому, Spectral clustering дозволяє виявляти кластери, які можуть мати складні топологічні властивості та нелінійні зв'язки.

Алгоритм Spectral clustering діє наступним чином:

Побудова графу схожості: Спочатку створюється граф, де вершини представляють об'єкти даних, а ваги ребер відображають ступінь схожості між

цими об'єктами. Схожі об'єкти мають ваги, близькі до нуля, тоді як об'єкти, які менше схожі, мають великі ваги.

Обчислення спектральної матриці: Для графу схожості обчислюється спектральна матриця, яка включає власні вектори та власні значення. Ця матриця допомагає перетворити дані в новий простір, де кластери можуть бути краще виділені [15].

Вибір власних векторів: Вибираються перші k власних векторів спектральної матриці, де k – кількість кластерів, яку ми хочемо знайти.

Кластеризація з використанням K-means: Отримані власні вектори слугують новими вхідними даними для алгоритму K-means. Відбувається кластеризація, де кожен об'єкт призначається до найближчого центроїда, утворюючи кінцеві кластери.

Spectral clustering відмінно підходить для даних зі складною топологією та нелінійними зв'язками, де інші алгоритми можуть не впоратися. Використання спектрального аналізу графів дозволяє виявити приховані залежності між об'єктами даних, забезпечуючи більш точну та глибоку кластеризацію.

Алгоритм Spectral clustering може бути розглянутий у наступних етапах:

1. Побудова графу схожості: Спочатку будується граф схожості, де кожен об'єкт даних представлений як вершина, а ваги ребер відображають ступінь схожості між ними. Це може виконуватися на основі різних метрик схожості, таких як евклідова відстань, кореляція тощо.
2. Обчислення спектральної матриці: Обчислюється спектральна матриця графу, яка включає власні вектори та власні значення. Власні вектори представляють розклад вершин графу у новому просторі, де вони можуть бути більш явно виділені.
3. Вибір власних векторів: Вибираються перші k власних векторів, де k - кількість кластерів, яку ми прагнемо знайти. Ці власні вектори стають новими вхідними даними для кластеризації.

4. Кластеризація з використанням K-means: Отримані власні вектори використовуються для кластеризації за допомогою алгоритму K-means. K-means розділяє дані на кластери, де кожен об'єкт призначається до найближчого центроїда.

Spectral clustering може бути використаний для виявлення структури в даних зі складною топологією та нелінійними зв'язками. Використання власних векторів спектральної матриці дозволяє перетворити дані в новий простір, де кластери можуть бути краще виділені та розділені [4].

Переваги Spectral clustering:

1. Здатність виявляти нелінійні та нероздільні кластери: Однією з основних переваг Spectral clustering є його здатність виявляти складні нелінійні залежності та нероздільні кластери. Це робить його ефективним для виявлення структур у даних зі складною топологією.
2. Відмінна продуктивність на даних зі складними структурами: Spectral clustering показує високу продуктивність на даних зі складними геометричними структурами, де інші методи кластеризації можуть виявитися недоцільними.
3. Не вимагає заздалегідь відомої кількості кластерів: Порівняно з деякими іншими методами, Spectral clustering не потребує попередньої інформації про кількість кластерів, що дозволяє виявляти кластери більш гнучко.

Недоліки Spectral clustering:

1. Обчислювально вимогливий: Однією з головних недоліків Spectral clustering є його висока обчислювальна складність, особливо при роботі з великими об'ємами даних. Обчислення спектральної матриці та власних векторів може бути обчасно витратним.
2. Вибір параметрів: Вибір параметрів, таких як кількість власних векторів або метрика схожості, може суттєво вплинути на результати Spectral clustering. Неправильний вибір параметрів може призвести до неправильної кластеризації.

3. Обмежена працездатність з великими даними: Складність спектрального аналізу обмежує застосовність Spectral clustering для дуже великих наборів даних, особливо коли обчислювальні ресурси обмежені.

Spectral clustering є потужним методом для виявлення структур у даних, особливо наявності нелінійних залежностей, але вимагає обережного підходу до вибору параметрів та оптимізації обчислювальних ресурсів.

Приклад коду для Spectral clustering на мові Python з використанням бібліотеки scikit-learn:

```
import numpy as np
from sklearn.cluster import SpectralClustering
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Створення штучного набору даних з трьома кластерами
n_samples = 300
n_clusters = 3
X, y = make_blobs(n_samples=n_samples, centers=n_clusters, random_state=42)

# Створення об'єкта SpectralClustering та кластеризація даних
spectral_clustering = SpectralClustering(n_clusters=n_clusters,
affinity='nearest_neighbors', random_state=42)
predicted_labels = spectral_clustering.fit_predict(X)

# Візуалізація результатів кластеризації
plt.scatter(X[:, 0], X[:, 1], c=predicted_labels, cmap='rainbow')
plt.title("Spectral Clustering")
plt.show()
```

Цей код створює штучний набір даних з трьома кластерами, застосовує Spectral clustering та візуалізує результати. Будь ласка, переконайтеся, що ви маєте встановлені бібліотеки scikit-learn та matplotlib для виконання цього прикладу.

2.5. Hierarchical clustering

Алгоритм ієрархічної кластеризації (Hierarchical clustering) є одним з популярних підходів до групування даних. Його основна ідея полягає в побудові деревоподібної структури кластерів, де кожен об'єкт спочатку розглядається як окремий кластер, а потім послідовно об'єднується з іншими

кластерами до тих пір, поки всі об'єкти не об'єднуються в один великий кластер або досягається задана кількість кластерів.

Принципи ієрархічної кластеризації можна розділити на два основних підходи: агломеративний та дивізійний.

1. Агломеративний підхід (Bottom-Up):

У даному підході кожен об'єкт спочатку вважається окремим кластером, а потім послідовно об'єднується з найближчими кластерами, поки не залишиться один загальний кластер. Процес об'єднання відбувається на основі відстаней або схожості між кластерами, іноді використовуються спеціальні метри агломерації.

2. Дивізійний підхід (Top-Down):

У цьому підході спочатку всі об'єкти знаходяться в одному великому кластері, який поділяється на менші кластери шляхом розбиття на підгрупи. Процес поділу також базується на відстанях або схожості, і він продовжується досягнення певного рівня глибини дерева або досягнення бажаної кількості кластерів.

Алгоритм ієрархічної кластеризації може бути використаний для візуалізації структури даних у вигляді дерева, а також допомагає знаходити кластери на різних рівнях дерева, що дозволяє аналізувати дані з різних поглядів [20].

Алгоритм ієрархічної кластеризації можна поділити на наступні кроки:

1. Ініціалізація: Кожен об'єкт даних спочатку вважається окремим кластером.
2. Обчислення матриці відстаней або схожості: Для кожної пари об'єктів обчислюється відстань або схожість. Це може бути, наприклад, Евклідова відстань, косинусна схожість або інша метрика.
3. Об'єднання найближчих кластерів: Знаходяться два найближчі кластери на основі обчисленої матриці відстаней або схожості, і їх об'єднують у новий кластер.

4. Оновлення матриці відстаней або схожості: Матриця відстаней або схожості оновлюється, враховуючи новий об'єднаний кластер.
5. Повторення кроків 3-4: Кроки 3 та 4 повторюються доти, доки всі об'єкти не об'єднаються у один загальний кластер або досягнеться бажана кількість кластерів.
6. Побудова дерева кластерів (дендрограми): Процес об'єднання кластерів може бути відображений у вигляді дерева, де висота кожної вершини представляє відстань або схожість між об'єктами або кластерами.

Алгоритм ієрархічної кластеризації може бути реалізований за допомогою різних методів, таких як "одиночний зв'язок" (single linkage), "повний зв'язок" (complete linkage), "середній зв'язок" (average linkage) тощо. Кожен з цих методів визначає спосіб обчислення відстані або схожості між кластерами.

Переваги Hierarchical clustering:

1. Інтерпретованість: Дерево кластерів (дендрограма) дозволяє візуалізувати інформацію про кластери та їхні взаємозв'язки, що сприяє легшому розумінню результатів.
2. Немає потреби у визначенні кількості кластерів: Ієрархічна кластеризація дозволяє природно поділити дані на кластери будь-якої кількості, оскільки вона створює дерево кластерів, з якого можна обрати потрібну кількість на будь-якому рівні.
3. Збереження ієрархічної структури: Дендрограма відображає структуру даних у вигляді дерева, що може бути корисним при аналізі.
4. Відображення внутрішнього подібності: Ієрархічна кластеризація здатна виявляти внутрішні структури та подібності в межах кластерів.

Недоліки Hierarchical clustering:

1. Обчислювальна складність: Алгоритм може бути вимогливим до обчислювальних ресурсів, особливо при обробці великих наборів даних, оскільки кількість обчислень залежить від кількості об'єктів.

2. Чутливість до шуму: Велика кількість шумових даних може призвести до помилкових з'єднань між кластерами.
3. Вибір метрики та методу з'єднання: Вибір правильної метрики схожості та методу з'єднання може суттєво вплинути на результати кластеризації.
4. Неефективність для великих наборів даних: При роботі з великими наборами даних ієрархічна кластеризація може бути громіздкою та важкою для обробки.
5. Неможливість змінювати розташування об'єктів: Після об'єднання об'єкти не можуть бути перенесені з одного кластера до іншого без перерахунку всієї структури.

Не дивлячись на свої недоліки, ієрархічна кластеризація залишається корисним інструментом для аналізу та візуалізації даних у вигляді дерева кластерів.

Приклад реалізації ієрархічної кластеризації на мові Python з використанням бібліотеки scikit-learn:

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt

# Генеруємо випадкові дані для прикладу
data, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Використовуємо ієрархічну кластеризацію
n_clusters = 4
agg_clustering = AgglomerativeClustering(n_clusters=n_clusters)
agg_labels = agg_clustering.fit_predict(data)

# Візуалізація результатів
plt.scatter(data[:, 0], data[:, 1], c=agg_labels, s=50, cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Hierarchical Clustering')
plt.show()
```

У цьому прикладі ми генеруємо випадкові дані з чотирьма центрами за допомогою make_blobs, потім використовуємо ієрархічну кластеризацію з

AgglomerativeClustering та візуалізуємо результати. Зверніть увагу, що n_clusters вказує кількість кластерів, яку ми хочемо знайти.

2.6. DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) – це алгоритм кластеризації, який використовує поняття щільності даних для виявлення кластерів. Основна ідея полягає в тому, щоб виділити області в просторі, де точки розташовані щільно, та розглядати їх як кластери. DBSCAN дозволяє виявляти кластери різних форм та розмірів, а також ідентифікувати окремі шумові точки, які не входять до жодного кластера.

Основні принципи DBSCAN:

Поняття щільності: DBSCAN базується на ідеї щільності точок. Точки, які розташовані близько одна до одної, утворюють області з високою щільністю, які можна вважати кластерами.

Точка ядра: Точка вважається "точкою ядра" (core point), якщо в радіусі eps від неї міститься щонайменше min_samples точок (включаючи саму точку). Тобто ця точка лежить в області високої щільності [17].

Сусіди: Точка також може бути "сусідом" (neighbor) іншої точки, якщо вона знаходиться в радіусі eps від неї, навіть якщо вона сама не є точкою ядра.

Перехід по сусідах: Два ядра (точки ядра) вважаються зв'язаними, якщо їх можна з'єднати через послідовність сусідів.

Формування кластерів: Кластер формується шляхом з'єднання точок ядер та їх сусідів у зв'язані компоненти.

Шум: Точки, які не можуть бути об'єднані з жодним іншим кластером, вважаються шумовими точками.

Алгоритм DBSCAN дозволяє виявляти кластери без заздалегідь відомої кількості та форми, а також виділяти шумові точки, що робить його потужним інструментом для кластеризації даних.

Алгоритм складається з декількох кроків, щоб визначити кластери та шумові точки в наборі даних.

Кроки алгоритму DBSCAN:

1. Вибір початкової точки: Вибираємо довільну невідвідану точку з набору даних.
2. Визначення сусідів: Знаходимо всі точки, які знаходяться в максимальній відстані eps від обраної точки. Ці точки стають її сусідами.
3. Перевірка на точку ядра: Якщо кількість сусідів обраної точки більше або дорівнює min_samples, то ця точка вважається точкою ядра.
4. Розповсюдження кластера: Розпочинаючи з точки ядра, розглядаємо її сусідів які також є точками ядрами та знаходимо всіх їхніх сусідів. Продовжуємо цей процес до тих пір, поки не будуть вичерпані всі точки ядра та їхні сусіди.
5. Формування кластерів: Якщо точка ядра має не менше min_samples сусідів, то всі точки, які були розглянуті під час розповсюдження кластера, входять до одного кластера. Якщо точка не має досить сусідів, вона вважається шумовою.
6. Повторення: Повторюємо кроки 1-5 для кожної невідвіданої точки, поки всі точки не будуть призначені до кластерів або визначені як шумові.

Алгоритм DBSCAN визначає кластери на основі щільності точок, дозволяючи виділити кластери різних форм та виявити шумові точки. Його ефективність залежить від вибору параметрів eps та min_samples, які визначають мінімальний радіус для визначення сусідів та мінімальну кількість сусідів для точки ядра.

Приклад реалізації алгоритму DBSCAN на мові програмування Python:

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt

# Створення вхідних даних (у цьому випадку - штучних даних про місяць)
X, _ = make_moons(n_samples=200, noise=0.05, random_state=0)
```

```
# Ініціалізація алгоритму DBSCAN
dbscan = DBSCAN(eps=0.3, min_samples=5)

# Передача даних для кластеризації
labels = dbscan.fit_predict(X)

# Візуалізація результатів
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('DBSCAN Clustering')
plt.show()
```

У цьому прикладі ми використали штучно створені дані про місяць та використали бібліотеку Scikit-learn для реалізації алгоритму DBSCAN. Результати кластеризації візуалізовані за допомогою діаграми розсіювання, де кожен кластер має свій власний колір.

2.7. OPTICS

Алгоритм OPTICS (Ordering Points To Identify the Clustering Structure) є методом кластеризації, який розширює можливості алгоритму DBSCAN, спрямований на виявлення структури кластерів у даних з різною густинею та формами. Основна ідея OPTICS полягає в аналізі взаємодії об'єктів у просторі даних та визначенні їх порядку досяжності.

Порядок досяжності (reachability distance) між двома об'єктами вказує на те, наскільки легко можна дістатися від одного об'єкта до іншого, пройшовши через інші точки даних. Це дає можливість виявити локальні групи об'єктів, які можуть бути кластерами.

Принцип роботи OPTICS:

Вибір початкової точки: Обирається початковий об'єкт даних, з якого розпочинається процес аналізу досяжності.

Обчислення досяжності: Для кожного об'єкта обчислюється його досяжність до інших об'єктів, враховуючи параметр "eps" (радіус досяжності). Це дозволяє визначити, наскільки легко можна дістатися від одного об'єкта до іншого [19].

Формування графу відносних відстаней: На основі досяжності будується граф відносних відстаней, де кожен об'єкт представляється вузлом, а ребра вказують на досяжність між об'єктами.

Сортування за досяжності: Об'єкти сортуються за зростанням досяжності. Це допомагає визначити локальну структуру даних та її порядок.

Виявлення кластерів: Аналізуючи граф відносних відстаней та порядок об'єктів, можна визначити кластери та їхню ієрархію.

Алгоритм OPTICS дозволяє виявити кластери різної форми та густини, а також виявити шумові точки. Він є потужним інструментом для аналізу структури даних та виявлення кластерів без попередньо заданої кількості чи форми.

Приклад реалізації алгоритму OPTICS на мові програмування Python з використанням бібліотеки scikit-learn:

```
import numpy as np
from sklearn.cluster import OPTICS
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Генерація синтетичних даних для прикладу
X, y = make_blobs(n_samples=300, centers=3, random_state=0, cluster_std=0.5)

# Ініціалізація та навчання алгоритму OPTICS
optics = OPTICS(min_samples=5, xi=0.05)
optics.fit(X)

# Визначення кластерів та досяжності для кожного об'єкта
labels = optics.labels_
reachability = optics.reachability_
ordering = optics.ordering_

# Вивід результатів
plt.figure(figsize=(10, 7))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)
plt.xlabel('Признак 1')
plt.ylabel('Признак 2')
plt.title('Кластеризація методом OPTICS')
plt.show()
```

Цей код генерує синтетичні дані, навчає алгоритм OPTICS і виводить графічне представлення результатів кластеризації.

2.8. BIRCH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) – це алгоритм кластеризації, який спеціально розроблений для ефективної обробки великих об'ємів даних. Основна ідея BIRCH полягає в створенні ієрархічної структури для представлення та кластеризації даних. Цей алгоритм дозволяє швидко виявляти кластери на різних рівнях деталізації та забезпечує баланс між часом виконання та точністю результатів.

Основні принципи та ідеї BIRCH:

Згортка даних: Алгоритм використовує поняття "згортки" для представлення даних на різних рівнях деталізації. Замість аналізу всіх об'єктів даних наразі, BIRCH поступово згортає їх у вузли дерева, які представляють кластери різних рівнів.

Об'єднання вузлів: Якщо вузол дерева стає переповненим, BIRCH може об'єднати його з іншими вузлами, утворюючи більші кластери. Це допомагає забезпечити баланс між розміром кластерів та швидкістю обчислень.

Ієрархічна структура: BIRCH побудовує ієрархічну структуру дерева, де кожен рівень представляє кластери різного рівня деталізації. Це дозволяє виявляти кластери на різних масштабах та враховувати різні аспекти даних.

Використання центроїдів: Вузли дерева містять центроїди, які представляють кластери. Це допомагає зменшити обчислювальну складність та сприяє ефективності алгоритму.

Ефективність обробки великих даних: BIRCH створює компактне представлення даних та забезпечує швидке виявлення кластерів. Це робить його особливо корисним для задач, пов'язаних з великими обсягами даних.

Алгоритм BIRCH може бути особливо ефективним для великих наборів даних, де важливо забезпечити баланс між точністю результатів та часом виконання.

Алгоритм складається з декількох кроків, які допомагають побудувати ієрархічну структуру та здійснити кластеризацію даних:

1. Побудова C-дерева (Clustering Feature Tree):
2. Дані розділяються на фіксовану кількість груп, називаних "лійками" (листками).
3. Для кожного лійки обчислюється центроїд (середнє значення) та дисперсія.
4. Лійки об'єднуються вищими рівнями C-дерева, де центроїди та дисперсії обчислюються знову.
5. Об'єднання лійок: Лійки, які мають близькі центроїди, можуть бути об'єднані в одну лійку на більш високому рівні C-дерева. Це дозволяє скоротити кількість лійок та спростити ієрархічну структуру.
6. Виявлення кластерів: Здійснюється пошук кластерів на різних рівнях C-дерева. Лійки, які задовольняють певним критеріям (наприклад, кількість об'єктів або дисперсія), вважаються кластерами.
7. Призначення до кластерів: Об'єкти даних призначаються до відповідних кластерів, заснованих на структурі C-дерева.

Алгоритм BIRCH може бути ефективним при роботі з великими об'ємами даних, оскільки він дозволяє зменшити обчислювальну складність за рахунок побудови ієрархічної структури. Він також дозволяє зберігати зведені дані у вузлах дерева, що сприяє швидкості інформаційної обробки та кластеризації.

Приклад реалізації алгоритму BIRCH на мові Python:

```
import numpy as np
from sklearn.cluster import Birch

# Створення випадкових даних для прикладу
data = np.random.rand(100, 2)

# Ініціалізація алгоритму BIRCH
birch = Birch(n_clusters=3, threshold=0.1)

# Кластеризація даних
birch.fit(data)

# Перелік приналежності кожного об'єкта до кластера
labels = birch.labels_
```

```
# Центроїди кластерів
centroids = birch.subcluster_centers_

# Вивід результатів
print("Приналежність до кластера:", labels)
print("Центроїди кластерів:", centroids)
```

У цьому прикладі ми спочатку імпортуємо необхідні бібліотеки, створюємо випадкові дані, ініціалізуємо алгоритм BIRCH з параметрами кількості кластерів і порогового значення, кластеризуємо дані та виводимо результати.

2.9. Порівняння методів бібліотеки scikit-learn

Бібліотека scikit-learn надає зручні інструменти для порівняння різних методів кластеризації та оцінки якості їх результатів. Для здійснення порівняння методів кластеризації за допомогою scikit-learn, можна використовувати наступні кроки:

Цей приклад демонструє, як використовувати бібліотеку scikit-learn для порівняння методів кластеризації на прикладі згенерованих даних.

Порівняння алгоритмів ілюстровано на рис.2.1:

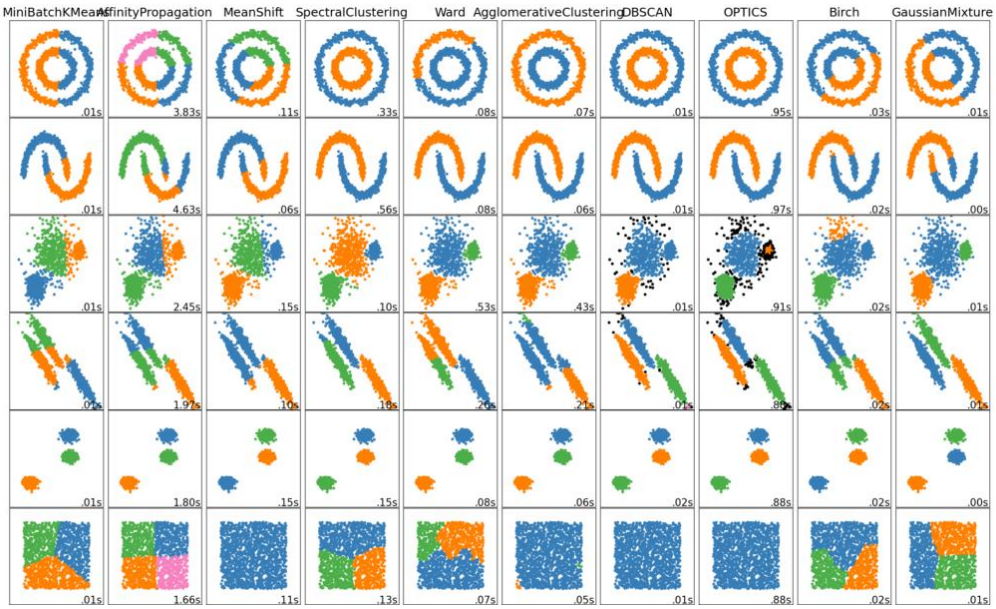

Рисунок 2.1 – Методи кластеризації бібліотеки “scikit-learn”

Порівняння результатів різних методів кластеризації є важливою частиною дослідження, оскільки воно допомагає визначити найефективніший

алгоритм для конкретної задачі. Після виконання кластеризації за допомогою різних методів, можна провести порівняльний аналіз результатів на основі різних метрик якості кластеризації. Деякі з популярних метрик включають:

Adjusted Rand Index (ARI): Вимірює подібність між справжніми мітками кластерів та мітками, присвоєними алгоритмом. Значення ARI лежать між -1 і 1, де 1 – вказує на ідеальне співпадіння, а 0 – на випадковий вибір.

Silhouette Score: Вимірює, наскільки добре об'єкти внутрішнього кластера подібні одне до одного, порівняно з іншими кластерами. Значення силуету лежать між -1 і 1, де високі значення вказують на добру якість кластеризації [24].

Homogeneity, Completeness, V-measure: Ці метрики вимірюють однорідність та повноту кластерів. Вони показують, наскільки об'єкти в одному кластері подібні між собою (однорідність) та наскільки всі об'єкти одного справжнього кластера належать одному прогнозованому кластеру (повнота).

Calinski-Harabasz Index (Variance Ratio Criterion): Вимірює відношення дисперсій між кластерами та дисперсій всередині кластерів. Високі значення вказують на добру якість кластеризації.

Для порівняння результатів методів кластеризації за допомогою бібліотеки scikit-learn, можна використовувати наступний підхід:

- Виконати кластеризацію за допомогою різних методів.
- Виміряти значення різних метрик якості кластеризації для кожного методу.
- Порівняти значення метрик та вибрати метод з найкращими показниками.
- Звернути увагу на особливості кожного методу, його переваги та недоліки, а також на придатність для конкретної задачі.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОГО ЕКСПЕРИМЕНТУ: ВИЯВЛЕННЯ АНОМАЛЬНОЇ АКТИВНОСТІ ЗА ДОПОМОГОЮ КЛАСТЕРИЗАЦІЇ

3.1. Опис набору даних

Для проведення дослідження був використаний датасет – NSL-KDD. Даний датасет має набір даних , який був отриманий з різних джерел у галузі кібербезпеки. Джерелами даних могли бути лог-файли мережевої активності, дані про вразливості систем, а також інші інформаційні джерела, які містять інформацію про мережеву активність та пов'язані з нею параметри. Для забезпечення якості даних були використані відомі та достовірні джерела, а також був застосований процес перевірки та обробки даних перед подальшим аналізом.

Набір даних має структуровану форму представлення, що дозволяє зберігати та організовувати інформацію про мережеву активність. Кожен запис у наборі даних має ряд атрибутів, які описують різні характеристики мережевої активності. Атрибути включають інформацію про продовження тривалості підключення, тип протоколу, використану послугу, стан прапорців, кількість байтів, спожитих на передачу, та багато інших параметрів, що характеризують мережеву активність. Детальніше опис датасету можна переглянути в таблиці 1.1.

Таблиця 1.1 Опис даних датасету NSL-KDD

№	Атрибут	Опис	Тип даних	Обмеження
1	duration	Тривалість з'єднання (секунди)	Ціле число	>=0
2	protocol_type	Тип протоколу на транспортному рівні (tcp, udp, icmp)	Категорійний	-
3	service	Мережева служба (http, ftp, telnet)	Категорійний	-
4	flag	Стандартний статус підключення (SF, S0)	Категорійний	-
5	src_bytes	Кількість байтів, переданих від джерела до призначення	Ціле число	>=0
6	dst_bytes	Кількість байтів, переданих від призначення до джерела	Ціле число	>=0

7	land	З'єднання «землі» (1, якщо з'єднання відправлене/отримане від того ж хосту/порту)	Бінарний	0 або 1
8	wrong_fragment	Кількість неправильних фрагментів	Ціле число	>=0
9	urgent	Кількість термінових пакетів	Ціле число	>=0
10	hot	Кількість «гарячих» індикаторів (введення пароля в буфер)	Ціле число	>=0
11	num_failed_logins	Кількість невдалих спроб логіну	Ціле число	>=0
12	logged_in	Успішна авторизація (1 - так)	Бінарний	0 або 1
13	num_compromised	Кількість «компрометованих умов»	Ціле число	>=0
14	root_shell	Чи має користувач root оболонку (1 - так)	Бінарний	0 або 1
15	su_attempted	Чи була спроба використання su команди (1, якщо так)	Бінарний	0 або 1, 2
16	num_root	Кількість «кореневих доступів»	Ціле число	>=0
17	num_file_creations	Кількість створених файлів	Ціле число	>=0
18	num_shells	Кількість оболонок	Ціле число	>=0
19	num_access_files	Кількість доступів до файлів	Ціле число	>=0
20	num_outbound_cmds	Кількість вихідних команд (в цьому датасеті ця властивість = 0)	Ціле число	= 0
21	is_host_login	Чи був хост логін (в цьому датасеті ця властивість = 0)	Бінарний	= 0
22	is_guest_login	Чи був гостьовий логін (1 - так)	Бінарний	0 або 1
23	count	Кількість підключень до одного і того ж хоста в межах 2 секунд	Ціле число	>=0
24	srv_count	Кількість підключень до одного і того ж сервісу в межах 2 секунд	Ціле число	>=0
25	serror_rate	Відсоток підключень, що мають «SYN» помилки	Дійсне число	[0.0, 1.0]
26	srv_serror_rate	Відсоток підключень, що мають «SYN» помилки (за сервіс)	Дійсне число	[0.0, 1.0]
27	rerror_rate	Відсоток підключень, що мають «REJ» помилки	Дійсне число	[0.0, 1.0]
28	srv_rerror_rate	Відсоток підключень, що мають «REJ» помилки (за сервіс)	Дійсне число	[0.0, 1.0]
29	same_srv_rate	Відсоток підключень до того ж сервісу	Дійсне число	[0.0, 1.0]
30	diff_srv_rate	Відсоток підключень до різних сервісів	Дійсне число	[0.0, 1.0]
31	srv_diff_host_rate	Відсоток підключень до різних хостів (за сервіс)	Дійсне число	[0.0, 1.0]
32	dst_host_count	Кількість підключень до одного і того ж хоста-призначення	Ціле число	>=0
33	dst_host_srv_count	Кількість підключень до одного і того ж сервісу на хості-призначенні	Ціле число	>=0
34	dst_host_same_srv_rate	Відсоток підключень до того ж сервісу на хості-призначенні	Дійсне число	[0.0, 1.0]

35	dst_host_diff_srv_rate	Відсоток підключень до різних сервісів на хості-призначенні	Дійсне число	[0.0, 1.0]
36	dst_host_same_src_port_rate	Відсоток підключень до хоста-призначення з того ж порта	Дійсне число	[0.0, 1.0]
37	dst_host_srv_diff_host_rate	Відсоток підключень до сервісу на хості-призначенні з різних хостів-джерел	Дійсне число	[0.0, 1.0]
38	dst_host_serror_rate	Відсоток підключень до хоста-призначення з «SYN» помилками	Дійсне число	[0.0, 1.0]
39	dst_host_srv_serror_rate	Відсоток підключень до сервісу на хості-призначенні з «SYN» помилками	Дійсне число	[0.0, 1.0]
40	dst_host_rerror_rate	Відсоток підключень до хоста-призначення з «REJ» помилками	Дійсне число	[0.0, 1.0]
41	dst_host_srv_rerror_rate	Відсоток підключень до сервісу на хості-призначенні з «REJ» помилками	Дійсне число	[0.0, 1.0]

Структура даних дозволяє систематизувати інформацію про різні аспекти мережевої активності, забезпечуючи зручну та ефективну аналітику та кластеризацію даних. Дані в наборі представлені у вигляді таблиці, де кожен рядок відповідає окремому запису, а стовпці – атрибутам даних.

Набір даних містить в собі:

- KDDTrain – Повний набір для тренування із 125973 об’єктами;

Файл відображає різні мережеві активності. У цьому наборі даних наявні дані з різних активностей, включаючи нормальну та аномальну мережеву активність.

Кожен запис у цьому наборі даних представляє одне з'єднання, яке є нормальним або аномальним (атака). Аномальні з'єднання поділяються на чотири основні категорії атак: DOS (Denial of Service), Probe, U2R (User to Root), і R2L (Remote to Local). Ця інформація важлива для визначення обсягу та складності обробки даних під час подальшого аналізу та використання методів кластеризації.

3.2 Попередня підготовка даних

Передпроцесінг даних є важливим етапом підготовки даних для подальшого аналізу та застосування методів кластеризації. Набір даних може містити різноманітні аномалії, пропуски, шум та інші некоректності, які можуть впливати на точність результатів.

Під час передпроцесінгу даних виконуються такі операції:

1. Видалення дублікатів: якщо в наборі даних є однакові записи, вони можуть бути видалені, щоб уникнути спотворення результатів.
2. Видалення пропусків: якщо деякі об'єкти мають відсутні значення атрибутів, такі записи можуть бути видалені або пропуски можуть бути заповнені певними значеннями (наприклад, середнім значенням атрибуту).
3. Кодування категоріальних ознак: якщо в наборі даних присутні категоріальні ознаки (наприклад, тип протоколу), вони можуть бути закодовані числовими значеннями, щоб можна було застосовувати алгоритми кластеризації.
4. Нормалізація/стандартизація: деякі алгоритми кластеризації чутливі до масштабу атрибутів. Тому атрибути можуть бути нормалізовані (перетворені так, щоб їх значення були у певному діапазоні) або стандартизовані (перетворені так, щоб середнє значення було 0, а стандартне відхилення – 1).
5. Видалення непотрібних ознак: якщо деякі ознаки не мають суттєвого впливу на кластеризацію, їх можна видалити для спрощення аналізу та покращення продуктивності.

Правильний передпроцесінг даних може суттєво вплинути на якість та точність результатів кластеризації, допомагаючи уникнути спотворень і невірних висновків під час аналізу мережевої активності.

Код для передпроцесінгу є загальним для всіх наступних програм:

```
# Завантаження бібліотек
```

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import warnings
# Завантаження даних
data = pd.read_csv('KDDTrain.csv', header=None)
# Визначення унікальних класів в об'єктах
column_index = 41
unique_classes = data[column_index].unique()
class_counts = data[column_index].value_counts()

# Виведення кількості об'єктів в кожному класі
print(f"Кількість об'єктів в кожному класі '{column_index}':\n", class_counts)
# Список відповідностей класу до норми/типу атаки
grouping_dict = {
    'normal': 'normal',
    'neptune': 'DoS',
    'satan': 'Probe',
    'ipsweep': 'Probe',
    'portsweep': 'Probe',
    'smurf': 'DoS',
    'nmap': 'Probe',
    'back': 'DoS',
    'teardrop': 'DoS',
    'warezclient': 'R2L',
    'pod': 'DoS',
    'guess_passwd': 'R2L',
    'buffer_overflow': 'U2R',
    'warezmaster': 'R2L',
    'land': 'DoS',
    'imap': 'R2L',
    'rootkit': 'U2R',
    'loadmodule': 'U2R',
    'ftp_write': 'R2L',
    'multihop': 'R2L',
    'phf': 'R2L',
    'perl': 'U2R'
}
# Перейменування міток класів
data[column_index] = data[column_index].map(grouping_dict)
# Визначення кількості записів класу "normal"
column_name = data.columns[column_index]
# Визначення кількості записів для кожного класу
```

```
class_counts = data[column_name].value_counts()
# Визначення загальної кількості записів
total_count = len(data[column_name])
# Прохід по кожному класу та виведення інформації
for class_name, count in class_counts.items():
    percentage = (count / total_count) * 100
    print(f"Клас: {class_name}, Кількість об'єктів: {count}, Відсоткове відношення:
{percentage:.2f}%")

# Видалення рядків, де у колонці з індексом 41 є NaN значення
data_cleaned = data.dropna(subset=[data.columns[41]])
# Перевірка, що рядки були видалені
print("Кількість рядків до видалення:", len(data))
print("Кількість рядків після видалення:", len(data_cleaned))
# Визначення числових, категоріальних ознак
numerical_features = data_cleaned.select_dtypes(include=['int64', 'float64']).columns
categorical_features = data_cleaned.select_dtypes(include=['object']).columns
# Для подальшого використання ЛИШЕ числових ознак
data_numerical = data_cleaned[numerical_features]
# Визначення бінарних числових ознак
binary_numerical_features = [col for col in numerical_features if
data_numerical[col].nunique() == 2]
# Виведення списку бінарних числових ознак та їх індексів
print("Бінарні числові ознаки та їх індекси:")
for feature in binary_numerical_features:
    print(f"{feature}: Індекс {data_numerical.columns.get_loc(feature)}")
# Індекси бінарних ознак
binary_feature_indices = [3, 8, 10, 17, 18]
# Отримання назв колонок з вказаних індексів
binary_feature_names = data_numerical.columns[binary_feature_indices].tolist()
# Видалення бінарних ознак з датафрейму за назвами колонок
data_no_binary = data_numerical.drop(columns=binary_feature_names)
# Припустимо, що 'data_numerical_no_binary' - це ваш датафрейм без бінарних ознак
selected_features = data_no_binary.columns[:34].tolist()
# Виведення вибраних ознак
print(selected_features)
# Заповнення пропущених значень середніми
imputer = SimpleImputer(strategy='mean')
data_numerical_imputed = imputer.fit_transform(data_no_binary)
# Нормалізація даних
# Створення об'єкту MinMaxScaler
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data_numerical_imputed)
```

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

Вибір оптимальної кількості кластерів є важливою складовою процесу кластеризації методом k-means. Неправильний вибір кількості кластерів може призвести до невірних або неінтерпретованих результатів. Для визначення оптимальної кількості кластерів можна використовувати різні підходи.

Один з популярних методів – метод "ліктя" (elbow method). Цей метод передбачає використання графіка, де по горизонтальній вісі відображена кількість кластерів, а по вертикальній - сума квадратів відстаней між об'єктами і центроїдами в кожному кластері. Графік нагадує форму ліктя, і оптимальна кількість кластерів вважається тією, де зменшення суми квадратів відстаней стає менш помітним – тобто "лікоть" графіка.

Цей алгоритм вважається ітераційним, який мінімузує суму квадратів відстаней між кожною точкою і центром кластера, до якого вона належить, для всіх кластерів.

$$J = \sum_{i=1}^{n} \sum_{j=1}^{k} z_{ij} \left\| x_i - \mu_j \right\|^2$$

Де: k – кількість кластерів; z_{ij} – бінарний індикатор; $\left\| x_i - \mu_j \right\|$ – євклідова відстань між точкою і центром кластера.

Після підготовки даних ми можемо перейти безпосередньо до реалізації алгоритму k-means. Основний принцип алгоритму полягає в тому, щоб розділити дані на k кластерів, де k – попередньо задане число кластерів.

Основні кроки алгоритму включають:

1. Ініціалізація центроїдів: Вибираємо випадковим чином k початкових центроїдів - один для кожного кластеру.
2. Призначення кластерів: Для кожного об'єкта з набору даних обчислюємо відстань до всіх центроїдів і призначаємо об'єкт до кластеру з найближчим центроїдом.

3. Оновлення центроїдів: Для кожного кластеру обчислюємо новий центроїд, який є середнім значенням всіх об'єктів в цьому кластері.
4. Повторення кроків 2-3: Повторюємо кроки призначення кластерів і оновлення центроїдів до тих пір, поки центроїди більше не змінюються або досягнуто максимальної кількості ітерацій.
5. Завершення алгоритму: Після закінчення ітерацій алгоритм завершує роботу, і ми отримуємо розділені на кластери об'єкти.

Після застосування алгоритму k-means і розділення даних на кластери важливо візуалізувати отримані результати для кращого розуміння та аналізу. Для цього можна використовувати різноманітні графічні методи та діаграми.

Графічна візуалізація результатів допомагає легше розуміти, як алгоритм розділив дані на кластери, і може надати важливі вказівки для подальшого аналізу та виявлення аномалій.

Аналіз отриманих кластерів є важливим етапом для розуміння структури даних та виявлення особливостей, що можуть впливати на подальші дії, такі як виявлення аномалій чи покращення моделей класифікації.

Розв’язок задачі на основі методу k-means

Оскільки реалізація алгоритму буде на базі мови програмування Python для реалізації алгоритмів потребується бібліотеки numpy, pandas а також matplotlib.

Код що використовувався:

```
# Ініціалізація списків
inertia = []
silhouette_scores = []
# Виконання k-means для різної кількості кластерів
clusters_range = range(1, 9)
for k in clusters_range:
    kmeans = KMeans(n_clusters=k, random_state=1)
    kmeans.fit(data_scaled)
    inertia.append(kmeans.inertia_)

    # Розрахунок силуетного коефіцієнта
    if k > 1:
        silhouette = silhouette_score(data_scaled, kmeans.labels_)
        silhouette_scores.append(silhouette)
```

```
# Виведення результатів
print("Метод Ліктя (Сума квадратів відстаней):", inertia)
print("Метод Силуетів (Силуетний коефіцієнт):", silhouette_scores)
# Графічне відображення результату методу "Ліктя"
plt.figure(figsize=(6, 6))
plt.plot(clusters_range, inertia, marker='o')
plt.title('Метод "Ліктя"')
plt.xlabel('Кількість кластерів')
plt.ylabel('Сума квадратів відстаней')
plt.show()
# Графічне відображення результату методу "Силуетів"
plt.figure(figsize=(6, 6))
plt.plot(range(2, 9), silhouette_scores, marker='o')
plt.title('Метод "Силуетів"')
plt.xlabel('Кількість кластерів')
plt.ylabel('Силуетний коефіцієнт')
plt.show()
#Виконання PCA
pca = PCA()
pca.fit(data_scaled)
# Визначення оптимальної кількості кластерів
optimal_clusters = 3
# Виконання K-Means кластеризації
kmeans = KMeans(n_clusters=optimal_clusters, random_state=1)
clusters = kmeans.fit_predict(data_scaled)
# Додавання міток кластерів до датафрейму
data_cleaned['cluster'] = clusters
# Виконання PCA для зменшення
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
# Створення DataFrame для PCA даних
pca_df = pd.DataFrame(data=data_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = clusters  # Додаємо мітки кластерів
cluster_labels = {
    0: 'DoS',
    1: 'normal',
    2: 'Probe',
    3: 'R2L',
    4: 'U2R'
}plt.figure(figsize=(8, 6))
for cluster_number, cluster_name in cluster_labels.items():
    # Відбір даних для кожного кластеру
    cluster_subset = pca_df[pca_df['cluster'] == cluster_number]

    # Будування графіка для кластеру
    sns.scatterplot(x='PC1', y='PC2', data=cluster_subset, label=cluster_name)

plt.title('Результати кластеризації')
plt.legend()
plt.show()
```

Результат роботи програмного коду відображається на рис.3.1, 3.2 і 3.3:

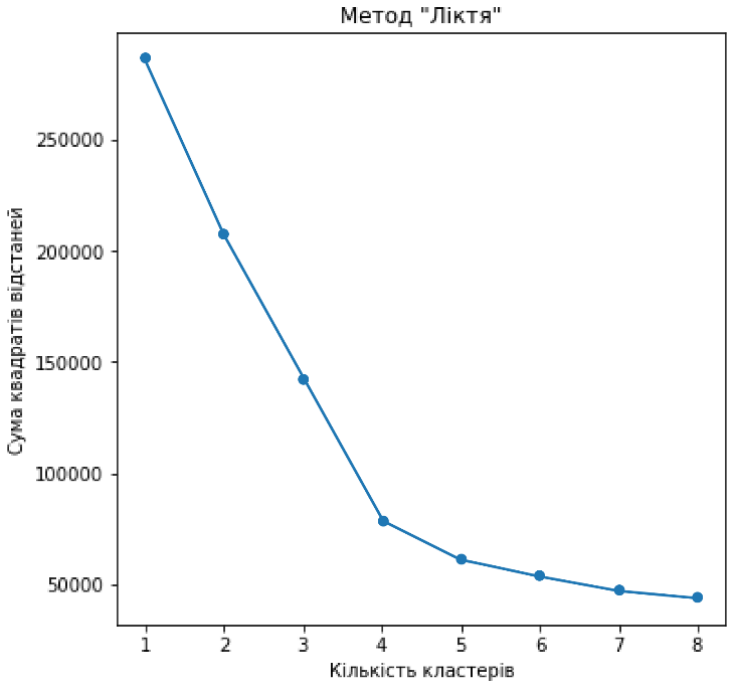

Рисунок 3.1 – Використання методу «Ліктя»

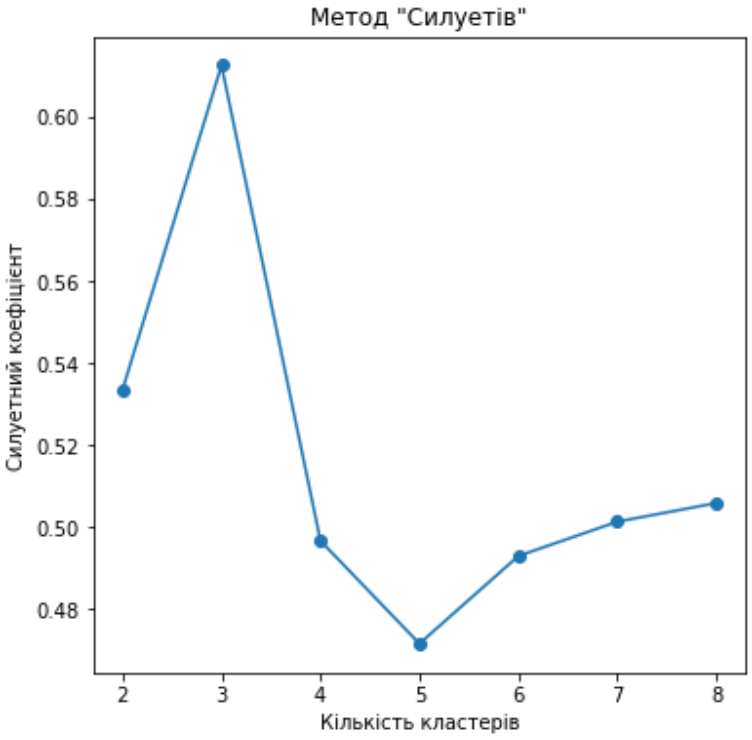

Рисунок 3.2 – Використання методу «Силуета»

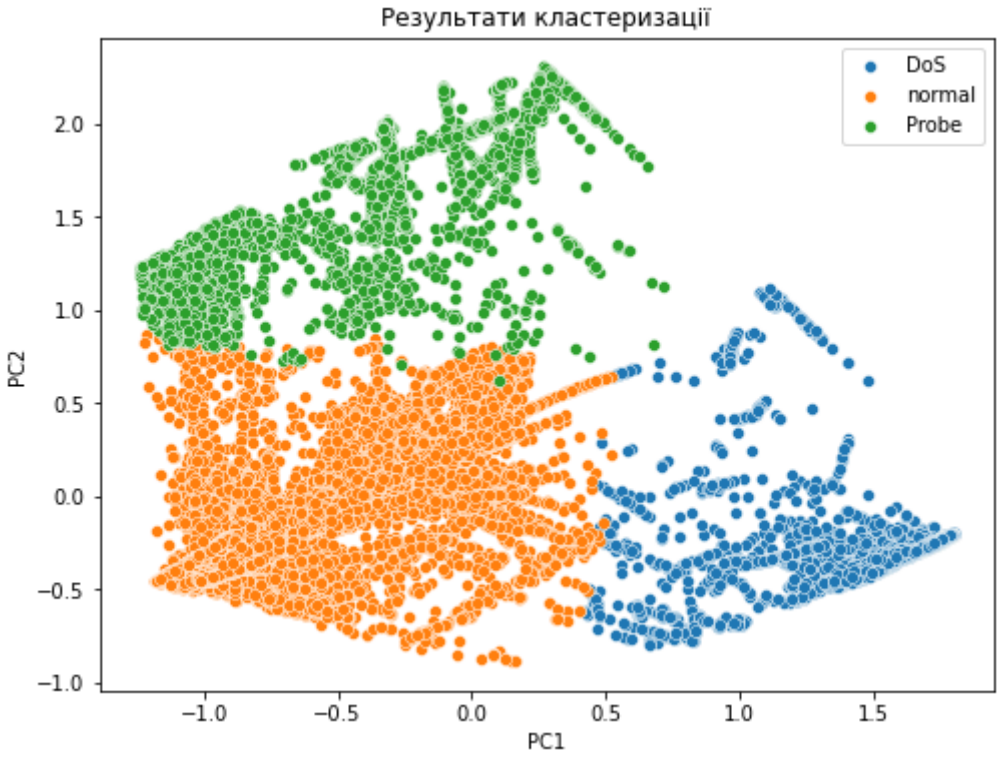

Рисунок 3.3 – Результат кластеризації методом k-means

Візуалізуємо також звіт, де показано точність у відсотковому значенні:

Таблиця 1.2 Класифікаційний звіт. Метод k-means

	Precision	Recall	F1-Score	Support
DoS	0.98	0.75	0.85	45927
Probe	0.35	0.45	0.39	11656
Normal	0.85	0.96	0.90	67343
Accuracy			0.83	125971

Маючи вище отримані результати, можемо зробити висновок:

Використовуючи метод «Ліктя» та «Силуета» було визначено кількість кластерів для кожної моделі. Кластери на графіках мають різні форми, розміри та щільність. Кластеризацію можна вважати успішно виконаною, оскільки атаки типу DoS та Probe були визначені, проте R2L та U2R не визначені через малу кількість об’єктів в даних. Точність кластеризації складає 83%.

3.4. Розв’язок задачі на основі методу Meanshift

Вибір параметрів методу Meanshift є критичним для ефективної роботи алгоритму кластеризації. Два ключові параметри, які впливають на результати методу, це – радіус зсуву та функція ядра.

Радіус зсуву визначає, яка відстань вважається прийнятною для зсуву центроїда. Велике значення радіусу може призвести до великих кластерів, деякі з яких можуть бути надмірно об'єднані. Занадто мале значення радіусу може призвести до виділення об'єктів в окремих кластерах, навіть якщо вони належать до одного більшого кластера.

Функція ядра визначає, як вага розподілена навколо кожного об'єкта під час розрахунку центроїдів. Різні функції ядра можуть впливати на швидкість збіжності алгоритму та роздільну здатність. Популярні функції ядра включають гаусівську (розподіл Гауса) та прямокутну функції.

Алгоритм Meanshift є ітеративним методом, який намагається знайти локальні максимуми у густині даних.

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

Де: $m(x)$ – вектор, який обчислюється як різниця між поточним центром кластеру та середнім значенням точок в його околі.

Основна ідея полягає у переміщенні кожної точки даних в напрямку більшого густини, поки точки не збігнуться в локальні максимуми густини.

Кроки реалізації алгоритму Meanshift:

1. Ініціалізація: Обираємо початкові центри для кожної точки даних.
2. Обчислення вагових коефіцієнтів: Для кожної точки обчислюємо вагові коефіцієнти в залежності від відстані від поточного центру.
3. Обчислення нового центру: Обчислюємо новий центр для кожної точки, як середню ваговану позицію всіх точок, які потрапляють у ваговий радіус.
4. Переміщення до нового центру: Переміщуємо кожну точку до нового обчисленого центру.

5. Повторення: Повторюємо кроки 2-4 до збігу центрів або до досягнення заданої точності.
6. Формування кластерів: Після збігу центрів, точки, які збіглися до одного центру, вважаються належними до одного кластера.

Алгоритм продовжує ітерації до збігу центрів кластерів. Оскільки Meanshift не вимагає заздалегідь визначеної кількості кластерів, він може знаходити динамічну кількість кластерів відповідно до густини даних.

Для реалізації алгоритму Meanshift можна використовувати програмні бібліотеки, такі як scikit-learn у Python, які містять готові функції для виконання цього алгоритму.

Розв'язок задачі на основі методу Meanshift

Код що використовувався:

```
#Виконання PCA
pca = PCA()
pca.fit(data_scaled)
# Оцінка оптимального bandwidth
bandwidth = estimate_bandwidth(data_scaled, quantile=0.3, n_samples=500)
# Застосування MeanShift
ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)
ms.fit(data_scaled)
clusters = ms.labels_
# Додавання міток кластерів до датафрейму
data_cleaned['cluster'] = clusters
# Обчислення силуетного коефіцієнта
if len(np.unique(clusters)) > 1:
    silhouette_avg = silhouette_score(data_scaled, clusters)
    print(f'Силуетний коефіцієнт: {silhouette_avg}')
else:
    print('Силуетний коефіцієнт не може бути обчислений для одного кластера')
# Виконання PCA
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
# Створення DataFrame для PCA даних
pca_df = pd.DataFrame(data=data_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = clusters
cluster_labels = {
    0: 'normal',
    1: 'DoS',
    2: 'Probe',
    3: 'U2R',
    4: 'R2L'
```

```
}
plt.figure(figsize=(8, 6))
for cluster_number, cluster_name in cluster_labels.items():
    # Відбір даних для кожного кластеру
    cluster_subset = pca_df[pca_df['cluster'] == cluster_number]

    # Будування графіка для кластеру
    sns.scatterplot(x='PC1', y='PC2', data=cluster_subset, label=cluster_name)

plt.title('Результати кластеризації')
plt.legend()
plt.show()
```

Результат роботи програмного коду відображається на рис.3.4:

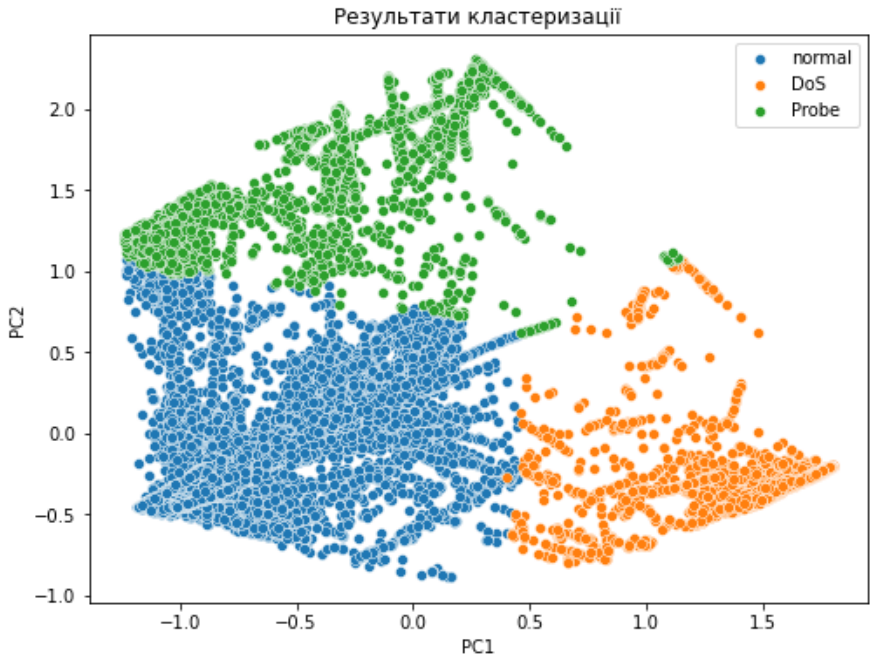

Рисунок 3.4 – Результат кластеризації методом Meanshift

Таблиця 1.3 Класифікаційний звіт. Метод Meanshift

	Precision	Recall	F1-Score	Support
DoS	0.98	0.75	0.85	45927
Probe	0.40	0.55	0.48	11656
Normal	0.85	0.96	0.90	67343
Accuracy			0.85	125971

Підведемо підсумки застосування методу Meanshift:

Оскільки даний метод базується на щільності, він може ідентифікувати викиди як окремі кластери, що може бути корисним при спробах виявлення аномальної поведінки. На графіках видно, що були чітко визначені кластери,

їх розподіл. Атаки R2L та U2R не визначені, проте точність кластеризації дорівнює 85%, завдяки визначенню нормальних даних, атак DoS та Probe.

3.5. Розв'язок задачі на основі методу Spectral Clustering

Для методу Spectral Clustering є важливим етапом вибір параметрів, оскільки це впливає на якість та ефективність кластеризації.Реалізація методу Spectral Clustering вимагає ретельної підготовки та налаштування параметрів, а також використання відповідних бібліотек та інструментів для обчислень та аналізу.

Цей метод використовує властивості графа даних для визначення кластерів. В основі методу лежить матриця суміжності A або матриця подібності, з якої отримують матрицю Лапласа:

$$L = D - A$$

Де: D – діагональна матриця ступенів вершин; A – матриця суміжності.

Після застосування методу Spectral Clustering до вхідних даних і отримання кластерів важливо провести візуалізацію результатів, щоб краще зрозуміти структуру та розподіл об'єктів у просторі ознак. Можна використати: Scatter Plot, Heatmap, 3D Plot, Network Graph, PCA або t-SNE.

Розв'язок задачі на основі методу Spectral Clustering

Код що використовувався:

```
gamma = 1.0
similarity_matrix = rbf_kernel(data_reduced, gamma=gamma)
# Визначення оптимальної кількості кластерів
optimal_clusters = 3
spectral = SpectralClustering(n_clusters=optimal_clusters, affinity='precomputed',
eigen_solver='arpack')
# Виконання спектральної кластеризації
clusters = spectral.fit_predict(similarity_matrix)
if len(np.unique(clusters)) > 1:  # Силуетний коефіцієнт вимагає більше ніж одного кластера
    silhouette_avg = silhouette_score(similarity_matrix, clusters)
    print(f'Силуетний коефіцієнт: {silhouette_avg}')
else:
    print('Силуетний коефіцієнт не може бути обчислений для одного кластера')
# Створення DataFrame для PCA даних
```

```
pca_df = pd.DataFrame(data=data_reduced, columns=['PC1', 'PC2'])
pca_df['cluster'] = clusters  # Додаємо мітки кластерів
cluster_labels = {
    0: 'normal',
    1: 'Probe',
    2: 'DoS',
    3: 'R2L',
    4: 'U2R'
}
plt.figure(figsize=(8, 6))
for cluster_number, cluster_name in cluster_labels.items():
    # Відбір даних для кожного кластеру
    cluster_subset = pca_df[pca_df['cluster'] == cluster_number]

    # Будування графіка для кластеру
    sns.scatterplot(x='PC1', y='PC2', data=cluster_subset, label=cluster_name)

plt.title('Результати кластеризації')
plt.legend()
plt.show()
```

Результат роботи програмного коду відображається на рис.3.5:

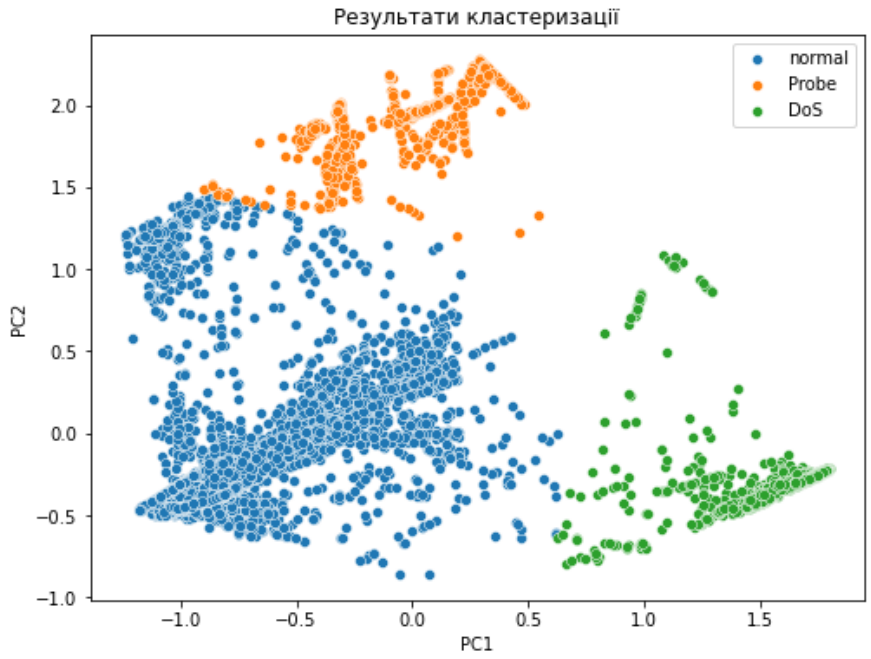

Рисунок 3.5 – Результат кластеризації методом Spectral Clustering

Таблиця 1.4 Класифікаційний звіт. Метод Spectral Clustering

	Precision	Recall	F1-Score	Support
DoS	0.98	0.74	0.85	6858
Probe	0.43	0.45	0.44	1847
Normal	0.85	0.99	0.92	10051

Accuracy			0.84	18895

Використовуючи метод Spectral Clustering, робимо висновок:

Під час роботи із цим методом, виникла проблема із обробкою даних – апаратне забезпечення не змогло виконати розрахунки у повному обсязі даних. Задля забезпечення експерименту, було вирішено виконати семплювання даних і взяти 15% даних, при більших значеннях видавало помилку, щодо нестачі ресурсів системи. Спектральне кластерування успішно виконано, були визначені атаки типу DoS і Probe, але через семплювання інші типи могли не увійти до 15%. Тому для виконання експерименту у повному обсязі необхідно мати потужне апаратне забезпечення. Точність кластеризації даної моделі складає 84%.

3.6. Розв’язок задачі на основі методу Agglomerative Clustering

В агломеративному кластерному методі необхідно визначити декілька параметрів, які впливатимуть на процес об'єднання об'єктів у кластери. Одним із ключових параметрів є метрика відстані, яка вказує, яким чином обчислюватиметься відстань між об'єктами. Вибір правильної метрики може значно вплинути на структуру кластерів.

Такий метод є методом ієрархічної кластеризації, який починає з того, що розглядає кожну точку як окремий кластер, а потім поступово об'єднує кластери на основі визначеної міри подібності. Основна ідея може бути представлена так:

$$d(C_i, C_j) = min_{x \in C_{i, y \in C_j}} d(x, y)$$

Де: $d(C_i, C_j)$ – відстань/подібність між двома кластерами; $d(x, y)$ – відстань між елементами x та y.

Крім того, необхідно визначити кількість кластерів, на яку будуть об'єднані об'єкти. В цьому випадку можна використовувати евристику або ж

використовувати методи внутрішньокластерного та міжкластерного розбиття для визначення оптимальної кількості кластерів.

Алгоритм Agglomerative Clustering базується на принципі поетапного об'єднання найближчих об'єктів або кластерів. Починаючи з кожного об'єкта власним кластером, алгоритм поступово об'єднує об'єкти в кластери до досягнення певної кількості кластерів або зупинки згідно з заданими умовами.

Основні кроки реалізації алгоритму Agglomerative Clustering:

1. Ініціалізація: Почати з кожного об'єкта як окремого кластера.
2. Обчислення відстаней: Обчислити матрицю відстаней між усіма парами кластерів або об'єктів, використовуючи обрану метрику відстані.
3. Об'єднання найближчих: Знайти два найближчих кластера або об'єкта на основі обчисленої матриці відстаней і об'єднати їх в новий кластер.
4. Оновлення матриці відстаней: Оновити матрицю відстаней, виключивши об'єкти або кластери, які були об'єднані, і додавши новий кластер.
5. Повторення: Повторювати кроки 3-4 до тих пір, поки не буде досягнуто потрібної кількості кластерів або зупинено згідно з заданими умовами.
6. Завершення: На цьому етапі кожен об'єкт буде належати до одного з кластерів, і кінцеві результати кластеризації будуть отримані.

Розв’язок задачі на основі методу Agglomerative Clustering

Код що використовувався:

```
linked = linkage(data_reduced, method='ward')  # Використання методу Ward для створення ієрархічного кластерування
plt.figure(figsize=(10, 7))
dendrogram(linked,
           orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Дендрограма')
plt.show()
# Ініціалізація списків
inertia = []
silhouette_scores = []
# Виконання k-means для різної кількості кластерів
clusters_range = range(1, 9)
for k in clusters_range:
    kmeans = KMeans(n_clusters=k, random_state=1)
```

```
    kmeans.fit(data_scaled)
    inertia.append(kmeans.inertia_)

    # Розрахунок силуетного коефіцієнта
    if k > 1:
        silhouette = silhouette_score(data_scaled, kmeans.labels_)
        silhouette_scores.append(silhouette)
# Виведення результатів
print("Метод Ліктя (Сума квадратів відстаней):", inertia)
print("Метод Силуетів (Силуетний коефіцієнт):", silhouette_scores)
# Визначення оптимальної кількості кластерів
optimal_clusters = 3
# Агломеративна кластеризація
agg_cluster = AgglomerativeClustering(n_clusters=optimal_clusters)
clusters = agg_cluster.fit_predict(data_reduced)
# Створення DataFrame для PCA даних
pca_df = pd.DataFrame(data=data_reduced, columns=['PC1', 'PC2'])
pca_df['cluster'] = clusters  # Додаємо мітки кластерів
cluster_labels = {
    0: 'normal',
    1: 'DoS',
    2: 'Probe',
    3: 'R2L',
    4: 'U2R'
}
plt.figure(figsize=(8, 6))
for cluster_number, cluster_name in cluster_labels.items():
    # Відбір даних для кожного кластеру
    cluster_subset = pca_df[pca_df['cluster'] == cluster_number]

    # Будування графіка для кластеру
    sns.scatterplot(x='PC1', y='PC2', data=cluster_subset, label=cluster_name)
plt.title('Результати кластеризації')
plt.legend()
plt.show()
```

Результат роботи програмного коду відображається на рис.3.6 і 3.7:

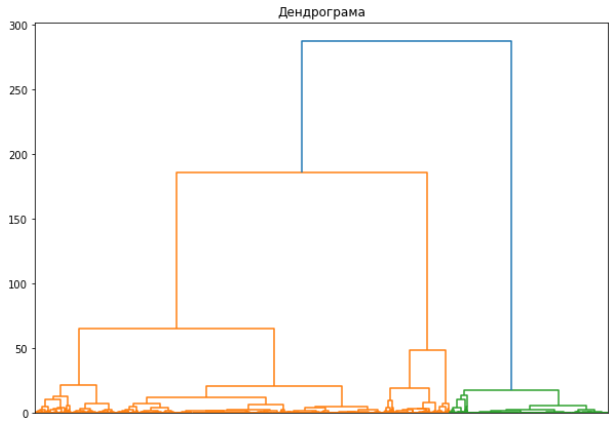

Рисунок 3.6 – Дендрограма

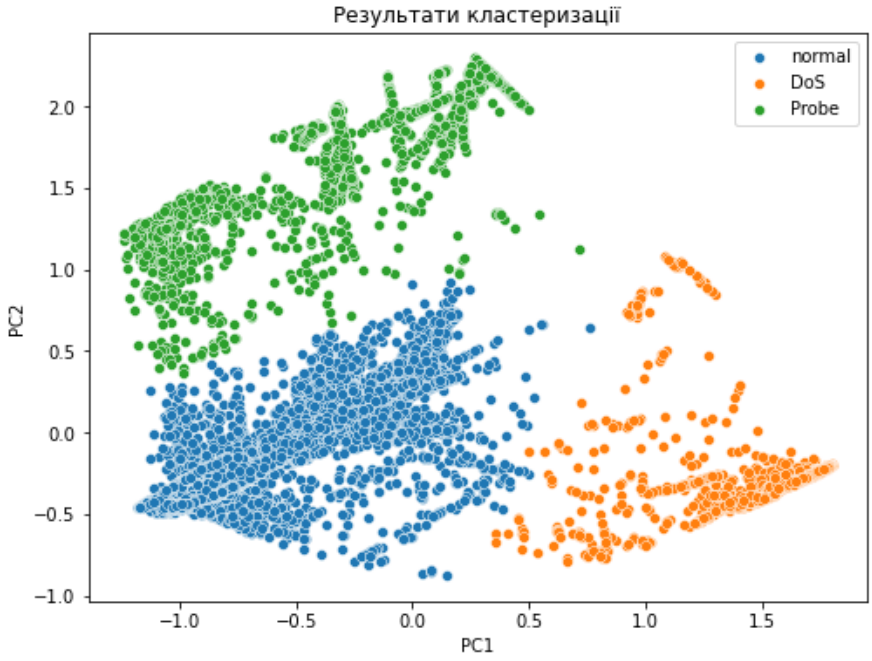

Рисунок 3.7 – Результат кластеризації методом Agglomerative Clustering

Таблиця 1.5 Класифікаційний звіт. Метод Agglomerative Clustering

	Precision	Recall	F1-Score	Support
DoS	0.98	0.75	0.85	13779
Probe	0.33	0.44	0.38	3468
Normal	0.84	0.95	0.89	20203
Accuracy			0.82	37791

Зробимо висновки щодо використання цього методу:

Як і Spectral Clustering, цей метод вимагає велику кількість обчислювальних ресурсів для виконання. Тому для проведення експерименту було виконано семплювання і взято 30% даних з датасету.

Дендрограма дозволила визначити оптимальну кількість кластерів для даної моделі. Цей метод кластеризації відобразив різну щільність та форму, що може вказувати на різноманітність типів поведінки в даних. Як бачимо, цей метод також не зміг виявити атаки R2L та U2R. Точність даної моделі складає 82%.Можна вважати, що відхилення від основних груп кластерів – аномалія, шум, атака. Проте, як бачимо, даний метод визначив основні групи даних без деталей, що приводить даний алгоритм до подальшого покращення або припинення його застосування.

3.7. Розв’язок задачі на основі методу DBSCAN

Це алгоритм, який кластеризує точки на основі щільності. Він визначає кластери як області високої щільності, розділені областями низької щільності.

$$N_\varepsilon(p) = \{q \in D | dist(p,q) \leq \varepsilon\}$$

Де: $N_\varepsilon(p)$ – сусідство точки p; D – набір даних; $dist(p,q)$– метрика відстані (евклідова).

Параметри методу DBSCAN, такі як радіус та мінімальна кількість точок, відіграють ключову роль у визначенні кластерів та аномалій. Вибір оптимальних значень цих параметрів важливий для отримання якісних результатів кластеризації.

Радіус: цей параметр визначає максимальну відстань між двома точками, які можуть вважатися сусідніми. Велике значення радіусу може призвести до злиття кластерів, тоді як замале значення може призвести до виділення окремих точок як аномалій. Варто експериментувати з різними значеннями та спостерігати за результатами.

Мінімальна кількість точок: цей параметр визначає мінімальну кількість точок, яка повинна бути в сусідньому радіусі, щоб точка вважалася ядром кластера. Мале значення може призвести до утворення багатьох кластерів, включаючи шумові, тоді як велике значення може призвести до об'єднання багатьох точок у один кластер.

Також слід враховувати природу даних та специфіку завдання, оскільки вибір параметрів може бути впливований обсягом даних, розміром кластерів та розподілом точок.

Алгоритм DBSCAN (Density-Based Spatial Clustering of Applications with Noise) є одним із популярних методів кластеризації, який базується на щільності точок у просторі. Давайте розглянемо кроки реалізації цього алгоритму:

1. Вибір параметрів: Першим кроком є вибір параметрів алгоритму: радіус ε (epsilon) і мінімальна кількість точок (minPts), які визначають, коли

точка вважається ядром кластера і коли точки вважаються сусідами. Ці параметри визначають розмір та щільність кластерів.

2. Обчислення сусідів: Для кожної точки обчислюються її сусіди, які знаходяться на відстані не більше ε. Ядро кластера включає точку, якщо її кількість сусідів перевищує minPts.
3. Формування кластерів: Для кожної точки ядра кластера обчислюється рекурсивно всі її сусіди, які також є ядрами кластера. Цей процес триває до того моменту, поки не будуть враховані всі точки, що належать до кластера.
4. Обробка шуму: Точки, які не є ядрами жодного кластера і не мають достатньо сусідів, щоб створити свій власний кластер, вважаються шумом.
5. Побудова кластерів: Кожен сформований кластер містить точки, які є сусідами одна для одної. Однак, точки можуть належати до декількох кластерів, якщо вони є сусідами для кількох ядер кластерів.

Після завершення цих кроків алгоритм DBSCAN відзначить кожну точку як належну певному кластеру або як шум. Цей підхід дає можливість автоматично виявляти кластери будь-якої форми та виділяти аномалії як окремі точки, які не входять до жодного кластера.

Розв’язок задачі на основі методу DBSCAN

Код що використовувався:

```
# eps
nearest_neighbors = NearestNeighbors(n_neighbors=5)
neighbors = nearest_neighbors.fit(data_scaled)
distances, indices = neighbors.kneighbors(data_scaled)
# Сортування відстаней
distances = np.sort(distances, axis=0)
distances = distances[:, 1]
plt.plot(distances)
plt.title("Графік найближчих відстаней")
plt.xlabel("Точки")
plt.ylabel("Відстань до найближчого сусіда")
plt.show()
#Виконання PCA
pca = PCA()
```

```
pca.fit(data_scaled)
# Кластеризація
dbscan = DBSCAN(eps=0.3, min_samples=2000)
clusters = dbscan.fit_predict(data_scaled)
# Додавання міток кластерів до датафрейму
data_cleaned['cluster'] = clusters
# Обчислення силуетного коефіцієнта
if len(np.unique(clusters)) > 1:
    silhouette_avg = silhouette_score(data_scaled, clusters)
    print(f'Силуетний коефіцієнт: {silhouette_avg}')
else:
    print('Силуетний коефіцієнт не може бути обчислений для одного кластера')
# Виконання PCA для зменшення
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
# Створення DataFrame для PCA даних
pca_df = pd.DataFrame(data=data_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = clusters  # Додаємо мітки кластерівcluster_labels = {
    0: 'DoS',
    1: 'normal',
    2: 'U2R',
    3: 'Probe',
    4: 'R2L'
}
plt.figure(figsize=(8, 6))
for cluster_number, cluster_name in cluster_labels.items():
    # Відбір даних для кожного кластеру
    cluster_subset = pca_df[pca_df['cluster'] == cluster_number]

    # Будування графіка для кластеру
    sns.scatterplot(x='PC1', y='PC2', data=cluster_subset, label=cluster_name)
plt.title('Результати кластеризації')
plt.legend()
plt.show()
```

Результат роботи програмного коду відображається на рис.3.8, 3.9:

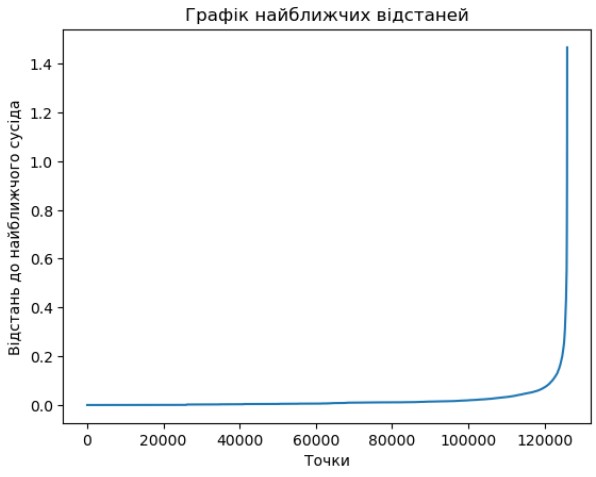

Рисунок 3.8 – визначення eps для DBSCAN

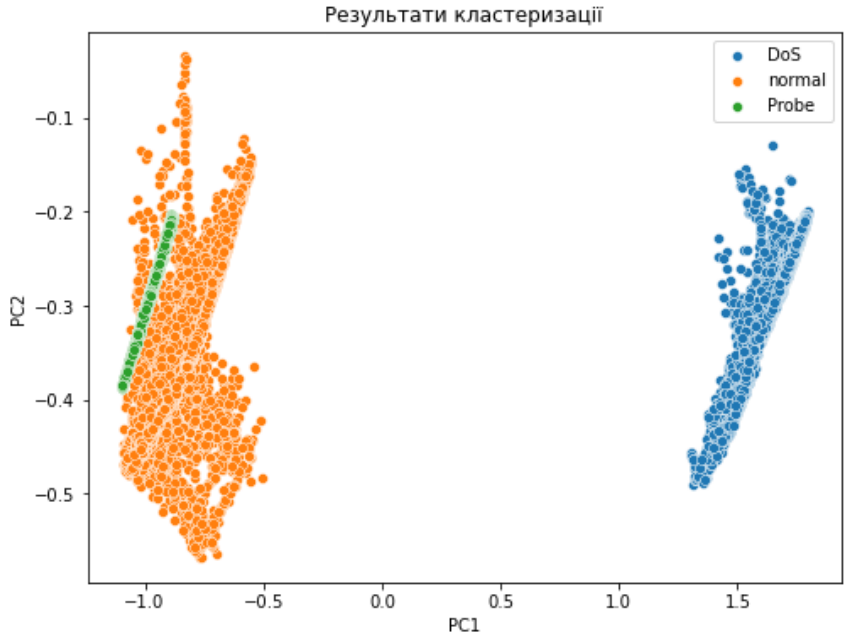

Рисунок 3.9 – Результат кластеризації методом DBSCAN

Таблиця 1.6 Класифікаційний звіт. Метод DBSCAN

	Precision	Recall	F1-Score	Support
DoS	1.00	0.72	0.83	45927
Probe	1.00	0.25	0.39	11656
Normal	1.00	0.61	0.76	67343
Accuracy			0.91	125971

Висновки відносно методу DBSCAN:

Даний метод є потужним інструментом для виявлення кластерів у складних наборах даних, тому його використання для визначення аномалій/відхилень є досить вражаючим. Він ідентифікував із 100% точністю атаки DoS, Probe та нормальні дані, проте через малу кількість об’єктів типу R2L і U2R не зміг визначити їх групи. Точність даної моделі складає 91%.

3.8 Розв’язок задачі на основі методу BIRCH

Під час використання методу BIRCH для кластеризації даних, важливий правильний вибіпараметрів, які відповідають особливостям даних та меті кластеризації.

Для визначення оптимальних параметрів можна використовувати методи сіткового пошуку ("grid search") або перехресну перевірку ("cross-validation"). Також важливо зрозуміти, яку інформацію ви хочете отримати з кластеризації та як вона вплине на ваші подальші аналізи.

Алгоритм BIRCH є ієрархічним методом кластеризації, який об'єднує об'єкти в структуру дерева, відому як CF-дерево (Clustering Feature Tree). Цей алгоритм підходить для великих наборів даних, оскільки він спрощує процес кластеризації шляхом побудови дерева.

Основні кроки реалізації алгоритму BIRCH:

1. Побудова CF-дерева: Спочатку об'єкти додаються до CF-дерева, де кожен вузол представляє кластер або підкластер. Для кожного вузла зберігаються статистичні характеристики, такі як середнє значення та кількість об'єктів.
2. Об'єднання вузлів: Під час вставки нового об'єкта до дерева перевіряється його відстань до наявних вузлів. Якщо об'єкт віддалений від усіх існуючих вузлів, він додається як новий вузол. Якщо об'єкт ближчий до існуючого вузла, то він додається до відповідного вузла, а статистика цього вузла оновлюється.
3. Виконання кластеризації: З CF-дерева можна виділити кластери, вибравши вузли з певною вартістю статистики, яка вказує на значущі групи об'єктів. Зазвичай ця вартість є порогом і визначається користувачем.
4. Візуалізація результатів: Отримані кластери можна візуалізувати на графіку, що допомагає зрозуміти структуру даних та їхні взаємозв'язки.

5. Аналіз результатів: Для кожного кластера можна провести аналіз, визначивши характеристики, які властиві цій групі об'єктів, та визначивши важливі атрибути, які внесли найбільший вплив на формування кластерів.

Алгоритм BIRCH відмінно підходить для кластеризації великих об'ємів даних, оскільки він має низьку складність та використовує ієрархічну структуру для зберігання та організації об'єктів.

Розв’язок задачі на основі методу BIRCH

Код що використовувався:

```
#Виконання PCA
pca = PCA()
pca.fit(data_scaled)
# Виконання кластеризації
birch_model = Birch(threshold=1.1, branching_factor=50)
clusters = birch_model.fit_predict(data_scaled)
# Додавання міток кластерів до датафрейму
data_cleaned['cluster'] = clusters
# Обчислення силуетного коефіцієнта
if len(np.unique(clusters)) > 1:
    silhouette_avg = silhouette_score(data_scaled, clusters)
    print(f'Силуетний коефіцієнт: {silhouette_avg}')
else:
    print('Силуетний коефіцієнт не може бути обчислений для одного кластера')
# Виконання PCA для зменшення
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
# Створення DataFrame для PCA даних
pca_df = pd.DataFrame(data=data_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = clusters  # Додаємо мітки кластерів
cluster_labels = {
    0: 'Probe',
    1: 'DoS',
    2: 'normal',
    3: 'R2L',
    4: 'U2R'
}
plt.figure(figsize=(8, 6))
for cluster_number, cluster_name in cluster_labels.items():
    # Відбір даних для кожного кластеру
    cluster_subset = pca_df[pca_df['cluster'] == cluster_number]

    # Будування графіка для кластеру
    sns.scatterplot(x='PC1', y='PC2', data=cluster_subset, label=cluster_name)
```

```
plt.title('Результати кластеризації')
plt.legend()
plt.show()
```

Результат роботи програмного коду відображається на рис.3.10:

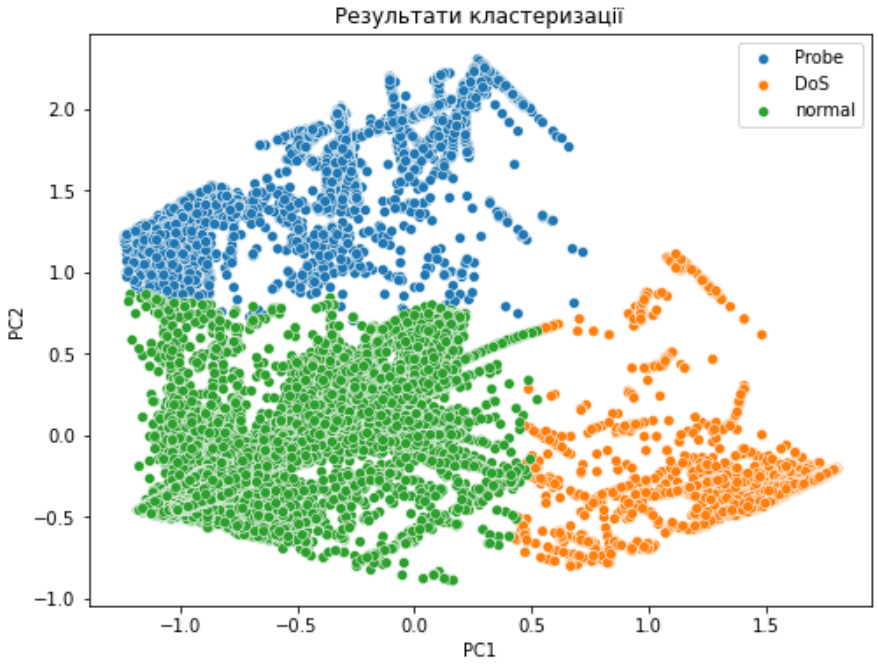

Рисунок 3.10 – Результат кластеризації методом BIRCH

Таблиця 1.7 Класифікаційний звіт. Метод BIRCH

	Precision	Recall	F1-Score	Support
DoS	0.98	0.75	0.85	45927
Probe	0.35	0.45	0.39	11656
Normal	0.85	0.96	0.90	67343
Accuracy			0.83	125971

Після виконання кластеризації, робимо висновки:

Ця модель показала себе доволі непогано. Звон таки, добре визначена атака DoS, Probe – має гірші показники виявлення. R2L та U2R – взагалі не виявлено. Точність моделі складає 83%.

3.9. Розв'язок задачі на основі методу Gaussian Mixture

Gaussian Mixture – ймовірнісний метод, який вважає, що всі дані генеруються з суміші кількох гаусівських розподілів, кожен з яких відповідає кластеру. Ймовірність точки моделюється як:

$$p(x) = \sum_{i=1}^{k} \pi_i \, N(x|\mu_i, \sum i)$$

Де: π_i – апріорна ймовірність кластера i; $N(x|\mu_i, \sum i)$ – ймовірність гаусівського розподілу з середнім μ_i та коваріаційною матрицею $\sum i$.

Для ефективного застосування цього методу, необхідне визначення параметрів, які відповідають за кількість кластерів та тип коваріаційної матриці для кожної компоненти. Оптимальний вибір цих параметрів допомагає отримати якісний розбиток даних на кластери.

Для визначення оптимальної кількості компонентів ми можемо використовувати метод "ліктя". Для цього ми обчислюємо значення інерції для різних кількостей компонентів і спостерігаємо, де відбувається зміна темпу зменшення інерції. Точка, де цей темп зміни змінюється, може вказувати на оптимальну кількість кластерів.

Також, ми можемо врахувати апріорні знання або доменну експертизу, які можуть допомогти визначити оптимальну кількість кластерів для конкретного досліджуваного сценарію.

Для визначення типу коваріаційної матриці ми можемо використовувати параметр "covariance_type". Він визначає, чи припускається однакова коваріаційна матриця для всіх компонентів (параметр "full"), чи використовується діагональна коваріаційна матриця (параметр "diag") або спільна коваріаційна матриця (параметр "tied") для всіх компонентів. Вибір параметра залежить від природи даних та припущень про їхню структуру.

Згідно з нашим набором даних, ми можемо спробувати різні значення параметрів кількості компонентів та типу коваріаційної матриці, і за

допомогою методів оцінки якості кластеризації, таких як Adjusted Rand Index (ARI) і V-міра, обрати найкращий набір параметрів для нашого випадку.

Основні кроки реалізації алгоритму Gaussian Mixture:

1. Імпорт бібліотек: Почніть з імпорту необхідних бібліотек, зокрема sklearn.mixture з scikit-learn.
2. Створення моделі: Створіть об'єкт моделі GaussianMixture та визначте кількість кластерів, яку ви хочете знаходити.
3. Навчання моделі: Використовуйте метод .fit() для навчання моделі на вашому наборі даних. В цьому кроці модель оцінює параметри гаусових розподілів кожного кластеру.
4. Прогнозування кластерів: Використовуйте метод .predict() для призначення кожному зразку вхідних даних ідентифікатора кластеру, до якого він належить.
5. Отримання параметрів: Після навчання можна отримати параметри кожного гаусового розподілу, наприклад, середніх та матриць коваріацій.
6. Візуалізація результатів: Для аналізу результатів кластеризації можна використовувати візуалізацію, таку як діаграми розсіювання.
7. Оцінка якості: Для оцінки якості кластеризації можна використовувати метрики, такі як Adjusted Rand Index, гомогенність, повнота, V-міра тощо.

Реалізація алгоритму Gaussian Mixture дозволяє виявити складні та перетинні кластери в даних, що може бути особливо корисно при аналізі реальних даних з багатьма вимірами.

Розв’язок задачі на основі методу Gaussian Mixture

Код що використовувався:

```
#Виконання PCA
pca = PCA()
pca.fit(data_scaled)
n_components = np.arange(2, 10)
bics = []
aics = []
silhouette_scores = []
```

```
# Визначення к-сті n_components та побудова графіку
for n in n_components:
    gmm = GaussianMixture(n_components=n)
    gmm.fit(data_scaled)
    clusters = gmm.predict(data_scaled)
    bics.append(gmm.bic(data_scaled))
    aics.append(gmm.aic(data_scaled))

    # Обчислення силуетного коефіцієнта, якщо є більше одного кластера
    if n > 1:
        silhouette_avg = silhouette_score(data_scaled, clusters)
        silhouette_scores.append(silhouette_avg)
    else:
        silhouette_scores.append(None)
plt.figure(figsize=(8, 8))
# BIC і AIC
plt.plot(n_components, bics, label='BIC')
plt.plot(n_components, aics, label='AIC')
plt.title('BIC та AIC')
plt.xlabel('Кількість компонентів')
plt.ylabel('Значення критерію')
plt.legend()
plt.grid(True)
# Силуетний коефіцієнт
plt.figure(figsize=(8, 8))
# Важливо використовувати n_components[1:], оскільки для n=1 силуетний коефіцієнт не
визначений
plt.plot(n_components[1:], silhouette_scores[1:], label='Силуетний коефіцієнт')
plt.title('Силуетний коефіцієнт для різної кількості компонентів GMM')
plt.xlabel('Кількість компонентів')
plt.ylabel('Силуетний коефіцієнт')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
# Кластеризація
gmm = GaussianMixture(n_components=4)
gmm.fit(data_scaled)
clusters = gmm.predict(data_scaled)
# Додавання міток кластерів до датафрейму
data_cleaned['cluster'] = clusters
# Виконання PCA для зменшення
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
# Створення DataFrame для PCA даних
pca_df = pd.DataFrame(data=data_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = clusters  # Додаємо мітки кластерів
cluster_labels = {
    0: 'normal',
    1: 'DoS',
```

```
    2: 'R2L',
    3: 'Probe',
    4: 'U2R'
}
plt.figure(figsize=(8, 6))
for cluster_number, cluster_name in cluster_labels.items():
    # Відбір даних для кожного кластеру
    cluster_subset = pca_df[pca_df['cluster'] == cluster_number]

    # Будування графіка для кластеру
    sns.scatterplot(x='PC1', y='PC2', data=cluster_subset, label=cluster_name)

plt.title('Результати кластеризації')
plt.legend()
plt.show()
```

Результат роботи програмного коду відображається на рис.3.11:

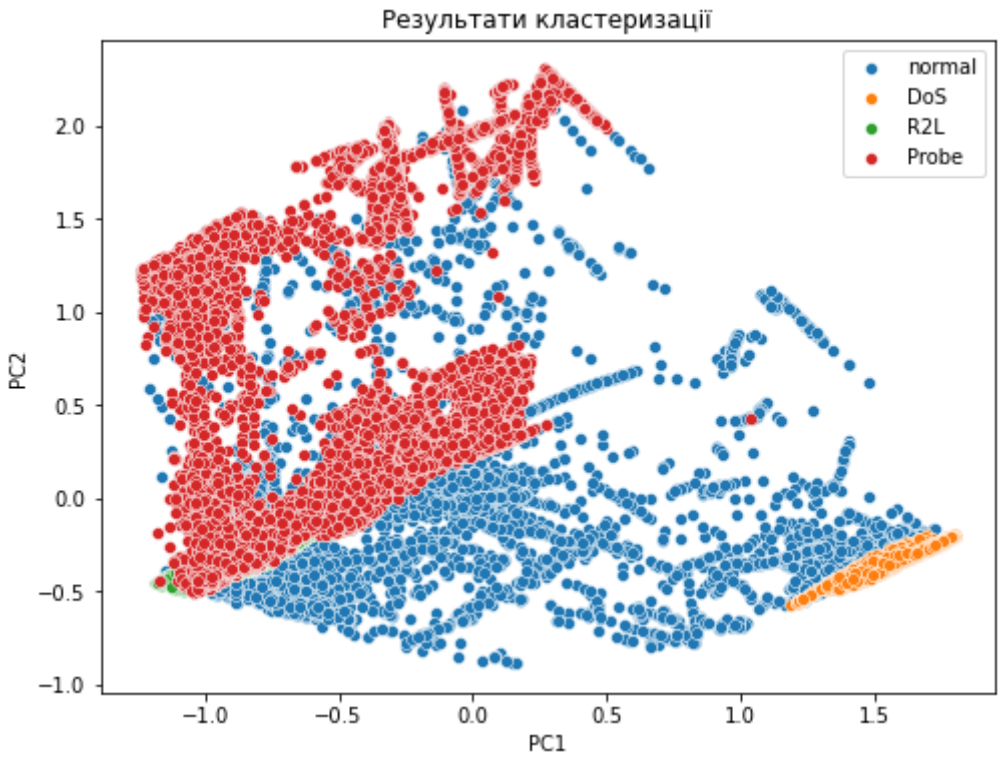

Рисунок 3.11 – Результат кластеризації методом Gaussian Mixture

Таблиця 1.8 Класифікаційний звіт. Метод Gaussian Mixture

	Precision	Recall	F1-Score	Support
DoS	1.00	0.70	0.82	45927
Probe	0.27	0.55	0.37	11656
R2L	0.01	0.46	0.02	993
Normal	0.50	0.30	0.40	67343
Accuracy			0.42	125971

Висновки, щодо методу Gaussian Mixture:

Даний статистичний метод кластеризації показав доволі погані показники. Атаки DoS визначені на 100%; Probe – 27%, що вказує на проблеми моделі із виявленням такої групи значень; R2L – єдиний метод, який зміг визначити цю групу атак, але лише у обсязі 1%; U2R – взагалі не виявлено. Нормальні дані визначені на 50%. Загальна точність моделі кластеризації складає 42%.

3.10 Розв'язок задачі на основі методу OPTICS

OPTICS, подібно до DBSCAN, використовує щільність для визначення кластерів, але замість того, щоб виділяти кластери за фіксованим ε, він створює упорядкованість точок, яка дозволяє визначати кластери на основі щільності

$$Reachability - Distance_{\varepsilon,MinPts}(p,o) = max\{Core - Distance_{\varepsilon,MinPts}(p), dist(p,o)\}$$

Де: $Core - Distance$ – найменша відстань між p та $MinPts$ її найближчих сусідів.

Для успішного застосування алгоритму OPTICS необхідно правильно налаштувати його параметри. Одними з ключових параметрів є «min_samples» та «eps», які визначаються в процесі налаштування методу.

Параметр «min_samples» визначає мінімальну кількість сусідніх точок, необхідних для того, щоб точка була розглянута як основна. Збільшення цього значення може призвести до менших кластерів, а також до ігнорування менших аномалій. Зменшення значення «min_samples» може призвести до виявлення більшої кількості кластерів та аномалій. Вибір оптимального значення цього параметру потребує деякої експертної оцінки та експериментів.

Параметр «eps» визначає максимальну відстань між точками, за якою вони вважаються сусідніми. Більше значення «eps» означатиме більші кластери та меншу деталізацію групування, а менше значення – менші та більш згорнуті кластери. Вибір оптимального значення «eps» також зазвичай вимагає експериментів та аналізу результатів.

Під час вибору параметрів важливо враховувати конкретні особливості вашого набору даних, його розмір та природу. Оптимальний вибір цих параметрів допоможе досягнути збалансованого та інформативного результату при використанні методу OPTICS.

Розв’язок задачі на основі методу OPTICS

Код що використовувався для тренування моделі:

```
#Виконання PCA
pca = PCA()
pca.fit(data_scaled)
# Виконання кластеризації
optics = OPTICS(min_samples=3500, min_cluster_size=52, max_eps=1)
clusters = optics.fit_predict(data_scaled)
# Додавання міток кластерів до датафрейму
data_cleaned['cluster'] = clusters
# Обчислення силуетного коефіцієнта
if len(np.unique(clusters)) > 1:
    silhouette_avg = silhouette_score(data_scaled, clusters)
    print(f'Силуетний коефіцієнт: {silhouette_avg}')
else:
    print('Силуетний коефіцієнт не може бути обчислений для одного кластера')
# Виконання PCA для зменшення
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
# Створення DataFrame для PCA даних
pca_df = pd.DataFrame(data=data_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = clusters  # Додаємо мітки кластерів
cluster_labels = {
    0: 'normal',
    1: 'DoS',
    2: 'Probe',
    3: 'R2L',
    4: 'U2R'
}
plt.figure(figsize=(8, 6))
for cluster_number, cluster_name in cluster_labels.items():
    # Відбір даних для кожного кластеру
    cluster_subset = pca_df[pca_df['cluster'] == cluster_number]

    # Будування графіка для кластеру
    sns.scatterplot(x='PC1', y='PC2', data=cluster_subset, label=cluster_name)
plt.title('Результати кластеризації')
plt.legend()
plt.show()
```

Результат роботи програмного коду відображається на рис.3.12:

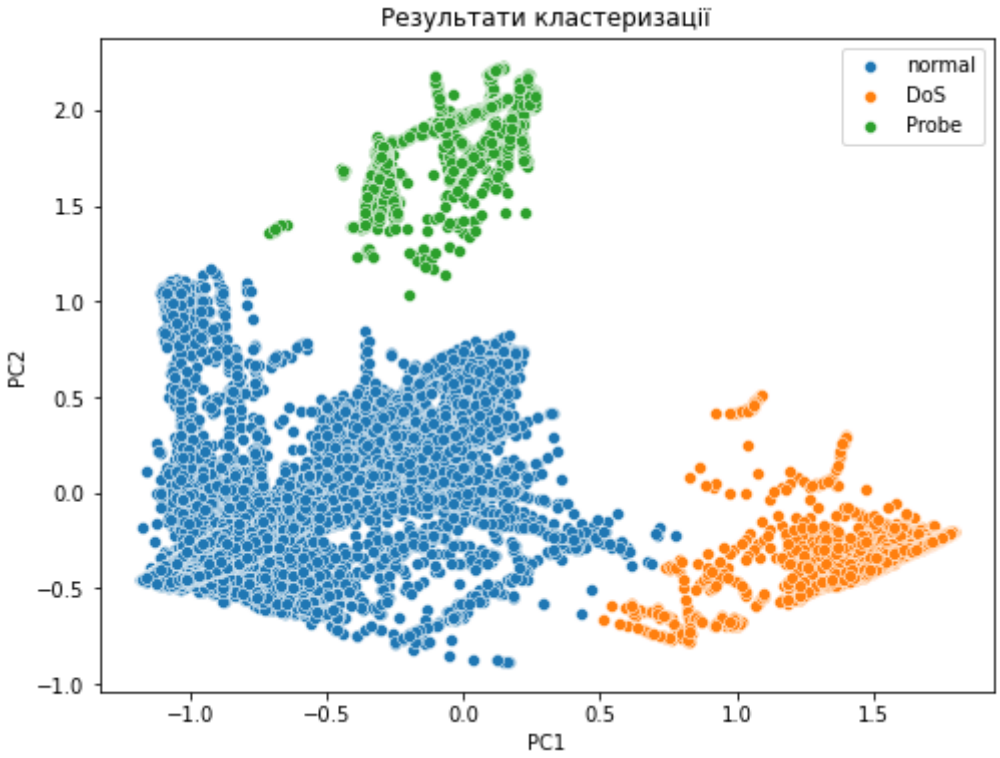

Рисунок 3.12 – Результат кластеризації методом OPTICS

Таблиця 1.9 Класифікаційний звіт. Метод OPTICS

	Precision	Recall	F1-Score	Support
DoS	1.00	0.75	0.85	45927
Probe	0.19	0.14	0.17	11656
Normal	0.85	0.96	0.90	67343
Accuracy			0.80	125971

Підведемо підсумки використання методу OPTICS:

Цей метод виявився найбільш «прискіпливим». Вимагає велику кількість обчислювальних ресурсів. Обробка даних займає багато часу. Дуже чутливий до вхідних параметрів «min_samples» та «max_eps». При спробі визначити найкращі параметри для моделі, використовуючи GridSearch – сітковий пошук, обчислення могли зайняти доби, а то і тижні.

Модель відмінно визначила нормальні дані, DoS атаки, частково Probe. R2L та U2R – не визначені. Точність моделі складає 80%.

3.11. Порівняння результатів, отриманих різними методами

Для кожного обраного методу кластеризації ми використовували підготовлені дані для тренування моделі. Оскільки метою було кластеризувати дані та виявити аномалії без прив'язки до міток класів, ми зосередилися на структурі даних та їхньому групуванні.

Кожен метод був налаштований з урахуванням його параметрів, які були визначені на попередніх етапах. Наприклад, для методу k-means було визначено кількість кластерів, для методу Meanshift - радіус інтервалу, для методу Spectral Clustering - кількість найбільших власних значень та інші параметри відповідно до їхньої специфіки.

Кожен метод виконувався на зазделегідь препідготовлених даних, і результатом є набір кластерів, до якого призначалися окремі об'єкти на основі їхніх характеристик та взаємозв'язків.

Цей етап експерименту моделей був ключовим для подальшого порівняння їхньої ефективності та аналізу результатів.

Щодо виявлення аномалій, ми аналізували кластери, які мали різну кількість об'єктів або містили об'єкти з відмінними характеристиками. Також, ми звертали увагу на об'єкти. Цей етап допоміг нам зрозуміти, наскільки ефективні наші моделі в розподілі даних на кластери та виявленні аномалій.

Схожість активності: порівнюючи кластери, ми виявили, що деякі типи мережевої активності мають схожі характеристики та були об'єднані в одні кластери. Це вказує на здатність методів кластеризації знаходити групи активності з подібними властивостями.

Виявлені аномалії: визначили, що деякі методи більш успішно виявляють аномалії, тобто незвичні або підозрілі активності. Наприклад, метод k-means, Meanshift та DBSCAN показали високу здатність до виявлення аномалій. Методи Spectral Clustering та Agglomerative Clustering підтвердили свої недоліки стосовно необхідних ресурсів для виконання подібних задач. Метод Gaussian Mixture хоч і показав низькі показники точності, проте це

єдиний метод, який зміг виявити атаки типу R2L. Метод OPTICS потребує дуже багато часу на обробку даних, до того ж – дуже чутливий до вхідних параметрів, що робить більшість експериментів недостовірними.

Зв'язки між кластерами: аналізуючи результати, було помічено, що деякі кластери мають схожі характеристики та можуть мати взаємодію між собою. Це може вказувати на певні зв'язки між різними видами мережевої активності.

Важливість атрибутів: було визначено, що деякі атрибути та параметри даних мають більший вплив на кластеризацію та виявлення аномалій, ніж інші. Наприклад, інтенсивність мережевого трафіку та характеристики з'єднань можуть бути важливими для точності кластеризації.

ВИСНОВКИ

В кваліфікаційній роботі було проведено аналіз активності комп'ютерних мереж з метою виявлення аномалій та кібератак з використанням даних «NSL-KDD», Canadian Institute for Cybersecurity, а також проведено порівняльний аналіз якості роботи різних методів кластеризації. Особливістю досліджуваних даних було те, що вони містили досить широкий набір аномальних активностей різного типу. Були використані такі методи: k-means, MeanShift, Spectral Clustering, Agglomerative Clustering, DBSCAN, BIRCH, OPTICS та Gaussian Mixture.

В результаті, дослідження показали, що найбільш ефективними для виявлення аномалій є такі методи: DBSCAN , Meanshift та k-means.

Кожен метод кластеризації показав свою ефективність, сильні та слабкі сторони:

1. DBSCAN: Accuracy моделі складає складає 91%. Precision дорівнює 100% під час визначення даних типів DoS, Probe та нормальних даних. Об'єкти типів R2L та U2R було не визначено.
2. Meanshift: Accuracy моделі складає 85%. Precision 98% для об'єктів типу DoS, 40% типу Probe та 85% нормальних даних.
3. K-means: Accuracy кластеризації 83%. Precision 98% для об'єктів типу DoS, 35% типу Probe та 85% нормальних даних.
4. Spectral Clustering: Під час роботи із цим алгоритмом виникли труднощі із обробкою даних, що пов'язані з потужністю комп'ютера. Тому для роботи було взято лише 15% даних (18895 об'єктів). Accuracy складає 84%. Precision 98% для об'єктів типу DoS, 43% типу Probe та 85% нормальних даних.
5. Agglomerative Clustering: Також виникли проблеми із обробкою даних, тому було взято для роботи 30% даних (37791 об'єкт). Precision 98% для об'єктів типу DoS, 33% типу Probe та 84% нормальних даних.

6. BIRCH: Accuracy кластеризації даних складає 83%. Precision 98% для об’єктів типу DoS, 35% типу Probe та 85% нормальних даних.
7. Gaussian Mixture: Має найнижчий показник accuracy кластеризації – 42%. Проте це єдиний метод, який зміг визначити з precision 1% об’єктів типу R2L, 100% типу DoS, 27% об’єктів типу Probe та 50% нормальних даних.
8. OPTICS: Найважчий метод з усіх через довгу обробку даних. При великій кількості об’єктів, обробка може тривати дні. Accuracy кластеризації даної моделі складає 80%. Precision для об’єктів типу DoS – 100%, Probe – 19%, нормальні дані – 85%.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Computer networking: a top-down approach (2018) / James F. Kurose, University of Massachusetts, Amherst,Keith W. Ross, NYU and NYU Shanghai.

2. Jain, A. K. (2010). Data clustering: 50 years beyond K-means. Pattern Recognition Letters, 31(8), 651-666.

3. Aggarwal, C. C., & Reddy, C. K. (2014). Data clustering: Algorithms and applications. CRC press.

4. Глотов, М. В., Татарський, Р. В., & Шкедь, О. В. (2008). Кластерний аналіз даних в статистичному пакеті R. Інформаційні технології та комп'ютерна інженерія, 3(2), 111-119

5. Shaojun Huang, Weiying Lin, and Shuhui Bu. (2019). Comparison of K-Means and Gaussian Mixture Model Clustering for Complex Data. In: 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA).

6. Sabzi, S., Sahebjamnia, N., & Cagliano, A. C. (2018). An experimental comparison of partitioning and hierarchical clustering methods. Expert Systems with Applications, 92, 357-373.

7. Кучинський, В. Г., & Гальченко, В. В. (2015). Застосування методів кластеризації для аналізу структури масивів даних. Інформаційні технології та комп'ютерна інженерія, (2), 47-56.

8. Онопченко, В. (2019). Використання мови програмування Python для аналізу даних та розробки веб-додатків. Вісник Національного університету "Львівська політехніка". Комп'ютерні науки та інформаційні технології, (932), 181-191.

9. Rokach, L., & Maimon, O. (2005). Clustering methods. Data mining and knowledge discovery handbook, 321-352

10. Jain, A. K., & Dubes, R. C. (1988). Algorithms for clustering data. Prentice-Hall.

11. Awate, S. P., & Whitaker, R. T. (2015). Evaluation of image segmentation and intensity models on histopathology images from an image analysis contest. Journal of pathology informatics, 6(1), 10.

12. Гуляєва, М. В., & Томчук, А. О. (2019). Методи кластерного аналізу в регіональних дослідженнях. Геополітика та економіка, 1(26), 81-90.

13. Ng, A. Y., Jordan, M. I., & Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. Advances in neural information processing systems, 14, 849-856.

14. Lloyd, S. P. (1982). Least squares quantization in PCM. IEEE Transactions on Information Theory, 28(2), 129-137.

15. Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. Journal of the royal statistical society: series B (methodological), 39(1), 1-38.

16. Гусятинська, І. В., & Боровська, О. В. (2019). Використання методів кластерного аналізу в економічних дослідженнях. Вісник ЖДТУ, (1), 64-72.

17. Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), 226-231.

18. Jain, A. K. (2010). Data clustering: 50 years beyond K-means. Pattern recognition letters, 31(8), 651-666.

19. Ankerst, M., Breunig, M. M., Kriegel, H. P., & Sander, J. (1999). OPTICS: ordering points to identify the clustering structure. ACM Sigmod Record, 28(2), 49-60.

20. Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. ACM Sigmod Record, 25(2), 103-114.

21. Глотов, М. В., Татарський, Р. В., & Шкедь, О. В. (2008). Кластерний аналіз даних в статистичному пакеті R. Інформаційні технології та комп'ютерна інженерія, 3(2), 111-119.

22. Хлань, Н., & Якубовська, І. (2017). Застосування методів кластеризації в управлінні якістю продукції. Молодий вчений, (2), 78-82.

23. Зарицька, І. В. (2016). Кластерний аналіз в оцінці соціально-економічного розвитку регіонів України. Економіка та держава, (11), 19-22.

24. Ward Jr, J. H. (1963). Hierarchical grouping to optimize an objective function. Journal of the American statistical association, 58(301), 236-244.

25. MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. Proceedings of the fifth Berkeley symposium on mathematical statistics and probability (Vol. 1, No. 14, pp. 281-297).

26. Cheng, Y., & Church, G. (2000). Biclustering of expression data. Proceedings of the 8th international conference on intelligent systems for molecular biology, 93-103.

27. Дмитренко, В. О. (2018). Використання мови програмування Python для аналізу даних. Науковий вісник Національного гірничого університету, (2), 92-97.

28. Клапчук, В. В., & Дерев'янко, А. В. (2015). Статистичний аналіз даних. Методи і практика: навчальний посібник. Київ: Фізматліт.

29. Хоменко, О. О., Михальченко, В. А., & Городнічий, О. О. (2018). Аналіз даних із використанням кластерного аналізу. Математичні машини і системи, 2, 30-41.

30. Мелентьєв, А. Г. (2016). Алгоритми кластерного аналізу та їх порівняння. Вісник Національного університету «Львівська політехніка», (846), 119-126.

31. Бондаренко, Ю. В. (2018). Використання методів кластерного аналізу в статистичних дослідженнях. Вісник Київського національного університету імені Тараса Шевченка. Серія: Економіка, (204), 7-11.