НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено
В.о. завідувача кафедри

_____ Микола ГРАЙВОРОНСЬКИЙ
(підпис)

«_____»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною программою «Системи, технології та математичні методи кібербезпеки»
спеціальності: 125 «Кібербезпека»

на тему: Алгоритм класифікації та кластерного аналізу DenStream для вирішення задач з забезпечення інформаційної безпеки

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-74
(шифр групи)

Лихошерст Владислав Романович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, канд.фіз. мат. наук, Микола Владленович Грайворонський _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ - 2021 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Фізико-технічний інститут
Кафедра інформаційної безпеки
Рівень вищої освіти – перший (бакалаврський)
спеціальності: 125 «Кібербезпека»

ЗАТВЕРДЖЕНО

В.о. завідувача кафедри

_____ Микола ГРАЙВОРОНСЬКИЙ

«___» _____ 2021 р.

ЗАВДАННЯ

на дипломну роботу на здобуття ступеня бакалавра студенту

Лихошерсту Владиславу Романовичу

1. Тема дипломної роботи: Алгоритм класифікації та кластерного аналізу DenStream для вирішення задач з забезпечення інформаційної безпеки, науковий керівник канд. фіз. мат. наук, доцент Микола Владленович Грайворонський, затверджений наказом по університету від «__» _____ 2021 р. №

2. Термін подання студентом дипломної роботи 31.05.2021 р.

3. Об’єкт дослідження: інформаційно-комп’ютерна мережа та потоки інформації, що описують її стан.

4. Предмет дослідження: особливості поведінки КМ в ситуаціях здійснення загроз інформаційній безпеці.

5. Перелік завдань, які потрібно розробити:

- вивчити властивості мережевого трафіку, що впливають на ефективність використання методів кластеризації при пошуку аномалій в них;
- дослідити методи виявлення аномальної поведінки мережевого трафіку та адаптація їх до умов використання при потоковому надходженні даних;
- запропонувати метод виявлення аномалій;
- виконати його програмну реалізацію та перевірити ефективність при вирішені заданого класу задач.

6. Перелік ілюстративного матеріалу: 15 ілюстрацій, 4 таблиці.

7. Перелік публікацій:

- «Використання стримінгових алгоритмів кластеризації для виявлення аномалій мережевого трафіку», XIX Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Теоретичні та прикладні проблеми фізики, математики та інформатики» КПІ ім. Ігоря Сікорського, 2021. С. 354-357.

8. Дата видачі завдання 12 квітня 2021р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Вивчення літератури за тематикою дипломного проекту	18.04.2021	
2	Аналіз властивостей мережевого трафіку, що впливають на ефективність використання методів кластеризації при пошуку аномалій в них	25.04.2021	
3	Дослідження методів виявлення аномальної поведінки мережевого трафіку та адаптація їх до умов використання при потоковому надходженні даних	30.04.2021	
4	Аналіз методів для виявлення аномального трафіку, що враховує стрімінговий характер надходження даних	07.05.2021	
5	Виконання програмної реалізації обраного методу	17.05.2021	
6.	Перевірка ефективності обраного методу при вирішені заданого класу задач	24.05.2021	
7.	Оформлення дипломного проекту	29.05.2021	

Студент _____В. Р. Лихошерст
(підпис)

Науковий керівник _____М. В. Грайворонський
(підпис)

РЕФЕРАТ

Обсяг роботи 80 сторінок, 15 ілюстрацій, 4 таблиці, 1 додаток, 11 джерел літератури.

Об’єктом дослідження стала комп’ютерна мережа і потоки інформації, що описують її стан.

Предметом дослідження особливості поведінки комп’ютерної мережі в ситуаціях здійснення загроз інформаційній безпеці.

Метою даної роботи є розробка програмного забезпечення для виявлення аномальної поведінки мережевого трафіку, що здатен врахувати потоковий (стрімінговий) характер надходження даних та оснований на ідеях кластерного аналізу.

Подальше використання матеріалів дослідження планується у вивченні технічної можливості підключення розробленого модулю до бібліотеки аналізу аномалій системи Splunk Machine Learning Toolkit.

Результати досліджень було апробовано на:

XIX Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Теоретичні та прикладні проблеми фізики, математики та інформатики» КПІ ім. Ігоря Сікорського, 13-14 травня 2021 року.

Публікація за результатами досліджень:

В.Р. Лихошерст, М. В. Грайворонський Використання стримінгових алгоритмів кластеризації для виявлення аномалій мережевого трафіку. Матеріали XIX Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Теоретичні та прикладні проблеми фізики, математики та інформатики» КПІ ім. Ігоря Сікорського, 2021. С. 354-357

Ключові слова: кластеризація, комп’ютерні атаки, система виявлення вторгнень, DenStream, DBSCAN.

ABSTRACT

The volume of work 80 pages, 15 illustrations, 4 tables, 1 appendix, 11 sources of literature.

The object of the study was a computer network and information flows describing its state.

The subject of research is the methods of streaming data clustering.

The purpose of this work is to develop software to detect abnormal behavior of network traffic, which is able to take into account the streaming (streaming) nature of data and based on the ideas of cluster analysis.

Further use of research materials is planned in studying the technical possibility of connecting the developed module to the library of anomalies analysis of the system Splunk Machine Learning Toolkit

The research results were tested on:

XIX All-Ukrainian scientific-practical conference of students, graduate students and young scientists "Theoretical and applied problems of physics, mathematics and computer science" KPI. Igor Sikorsky, May 13-14, 2021.

Publication based on research results:

V.R. Lykhosherst, M. V.Graivoronsky Use of streaming clustering algorithms for detection of network traffic anomalies. Proceedings of the XIX All-Ukrainian scientific-practical conference of students, graduate students and young scientists "Theoretical and applied problems of physics, mathematics and computer science" KPI. Igor Sikorsky, 2021. p. 354-357

Keywords: clusterization, computer attacks, invasion detection system, DenStream, DBSCAN.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система;

КМ – комп’ютерна мережа;

СВВ – система виявлення вторгнень;

IDS – intrusion detection system;

МСВВ – мережева система виявлення вторгнень;

NIDS – network intrusion detection system;

HAC – hierarchical agglomerative clustering;

BIRCH – balanced iterative reducing and clustering using hierarchies;

ODAC – online divisive agglomerative clustering;

DBSCAN – Density Based Spatial Clustering Of Applications with Noise.

ВСТУП

Актуальність роботи. Використання комп'ютерів в всіх сферах сучасного життя, швидкий розвиток мережевих технологій, крім безперечних переваг, спричинили появу ряду специфічних проблем. Однією з них являється необхідність забезпечення ефективного захисту інформації, обумовлена зростанням правопорушень, пов'язаних з викраденням і несанкціонованим доступом до даних, які зберігаються в пам'яті комп'ютерів і які передаються по лініях зв'язку.

Комп'ютерні злочини відбуваються кожного дня у всіх країнах світу і поширенні в багатьох областях людської діяльності. Вони характеризуються високою скритністю, складність збору доказів по встановлених фактах здійснення злочину і складністю швидкого виявлення скоєних атак. Розмір збитків, які нанесені користувачам Всесвітньої павутини в результаті хакерських нападів, збільшується з кожним роком. Нажаль, навіть настільки загрозлива статистика не заважає великій кількості компаній і користувачам персональних комп'ютерів ігнорувати правила комп'ютерної безпеки.

Зараз в Інтернеті доступна велика кількість ресурсів, які допомагають зловмисникам проникати в комп'ютерні мережі. Детальна інформація про вразливі місця програм публічно обговорюється в групах новин, форумах та чатах. Існують легкодоступні інструкції по організації атак, в яких описується, як правильно написати програму для проникнення в комп'ютерну мережу, використовуючи інформацію про вразливі місця в програмах. І тисячі таких програмних додатків, які дозволяють організувати атаку, вже написані. Ці програми по організації комп'ютерних атак доступні для отримання будь-яким користувачем Інтернет. Але мало того, що ці програми легкодоступні, тепер ці програми стало легше використовувати. Декілька років тому треба було мати Unix-систему, щоб організувати атаку і треба було мати скомпільований програмний код цієї атаки. Сьогодні ж ці програми мають дружелюбний графічний інтерфейс і можуть працювати під управлянням Windows. Існують

спеціальні скрипти для організації автоматизованих атак, які дозволяють легко організувати дуже небезпечні атаки. Тому системним адміністраторам важливо розуміти небезпечність цих атак і вміти захищати свої мережі від них.

Мета і завдання дослідження полягає в тому, що в епоху швидкого росту кількості цифрової інформації, поширення доступу до Інтернету і користувальницьких сервісів, які залежать від стабільності мережевої інфраструктури, проблема виявлення аномальної поведінки мережевого трафіку стає критично важливою.

Об'єктом дослідження кваліфікаційної роботи являється комп'ютерна мережа та потоки інформації, що описують її стан.

Предмет дослідження – особливості поведінки КМ в ситуаціях здійснення загроз інформаційній безпеці.

Поширені сьогодні методи кластеризації в багатьох аспектах не враховують особливості мережевого трафіку, як об'єкту аналізу та потребують відповідної адаптації.

Метою роботи є програмна реалізація методу виявлення аномальної поведінки мережевого трафіку, що здатен врахувати потоковий (стрімінговий) характер надходження даних та оснований на ідеях кластерного аналізу.

Згідно з метою дослідження розв'язувались такі завдання:

- вивчити властивості мережевого трафіку, що впливають на ефективність використання методів кластеризації при пошуку аномалій в них;

- дослідити методи виявлення аномальної поведінки мережевого трафіку та адаптація їх до умов використання при потоковому надходженні даних;

- запропонувати метод виявлення аномалій;

- виконати його програмну реалізацію та перевірити ефективність при вирішені заданого класу задач.

Методи дослідження. У роботі використано комплекс дослідницьких методів:

загальнонауковий – при обґрунтуванні основних теоретичних положень та узагальнені отриманих результатів дослідження;

логічний – при визначені й характеристиці основних понять дослідження;

системний – при практичній реалізації дослідження.

Наукова новизна одержаних результатів полягає в порівнянні методів, на основі якого був обраний найкращий метод для виявлення аномального трафіку, що враховує стрімінговий характер надходження даних.

Практичне значення одержаних результатів. Проаналізувавши результати кластеризації обраного потокового алгоритму кластеризації даних DenStream та дані метрик, що використовуються для оцінювання якості кластеризації можна зробити висновок, що даний алгоритм успішно виділяє аномальну поведінку в роботі комп'ютерної мережі. Також він спроможний вдало виявляти аномальну поведінку для різних видів атак. Даний алгоритм кластеризації доречно використовувати в системах виявлення аномалій. Оскільки він дозволить з високою точністю адміністратору комп'ютерної мережі швидко виявити моменти, коли в мережі була присутня аномальна поведінка, тобто вона була під впливом мережевої атаки. Адміністратор в подальшому може самостійно проаналізувати на які показники мережі дана атака вплинула та здійснити відповідні дії для захисту мережі від даного типу атаки.

Апробація результатів роботи. XIX Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Теоретичні та прикладні проблеми фізики, математики та інформатики» КПІ ім. Ігоря Сікорського, 13-14 травня 2021 року.

Публікація.

В.Р. Лихошерст, М. В. Грайворонський, Використання стримінгових алгоритмів кластеризації для виявлення аномалій мережевого трафіку. Матеріали XIX Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Теоретичні та прикладні проблеми фізики, математики та інформатики» КПІ ім. Ігоря Сікорського, 2021. С. 354-357

1 АНАЛІЗ СИСТЕМ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

1.1 Аномалії, типи аномалій, аномалії мережевого трафіку

В умовах постійного росту великих об’ємів даних, а також зростаючої значущості результатів їх аналізу, питання ідентифікації існуючих в них аномалій стоїть особливо гостро. Результати аналізу без попереднього виключення аномальних екземплярів даних можуть бути значно спотворені. Виявлення аномалій відноситься до пошуку непередбачуваних значень (патернів) в потоках даних. Аномалія (викид, помилка, відхилення або виключення) – це відхилення поведінки системи від стандартного (очікуваного). Вони можуть виникати в даних різної природи і структури в результаті технічних збоїв, аварій, навмисних зломів і т.д. В даний час створено багато методів і алгоритмів пошуку аномалій для різних типів даних.

Аномалії в даних можуть бути віднесені до одного з трьох основних типів:

Точкові аномалії виникають в ситуації, коли окремий екземпляр даних може розглядатися як аномальний по відношенню до інших екземплярів даних.

Рисунок 1.1 – Приклад точкових аномалій

На рисунку 1.1 екземпляр A1, а також група екземплярів A2 являються аномальними при нормальних екземплярах в групах C1 та C2. Даний вид аномалій являється найбільш легко розпізнаваним, більшість існуючих методів створено для розпізнавання точкових аномалій.

Контекстуальні аномалії спостерігаються, якщо екземпляр даних являється аномальним лише в певному контексті (даний вид аномалій також називають

умовним). Для визначення аномалій цього типу основним являється виділення контекстуальних і поведінкових атрибутів.

Контекстуальні атрибути використовуються для визначення контексту (або оточення) для кожного екземпляру. В часових рядах контекстуальним атрибутом являється час, який визначає положення екземпляра в цілій послідовності. Контекстуальним атрибутом також може бути положення в просторі або більш складні комбінації властивостей.

Поведінкові атрибути визначають не контекстуальні характеристики, які відносяться до конкретного екземпляру даних.

Аномальна поведінка визначається за допомогою значень поведінкових атрибутів виходячи з конкретного контексту. Таким чином, екземпляр даних може бути контекстуальною аномалією при даних умовах, але при таких же поведінкових атрибутах рахується нормальним в інших контекстах. Так на рисунку 1.2 в точці А спостерігається аномалія, на відміну від точок N1-N5, які мають аналогічні значення. При виявленні контекстуальних аномалій ця властивість являється ключовою в розділенні контекстуальних та поведінкових атрибутів.

Рисунок 1.2 – Приклад контекстуальних аномалій

Колективні аномалії виникають тоді, коли послідовність зв’язаних екземплярів даних (наприклад, частина часового ряду) являється аномальною по відношенню до цілого набору даних. Окремий екземпляр даних в такій послідовності може не бути відхиленням, проте спільна поява таких екземплярів є колективною аномалією. На рисунку 1.3 частина А являється колективною аномалією.

Рисунок 1.3 – Приклад колективних аномалій

Крім того, в той час як точкові або контекстуальні аномалії можуть спостерігатися в будь якому наборі даних, колективні спостерігаються тільки в тих, де дані пов’язані між собою.

Варто відмітити, що точкові та колективні аномалії можуть бути в той же час і контекстуальними.

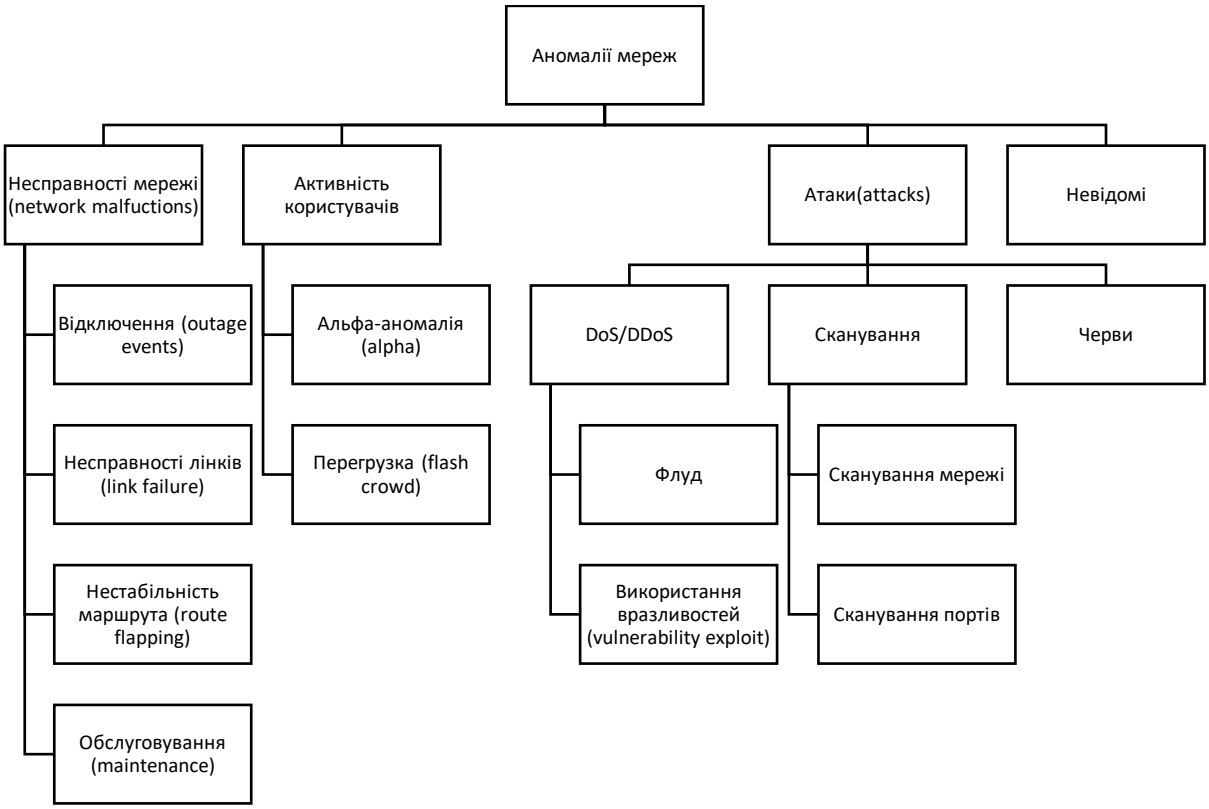


Рисунок 1.4 – Класифікація аномалій мереж

На рисунку 1.4 відображена класифікація аномалій мереж за причинами їх виникнення. До основних причин відносять несправності мережі, активність користувачів, атаки на мережу та невідомі дії, які негативно впливають на мережу.

1.2 Класифікація комп’ютерних атак

Коли йде мова про “комп’ютерну атаку”, ми розуміємо це як запуск зловмисником програми для отримання несанкціонованого доступу до комп’ютера. Форми організації атак дуже різноманітні, але в цілому вони належать до однієї з наступних категорій:

– віддалене проникнення в комп’ютер: програми, які отримують неавторизований доступ до другого комп’ютера через інтернет;

– локальне проникнення в комп’ютер: програми, які отримують неавторизований доступ до комп’ютера, на якому вони працюють;

– віддалене блокування комп’ютера: програми, які через інтернет блокують роботу всього віддаленого комп’ютера або окремої програми на ній;

– локальне блокування комп’ютера: програми, які блокують роботу комп’ютера, на якому вони працюють;

– мережеві сканери: програми, які здійснюють збір інформації про мережу, щоб визначити, які з комп’ютерів або програм, що працюють в цій мережі, потенційно вразливі до атак;

– сканери вразливих місць програм: програми, провіряють великі групи комп’ютерів, вразливих до певного виду атак;

– вскривачі паролів: програми, які виявляють паролі, що легко вгадуються в зашифрованих файлах паролів;

– аналізатори мережі (sniffers): програми, що прослуховують мережевий трафік. Часто в них є можливість автоматичного виділення імен користувачів, паролів і номерів кредитних карток з трафіка.

Найбільш популярними видами атак є:

1) Sendmail – це наглядний доказ того, що в складних програмах рідко буває виправлено всі помилки, так як розробники постійно добавляють нові можливості, через які виникають нові вразливі місця в системі;

2) Smurf використовує мережу, в якій машини обробляють ping-пакети для переповнення жертви пакетами відповідей на ping. Цю атаку можна

представити як підсилювач, який дозволяє зловмиснику анонімно блокувати роботу комп'ютера жертви через отримання через мережу великої кількості пакетів;

3) Teardrop повністю блокує комп'ютер з Linux, використовуючи помилку в підпрограмах мережевих драйверів, які обробляють фрагментовані пакети;

4) IMAP дозволяє користувачам отримувати їхні електроні листи з почтового серверу. В програмному забезпеченні серверу IMAP, були помилки, які дозволяли віддаленому зловмиснику отримувати повний контроль над сервером;

5) Nmap це складний засіб для сканування мережі. Nmap може сканувати за допомогою декількох протоколів, які працюють в прихованому режимі і автоматично ідентифікувати віддалені операційні мережі;

6) Back Orifice це троянський кінь, який дозволяє користувачу віддалено управляти комп'ютером за допомогою зручного графічного інтерфейсу;

7) WinNuke повністю блокує роботу комп'ютера шляхом відправки йому особливого пакету "термінові дані" по протоколу TCP.

1.3 Захист від комп'ютерних атак

Захист мережі від комп'ютерних атак – це постійна і нетривіальна задача, але ряд простих засобів захисту дозволить зупинити більшість спроб проникнути в мережу. До основних засобів захисту належать:

– Оперативна установка виправлень для програм (Patching). Компанії часто випускають виправлення для програм, щоб ліквідувати негативні наслідки помилок в них. Якщо не зробити виправлення в програмі, зловмисник може скористатися цими помилками і проникнути в ваш комп'ютер.

– Виявлення вірусів та троянських коней. Гарні антивірусні програми – незамінний засіб для підвищення безпеки в будь-якій мережі. Вони спостерігають за роботою комп'ютерів і виявляють на них небезпечні

програми. Єдина проблема полягає в тому, що для максимальної ефективності вони повинні бути встановленні на всіх комп'ютерах і постійно оновлюватися.

– Міжмережеві екрани (firewalls) – вони контролюють мережевий трафік. Можуть блокувати передачу в мережу будь-який вид трафіку або виконувати ті або інші перевірки іншого виду трафіку. Правильно сконфігурований міжмережевий екран може зупинити більшість відомих комп'ютерних атак.

– Шифрування. Зловмисники часто проникають в мережу за допомогою прослуховування мережевого трафіка в найбільш важливих місцях і виділення з нього імен користувачів та паролів. Тому з'єднання з віддаленими машинами, які захищенні паролями має бути зашифрованим.

– Сканери вразливих місць. Це програми, які сканують мережу в пошуках комп'ютерів, вразливих до певних видів атак. Сканери мають велику базу даних вразливих місць, яку вони використовують при перевірці того чи іншого комп'ютера на наявність в нього вразливостей.

– Грамотне конфігурування комп'ютерів в відношенні безпеки. Комп'ютери з заново встановленими ОС часто вразливі до атак. Причина полягає в тому, що при початковій установці ОС зазвичай дозволяються всі мережі засоби і часто дозволяються небезпечним способом. Це дозволяє зловмиснику використовувати багато способів для організації атаки на машину. Всі непотрібні мережеві засоби мають бути відключенні.

– Системи виявлення атак оперативно виявляють комп'ютерні атаки. Вони можуть бути встановленні за міжмережевим екраном, щоб виявляти атаки зсередини мережі або перед ним, щоб виявляти атаки на сам міжмережевий екран.

– Засоби виявлення топології мережі і сканери портів. Ці програми дозволяють скласти повну картину того, як облаштована ваша мережа і які комп'ютери в ній працюють, а також виявляти всі сервіси, які працюють на цій машині. Зловмисники використовують ці засоби для виявлення вразливих комп'ютерів та програм на них. Системні адміністратори повинні

використовувати ці засоби для нагляду за тим, які програми і на яких комп’ютерах працюють в їхній мережі.

– Тестування міжмережевих екранів і WWW серверів на стійкість до спроб їх блокування. Атаки на блокування комп’ютера широко поширені в Інтернеті. Зловмисники постійно виводять зі строю сайти, перевантажують комп’ютери і переповнюють мережі безглуздими пакетами. Мережеві адміністратори, які піклуються про безпеку своєї мережі, можуть організувати атаки проти самих себе, щоб виявити який збиток буде їм нанесено.

Список перерахованих комп’ютерних атак не є вичерпним. Постійно з’являються все нові та нові загрози. В швидко змінній та відносно новому середовищі Інтернет неможливо забезпечити ідеальний захист. Але можна знизити вірогідність серйозних збитків або катастрофічних наслідків, застосувавши адекватні запобіжні заходи. Якщо організація правильно захищена, а співробітники та і взагалі користувачі дотримуються правил комп’ютерної безпеки, то завжди можливе відновлення втраченої інформації, а потенційні втрати зводяться до мінімуму.

1.4 Виявлення атак та зловживань у комп’ютерній мережі

Загальноприйнята класифікація систем виявлення атак за способами виявлення атак включає системи виявлення аномалій і системи виявлення зловживань.

Алгоритм виявлення аномалій в мережевому трафіку виглядає наступним чином. Дані, які використовуються в системі виявлення аномалій, являють собою мережевий трафік. Він містить в собі набір мережевих пакетів, які в загальному випадку фрагментовані на рівні IP. Дані які збираються будуть використані при формуванні шаблону для наступних аналізів. Шаблон нормальної поведінки являє собою набір характеристик нормальної діяльності об’єкта (системи чи користувача). Так, отримані дані можуть бути агреговані за

визначений часовий інтервал і нормалізовані з цілю виявлення атрибутів загального виду, які будуть необхідні при побудові поточного профілю активності. Отриманий набір ознак порівнюється з шаблоном нормальної поведінки, тобто набором характеристик нормальної діяльності користувача або системи. В випадку знаходження значної зміни в параметрах які порівнюються, отримується повідомлення про знаходження (виявлення) мережевої аномалії. В іншому випадку шаблон нормальної поведінки змінюються за зміни параметрів його налаштування з урахуванням поточного профілю мережевої активності.

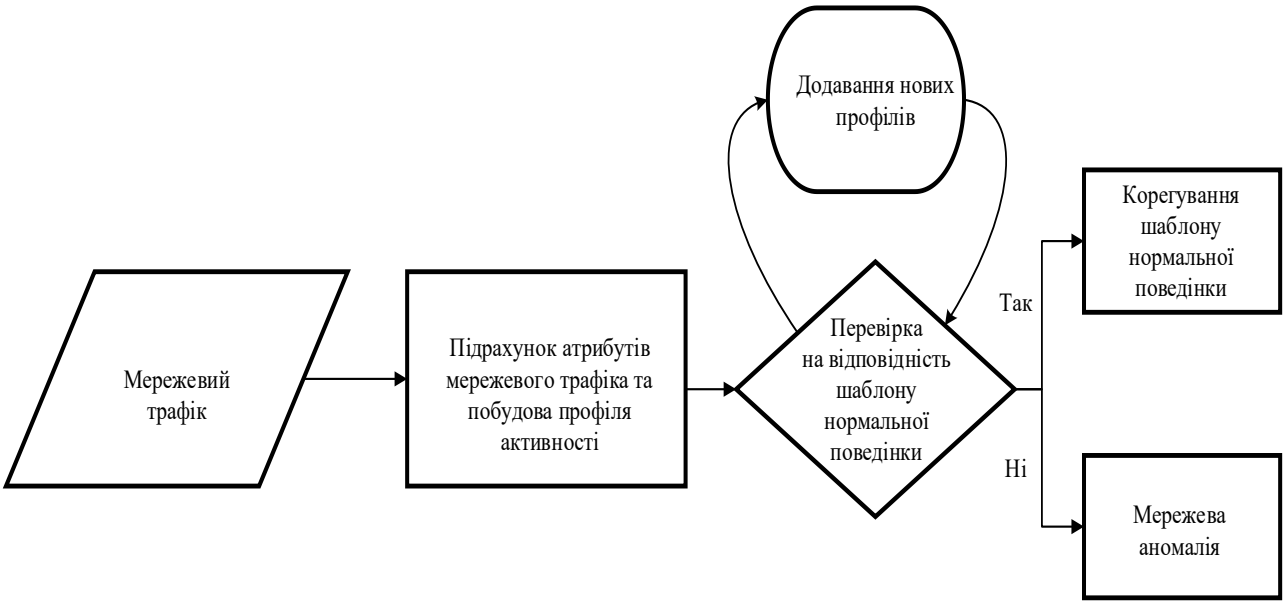


Рисунок 1.5 – Схема виявлення мережевих аномалій

Описаний вище алгоритм може включати декілька варіантів виконання для реалізації підсистеми перевірки на відповідність шаблону нормальної поведінки. Найпростішим з них є процедура порівняння з пороговою величиною, коли накопичені результати, які описують реальну мережеву активність, порівнюються з експертно-заданою числовою планкою. В цьому підході випадок перевищення значень параметрів, що розглядаються, вказаної границі являються ознакою мережевої аномалії.

Створення шаблону нормальної поведінки не є тривіальною задачею. На практиці можна побачити, що не кожна мережева аномалія являється атакою, проте кожну аномалію потрібно вміти виявляти. Наприклад, якщо мережевий

адміністратор використовує для налаштування мережі утиліти, такі як ping, traceroute, mtr, для діагностики мережевого оточення, вони не являються шкідливим програмним забезпеченням і не передбачають нелегальних намірів, але системи виявлення аномалій виявляють таку діяльність як аномальну мережеву активність.

На рисунку 1.6 показана схема виявлення зловживань в мережевому трафіку.

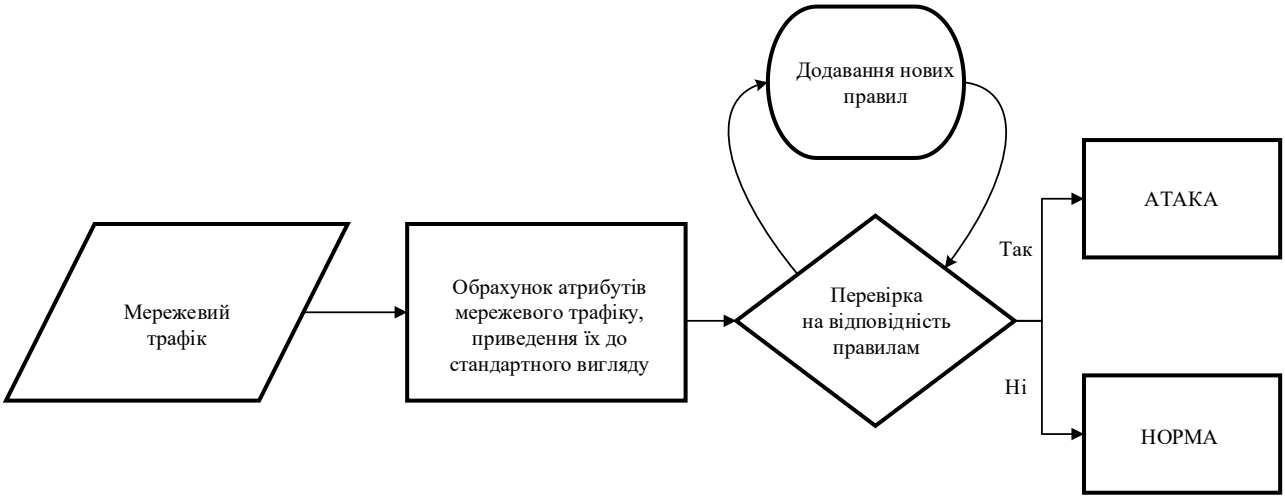


Рисунок 1.6 – Схема виявлення зловживань в мережевому трафіку

Система виявлення зловживань дає можливість мережевому адміністратору ідентифікувати несанкціоновані дії, якщо вони представлені в вигляді шаблону атак. Під шаблоном атак можна розуміти деяку сукупність дій, які явно описують конкретну атаку. Тут під шаблоном атаки розуміється деяка сукупність явно описуючих конкретну атаку дій, які використовуються до ознак і полів об’єкта, якого ідентифікують, і які дають можливість отримати однозначну відповідь про його відповідність до конкретної атаки. В системі виявлення вторгнень первинними даними для аналізу являється мережевий трафік. Виділені атрибути і поля мережевих пакетів передаються в модуль, який виконує пошук і перевірку на відповідність вхідних даних правилам і оповіщає про наявність загрози в випадку позитивного спрацювання одного з правил.

Головною проблемою, яку дуже важко вирішити при створенні системи виявлення зловживань є питання про створення ефективного механізму завдання правил. Зрозуміло, що створення вичерпної бази правил для виявлення різноманітних атак являється неможливим через декілька факторів. Один з цих факторів полягає в тому, що опис різних варіацій атакуючих дій негативно позначається на продуктивності системи. А оскільки навіть невелика зміна в атаці приводить до неможливості її визначення методами на основі зловживань, правила, які задаються, мають бути універсальними і покривати як можна більше число відомих модифікацій мережевих атак.

1.5 Класифікація методів виявлення атак

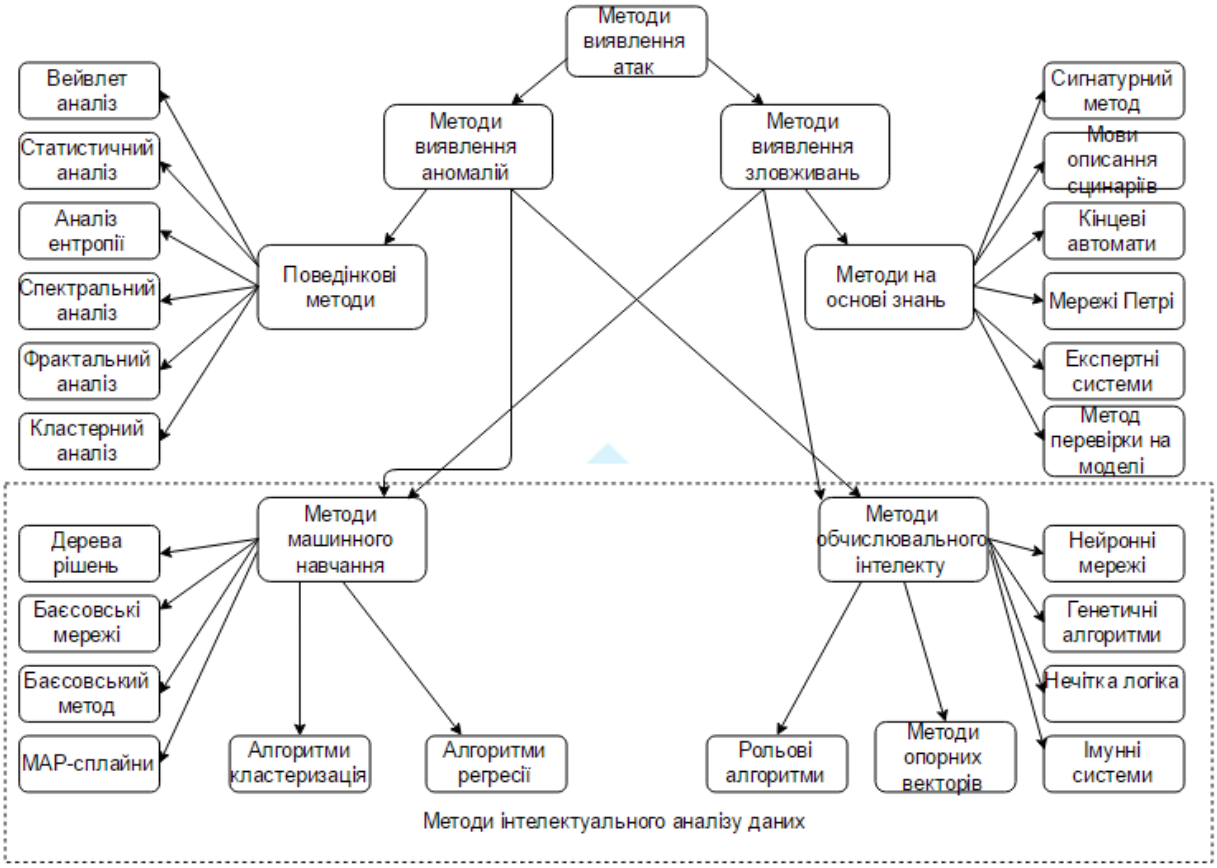


Рисунок 1.7 – Класифікація методів виявлення атак

Наведена вище схема класифікації методів виявлення атак являється умовною. Оскільки деякі з підходів можуть відноситись відразу до декількох груп. Наприклад, експертні системи і кінцеві автомати можуть

використовуватися для виявлення аномалій, але в більшості випадків вони використовуються саме для виявлення зловживань. Також такі методи обчислювального інтелекту, як нейронні мережі і метод опорних векторів, часто відносять до методів машинного навчання. В вищенаведеній схемі вибране таке розбиття, при якому методи інтелектуального аналізу даних класифікуються по критерію їх належності до біоподібних алгоритмів і тому включають в себе два класи: методи обчислювального інтелекту і методи машинного навчання.

– Поведінкові методи – це методи, які використовують інформацію про нормальний стан системи і порівнюють її з параметрами поведінки, за якою спостерігають. В процесі своєї роботи системи використовують наступний підхід, порівнюють поточні показники системи з шаблоном нормальної поведінки і в випадку виявлення значних відхилень система сигналізує про наявність атаки.

В більшості таких систем відбувається етап проведення попереднього налаштування (навчання системи), під якого створена система “набирається досвіду” для створення шаблону нормальної поведінки. Мінусом цього етапу є те, що налаштування відбувається протягом велику інтервалу часу, від декількох тижнів, а інколи до декількох місяців.

– Методи на основі знань. До методів на основі знань відносять такі методи, які в контексті заданих фактів, правил виводу і співставлення, які відображають ознаки заданих атак, проводять дії по виявленню атак на основі закладеного механізму пошуку. В якості процедури пошуку можуть використовуватися співставлення за прикладом, апарат регулярних виразів, аналіз переходу станів і т.д. Своєю назвою ці методи зобов’язані тим, що системи, основані на їх використані, працюють з базою знань, в яку включений опис вже відомих атак. Тут ця база знань представлена сховищем, що містить записи експертів з підтримкою логіки їх обробки і інтерпретації.

– Методи машинного навчання. До методів машинного навчання (machine learning) відносяться три широкі техніки виявлення атак:

1) Техніка виявлення аномалій без вчителя (Unsupervised anomaly detection techniques). Використовується при відсутності апріорної інформації про дані. Алгоритми розпізнавання в режимі без вчителя базуються на припущенні про те, що аномальні екземпляри зустрічаються набагато рідше нормальних. Данні обробляються, найбільш віддалені визначаються як аномалії. Для використання цієї методики повинен бути доступний весь набір даних, тобто ця методика повинна застосовуватися в режимі реального часу.

2) Техніка виявлення аномалій з вчителем (Supervised anomaly detection techniques). Дана методика вимагає наявності навчальної вибірки, яка повноцінно представляє систему і екземпляри даних нормального і аномального класу. Робота алгоритму відбувається в два етапи: навчання і розпізнавання. На першому етапі будується модель, з якої порівнюються екземпляри, які не мають мітки. В більшості випадків передбачається, що дані не міняють свої статистичні характеристики, інакше виникає необхідність змінювати класифікатор. Головною складністю алгоритмів, які працюють в режимі розпізнавання з вчителем, являється формування даних для навчання. Часто аномальний клас представлений значно меншим числом екземплярів, ніж нормальний, що може приводити до неточностей в отриманій моделі. В таких випадках використовується штучна генерація аномалій

3) Техніка розпізнавання частково з вчителем(semi-supervised anomaly detection techniques). Вихідні дані в цьому підході представляють тільки нормальний клас. Здійснивши навчання на одному класі, система може визначати належність нових даних до нього, таким чином визначаючи протилежний. Алгоритми, які працюють в режимі розпізнавання частково з вчителем, не потребують інформації про аномальний клас екземплярів, внаслідок чого вони широко застосовуються і дозволяють розпізнавати відхилення за відсутності заздалегідь певної інформації про них.

– Методи обчислюваного інтелекту. Штучна нейронна мережа являє собою сукупність елементів, що обробляються – нейронів, які зв’язані між собою синапсами і які перетворюють набір вхідних даних в набір бажаних

вихідних значень. Нейронні мережі використовуються в широкому спектрі додатків: розпізнавання образів, теорія управління, криптографія, стиснення даних. Нейроні мережі володіють здатністю навчання за прикладом і узагальнення з зашумлених і неповних даних. В процесі навчання відбувається налаштування коефіцієнтів, які асоціюються з синаптичними вагами.

Для навчання нейронних мереж існує декілька методів. Один з найбільш популярних алгоритмів є метод зворотного поширення помилки. Цей алгоритм представляє собою градієнтний спуск з мінімізацією середньоквадратичної помилки на кожній ітерації свого виконання.

1.6 Системи виявлення вторгнень

Важливе місце серед інструментів мережевого адміністратора для виявлення мережевих аномалій займають Системи виявлення вторгнення (Системи виявлення атак) (англ. IDS - Intrusion detection system). Вторгнення – це набір дій, націлених на компрометацію безпеки комп’ютера або мережі в області конфіденційності, цілісності і доступності. Вторгнення може бути як зовнішнє, так і внутрішнє. Для протидії вторгненню і для захисту мережевої інфраструктури підприємства СВВ надають механізм для збору та аналізу даних із мережі або з конкретного комп’ютера. На рисунку 1.8 представлена базова класифікація СВВ.

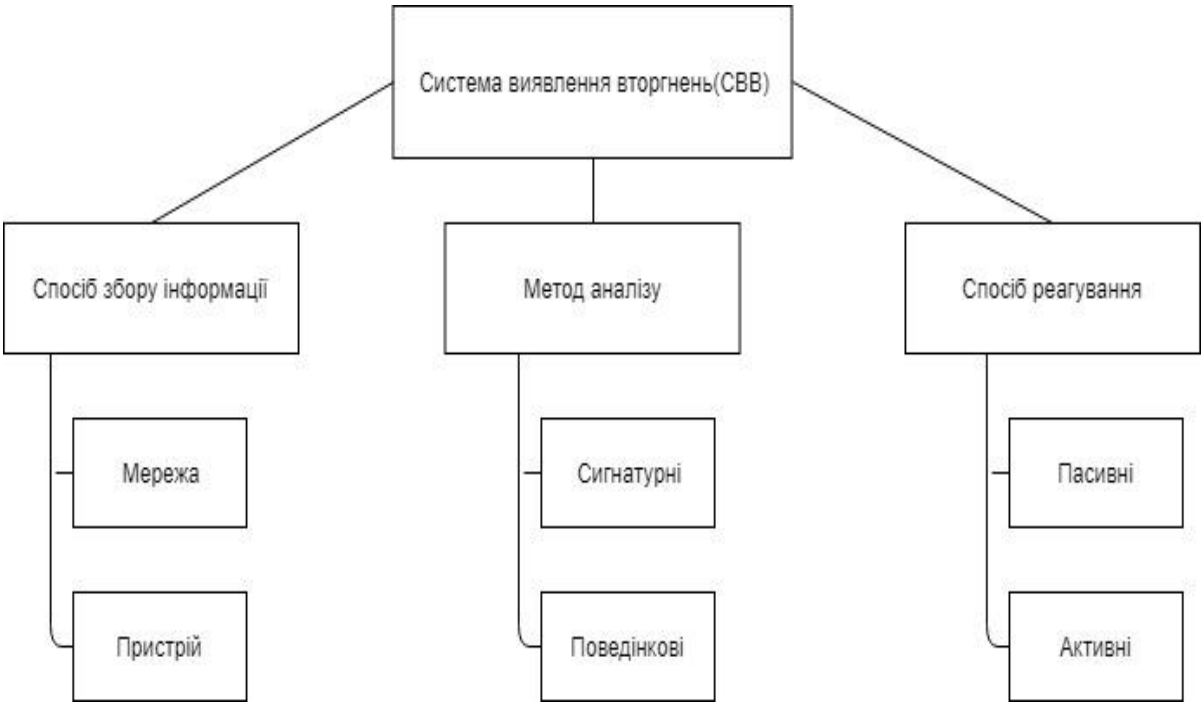


Рисунок 1.8 – Базова класифікація СВВ

Як видно, СВВ можуть класифікуватися по місцю базування в мережі, за принципом виявлення та за реакцією на вторгнення. У будь якої з цих характеристик є як і переваги, так і недоліки. Наприклад, у СВВ на основі сигнатур, в якості індикатора виявлення відхилення від норми в мережі, не можуть виявити відхилення, яких немає в їхній базі, навіть якщо вони не суттєво відрізняються від відомих. В той же час у поведінкових СВВ (anomaly-based IDS), які в якості індикатора використовують вивчену поведінку мережі прийняту за нормальну, можливі часті неправильні спрацювання, частіше чим у СВВ на основі сигнатур. В цілому можна сказати, що сигнатурні СВВ шукають в потоці трафіку відомі ознаки поведінки загроз, в той час як поведінкові СВВ пробують відстежувати нехарактерні ознаки потоку трафіку. Відмічається, що зараз дослідники більше сконцентровані в області поведінкових систем на основі аномалій, так як вони здатні виявляти як відомі, так і невідомі аномалії [1].

Ще однією важливою характеристикою являється місце, де встановлена СВВ. Вона може бути встановлена в правильному місці мережі і перехвачувати весь (в ідеальних умовах) вхідний та вихідний трафік з цілю аналізу і прийняття рішення про його легітимність. Такі СВВ називаються МСВВ (NIDS (Network

Intrusion Detection System)). Прикладами існуючих популярних NIDS являються Snort, Suricata, Bro NIDS і так далі. Також можна встановити СВВ на кінцевому пристрої. Такі системи називаються HIDS (Host Intrusion Detection Systems). Вони встановлюються на конкретні пристрої в мережі і слідкують за вхідними та вихідними пакетами тільки цього пристрою, повідомляючи системного адміністратора про зафіксовану підозрілу активність.

Для більш повного розуміння МСВВ нижче приведена архітектура типової МСВВ.

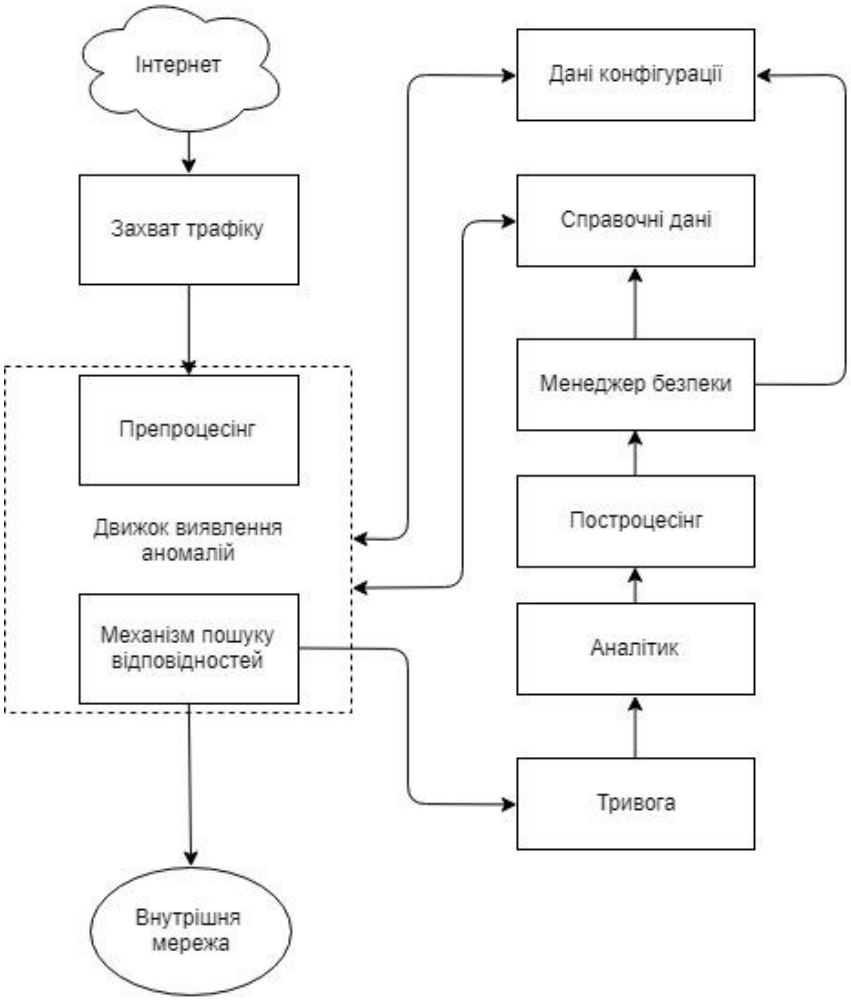


Рисунок 1.9 – Типова архітектура СВВ

На рисунку 1.9 представленні базові елементи архітектури МСВВ. Нижче представлений опис кожного компонента архітектури.

Движок виявлення аномалій це основний компонент МСВВ. В нього входять препроцесор і механізм пошуку відповідностей. Препроцесор виясняє, чи відома йому ця аномалія. Якщо відома, то вона буде виявлена за допомогою

сигнатурної техніки. Якщо аномалія невідома – дані будуть відправлені далі механізму пошуку відповідностей.

Механізм пошуку відповідностей шукає в мережевому трафіку відповідності з шаблоном або профілем, який може мати інформацію про загрози. Важливі вимоги до механізму пошуку відповідностей:

– механізм повинен володіти високою точністю;

– пошук відповідностей повинен бути швидким;

– зберігання елементів (шаблони, профілі) повинно бути організовано ефективно.

Довідникові дані зберігають інформацію про відомі вторгнення (сигнатури) або профілі нормальної поведінки. Можливим типом змісту являються профілі, сигнатури і правила. У випадку з поведінковими СВВ це, як правило, профілі. Профіль оновлюється, як тільки нові дані про поведінку стають відомими.

Дані конфігурації це проміжкові дані, наприклад, частково змінений профіль поведінки (після отримання нових даних про трафік). Враховуючи сучасні об’єми передачі даних через мережу, може вимагати достатньо багато простору.

Тривога. Цей компонент архітектури відповідає за генерацію оповіщення, покладаючись на дані, отримані від движка.

Аналітик відповідає за аналіз, інтерпретацію і прийняття необхідних дій до тривожної інформації, отриманої від движка. Аналітик також виконує деякі дії для полегшення роботи постпроцесинга.

Постпроцесинг – це важливий модуль МСВВ для пост-обробки згенерованих тривог для вияснення, являються вони дійсно аномаліями.

Захват трафіку також один з найбільш важливих модулів архітектури. “Сирий” трафік перехвачується на обох рівнях, пакетному та поточному. Пакети можуть бути перехоплені будь якою для цього утилітою, наприклад, Wireshark і потім оброблені препроцесингом перед відправкою до движка. Потік в високошвидкісних мережах складається з інформації, отриманої з

одного або більше пакета. Популярні інструменти для захвату потоку це, наприклад: Nfdump, NfSen і Cisco Netflow V.9.

Менеджер безпеки слідкує за актуальністю сигнатур або поведінкових профілів.

Системи виявлення вторгнень на даний момент крім позитивних моментів володіють рядом недоліків та обмежень. До таких обмежень належать:

1) Високі вимоги до апаратних ресурсів. Для того щоб в реальному часі обробляти великі об’єми трафіку необхідні серйозні обчислювальні потужності, в іншому випадку СВВ стане “шийкою пляшки”(вразливим слабким місцем) в мережі.

2) Для сигнатурних СВВ особливо важливо тримати базу даних сигнатур в актуальному стані через те, що загрози та атаки вдосконалюються кожного дня. З цього зрозуміло, що час, між появою загрози і отриманням свіжої версії бази даних сигнатур повинен бути як можна менший, але це не завжди можливо здійснити. В період цього часового “лагу” нова загроза не може бути виявлена.

3) Інформація в дампах трафіку вміщує адресу хостів, що дуже зручно і наглядно. Але, таку інформацію можна легко підробити ще до надходження її на мережевий адаптер МСВВ. З чого слідує, що визначення джерела проблеми не завжди можливо здійснити.

4) Обмежені можливості оновлення. Часто, достатньо складно оновити існуючі системи новими механізмами виявлення аномалій. Нова підсистема повинна вписуватися і почати взаємодію з іншими підсистемами, що деколи неможливо через особливість архітектури МСВВ.

5) Через природу мережевих СВВ вони можуть підлягати впливу різних мережевих загроз та аномалій, через те що їм потрібно аналізувати дані, які отримані з мережі.

6) Відсутня єдина методика тестування СВВ.

7) Обмежена портативність. По цей день більшість СВВ створюється для використання на конкретній апаратурі (наприклад, готові апаратні рішення від великих виробників).

Також є ряд недоліків, які відносяться до реалізованих методів виявлення аномалій. Серед них наступні:

– Висока частота неправильних спрацювань через мережевий шум обмежує ефективність МСВВ. Шум в мережі може виникати, наприклад, через помилки в програмному забезпеченні. Звідси виходить ситуація, коли дійсно аномальну ситуацію можна пропустити;

– Для сигнатурних СВВ характерні низькі можливості з виявлення нових атак (атак нульового дня);

– Велика кількість вторгнень неможливо визначити на начальних етапах;

– Часто неможливо визначити цілі атаки, які викликають аномалію;

– В високошвидкісних мережах буває дуже складно виявити вторгнення в реальному часі з необхідною повнотою.

Неможливо не згадати проблеми зі збором даних. Для точного виявлення вторгнення необхідна надійна і вичерпна інформація про об’єкт (в нашому випадку мережевий трафік). В сучасних комп’ютерних мережах з гігабітними швидкостями можуть бути необхідні сотні гігабайт дискового простору за добу.

До недоліків які зараз існують на ринку рішення в області виявлення аномалій в мережі можна віднести, наприклад те, що комерційні СВВ по перше достатньо дорогі, хоча і мають якісну підтримку зі сторони виробника, але для кінцевого користувача пропонується в якості “чорного ящика”, алгоритми роботи якого являються об’єктом комерційної таємниці. Не дивлячись на загальну схожість принципів дії, як правило “серце” системи виявлення – движок виявлення аномалій – представляють собою запатентовану технологію. Часто перевірити заявлену ефективність не є можливим, а підбір критеріїв для порівняння рішень від різних виробників представляє достатньо складну задачу. В свою чергу, безкоштовні СВВ на базі open-source рішень часто достатньо складні в налаштуванні і немає необхідної документації до неї. В якості недоліку можна вказати, наприклад те, що Стандартний набір правил Snort поширюється безкоштовно, розширений, який вміщує найбільш актуальні сигнатури відомих атак – за платною підпискою. Окремо необхідно вказати, що

запуск на підприємстві подібних систем – серйозна задача для персоналу і не завжди компанія володіє спеціалістами такого рівня, що тягне за собою підвищені витрати (навчання персоналу, пошук нових співробітників).

В якості рекомендацій при розробці методів або систем виявлення аномальних властивостей можна згадати:

– гібридизація існуючих методів, щоб забезпечити відсутність недоліків кожного з них;

– підвищення продуктивності алгоритмів для покращення показників балансу якості виявлення/ швидкість спрацювань;

– робота над точністю алгоритмів для зменшення числа неправильних спрацювань.

Враховуючи все вищесказане можна сказати, що подальше вдосконалення існуючих і розробка нових методів виявлення підозрілої мережевої активності являється актуальною і перспективною задачею. Для цього необхідно розуміти природу мережевого трафіку, особливості роботи основних протоколів, причини виявлення аномалій в мережі.

Висновок до розділу 1

Аналіз мережевого трафіку є дуже важливою та актуальною задачею. Оскільки стан і працездатність комп’ютерної мережі має величезний вплив на роботу підприємств, компаній. Для забезпечення працездатності мережі на даний час використовується велика кількість програмних і апаратних засобів, які здатні виявити і здатні запобігти виведенню з працездатного стану або несанкціонованого доступу до ресурсів в мережі. Головним програмним засобом, який здатний виконати вищевказані функції являється система виявлення вторгнень. За допомогою методів моделювання множини станів, описової статистики, моделювання правил та нейронних мереж ці системи здатні виявляти як нові, так і вже відомі аномалії комп’ютерної мережі.

2 DATA STREAM MINING, СТРИМІНГОВІ АЛГОРИТМИ КЛАСТЕРИЗАЦІЇ ДЛЯ ВИРІШЕННЯ ЗАДАЧ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

2.1 Data Mining

В результаті розвитку інформаційних технологій, кількість даних, накопичених людством в електронному вигляді, збільшується швидкими темпами. Ці дані існують навколо нас в різних видах: тексти, зображення, аудіо, відео, гіпертекстові документи, реляційні бази даних і т.д. Величезна кількість даних з’явилась в результаті повсюдного використання мережі Інтернет, яка значно полегшила доступ до інформації з географічно віддалених точок Землі. Однак більша частина доступної інформації не несе для конкретної людини ніякої користі. Людина не в стані обробити таку кількість даних. Виникає проблема виділення корисної для користувача інформації з великого об’єму “сирих” даних. В результаті великі бази даних стали “могилами” даних – архівами, які рідко використовуються. Як наслідок, важливі рішення приймаються не на основі інформаційно-насичених баз даних, а на основі інтуїції людини, яка приймає рішення, так як вона немає належних інструментів для видобутку корисних знань з великих об’ємів даних. Технологія Інтелектуального Аналізу Даних дозволяє виділити корисні знання, важливі патерни, сприяючи вдосконаленню бізнес-стратегії, баз знань, наукових та медичних досліджень.

Інтелектуальний аналіз даних – це процес визначення нових, коректних і потенційно корисних знань на основі великих масивів даних. Виділення знань в результаті інтелектуального аналізу даних називається патерном. Патерном може бути, наприклад, деяке нетривіальне твердження про структуру даних, про закономірності які існують, про залежності між атрибутами і т.д.

Таким чином, задачею Інтелектуального аналізу даних являється ефективне виділення осмислених патернів з існуючого масиву даних великого розміру. Для відсіву великої кількості можливих малокорисних патернів може

вводитися функція корисності. В реальності оцінка корисності знань має суб’єктивний характер, тобто залежить від конкретного користувача. Можна виділити дві головні характеристики “цікавих” знань:

Несподіванка. Знання несподіване для користувача і потенційно несе нову інформацію.

Застосовність. Користувач може використовувати нові знання для отримання своїх цілей.

Цікаві знання, закономірності, високорівнева інформація, отримані в результаті аналізу даних, можуть бути використані для прийняття рішення, контролю за процесами, управління інформацією і обробкою запитів. Тому технологія Data Mining розглядається як одна з найбільш важливих і багатообіцяюча для досліджень і застосування в галузі інформаційних технологій.

Традиційно виділяють наступні етапи в процесі інтелектуального аналізу даних:

1) Вивчення предметної області, в результаті чого формуються основні цілі аналізу.

2) Збір даних.

3) Попередня обробка даних:

a. Очистка даних – виключення протиріч і випадкових “шумів” з вихідних даних

b. Інтеграція даних – об’єднання даних з декількох можливих джерел в одному сховищі

c. Перетворення даних – на даному етапі дані перетворюються до форми, яка підходить до аналізу. Часто використовується агрегація даних і скорочення розмірності.

4) Аналіз даних. В рамках даного етапу використовується інтелектуальний аналіз з ціллю виділення патернів.

5) Інтерпретація найдених патернів. Даний етап може включати візуалізацію виділених патернів, визначення дійсно корисних патернів на основі деякої функції корисності.

6) Використання нових знань.

Зазвичай в системах інтелектуального аналізу даних виділяються наступні головні компоненти:

1) База даних, сховище даних або інший репозиторій інформації. Це може бути одна або декілька баз даних, сховищ даних, електроні таблиці, інші види репозиторій, над якими можна виконати очистку та інтеграцію. Види баз даних:

– реляційні бази даних;
– сховища даних;
– об’єктно-орієнтовні бази даних;
– просторові бази даних (Spatial databases);
– часові бази даних (Temporal databases);
– текстові бази даних;
– мультимедійні бази даних;
– різнорідні бази даних;
– World Wide Web.

2) Сервер бази даних або сховище даних. Даний сервер відповідає за виділення існуючих даних на основі користувальницького запиту.

3) База знань. Це знання про предметну область, які вказують, як проводити пошук та оцінювати корисність патернів.

4) Служба видобутку знань. Являється невід’ємною частиною системи інтелектуального аналізу даних і включає в себе набір функціональних модулів для таких задач, як характеризація, пошук асоціацій, класифікація, кластерний аналіз і аналіз відхилень.

5) Модуль оцінки патернів. Даний компонент обчислює заходи інтересу або корисності патернів.

6) Графічний користувальницький інтерфейс. Цей модуль відповідає за комунікацію між користувачем та системою інтелектуального аналізу даних, візуалізацію патернів в різні форми.

В технологію Data Mining покладена концепція шаблонів, які представляють собою закономірності. В результаті виявлення цих прихованих від людського ока закономірностей вирішуються задачі Data Mining. Задачам Інтелектуального аналізу даних відповідають різні типи закономірностей, які можуть бути виражені в формі, зрозумілій людині. Задачі Data Mining інколи називають закономірностями або техніками. Зазвичай виділяють наступні: класифікація, кластеризація, прогнозування, асоціація, візуалізація, аналіз та виявлення відхилень, оцінювання, аналіз зв'язків, підведення підсумків.

Класифікація (classification) найбільш проста і поширена задача Інтелектуального аналізу даних. В результаті вирішення задачі класифікації виявляються ознаки, які характеризують групи об'єктів набору даних, який досліджується – класи, за цими ознаками новий об'єкт можна віднести до того чи іншого класу. Для рішення задач класифікації можуть використовуватися наступні методи: найближчого сусіда (Nearest Neighbor), k-найближчого сусіда (k-Nearest Neighbor), байесовські мережі (Bayesian Networks), індукція дерев рішень, нейронні мережі (neural networks).

Кластеризація (clustering analysis) являється логічним продовженням ідеї класифікації. Ця задача більш складна, особливість кластеризації заключається в тому, що класи являються розбиттям об'єктів на групи. Приклад методу рішення задачі кластеризації: навчання без вчителя особливого виду нейронних мереж – самоорганізованих карт Кохонена.

Асоціація(Associations). В ході рішення задач пошуку асоціативних правил відшукуються закономірності між пов'язаними подіями в наборі даних. Відмінність асоціації від двох попередніх задач Data Mining пошук закономірностей здійснюється не на основі властивостей об'єкту, що досліджується, а між декількома подіями, які відбуваються одночасно.

Найбільш відомий алгоритм рішення задачі пошуку асоціативних правил – алгоритм Apriori.

Послідовність (Sequence), або послідовна асоціація (sequential association) дозволяє знайти часові закономірності між транзакціями. Задача послідовності подібна асоціації, але її цілю являється встановлення закономірностей не між одночасно наступаючими подіями, а між подіями, які пов'язані в часі, тобто які відбуваються з деяким визначеним інтервалом часу. Іншими словами, послідовність визначається високою ймовірністю ланцюжка пов'язаних в часі подій. Фактично, асоціація являється окремим випадком послідовності з часовим лагом, рівним нулю. Цю задачу Data Mining також називають задачею знаходження послідовних шаблонів (sequential pattern). Правило послідовності: після події X через деякий час відбудеться подія Y.

Прогнозування (Forecasting). В результаті рішення задачі прогнозування на основі особливостей історичних даних оцінюється припущення або ж майбутнє значення цільових числових показників. Для рішення таких задач широко використовуються методи математичної статистики, нейронні мережі.

Виявлення відхилень або викидів. (Deviation Detection), аналіз відхилень або викидів. Ціль рішення даної задачі – виявлення та аналіз даних, які найбільше відрізняються від загальної множини даних, виявлення нехарактерних шаблонів.

Оцінювання (Estimation) – ця задача зводиться до передбачення неперервних значень ознак.

Аналіз зв'язків (Link Analysis) – задача знаходження залежностей в наборі даних.

Візуалізація (Visualization). В результаті візуалізації створюється графічний образ даних, що аналізуються. Для рішення задач візуалізації використовуються графічні методи, які показують наявність залежностей в даних. Приклад методів візуалізації – представлення даних в 2-D та 3-D вимірах.

Підведення підсумків. (Summarization) – задача, ціль якої – опис конкретних груп об’єктів з набору даних, що аналізуються.

Інтелектуальний аналіз даних володіє рядом недоліків. До основних належать:

– Data Mining не зможе замінити в повній мірі аналітика;

– складність підготовки та пошуку даних для аналізу методів Data Mining;

– великий процент помилкових, недостовірний або безглуздих отриманих результатів;

– висока ціна отриманих рішень.

2.2 Кластеризація даних

Кластеризація (або кластерний аналіз) – це задача розбиття множини об’єктів на групи (кластери). В середині кожної групи повинні бути “схожі” об’єкти, а об’єкти різних груп повинні як можна більше відрізнятися. Головна різниця кластеризації від класифікації в тому, що в задачах кластеризації перелік груп заздалегідь не вказаний і визначається саме в процесі роботи алгоритму.

Використання кластерного аналізу в загальному вигляді зводиться до наступних етапів:

1. Відбір вибірки об’єктів для кластеризації.
2. Визначення множини змінних, за якими будуть оцінюватися об’єкти в вибірці. При необхідності – нормалізація значень змінних.
3. Підрахунок значень міри схожості між об’єктами.
4. Використання методу кластерного аналізу для створення схожих об’єктів (кластерів).
5. Представлення та інтерпретація результатів аналізу.

Після отримання і аналізу результатів можливий вибір необхідної метрики та методу кластеризації для отримання оптимального результату.

Як же визначити схожість об’єктів? Для початку необхідно створити вектор характеристик для кожного об’єкту – як правило, це набір числових даних, зокрема таких, що отримані методами статистичного аналізу. Також існують алгоритми, які працюють з якісними характеристиками.

Після того, як визначено вектор характеристик, можна провести нормалізацію, щоб всі компоненти давали однаковий внесок при розрахунку відстаней. В процесі нормалізації всі значення приводяться до деякого діапазону, наприклад [-1,1] або [0,1].

Для кожної пари об’єктів визначається “відстань” між ними – ступінь схожості.

Основні метрики для визначення ступеня схожості:

1. Евклідова відстань. Найпопулярніша функція відстані. Представляє собою геометричну відстань в багатовимірному просторі

$$p(x, x') = \sqrt{\sum_{i}^{n} (x_i - x'_i)^2}$$

2. Квадрат евклідової відстані. Використовується для додання більшої ваги більш віддаленим один від одного об’єктам.

$$p(x, x') = \sum_{i}^{n} (x_i - x'_i)^2$$

3. Відстань «міських кварталів» (манхетенська відстань). Ця відстань являється середня різниця за координатами. В більшості випадків ця міра відстані приводить до результатів, які подібні до результатів отриманих при використанні відстані Евкліда. Але для цієї міри вплив окремих великих викидів зменшується.

$$p(x, x') = \sum_{i}^{n} |x_i - x'_i|$$

4. Відстань Чебишева. Ця відстань може бути корисною, коли необхідно визначити два об’єкта як “різні”, якщо вони відрізняються по якійсь з однієї координат.

$$p(x, x') = \max(|x_i - x'_i|)$$

Вибір метрики повністю залежить від користувача, оскільки результати кластеризації можуть суттєво відрізнятися при використанні різних мір.

2.3 Інтелектуальний аналіз потокових даних (Data Stream Mining)

Data stream mining це область досліджень, що представляє великий інтерес в останні роки серед дослідників. Основною задачею інтелектуального аналізу потокових даних є витягування (знаходження) цінних знань в реальному часі з масового, безперервного, динамічного потоку даних методом одноразового сканування. Традиційна кластеризація не дає можливості провести кластеризацію потокових даних. Саме тому і потрібні спеціально адаптовані для вказаної умови алгоритми потокової кластеризації. Кластеризація потокових даних використовується у різних сферах, таких як фінансові операції, телефоні записи, моніторинг мереж, телекомунікації, аналіз веб-сайтів, моніторинг погоди і веб-бізнесу.

Під час використання кластеризації потоків даних виникають деякі проблеми:

– під час виконання процесу одноразового сканування, необхідно використовувати короткий часовий інтервал з обмеженою пам’яттю;

– алгоритми кластеризації мають вміти виявляти в потокових даних шум та викиди;

– алгоритми мають формувати кластери довільної форми.

Термін “потік даних” стосується великої, потенційно нескінченної, безперервної послідовності інформації, що поступає з великою швидкістю. Типові приклади потокових даних включають в себе технічні дані, наукові дані, дані часових рядів та дані, створені в інших динамічних областях, такі як

телефонні записи, моніторинг мереж, телекомунікації, аналіз веб-сайтів, моніторинг погоди, кредитні картки та електронний бізнес.

На відміну від традиційних форм даних, які є незмінними та статичними, потік даних має свої унікальні характеристики:

– він складається з безперервного потоку даних;

– це дані, що швидко надходять в режимі реального часу;

– багаторазова обробка потокових даних практично неможлива, тому необхідно використовувати алгоритм який здійснює обробку даних лише за один прохід;

– збереження потокових даних обмежене, тому лише синопсис даних може бути збережений;

– потокові дані багатовимірні, тому для отримання цінних результатів необхідні складні алгоритми.

Таблиця 2.1 – Порівняння обробки потокових та традиційних даних

Обробка потокових даних	**Обробка традиційних даних**
Обробка виконується в режимі реального часу	Обробка даних виконується в пакетному режимі доступу.
Зберігання даних неможливо	Можливо зберігати дані
Наближені результати являються прийнятні	Необхідно досягнути якомога точніших результатів
Обробка синопсису даних – це звичайне завдання	Обробка кожного елемента / запису даних – це звичайне завдання
Зберігання лише агрегованих та підсумкових даних	Зберігання вхідних та результуючих даних
Важливі просторові та – особливо часові контексти даних	Просторові та часові контексти розглядаються для певних класів додатків
Лінійні та сублінійні обчислювальні методи широко використовуються	При необхідності використовуються методики з високою просторово-часовою складністю

Процес інтелектуального аналізу потокових даних виконує виділення цінних патернів в режимі реального часу з динамічних потокових даних лише в одноразовому скануванні, що робить це виділення дуже важкою задачею. Проте процес кластеризації потокових даних являється об’єктом великої уваги дослідників в галузі Data Mining завдяки своїй ефективності в інтелектуальному аналізі даних. Метою цього процесу є класифікація подібних об’єктів у одному кластері, тоді як об’єкти які не схожі розмістити в різні кластери.

По суті, алгоритми кластеризації, які використовуються для обробки даних, є одним з основних методів, які можуть бути застосовані для виявлення патернів, розпізнавання шаблонів. Потоковий доступ виконується краще, ніж доступ до великих об’ємів даних, що зберігаються на жорстких дисках чи у вигляді потокових даних. Отже, потокові алгоритми необхідні для кластеризації (групування) таких даних. Однак, через природу потокових даних, а саме те, що вони масивні та динамічні, традиційні методи кластеризації не можуть бути застосовані. Таким чином, стало важливо розробляти нові та вдосконалювати існуючі технології кластерного аналізу.

Процес обробки потокових даних за допомогою створення кластерів цих даних є проблемним через ряд різноманітних факторів:

– одноразова кластеризація: кластеризація потокових даних виконується лише за допомогою одноразового сканування;

– обмежений час: потокові дані повинні оброблятися та формуватися в кластери в режимі реального часу та за обмеженого часового періоду;

– обмежена пам’ять: алгоритм кластеризації працює лише з обмеженою кількістю пам’яті, проте він повинен обробляти безперервний вхідний, нескінченний потік даних;

– невідома кількість і форма кластерів: ці аспекти потоку даних залишаються невідомими до обробки потокових даних;

– дані, які розвиваються: алгоритм повинен бути розроблений таким чином, щоб бути готовими до обробки постійно мінливих параметрів потокових даних;

– шум: шум у даних впливає на результати кластеризації, тому алгоритм кластеризації повинен виявляти шум, який існує в потоці даних.

2.4 Стримінгові методи кластеризації

Одним з методів кластеризації являється ієрархічний, який можна розділити на два основних типи, а саме: агломераційний та розбіжний. Перший тип об’єднує набір ’n’ об’єктів у більш загальні категорії, а останній тип розділяє ’n’ об’єкти на більш дрібні кластери послідовно.

Hierarchical agglomerative clustering (HAC) [2] – являється часто використовуваним методом з можливістю вручну визначити кількість кластерів. Алгоритми CURE [3] та ROCK [4] є прикладами алгоритму HAC, який використовує статичну модель при виборі схожих кластерів, які мають бути інтегровані. Тим не менш, один недолік таких алгоритмів полягає в тому, що коли точка даних об’єднана в певний кластер, її членство є незмінним, тобто майбутнє видалення точки з кластеру неможливе . Однак для подолання цього обмеження був розроблений метод видалення кластерів. Але нажаль використання цього підходу є недоцільним для використання обробки потокових даних, оскільки цей метод вимагає декількох сканувань даних.

BIRCH [5] спочатку був розроблений для аналізу та обробки традиційних статичних даних. Однак він також використовувався для аналізу і потокових даних через його придатність для використання з великими обсягами даних, що породжують мікро- та макрокластерні концепції. Ці два поняття дозволяють BIRCH подолати два основних недоліки, знайдені в алгоритмі HAC, а саме: масштабованість і неможливість видалення точок з кластеру.

Даний алгоритм виконується в два етапи: на першому кроці він сканує базу даних, а потім створює дерево, що складається з інформації про кластери даних. На другому кроці BIRCH скорочує дерево, видаливши розріджені вузли

(виходи) і генеруючи нові оригінальні кластери. Однак цей метод має великий недолік у вигляді обмеженої потужності для обробки гілок дерев.

Оскільки алгоритм кластеризації BIRCH контролює границю кластера, застосовуючи поняття радіус / діаметр, застосування даного алгоритму можливе лише для кластерів сферичної форми.

Online divisive agglomerative clustering (ODAC) [6] це потоковий метод кластеризації для обробки часових рядів. Цей алгоритм здатний обробляти тренд використовуючи як агломеративні, так і розмежувальні ієрархічні методи. Він використовує стратегію “зверху вниз” для підтримки деревоподібної ієрархії кластерів. Для розщеплення кожного вузла використовується міра відмінності (критерій розщеплення) на кореляційній основі, після чого агломеративна стратегія використовується для покращення виявлення концепції дрейфу серед потокових даних.

Також запропоновано алгоритм E-Stream [7], який є еволюційним підходом для кластеризації потокових даних. Цей алгоритм підтримує п’ять видів еволюції:

- поява нового кластеру шляхом агломерації достатніх точок в області;
- зникнення існуючих кластерів, використовуючи метод затухання;
- розвиток кластера шляхом зміни поведінка даних;
- злиття пари схожих кластерів;
- розщеплення кластера на два підкластера;

Існує і розширення алгоритму E-Stream, відомий як HUE-Stream [8], це алгоритм кластеризації, який оснований на розвитку, для підтримки невизначеності в неоднорідних потокових даних. В даному алгоритмі для обробки невизначеності у категоріальних атрибутах використовується функція відстані з розподілом ймовірності двох атрибутів. Запропонована функція відстані використовується для об’єднання кластерів та пошуку найближчого кластеру для отриманих нових потоків даних. В науковій роботі “A framework for clustering uncertain data streams” було проаналізовано результати роботи двох алгоритмів, а саме HUE-Stream порівнювався UMicro, який є алгоритмом

групування невизначених потокових даних, і було встановлено, що HUE-Stream перевершив UMicro за рівнем якості створених кластерів, але потребував більшої кількості параметрів, які необхідно задати користувачем.

Деякі методи кластеризації потокових даних базуються на методах розподілу, таких як k-медіана та k-середнє. Для групування потокових даних використовується алгоритм StreamLearch, даний алгоритм функціонує на основі k-медіана. Даний алгоритм функціонує в два етапи. Алгоритмом STREAM здійснюється визначення розміру вибірки. Після цього якщо розмір зразка більший, ніж результат, визначений із заздалегідь визначеного рівняння, застосовується алгоритм LSEARCH. Кожна частина даних обробляється аналогічно, після чого алгоритм LSEARCH застосовується знову вже для створюваних кластерних центрів.

Для створення бінарних кластерів потокових даних, існує інкрементний алгоритм k-середніх. Експерименти які описанні в роботі C. Ordonez, “Clustering binary data streams with K-means” показали, що цей модифікований алгоритм набагато кращий, ніж масштабований підхід k-середніх. Головними перевагами використання бінарних даних є те, що це полегшує маніпулювання категоріальними даними та можна обминути один з етапів підготовки даних до кластеризації, а саме нормування даних. Даний алгоритм діє на основі оновлення центру і ваги кожного кластеру після перевірки кількох транзакцій.

Інший метод розділення, CluStream [9], - це двоетапний метод кластеризації, який об’єднує дані за допомогою онлайнового мікрокластерування та автономного макрокластерного компонента. Перший крок передбачає отримання статистичних даних з потоку даних, який виконується на етапі онлайнового мікрокластерування. Потім другий етап використовує ці статистичні дані, а також інші дані для створення кластерів. Тим не менше, алгоритм k-середніх, вбудований в етап офлайнового макрокластера цього двоетапного методу, має ряд недоліків, основним з яких є неможливість алгоритму виявлення кластерів довільної форми. K-means більше зосереджується на виявленні сферичних кластерів, хоча не опуклі та

переплетені кластери також використовуються в різних областях застосування. Головним недоліком цього алгоритму є те, що він не здатний виявляти шум та викиди, а також він не придатний до використання з великими об’ємами даних, оскільки він потребує багаторазового сканування. Таким чином, для обробки цих недоліків, алгоритм кластеризації CluStream стискає вхідні потоки даних в мікрокластери під час фази так званої онлайн кластеризації, а потім обробляє ці мікрокластери вже в автономному режимі.

Алгоритм HPStream [10] був розроблений як розширення для алгоритму CluStream. Новий алгоритм це проектована кластеризація для багаторозмірних потокових даних. Головною причиною створення цього алгоритму було те, що CluStream працює неефективно, коли необхідно здійснити кластеризацію багаторозмірних потокових даних.

Алгоритм SWClustering здатний ідентифікувати кластери в потокових даних використовуючи модель ковзного вікна (the sliding windows model). За допомогою цього алгоритму введена нова структура даних, відома як функція кластерної експоненціальної гістограми (EHCF), і вона здатна виявляти еволюцію в кластері. Даний алгоритм здатний розрахувати кластери на основі синопсисів, які забезпечує EHCF. Незважаючи на це, SWClustering був розроблений за допомогою алгоритму k-середніх, таким чином він не може виявляти кластери довільної форми, а також нездатний для обробки викидів та шумів у вхідних даних.

STREAMKM ++ [11] - це алгоритм кластеризації який в своїй основі використовує алгоритм k-means. Він підходить для групування потоків даних у евклідовому просторі. Якщо кількість кластерних центрів велика, якість результатів, отриманих з цього алгоритму, краща, ніж у BIRCH і StreamLearch, але з точки зору часу роботи цей алгоритм працює повільніше, ніж BIRCH.

Алгоритми кластеризації на основі сітки, такі як CLIQUE, WaveCluster та STING мають унікальний принцип дії, оскільки їхній час обробки не залежить від кількості точок даних, що робить їх такими, що слабко залежать від розміру набору даних. Ці алгоритми використовують багатоступеневу структуру сітки.

Ця структура розділяє простір об’єкта на заздалегідь визначену кількість клітинок. Потім ці клітинки утворюють сіткову структуру, де виконуються всі процеси кластеризації.

GCHDS – це алгоритм на основі використання сіткової структури для кластеризації багаторозмірних потокових даних. Він відповідає трьом вимогам щодо кластеризації потокових даних, а саме виконання одноразового сканування даних, високошвидкісна обробка цих даних та обмежене використання пам’яті. У цьому алгоритмі структура сітки використовується для створення синопсису потокових даних. Час збереження структури сітки занадто короткий, і ним можна знехтувати, коли мережа була розширена досить широко, щоб містити більшість даних у потоці. Аналізуючи розподіл даних по кожному вимірюванню, необхідні розміри даних вибираються для побудови підпростору, в якому виконується процес кластеризації. Експерименти показали, що алгоритм GCHDS має високу точність кластеризації, коли параметри правильно встановлені. Він перевершує алгоритм HPStream, який також є алгоритмом кластеризації підпростору для високомірних потоків даних з точки зору точності кластеризації. Проте алгоритм GCHDS може знаходити лише кластери, що належать одному і тому ж підпростору, тоді як у реальних наборах даних кластери можуть належати до різних підпросторів.

Для вирішення цієї проблеми запропоновано grid-алгоритм кластеризації підпросторів для широкоформатних потоків даних (GSCDS), який може виявити кластери в різних підпросторах. Для створення синопсису потокових даних цей алгоритм використовує структуру даних сітки, яка була розподілена рівномірно. Після цього, щоб знайти підпростори, які складаються з кластерів, застосовується методика, що базується на вершині сітки. Потім, для виявлення кластерів у кожному підпросторі, GSCDS застосовує технологію, основану на сітці знизу вгору. Експериментальні результати показують, що GSCDS перевершує GCHDS з точки зору якості кластеризації.

Для потоків даних, сформованих в сенсорних мережах запропоновано розподілений алгоритм кластеризації з використанням сітки, відомий як

DGClust. Він використовує структуру сітки, щоб підсумувати потік даних. Цей алгоритм дозволяє кожному локальному датчику зберігати онлайн-дискретизацію своїх потокових даних, і виконується з фіксованим часом та оновленням, щоб зменшити розмірність та навантаження на зв’язок.

Алгоритми на основі щільності мають значні переваги для кластеризації даних, такі як:

– здатність виявляти кластери довільної форми;
– здатність виявляти шум;
– вимагають лише одноразового сканування вихідних даних.

Крім того, такі алгоритми не потребують попереднього знання кількості кластерів (k), на відміну від алгоритмів k-means, для яких потрібно заздалегідь знати число кластерів.

DBSCAN , GDBSCAN і DENCLUE – це алгоритми кластеризації на базі щільності, які можуть бути використані для виявлення будь-яких скупчень довільної форми. Однак вони непридатні для виявлення кластерів у потоках даних. Алгоритм DBSCAN (Density Based Spatial Clustering Of Applications with Noise) – алгоритм для кластеризації потокових даних з наявністю шуму. Був запропонований як рішення проблеми розбиття (в першу чергу для потокових) даних на кластери довільної форми. Більшість алгоритмів створюють кластери по формі близькі до сферичних, так як мінімізують відстань об’єктів до центру кластера. Автори DBSCAN експериментально показали, що їх алгоритм здатний розпізнавати кластери довільної форми.

Ідея, яка покладена в основу алгоритму, полягає в тому, що всередині кожного кластеру спостерігається типова щільність точок(об’єктів), яка помітно більша, чим щільність зовні кластера, а також щільність в областях з шумом нижче щільності будь-якого з кластерів.

Розглянутий алгоритм кластеризації володіє рядом переваг, а саме:

1) Алгоритм не чутливий до викидів. Тобто в процесі кластеризації всі викиди виносяться в окремий кластер з заздалегідь заданою міткою.

2) Даний метод не потребує апріорного задання кількості кластерів.

3) Використання даного методу дозволяє працювати з кластерами різної природи (форми).

4) Використання даного алгоритму дозволяє працювати з вибірками великого об’єму.

Проте суттєвим недоліком алгоритму являється достатньо трудомістка процедура визначення необхідних параметрів для коректної роботи алгоритму та неможливість обробки потокових даних.

Було розроблено розширення DBSCAN, відоме як incremental-DBSCAN. Цей метод вміло додає та видаляє точки поступово в сховищах даних. За допомогою правильно налаштованих параметрів він здатний виявляти довільно сформовані кластери.

Для статичних даних, алгоритм the OPTICS – це реалізація алгоритму кластеризації на основі щільності, який залежить від тих же параметрів що і в алгоритмі кластеризації DBSCAN, а саме максимальною відстанню між точками та мінімальну кількість точок в кластері. Він містить два методи для організації точок: 1) основна відстань і 2) відстань досяжності. У процесі групування, відстань досяжності та просторового позиціонування, організовані точками слід додати до списку структур кластеризації. Він включає в себе комплексне налаштування параметрів для кластеризації однієї структури. На жаль, OPTICS непридатний для використання для потокових даних, хоча він ідеально підходить для залежних від параметра проблем і здатний виявити накладені кластери та довільні кластери.

Також було запропоновано ще одне вдосконалення алгоритму DBSCAN, відомий як LDBSCAN. Цей алгоритм використовує поняття локальної щільності кластеризації. Він здатний виявляти локальні викиди та шум, що основані на щільності. Однак цей алгоритм не працює належним чином для потокових даних.

Алгоритм на основі двофазної схеми щільності, відомий як DenStream, був розроблений для кластеризації потокових даних. На першому етапі цей алгоритм використовує модель згасаючого вікна (the fading window model –

модель, заснована на використанні вікна спостереження з згасаючим врахуванням значень даних), щоб створити синопсис даних. Потім на другій фазі використовується синопсис даних, що зберігаються з першої фази, для отримання результату кластеризації. Цей алгоритм може обробляти довільні сформовані кластери.

Покращенням алгоритму DenStream є rDenStream, який є трифазним алгоритмом кластеризації. У цьому алгоритмі попередньо відкинуті незначні кластери зберігаються в буфері. Цей підхід гарантує, що ці дані мають можливість формувати кластери та підвищити точність кластеризації. rDenStream може обробляти величезну кількість потоків, перші два етапи яких можна порівняти з DenStream, але має додатковий етап, відомий як ретроспектива. Цей етап дозволяє алгоритму вивчати відкинуті дані, щоб підвищити його точність. Від експериментального порівняння, rDenStream перевершує DenStream на початковій стадії. Однак цей алгоритм вимагає більше часу та пам’яті у порівнянні з DenStream, оскільки він обробляє та зберігає історичний буфер. Алгоритм D-Stream не в змозі обробляти дуже великі об’ємні дані; однак алгоритм DenStream не має труднощів при обробці таких даних.

Алгоритм MR-Stream розподіляє всі нові дані на відповідні клітинки на кожному інтервалі між штампами в режимі онлайн та також оновлює підсумкові дані. У порівнянні між MR-Stream та D-Stream, MR-Stream показав кращу продуктивність.

Інший алгоритм кластеризації, оснований на щільності, для потокових даних – алгоритм DSCLU. DSCLU використовує мікрокластери для виявлення відповідних кластерів, зосереджуючи увагу на локалізації домінуючих мікрокластерів на основі ваги сусідів. Вона здатна виявляти кластери в середовищах з великою щільністю.

OPCluStream – це ще один алгоритм, оснований на щільності для кластеризації потоків даних. Цей алгоритм використовує топологію дерева для

організації точок та напрямних покажчиків, щоб пов’язати всі точки разом. Цей алгоритм здатний виявити довільні та перекриваючі кластери.

Інший тип кластеризації – це метод, заснований на моделях, який керує гіпотетичною моделлю для кожного кластеру та визначає, які дані ідеально відповідають моделі. Алгоритм COBWEB – це один з таких алгоритмів на основі моделей, який є додатковим концептуальним методом для кластерного аналізу. Цей метод використовує структуру дерева, створену за допомогою процесу категоріювання. Він генерує ієрархічну кластеризацію у вигляді дерева класифікації. У цій формі кожен вузол зберігає дані і має імовірнісний опис цих даних, що узагальнює об’єкти, класифіковані за вузлами. COBWEB може виявити викиди, але оскільки він використовує структуру дерева, це обмежує можливості листків.

CluDistream – це алгоритм, розроблений на основі методу максимізації очікувань для кластеризації потокових даних. Цей алгоритм здатний розглядати лише проблему кластеризації в орієнтовному часову вікні з максимізацією очікувань, виконуваною на кожному вузлі розподіленої мережі. Тим не менш, CluDistream показав значні результати, особливо коли він реалізується в розподілених поточних середовищах, де передані дані можуть бути або шумними, або відсутніми.

Алгоритм SWEM об’єднує потоки даних у затухаючому вікні за часом з технікою максимізації очікувань. Цей алгоритм складається з двох фаз, на першому етапі шляхом сканування даних, він створює синопсис цих даних як мікрокомпонентів. Після цього, на другій фазі, цей синопсис даних використовується для створення глобальних кластерів даних. Ця двоступенева структура призначена для вирішення проблем з обмеженою пам’яттю та однокористувальницькою обробкою потоку даних. Цей алгоритм здатний виявляти шум і правильно обробляти відсутні дані.

Всі вищеназвані алгоритми виконують процес кластеризації фокусуючись на різних аспектах. Наприклад, деякі з них підкреслюють необхідність обробки шумів та викидів, тоді як інші нехтують цим аспектом. Деякі з цих алгоритмів є

більш точними, ніж інші, але вони часто мають недолік високої складності. Деякі з них використовують весь потік даних для створення кластерів даних, але інші просто використовують синопсис потоку даних. Отже, в даний час немає алгоритму, який пропонує найкращу продуктивність в плані всіх необхідних функцій, таких як висока якість, низький обчислювальний процес, виявлення шумів і т.д.

Головні переваги та недоліки методів кластеризації наведені в наступній таблиці.

Таблиця 2.2 – Переваги та недоліки методів кластеризації

Методи	Переваги	Недоліки
На основі розбиття	1) Легко реалізувати 2)Використання ітеративного способу створення кластерів	1)Кількість кластерів повинна бути попередньо визначена користувачем 2)Можна визначити лише сферичні кластери
Ієрархічні	Легко обробляти будь-які форми подібності або відстані	1)Неоднозначність критеріїв припинення дії 2)Висока складність
На основі сітки	1)Швидкий час оброблення 2)Можливість виявлення шумів	1)Неможливо застосувати до великих розмірів даних 2)Розмір сітки повинен бути попередньо визначений
На основі щільності	1)Можливість виявляти кластери будь-якої форми 2)Можливість виявлення шуму	1)Кілька параметрів потрібно заздалегідь надавати 2)Не працює добре в багатозначних даних
На основі моделі	1)Вказівка кількості кластерів автоматично на основі стандартної статистики 2)Може справлятися з шумами	Залежно від гіпотезованої моделі або структур

Висновок до розділу 2

Отже, кластерний аналіз – це задача розбиття множини об’єктів на групи (кластери). Методи кластеризації діляться на методи за способом обробки даних, за способом аналізу даних та за часом виконання алгоритму кластеризації. Для здійснення кластеризації використовуються різні метрики, а саме метрика Евкліда, манхетенська відстань та відстань Чебишева.

Основним напрямком досліджень в області обробки потокових даних є розробка найбільш ефективних методів обробки та аналізу цих даних. Проте задача видобутку цінних, корисних даних ускладнена через специфічні характеристики потокових даних; вони масивні, навіть потенційно нескінчені, і, крім того, безперервні, вимагають одноразового сканування і динамічно змінюються протягом часу, що вимагає швидкої реакції зазвичай в режимі реального часу. Підхід кластеризації потокових даних є одним із методів виведення даних, який може витягувати знання з таких даних. Звичайні методи кластеризації не є достатньо гнучкими для вирішення цих проблем.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПОТОКОВОГО АЛГОРИТМУ КЛАСТЕРИЗАЦІЇ DENSTREAM

3.1 Вибір програмного засобу

У розробленні програми був реалізований один з описаних вище алгоритмів кластеризації, а саме алгоритм DenStream.

Вся логіка та реалізація алгоритму кластеризації була написана мовою Python у середовищі розробки Pycharm. Крім того для належної візуалізації використовувалась бібліотека Matplotlib та для порівняння ефективності роботи з іншими алгоритмами кластеризації використовувалася бібліотека scikit-learn.

Програмний продукт призначено для використання на персональних комп'ютерах під управлінням будь-якої операційної системи.

Добре відомо, що Python є одним із лідерів в сенсі застосування мов програмування в області data science. Доречність використання Python для інтелектуального аналізу даних полягає в наступному.

Об'єктно-орієнтованість. Python є об'єктно-орієнтованою мовою програмування. Його об'єктна модель підтримує такі поняття, як поліморфізм, перевантаження операторів і множинне спадкування, однак, враховуючи простоту синтаксису і типізації Python, ООП не викликає складнощів в застосуванні.

Доступність. Python може використовуватися і розповсюджуватися абсолютно безкоштовно. Як і у випадку з іншими відкритими програмними продуктами, такими як Tel, Perl, Linux і Apache, ви зможете отримати в Інтернеті повні вихідні тексти реалізації Python. Немає ніяких обмежень на його копіювання, вбудовування в свої системи або поширення в складі ваших продуктів. Фактично ви зможете навіть продавати вихідні тексти Python, якщо з'явиться таке бажання. Але "доступний" не означає, що він не підтримується. Навпаки, співтовариство прихильників Python в Інтернеті відповідає на запитання користувачів зі швидкістю, якої могли б позаздрити більшість

розробників комерційних продуктів. Крім того, вільне поширення вихідних текстів Python сприяє розширенню команди експертів з реалізації.

Кросплатформеність. Стандартна реалізація мови Python написана на переносимому ANSI 3, завдяки чому він компілюється і працює практично на всіх основних платформах. Наприклад, програми на мові Python можуть виконуватися на найширшому спектрі пристроїв, починаючи від наладонних комп'ютерів (PDA) і закінчуючи суперкомп'ютерами. Нижче наводиться далеко неповний список операційних систем і пристроїв, де можна використовувати Python:

1) Операційні системи Linux і UNIX;
2) Microsoft Windows і DOS (всі сучасні версії);
3) Mac OS (обидва різновиди: OS X і Classic);
4) BeOS, OS / 2, VMS і QNX;
5) Системи реального часу, такі як VxWorks;
6) Суперкомп'ютери Cray і EOM виробництва компанії IBM;
7) Наладонні комп'ютери, що працюють під управлінням PalmOS, PocketPC або Linux;
8) Стільникові телефони, що працюють під управлінням операційних систем;
9) Symbian і Windows Mobile.

Крім самого інтерпретатора мови в складі Python поширюється стандартна бібліотека модулів, яка також реалізована належним шляхом. Крім того, програми на мові Python компілюються в байт-код, який однаково добре працює на будь-яких платформах, де встановлена сумісна версія Python .

Потужність. З точки зору функціональних можливостей Python можна назвати гібридом. Його інструментальні засоби вкладаються в діапазон між традиційними мовами сценаріїв (такими як Tel, Scheme і Perl) і мовами розробки програмних систем (такими як C, C ++ і Java). Python забезпечує простоту і невимушеність мови сценаріїв і міць, яку зазвичай можна знайти в компільованих мовами. Перевищуючи можливості інших мов сценаріїв, така

комбінація робить Python зручним засобом розробки великомасштабних проектів. Для попереднього ознайомлення нижче наводиться список основних можливостей, які є в арсеналі Python.

Динамічна типізація. Python сам стежить за типами об’єктів, що використовуються в програмі, завдяки чому не потрібно писати довгі і складні оголошення в програмному коді. Насправді, в мові Python взагалі відсутні поняття типу і необхідність оголошення змінних. Так як програмний код на мові Python не обмежений рамками типів даних, він автоматично може обробляти цілий діапазон об’єктів.

Автоматичне управління пам’яттю. Python автоматично розподіляє пам’ять під об’єкти і звільняє її, коли об’єкти стають непотрібними. Більшість об’єктів можуть збільшувати і зменшувати обсяг пам’яті, яку вони обіймають по мірі необхідності. Як відомо, Python сам виконує всі низькорівневі операції з пам’яттю, тому вам не доведеться турбуватися про це.

Модульне програмування. Для створення великих систем Python надає такі можливості, як модулі, класи і виключення. Вони дозволяють розбити систему на складові, застосовувати ООП для створення програмного коду багаторазового користування і елегантно обробляти події та помилки, що виникають.

Вбудовані типи об’єктів. Python надає найбільш типові структури даних, такі як списки, словники і рядки, у вигляді особливостей, властивих самій мові програмування. Як ви побачите пізніше, ці типи відрізняються високою гнучкістю і зручністю. Наприклад, вбудовані об’єкти можуть розширюватися і стискатися в міру необхідності, можуть комбінуватися один з одним для представлення даних зі складною структурою і багато іншого.

Вбудовані інструменти. Для роботи з усіма цими типами об’єктів в складі Python є потужні і стандартні засоби, включаючи такі операції, як конкатенація (об’єднання колекцій), отримання зрізів (витяг частини колекції), сортування, відображення і багато іншого.

Бібліотеки утиліт. Для виконання більш вузьких завдань до складу Python також входить велика колекція бібліотечних інструментів, які підтримують практично все, що тільки може знадобитися, - від пошуку з використанням регулярних виразів до роботи в мережі. Бібліотечні інструменти мови Python – це те місце, де виконується велика частина операцій.

Утиліти сторонніх розробників. Python – це відкритий програмний продукт і тому розробники можуть створювати свої попередньо скомпільовані інструменти підтримки завдань. У мережі ви знайдете вільну реалізацію підтримки COM, засобів для роботи з зображеннями, розподілених об'єктів CORBA, XML, механізмів доступу до баз даних і багато іншого. Незважаючи на широкі можливості, Python має надзвичайно простий синтаксис і архітектуру. В результаті ми маємо потужний інструмент програмування, що володіє простотою і зручністю, властивими мовам сценаріїв.

Numpy — розширення мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами. Попередник Numpy, Numeric, був спочатку створений Jim Hugunin. Numpy — відкрите програмне забезпечення і має багато розробників. Оскільки Python — інтерпретована мова, математичні алгоритми, часто працюють в ньому набагато повільніше ніж у компільованих мовах, таких як C або навіть Java. NumPy намагається вирішити цю проблему для великої кількості обчислювальних алгоритмів забезпечуючи підтримку багатовимірних масивів і безліч функцій і операторів для роботи з ними. Таким чином будь-який алгоритм який може бути виражений в основному як послідовність операцій над масивами і матрицями працює також швидко як еквівалентний код написаний на C.

Бібліотека scikit-learn представляє реалізацію цілого ряду алгоритмів для навчання з вчителем і навчання без вчителя через інтерфейс для мови програмування Python. Scikit-learn побудована поверх SciPy, який має бути встановлений перед використанням Scikit-learn. Дана бібліотека включає в себе:

– NumPy: розширення мови Python, яке добавляє підтримку великих багатовимірних масивів та матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами;

– SciPy: відкрита бібліотека високоякісних наукових інструментів для мови програмування Python;

– Matplotlib: бібліотека для візуалізації даних;

– IPython: інтерактивна оболонка для мови програмування Python, яка надає додатковий командний синтаксис, підсвічування коду і автоматичне доповнення;

– Pandas: різні структури даних і аналіз.

3.2 Потоковий алгоритм кластеризації DenStream

DenStream це новий алгоритм для виявлення кластерів (груп об’єктів) довільної форми в потокових даних. До основних характеристик цього алгоритму відноситься:

– Базовий мікрокластерний синопсис призначений для узагальнення кластерів з довільною формою в потоках даних.

– Він включає в себе нову стратегію відсікання, що забезпечує можливість створення нових кластерів, одночасно швидко позбавляючись від викидів.

– Вводиться вихідний буфер для відокремлення обробки потенційних ядрових-мікрокластерів та мікрокластерів викидів, що покращує ефективність роботи DenStream.

– Завдяки своїй адаптивності до зміни кластерів та здатності керувати викидами в потоках даних, DenStream забезпечує постійну високу якість кластеризації.

Кластери в потокових даних часто обчислюються на основі певних часових інтервалів (або вікон). Є три відомі моделі вікон: орієнтовні вікна, розсувні та затухаючі вікна.

Ми розглядаємо проблему кластеризації потоку даних в моделі з затухаючим вікном, в якій вага кожної точки даних експоненціально зменшується з часом *t* за допомогою функції

$$f(t) = 2^{-\alpha * t}, \text{ де } \alpha > 0.$$

Експоненціально затухаюча функція широко використовується у часових програмах, де бажано поступово знижувати історію колишньої поведінки. Чим вище значення α, тим менша важливість історичних даних у порівнянні з останніми даними. І загальна вага потокових даних це константа

$$W = v\left(\sum_{t=0}^{t=t_c} 2^{-\alpha * t}\right) = \frac{v}{1-2^{-\alpha}},$$ де t_c $(t_c \to \infty)$ це поточний час,

v означає швидкість потоку даних, тобто це кількість точок що приходять за одиницю часу.

У статичному середовищі кластери з довільною формою представлені всіма точками, що входять в вхідний датасет. Коли приходить запит на кластеризацію, ці точки можуть бути об’єднані алгоритмом DBSCAN [9], щоб отримати кінцевий результат. Результатом кластеризації є група (зважених) основних об’єктів С з відповідними кластерними мітками, які гарантують, що об’єднання ∈-сусідства С охоплює область щільності.

Основний об’єкт визначається як об’єкт, в чийому районі $\in -$сусідства загальна вага точок даних є принаймні цілим числом μ.

Область щільності визначається як об’єднання $\in -$сусідніх основних об’єктів.

Практично неможливо забезпечити точний результат. Тому використовують мікрокластерну структуру.

Основний-мікрокластер(*с-мікрокластер*) в момент часу t визначається як сукупність параметрів (вага, радіус та центр мікрокластеру) для групи близьких точок $p_{i_1}, \cdots, p_{i_n}$ з часовими мітками $T_{i_1}, \cdots, T_{i_n}$, де $w = \sum_{j=1}^{n} f(t - T_{i_j}), w \geq \mu,$ це вага, $c = \frac{\sum_{j=1}^{n} f\left(t - T_{i_j}\right) * p_{i_j}}{w}$, це центр. $r = \frac{\sum_{j=1}^{n} f(t - T_{i_j}) * dist(p_{i_j}, c)}{w}, r \leq \in$, це радіус, де $dist(p_{i_j}, c)$ визначається як відстань Евкліда між точкою p_{i_j} та центром c.

Зверніть увагу, що *вага с-мікрокластеру* має бути більша або рівна μ та *радіус* має бути меншим або рівним ∈. Через обмеження на радіус, N_c- кількість *с-мікрокластерів* набагато перевищує кількість природних кластерів. З іншого боку, це значно менше, ніж число точок у потоці даних через обмеження ваги мікрокластеру. Оскільки кожна точка однозначно присвоюється одному з *с-мікрокластерів*, то N_c нижче або дорівнює $\frac{W}{\mu}$. В додаток, коли приходить запит на кластеризацію кожен с-мікрокластер буде помічений, щоб отримати кінцевий результат, як це показано на малюнку 3.1.

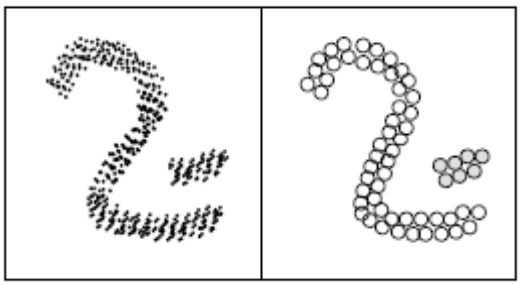


Рисунок 3.1 – а) потокові дані б) створені мікрокластери

Кластери з довільною формою в потоках даних описуються як *набір с-мікрокластерів*. Набори отриманих мікрокластерів повинні повністю описувати всі вхідні точки.

У потокових даних часто відбувається обмін роллю кластерів та викидів, і будь-який *с-мікрокластер* утворюється поступово, коли відбувається надходження потоку даних. Тому ми вводимо структури потенційних *с-мікрокластерів* та *о-мікрокластерів*. Основними відмінностями між ними є їх різні обмеження на вагу,$w \geq \beta * \mu$ та ,$w < \beta * \mu$, відповідно.

Потенційний *с-мікрокластер (р-мікрокластер)* в час *t* для групи близьких точок $p_{i_1}, \cdots, p_{i_n}$ з часовими мітками $T_{i_1}, \cdots, T_{i_n}$ визначається як $\{\overline{CF^1}, \overline{CF^2}, w\}. w = \sum_{j=1}^{n} f(t - T_{i_j}), w \geq \beta\mu$, це вага. $\beta, 0 < \beta \leq 1$ є параметром для визначення порога викиду відносно с-мікрокластерів. $\overline{CF^1} = \sum_{j=1}^{n} f(t - T_{i_j}) p_{i_j}$ - це зважена лінійна сума точок. $\overline{CF^2} = \sum_{j=1}^{n} f(t - T_{i_j}) p_{i_j}^2$ - це зважена квадратна сума точок.

Центр в *p-мікрокластері* це $c = \frac{\overrightarrow{CF^1}}{w}$. А радіус в *p-мікрокластері* це $r = \sqrt{\frac{|\overrightarrow{CF^2}|}{w} - \left(\frac{\overrightarrow{CF^1}}{w}\right)^2}, (r \leq \in)$

Мікрокластер викиду (*o-мікрокластер*) в час t для групи близьких точок $p_{i_1}, \cdots, p_{i_n}$ з часовими мітками $T_{i_1}, \cdots, T_{i_n}$ визначається як $\{\overrightarrow{CF^1}, \overrightarrow{CF^2}, w, t_0\}$. Визначення для $\overrightarrow{CF^1}, \overrightarrow{CF^2}, w$, центру та радіусу таке ж як і для *p-мікрокластеру*. $t_0 = T_{i_1}$ означає як час створення *o-мікрокластеру*, який використовується для опису життєвого циклу *o-мікрокластеру*. Проте $w < \beta\mu$. Це тому що вага менша чим поріг викиду, мікрокластер відповідає викидам.

Алгоритм кластеризації DenStream можна розділити на дві частини: онлайн частина обслуговування мікрокластерів та офлайн частина генерування фінального кластеру, на вимогу користувача.

Щоб виявити кластери в потокових даних ми знаходимо групу *p-мікрокластерів* та *o-мікрокластерів* в режимі онлайн. Всі *o-мікрокластери* зберігаються в окремому просторі пам'яті, скажімо, буфер викидів. Це базується на спостереженні, що більшість нових точок належать до існуючих кластерів і тому можуть бути поглинені існуючими *p-мікрокластерами*.

Коли приходить нова точка *p,* процедура приєднання описана нижче (Алгоритм 1):

1. Спочатку, ми намагаємось приєднати точку p до найближчого *p-мікрокластеру* c_p. Якщо r_p, новий радіус для c_p, менший або рівний $\in$, приєднуємо p до c_p. Це може бути досягнуто за рахунок інкрементної властивості *p-мікрокластерів*.

2. В іншому випадку, ми намагаємось приєднати *p* до найближчого до неї *o-мікрокластеру* c_o. Якщо r_o, новий радіус для c_o, менший або рівний $\in$, приєднуємо *p* до c_o. І після цього перевіряємо w нову вагу c_o. Якщо w вища за $\beta\mu$, це значить що c_o виріс в потенційний с-мікрокластер. Після цього ми видаляємо c_o з буферу викидів та створюємо новий р-мікрокластер з c_o.

3. Інакше ми створюємо новий *о-мікрокластер* c_o з p та вставляємо c_o в буфер викидів. Це тому, що p вписується в будь-який існуючий мікрокластер. P може бути викидом або основою для нового мікрокластеру.

Algorithm 1 Merging (p)

```
1: Try to merge p into its nearest p-micro-cluster c_p;
2: if r_p (the new radius of c_p) ≤ ε then
3:    Merge p into c_p;
4: else
5:    Try to merge p into its nearest o-micro-cluster c_o;
6:    if r_o (the new radius of c_o) ≤ ε then
7:       Merge p into c_o;
8:       if w (the new weight of c_o) > βμ then
9:          Remove c_o from outlier-buffer and create a
            new p-micro-cluster by c_o;
10:      end if
11:   else
12:      Create a new o-micro-cluster by p and insert it
         into the outlier-buffer;
13:   end if
14: end if
```

Для кожного існуючого *p-мікрокластеру* c_p, якщо в нього не буде приєднано нової точки, вага c_p буде поступово поступово зменшуватися. Якщо вага менша за $\beta\mu$, це значить що c_p перетворюється в викид і він буде видалений та його простір пам'яті буде призначений для нового *p-мікрокластеру*. Ми маємо періодично перевіряти цю вагу для кожного *p-мікрокластеру*. Постає проблема як визначити цей період перевірки. Мінімальний проміжок часу для виходу *p-мікрокластера* у витік:

$T_p = \left\lceil \frac{1}{\alpha} \log\left(\frac{\beta\mu}{\beta\mu - 1}\right) \right\rceil$, який визначається рівнянням $2^{-\alpha T_p}\beta\mu + 1 = \beta\mu$. Тому ми перевіряємо кожний *p-мікрокластер* кожні T_p періоди часу. Ця стратегія перевірки забезпечує що максимальне число *p-мікрокластерів* в пам'яті рівна $\frac{W}{\beta\mu}$, загальна вага потоку даних це константа W.

Проблема полягає в тому, що кількість *о-мікрокластерів* може постійно збільшуватися. Алгоритм працює гірше, коли існує велика кількість викидів. З

іншого боку, ми повинні зберігати *о-мікрокластери*, які ростуть у *р-мікрокластери*, оскільки на початковій стадії будь-якого нового кластеру його вага порівняно невелика в порівнянні з існуючими кластерами. Отже, ми повинні надати можливість *о-мікрокластеру* перетворитися на *р-мікрокластер*, одночасно швидко позбавившись *о-мікрокластера*, який є реальним викидом. Для того щоб визначити чи *о-мікрокластер* може стати *р-мікрокластером* чи ні необхідно чекати нескінчений час. Проте ця стратегія коштує великої пам'яті. Тому ми запроваджуємо приблизний спосіб розрізнити ці два типи о-*мікрокластерів* та періодично обрізати "реальні" мікрокластери, які точно не зможуть перерости в *р-мікрокластери*.

Необхідно перевіряти кожен *о-мікрокластер* кожні T_p періоди часу. У цей період ми порівнюємо вагу кожного *о-мікрокластеру* з його нижньою межею ваги (позначається як ε). Якщо вага *о-мікрокластеру* нижча або рівна цієї нижньої межі ваги, то *о-мікрокластер* не може перетворитися на *р-мікрокластер*. І ми можемо його безпечно видалити з буферу викидів. Нижня межа ваги вираховується:

$$\varepsilon(t_c, t_o) = \frac{2^{-\alpha(t_c - t_o + T_p)} - 1}{2^{-\alpha T_p} - 1}$$

де функцією t_c (являється поточним часом) і t_o (являється часом створення *о-мікрокластеру*). Коли $t_c = t_o$, тобто поточний час збігається з часом створення кластеру, $\varepsilon = 1$. По закінченню часу, ε збільшується та $\lim_{t_c \to \infty} \varepsilon(t_c) = \frac{1}{1 - 2^{-\alpha T_p}} = \beta\mu$. Тобто, чим довше існує о-мікрокластер, тим більша її вага. Детальна процедура описана в алгоритмі 2.

Algorithm 2 DenStream (DS, ϵ, β, μ, λ)

$T_p = \lceil \frac{1}{\lambda} \log(\frac{\beta\mu}{\beta\mu-1}) \rceil$;
Get the next point p at current time t from data stream DS;
Merging(p);
if (t mod T_p)=0 **then**
  for each p-micro-cluster c_p **do**
    if w_p(the weight of c_p)< $\beta\mu$ **then**
      Delete c_p;
    end if
  end for
  for each o-micro-cluster c_o **do**
    $\xi = \frac{2^{-\lambda(t-to+T_p)}-1}{2^{-\lambda T_p}-1}$;
    if w_o(the weight of c_o)< ξ **then**
      Delete c_o;
    end if
  end for
end if
if a clustering request arrives **then**
  Generating clusters;
end if

Очевидно, що стратегія обрізки *о-мікрокластерів* може ввести деяку похибку на ваги *о-мікрокластерів* і *р-мікрокластерів*. На щастя, ми можемо гарантувати, якщо вага c_p вище $\beta\mu$, даний мікрокластер c_p повинен входити в буфер викидів або групу *р-мікрокластерів*. І якщо поточна вага c_p більша за $2\beta\mu$, він мусить входити в групу *р-мікрокластерів*.

Ініціалізація. Ми використовуємо алгоритм DBSCAN для перших InitN точок $\{P\}$ для ініціалізації онлайнового процесу. Ми ініціалізуємо групу *р-мікрокластерів* за допомогою сканування $\{P\}$.

Мікрокластер, що створюється в режимі онлайн, захоплює область точок потокових даних. Для того, щоб отримати значущі кластери, нам потрібно застосувати деякий алгоритм кластеризації для отримання остаточного результату. Коли з'являється запит на кластеризацію, для набору *р-мікрокластерів*, що створені в онлайн фазі застосовується варіант алгоритму DBSCAN, щоб отримати кінцевий результат кластеризації. Кожен *р-*

мікрокластер c_p розглядається як віртуальна точка, розташована в центрі c_p з вагою *w*.

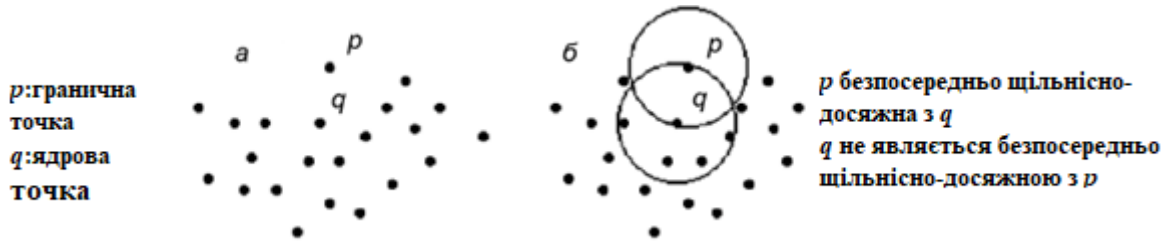
Ідея, яка покладена в основу алгоритму, полягає в тому, що всередині кожного кластеру спостерігається типова щільність точок (об'єктів), яка помітно більша, чим щільність зовні кластера, а також щільність в областях з шумом нижче щільності будь-якого з кластерів.

ε-сусідство точки p, позначається як N_ε, визначається множинна об'єктів, які находяться від точки *p* на відстань не більше ε: $N_\varepsilon(p) = \{q \in D \mid dist(p,q) \le \varepsilon\}$

Евклідова відстань (Евклідова метрика) – формула традиційної відстані між двома точками $p = (p_1, p_2, \dots, p_n)$ та $q = (q_1, q_2, \dots, q_n)$ для Евклідового простору:

$$dist(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Точка p безпосередньо щільно-досяжна з точки q (при заданих ε і MinPts),якщо $p \in N_\varepsilon(p)$, $|N_\varepsilon(p) \ge MinPts|$.



Точка p щільно-досяжна з точки q (при заданих ε і MinPts, якщо існує послідовність точок $q = p_1, p_2, ..., p_n = p : p_{i+1}$, безпосередньо щільно-досяжних з *p*). Це відношення транзитивне, але не симетричне в загальному випадку, проте симетричне для двох ядрових точок.

Точка p щільно-зв'язана з точкою q (при заданих ε і MinPts), якщо існує точка *o: p і q* щільно-досяжні з *o*.

Кластер – це множинна щільно-зв'язаних точок. В кожному кластері знаходиться хоча б *MinPt*s об'єктів.

Шум – це підмножина об’єктів, які не належать жодному кластеру:

$$\left\{p \in D \middle| p \notin C_j, j = 1,2,..,|C|\right\}$$

3.3 Аналіз даних для використання в методі кластеризації DenStream

В якості бази даних яка використовувалась для перевірки роботи вибраного алгоритму кластеризації було вибрано базу даних KDD-99. Програмне забезпечення для виявлення мережевих вторгнень захищає комп’ютерну мережу від несанкціонованого допуску, в тому числі і можливих інсайдерів.

Програма DARPA Intrusion Detection Evaluation в 1998 році була підготовлена і розроблена MIT Lincoln Labs. Для створення системи розпізнавання мережевих вторгнень в першу чергу необхідно орієнтуватись на дані, які безпосередньо циркулюють в комп’ютерній мережі. Але для побудови, тестування і проведення заключних експериментів необхідний вихідний набір даних, в якому представленні основні види активності комп’ютерної мережі. При чому цей набір мусить включати як атаки, так і нормальні з’єднання. Для цієї задачі була створена база записів про мережеві з’єднання KDD-99, яка була представлена на третьому міжнародному змаганні KDD Cup 1999, які проходили паралельно з п’ятою міжнародною конференцією KDD-99.

Ця база містить велику кількість мережевих підключень, які пов’язані з нормальними і аномальними станами мережі. Кількість записів вихідного файлу kddcup.data досягає 4 898 413, кожна з яких – це характеристичний 41-розмірний вектор. Крім цього, запису відповідає поле, в якому вказується характер з’єднання: конкретний тип атаки або нормальний стан. Нормальний стан означає відсутність загрози функціональності вузла і несанкціонованого доступу до даних. Ці записи моделюють адекватну роботу системи. Нормальний стан мережі представлений 972 781 векторами характеристик, що

складає 19,8% від всього об’єму бази. Кожен запис відповідає одному TCP/IP з’єднанню.

protocol	service	flag	src_bytes	dst_bytes	land	wrong_fra	urgent	hot	num_faile	logged_in	num_com	root_shell	su_atemp	num_root	num_fl_cr	num_shel	num_acce	num_outb
tcp	http	SF	181	5450	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	239	486	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	235	1337	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	219	1337	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	217	2032	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	217	2032	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	212	1940	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	159	4087	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	210	151	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	212	786	0	0	0	1	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	210	624	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	177	1985	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	222	773	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	256	1169	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	241	259	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	260	1837	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	241	261	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	257	818	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	233	255	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	233	504	0	0	0	0	0	1	0	0	0	0	0	0	0	0
tcp	http	SF	256	1273	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Рисунок 3.2 – База записів KDD-99

Параметри записів 1-9 представляють собою основоположні характеристики кожного окремого TCP-з’єднання, такі як тривалість, тип протоколу, кількість байт від джерела/отримувача. Параметри 10-22 описують з’єднання на відомих домену характеристиках і включають число операцій створення файлу, кількість невдалих спроб реєстрації і т.д. Параметри 23-41 відносяться до опису трафіку, це величини, які вираховуються з використанням часового вікна 2с.

В випадку з базою даних KDD атаки представленні наступними чотирма класами:

1) DoS (denial of service) атаки – це мережеві атаки, спрямовані на виникнення ситуації, коли в системі, що атакується відбувається відмова в обслуговуванні. Дані атаки характеризуються генерацією великого обсягу трафіку, що призводить до перевантаження та блокування сервера. Виділяють шість DoS атак: back, land, neptune, pod, smurf, teardrop. Атаки категорії DoS найбільш численні. Їм відповідає 79,3% (3 883 370) всіх записів, 2/3 цього об’єму складає один з різновидів DoS атак – smurf. Ця атака виключно ефективна і широко поширена в використанні зловмисниками на даний час в мережі Інтернет.

2) U2R (user-to-root) атаки передбачають отримання зареєстрованим користувачам привілеїв локального суперкористувача (адміністратора). Виділяють чотири типи U2R атак: buffer_overflow, loadmodule, perl, rootkit.

3) R2L (remote-to-local) атаки характеризуються отриманням доступу незареєстрованого користувача до комп'ютера з боку віддаленої машини. Виділяють вісім типів R2L атак: ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster.

4) Probe атаки полягають в скануванні мережевих портів з метою отримання конфіденційної інформації. Виділяють чотири типи Probe атак: ipsweep, nmap, portsweep, satan.

Для використання вибраного методу кластеризації було вирішено враховувати з вище сказаної бази даних десять параметрів, які забезпечать повний опис як нормального стану мережі, так і стану мережі під дією певних атак.

protocol	service	src_bytes	dst_bytes	wrong_frg	logged_in	count	srv_count	dst_host_count	dist_host_srv_count	status
tcp	http	181	5450	0	1	8	8	9	9	normal.
tcp	http	239	486	0	1	8	8	19	19	normal.
tcp	http	235	1337	0	1	8	8	29	29	normal.
tcp	http	219	1337	0	1	6	6	39	39	normal.
tcp	http	217	2032	0	1	6	6	49	49	normal.
tcp	http	217	2032	0	1	6	6	59	59	normal.
tcp	http	212	1940	0	1	1	2	1	69	normal.
tcp	http	159	4087	0	1	5	5	11	79	normal.
tcp	http	210	151	0	1	8	8	8	89	normal.
tcp	http	212	786	0	1	8	8	8	99	normal.
tcp	http	210	624	0	1	18	18	18	109	normal.
tcp	http	177	1985	0	1	1	1	28	119	normal.
tcp	http	222	773	0	1	11	11	38	129	normal.
tcp	http	256	1169	0	1	4	4	4	139	normal.
tcp	http	241	259	0	1	1	1	14	149	normal.
tcp	http	260	1837	0	1	11	11	24	159	normal.
tcp	http	241	261	0	1	2	2	34	169	normal.
tcp	http	257	818	0	1	12	12	44	179	normal.
tcp	http	233	255	0	1	2	8	54	189	normal.
tcp	http	233	504	0	1	7	7	64	199	normal.
tcp	http	256	1273	0	1	17	17	74	209	normal.
tcp	http	234	255	0	1	5	5	84	219	normal.

Рисунок 3.3 – Параметри, які будуть використовуватись з бази даних KDD99

Таблиця 3.1 – Опис даних, що використовуються при кластеризації

№	Параметр	Опис
1	protocol_type	Тип протоколу (tcp,udp і т.п)
2	service	Мережева служба отримувача (http, telnet)
3	src_bytes	Кількість байтів які були передані від джерела до отримувача
4	dst_bytes	Кількість байтів які були передані від отримувача до джерела
5	wrong_fragment	Число “неправильних” фрагментів. Неправильна фрагментація пакетів (прийшов фрагмент з id-номером, який не очікувався; останній фрагмент прийшов з міткою "more fragments"
6	logged_in	1,якщо успішно ввійшли в систему; 0 в іншому випадку
7	count	Кількість підключень до поточного хоста за останні 2 секунди
8	srv_count	Кількість підключень до поточної служби за останні 2 секунди
9	dst_host_count	Кількість підключень до віддаленого хоста за останні 2 секунди
10	dst_host_serv_count	Кількість підключень до віддаленої служби за останні 2 секунди

В рамках використання методів класифікації та кластеризації будемо використовувати 10%-кову вибірку з вихідного файлу kddcup.data – файл kddcup10.data. Кількість записів в цій базі зменшено на порядок за рахунок скорочення числа станів smurf, Neptune, normal, satan, ipsweep, portsweep і nmap.

3.4 Реалізація вибраного методу кластеризації

Вище було вказано, що реалізація щільнісного алгоритму кластеризації з наявністю шумів буде виконано за допомогою інтерпретованої об'єктно-орієнтованої мови програмування Python з використанням бібліотек NumPy, Matplotlib та Sklearn.

До основних вхідних параметрів, які використовуються для виконання алгоритму кластеризації відносяться:

– λ – це історична цінність наших даних. Тобто цей показник відображає наскільки для нас важливі дані про кластери, які вже пройшли фазу онлайн кластеризації.

– $\in$ - даний показник використовуються під виконання функції приєднання нової вхідної точки в кластер.

– β і μ - дані показники використовуються під час перевірки ваги кластерів і допомагають визначити чи кластери вже втратили свою важливість для нас чи ні.

Дані вхідні параметри використовуються лише при ініціалізації онлайнової фази алгоритму DenStream. В офлайн фазі ми використовуємо алгоритм кластеризації DBSCAN. Основними вхідними параметрами для даного алгоритму кластеризації являється eps –це "радіус", тобто максимальна відстань між сусідніми об'єктами, та MinPts – мінімальна кількість сусідніх об'єктів, які необхідні для створення кластеру.

Існують евристики для вибору значення параметрів eps та MinPts. Найчастіше використовується наступний метод:

1) Вибираємо значення MinPts. Зазвичай використовують значення від 3 до 9, чим більш неоднорідним очікується дата сет, і чим більше рівень шуму, тим більше необхідно взяти значення MinPts.

2) Вираховуємо середню відстань по MinPts сусідам для кожної точки. Тобто, якщо MinPts = 3, необхідно вибрати трьох найближчих сусідів, скласти

відстань до них і поділити на три. Відстань від точки до всіх інших точок вираховується за допомогою використання метрики Евкліда.

3) Сортуємо отриманні значення в порядку зростання і виводимо на екран. Отримуємо графік наступного вигляду.

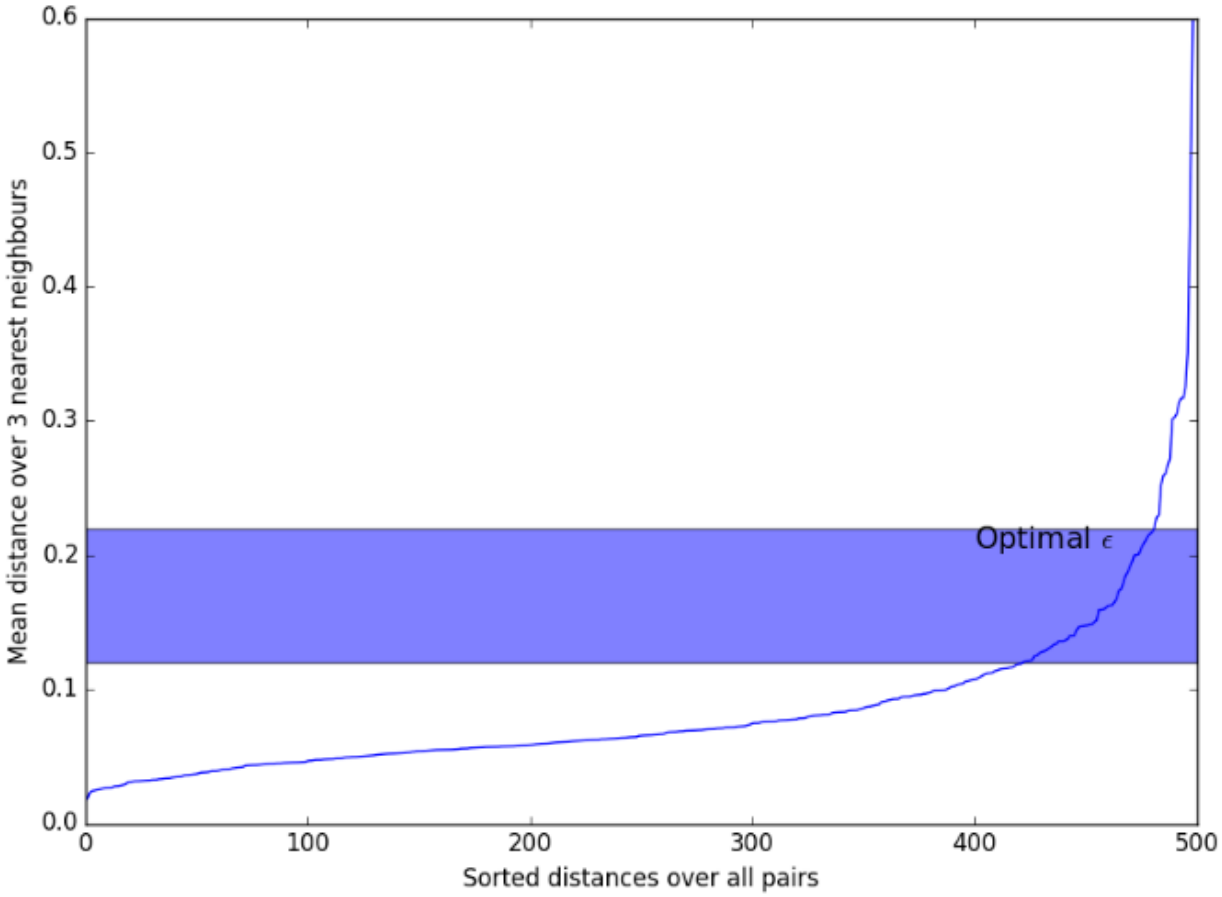


Рисунок 3.4 – Демонстраційний графік, для знаходження оптимального значення eps

4) Бачимо щось на кшталт різко зростаючого графіка. Необхідно взяти eps де-небудь, де відбувається найсильніший перегин. Чим більше eps, тим більші виходять кластери, і тим менше їх буде.

В створеному програмному застосунку використовується цей же алгоритм для знаходження eps та MinPts.

Перед виконанням самого процесу кластеризації методом DBSCAN в програмному засобі використовуються дані, які отримані після онлайн етапу алгоритму DenStream, а саме ваги та центри *р-мікрокластерів*. Потім здійснюється процес нормалізації даних. Необхідність нормалізації даних в

вибірці обумовлена самою природою даних які використовуються. Будучи різними за фізичною своєю природою, вони часто можуть сильно відрізнятися між собою за абсолютними величинами.

В розробленому програмному засобі для того, щоб здійснити нормалізацію вхідних даних виконуються наступні кроки:

1) Для кожного з вхідних параметрів знаходиться максимальне значення в даному параметрі.

2) Після знаходження максимального значення, кожне значення параметрів ділиться на відповідне для цього параметру максимальне значення.

3.5 Аналіз отриманих результатів кластеризації

Для перевірки роботи створеного програмного засобу на базі алгоритму DenStream спочатку використовувалися штучно згенеровані дані різної форми.

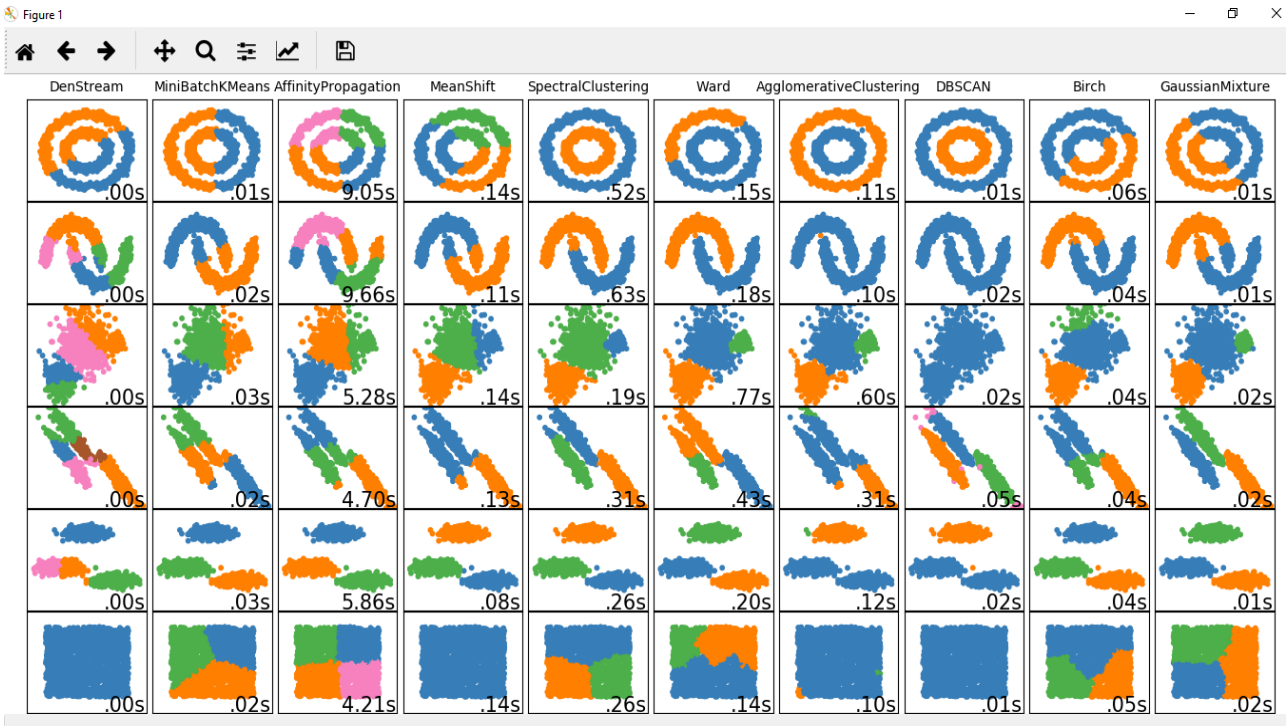


Рисунок 3.5 – Результати кластеризації даних різної форми

Було здійснено перевірку ефективності роботи саме офлайн етапу DenStream, використовуючи найбільш популярні алгоритми кластеризації, які належать до різних типів.

Оцінюючи час виконання та графічне відображення отриманих кластерів, можна зробити висновок, що саме щільнісний алгоритм кластерації DBSCAN найкраще справляється з поставленою перед ним задачею, а саме отримання кластерів різної форми та за наявності в них шумів.

Було проведено кластеризацію вибраної вибірки роботи комп'ютерної мережі для виявлення певних атак і аналіз можливості виявлення декількох атак одночасно.

Над даними вибірки було проведено 5 експериментів для виявлення 5 різних видів атак, а саме атак: smurf, Neptune, nmap, satan, portsweep. Для здійснення експерименту було вибрано 400 даних нормальної роботи мережі та 50 даних, коли відбувалась різні типи атак на мережу.

Завдяки датасету KDD-99 отримуємо інформацію скільки кластерів в себе будуть вміщувати вхідні дані. Після кожного експерименту за допомогою бібліотеки sklearn буде здійснюватися оцінка отриманих кластерів. Використовувалися наступні метрики:

– Коефіцієнт силуету – це середня величина силуету об'єктів даної вибірки. Даний коефіцієнт показує, наскільки середня відстань до об'єктів свого кластеру відрізняється від середньої відстані до об'єктів інших кластерів. Дана величина лежить в діапазоні [-1,1]. Значення які близькі до -1, відповідають поганій кластеризації, значення які близькі до нуля говорять про те, що кластери перетинаються і накладаються один на одного. Значення, які близькі 1, відповідають "щільнісним" чітко виділеним кластерам. Таким чином, чим більший силует, тим більш чітко виділені кластери з вибірки, і вони представляють собою компактні, чітко згруповані області точок.

– Індекс Калінскі-Харабаза – дозволяє виміряти схожість об'єктів в групі і відмінності між групами. Показує співвідношення дисперсії між кластерами та всередині кластера.

– Індекс Девіда – Болдуїна – визначає середню схожість між кластером і найбільш близьким до нього кластером. Оскільки, мається на увазі, що

кластери в структурі значно відрізняються один від одного, найкращою буде структура з мінімальним показником цього індексу.

Першим експериментом було здійснено кластеризацію вибірки даних в якій відображалась тільки нормальна робота мережі. Отримані наступні результати:

Таблиця 3.2 – Результати експериментів проведених над даними датасету KDD-99

	Кількість потенційних мікрокластерів	Кількість створених кластерів	Кількість мікрокластерів, що віднесено до шуму	Коефіцієнт силуета	Індекс Калінскі-Харабаза	Індекс Девіда – Болдуїна
Нормальна поведінка	26	1	10	0.651	60.760	0.485
Атака Neptune	51	3	6	0.680	28.526	1.524
Атака Nmap	50	3	3	0.737	188.297	0.320
Атака Satan	52	3	7	0.679	37.739	1.357
Атака portsweep	53	4	6	0.659	29.887	1.820
Атака Smurf, Portsweep	48	2	5	0.543	130.858	0.450
Атака Smurf, Portsweep, Neptune	47	2	3	0.667	93.668	1.490

Після перевірки роботи алгоритму з даними без аномальної поведінки, здійснювалися експерименти для даних в яких окрім нормальної поведінки мережі були присутні і аномальні дані, які відносилися до різних типів атак.

А саме атаки Neptune, Nmap, Satan, portsweep та поєднання цих атак.

Висновки до розділу 3

Отже, проаналізувавши результати кластеризації обраного потокового алгоритму кластеризації даних DenStream та дані метрик, що використовуються для оцінювання якості кластеризації можна зробити висновок, що даний алгоритм успішно виділяє аномальну поведінку в роботі комп'ютерної мережі. Також він спроможний вдало виявляти аномальну поведінку для різних видів атак. Даний алгоритм кластеризації доречно використовувати в системах виявлення аномалій. Оскільки він дозволить з високою точністю адміністратору комп'ютерної мережі швидко виявити моменти, коли в мережі була присутня аномальна поведінка, тобто вона була під впливом мережевої атаки. Адміністратор в подальшому може самостійно проаналізувати на які показники мережі дана атака вплинула та здійснити відповідні дії для захисту мережі від даного типу атаки.

ВИСНОВКИ

В даній роботі було досліджено існуючі методи аналізу мережевого трафіку, що використовуються в системах запобіганню вторгнень. Досліджено які існують методи виявлення аномалій та зловживань комп'ютерній мережі. Проаналізовано методи потокової кластеризації, які можливо використовувати в задачах захисту мереж. Обрано двокроковий метод кластеризації DenStream, що поєднує динамічну мікрокластеризацію та статичне використання алгоритму DBSCAN та здатен працювати в умовах, коли вхідні дані потокові. На основі цього алгоритму створено програмний продукт, за допомогою якого було проаналізовано можливість даного методу виділяти групи даних (кластери), які включають в себе аномальну поведінку мережі на основі реальних даних про стан мережі KDD-99. Було виявлено його основні обмеження та можливість виділення аномальних даних в інший кластер.

Подальший розвиток виконаної роботи може полягати, наприклад, в вивченні технічної можливості підключення розробленого модулю до бібліотеки аналізу аномалій системи Splunk Machine Learning Toolkit, а також в вдосконаленні самого методу з точки зору врахування складної динаміки змін кластерів в реальних даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Грайворонський М. В., Новіков О. М. Безпека інформаційно-комунікаційних систем. — К. : Видавнича група ВНУ, 2009. — С. 474–482 [Електронний ресурс] – Режим доступу до ресурсу: http://is.ipt.kpi.ua/wp-content/uploads/sites/4/2015/03/Graivorovskyi_Novikov.pdf.

2. J. Han, M. Kamber, and J. Pei, Data mining: concepts and techniques:Morgan kaufmann, 2006.

3. S. Guha, R. Rastogi, and K. Shim, "CURE: an efficient clustering algorithm for large databases," in ACM SIGMOD Record, 1998, pp. 73-84.

4. S. Guha, R. Rastogi, and K. Shim, "ROCK: A robust clustering algorithm for categorical attributes," Information systems, vol. 25, pp. 345-366, 2000.

5. T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: an efficient data clustering method for very large databases," in ACM SIGMOD Record, 1996, pp. 103-114.

6. P. P. Rodrigues, J. Gama, and J. P. Pedroso, "Hierarchical clustering of timeseries data streams," Knowledge and Data Engineering, IEEE Transactions on, vol. 20, pp. 615-627, 2008.

7. K. Udommanetanakit, T. Rakthanmanon, and K. Waiyamai, "E-stream: Evolution-based technique for stream clustering," in Advanced Data Mining and Applications, ed: Springer, 2007, pp. 605-615.

8. W. Meesuksabai, T. Kangkachit, and K. Waiyamai, "HUE-Stream: evolution based clustering technique for heterogeneous data streams with uncertainty," in Advanced Data Mining and Applications, ed: Springer, 2011, pp. 27-40.\

9. A. Zhou, F. Cao, W. Qian, and C. Jin, "Tracking clusters in evolving data streams over sliding windows," Knowledge and Information Systems, vol. 15, pp. 181-214, 2008.

10. C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for projected clustering of high dimensional data streams," in Proceedings of the

Thirtieth international conference on Very large data bases-Volume 30, 2004, pp. 852-863.

11. M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler, "StreamKM++: A clustering algorithm for data streams," Journal of Experimental Algorithmics (JEA), vol. 17, p. 2.4, 2012.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Клас Microcluster

```
import numpy as np
class MicroCluster:
    def __init__(self,lambd,creation_time):
        self.lambd = lambd
        self.decay_factor = 2 ** (-lambd)
        self.mean = 0
        self.variance = 0
        self.sum_of_weights = 0
        self.creation_time = creation_time

    def insert_sample(self,sample,weight):
        if self.sum_of_weights != 0:
            # Update sum of weights
            old_sum_of_weights = self.sum_of_weights
            new_sum_of_weights = old_sum_of_weights * self.decay_factor + weight

            # Update mean
            old_mean = self.mean
            new_mean = old_mean + (weight / new_sum_of_weights) * (sample - old_mean)

            # Update variance
            old_variance = self.variance
            new_variance = old_variance * ((new_sum_of_weights - weight) / old_sum_of_weights)+ weight * (sample - new_mean) * (sample - old_mean)

            self.mean = new_mean
            self.variance = new_variance
            self.sum_of_weights = new_sum_of_weights
        else:
            self.mean = sample
            self.sum_of_weights = weight


    def radius(self):
        if self.sum_of_weights > 0:
            return np.linalg.norm(np.sqrt(self.variance / self.sum_of_weights))
        else:
            return float('nan')

    def center(self):
        return self.mean

    def weight(self):
        return self.sum_of_weights

    def __copy__(self):
```

```
        new_micro_cluster = MicroCluster(self.lambd, self.creation_time)
        new_micro_cluster.sum_of_weights = self.sum_of_weights
        new_micro_cluster.variance = self.variance
        new_micro_cluster.mean = self.mean
        return new_micro_cluster
```

Клас DenStream

```
import sys
import numpy as np
from sklearn.utils import check_array
from copy import copy
from MicroCluster import MicroCluster
from math import ceil
from sklearn.cluster import DBSCAN


class DenStream:

    def __init__(self, lambd=1, eps=1, beta=2, mu=2):
        self.lambd = lambd
        self.eps = eps
        self.beta = beta
        self.mu = mu
        self.t = 0
        self.p_micro_clusters = []
        self.o_micro_clusters = []
        if lambd > 0:
            self.tp = ceil((1 / lambd) * np.log((beta * mu) / (beta * mu - 1)))
        else:
            self.tp = sys.maxsize

    def partial_fit(self, X, y=None, sample_weight=None):
        n_samples, _ = np.shape(X)

        sample_weight = self._validate_sample_weight(sample_weight, n_samples)

        for sample, weight in zip(X, sample_weight):
            self._partial_fit(sample, weight)
        return self


    def fit_predict(self, X, y=None, sample_weight=None):

        n_samples, _ = X.shape

        sample_weight = self._validate_sample_weight(sample_weight, n_samples)

        for sample, weight in zip(X, sample_weight):
            self._partial_fit(sample, weight)
```

```
    p_micro_cluster_centers = np.array([p_micro_cluster.center() for
                                p_micro_cluster in
                                self.p_micro_clusters])
    p_micro_cluster_weights = [p_micro_cluster.weight() for p_micro_cluster in
                        self.p_micro_clusters]
    dbscan = DBSCAN(eps=0.3, algorithm='brute')
    dbscan.fit(p_micro_cluster_centers,
            sample_weight=p_micro_cluster_weights)
    y = []
    for sample in X:
        index, _ = self._get_nearest_micro_cluster(sample,
                                        self.p_micro_clusters)
        y.append(dbscan.labels_[index])

    return y

def _get_nearest_micro_cluster(self, sample, micro_clusters):
    smallest_distance = sys.float_info.max
    nearest_micro_cluster = None
    nearest_micro_cluster_index = -1
    for i, micro_cluster in enumerate(micro_clusters):
        current_distance = np.linalg.norm(micro_cluster.center() - sample)
        if current_distance < smallest_distance:
            smallest_distance = current_distance
            nearest_micro_cluster = micro_cluster
            nearest_micro_cluster_index = i
    return nearest_micro_cluster_index, nearest_micro_cluster

def _try_merge(self, sample, weight, micro_cluster):
    if micro_cluster is not None:
        micro_cluster_copy = copy(micro_cluster)
        micro_cluster_copy.insert_sample(sample, weight)
        if micro_cluster_copy.radius() <= self.eps:
            micro_cluster.insert_sample(sample, weight)
            return True
    return False

def _merging(self, sample, weight):
    # Try to merge the sample with its nearest p_micro_cluster
    _, nearest_p_micro_cluster = \
        self._get_nearest_micro_cluster(sample, self.p_micro_clusters)
    success = self._try_merge(sample, weight, nearest_p_micro_cluster)
    if not success:
        # Try to merge the sample into its nearest o_micro_cluster
        index, nearest_o_micro_cluster = \
            self._get_nearest_micro_cluster(sample, self.o_micro_clusters)
        success = self._try_merge(sample, weight, nearest_o_micro_cluster)
        if success:
            if nearest_o_micro_cluster.weight() > self.beta * self.mu:
                del self.o_micro_clusters[index]
                self.p_micro_clusters.append(nearest_o_micro_cluster)
        else:
```

```
            # Create new o_micro_cluster
            micro_cluster = MicroCluster(self.lambd, self.t)
            micro_cluster.insert_sample(sample, weight)
            self.o_micro_clusters.append(micro_cluster)

    def _decay_function(self, t):
        return 2 ** ((-self.lambd) * (t))

    def _partial_fit(self, sample, weight):
        self._merging(sample, weight)
        if self.t % self.tp == 0:
            self.p_micro_clusters = [p_micro_cluster for p_micro_cluster
                                     in self.p_micro_clusters if
                                     p_micro_cluster.weight() >= self.beta *
                                     self.mu]
            Xis = [((self._decay_function(self.t - o_micro_cluster.creation_time
                                          + self.tp) - 1) /
                    (self._decay_function(self.tp) - 1)) for o_micro_cluster in
                   self.o_micro_clusters]
            self.o_micro_clusters = [o_micro_cluster for Xi, o_micro_cluster in
                                     zip(Xis, self.o_micro_clusters) if
                                     o_micro_cluster.weight() >= Xi]
        self.t += 1

    def _validate_sample_weight(self, sample_weight, n_samples):
        """Set the sample weight array."""
        if sample_weight is None:
            # uniform sample weights
            sample_weight = np.ones(n_samples, dtype=np.float64, order='C')
        else:
            # user-provided array
            sample_weight = np.asarray(sample_weight, dtype=np.float64,
                                       order="C")
        if sample_weight.shape[0] != n_samples:
            raise ValueError("Shapes of X and sample_weight do not match.")
        return sample_weight
```

Демонстрація роботи demonstration.py

```
from DenStream import DenStream
import csv
import matplotlib.pyplot as plt
import math
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
import numpy as np
from sklearn import metrics
from numpy import array


put = input("Please put a path to a file:  ")
```

```
f = open(put, "r")
lines = f.read().split("\n")

data=[]
for line in lines:
    if line != "":
        cols = line.split(";")
        mm=list(map(int,cols))
        data.append(mm)
data = array(data)
clusterer = DenStream(lambd=0.1, eps=50, beta=0.5, mu=3)
p_micro_cluster_centers =[]
p_micro_cluster_weights = []
dlina_p_micro_clusters = []
tt = []
for row in data:
    clusterer.partial_fit([row], None)
    print(f"Number of o_micro_clusters is {len(clusterer.o_micro_clusters)}")
    print(f"Number of p_micro_clusters is {len(clusterer.p_micro_clusters)}")
    dlina_p_micro_clusters.append(len(clusterer.p_micro_clusters))


for ii in range(len(dlina_p_micro_clusters)):
    tt.append(ii+1)
    ii += 1

for i in range(len(clusterer.p_micro_clusters)):
    p_micro_cluster_centers.append(clusterer.p_micro_clusters[i].center())
    p_micro_cluster_weights.append(clusterer.p_micro_clusters[i].weight())
    i += 1

p_micro_cluster_centers = array(p_micro_cluster_centers)
p_micro_cluster_weights = array(p_micro_cluster_weights)
p_micro_cluster_weights = p_micro_cluster_weights.reshape(-1,1)
p_micro_cluster_weights = StandardScaler().fit_transform(p_micro_cluster_weights)
np.seterr(divide='ignore', invalid='ignore')
db = DBSCAN(eps=0.3, min_samples=5,algorithm='brute')
db.fit(p_micro_cluster_weights,sample_weight=None)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
print(labels)
x = tt
y = dlina_p_micro_clusters
plt.scatter(x,y,label = "some dots",color = 'k')
plt.xlabel('Number of microclusters')
plt.ylabel('Tp')
plt.grid(True)
plt.savefig('data.png', fmt = 'png')
plt.legend()
plt.show()
```