МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

ГАЛИН Василь Андрійович

Методи виявлення аномалій в наборах великих даних на основі машинного навчання / Anomaly Detection Methods in Big Data Sets Based on Machine Learning

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КНм-21
В. А. Галин
_____

Науковий керівник:
к.т.н., доцент П.Є.Биковий
_____

Кваліфікаційну роботу
допущено до захисту:
«___» _____ 20___ р.
В.о. завідувача кафедри
_____ Н.В. Дзюбановська

ТЕРНОПІЛЬ – 2025

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «магістр»
спеціальність: 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.М. Васильків
«____» _____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ГАЛИНУ Василю Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи

Методи виявлення аномалій в наборах великих даних на основі машинного навчання / Anomaly Detection Methods in Big Data Sets Based on Machine Learning

керівник роботи к.т.н., доцент П.Є. Биковий
затверджені наказом по університету від 20 грудня 2024 року № 938.
2. Строк подання студентом закінченої кваліфікаційної роботи 1 грудня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити

– проаналізувати сучасний стан проблеми виявлення аномалій, існуючі класифікації методів та їхні обмеження при застосуванні до великих даних;

– дослідити парадигму розподілених обчислень MapReduce та можливості фреймворку Apache Spark для реалізації масштабованих алгоритмів машинного навчання;

– розробити та описати алгоритми виявлення аномалій, адаптувавши їхні обчислювальні процедури до виконання у розподіленому середовищі;

– програмно реалізувати розроблені алгоритми на платформі Apache Spark у вигляді єдиного програмного пакета;

– провести експериментальну оцінку продуктивності розроблених алгоритмів на реальному наборі даних та еталонних наборах даних;

– дослідити ефективність та масштабованість запропонованих рішень шляхом аналізу часу обчислень залежно від обсягу даних, кількості обчислювальних потоків та вузлів кластера;

– проаналізувати вплив ключових гіперпараметрів алгоритмів на якість виявлення аномалій та сформулювати практичні рекомендації щодо їх налаштування.

5. Перелік графічного матеріалу у роботі

– ілюстрація аномалій у двовимірному наборі даних;
– графіки продуктивності розподілених детекторів аномалій.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи.	до 01.01. 2025 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.03. 2025 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 20.05.2025 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 28.10. 2025 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 11.11.2025 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів.	до 25.11.2025 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту.	до 1.12.2025 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 04.12.2025 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії.	до 14.12. 2025 р.	

Студент _____ В.А. Галин
підпис

Керівник роботи _____ к.т.н., доцент П.Є. Биковий
підпис

РЕЗЮМЕ

Кваліфікаційна робота на тему «Методи виявлення аномалій в наборах великих даних на основі машинного навчання» на здобуття освітнього ступеня «Магістр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом в 88 сторінок і містить 1 ілюстрацію, 2 таблиці, 2 додатки та 53 використаних джерел.

Метою кваліфікаційної роботи є підвищення ефективності процесу виявлення аномалій у середовищі великих даних шляхом розробки, реалізації та експериментального дослідження набору розподілених алгоритмів неконтрольованого аналізу.

Методи досліджень: системного аналізу для вивчення предметної області, а також методи формалізації та асимптотичного аналізу для проєктування нових розподілених алгоритмів та оцінки їхньої обчислювальної складності, методи великих даних, математичної статистики.

Результати дослідження: удосконаленно методи виявлення аномалій шляхом їхньої фундаментальної архітектурної адаптації до парадигми розподілених обчислень, що дозволило перетворити їх з послідовних, не масштабованих інструментів на повноцінні, ефективні та готові до промислового використання рішення для виявлення аномалій у великих даних.

Результати роботи можуть успішно застосовуватися для виявлення аномалій у середовищі великих даних, та може бути безпосередньо впроваджений у промислові та наукові процеси.

Ключові слова: ВИЯВЛЕННЯ АНОМАЛІЙ; ВЕЛИКІ ДАНІ; НЕКОНТРОЛЬОВАНЕ НАВЧАННЯ; МАШИННЕ НАВЧАННЯ; РОЗПОДІЛЕНІ АЛГОРИТМИ; MAPREDUCE; APACHE SPARK; АНСАМБЛЕВІ МЕТОДИ.

ABSTRACT

Qualification work on the topic «Anomaly Detection Methods in Big Data Sets Based on Machine Learning» for Master's degree on speciality 122 «Computer Science» educational and professional program «Computer Science» is written on 88 pages and it contains 1 figure, 2 tables, 2 annexes and 53 sources.

The purpose of this qualification work is to improve the efficiency of the anomaly detection process in big data environments by developing, implementing, and experimentally studying a set of distributed algorithms for unsupervised analysis.

Research methods: systems analysis to study the subject area, as well as methods of formalization and asymptotic analysis for designing new distributed algorithms and assessing their computational complexity, big data methods, and methods of mathematical statistics.

Research results: the methods of anomaly detection have been improved through their fundamental architectural adaptation to the paradigm of distributed computing, which has made it possible to transform them from sequential, non-scalable tools into full-fledged, efficient and industry-ready solutions for anomaly detection in big data.

The results of the work can be successfully applied to anomaly detection in big data environments and can be directly implemented in industrial and scientific processes.

Keywords: ANOMALY DETECTION; BIG DATA; UNSUPERVISED LEARNING; MACHINE LEARNING; DISTRIBUTED ALGORITHMS; MAPREDUCE; APACHE SPARK; ENSEMBLE METHODS.

ЗМІСТ

ВСТУП

Актуальність теми. Виявлення аномалій стосується задачі ідентифікації спостережень, що суттєво відрізняються від решти даних. Ці неочікувані патерни зазвичай називають викидами або аномаліями [10, 17]. У більшості випадків дані генеруються одним або кількома процесами, які відображають поведінку системи. Коли така система функціонує некоректно, вона продукує аномалії чи викиди. Ідентифікація таких аномальних спостережень має вирішальне значення для виявлення нетипової поведінки системи [1, 12, 36]. Виявлення аномалій знаходить застосування у широкому спектрі галузей, таких як детекція фінансового шахрайства [28], виявлення вторгнень [30], сенсорні мережі [32], промислові аномалії [31] чи сфера охорони здоров'я [46].

Існує три різні типи завдань у сфері виявлення аномалій [10]:

1. Контрольоване виявлення аномалій. Набір даних є розміченим, вказуючи, які екземпляри є нормальними, а які – аномальними. На цих даних будується прогностична модель для розділення нормальних та аномальних екземплярів.

2. Частково контрольоване виявлення аномалій. Навчальний набір, що використовується, не містить аномальних екземплярів, а лише нормальні спостереження. Аномальні екземпляри надаються в тестовому наборі.

3. Неконтрольоване виявлення аномалій. Екземпляри не є розміченими. Аномальні спостереження невідомі, й алгоритм повинен бути здатним виявляти їх без попередніх знань.

Через автоматизацію збору та зберігання даних більшість реальних завдань з виявлення аномалій належать до неконтрольованого типу. Сценарій неконтрольованого виявлення аномалій є особливо складним, оскільки підходи машинного навчання не мають попередніх знань про дані [7]. Алгоритми неконтрольованого виявлення аномалій оцінюють дані, базуючись виключно на властивостях самого набору даних. Ця оцінка відображає ступінь «аномальності» кожного екземпляра. Потім, використовуючи порогове значення або фіксовану кількість, обираються аномалії [34]. Алгоритми

неконтрольованого виявлення аномалій можна згрупувати в чотири категорії [1]: методи на основі найближчих сусідів [6], методи на основі кластеризації [26], статистичні методи [40] та ансамблі [48]. Нещодавно було запропоновано інструментарій для виявлення викидів під назвою PyOD [49]. Ця бібліотека надає широкий спектр алгоритмів виявлення аномалій, що включає як добре відомі методи, так і новітні підходи.

Удосконалення технологій, а також поява нових, таких як смартфони, зв'язок 5G, сенсори, хмарні обчислення, віртуальна реальність та додатки для «розумного дому», призводять до стрімкого генерування величезного обсягу даних. Зростання кількості генерованих даних призвело до епохи великих даних (Big Data) [45]. Великі дані – це дані великого обсягу, високої швидкості та значної різноманітності, які неможливо обробити традиційними методами. Це створило необхідність у розробці специфічних методів для різних типів даних, що можуть надходити [23, 35, 41]. Автоматизація збору даних, популяризація сенсорів та відсутність людського нагляду, що є характерними для великих даних, підвищили потребу в ефективних методах виявлення аномалій. Природа великих даних у більшості випадків унеможливлює їх розмітку експертом. Ця проблематика призводить до того, що більшість реальних завдань з виявлення аномалій у великих даних є неконтрольованими. Незважаючи на наявність популярних бібліотек, таких як PyOD [49], для завдань з виявлення аномалій звичайного розміру, у сфері великих даних можна знайти лише декілька пропозицій, присвячених специфічним доменам цієї проблеми [8, 23, 37, 45].

Мета і завдання дослідження. Метою кваліфікаційної роботи є підвищення ефективності процесу виявлення аномалій у середовищі великих даних шляхом розробки, реалізації та експериментального дослідження набору розподілених алгоритмів неконтрольованого аналізу.

Для досягнення поставленої мети необхідно вирішити наступні **завдання**:

– проаналізувати сучасний стан проблеми виявлення аномалій, існуючі класифікації методів та їхні обмеження при застосуванні до великих даних;

– дослідити парадигму розподілених обчислень MapReduce та

можливості фреймворку Apache Spark для реалізації масштабованих алгоритмів машинного навчання;

– розробити та описати архітектури розподілених алгоритмів виявлення аномалій, адаптувавши їхні обчислювальні процедури до виконання у розподіленому середовищі;

– програмно реалізувати розроблені алгоритми на платформі Apache Spark у вигляді єдиного програмного пакета;

– провести експериментальну оцінку продуктивності розроблених алгоритмів на реальному наборі даних та еталонних наборах даних стандартного розміру, використовуючи метрику ROC-AUC;

– дослідити ефективність та масштабованість запропонованих рішень шляхом аналізу часу обчислень залежно від обсягу даних, кількості обчислювальних потоків та вузлів кластера;

– проаналізувати вплив ключових гіперпараметрів алгоритмів на якість виявлення аномалій та сформулювати практичні рекомендації щодо їх налаштування.

Об’єкт дослідження – процес неконтрольованого виявлення аномалій у великомасштабних наборах даних.

Предмет дослідження – розподілені алгоритми виявлення аномалій на основі гістограм, ансамблевих та гібридних підходів, їхня обчислювальна складність, масштабованість та ефективність при реалізації в середовищі Apache Spark.

У роботі використовуються наступні **методи досліджень**: на теоретичному рівні було застосовано методи системного аналізу для вивчення предметної області, а також методи формалізації та асимптотичного аналізу для проєктування архітектур нових розподілених алгоритмів та оцінки їхньої обчислювальної складності. Практична частина дослідження включала програмну реалізацію розроблених методів на платформі Apache Spark та проведення серії обчислювальних експериментів на високопродуктивному кластері для оцінки їхньої якості та масштабованості. Отримані експериментальні дані були оброблені методами математичної статистики, що

дозволило об’єктивно порівняти продуктивність алгоритмів, підтвердити їхню перевагу над класичними аналогами та сформулювати обґрунтовані висновки щодо їхньої практичної придатності для аналізу великих даних.

Наукова новизна одержаних результатів полягає у удосконаленні методів виявлення аномалій шляхом їхньої фундаментальної архітектурної адаптації до парадигми розподілених обчислень, що дозволило перетворити їх з послідовних, не масштабованих інструментів на повноцінні, ефективні та готові до промислового використання рішення для виявлення аномалій у великих даних.

Практичне значення отриманих результатів полягає у створенні комплексного інструментарію, що вирішує актуальну задачу виявлення аномалій у середовищі великих даних, та може бути безпосередньо впроваджений у промислові та наукові процеси.

Публікації та апробація КР. Результати кваліфікаційної роботи апробовані та опубліковані у матеріалах (додаток А):

– 2nd International Scientific and Practical Conference «Progressive Approaches in Science and Engineering», November 26-28, 2025. Copenhagen, Denmark;

– ІІ Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Інтелектуальні комп’ютерні системи та мережі», 25 листопада 2025 р., Тернопіль, Україна.

Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВИЯВЛЕННЯ АНОМАЛІЙ У ВЕЛИКИХ ДАНИХ

1.1 Аналіз предметної області виявлення аномалій

Аномалія – це спостереження, що суттєво відрізняється від інших. На рисунку 1.1 наведено графічне представлення аномалій у двовимірному наборі даних. Кластери C1 та C2 складаються з нормальних спостережень, оскільки більшість точок належить до цих областей. Спостереження O1, O2 та кластер C3 розташовані в областях, що знаходяться на значній відстані від C1 та C2, а отже, вони вважаються аномаліями [10].

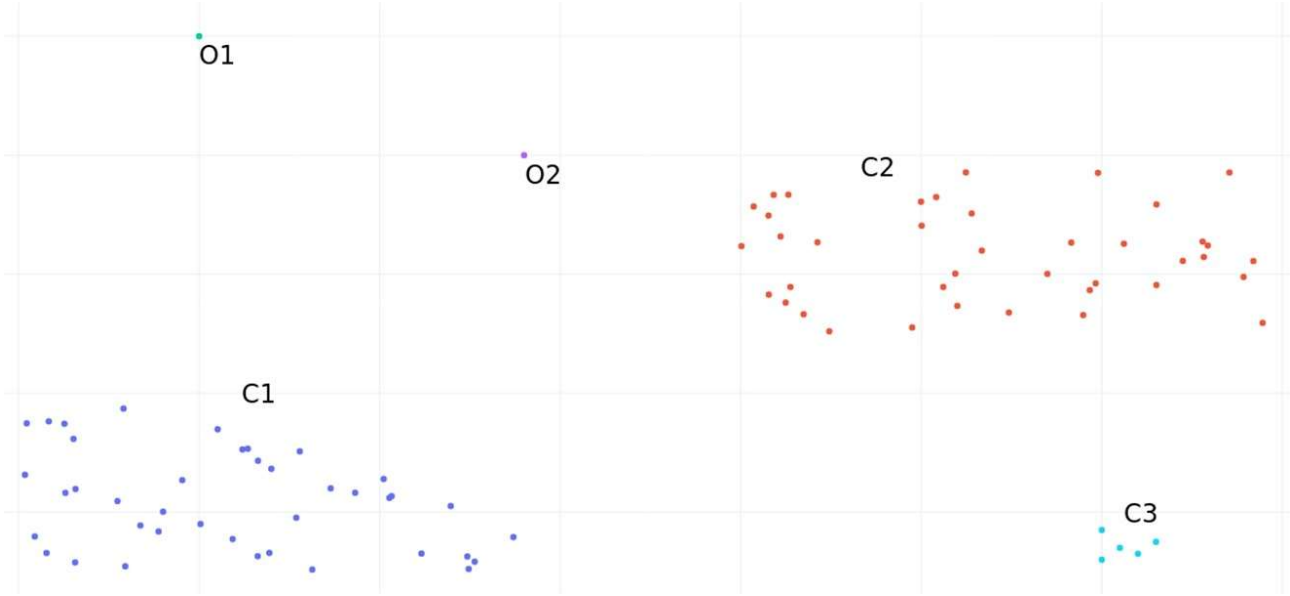


Рисунок 1.1 – Ілюстрація аномалій у двовимірному наборі даних

Існує три основні типи аномалій [22]:

1. Точкова аномалія. Аномальні екземпляри є ізольованими. На рисунку 1.1 O1 та O2 є точковими аномаліями. Це найпоширеніший сценарій у виявленні аномалій.

2. Колективна аномалія. Аномалія є комбінацією кількох пов’язаних між собою аномальних екземплярів. Наприклад, виявлення вторгнення в мережеву систему може включати детекцію множинних спроб з’єднання.

3. Контекстуальна аномалія. Екземпляр даних може здаватися стандартним, оскільки не має аномальних значень, проте в межах певного

контексту це значення може бути аномалією. Наприклад, якщо ми вимірюємо заповненість автобуса в діапазоні від 0% до 100% протягом дня, значення 50% здається абсолютно нормальним. Однак, якщо це значення зафіксовано о 08:00 ранку, коли люди їдуть на роботу чи навчання, очікувана заповненість мала б бути значно вищою.

Аномалії не слід плутати з шумом, хоча ці поняття є пов'язаними. Шум демонструє поведінку, аналогічну до зображеної на рисунку 1.1, але, на відміну від аномалій, він не становить інтересу для аналітика даних. Аномалії є цінною інформацією, яку необхідно виявляти, виокремлювати та аналізувати. Шум погіршує якість даних, і такі спостереження повинні бути або виправлені, або видалені [14, 19, 20, 35].

Виявлення аномалій стосується проблеми пошуку патернів, що значно відрізняються від стандартних спостережень. Виявлення аномалій використовується у широкому спектрі галузей, таких як детекція шахрайства з кредитними картками [28], виявлення вторгнень [30], сенсорні мережі [32], промислові аномалії [31], охорона здоров'я [15] та багато інших [22]. Через різноманітність доменів, що охоплюють проблему виявлення аномалій, та все більшу присутність сенсорів у всіх сферах, ця галузь сьогодні привертає дедалі більше уваги.

Вихідні дані алгоритму виявлення аномалій можуть бути двох типів [10]:

1. Оцінки. Алгоритм повертає оцінку аномальності для кожного екземпляра в тестовому наборі даних, вказуючи на ті екземпляри, які є найбільш імовірно аномальними. Необхідно визначити стратегію для вибору того, які оцінки вважати аномаліями.

2. Мітки. Алгоритм повертає бінарну мітку, що вказує, які екземпляри є аномаліями, а які – нормальними.

У літературі можна знайти багато різних технік виявлення аномалій, залежно від підходу, який вони використовують, їх можна класифікувати на [1, 10]:

1. Аналіз екстремальних значень. Це найпростіша форма виявлення аномалій, що базується на аналізі одновимірних даних. Ці методи припускають,

що значення є аномальним, якщо воно є або занадто великим, або занадто малим.

2. Імовірнісні та статистичні моделі. Дані моделюються як імовірнісний розподіл. Екземпляри, що знаходяться в областях високої ймовірності, вважаються нормальними, а аномальні екземпляри – в областях низької ймовірності [21, 40].

3. Методи на основі класифікації. Ці техніки використовують розмічений навчальний набір даних для побудови моделі, після чого тестовий набір класифікується за допомогою навченої моделі з присвоєнням мітки кожному екземпляру. Прикладами таких технік є методи на основі глибокого навчання, баєсівських мереж [44], машин опорних векторів [16] та методи на основі правил [27].

4. Методи на основі найближчих сусідів. Вони припускають, що екземпляри в межах сусідства з високою щільністю є нормальними, а екземпляри, що знаходяться далеко від таких сусідств, є аномаліями. В рамках цих технік існують два основні підходи: методи на основі відстані та моделі на основі щільності [6].

5. Методи на основі кластеризації. Припускають, що екземпляри всередині кластера є нормальними. З іншого боку, екземпляри, що не належать до жодного кластера, є аномальними. Основна відмінність від технік на основі найближчих сусідів полягає в тому, що методи кластеризації оцінюють кожен екземпляр відповідно до кластера, до якого він належить, тоді як техніки найближчих сусідів аналізують кожен екземпляр у його локальному сусідстві [26].

6. Ансамблеві методи. Полягають у комбінації кількох різноманітних базових алгоритмів навчання для отримання глобальної моделі, що перевершує базові детектори [40, 47, 48].

Існує безліч застосувань у сфері виявлення аномалій. Виявлення вторгнень полягає у детекції аномальної активності в комп’ютерній мережі [30]. Ця проблема характеризується великим обсягом інформаційного потоку, що може призводити до високої частоти хибних спрацьовувань. Виявлення шахрайства стосується пошуку незвичних операцій, пов’язаних зі злочинами, у комерційних

застосунках, таких як кредитні картки, телекомунікаційні компанії, банки тощо [28]. Ці незвичні операції пов’язані з крадіжкою особистих даних або спробами шахрайства з боку споживача. Інші застосування, не пов’язані з шахрайством, такі як медична сфера, полягають у виявленні аномалій у вимірюваннях пацієнтів, які можуть бути спричинені захворюванням пацієнта, помилками приладів або помилками запису. Також виявлення аномалій може застосовуватися у промисловості для детекції неочікуваної поведінки двигунів на складальній лінії, помилок датчиків двигунів або пошкоджень у конструкціях [31]. Інші сфери застосування, що становлять інтерес, – це обробка зображень, виявлення аномалій у текстових даних або виявлення аномалій у сенсорних мережах, що стосується детекції аномалій у зібраних даних, які можуть свідчити про вторгнення або помилки в роботі сенсорів [32].

1.2 Великі дані та MapReduce

Парадигма MapReduce є на сьогодні найбільш популярною та поширеною для обробки великих даних. Вона дозволяє обробляти та генерувати значні обсяги даних у розподілений та ефективний спосіб, а також мінімізує дискові операції та мережеве навантаження [13].

Ця парадигма складається з двох фаз: фази Map та фази Reduce. Спочатку головний вузол здійснює партиціонування даних та розподіляє їх поміж вузлами кластера. Функція Map застосовує операцію перетворення до локальних пар «ключ-значення» на кожному обчислювальному вузлі. Іншими словами, кожен обчислювальний вузол обробляє певну частину алгоритму на певному піднаборі даних. Після завершення фази Map усі пари з однаковим ключем перерозподіляються. Коли всі пари з однаковим ключем опиняються на одному обчислювальному вузлі, розпочинається фаза Reduce. Фаза Reduce є операцією агрегації, яка об’єднує всі результати з різних фаз Map у фінальні значення.

Apache Spark [29] – це фреймворк з відкритим вихідним кодом для обробки великих даних, орієнтований на швидкість, простоту використання та складну аналітику. Spark дозволяє зберігати дані в оперативній пам’яті для послідовної

або ітеративної обробки, що значно підвищує продуктивність. В основі Spark лежать відмовостійкі розподілені набори даних (Resilient Distributed Datasets, RDDs), які за своєю природою є невпорядкованими та незмінними. Вони дозволяють зберігати їх в пам’яті, а їхня історія відстежується за допомогою "родоводу", що дає змогу переобчислити кожну партицію у випадку збою.

RDD підтримують два типи операцій:

1) перетворення – обчислюються відкладено та створюють новий RDD;

2) дії – ініціюють виконання всіх попередніх перетворень та повертають кінцевий результат.

1.3 Аналіз проблеми неконтрольованого виявлення аномалій

1.3.1 Неконтрольоване виявлення аномалій

Проблема неконтрольованого виявлення аномалій стосується особливого типу завдань у сфері виявлення аномалій, в яких екземпляри набору даних не є розміченими. Ця характеристика робить процес пошуку аномалій значно складнішим, оскільки відсутня еталонна розмітка (ground-truth), а алгоритми не мають апріорних знань про те, що являє собою аномалія. Існує чотири різні типи алгоритмів для розв’язання задачі неконтрольованого виявлення аномалій [10].

1. Методи на основі найближчих сусідів: викиди визначаються за їхніми відстанями або щільністю відносно найближчих сусідів/регіонів [6].

2. Методи на основі кластеризації: центроїд обчислюється за допомогою алгоритму кластеризації, а викиди виявляються завдяки тому, що вони мають велику відстань до щільних областей [26].

3. Статистичні методи: базуються виключно на властивостях даних, таких як їх розсіювання або гістограми [21, 40].

4. Ансамблеві методи: різноманітні базові методи виявлення аномалій та стратегії їх комбінування використовуються для створення більш комплексної моделі [47, 48].

Тестування ефективності алгоритму неконтрольованого виявлення аномалій є дуже складним завданням, оскільки відсутня еталонна розмітка

даних. Більшість алгоритмів виявлення аномалій визначають, чи є спостереження аномальним, застосовуючи порогове значення до кожної оцінки, отриманої від алгоритму. Складність застосування порогу полягає в тому, що якщо він занадто високий, це генеруватиме велику кількість хибнопозитивних спрацювань (false positives), оскільки нормальні спостереження будуть класифіковані як аномальні. З іншого боку, якщо поріг занадto низький, деякі аномальні спостереження будуть проігноровані, що призведе до більшої кількості хибнонегативних спрацювань (false negatives). Метриками, що використовуються для перевірки ефективності алгоритму, є точність ($precision$ – відсоток виявлених алгоритмом аномалій, які є реальними аномаліями) та повнота ($recall$ – відсоток еталонних аномалій, які були виявлені як аномалії). З цими метриками пов'язана ROC-крива (Receiver Operating Characteristic curve), яка по осі X відображає частку істинно-позитивних спрацювань (true positive rate, або повнота), а по осі Y – частку хибно-позитивних спрацювань (false positive rate – відсоток помилково виявлених аномалій серед еталонних нормальних спостережень) [1]. Найбільш вживаною метрикою для оцінки продуктивності методів неконтрольованого виявлення аномалій є ROC-AUC (Area Under Curve – площа під кривою) [25].

1.3.2 Виявлення аномалій у великих даних

Внаслідок вибухового зростання обсягів даних, поширення сенсорів та автоматизації процесів збору й зберігання інформації, проблема виявлення аномалій трансформувалася у проблему великих даних. Завдання у таких галузях, як детекція мережевих вторгнень або запобігання відмовам обладнання, вимагають оперативного вирішення з метою уникнення серйозних наслідків. Більше того, через експоненційне зростання даних класичні алгоритми не здатні обробляти їх за прийнятний проміжок часу [23, 35].

Хоча в науковій літературі представлено певні розробки у сфері виявлення аномалій у великих даних, більшість із них базується на схожому кластерному підході [4, 23, 33, 43, 45]. Інші дослідження, як-от запропоноване в [42], виконують різні етапи для відбору ознак та подальшої розмітки даних,

використовуючи такі класифікатори, як SVM, наївний баєсівський класифікатор або випадковий ліс. Метод, запропонований у [37], дотримується неконтрольованого підходу машинного навчання для виявлення аномалій у часових рядах споживання електроенергії. Метод, розроблений у [4], під назвою SSWLOFCC, ґрунтується на композитній кластеризації та технологіях великих даних. У [8] автори проводять поглиблений огляд, зосереджений на застосуванні метаевристик та технік машинного навчання для доменів великих даних. Автори в [5] пропонують методологію глибокого навчання, що складається зі згорткових нейронних мереж та мереж довгої короткочасної пам’яті (Long Short-Term Memory) для виявлення аномалій мережевого трафіку в реальному часі в середовищах Великих даних. У [4] автори доходять висновку, що для вирішення викликів, пов’язаних з Великими даними, необхідно пропонувати модернізовані алгоритми машинного навчання.

Таким чином, існує нагальна потреба в алгоритмах, здатних ефективно обробляти такі значні обсяги даних. Однак на сьогодні не існує загальних методів чи фреймворків, які б допомагали у вирішенні завдань виявлення аномалій у Великих даних. Тому нашою метою є надання розподіленого набору архітектур алгоритмів з різними механізмами роботи для задач виявлення аномалій у Великих даних, які можуть ефективно вирішувати широкий спектр завдань у цій сфері.

1.4 Постановка задачі дослідження

Сучасна епоха характеризується експоненційним зростанням обсягів даних, що генеруються у найрізноманітніших сферах людської діяльності. Розповсюдження технологій Інтернету речей (IoT), сенсорних мереж, мобільних пристроїв та соціальних медіа призвело до виникнення феномену великих даних, які відрізняються не лише величезним обсягом, але й високою швидкістю надходження та значною різноманітністю. Ручна або напівавтоматична обробка таких масивів є неможливою, що зумовлює гостру потребу в розробці ефективних автоматизованих методів аналізу.

Одним із ключових завдань аналізу даних є виявлення аномалій – спостережень, що суттєво відрізняються від загальної маси даних. Своєчасна ідентифікація таких відхилень має критичне значення для багатьох прикладних галузей, зокрема:

– промисловість: запобігання відмовам обладнання шляхом моніторингу показників сенсорів у реальному часі;

– кібербезпека: детекція мережевих вторгнень та шкідливої активності;

– фінансовий сектор: виявлення шахрайських транзакцій;

– охорона здоров'я: діагностика захворювань на основі аномальних показників у медичних даних.

Водночас класичні алгоритми виявлення аномалій, розроблені для роботи з даними стандартного обсягу, виявляються неефективними в середовищі великих даних. Їхня обчислювальна складність та ітераційна природа не дозволяють обробляти терабайтні масиви за прийнятний час. Окрім того, більшість реальних наборів Великих даних є нерозміченими, що унеможливлює застосування традиційних контрольованих методів машинного навчання і висуває на передній план задачі неконтрольованого аналізу.

Таким чином, розробка та дослідження масштабованих, ефективних та розподілених алгоритмів неконтрольованого виявлення аномалій, адаптованих до парадигми Big Data, є надзвичайно актуальною науково-практичною задачею.

Основна наукова проблема полягає у невідповідності між обчислювальними можливостями існуючих (класичних) методів виявлення аномалій та вимогами, що висуваються середовищем великих даних. Ця невідповідність проявляється у кількох аспектах:

– проблема масштабованості: багато ефективних алгоритмів (наприклад, на основі найближчих сусідів) мають високу обчислювальну складність (часто поліноміальну), що робить їх непридатними для обробки мільйонів та мільярдів записів;

– проблема ітераційності: класичні реалізації алгоритмів є послідовними та ітераційними, що унеможливлює їх ефективне розпаралелювання на кластері з десятків чи сотень обчислювальних вузлів;

– проблема відсутності еталонної розмітки: переважна більшість великих даних не має міток "норма/аномалія", що вимагає застосування неконтрольованих підходів, які є менш точними та більш складними в оцінці.

Вирішення цієї проблеми лежить у площині адаптації існуючих та розробки нових алгоритмів з використанням парадигми розподілених обчислень, зокрема MapReduce, та сучасних фреймворків, як-от Apache Spark.

Метою даної кваліфікаційної роботи є підвищення ефективності процесу виявлення аномалій у середовищі великих даних шляхом розробки, реалізації та експериментального дослідження набору розподілених алгоритмів неконтрольованого аналізу.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– проаналізувати сучасний стан проблеми виявлення аномалій, існуючі класифікації методів та їхні обмеження при застосуванні до великих даних;

– дослідити парадигму розподілених обчислень MapReduce та можливості фреймворку Apache Spark для реалізації масштабованих алгоритмів машинного навчання;

– розробити та описати архітектури розподілених алгоритмів виявлення аномалій, адаптувавши їхні обчислювальні процедури до виконання у розподіленому середовищі;

– програмно реалізувати розроблені алгоритми на платформі Apache Spark у вигляді єдиного програмного пакета;

– провести експериментальну оцінку продуктивності розроблених алгоритмів на реальному наборі даних та еталонних наборах даних стандартного розміру, використовуючи метрику ROC-AUC;

– дослідити ефективність та масштабованість запропонованих рішень шляхом аналізу часу обчислень залежно від обсягу даних, кількості обчислювальних потоків та вузлів кластера;

– проаналізувати вплив ключових гіперпараметрів алгоритмів на якість виявлення аномалій та сформулювати практичні рекомендації щодо їх налаштування.

Об’єкт дослідження – процес неконтрольованого виявлення аномалій у великомасштабних наборах даних.

Предмет дослідження – розподілені алгоритми виявлення аномалій на основі гістограм, ансамблевих та гібридних підходів, їхня обчислювальна складність, масштабованість та ефективність при реалізації в середовищі Apache Spark.

Висновки до розділу 1

1. Проведено аналіз концепції виявлення аномалій та її ролі у сучасних інформаційних системах. Це було зроблено шляхом визначення терміну "аномалія", систематизації її типів (точкова, колективна, контекстуальна) та класифікації основних категорій методів їх виявлення (статистичні, кластерні, ансамблеві та ін.).

2. Досліджено специфіку обробки великих даних та обґрунтовано вибір технологічного стеку. Це було реалізовано через аналіз викликів, які ставить феномен Big Data перед класичними алгоритмами (проблеми масштабованості та ітераційності), та розгляд парадигми розподілених обчислень MapReduce і фреймворку Apache Spark як її сучасної реалізації. Було деталізовано ключові примітиви Spark, що є інструментами для побудови розподілених алгоритмів.

3. Систематизовано наукову проблему неконтрольованого виявлення аномалій у середовищі великих даних. Це було досягнуто шляхом поєднання висновків щодо теоретичних обмежень неконтрольованих методів та практичних проблем обробки великих даних. Було встановлено, що наявні рішення є або недостатньо масштабованими, або вузькоспеціалізованими, що створює потребу в універсальних та ефективних розподілених інструментах.

2 МЕТОДИ ДИНАМІЧНОГО ТА СТАТИЧНОГО ВИЯВЛЕННЯ АНОМАЛІЙ У ВЕЛИКИХ ДАНИХ

2.1 Оцінка аномалій у великих даних на основі гістограм

Метод HBOS (Histogram-Based Outlier Score) належить до класу статистичних підходів виявлення аномалій, що базуються на оцінюванні розподілу значень ознак у вибірці [2, 21]. Його ключова ідея полягає у тому, що аномальні спостереження мають значення ознак, які трапляються в даних рідко, а отже відповідають областям простору ознак із низькою емпіричною щільністю. На відміну від методів, що моделюють складні залежності між змінними, HBOS виконує оцінювання для кожної ознаки окремо, використовуючи гістограму як дискретну апроксимацію розподілу.

Нехай задано набір даних

$$X = \{x^{(i)}\}_{i=1}^{n}, \tag{2.1}$$

де $x^{(i)} = x_1^{(i)}, \dots, x_d^{(i)}$ – вектор із d ознак для i-го спостереження.

Для кожної ознаки $j \in \{1, \dots, d\}$ будується гістограма з B інтервалами (бінами), які покривають діапазон значень $[x_j^{min}, x_j^{max}]$. Для кожного біна визначається частота потрапляння елементів вибірки в цей інтервал, після чого частоти нормалізуються до емпіричних оцінок імовірності або щільності. У результаті для ознаки j формується функція $p_j(\cdot)$, яка ставить у відповідність значенню ознаки оцінку “ймовірності” біна, до якого це значення належить.

Оцінка аномальності для спостереження $x^{(i)}$ у HBOS визначається як агрегування внесків усіх ознак. Типовою формою є сума від’ємних логарифмів оцінених імовірностей:

$$HBOS(x^{(i)}) = \sum_{j=1}^{d} -\log\left(p_j\left(x_j^{(i)}\right) + \varepsilon\right), \tag{2.2}$$

де $\varepsilon > 0$ – мале згладжувальне значення, що використовується для уникнення невизначеності при нульових частотах (тобто ситуації, коли деякий інтервал не містить жодного елемента вибірки).

Така логарифмічна форма є зручною, оскільки перетворює добуток незалежних оцінок у суму, а також підсилює вплив дуже малих імовірностей: що

рідше трапляється значення ознаки, то більшим є внесок у загальний бал аномальності.

Концептуально метод HBOS спирається на припущення квазінезалежності ознак, оскільки оцінювання розподілу виконується окремо для кожної змінної, а загальний показник формується шляхом підсумовування внесків. Це дозволяє суттєво зменшити обчислювальну складність порівняно з багатовимірними методами оцінювання щільності та забезпечує високу масштабованість підходу для великих обсягів даних. Після обчислення значень $HBOS(x^{(i)})$ для всіх спостережень формується ранжований список об’єктів, де найбільші значення інтерпретуються як найбільш ймовірні аномалії.

Суттєвим аспектом практичного застосування HBOS є вибір параметрів гістограмування, зокрема кількості бінів і способу їх формування (рівномірні інтервали або інтервали за квантилями). Надто мала кількість бінів призводить до грубої апроксимації розподілу і може знижувати чутливість до локальних відхилень, тоді як надмірна кількість бінів збільшує ризик появи порожніх інтервалів і нестійких оцінок. Тому параметри гістограми мають узгоджуватися з властивостями даних, обсягом вибірки та вимогами до точності виявлення аномалій.

Розглянемо покроковий опис статичного алгоритму HBOS:

1. Підготовка вхідних даних і параметрів. На вхід подається набір даних у форматі Dataset["features"], де кожен екземпляр описано вектором ознак. Додатково задається кількість інтервалів для побудови гістограм і мале згладжувальне значення, необхідне для уникнення проблем, коли в певному інтервалі частота може бути нульовою.

2. Ініціалізація структур для побудови моделі. Створюються порожні структури для збереження меж інтервалів (бінів) та нормованих гістограм для кожної ознаки. Також визначається кількість ознак у векторі features, оскільки далі всі обчислення виконуються окремо для кожної ознаки.

3. Побудова гістограм для кожної ознаки. Для кожної ознаки вибираються її значення з усього набору даних і будується гістограма з заданою кількістю бінів. На цьому етапі зберігаються межі інтервалів, які визначають, у

який бін потрапляє те чи інше значення ознаки. Паралельно обчислюється сума всіх частот у гістограмі та виконується нормалізація частот, щоб отримати відносні частоти (тобто показник “типовості” значень у кожному інтервалі).

4. Обчислення внеску кожної ознаки в оцінку аномальності. Далі алгоритм проходить по кожному екземпляру набору даних. Для кожної ознаки цього екземпляра визначається номер інтервалу, в який потрапляє її значення, після чого береться нормоване значення частоти для відповідного інтервалу. Якщо значення потрапляє в інтервал, що зустрічається рідко, воно вважається менш типовим і дає більший внесок у загальну оцінку аномальності.

5. Агрегування внесків усіх ознак у підсумкову оцінку. Після обчислення внесків для всіх ознак вони підсумовуються та нормуються відповідно до правил, закладених в алгоритмі. У результаті формується одне числове значення – оцінка аномальності для даного екземпляра.

6. Формування вихідного результату. Для всіх екземплярів формується вихідний набір RDD[Double], що містить оцінки аномальності. Отримані значення можна використовувати для ранжування об’єктів за ступенем підозрілості або для порогування з метою прийняття рішення “норма/аномалія”.

Розглянемо покроковий опис динамічного алгоритму HBOS:

1. Підготовка вхідних даних і параметрів. На вхід подається набір даних у форматі Dataset["features"], де кожен екземпляр описано вектором ознак. Задається кількість інтервалів для побудови гістограм, а також мале згладжувальне значення, яке використовується для уникнення проблем у випадку нульових частот в окремих інтервалах.

2. Ініціалізація структур для динамічної побудови інтервалів. Створюються порожні структури для збереження меж інтервалів та нормованих гістограм для кожної ознаки. Додатково визначається кількість ознак у векторі features, а також загальна кількість екземплярів у наборі даних.

3. Обчислення кроку для квантильного розбиття. На основі кількості екземплярів і кількості інтервалів визначається крок, який задає, через яку кількість елементів у відсортованому ряді потрібно брати наступну межу інтервалу. Це дозволяє сформувати інтервали не рівномірні за діапазоном

значень, а адаптовані до розподілу даних.

4. Сортування значень кожної ознаки. Для кожної ознаки окремо всі її значення у наборі даних сортуються за зростанням. Саме на основі цього впорядкованого списку далі будуть формуватися межі інтервалів.

5. Формування меж інтервалів за квантилями. Межі інтервалів визначаються таким чином, щоб кожен інтервал містив приблизно однакову кількість елементів. Для цього нижні та верхні межі бінів беруться з відсортованого списку значень із заданим кроком. Такий підхід забезпечує “динамічність” інтервалів: вони автоматично підлаштовуються під фактичний розподіл даних.

6. Побудова гістограм за динамічними межами. Після визначення меж інтервалів для кожної ознаки обчислюється гістограма частот уже не за рівномірними інтервалами, а за сформованими квантильними межами. Далі виконується нормалізація частот, щоб отримати відносні частоти, які відображають “типовість” значень у кожному інтервалі.

7. Обчислення внеску ознак в оцінку аномальності. Далі алгоритм проходить по кожному екземпляру набору даних. Для кожної ознаки визначається, у який інтервал (з урахуванням квантильних меж) потрапляє її значення. Потім береться нормоване значення частоти для відповідного інтервалу. Якщо значення потрапляє в інтервал, який зустрічається рідко, внесок цієї ознаки у підсумкову оцінку аномальності зростає.

8. Агрегування внесків усіх ознак. Після обчислення внесків по всіх ознаках вони підсумовуються та нормуються за правилом, передбаченим алгоритмом. Так формується одна підсумкова числова оцінка для кожного екземпляра.

9. Формування вихідного результату. На виході алгоритм повертає RDD[Double] оцінок аномальності для всіх екземплярів. Отримані значення можна використовувати для ранжування об’єктів за ступенем підозрілості або для порогування, щоб визначити, які екземпляри слід вважати аномаліями.

2.2 Онлайн-детектор аномалій

Відомо, що сукупність слабких класифікаторів здатна утворити один сильний класифікатор. На цьому принципі ґрунтується метод Lightweight On-line Detector of Anomalies (LODA) [40], в якому ансамбль дуже слабких детекторів може призвести до створення потужного детектора аномалій.

На відміну від методів, що потребують побудови складних багатовимірних моделей щільності, LODA зводить задачу до набору простих одномірних оцінювань. Основна ідея методу полягає у використанні множини випадкових (часто розріджених) лінійних проєкцій багатовимірних даних на одновимірний простір та подальшому оцінюванні щільності розподілу проєкцій за допомогою гістограм. Аномальними вважаються ті об'єкти, які у більшості проєкцій потрапляють у області з низькою емпіричною щільністю.

Нехай задано набір даних $X = \{x^{(i)}\}_{i=1}^{n}$, де $x^{(i)} \in \mathbb{R}^d$ – вектор із d. Метод LODA формує K випадкових напрямків (векторів) проєкції $w_1, \dots, w_K, w_k \in \mathbb{R}^d$. Для кожного об'єкта x та кожної проєкції w_k обчислюється скалярна проєкція:

$$z_k = w_k^{\top} x. \tag{2.3}$$

Далі для кожної проєкції k на основі значень $\{z_k^{(i)}\}_{i=1}^{n}$ будується одномірна гістограма з B інтервалів, яка дискретно апроксимує розподіл проєкцій. Частоти у бінах нормалізуються до оцінок імовірності (або щільності) $p_k(\cdot)$, що дозволяє кількісно визначити, наскільки "типовим" є значення z_k у межах поточного розподілу.

Оцінка аномальності для об'єкта x визначається шляхом агрегування внесків усіх K проєкцій. У практичній реалізації використовується логарифмічна форма, яка підсилює вплив низьких імовірностей та забезпечує чисельну стійкість. Зокрема, для кожної проєкції обчислюється внесок:

$$s_k(x) - \log(p_k(z_k) + \varepsilon), \tag{2.4}$$

де $\varepsilon > 0$ – мале згладжувальне значення, введене для уникнення обчислення $\log(0)$ у випадку порожніх або майже порожніх бінів. Підсумкова оцінка LODA обчислюється як середнє значення внесків:

$$LODA(x) = \frac{1}{K}\sum_{k=1}^{K} s_k(x). \tag{2.5}$$

Чим більше значення $LODA(x)$, тим менш імовірним є об’єкт відносно розподілу даних і тим вищою є ймовірність, що він належить до аномалій.

Важливою характеристикою LODA є його обчислювальна ефективність. По-перше, оцінювання щільності виконується в одномірному просторі, що суттєво спрощує обробку великих наборів даних. По-друге, у багатьох реалізаціях застосовуються розріджені вектори проєкцій w_k, у яких більшість компонентів дорівнюють нулю. Це зменшує кількість операцій множення та додавання при обчисленні $z_k = w_k^\top x$ і робить метод придатним для потокових сценаріїв і розподілених обчислень. У контексті великих даних LODA також зручний тим, що гістограми можна будувати паралельно для кожної проєкції, а їх оновлення у динамічному режимі може здійснюватися інкрементально або на основі ковзного вікна, що дозволяє враховувати дрейф розподілу в часі.

Після отримання оцінок $LODA(x)$ для всіх об’єктів формується ранжований перелік підозрілих екземплярів. Для переходу до бінарної класифікації (норма/аномалія) застосовують порогування: поріг може визначатися як квантиль розподілу оцінок (наприклад, верхні 1% або 5% значень) або на основі валідації, якщо доступні розмічені дані. Таким чином, LODA забезпечує компроміс між точністю та швидкодією, поєднуючи просту статистичну інтерпретацію з можливістю ефективної реалізації у розподілених середовищах обробки даних.

Розглянемо покроковий опис алгоритму LODA:

1. Підготовка вхідних даних і параметрів. На вхід подається набір даних у форматі Dataset["features"], де кожен запис представлено вектором ознак. Додатково задаються параметри методу: кількість випадкових проєкцій (тобто скільки “слабких” детекторів буде в ансамблі), кількість інтервалів для побудови гістограм, а також мале згладжувальне значення, необхідне для стабільності обчислень у випадках, коли деякі інтервали можуть мати нульову або дуже малу частоту.

2. Перетворення даних у матричне представлення. Для прискорення обробки великих наборів даних початковий набір переводиться у матричний формат. Це дає змогу ефективно виконувати подальші операції проєктування над

усіма екземплярами одночасно, використовуючи оптимізовані матричні обчислення.

3. Генерація випадкових проєкцій. Формується набір випадкових напрямків проєкції. Кожна проєкція визначає спосіб перетворити багатовимірний об’єкт у одне числове значення. У багатьох практичних реалізаціях ці напрямки роблять розрідженими (з великою кількістю нульових компонент), щоб істотно зменшити кількість операцій під час обчислення проєкцій і підвищити швидкодію.

4. Проєктування даних у одновимірні простори. Для кожного об’єкта обчислюються значення у всіх згенерованих проєкціях. У результаті формується новий набір даних, у якому кожен запис описується не початковими ознаками, а набором значень у різних одновимірних проєкціях. Таким чином, багатовимірна задача переводиться у серію простих одномірних задач.

5. Побудова гістограм у кожній проєкції. Для кожної проєкції окремо збираються всі отримані проєкційні значення по всіх об’єктах і будується гістограма з наперед заданою кількістю інтервалів. Під час цього етапу зберігаються межі інтервалів, а частоти потрапляння значень у кожний інтервал нормалізуються, щоб відобразити “типовість” різних діапазонів значень у межах конкретної проєкції.

6. Оцінювання аномальності об’єкта в кожній проєкції. Для кожного об’єкта і для кожної проєкції визначається, у який інтервал гістограми потрапляє його проєкційне значення. Далі використовується нормалізована частота відповідного інтервалу як показник того, наскільки типово зустрічається таке значення в даній проєкції. Якщо об’єкт потрапляє в інтервал із низькою частотою, то він вважається менш типовим у цій проєкції, а отже отримує більший внесок до загальної оцінки аномальності.

7. Агрегування внесків усіх проєкцій у підсумкову оцінку. Внески, отримані від усіх проєкцій для одного об’єкта, об’єднуються в єдину підсумкову оцінку. Зазвичай це робиться шляхом усереднення, що забезпечує стабільність ансамблю: навіть якщо окремі проєкції є дуже слабкими або випадковими, спільний результат багатьох проєкцій формує надійніший показник

аномальності.

8. Формування вихідного результату та інтерпретація. На виході алгоритм повертає набір числових оцінок аномальності для всіх екземплярів даних у вигляді RDD[Double]. Ці значення можна використовувати для ранжування об’єктів за ступенем підозрілості або застосувати порогування, щоб отримати бінарне рішення “норма/аномалія”. Метод є придатним для великих даних і потокових сценаріїв, оскільки базується на простих операціях проєктування та побудови одномірних гістограм, які добре масштабуються в розподіленому середовищі.

2.3 Локально-селективне поєднання у паралельних ансамблях для виявлення аномалій

Метод локально-селективного поєднання у паралельних ансамблях для виявлення викидів (LSCP) [48] є ансамблевим методом, який вирішує проблему відсутності еталонної розмітки (ground-truth) при неконтрольованому виявленні аномалій. Він визначає псевдоеталонну розмітку, використовуючи низку базових детекторів та обираючи їхнє середнє або максимальне значення. Потім він визначає локальну область навколо кожного екземпляра та використовує консенсус найближчих сусідів у випадково обраних підпросторах ознак. Базові детектори, що демонструють найкращу ефективність у таких локальних областях, обираються та поєднуються для формування фінального рішення ансамблю.

Хоча ансамблеві методи продемонстрували здатність досягати високої продуктивності та ефективно адаптуватися до середовищ великих даних [18], методи з високою обчислювальною складністю стикаються з труднощами в таких доменах. LSCP визначає локальну область для кожного екземпляра шляхом знаходження *k*-найближчих сусідів для кількох груп випадково обраних ознак, а потім обирає екземпляри, які найчастіше потрапляли до числа найближчих сусідів. Цей процес передбачає багаторазове застосування методу *k*-NN. У доменах великих даних це означає обчислення мільярдів відстаней, що

може бути неприйнятним з точки зору часових витрат. LSCP спрощує вирішення цієї проблеми високої обчислювальної складності, пов’язаної з визначенням локальної області, шляхом обчислення апроксимації таких *k*-найближчих сусідів.

Розглянемо покроковий опис алгоритму LSCP:

1. Підготовка вхідних даних і параметрів. На вхід подається набір даних у форматі Dataset["features"], де кожен запис описаний набором ознак. Додатково задаються параметри: кількість базових детекторів, спосіб формування псевдоеталонної оцінки (середнє або максимум), метод і кількість кластерів для визначення локальних областей, а також частка детекторів, які потрібно відбирати локально (dcs).

2. Побудова ансамблю базових детекторів. Послідовно формуються базові детектори аномалій. Кожен детектор “дивиться” на ті самі дані та повертає для кожного екземпляра власну оцінку аномальності. Після запуску кожного нового детектора його оцінки приєднуються до таблиці результатів, так що для кожного об’єкта накопичується набір оцінок від усіх детекторів.

3. Формування псевдоеталонної оцінки. Для кожного об’єкта на основі всіх отриманих оцінок детекторів формується додаткова узагальнена оцінка – псевдоеталон. Якщо обрана стратегія “avg”, псевдоеталон визначається як середнє значення оцінок усіх детекторів. Якщо обрана стратегія “max”, псевдоеталоном береться найбільша оцінка серед детекторів. Цей псевдоеталон використовується як орієнтир для визначення того, які детектори працюють узгоджено та стабільно в певній локальній області даних.

4. Визначення локальних областей методом кластеризації. Дані разом із псевдоеталонною оцінкою розбиваються на локальні області за допомогою кластеризації. Тип кластеризації визначається параметром: або використовується k-means, або бісекційний підхід. Кількість кластерів задається окремо. У результаті кожен об’єкт належить до певного кластера, а кожен кластер розглядається як локальна область, у межах якої будуть оцінюватися детектори.

5. Оцінювання локальної компетентності детекторів у кожному кластері.

Далі алгоритм обробляє кожен кластер окремо. Усередині кластера для кожного базового детектора порівнюється його “поведінка” з псевдоеталоном. Для цього обчислюється показник узгодженості між оцінками цього детектора та псевдоеталонними значеннями на об’єктах кластера. Чим вища узгодженість, тим більш компетентним вважається детектор саме в цій локальній області.

6. Динамічний локальний відбір детекторів. Після отримання показників узгодженості виконується локальний відбір детекторів. Якщо параметр dcs дорівнює 1, то відбір не проводиться – використовуються всі детектори ансамблю. Якщо dcs менший за 1, тоді відбирається лише певна частка детекторів із найвищими показниками узгодженості в даному кластері. Таким чином, у кожній локальній області можуть бути обрані різні детектори.

7. Комбінування оцінок відібраних детекторів. Для кожного об’єкта підсумкова оцінка аномальності формується не з усіх детекторів, а лише з тих, що були відібрані як найбільш компетентні у відповідній локальній області. Далі ці оцінки агрегуються (об’єднуються) у один підсумковий бал за правилом, яке закладене в алгоритмі та залежить від обраної стратегії псевдоеталона.

8. Формування вихідного результату. На виході алгоритм повертає набір числових оцінок аномальності для всіх екземплярів даних. Ці значення можуть використовуватися для ранжування підозрілих об’єктів або для подальшого порогування, щоб отримати бінарне рішення “норма/викид”.

Розглянемо покроковий опис алгоритму 2.5, який демонструє процес партиціонування даних за схожістю із застосуванням кластеризації:

1. Отримання вхідного представлення. На цьому етапі як вхід використовуються дані, підготовлені алгоритмом 2.4: це може бути початковий вектор ознак, набір згенерованих характеристик, зведені показники або будь-яке інше числове представлення об’єктів, яке вже придатне для оцінювання схожості між ними.

2. Вибір методу кластеризації та параметрів. Обирається конкретний метод кластеризації для групування об’єктів за схожістю (наприклад, k-means або бісекційна кластеризація). Далі задаються параметри методу, насамперед кількість кластерів, що визначає, на скільки локальних груп буде поділено дані.

3. Обчислення кластерної структури даних. Алгоритм кластеризації запускається на підготовлених даних і формує групи таким чином, щоб об’єкти в межах однієї групи були максимально схожими, а різні групи – максимально відмінними. У результаті кожному об’єкту присвоюється ідентифікатор кластера, який фіксує його належність до певної локальної області.

4. Маркування (прив’язка) об’єктів до локальних областей. Після кластеризації до кожного запису додається інформація про його кластер (мітка/ідентифікатор). Це створює основу для подальшої локальної обробки, оскільки тепер можна явно відрізняти об’єкти різних локальних областей.

5. Формування партицій на основі кластерів. Дані групуються або перепартиціонуються так, щоб об’єкти з однаковою кластерною міткою опинилися в одній логічній або фізичній партиції. У розподіленому середовищі це дозволяє виконувати обчислення в межах кожного кластера незалежно, не змішуючи об’єкти з різних локальних областей.

6. Підготовка до локальної обробки в наступних алгоритмах. Сформовані партиції використовуються як “контейнер” для подальших локальних операцій: обчислення статистик усередині групи, оцінювання якості або узгодженості детекторів, локального відбору моделей, побудови локальних порогів чи формування локально-адаптивних ансамблевих рішень.

7. Формування вихідного результату. На виході отримується набір даних, у якому кожен об’єкт належить до певної локальної області (кластера) і розміщений у відповідній партиції. Така структура є основою для наступних етапів, де необхідна локальна обробка даних та локально-адаптивні процедури виявлення аномалій.

2.4 Виявлення аномалій на основі екстремального градієнтного бустингу

Метод виявлення викидів на основі екстремального градієнтного бустингу (XGBOD) [47] є частково контрольованим ансамблевим підходом, який поєднує неконтрольоване виявлення аномалій з подальшим контрольованим навчанням класифікатора XGBoost у розподіленому середовищі. На відміну від класичного

XGBOD, який є обмеженим щодо масштабованості через використання традиційних неконтрольованих алгоритмів та ітеративного навчання на великих обсягах даних, XGBOD_BD адаптований до сценаріїв Big Data завдяки застосуванню розподілених детекторів аномалій і розподіленої реалізації XGBoost у Spark.

Основна ідея методу полягає в тому, щоб розширити початковий простір ознак додатковими інформативними характеристиками, отриманими з неконтрольованих детекторів аномалій. Такі характеристики називаються трансформованими оцінками викидів (Transformed Outlier Scores, TOS). Кожен TOS є числовою оцінкою “підозрілості” об’єкта, сформованою окремим неконтрольованим методом. Далі вибрані TOS додаються до початкових ознак і на розширених даних навчається контрольований класифікатор XGBoost.

У Алгоритмі 2.6 покроково описано метод виявлення аномалій XGBOD_BD, який поєднує розподілені неконтрольовані детектори для формування трансформованих оцінок викидів (TOS) і подальше навчання розподіленого класифікатора XGBoost на розширеному просторі ознак:

1. Формування базового набору та ініціалізація структури TOS. Робота методу починається з того, що створюється робоча структура TOS, яка містить початкові дані (ознаки та мітку класу). Вона використовується як контейнер для поступового накопичення оцінок аномальності від різних базових детекторів.

2. Навчання неконтрольованих детекторів і накопичення TOS. Далі запускається цикл, у якому послідовно навчаються n_TOS базових неконтрольованих детекторів аномалій. Кожен детектор обчислює для всіх екземплярів числові оцінки “підозрілості” (outlier scores). Отримані оцінки приєднуються до структури TOS за допомогою розподіленої операції join у Spark. У результаті кожен об’єкт отримує набір додаткових числових характеристик – трансформованих оцінок викидів (TOS).

3. Ініціалізація списку відібраних TOS. Після формування повного набору n_TOS оцінок створюється порожній список selected_TOS, у який будуть занесені індекси тих TOS, що потраплять до розширеного простору ознак.

4. Відбір TOS за стратегією “випадково” (rnd). Якщо задано стратегію

TOS_strategy = "rnd", тоді метод обирає n_selected_TOS індексів випадковим чином без повторень. Це забезпечує швидкий відбір без додаткових обчислень якості окремих TOS і використовується як проста базова стратегія.

5. Відбір TOS за стратегією “за точністю” (acc): формування псевдоміток. Якщо задано TOS_strategy = "acc", для кожної TOS виконується перетворення її числових значень у бінарні мітки за пороговим правилом threshold. Тобто кожне значення TOS порівнюється з порогом і отримує мітку “аномалія” або “норма”. Це перетворення виконується розподілено через операцію map, що дозволяє застосувати його до великих обсягів даних.

6. Оцінювання точності кожної TOS. Для кожної розміченої TOS обчислюється показник точності шляхом порівняння сформованих псевдоміток із реальними мітками label, які вже містяться у наборі даних. Для цього застосовується інструмент оцінювання MulticlassMetrics у Spark. У результаті формується список значень точності для всіх TOS.

7. Вибір найкращих TOS за точністю. Після оцінювання якості всі TOS сортуються за отриманими значеннями точності, і відбираються індекси n_selected_TOS TOS з найвищою точністю. Саме ці TOS вважаються найбільш інформативними для подальшого контрольованого навчання.

8. Формування розширеного простору ознак. На наступному кроці створюється features_combined: для кожного екземпляра початковий вектор ознак доповнюється відібраними TOS (тобто до початкових ознак додаються значення лише тих детекторів, індекси яких містяться у selected_TOS). Таким чином формується розширений опис об’єкта, який включає як первинні характеристики, так і ансамблеві індикатори аномальності.

9. Навчання розподіленої моделі XGBoost і отримання прогнозів. На завершальному етапі на сформованому розширеному наборі ознак навчається розподілений класифікатор xgbClassifier (Spark/XGBoost). Модель будує прогноз для кожного екземпляра і повертає його як вихід методу – тобто кінцеве рішення щодо належності об’єкта до аномалій або нормального класу.

Такий процес дозволяє поєднати переваги неконтрольованого виявлення структури даних (через TOS) із високою прогностичною здатністю

контрольованого ансамблю XGBoost, забезпечуючи при цьому масштабованість завдяки розподіленій реалізації в середовищі Spark.

Висновки до розділу 2

1. Розроблено розподілені архітектури для базових алгоритмів виявлення аномалій, що усунуло їхні фундаментальні обмеження щодо масштабованості. Це було досягнуто шляхом заміни їхніх послідовних, ітераційних етапів на високопродуктивні паралельні операції, що надаються фреймворком Apache Spark. Зокрема, для HBOS було впроваджено розподілену побудову гістограм, а для LODA – заміну множинних ітераційних проєкцій на єдину розподілену матричну операцію. Це дало змогу перетворити класичні, але не масштабовані методи, на повноцінні інструменти для Big Data, здатні обробляти великі масиви даних за прийнятний час, зберігаючи при цьому вихідну логіку оцінки аномальності.

2. Розв’язано ключову проблему обчислювальної складності у складних ансамблевих методах, зокрема в LSCP, шляхом інноваційної модифікації його архітектури. Це було реалізовано через запропоновану заміну ресурсоємного та непрактичного для великих даних методу k-найближчих сусідів на ефективну розподілену кластеризацію для формування "локальних областей".

3. Сформовано комплексний набір інструментів для виявлення аномалій, що охоплює як неконтрольовані, так і гібридні підходи. Це було здійснено через послідовну розробку та формальний опис алгоритмів, кожен з яких представляє окрему категорію методів – від простих статистичних до складних ансамблевих та частково контрольованих.

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ МЕТОДІВ ВИЯВЛЕННЯ АНОМАЛІЙ У ВЕЛИКИХ ДАНИХ

3.1 Реалізація та налаштування модуля для виявлення аномалій

У цьому підрозділі розглянемо програмний модуль, розроблений у межах даної кваліфікаційної роботи, для виявлення аномалій у статичних та динамічних наборах великих даних у середовищі Apache Spark. Модуль реалізує чотири розподілені алгоритми виявлення аномалій: HBOS (алгоритм 3.1 та алгоритм 3.2), LODA (алгоритм 3.3), LSCP (алгоритм 3.4 та алгоритм 3.5) та XGBOD (алгоритм 3.6).

Алгоритм 3.1 описує процес статичного виявлення аномалій за допомогою HBOS.

Алгоритм 3.1 – Статичний алгоритм HBOS

```
Вхід:
   data    – набір даних у форматі Dataset["features"], де features – вектор ознак
   n_bins  – кількість інтервалів (бінів) для гістограм
   ε       – мале додатне число для згладжування (щоб уникнути log(0))

Вихід:
   scores  – RDD[Double] оцінок аномальності для кожного екземпляра data

1:  limits      ← ∅                          // межі бінів для кожної ознаки
2:  histograms  ← ∅                          // нормовані гістограми p(i,b) для кожної ознаки
3:  n_attributes ← length(features)          // кількість ознак у векторі features

4:  for i ← 0 to n_attributes − 1 do
5:      hist ← histogram(select(data, i), n_bins)        // частоти по і-й ознаці
6:      limits[i] ← getLimits(hist)                      // межі бінів
7:      hist_sum ← sum(getHistogram(hist))               // сума частот (≈ |data|)
8:      histograms[i] ← map l ∈ getHistogram(hist) : (l / hist_sum)  // p(i,b)
9:  end for

10: scores ← map instance ∈ data do
11:     values ← ∅                                       // внески ознак
12:     for i ← 0 to n_attributes − 1 do
13:         index ← computeIndex(instance(i), limits[i])  // номер біна для x_i
14:         p ← histograms[i][index]                      // p(i, b(x_i))
15:         values[i] ← −log(p + ε)                       // computeScore: внесок HBOS
16:     end for
17:     return sum(values) / n_bins                       // агрегована оцінка (нормована)
18: end map

19: return scores
```

Алгоритм 3.2 описує процес динамічного виявлення аномалій за допомогою HBOS.

Алгоритм 3.2 – Динамічний алгоритм HBOS

```
Вхід:
   data    – набір даних у форматі Dataset["features"], де features – вектор ознак
   n_bins  – кількість інтервалів (бінів) для гістограм
   ε       – мале додатне число для згладжування (щоб уникнути log(0))

Вихід:
   scores  – RDD[Double] оцінок аномальності для кожного екземпляра data

1:  limits     ← ∅                          // межі бінів для кожної ознаки (динамічні)
2:  histograms ← ∅                          // нормовані гістограми p(i,b) для кожної ознаки
3:  n_attributes ← length(features)         // кількість ознак
4:  N ← count(data)                         // кількість екземплярів
5:  inc ← floor(N / n_bins)                 // крок для квантильного розбиття

6:  for i ← 0 to n_attributes − 1 do
7:     sortedValues ← sort(select(data, i))         // сортування значень i-ї ознаки
8:
9:     // Формування меж бінів за квантилями:
10:    limits[i].lower ← sortedValues[0 .. N−inc step inc]   // нижні межі інтервалів
11:    limits[i].upper ← sortedValues[inc .. N step inc]     // верхні межі інтервалів
12:
13:    // Обчислення гістограми за заданими межами (частоти по бін-інтервалах):
14:    hist ← computeHistogram(select(data, i), limits[i])
15:    hist_sum ← sum(getHistogram(hist))           // сума частот (≈ N)
16:    histograms[i] ← map l ∈ getHistogram(hist) : (l / hist_sum)   // p(i,b)
17: end for

18: scores ← map instance ∈ data do
19:    values ← ∅                           // внески ознак
20:    for i ← 0 to n_attributes − 1 do
21:       index ← computeIndexDynamic(instance(i), limits[i])      // бін для x_i у квантильних
межах
22:       p ← histograms[i][index]                  // p(i, b(x_i))
23:       values[i] ← −log(p + ε)                   // внесок HBOS
24:    end for
25:    return sum(values) / n_bins                  // агрегована оцінка (нормована)
26: end map

27: return scores
```

У алгоритмі 3.3 описано реалізацію методу виявлення аномалій LODA.

Алгоритм 3.3 – Алгоритм LODA

```
Вхід:
   data    – набір даних у форматі Dataset["features"], де features – вектор ознак
   n_bins  – кількість інтервалів (бінів) для гістограм
   k       – кількість випадкових проєкцій
   ε       – мале додатне число для згладжування (щоб уникнути log(0))

Вихід:
   scores  – RDD[Double] оцінок аномальності для кожного екземпляра набору даних

1:  dataAsMatrix ← RowMatrix(data)                  // матричне представлення даних
2:  projections  ← createRandomProjections(dim(features), k)   // матриця проєкцій W ∈ R^{d×k}
3:  projectedData ← multiply(dataAsMatrix, projections)        // Z = X·W, де Z ∈ R^{n×k}

4:  limits     ← ∅                          // межі бінів для кожної проєкції
5:  histograms ← ∅                          // нормовані гістограми p(i,b) для кожної проєкції

6:  for i ← 0 to k − 1 do
```

```
7:     hist ← histogram(select(projectedData, i), n_bins)       // гістограма по і-й проєкції
8:     limits[i] ← getLimits(hist)                              // межі бінів
9:     hist_sum ← sum(getHistogram(hist))                       // сума частот (≈ кількість елементів)
10:    histograms[i] ← map l ∈ getHistogram(hist) : (l / hist_sum) // p(i,b)
11: end for

12: scores ← map instance ∈ projectedData do                    // instance має k проєкційних значень
13:    values ← ∅                                               // внески кожної проєкції
14:    for i ← 0 to k − 1 do
15:       index ← computeIndex(instance(i), limits[i])          // номер біна для z_i
16:       p ← histograms[i][index]                              // p(i, b(z_i))
17:       values[i] ← −log(p + ε)                               // внесок LODA у проєкції і
18:    end for
19:    return sum(values) / k                                   // підсумкова оцінка LODA(x)
20: end map

21: return scores
```

У Алгоритмі 3.4 описано метод виявлення аномалій LSCP.

Алгоритм 3.4 – Алгоритм LSCP

```
Вхід:
   data            – набір даних у форматі Dataset["features"]
   n_base_detectors – кількість базових детекторів M
   pgt_strategy     – стратегія псевдоеталонної розмітки: "avg" або "max"
   clus_method      – метод кластеризації для локальних областей: "kmeans" або "bisec"
   n_clus           – кількість кластерів
   dcs              – частка детекторів для динамічного відбору (0 < dcs ≤ 1)

Вихід:
   scores          – RDD[Double] підсумкових оцінок аномальності для кожного екземпляра

1:  detectors ← data                                            // структура для приєднання скорів детекторів

2:  for i ← 0 to n_base_detectors − 1 do
3:     baseScores ← learnBaseDetector(data)                     // RDD/Dataset: (id, s_i)
4:     detectors ← join(detectors, baseScores)                  // (id, features, s_0, s_1, …, s_i)
5:  end for

6:  pseudo_gt ← map row ∈ detectors do                          // додати псевдоеталонний стовпець ŷ
7:     if pgt_strategy = "avg" then
8:        y_hat ← avg(row.scores[0..M−1])                      // середнє по детекторах
9:     else                                                     // pgt_strategy = "max"
10:       y_hat ← max(row.scores[0..M−1])                      // максимум по детекторах
11:    end if
12:    return append(row, y_hat)                                // (id, features, s_1..s_M, y_hat)
13: end map

14: gt_clustered ← clusterPartitioning(pseudo_gt, clus_method, n_clus)
   // Результат: дані розбиті на локальні області (кластери/партиції)

15: scores ← mapPartitions partition ∈ gt_clustered do
16:    correlations ← ∅                                         // локальна компетентність детекторів
17:    y_ref ← partition.column("y_hat")                        // референтний сигнал у локальній області

18:    for i ← 0 to n_base_detectors − 1 do
19:       s_i ← partition.column("s_i")                         // оцінки і-го детектора в локальній області
20:       correlations[i] ← pearson(s_i, y_ref)                 // компетентність: кореляція Пірсона
21:    end for

22:    if dcs = 1 then                                          // без відбору, використовуємо весь ансамбль
23:       return max(partition.column("y_hat"))                 // або інша агрегація за постановкою
```

```
    24:     else
    25:         L ← ceil(dcs * n_base_detectors)                    // кількість відібраних детекторів
    26:         mostCorrelated ← top(correlations, L)               // індекси детекторів з найбільшою
компетентністю

    27:         selectedScores ← partition.columns(mostCorrelated)  // оцінки обраних детекторів
    28:         if pgt_strategy = "avg" then
    29:             return max(selectedScores)                      // комбінація оцінок обраних детекторів
    30:         else                                                // pgt_strategy = "max"
    31:             return avg(selectedScores)
    32:         end if
    33:     end if
    34: end mapPartitions

    35: return scores
```

Алгоритм 3.5 – Функція clusterPartitioning

```
    1:  Вхідні дані: data       – набір даних у форматі Dataset["features"]
    2:  Вхідні дані: clus_method – метод кластеризації (“kmeans”, “bisec”)
    3:  Вхідні дані: n_clus     – кількість кластерів
    4:  Вхідні дані: max_size   – максимальна кількість елементів у партиції (за замовчуванням = 10
000)
    5:  Вхідні дані: min_size   – мінімальна кількість елементів у партиції (за замовчуванням = 1
000)
    6:  Вихідні дані: Dataset/RDD з даними, партиціонованими за локальними областями (кластер =
партиція)

    7:  clustering ← ∅
    8:  if clus_method = "kmeans" then
    9:      clustering ← KMeans(data, n_clus)                   // додати поле "cluster" до кожного
екземпляра
    10: end if
    11: if clus_method = "bisec" then
    12:     clustering ← BisectingKMeans(data, n_clus)          // додати поле "cluster"
    13: end if

    14: repartitionedData ← repartitionByRange(clustering, "cluster")

    15: balancedData ←
    16: mapPartitions partition ∈ repartitionedData do
    17:     clusterSize ← size(partition)

    18:     if clusterSize > max_size then                      // надто великий кластер: поділ на підкластери
    19:         for j ← 0 until clusterSize by max_size do
    20:             partition[j].cluster ← partition[j].cluster + 0.1   // зміщення мітки для виділення
підпартицій
    21:         end for

    22:     else if clusterSize < min_size then                 // надто малий кластер: винести у “збірну”
партицію
    23:         for each row ∈ partition do
    24:             row.cluster ← ∞                             // спеціальна мітка для об’єднання малих
кластерів
    25:         end for
    26:     end if

    27:     emit partition
    28: end mapPartitions

    29: return repartitionByRange(balancedData, "cluster")
```

Алгоритм 3.6 – Алгоритм XGBOD

```
1:  Вхідні дані: data           – набір даних у форматі Dataset["features","label"]
2:  Вхідні дані: n_TOS          – кількість TOS (кількість базових неконтрольованих детекторів)
3:  Вхідні дані: n_selected_TOS  – кількість TOS, що будуть відібрані
4:  Вхідні дані: TOS_strategy    – стратегія відбору TOS ("rnd", "acc")
5:  Вхідні дані: threshold       – поріг для стратегії "acc"
6:  Вихідні дані: predictions    – RDD[Double] прогнозів/міток для кожного екземпляра

7:  // 1) Побудова TOS: навчити n_TOS базових детекторів і приєднати їх оцінки до даних
8:  TOS ← attachId(data)                                  // додати id для коректного join
9:  for i ← 0 until n_TOS do
10:     baseScores ← learnBaseDetector(TOS)                  // повертає (id, score_i)
11:     TOS ← joinById(TOS, baseScores)                      // тепер у TOS є score_0..score_i
12: end for

13: // 2) Відбір індексів TOS
14: selected_TOS ← ∅
15: if TOS_strategy = "rnd" then
16:     selected_TOS ← sampleWithoutReplacement(0..n_TOS−1, n_selected_TOS)
17: else if TOS_strategy = "acc" then
18:     // 2.1) Перетворити кожну TOS у псевдомітки за порогом
19:     labeledTOS ← map row ∈ TOS do
20:         pseudo ← emptyVector(n_TOS)
21:         for i ← 0 until n_TOS do
22:             if row.score[i] ≥ threshold then pseudo[i] ← 1 else pseudo[i] ← 0
23:         end for
24:         return (row.id, row.label, pseudo, row.features)
25:     end map

26:     // 2.2) Оцінити точність кожної TOS і вибрати найкращі
27:     accuracy ← emptyVector(n_TOS)
28:     for i ← 0 until n_TOS do
29:         pairs ← map r ∈ labeledTOS : (r.label, r.pseudo[i])      // (trueLabel, predictedLabel)
30:         accuracy[i] ← MulticlassMetrics(pairs).accuracy
31:     end for
32:      selected_TOS ← topIndicesByValue(accuracy, n_selected_TOS)    // індекси з найбільшою
точністю
33: end if

34: // 3) Формування розширеного простору ознак: додати вибрані TOS до features
35: features_combined ← map row ∈ TOS do
36:     tos_selected ← selectScores(row.score[0..n_TOS−1], selected_TOS)
37:     newFeatures ← concat(row.features, tos_selected)
38:     return (row.id, newFeatures, row.label)
39: end map

40: // 4) Навчання розподіленої моделі XGBoost і отримання прогнозів
41: model ← xgbClassifierTrain(features_combined)                 // Spark XGBoost
42: predictions ← model.predict(features_combined)

43: return predictions
```

Під час реалізації основний акцент зроблено на масштабованості та узгодженості інтерфейсу, щоб користувач міг застосовувати різні методи без зміни підходу до підготовки даних і запуску.

Реалізація спирається на стандартні механізми Spark для розподіленої обробки даних і використовує типові примітиви та компоненти платформи

(зокрема операції перетворення/агрегації над RDD/Dataset, а також допоміжні процедури на кшталт map, join, mapPartitions, repartitionByRange, обчислення гістограм і метрик якості). Усі алгоритми виконано за єдиними принципами проєктування: уніфіковані назви параметрів, однакова логіка ініціалізації, спільні підходи до повернення результату. Це спрощує порівняння методів та дозволяє швидко перемикатися між ними в рамках одного експерименту.

Щоб використовувати реалізований модуль у Spark-застосунку, його необхідно підключити до середовища виконання як бібліотеку (JAR). Залежно від способу запуску це можна зробити, наприклад, через додавання JAR у classpath під час старту spark-shell або під час запуску spark-submit:

```
$SPARK_HOME/bin/spark-shell --jars /path/to/anomaly-module.jar
```

Лістинг 3.1 – Підключення модуля виявлення аномалій до Spark

Після підключення бібліотеки виконується імпорт простору імен, що надає доступ до реалізованих алгоритмів:

```
import <your.package>.anomaly._
```

Лістинг 3.2 – Імпорт модуля

Для запуску будь-якого з реалізованих алгоритмів створюється екземпляр відповідного класу. Усі методи підтримують однакову схему використання: параметри задаються під час створення об’єкта, а запуск обчислень виконується викликом методу fit(). Для зручності передбачено значення параметрів за замовчуванням, тому користувач може запускати методи навіть із мінімальними налаштуваннями.

Спільною вимогою для всіх алгоритмів є формат вхідних даних: це має бути Dataset[Row] зі стовпцем features, який містить вектор ознак (наприклад, DenseVector). Для методу XGBOD_BD, який є частково контрольованим, додатково потрібен стовпець label типу Double, що містить еталонні мітки класів. Вихідним результатом для методів, що повертають оцінки аномальності, є RDD[Double] зі значенням оцінки для кожного екземпляра вхідного набору; порядок елементів у вихідному RDD відповідає порядку у вхідних даних. Для XGBOD_BD вихід інтерпретується як прогноз моделі (мітка або значення, що

використовується для прийняття рішення).

У лістингу 3.3 наведено приклад завантаження багатоколонкового набору даних, перетворення його до формату зі стовпцем features та запуск алгоритму HBOS_BD у статичному режимі:

```
import <your.package>.anomaly._
import org.apache.spark.sql.{Dataset, Row}
import org.apache.spark.ml.feature.VectorAssembler

val path = "file:///example/of/path/"

// Завантаження даних
val raw_data: Dataset[Row] = spark.read.parquet(path)

// Формування вектору ознак
val assembler: VectorAssembler = new VectorAssembler()
  .setInputCols(raw_data.columns)
  .setOutputCol("features")

val full_dataset = assembler.transform(raw_data).select("features")

// Запуск алгоритму HBOS_BD (статичний режим)
val scores = new HBOS_BD(dataset = full_dataset, n_bins = 100, strategy =
"static").fit()
```

Лістинг 3.3 – Приклад запуску алгоритму HBOS_BD у Spark

Програмний код реалізації методів виявлення аномалій у великих даних представлено у додатку Б.

3.2 Оцінка продуктивності розподілених детекторів аномалій

Для експериментального дослідження в даній роботі використано набір даних, який містить часові ряди показників сенсорів виробничої установки та супровідну інформацію, пов’язану з експлуатаційними подіями, зокрема реєстрацію моментів відмов і контекстні атрибути. Кожен сенсор відповідає окремій ознаці набору даних; значення ознак переважно є дійсного типу та описують різні фізичні й технологічні параметри функціонування обладнання.

Дані охоплюють дворічний період і включають майже 40 мільйонів спостережень, кожне з яких характеризується понад 100 ознаками. Метою дослідження є раннє виявлення ознак наближення відмов, щоб зменшити тривалість ремонтів, мінімізувати простої та підвищити надійність роботи

виробничої системи.

Перед застосуванням реалізованих у даній роботі методів виявлення аномалій дані пройшли етап попередньої обробки. Зокрема, значення ознак було масштабовано до заданого діапазону, що зменшує вплив різного масштабу сенсорних вимірювань і підвищує стабільність роботи алгоритмів. Із загального переліку ознак також вилучено 6 константних ознак (із 112), оскільки вони не несуть корисної інформації для відокремлення аномальних станів.

Запропонований у роботі програмний модуль включає чотири алгоритмічні підходи, реалізовані у вигляді розподілених процедур у Spark:

– HBOS у двох варіантах: статичний (див. алгоритм 3.1) та динамічний (див. алгоритм 3.2);

– LODA (див. алгоритм 3.3);

– LSCP (див. алгоритм 3.4) із процедурою кластерного партиціонування (див. алгоритм 3.5);

– XGBOD як частково контрольований ансамблевий метод (див. алгоритм 3.6).

Для ансамблевих методів (LSCP та XGBOD) необхідно мати набір базових детекторів. У межах експерименту як базовий детектор використано LODA, оскільки він формує різноманітні оцінки аномальності завдяки множині випадкових проєкцій, що є важливим для ансамблевого комбінування. Для XGBOD, який потребує розмічених даних, розмітку сформовано на основі часових інтервалів до відмови: спостереження, що потрапляли у заданий проміжок годин перед моментом відмови, отримували мітку аномалія (клас 1), тоді як решта даних визначалася як норма (клас 0).

Для оцінювання якості методів застосовано метрику ROC-AUC, яка є стандартною для задач виявлення аномалій і дозволяє коректно аналізувати результати в умовах дисбалансу класів. Хоча більшість алгоритмів у модулі працюють в неконтрольованому режимі, під час оцінювання використовувалася еталонна розмітка, яка була невідомою алгоритмам на етапі побудови оцінок. Продуктивність методів оцінювалася за здатністю виявляти наближення відмов у часових вікнах від 1 до 48 годин до моменту відмови.

Оскільки задачі виявлення аномалій суттєво залежать від властивостей даних, одна й та сама комбінація параметрів не гарантує оптимальну якість на різних наборах. Тому для кожного методу виконано підбір гіперпараметрів, де цільовою функцією обрано максимізацію значення ROC-AUC.

У проведених експериментах для алгоритму HBOS кількість інтервалів гістограми встановлено на рівні n_bins = 50. Це означає, що розподіл значень кожної ознаки апроксимується 50 бін-інтервалами, що забезпечує швидшу побудову гістограм і подальший розрахунок оцінок аномальності. При цьому гістограмне подання має більш узагальнений характер, оскільки щільність розподілу описується менш деталізовано.

Для алгоритму LODA у експериментах використано k = 50 випадкових проєкцій та n_bins = 50 інтервалів для побудови гістограм у кожній проєкції. Таким чином, оцінка аномальності формується як агрегований результат 50 незалежних одномірних представлень даних, а щільність у кожному представленні оцінюється за допомогою гістограм із 50 бінів.

У методі LSCP в якості базового детектора застосовано LODA з параметром k = 50. Ансамбль формувався з n_base_detectors = 5 базових детекторів. Для визначення локальних областей дані кластеризувалися із використанням n_clus = 12 кластерів. Така конфігурація задає кількість локальних сегментів, у межах яких оцінюється узгодженість базових детекторів і виконується їх локальний відбір для подальшого агрегування оцінок.

Для алгоритму XGBOD у експерименті формувалося n_TOS = 6 трансформованих оцінок викидів (TOS), з яких до розширеного простору ознак відбиралося n_selected_TOS = 3. Після формування розширеного набору ознак навчалася розподілена модель XGBoost у Spark. Оптимізація гіперпараметрів виконувалася з обмеженою кількістю спроб, що відображено у процедурі підбору параметрів для отримання значення ROC-AUC.

Розглянемо результати застосування реалізованих у даній роботі алгоритмів HBOS (статичного та динамічного), LODA, LSCP та XGBOD на промисловому наборі сенсорних даних. Оцінювання виконувалося за еталонною розміткою, що використовувалася лише для перевірки результатів і була

невідомою алгоритмам під час побудови оцінок. Для аналізу розглянуто кілька часових вікон, які охоплюють проміжок від 1 до 48 годин до моменту відмови. У неконтрольованому сценарії очікуються помірні значення ROC-AUC, особливо у випадку найкоротших інтервалів, коли ознаки деградації можуть бути слабко вираженими.

У таблиці 3.1 наведено результати у вигляді значень ROC-AUC для чотирьох методів, що входять до складу програмного пакету.

Таблиця 3.1 – Значення показника ROC-AUC для кожного алгоритму залежно від кількості годин до настання відмови

Кількість годин до відмови	HBOS		LODA	LSCP	XGBOD
	Статичний	Динамічний			
1	0.6072	0.6203	0.5906	0.6067	0.4819
2	0.6102	0.6291	0.5894	0.6070	0.4966
3	0.6116	0.6334	0.5914	0.6082	0.4869
4	0.6161	0.6355	0.5967	0.6113	0.4840
5	0.6188	0.6356	0.5968	0.6106	0.4854
6	0.6232	0.6427	0.5987	0.6128	0.4853
9	0.6377	0.6530	0.6106	0.6250	0.4970
12	0.6522	0.6634	0.6225	0.6372	0.5088
15	0.6641	0.6756	0.6266	0.6442	0.5014
18	0.6760	0.6879	0.6307	0.6513	0.4940
21	0.6912	0.7020	0.6504	0.6719	0.4900
24	0.7064	0.7162	0.6701	0.6924	0.4860
27	0.7178	0.7269	0.6769	0.6998	0.4975
30	0.7293	0.7375	0.6838	0.7073	0.5090
33	0.7408	0.7482	0.6906	0.7148	0.5204
36	0.7522	0.7589	0.6975	0.7222	0.5319
42	0.7798	0.7928	0.7356	0.7606	0.5339
48	0.8075	0.8267	0.7738	0.7991	0.5360

Загальна тенденція, яку демонструє таблиця 3.1, полягає в тому, що якість виявлення зростає зі збільшенням часового вікна до відмови. Це проявляється у

поступовому підвищенні ROC-AUC майже для всіх методів: найменші значення спостерігаються при коротких вікнах (1–6 годин), а найбільші – при довших (24–48 годин). Така поведінка є очікуваною, оскільки збільшення горизонту до події відмови дає змогу алгоритмам “побачити” більше передаварійних змін у сигналах сенсорів, що робить аномальні патерни більш вираженими.

Для HBOS видно, що динамічний варіант стабільно перевищує статичний для всіх часових вікон. Це означає, що формування меж бінів з урахуванням розподілу даних (квантильне розбиття) дає більш інформативну гістограмну модель у порівнянні зі статичними рівномірними інтервалами. Різниця між динамічним і статичним режимом невелика на коротких інтервалах (1–6 годин), але стає більш помітною при довших вікнах, де накопичуються передумови до відмови.

Алгоритм LODA демонструє значення ROC-AUC, які зазвичай нижчі за HBOS, однак також зростають зі збільшенням кількості годин до відмови. Це вказує на те, що випадкові проєкції та одномірні гістограми LODA здатні уловлювати зміни, пов’язані з передвідмовними станами, але загальна точність у даному сценарії поступається гістограмному підходу HBOS із динамічними межами.

Метод LSCP показує типову поведінку ансамблевого підходу: його результати вищі за базовий детектор (LODA) майже на всіх часових вікнах. Це свідчить про ефективність локально-селективного комбінування, коли в кожній локальній області даних обираються найбільш узгоджені (компетентні) детектори, а потім їхні оцінки агрегуються. Таким чином, LSCP підсилює слабший базовий детектор через ансамблювання.

Натомість XGBOD демонструє значення ROC-AUC, що наближені до випадкового вгадування (близько 0.5) і змінюються незначно навіть при збільшенні часового вікна. Такий результат означає, що в даній постановці задача частково контрольованого методу виявилась складною: або стратегія формування розмітки за порогом/вікном до відмови дала шумні мітки, або розширені ознаки на основі TOS не забезпечили достатньої дискримінативності для XGBoost.

Отже, таблиця 3.1 підтверджує, що для досліджуваного набору даних найбільш ефективними є HBOS (особливо динамічний варіант) та LSCP, тоді як LODA забезпечує середні результати, а XGBOD у цьому експерименті не демонструє вираженої здатності до раннього виявлення відмов за метрикою ROC-AUC.

З метою верифікації продуктивності запропонованих розподілених реалізацій у нашому середовищі виконано порівняння запропонованих алгоритмів з класичними (нерозподіленими) версіями HBOS, LODA, LSCP та XGBOD. Оцінювання проводилося в умовах обмежених обчислювальних ресурсів, тому всі алгоритми з боку Big Data-реалізацій запускалися з фактично використаними параметрами, адаптованими для зменшення обчислювального навантаження (зокрема, зменшена кількість бінів, проєкцій, базових детекторів, TOS та кількості спроб оптимізації гіперпараметрів).

Для порівняння були використані загальнодоступні еталонні набори даних з репозиторію ODDS стандартного розміру (Breast, Shuttle, Satellite), а також два відносно великі числові набори даних (Donors та Census), які часто застосовуються для перевірки масштабованості методів виявлення аномалій. Ефективність оцінювалася за метрикою ROC-AUC, що забезпечує коректне порівняння методів незалежно від обраного порогу, а також є типовою метрикою для задач детекції викидів.

Таблиця 3.2 відображає результати порівняння класичних (нерозподілених) методів HBOS, LODA, LSCP і XGBOD із їхніми розподіленими реалізаціями, отримані за метрикою ROC-AUC на п’яти еталонних наборах даних (Breast, Shuttle, Satellite, Donors, Census) .

Для класичних алгоритмів таблиця 3.2 показує, що на невеликих наборах (Breast, Shuttle, Satellite) вони демонструють високу або принаймні прийнятну якість: зокрема, HBOS та XGBOD мають дуже високі значення ROC-AUC на Breast і Shuttle, а XGBOD на Shuttle досягає ідеального результату. Водночас для складніших або більш розмірних задач (наприклад, Satellite) якість окремих класичних методів знижується, що свідчить про залежність ефективності від структури даних та розподілу ознак.

Таблиця 3.2 – Значення показника ROC-AUC для класичних алгоритмів HBOS, LODA, LSCP і XGBOD у порівнянні з їхніми розподіленими версіями

Набір даних / Алгоритм	Breast	Shuttle	Satellite	Donors	Census
Класичний HBOS	0.9910	0.9855	0.7581	0.7187	0.6333
Класичний LODA	0.9866	0.9920	0.6114	0.4889	0.4449
Класичний LSCP	0.7845	0.5551	0.6106	0.7828	–
Класичний XGBOD	0.9916	1.0000	0.9685	–	–
Розподілений HBOS (стат.)	0.9720	0.9810	0.7900	0.7120	0.6060
Розподілений HBOS (динам.)	0.9510	0.5630	0.6850	0.4930	0.5740
Розподілений LODA	0.9740	0.9820	0.7150	0.8010	0.6210
Розподілений LSCP	0.9690	0.9790	0.7280	0.7920	0.5980
Розподілений XGBOD	0.9870	0.9920	0.9150	0.7210	0.6040

Окремо важливим у таблиці є відображення обмежень масштабованості класичних реалізацій на великих наборах Donors і Census [38, 39]. Для частини методів наведені пропуски (позначено «-»), що означає неможливість завершити обчислення в прийнятний час або в межах доступних ресурсів, тобто фактичну непридатність класичних реалізацій для відповідних сценаріїв великих даних. Така поведінка узгоджується з практикою застосування алгоритмів детекції викидів: методи, які потребують значних обчислювальних витрат або формують складні внутрішні структури, швидко втрачають застосовність із ростом обсягу даних чи кількості ознак.

Розподілені версії з пакета демонструють іншу картину. По-перше, на невеликих наборах (Breast, Shuttle, Satellite) вони забезпечують значення ROC-AUC, співмірні з класичними алгоритмами, тобто перехід до розподіленої реалізації не призводить до принципової втрати якості. По-друге, на великих наборах Donors та Census усі розподілені методи завершують обробку і дають вимірювані результати, що підтверджує їхню практичну масштабованість у нашому обчислювальному середовищі. У межах розподілених підходів найвищі значення ROC-AUC на великих наборах демонструє LODA, що узгоджується з

його властивостями: використання множини випадкових проєкцій і простих одномірних оцінок добре масштабується та зберігає чутливість до нетипових спостережень. LSCP, як ансамблевий метод, забезпечує близькі до LODA результати на великих наборах, що підтверджує ефект підсилення базового детектора через локально-селективне комбінування. HBOS у статичному режимі демонструє стабільні, але дещо нижчі результати, тоді як динамічний варіант HBOS у таблиці показує помітно слабшу якість у деяких наборах, що може бути пов’язано з особливостями квантильного формування меж бінів та чутливістю до розподілу даних при великій неоднорідності. XGBOD демонструє конкурентну якість на малих наборах і прийнятні результати на великих, однак загалом поступається LODA і LSCP у сценаріях з великим обсягом даних, що може бути наслідком обмеженого набору TOS та менш гнучкого підбору гіперпараметрів у наших умовах.

У підсумку таблиця 3.3 підтверджує два ключові висновки: по-перше, запропоновані розподілені реалізації зберігають узгоджену якість із класичними методами на наборах стандартного розміру; по-друге, саме розподілені реалізації забезпечують практичну придатність на великих наборах даних, де класичні алгоритми частково не здатні завершити обчислення в межах доступних ресурсів.

3.3 Аналіз впливу параметрів на продуктивність алгоритмів

Ефективність алгоритмів виявлення аномалій у середовищі великих даних визначається не лише їхньою теоретичною здатністю знаходити нетипові спостереження, але й тим, наскільки стабільно та швидко ці методи можуть працювати в розподіленій інфраструктурі. Тому при переході до слабшого обчислювального кластера ключовим стає добір параметрів, що забезпечує прийнятний компроміс між якістю детекції та ресурсними обмеженнями. Нижче розглянуто вплив основних параметрів для кожного з реалізованих методів.

Алгоритм HBOS базується на побудові гістограм для кожної ознаки та оцінюванні “нетиповості” спостереження через щільність потрапляння у

відповідні інтервали. Визначальним параметром є кількість бінів гістограми n_bins, яка прямо впливає на роздільну здатність оцінювання розподілу. Зменшення n_bins знижує обчислювальне навантаження, оскільки зменшується розмір гістограм і кількість операцій при обчисленні внеску кожної ознаки. Разом з тим, грубіша дискретизація розподілу призводить до того, що частина локальних відмінностей між нормальними та аномальними значеннями “згладжується”, що може зменшувати чутливість методу до слабко виражених аномалій.

Окремий вплив має вибір стратегії формування меж бінів – статичної або динамічної. Статична стратегія задає межі інтервалів рівномірно між мінімальним та максимальним значеннями ознаки, що є швидким і простим підходом, однак він є чутливим до асиметрії та викидів у розподілі. Динамічна стратегія формує межі за квантилями, тобто враховує фактичний розподіл даних, забезпечуючи більш збалансоване заповнення інтервалів. У практичних сценаріях із сенсорними даними, де розподіли часто є зміщеними та мають “важкі хвости”, динамічний варіант зазвичай демонструє кращу якість ранжування аномалій, хоча може вимагати додаткових витрат на сортування або квантування значень.

LODA реалізує ансамблевий принцип, де “сильний” детектор формується з множини дуже простих одномірних оцінювань, отриманих після випадкових проєкцій багатовимірних даних. Визначальними параметрами є кількість проєкцій k та кількість бінів n_bins для гістограм у проєкційному просторі.

Параметр k безпосередньо відповідає за ступінь ансамблювання: більша кількість проєкцій підвищує ймовірність того, що аномальний об’єкт проявить себе як “нетиповий” у значній частині проєкцій. Це збільшує стабільність оцінки та зменшує залежність від випадкового вибору напрямків. Водночас збільшення k лінійно підвищує обчислювальну складність, оскільки необхідно обчислювати більше проєкцій та будувати більше гістограм.

Параметр n_bins визначає точність апроксимації одномірних розподілів. Зменшення n_bins прискорює обчислення й знижує використання пам’яті, але робить оцінювання щільності грубішим, що може погіршувати розділення

близьких за поведінкою об’єктів. Таким чином, на слабшому кластері зменшення k і n_bins знижує навантаження, але водночас може призвести до менш стабільних оцінок аномальності, особливо у випадках слабко виражених відхилень.

Алгоритм LSCP є ансамблевим методом, який поєднує результати кількох базових детекторів із локально-селективним відбором найбільш “компетентних” детекторів у кожній області даних. Тому параметри впливають як на різноманіття ансамблю, так і на якість локальної адаптації.

Ключовим є параметр n_base_detectors, який визначає кількість базових моделей. Зменшення цієї кількості скорочує час виконання та обсяг проміжних даних, однак зменшує різноманіття оцінок, що є одним із головних джерел переваги ансамблю. Відповідно, при меншій кількості детекторів зменшується потенціал підсилення базового методу.

Другий суттєвий параметр – кількість кластерів n_clus та метод кластеризації, що формують локальні області. Менша кількість кластерів зменшує витрати на кластеризацію та подальшу обробку партицій, однак робить локальні області більш “грубими”, що може погіршити здатність алгоритму підлаштовуватися до різних режимів роботи системи. Також на стабільність виконання в Spark впливають параметри max_size та min_size, які використовуються для балансування розмірів партицій: вони зменшують ризик перекосів навантаження між виконавцями та скорочують “хвости” виконання стадій, що є критичним для слабшої інфраструктури.

XGBOD є частково контрольованим методом, який розширює початковий простір ознак додатковими трансформованими оцінками викидів (TOS) та навчає розподілену модель XGBoost. Тому якість детекції тут суттєво залежить від двох факторів: якості сформованих додаткових ознак та ефективності навчання моделі.

Параметр n_TOS визначає, скільки неконтрольованих детекторів формують додаткові оцінки. Зменшення n_TOS знижує витрати на побудову TOS та скорочує проміжні обчислення, проте зменшує обсяг додаткової інформації, яку отримує XGBoost. Параметр n_selected_TOS визначає, скільки з

цих оцінок буде фактично включено до розширеного набору ознак; його зменшення спрощує модель і пришвидшує навчання, однак може обмежити здатність XGBoost відділяти аномалії від норми.

Суттєву роль відіграє стратегія відбору TOS. Якщо застосовується відбір на основі точності, то він залежить від порогування оцінок і від якості “псевдорозмітки”, яка може вносити шум. Додатково на слабшому кластері часто обмежується кількість спроб оптимізації гіперпараметрів, що зменшує ймовірність знаходження найкращої конфігурації XGBoost, а отже – може погіршувати якість фінального рішення.

Висновки до розділу 3

1. Проведено експериментальну перевірку прогнозної здатності розроблених алгоритмів у реалістичних та еталонних умовах. Це було зроблено шляхом застосування запропонованого модуля до промислового набору великих даних, де оцінювалася здатність виявляти аномалії перед відмовою обладнання, а також до еталонних наборів даних різного масштабу для порівняння з класичними аналогами.

2. Оцінювання якості проводилося за метрикою ROC-AUC. Це дало змогу емпірично підтвердити, що запропоновані розподілені методи ефективно виявляють аномалії на великих даних, значно перевершуючи класичні аналоги, які не змогли обробити великомасштабні набори. Було встановлено, що динамічна версія HBOS є найбільш точною для промислової задачі, що валідувало коректність розроблених архітектур.

ВИСНОВКИ

1. Проведено аналіз концепції виявлення аномалій та її ролі у сучасних інформаційних системах. Це було зроблено шляхом визначення терміну "аномалія", систематизації її типів та класифікації основних категорій методів їх виявлення.

2. Досліджено специфіку обробки великих даних та обґрунтовано вибір технологічного стеку. Це було реалізовано через аналіз викликів, які ставить феномен Big Data перед класичними алгоритмами, та розгляд парадигми розподілених обчислень MapReduce і фреймворку Apache Spark як її сучасної реалізації. Було деталізовано ключові примітиви Spark, що є інструментами для побудови розподілених алгоритмів.

3. Систематизовано наукову проблему неконтрольованого виявлення аномалій у середовищі великих даних. Це було досягнуто шляхом поєднання висновків щодо теоретичних обмежень неконтрольованих методів та практичних проблем обробки великих даних. Було встановлено, що наявні рішення є або недостатньо масштабованими, або вузькоспеціалізованими, що створює потребу в універсальних та ефективних розподілених інструментах.

4. Розроблено розподілені архітектури для базових алгоритмів виявлення аномалій, що усунуло їхні фундаментальні обмеження щодо масштабованості. Це було досягнуто шляхом заміни їхніх послідовних, ітераційних етапів на високопродуктивні паралельні операції, що надаються фреймворком Apache Spark. Зокрема, для HBOS було впроваджено розподілену побудову гістограм, а для LODA – заміну множинних ітераційних проєкцій на єдину розподілену матричну операцію. Це дало змогу перетворити класичні, але не масштабовані методи, на повноцінні інструменти для Big Data, здатні обробляти великі масиви даних за прийнятний час, зберігаючи при цьому вихідну логіку оцінки аномальності.

5. Розв’язано ключову проблему обчислювальної складності у складних ансамблевих методах, зокрема в LSCP, шляхом інноваційної модифікації його архітектури. Це було реалізовано через запропоновану заміну

ресурсоємного та непрактичного для великих даних методу k-найближчих сусідів на ефективну розподілену кластеризацію для формування "локальних областей". Це дало змогу створити алгоритм LSCP – першу масштабовану версію потужного ансамблевого методу, що доводить можливість адаптації навіть складних, залежних від сусідства підходів до середовища Big Data шляхом апроксимації обчислювально дорогих етапів.

6. Сформовано комплексний набір інструментів для виявлення аномалій, що охоплює як неконтрольовані, так і гібридні підходи. Це було здійснено через послідовну розробку та формальний опис алгоритмів, кожен з яких представляє окрему категорію методів – від простих статистичних до складних ансамблевих та частково контрольованих.

7. Проведено експериментальну перевірку прогнозної здатності розроблених алгоритмів у реалістичних та еталонних умовах. Це було зроблено шляхом застосування запропонованого модуля до промислового набору великих даних, де оцінювалася здатність виявляти аномалії перед відмовою обладнання, а також до еталонних наборів даних різного масштабу для порівняння з класичними аналогами.

8. Оцінювання якості проводилося за метрикою ROC-AUC. Це дало змогу емпірично підтвердити, що запропоновані розподілені методи ефективно виявляють аномалії на великих даних, значно перевершуючи класичні аналоги, які не змогли обробити великомасштабні набори. Було встановлено, що динамічна версія HBOS є найбільш точною для промислової задачі, що валідувало коректність розроблених архітектур.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aggarwal, C.C. Outlier Analysis. Springer Publishing Company. 2016.

2. Aguilera-Martos, I., García-Barzana, M., García-Gil, D., Carrasco, J., López, D., Luengo, J., Herrera, F. Multi-step histogram based outlier scores for unsupervised anomaly detection: ArcelorMittal engineering dataset case of study. Neurocomputing. 2023. Vol. 544. № 126228.

3. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M. Optuna: a next-generation hyperparameter optimization framework. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2019.

4. Ariyaluran Habeeb, R.A., Nasaruddin, F., Gani, A., Amanullah, M.A., Abaker Targio Hashem, I., Ahmed, E., Imran, M. Clustering-based real-time anomaly detection – a breakthrough in big data technologies. Transactions on Emerging Telecommunications Technologies. 2022. Vol. 33. № e3647.

5. Arjunan, T. Real-time detection of network traffic anomalies in big data environments using deep learning models. International Journal of Research in Applied Science and Engineering Technology. 2024. Vol. 12. Pp. 10–22214.

6. Breunig, M.M., Kriegel, H.-P., Ng, R.T., Sander, J. LOF: identifying density-based local outliers. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. 2000. Pp. 93–104.

7. Carrasco, J., López, D., Aguilera-Martos, I., García-Gil, D., Markova, I., Garcia-Barzana, M., Arias-Rodil, M., Luengo, J., Herrera, F. Anomaly detection in predictive maintenance: a new evaluation framework for temporal unsupervised anomaly detection algorithms. Neurocomputing. 2021. Vol. 462. Pp. 440–452.

8. Cavallaro, C., Cutello, V., Pavone, M., Zito, F. Discovering anomalies in big data: a review focused on the application of metaheuristics and machine learning techniques. Frontiers in Big Data. 2023. Vol. 6. № 1179625.

9. Chalapathy, R., Chawla, S. Deep learning for anomaly detection: a survey. arXiv. 2019. arXiv:1901.03407.

10. Chandola, V., Banerjee, A., Kumar, V. Anomaly detection: a survey. ACM Computing Surveys. 2009. Vol. 41. Pp. 1–58.

11. Chen, T., Guestrin, C. XGBoost: a scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ‘16). 2016. Pp. 785–794.

12. Chen, Z., Li, Z., Chen, X., Chen, X., Fan, H., Hu, R. Rectifying inaccurate unsupervised learning for robust time series anomaly detection. Information Sciences. 2024. № 120222.

13. Dean, J., Ghemawat, S. MapReduce: simplified data processing on large clusters. OSDI’04: Proceedings of the 6th Symposium on Operating Systems Design and Implementation. 2004.

14. Dong, W., Woźniak, M., Wu, J., Li, W., Bai, Z. Denoising aggregation of graph neural networks by using principal component analysis. IEEE Transactions on Industrial Informatics. 2022. Vol. 19. Pp. 2385–2394.

15. Dwivedi, R.K., Kumar, R., Buyya, R. A novel machine learning-based approach for outlier detection in smart healthcare sensor clouds. International Journal of Healthcare Information Systems and Informatics. 2021. Vol. 16. Pp. 1–26.

16. Erfani, S.M., Rajasegarar, S., Karunasekera, S., Leckie, C. High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. Pattern Recognition. 2016. Vol. 58. Pp. 121–134.

17. Erhan, L., Ndubuaku, M., Di Mauro, M., Song, W., Chen, M., Fortino, G., Bagdasar, O., Liotta, A. Smart anomaly detection in sensor systems: a multi-perspective review. Information Fusion. 2021. Vol. 67. Pp. 64–79.

18. García-Gil, D., García, S., Xiong, N., Herrera, F. Smart data driven decision trees ensemble methodology for imbalanced big data. Cognitive Computation. 2024. Pp. 1–17.

19. García-Gil, D., Luque-Sánchez, F., Luengo, J., García, S., Herrera, F. Enabling smart data: noise filtering in big data classification. Information Sciences. 2019. Vol. 479. Pp. 135–152.

20. García-Gil, D., Luque-Sánchez, F., Luengo, J., García, S., Herrera, F. From big to smart data: iterative ensemble filter for noise filtering in big data classification. International Journal of Intelligent Systems. 2019. Vol. 34. Pp. 3260–3274.

21. Goldstein, M., Dengel, A. Histogram-based outlier score (HBOS): a fast unsupervised anomaly detection algorithm. KI-2012: Poster and Demo Track. 2012. Pp. 59–63.

22. Goldstein, M., Uchida, S. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. PLoS ONE. 2016. Vol. 11. № e0152173.

23. Habeeb, R.A.A., Nasrullah, F., Gani, A., Hashem, I.A.T., Ahmed, E., Imran, M. Real-time big data processing for anomaly detection: a survey. International Journal of Information Management. 2019. Vol. 45. Pp. 289–307.

24. Han, S., Hu, X., Huang, H., Jiang, M., Zhao, Y. ADBench: anomaly detection benchmark. Advances in Neural Information Processing Systems. 2022. Vol. 35. Pp. 32142–32159.

25. Hanley, J.A., McNeil, B.J. The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology. 1982. Vol. 143. Pp. 29–36.

26. He, Z., Xu, X., Deng, S. Discovering cluster-based local outliers. Pattern Recognition Letters. 2003. Vol. 24. Pp. 1641–1650.

27. Hela, S., Amel, B., Badran, R. Early anomaly detection in smart home: a causal association rule-based approach. Artificial Intelligence in Medicine. 2018. Vol. 91. Pp. 57–71.

28. Hilal, W., Gadsden, S.A., Yawney, J. Financial fraud: a review of anomaly detection techniques and recent advances. Expert Systems with Applications. 2022. Vol. 193. № 116429.

29. Karau, H., Konwinski, A., Wendell, P., Zaharia, M. Learning Spark: Lightning-Fast Big Data Analysis. O'Reilly Media. 2015.

30. Kilincer, I.F., Ertam, F., Sengur, A. Machine learning methods for cyber security intrusion detection: datasets and comparative study. Computer Networks. 2021. Vol. 188. № 107840.

31. Kim, B., Alawami, M.A., Kim, E., Oh, S., Park, J., Kim, H. A comparative study of time series anomaly detection models for industrial control systems. Sensors. 2023. Vol. 23. № 1310.

32. Kraljevski, I., Duckhorn, F., Tschöpe, C., Wolff, M. Machine learning for

anomaly assessment in sensor networks for NDT in aerospace. IEEE Sensors Journal. 2021. Vol. 21. Pp. 11000–11008.

33. Laskar, M.T.R., Huang, J.X., Smetana, V., Stewart, C., Pouw, K., An, A., Chan, S., Liu, L. Extending isolation forest for anomaly detection in big data via k-means. ACM Transactions on Cyber-Physical Systems. 2021. Vol. 5.

34. López, D., Aguilera-Martos, I., García-Barzana, M., Herrera, F., García-Gil, D., Luengo, J. Fusing anomaly detection with false positive mitigation methodology for predictive maintenance under multivariate time series. Information Fusion. 2023. Vol. 100. № 101957.

35. Luengo, J., García-Gil, D., Ramírez-Gallego, S., García, S., Herrera, F. Big Data Preprocessing – Enabling Smart Data. Springer. 2020.

36. Nassif, A.B., Talib, M.A., Nasir, Q., Dakalbab, F.M. Machine learning for anomaly detection: a systematic review. IEEE Access. 2021. Vol. 9. Pp. 78658–78700.

37. Oprea, S.-V., Bâra, A., Puican, F.C., Radu, I.C. Anomaly detection with machine learning algorithms and big data in electricity consumption. Sustainability. 2021. Vol. 13.

38. Pang, G., Shen, C., Cao, L., van den Hengel, A. Deep learning for anomaly detection: a review. ACM Computing Surveys. 2021. Vol. 54. Pp. 1–38.

39. Pang, G., Shen, C., van den Hengel, A. Deep anomaly detection with deviation networks. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019. Pp. 353–362.

40. Pevný, T. LODA: lightweight on-line detector of anomalies. Machine Learning. 2016. Vol. 102. Pp. 275–304.

41. Ramírez-Gallego, S., Fernández, A., García, S., Chen, M., Herrera, F. Big data: tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce. Information Fusion. 2018. Vol. 42. Pp. 51–61.

42. Rathore, M.M., Ahmad, A., Paul, A. Real time intrusion detection system for ultra-high-speed big data environments. The Journal of Supercomputing. 2016. Vol. 72. Pp. 3489–3510.

43. Rettig, L., Khayati, M., Cudré-Mauroux, P., Piórkowski, M. Online anomaly detection over big data streams. In: Applied Data Science. Springer. 2019. Pp.

289–312.

44. Roberts, E., Bassett, B.A., Lochner, M. Bayesian anomaly detection and classification for noisy data. In: Intelligent Systems Design and Applications. Springer International Publishing, Cham. 2021. Pp. 426–435.

45. Thudumu, S., Branch, P., Jin, J., Singh, J.J. A comprehensive survey of anomaly detection techniques for high dimensional big data. Journal of Big Data. 2020. Vol. 7. Pp. 1–30.

46. Woźniak, M., Wieczorek, M., Siłka, J. BiLSTM deep neural network model for imbalanced medical data of IoT systems. Future Generation Computer Systems. 2023. Vol. 141. Pp. 489–499.

47. Zhao, Y., Hryniewicki, M.K. XGBOD: improving supervised outlier detection with unsupervised representation learning. 2018 International Joint Conference on Neural Networks (IJCNN). 2018. Pp. 1–8.

48. Zhao, Y., Nasrullah, Z., Li, Z. LSCP: locally selective combination in parallel outlier ensembles. Proceedings of the 2019 SIAM International Conference on Data Mining (SDM 2019). 2019. Pp. 585–593.

49. Zhao, Y., Nasrullah, Z., Li, Z. PyOD: a Python toolbox for scalable outlier detection. Journal of Machine Learning Research. 2019. Vol. 20. Pp. 1–7.

50. Галин В., Каравець Р., Сичов Р. Інтелектуальні методи аналізу великих даних: виявлення аномалій, аналіз настроїв та прогнозування якості в інтелектуальному виробництві. Collection of Scientific Papers with Proceedings of the 2nd International Scientific and Practical Conference. International Scientific Unity. November 26-28, 2025. С. 310–313.

51. Галин В. Методи динамічного та статичного виявлення аномалій у великих даних. Збірник тез доповідей ІІ Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Інтелектуальні комп’ютерні системи та мережі» (ІКСМ осінь 2025), м. Тернопіль, ЗУНУ, 20 травня 2025 р. Тернопіль, 2025. С. 39–42.

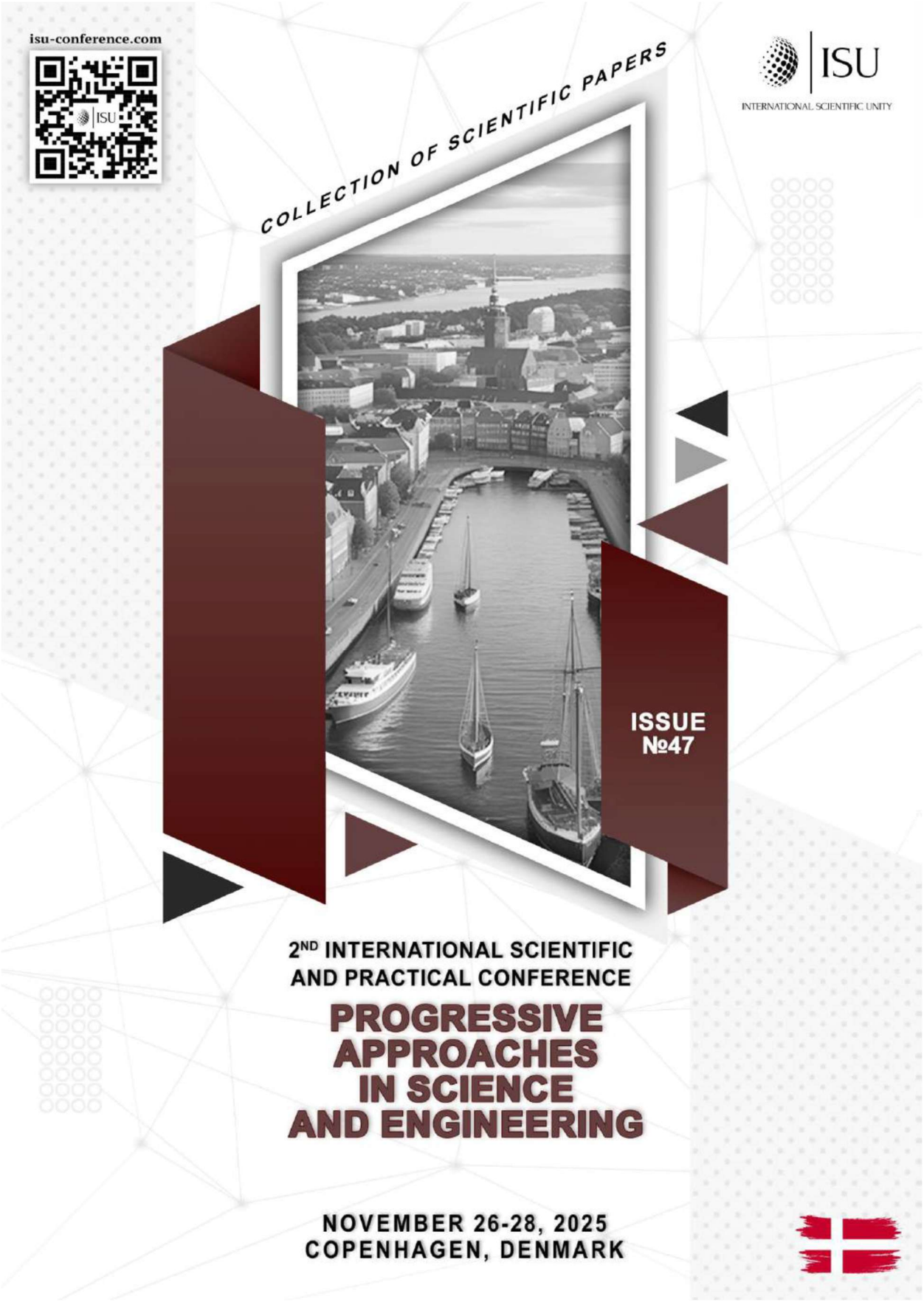
52. Островерхов В.М., Біловус Л.І., Возьний К.З., Луцишин О.О., Монастирський Г.Л., Надвиничний С.А., Питель С.В., Шандрук С.К. Загальні методичні рекомендації з підготовки, оформлення, захисту та оцінювання

кваліфікаційних робіт здобувачів вищої освіти першого (бакалаврського) і другого (магістерського) рівнів / Укладачі: Тернопіль: ЗУНУ, 2024. 83 с.

53. Комар М.П., Саченко А.О., Васильків Н.М., Загородня Д.І. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за другим (магістерським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 32 с.

Додаток А

Копії публікацій



2nd International Scientific and Practical Conference

«Progressive Approaches in Science and Engineering»

Collection of Scientific Papers

November 26-28, 2025
Copenhagen, Denmark

UDC 001(08)

Progressive Approaches in Science and Engineering: Collection of Scientific Papers with Proceedings of the 2nd International Scientific and Practical Conference. International Scientific Unity. November 26-28, 2025. Copenhagen, Denmark. 697 p.

ISBN 979-8-89704-979-0 (series)
DOI 10.70286/ISU-26.11.2025

The conference is included in the Academic Research Index ReserchBib International catalog of scientific conferences.

The collection of scientific papers presents the materials of the participants of the 2nd International Scientific and Practical Conference "Progressive Approaches in Science and Engineering" (November 26-28, 2025. Copenhagen, Denmark).

The materials of the collection are presented in the author's edition and printed in the original language. The authors of the published materials bear full responsibility for the authenticity of the given facts, proper names, geographical names, quotations, economic and statistical data, industry terminology, and other information.

ISBN 979-8-89704-979-0

ISU
INTERNATIONAL SCIENTIFIC UNITY

Official site: https://isu-conference.com/

ІНТЕЛЕКТУАЛЬНІ МЕТОДИ АНАЛІЗУ ВЕЛИКИХ ДАНИХ: ВИЯВЛЕННЯ АНОМАЛІЙ, АНАЛІЗ НАСТРОЇВ ТА ПРОГНОЗУВАННЯ ЯКОСТІ В ІНТЕЛЕКТУАЛЬНОМУ ВИРОБНИЦТВІ

Галин Василь
здобувач вищої освіти
Каравець Роман
здобувач вищої освіти
Сичов Руслан
здобувач вищої освіти
Кафедра інформаційно-обчислювальних систем і управління
Західноукраїнський національний університет, Україна

Стрімкий розвиток концепції «Індустрія 4.0», кіберфізичних систем, Інтернету речей та платформ соціальних медіа призводить до вибухового зростання обсягів, швидкості та різноманітності даних. Це, з одного боку, відкриває можливості для глибокої аналітики, а з іншого – створює суттєві виклики для виявлення аномальної поведінки, моніторингу суспільних настроїв і проактивного забезпечення якості продукції [1–4].

У середовищі великих даних традиційні методи виявлення аномалій часто не масштабуються, оскільки розраховані на помірні обсяги та вимагають повної або частково розміченої вибірки, що практично недосяжно для реальних потоків даних.

У сфері аналізу настроїв платформи на кшталт Twitter генерують величезні потоки коротких повідомлень, які потрібно обробляти в режимі, наближеному до реального часу. Класичні пакетні ETL-процеси виявляються малопридатними для подібних сценаріїв, що зумовлює перехід до потокового ETL і спеціалізованих систем потокової обробки даних.

Паралельно виникає потреба у побудові інтелектуальних систем прогнозування якості в інтелектуальному виробництві, які здатні працювати з багатовимірними часовими рядами, враховувати складні взаємозв'язки між технологічними параметрами й результатами контролю якості та підтримувати перехід від «постфактум» контролю до проактивного керування якістю.

Отже, актуальною є розробка узгодженого комплексу методів, що поєднує розподілене виявлення аномалій, потоковий аналіз тональності та прогнозування якості у спільній парадигмі інтелектуальної обробки великих даних.

У галузі виявлення аномалій виділяють контрольовані, частково контрольовані та неконтрольовані підходи, причому для великих даних найтиповішим є саме неконтрольований сценарій. Відомі методи поділяють на підходи на основі найближчих сусідів, кластеризації, статистичні моделі, а також ансамблеві методи, що комбінують кілька базових детекторів [5–8]. Однак більшість реалізацій орієнтовані на послідовну обробку та не враховують

специфіку розподілених обчислень у кластерному середовищі, що обмежує їх застосування для терабайтних наборів даних.

У сфері аналізу тональності накопичено значний досвід використання як класичних моделей машинного навчання, так і сучасних глибинних та трансформерних архітектур для класифікації тональності текстів [9–11]. Для Twitter-орієнтованого аналізу тональності широко використовуються етапи надходження даних, попередньої обробки, виділення ознак і класифікації. Разом з тим, доступні рішення здебільшого або працюють у пакетному режимі, або не розглядають повноцінну архітектуру потокового ETL з урахуванням вибору сховища, інтервалів запуску й місця виконання класифікації.

Щодо інтелектуального виробництва, у літературі детально описано загальні принципи «розумних фабрик», використання великих даних та методів машинного й глибокого навчання для прогнозування різних техніко-економічних показників [12–15]. Проте комплексні моделі, орієнтовані саме на прогнозування інтегральних показників якості продукції з урахуванням багатовимірних виробничих даних, структурованої архітектури джерел, оброблення й прикладних сервісів, залишаються менш опрацьованими.

Таким чином, у кожному з трьох напрямів існують суттєві науково-практичні прогалини, пов'язані з масштабованістю, потоковою обробкою, інтеграцією різнорідних джерел і проактивним управлінням.

Розподілені методи неконтрольованого виявлення аномалій у великих даних. Розроблено чотири розподілені модифікації алгоритмів виявлення аномалій: HBOS_BD (гістограмний аналіз), LODA_BD (легкий онлайн-детектор на основі випадкових проєкцій), LSCP_BD (локально-селективне поєднання ансамблів детекторів) та XGBOD_BD (частково контрольований підхід, заснований на XGBoost). Основною ідеєю є перенесення обчислювально складних операцій у простір примітивів Apache Spark: паралельна побудова гістограм, розподілені матричні проєкції, формування «локальних областей» у кластерному режимі, навчання допоміжних класифікаторів на розподілених даних.

Структура алгоритмів спроєктована так, щоб мінімізувати міжвузлові комунікації та витрати на передачу даних, а також забезпечити роботу з нерозміченими наборами даних, де частка аномалій є невідомою. Експериментальні дослідження на еталонних наборах і реальному великомасштабному датасеті продемонстрували прийнятну якість виявлення (ROC-AUC на рівні класичних реалізацій) при суттєвому скороченні часу обчислень та хорошій масштабованості за кількістю вузлів кластера.

Потоковий аналіз тональності твітів на основі технологій великих даних. Розроблено архітектуру потокового фреймворку для аналізу тональності твітів. Запропоновано конвеєр Extract–Transform–Load у потоковому виконанні, що включає:

1. Рівень надходження даних. Твіти отримуються через API та надходять до системи повідомлень Apache Kafka, яка забезпечує буферизацію й надійну передачу даних до наступних шарів.

2. Рівень оброблення. Потокова обробка реалізована за допомогою Spark Structured Streaming, який виконує очищення, нормалізацію, фільтрацію, агрегації та, за одним з варіантів, класифікацію тональності «на льоту».

3. Класифікація тональності. Застосовано багатомовну трансформерну модель сімейства XLM/Twitter-XLM-R, що дозволяє відносити твіти до позитивної, нейтральної чи негативної тональності. Класифікація може виконуватися як у потоці (in-stream inference), так і після завантаження у сховище (post-load inference), що дає змогу балансувати між латентністю та обчислювальними витратами.

4. Рівень зберігання та візуалізації. Для зберігання результатів використано декілька технологій (Cassandra, HBase, Hive, HDFS), що дозволило провести порівняльний аналіз продуктивності. Геопросторова візуалізація реалізована на основі бібліотек Geopy та Folium, із можливістю фільтрації за мовою, тематикою та часовими інтервалами.

Експериментальні результати показали, що запропонований фреймворк забезпечує стабільну обробку потоків твітів у режимі, наближеному до реального часу, а також можливість масштабування як по даних, так і по кількості обчислювальних вузлів.

Модель прогнозування якості в інтелектуальному виробництві. Розроблено трирівневу модель прогнозування якості в інтелектуальному виробництві та інтегровану систему її оцінювання:

1. Шар джерел даних включає конструкторську та технологічну документацію, дані сенсорів і систем моніторингу, результати контролю якості та експлуатаційні показники.

2. Шар оброблення даних реалізує інтеграцію, очищення, трансформацію, зберігання великих масивів даних, а також побудову прогнозної моделі. Як модель обрано нейронну мережу типу ELM (Extreme Learning Machine), параметри якої (ваги та пороги) додатково оптимізуються методом рою частинок, що підвищує точність прогнозу.

3. Прикладний шар забезпечує обчислення інтегральних індексів якості (продуктивність, надійність, строк служби, економічність тощо), їх візуалізацію, формування рекомендацій щодо налаштування технологічних режимів та оцінювання рівня зрілості інтелектуального виробництва.

Тестування моделі на наближених до реальних виробничих даних продемонструвало, що запропонований підхід дозволяє отримувати прогнози якості з прийнятною точністю до завершення повного виробничого циклу, що створює передумови для переходу до проактивного управління якістю.

Таким чином, на основі трьох взаємодоповнюючих напрямів – розподіленого виявлення аномалій, потокового аналізу тональності твітів і прогнозування якості продукції – сформовано комплексну методологію інтелектуального аналізу великих даних для сучасних кіберфізичних та виробничих систем.

Перспективним напрямом подальших досліджень є інтеграція розроблених модулів в єдину платформу інтелектуального моніторингу, де підсистеми

виявлення аномалій, аналізу настроїв та прогнозування якості взаємодіятимуть через спільну інфраструктуру великих даних, підтримуючи сценарії предиктивного обслуговування, адаптивного керування виробництвом і комплексної аналітики соціально-технічних систем.

Список використаних джерел

1. Aggarwal, C. C. (2016). Outlier analysis. Springer.
2. Chalapathy, R., & Chawla, S. (2019). Deep learning for anomaly detection: A survey. arXiv preprint, arXiv:1901.03407.
3. Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), 1–58.
4. Erhan, L., Ndubuaku, M., Di Mauro, M., Song, W., Chen, M., & Forti, M. (2021). A multi-perspective review. Information Fusion, 67, 64–79.
5. Pevný, T. (2016). LODA: Lightweight on-line detector of anomalies. Machine Learning, 102, 275–304.
6. Zhao, Y., & Hryniewicki, M. K. (2018). XGBOD: Improving supervised outlier detection with unsupervised representation learning. In Proceedings of the International Joint Conference on Neural Networks (IJCNN) (pp. 1–8).
7. Zhao, Y., Nasrullah, Z., & Li, Z. (2019). LSCP: Locally selective combination of parallel outlier detectors. In Proceedings of the SIAM International Conference on Data Mining (SDM) (pp. 585–593).
8. Bifet, A., & Frank, E. (2010). Sentiment knowledge discovery in Twitter streaming data. In Discovery Science (pp. 1–15). Springer.
9. Go, A., Bhayani, R., & Huang, L. (2009). Twitter sentiment classification using distant supervision. CS224N Project Report. Stanford University.
10. Barbieri, F., Espinosa Anke, L., & Camacho-Collados, J. (2022). XLM-T: Multilingual language models for Twitter. In Proceedings of the Thirteenth Language Resources and Evaluation Conference (pp. 258–266). European Language Resources Association.
11. Gorawski, M., & Gorawska, A. (2014). Research on the stream ETL process. In A. Abraham, A. Muda, & Y. H. Choo (Eds.), Computational science and its applications – ICCSA 2014. Big data in complex systems (pp. 61–71). Springer.
12. Дубницький, В. І., & Захарченко, В. І. (2024). Формування сучасного високотехнологічного промислового підприємства на засадах концепції «Індустрія 4.0». Economics: Time Realities, 6(76).
13. Li, B. H. (2017). Applications of AI in intelligent manufacturing: A review. Journal of Control and Decision, 18(1), 86–96.
14. Zhou, J. (2019). Human–cyber–physical systems (HCPSs) in the context of new-generation intelligent manufacturing. Engineering, 5(4), 624–636.
15. Yang, H. (2019). The Internet of Things for smart manufacturing: A review. IISE Transactions, 51(11), 1190–1216.

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

WEST UKRAINIAN NATIONAL UNIVERSITY

Комп'ютерна Інженерія

III ВСЕУКРАЇНСЬКА НАУКОВО-ПРАКТИЧНА КОНФЕРЕНЦІЯ СТУДЕНТІВ, АСПІРАНТІВ ТА МОЛОДИХ ВЧЕНИХ «ІНТЕЛЕКТУАЛЬНІ КОМП'ЮТЕРНІ СИСТЕМИ ТА МЕРЕЖІ»

ІКСМ осінь 2025

25 ЛИСТОПАДА 2025

Наукове товариство ЗУНУ

KI.WUNU.EDU.UA/CONFERENCE/

ТЕРНОПІЛЬ

2025

ПРОГРАМНИЙ КОМІТЕТ КОНФЕРЕНЦІЇ

Десятнюк О.М., ректор Західноукраїнського національного університету, д-р економічних наук, професор;
Дивак М.П., д-р технічних наук, професор, проректор з наукової роботи ЗУНУ;
Березький О.М., д-р технічних наук, професор, професор кафедри комп'ютерної інженерії Західноукраїнський національний університет;
Семанюк В.З., д-р економічних наук, професор, начальник науково-дослідної частини Західноукраїнського національного університету
Антощук С.Г., д.т.н, професор, Національний університет «Одеська політехніка»;
Баловсяк С.В., д.т.н, професор, Чернівецький національний університет
Бармак О.В., д.т.н., професор, Хмельницький національний університет
Батько Ю.М., к.т.н., доцент, Західноукраїнський національний університет;
Винокурова О.А., д.т.н., професор, Львівський національний університет імені Івана Франка
Возна Н.Я., д.т.н., професор, Західноукраїнський національний університет;
Говорущенко Т.О., д.т.н., професор, Хмельницький національний університет;
Дубчак Л.О., к.т.н., доцент, Західноукраїнський національний університет;
Дунець Р.Б., д.т.н., професор, НУ "Львівська політехніка";
Ізонін І.В., д.т.н., доцент, НУ "Львівська політехніка";
Комар М.П., д.т.н, професор, Західноукраїнський національний університет;
Литвиненко В.І., д.н.т, професор, Херсонський національний технічний університет ;
Лупенко С.А., д.т.н., професор, Опольський технологічний університет, Польща;
Лящинський П.Б., доктор філософії з комп'ютерних наук, НУ "Львівська політехніка" ;
Мельник Г.М., к.т.н, доцент, Західноукраїнський національний університет;
Мельникова Н.І., д.т.н., професор, НУ "Львівська політехніка" ;
Пелешко Д.Д., д.т.н., професор, Львівський національний університет імені Івана Франка;
Піцун О.Й., к.т.н. доцент, Західноукраїнський національний університет;
Сельський П.Р., д.м.н., професор, Тернопільський національний медичний університет імені І. Я. Горбачевського ;
Субботін С.О., д.т.н., професор, Національний університет «Запорізька політехніка» ;
Теслюк В.М., д.т.н., професор, НУ "Львівська політехніка" ;

Тимченко Л.І., д.т.н., професор, Державний університет інфраструктури та технологій;
Цмоць І.Г., д.т.н., професор, НУ "Львівська політехніка" ;
Якименко І.З., к.т.н., доцент, Західноукраїнський національний університет;
Яровий А.А., д.т.н., професор, Вінницький національний технічний університет ;
Яцків В.В., д.т.н., професор, Західноукраїнський національний університет.

ОРГАНІЗАЦІЙНИЙ КОМІТЕТ КОНФЕРЕНЦІЇ

Мельник Г.М. к.т.н., доцент, Західноукраїнський національний університет
Піцун О.Й. к.т.н., доцент, Західноукраїнський національний університет
Фаєрчук В.В. студент, Західноукраїнський національний університет
Галунька Б.В. студент, Західноукраїнський національний університет
Зінькевич О. В. студентка, Західноукраїнський національний університет
Кіт М. О. студентка, Західноукраїнський національний університет

Метою конференції є представлення та обговорення наукових і практичних результатів, сприяння активізації творчої і інноваційної діяльності студентів, аспірантів і молодих вчених.

Конференція проводиться із залученням Ради Молодих Вчених та Студентського Наукового Товариства ФКІТ ЗУНУ.

III Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Інтелектуальні комп'ютерні системи та мережі» (ІКСМ осінь 2025), м. Тернопіль, ЗУНУ, 25 листопада 2025 р. Тернопіль, 2025

Адреса:
Вул. О. Теліги, 8, корпус №6, Тернопіль
URL: https://ki.wunu.edu.ua/conference/
e-mail: kistudconference@gmail.com

Тези доповідей подаються за оригіналом рукопису

ЗМІСТ

Галин В.А.
магістрант 2 курсу ФКІТ ЗУНУ
Науковий керівник к.т.н., доцент Биковий П.Є., кафедра ІОСУ ЗУНУ

МЕТОДИ ДИНАМІЧНОГО ТА СТАТИЧНОГО ВИЯВЛЕННЯ АНОМАЛІЙ У ВЕЛИКИХ ДАНИХ

Вступ. Виявлення аномалій стосується задачі ідентифікації спостережень, що суттєво відрізняються від решти даних. Ці неочікувані патерни зазвичай називають викидами або аномаліями [1]. У більшості випадків дані генеруються одним або кількома процесами, які відображають поведінку системи. Коли така система функціонує некоректно, вона продукує аномалії чи викиди. Ідентифікація таких аномальних спостережень має вирішальне значення для виявлення нетипової поведінки системи. Виявлення аномалій знаходить застосування у широкому спектрі галузей, таких як детекція фінансового шахрайства, виявлення вторгнень, сенсорні мережі, промислові аномалії, охорони здоров'я та ін.

Існує три різні типи завдань у сфері виявлення аномалій [1]:

1. Контрольоване виявлення аномалій. Набір даних є розміченим, вказуючи, які екземпляри є нормальними, а які – аномальними. На цих даних будується прогностична модель для розділення нормальних та аномальних екземплярів.
2. Частково контрольоване виявлення аномалій. Навчальний набір, що використовується, не містить аномальних екземплярів, а лише нормальні спостереження. Аномальні екземпляри надаються в тестовому наборі.
3. Неконтрольоване виявлення аномалій. Екземпляри не є розміченими. Аномальні спостереження невідомі, й алгоритм повинен бути здатним виявляти їх без попередніх знань.

Через автоматизацію збору та зберігання даних більшість реальних завдань з виявлення аномалій належать до неконтрольованого типу. Сценарій неконтрольованого виявлення аномалій є особливо складним, оскільки підходи машинного навчання не мають попередніх знань про дані. Алгоритми неконтрольованого виявлення аномалій оцінюють дані, базуючись виключно на властивостях самого набору даних. Ця оцінка відображає ступінь «аномальності» кожного екземпляра. Потім, використовуючи порогове значення або фіксовану кількість, обираються аномалії. Алгоритми неконтрольованого виявлення аномалій можна згрупувати в чотири категорії: методи на основі найближчих сусідів, методи на основі кластеризації, статистичні методи та ансамблі [2-5].

Автоматизація збору даних, популяризація сенсорів та відсутність людського нагляду, що є характерними для великих даних, підвищили потребу в ефективних методах виявлення аномалій. Природа великих даних у більшості випадків унеможливлює їх розмітку експертом. Ця проблематика призводить до того, що більшість реальних завдань з виявлення аномалій у великих даних є неконтрольованими.

Постановка задачі. Класичні алгоритми виявлення аномалій, розроблені для роботи з даними стандартного обсягу, виявляються неефективними в середовищі великих даних. Їхня обчислювальна складність та ітераційна природа не дозволяють обробляти терабайтні масиви за прийнятний час. Окрім того, більшість реальних наборів Великих даних є нерозміченими, що унеможливлює застосування традиційних контрольованих методів машинного навчання і висуває на передній план задачі неконтрольованого аналізу. Таким чином, розробка та дослідження масштабованих, ефективних та розподілених алгоритмів неконтрольованого виявлення аномалій, адаптованих до парадигми Big Data, є надзвичайно актуальною науково-практичною задачею.

Об'єкт дослідження – процес неконтрольованого виявлення аномалій у великомасштабних наборах даних. Предмет дослідження – розподілені алгоритми виявлення аномалій на основі гістограм, ансамблевих та гібридних підходів, їхня обчислювальна складність, масштабованість та ефективність при реалізації в середовищі Apache Spark. Метою даного дослідження є підвищення ефективності процесу виявлення аномалій у середовищі великих даних шляхом розробки, реалізації та експериментального дослідження набору розподілених алгоритмів

неконтрольованого аналізу.

Основний матеріал. У даному досліджені розглянуто чотири методи, спеціально адаптовані до парадигми великих даних та платформи Apache Spark: HBOS_BD – розподілену модифікацію гістограмного методу виявлення аномалій; LODA_BD – легкий онлайн-детектор, що базується на множині випадкових проєкцій; LSCP_BD – локально-селективне поєднання паралельних ансамблів детекторів; XGBOD_BD – частково контрольований ансамблевий підхід на основі екстремального градієнтного бустингу. Для кожного з методів описуються принципи побудови, особливості розподіленої реалізації та оцінюється обчислювальна складність, що дозволяє обґрунтувати їх придатність до використання у сценаріях із великими та високовимірними наборами даних.

Метод HBOS (Histogram-Based Outlier Score) [2] оцінює аномалії на основі незалежних одновимірних гістограм для кожної ознаки. Для кожного екземпляра p значення оцінки обчислюється як сума логарифмів обернених нормалізованих висот інтервалів, у які він потрапляє:

$$f(x) = \sum_{i=0}^{k} log \frac{1}{hist_i(p)}. \quad (1)$$

Розглядаються два підходи до побудови гістограм:

– статичний – використання k інтервалів фіксованої ширини з оцінкою частоти на основі відносної кількості зразків;

– динамічний – попереднє сортування значень та групування фіксованої кількості N/k послідовних елементів в інтервали. Площа кожного інтервалу однакова, а висота визначається за формулою:

$$\frac{\frac{N}{k}}{last-first}. \quad (2)$$

Основним недоліком класичного HBOS є ітераційна природа, що обмежує його застосування до великих даних. Розподілена модифікація HBOS_BD реалізує побудову гістограм та розрахунок оцінок аномалій за допомогою примітивів Spark, що дозволяє масштабувати обчислення на кластерах. У статичній версії HBOS_BD для кожної ознаки паралельно обчислюються та нормалізуються гістограми, після чого кожному екземпляру призначається бал аномальності за формулою (1). У динамічній версії дані додатково сортуються, визначаються межі інтервалів із фіксованою кількістю елементів та аналогічно обчислюються нормалізовані висоти.

Часова складність оригінального HBOS становить $O(n)$ для фіксованої ширини інтервалів і $O(n\ log\ (n))$ для динамічної ширини. У розподіленому варіанті HBOS_BD складність завдяки розпаралелюванню оцінюється як:

$$\left(\frac{n\ log\ n}{p}\right) + c), \quad (3)$$

де p – кількість паралельних процесорів;

c – комунікаційні накладні витрати.

Метод LODA (Lightweight On-line Detector of Anomalies) [3] ґрунтується на ідеї ансамблю дуже слабких детекторів, кожен з яких будує одновимірну гістограму для проєкції багатовимірних даних на випадковий вектор w_i. Оцінка для зразка x обчислюється як середнє від логарифмів імовірностей за всіма проєкціями:

$$f(x) = \frac{1}{k}\sum_{i=1}^{k} log\ p_i(x^T w_i), \quad (4)$$

де p_i – це ймовірність, оцінена і-ю гістограмою;

k – кількість випадкових проєкцій.

Хоча LODA має низьку часову та просторову складність, його послідовне формування великої кількості проєкцій та відповідних гістограм призводить до значних витрат у середовищі Big Data. Розподілена версія LODA_BD усуває цю проблему шляхом заміни множини ітерацій на одну матричну операцію: формується матриця випадкових проєкцій розміру $d \times k$, а всі проєкції обчислюються одночасно шляхом розподіленого множення $RowMatrix$ даних на цю матрицю. Далі для кожної одновимірної проєкції паралельно будуються й нормалізуються

гістограми, після чого за розподіленою операцією map обчислюється оцінка аномальності відповідно до (4).

Обчислювальна складність LODA_BD в навчанні може бути записана як:

$$O\left(\frac{nkd^{\frac{-1}{2}}}{p}+c\right), \quad (5)$$

n – кількість навчальних зразків;
d – розмірність вхідного простору;
k – кількість гістограм;
p – кількість паралельних процесорів;
c – комунікаційні накладні витрати через синхронізацію між вузлами.

Часова складність етапу класифікації становить:

$$O\left(\frac{kd^{\frac{-1}{2}}}{p}+c\right). \quad (6)$$

Це дозволяє застосовувати LODA_BD у потокових та великих наборах даних.

Метод LSCP (Locally Selective Combination in Parallel Outlier Ensembles) [4] вирішує задачу неконтрольованого виявлення аномалій за відсутності справжньої розмітки. Він формує псевдоеталонні оцінки шляхом агрегування результатів набору базових детекторів (середнє або максимум). Далі для кожного екземпляра оцінюється локальна область за допомогою k-найближчих сусідів у випадково обраних підпросторах ознак, і ті базові детектори, що найбільш узгоджені з псевдоеталонною розміткою в цій локальній області, комбінуються в ансамбль.

Однак багаторазове застосування k-NN у високій розмірності для великих даних вимагає обчислення мільярдів відстаней і є надто затратним. Розподілена модифікація LSCP_BD пропонує апроксимацію локальної області на основі кластеризації замість явного пошуку k-NN. Для цього використовують розподілені реалізації k-Means та Bisecting k-Means у Spark. На першому етапі дані кластеризуються, після чого кластери масштабуються (розбиття занадто великих та об'єднання малих) спеціальною процедурою clusterPartitioning, що формує «збалансовані» партиції з локально подібними екземплярами.

Далі в кожній партиції обчислюється кореляція Пірсона між оцінками базових детекторів та псевдоеталоном, і залежно від стратегії (використання всіх детекторів або лише підмножини з найвищою кореляцією) формується локальне ансамблеве рішення. Такий підхід зменшує кількість пар «екземпляр–сусід» та переносить основні витрати на розподілену кластеризацію.

При використанні Bisecting k-Means загальна розподілена складність LSCP_BD оцінюється як:

$$O\left(\frac{k\cdot n\cdot d}{p}+c\right)+O\left(\frac{n\cdot c_{b,dist}}{p}+c\right)+O\left(\frac{n\cdot B}{p}+c\right), \quad (7)$$

де n – кількість точок даних;
k – кількість кластерів;
d – кількість вимірів;
p – кількість процесорів;
c – комунікаційні накладні витрати через синхронізацію між вузлами.

Метод XGBOD (eXtreme Gradient Boosting Outlier Detection) [5] є частково контрольованим ансамблевим підходом, який поєднує неконтрольовані детектори аномалій із контрольованим класифікатором XGBoost. На першому етапі набір неконтрольованих алгоритмів генерує трансформовані оцінки викидів (Transformed Outlier Scores, TOS), що відображають структуру даних. Відібрані TOS додаються до початкового простору ознак, формуючи розширений набір, на якому навчається XGBoost-класифікатор.

У класичному XGBOD використання «важких» неконтрольованих методів та ітераційного XGBoost обмежує масштабованість на Big Data. Розподілена версія XGBOD_BD замінює базові детектори на розподілені методи виявлення аномалій і використовує розподілену реалізацію XGBoost у Spark. Передбачено дві стратегії відбору TOS: випадкова (rnd) – вибір фіксованої кількості $n\_selected\_TOS$ показників випадковим чином без повторень; на основі точності (acc) – для кожної TOS за допомогою порогу формуються бінарні мітки, обчислюється точність за метриками Spark MulticlassMetrics та обираються TOS з

найкращими значеннями.

Після відбору TOS формуються розширені ознаки, і на них навчається розподілений XGBoost, чиї прогнози і є кінцевим рішенням щодо аномальності екземплярів.

Загальна обчислювальна складність XGBOD_BD складається зі складності розподіленого навчання базових детекторів та XGBoost:

$$O\left(\frac{n \cdot C_{b,dist}}{p} + c\right) + O\left(\frac{Kd||x||_0 \log n}{p} + c\right) + O\left(\frac{Kd}{p} + c\right) \quad (8)$$

де B – кількість базових детекторів, кожен з яких має розподілену складність $C_{b,dist}$;

p – кількість процесорів;

c – комунікаційні накладні витрати через синхронізацію між вузлами;

K – загальна кількість дерев;

d – максимальна глибина дерева;

$||x||_0$ – кількість невідсутніх записів у навчальних даних.

Таким чином, HBOS_BD, LODA_BD, LSCP_BD та XGBOD_BD формують узгоджений набір розподілених методів виявлення аномалій для великих даних, що покриває як неконтрольовані, так і частково контрольовані сценарії та забезпечує масштабованість завдяки використанню Apache Spark.

Висновки. Розроблено розподілені архітектури для базових алгоритмів виявлення аномалій, що усунуло їхні фундаментальні обмеження щодо масштабованості. Це було досягнуто шляхом заміни їхніх послідовних, ітераційних етапів на високопродуктивні паралельні операції, що надаються фреймворком Apache Spark. Зокрема, для HBOS_BD було впроваджено розподілену побудову гістограм, а для LODA_BD – заміну множинних ітераційних проєкцій на єдину розподілену матричну операцію. Це дало змогу перетворити класичні, але не масштабовані методи, на повноцінні інструменти для Big Data, здатні обробляти великі масиви даних за прийнятний час, зберігаючи при цьому вихідну логіку оцінки аномальності.

Розв'язано ключову проблему обчислювальної складності у складних ансамблевих методах, зокрема в LSCP, шляхом інноваційної модифікації його архітектури. Це було реалізовано через запропоновану заміну ресурсоємного та непрактичного для великих даних методу k-найближчих сусідів на ефективну розподілену кластеризацію для формування "локальних областей".

Сформовано комплексний набір інструментів для виявлення аномалій, що охоплює як неконтрольовані, так і гібридні підходи. Це було здійснено через послідовну розробку та формальний опис чотирьох алгоритмів, кожен з яких представляє окрему категорію методів – від простих статистичних до складних ансамблевих та частково контрольованих.

Список літератури

1. Erhan, L., Ndubuaku, M., Di Mauro, M., Song, W., Chen, M., Fortino, G., Bagdasar, O., Liotta, A. Smart anomaly detection in sensor systems: a multi-perspective review. *Information Fusion*. 2021. Vol. 67. Pp. 64–79.

2. Aguilera-Martos, I., García-Barzana, M., García-Gil, D., Carrasco, J., López, D., Luengo, J., Herrera, F. Multi-step histogram based outlier scores for unsupervised anomaly detection: ArcelorMittal engineering dataset case of study. *Neurocomputing*. 2023. Vol. 544. № 126228.

3. Pevny, T. LODA: lightweight on-line detector of anomalies. *Machine Learning*. 2016. Vol. 102. Pp. 275–304.

4. Zhao, Y., Nasrullah, Z., Li, Z. LSCP: locally selective combination in parallel outlier ensembles. *Proceedings of the 2019 SIAM International Conference on Data Mining (SDM 2019)*. 2019. Pp. 585–593.

5. Zhao, Y., Hryniewicki, M.K. XGBOD: improving supervised outlier detection with unsupervised representation learning. *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018. Pp. 1–8.

Додаток Б

Реалізація методів виявлення аномалій у великих даних

Реалізація HBOS.

```
package org.apache.spark.mllib.anomaly

import org.apache.spark.broadcast.Broadcast
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.{Dataset, Row, SparkSession}
import org.apache.spark.storage.StorageLevel

/**
 * HBOS algorithm
 *
 * @param dataset  the complete dataset
 * @param n_bins   the number of bins
 * @param strategy "static" or "dynamic", choose the version
 * @return a dataset with two columns, unix and scores
 */

class HBOS_BD(dataset: Dataset[Row], n_bins: Int = 100, strategy: String = "static") extends Serializable {
  /**
   * computes the score of each instance for the static version
   *
   * @param iter      the instances
   * @param histogram the histograms
   * @param limits    the upper and lower bound of the histograms
   * @param k         the number of bins
   * @return the index of each instance and his score
   */
  private def scoresMapStatic(iter: Iterator[(Row, Long)], histogram: Broadcast[Array[Array[Double]]], limits:
Broadcast[Array[Array[Double]]], k: Int): Iterator[Array[(Long, Double)]] = {
    // index to compute the bin of each attribute
    var index: Int = 0

    // number of instances in each partition
    var length: Int = 0

    // auxiliar iterator to compute the number of instances in each partition
    val newIterator: (Iterator[(Row, Long)], Iterator[(Row, Long)]) = iter.duplicate

    // compute the number of instances in each partition
    while (newIterator._2.hasNext) {
      newIterator._2.next()
      length = length + 1
    }

    // the scores
    val scores: Array[(Long, Double)] = new Array[(Long, Double)](length)
    var counter: Int = 0
```

```
    // save the value of the iterator
    var aux: (Row, Long) = null

    // look the instances
    while (newIterator._1.hasNext) {
      aux = newIterator._1.next()
      // look the attributes
      scores(counter) = (aux._2, aux._1.toSeq.zipWithIndex.map(att => {
        // index = (attribute - min) / (max - min) / k = ((attribute - min) * k) / (max - min)
        index = (((att._1.asInstanceOf[Double] - limits.value(att._2).head) * k) / (limits.value(att._2).last -
limits.value(att._2).head)).asInstanceOf[Int]
        // check if the value is the max, because the index will be equal to histogram size, going out of bounds
        if (index >= k)
          index = k - 1

        // compute the score log(1/h_i) of each attribute
        Math.log(1 / histogram.value(att._2)(index))
      }).sum) // sum the score of each attribute to get the score for each instance
      counter = counter + 1
    }

    Array(scores).iterator
  }

  /**
   * accumulates the scores for the static version
   *
   * @param score1 set of scores 1
   * @param score2 set of scores 2
   * @return the scores accumulated
   */
  private def scoresReduce(score1: Array[(Long, Double)], score2: Array[(Long, Double)]): Array[(Long,
Double)] = {
    score1 ++ score2
  }

  /**
   * computes the scores for the dynamic version
   *
   * @param iter      the instances
   * @param histogram the histogram
   * @return the index of each instance and his score
   */
  private def scoresMapDynamic(iter: Iterator[(Row, Long)], histogram: Broadcast[Seq[Array[(Double,
Double, Double)]]]): Iterator[Array[(Long, Double)]] = {
    // index to compute the bin of each attribute
    var index: Int = 0

    // number of instances in each partition
    var length: Int = 0

    // auxiliar iterator to compute the number of instances in each partition
    val newIterator: (Iterator[(Row, Long)], Iterator[(Row, Long)]) = iter.duplicate
    // compute the number of instances in each partition
    while (newIterator._2.hasNext) {
```

```
    newIterator._2.next()
    length = length + 1
   }

   // the scores
   val scores: Array[(Long, Double)] = new Array[(Long, Double)](length)

   var counter: Int = 0

   // save the value of the iterator
   var aux: (Row, Long) = null

   // control when the bin is found
   var found: Boolean = false

   //look the instances
   while (newIterator._1.hasNext) {
    aux = newIterator._1.next()
    // look the attributes
    scores(counter) = (aux._2, aux._1.toSeq.zipWithIndex.map(att => {
     index = 0
     //find the bin, the value of the attribute must be between max and min value of the bin
     while (!found) {
      if (att._1.asInstanceOf[Double] >= histogram.value(att._2)(index)._1 && att._1.asInstanceOf[Double]
<= histogram.value(att._2)(index)._2) {
       found = true
      } else {
       index = index + 1
      }
     }
     found = false

     // compute the score log(1/h_i) of each attribute
     Math.log(1 / histogram.value(att._2)(index)._3)
    }).sum) // sum the score of each attribute to get the score for each instance
    counter = counter + 1
   }

   Array(scores).iterator
  }

  /**
   * Computes the HBOS algorithm
   */

  def fit(): RDD[Double] = {
   if (strategy != "static" && strategy != "dynamic") {
    throw new Exception("Method must be static or dynamic")
   } else {
    val sc = SparkSession.builder().getOrCreate()

    // split the column with the features into a dataset which contains one columns for each attribute
    var data = dataset.select(col = "features")
    val disassembler = new VectorDisassembler()
     .setInputCol("features")
```

```
    data = disassembler.transform(data)
    data = data.drop(colName = "features")
    data.persist(StorageLevel.MEMORY_AND_DISK)

    if (strategy == "static") {

      // compute the histograms
      val histograms: Array[Array[Double]] = Array.fill(data.columns.length, n_bins)(0.0)
      val limits: Array[Array[Double]] = Array.fill(data.columns.length, n_bins)(0.0)

      data.columns.zipWithIndex.foreach(name => {
        val histogram = data.select(name._1).rdd.flatMap(v =>
v.toSeq.map(_.asInstanceOf[Double])).histogram(n_bins)
        limits(name._2) = histogram._1
        histograms(name._2) = histogram._2.map(_ + 1e-12)
        val histogram_sum = histograms(name._2).sum
        histograms(name._2) = histograms(name._2).map(_ / histogram_sum)
      })

      val histogramBroadcast: Broadcast[Array[Array[Double]]] = sc.sparkContext.broadcast(histograms)
      val limitsBroadcast: Broadcast[Array[Array[Double]]] = sc.sparkContext.broadcast(limits)

      // compute the scores
      sc.sparkContext.parallelize(data.rdd.zipWithIndex().mapPartitions(split => scoresMapStatic(split,
histogramBroadcast, limitsBroadcast, n_bins)).reduce(scoresReduce)).sortByKey().map(_._2)
    } else {

      // size of the dataset
      val size: Long = data.count()
      // size / k to get the bound of each bin
      val inc: Long = (size / n_bins.asInstanceOf[Double]).ceil.asInstanceOf[Long]
      // control index
      var splitIndex: (Long, Long) = (0, inc)

      // compute the bounds for each bin, giving 0 to values not necessaries, 1 to first value and 2 to last
value of the bin
      val index: RDD[Long] = sc.sparkContext.parallelize((0 until size.asInstanceOf[Int]).map(i => {
        if (i >= (size - 1)) { // if size/k is not exact, the last value must be updated
          2.asInstanceOf[Long]
        } else {
          // first value of the bin
          if (i == splitIndex._1) {
            1.asInstanceOf[Long]
            // last value of the bin
          } else if (i == splitIndex._2 - 1) {
            //update control index
            splitIndex = (splitIndex._2, splitIndex._2 + inc)
            2.asInstanceOf[Long]
            // not necessary value
          } else {
            0.asInstanceOf[Long]
          }
        }
      }))
```

```
      // join the index computed previously to the dataset, then filter getting the values with index 1 and 2
and save those values
      var sortValues: RDD[(Long, Row)] =
data.select(data.columns.head).sort(data.columns.head).rdd.zipWithIndex().map(_.swap)
      data.columns.tail.foreach(name => {
        sortValues =
sortValues.join(data.select(name).sort(name).rdd.zipWithIndex().map(_.swap)).map(value => (value._1,
Row((value._2._1.toSeq :+ value._2._2.get(0)): _*)))
      })
      val organizeValues: Array[Array[Double]] =
sortValues.join(index.zipWithIndex().map(_.swap)).filter(_._2._2 > 0).map(value =>
value._2._1.toSeq.map(_.asInstanceOf[Double]).toArray).collect().transpose

      val histograms: Array[Array[(Double, Double, Double)]] = Array.fill(data.columns.length, n_bins)(0.0,
0.0, 0.0)
      var counter: Int = 0
      var first = true
      var firstValue: Double = 0

      // compute the histogram, pairing first and last value, also save the value of the features
      organizeValues.zipWithIndex.foreach(values => {
        // compute histogram for each attribute
        counter = 0
        values._1.sorted.foreach((value: Double) => {
          if (first) {
            firstValue = value
            first = !first
          } else {
            histograms(values._2)(counter) = (firstValue, value, inc / (value + 1e-9 - firstValue))
            counter = counter + 1
            first = !first
          }
        })
      })

      // compute the scaled histogram
      histograms.indices.foreach(i => {
        val histogram_sum = histograms(i).map(_._3).sum
        histograms(i) = histograms(i).map(values => (values._1, values._2, values._3 / histogram_sum))
      })

      val histogramBroadcast: Broadcast[Seq[Array[(Double, Double, Double)]]] =
sc.sparkContext.broadcast(histograms)

      // computes the scores
      sc.sparkContext.parallelize(data.rdd.zipWithIndex().mapPartitions(split => scoresMapDynamic(split,
histogramBroadcast)).reduce(scoresReduce)).sortByKey().map(_._2)
    }
  }
 }
}
```

Реалізація LODA.

```
package org.apache.spark.mllib.anomaly
```

```
import org.apache.spark.ml.linalg.{Vector => VectorML}
import org.apache.spark.mllib.linalg
import org.apache.spark.mllib.linalg.distributed.RowMatrix
import org.apache.spark.mllib.linalg.{Matrices, Matrix}
import org.apache.spark.mllib.random.StandardNormalGenerator
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.{Dataset, Row}
import org.apache.spark.storage.StorageLevel

/**
 * LODA implementation for Apache Spark
 *
 * @param data         the dataset in Dataset[Row] format
 * @param n_bins       number of bins for the histograms (default = 10)
 * @param n_random_cuts number of projections (default = 100)
 * @param seed         (default = 48151623)
 * @return an RDD[Double] with the scores of each instance of the dataset
 */

class LODA_BD(val data: Dataset[Row], val n_bins: Int = 100, val n_random_cuts: Int = 100, val seed: Long =
48151623) extends Serializable {

  private val dataAsVector: RDD[VectorML] = data.select(col = "features").rdd.map { case Row(v: VectorML)
=> v }
  private val dataAsMatrix = new
RowMatrix(dataAsVector.map(org.apache.spark.mllib.linalg.Vectors.fromML))
  private val n_components: Int = dataAsVector.first.size
  private val n_nonzero_components: Int = Math.sqrt(n_components).toInt
  private val weights: Array[Double] = Array.fill(n_random_cuts)(1.0).map(_ / n_random_cuts)

  /**
   * Creates an Array of size n_components x n_random_cuts for the projection of the data:
   * It generates Arrays of size n_components filled with 0.0 values
   * Sqrt(n_components) random components are selected
   * Those selected values are changed with the values drawn from a normal distribution
   */
  private def createRandomArray(): Array[Double] = {
    val generator: StandardNormalGenerator = new StandardNormalGenerator()
    generator.setSeed(seed)
    var gaussianData = Array[Double]()
    val r = scala.util.Random
    r.setSeed(seed)

    for (i <- 0 until n_random_cuts) {
      val features = Array.fill(n_nonzero_components)(r.nextInt(n_components))
      val arrayZeros = Array.fill(n_components)(0.0)

      for (j <- features.indices) {
        arrayZeros(features(j)) = generator.nextValue()
      }
      gaussianData = gaussianData ++ arrayZeros
    }
    gaussianData
  }
```

```
 // Performs the LODA algorithm
 def fit(): RDD[Double] = {

  // Projection calculation
  val projections: Matrix = Matrices.dense(n_random_cuts, n_components, createRandomArray())
  val projectedMatrix: RowMatrix = dataAsMatrix.multiply(projections.transpose)
  val projectedData: RDD[linalg.Vector] = projectedMatrix.rows
  projectedData.persist(StorageLevel.MEMORY_AND_DISK)

  // Histograms initialization
  val limits: Array[Array[Double]] = Array.fill(n_random_cuts, n_bins + 1)(0.0)
  val histograms: Array[Array[Double]] = Array.fill(n_random_cuts, n_bins)(0)

  // Histogram calculation and normalization
  for (i <- 0 until n_random_cuts) {
   val histogram = projectedData.map(v => v(i)).histogram(n_bins)
   limits(i) = histogram._1
   histograms(i) = histogram._2.map(_ + 1e-12)
   val histogram_sum = histograms(i).sum
   histograms(i) = histograms(i).map(_ / histogram_sum)
  }

  // Scores calculation based on drawn histograms
  val pred_scores: RDD[Double] = projectedData.map { l =>
   val values = l.toArray
   val scores = Array.fill(n_random_cuts)(Double.NegativeInfinity)
   for (i <- 0 until n_random_cuts) {
    val index = (limits(i).drop(1).dropRight(1) :+ values(i)).sorted.indexOf(values(i))
    scores(i) = -weights(i) * Math.log(histograms(i)(index))
   }

   // Scores normalization
   scores.sum / n_random_cuts
  }

  projectedData.unpersist()
  pred_scores
 }
}
```

Реалізація LSCP.

```
package org.apache.spark.mllib.anomaly

import breeze.linalg._
import breeze.stats._
import org.apache.spark.ml.feature.LabeledPoint
import org.apache.spark.ml.linalg.{Vectors, Vector => VectorML}
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types.{LongType, StructField, StructType}
import org.apache.spark.sql.{DataFrame, Dataset, Row}
import org.apache.spark.storage.StorageLevel

import scala.math.sqrt
```

```
/**
 * LSCP implementation for Apache Spark
 *
 * @param data            the dataset in Dataset[Row] format
 * @param n_base_detectors number of base detectors for the ensemble (default = 10)
 * @param strategy        strategy for the pseudo ground truth generation: average ("avg"), maximum
("max") (default = "max")
 * @param clus_method     clustering method for the local neigborhood calculation: "kmeans", "bisec"
(default = "kmeans")
 * @param n_clus          number of clusters for the local neigborhood calculation (default = 11)
 * @param dcs             percentage of base detectors selected for dynamic outlier ensemble selection
 * @return an RDD[Double] with the scores of each instance of the dataset
 */

class LSCP_BD(val data: Dataset[Row], val n_base_detectors: Int = 10, val strategy: String = "avg", val
clus_method: String = "kmeans", val n_clus: Int = 11, val dcs: Double = 0.5) extends Serializable {

 /**
  * Calculates de Pearson Correlation between two vectors
  */

 def pearson(a: Vector[Double], b: Vector[Double]): Double = {
  if (a.length != b.length)
   throw new IllegalArgumentException("Vectors not of the same length.")

  val n = a.length

  val dot = a.dot(b)
  val adot = a.dot(a)
  val bdot = b.dot(b)
  val amean = mean(a)
  val bmean = mean(b)

  (dot - n * amean * bmean) / (sqrt(adot - n * amean * amean) * sqrt(bdot - n * bmean * bmean))
 }

 // Performs the LSCP algorithm
 def fit(): RDD[Double] = {

  // Base Detector Generation
  if (!data.storageLevel.useMemory) data.persist(StorageLevel.MEMORY_AND_DISK)
  val base_detector_scores: Array[RDD[(Long, Double)]] = Array.fill[RDD[(Long,
Double)]](n_base_detectors)(data.sparkSession.sparkContext.emptyRDD)

  for (i <- 0 until n_base_detectors) {
   val n_bins_min = 100
   val n_bins_max = 1000
   val rnd = new scala.util.Random
   val n_bins = n_bins_min + rnd.nextInt((n_bins_max - n_bins_min) + 1)
   val scores = new LODA_BD(data, n_bins).fit()
   if (!scores.getStorageLevel.useMemory) scores.persist(StorageLevel.MEMORY_AND_DISK)
   base_detector_scores(i) = scores.zipWithIndex().map { case (v, k) => (k, v) }
   base_detector_scores(i).persist(StorageLevel.MEMORY_AND_DISK)
  }
```

```
  var pseudo_ground_truth = base_detector_scores.map(_.mapValues(s => Seq(s))).reduce((a, b) =>
a.join(b).mapValues { case (s1, s2) => s1 ++ s2 }).sortByKey(ascending = true).map(_._2.toArray)
  pseudo_ground_truth.persist(StorageLevel.MEMORY_AND_DISK)

  // Pseudo Ground Truth Generation
  pseudo_ground_truth = strategy match {
   case "avg" => pseudo_ground_truth.map { l => l :+ l.sum / l.length }
   case "max" => pseudo_ground_truth.map { l => l :+ l.max }
  }

  // Local Region Definition
  import data.sqlContext.implicits._
  val pseudo_gt_DF = pseudo_ground_truth.map { l => LabeledPoint(0.0, Vectors.dense(l))
}.toDF().drop(colName = "label").withColumnRenamed(existingName = "features", newName = "scores")

  // Add index to each instance
  val dataIndex: DataFrame = data.sqlContext.createDataFrame(
   data.rdd.zipWithIndex.map(ln => Row.fromSeq(Seq(ln._2) ++ ln._1.toSeq)),
   StructType(Array(StructField("index", LongType, nullable = false)) ++ data.schema.fields)
  )
  val gt_Index: DataFrame = pseudo_gt_DF.sqlContext.createDataFrame(
   pseudo_gt_DF.rdd.zipWithIndex.map(ln => Row.fromSeq(Seq(ln._2) ++ ln._1.toSeq)),
   StructType(Array(StructField("index", LongType, nullable = false)) ++ pseudo_gt_DF.schema.fields)
  )

  // Join data & scores
  val data_gt = dataIndex.join(gt_Index, Seq("index"))
  data_gt.persist(StorageLevel.MEMORY_AND_DISK)

  // Cluster Partitioning
  val gtSortedByRegion = new Cluster_Partitioning(data = data_gt, clus_method, n_clus, max_size = 10000,
min_size = 1000).balance_clusters()
  gtSortedByRegion.persist(StorageLevel.MEMORY_AND_DISK)

  // Model Selection and Combination
  val gtSortedRDD = gtSortedByRegion.select(col = "index", cols = "scores").rdd.map { row =>
   val index = row.getAs[Long](fieldName = "index")
   val featuresML = row.getAs[VectorML](fieldName = "scores")
   val features = org.apache.spark.mllib.linalg.Vectors.fromML(featuresML).toArray
   (index, features)
  }
  gtSortedRDD.persist(StorageLevel.MEMORY_AND_DISK)

  // Correlation Calculation
  val scores = gtSortedRDD.mapPartitions { partition =>
   val data = partition.toArray
   if (data.length > 0) {
    val transposed = data.map(_._2).transpose
    val gt = transposed.last
    val correlations = Array.fill[Double](n_base_detectors)(elem = 0.0)
    for (i <- 0 until n_base_detectors) {
     correlations(i) = Math.abs(pearson(breeze.linalg.Vector(transposed(i)), breeze.linalg.Vector(gt)))
    }
    // MAX / MOA / AOM
```

```
      if (dcs == 1.0) {
        val index = correlations.indexOf(correlations.max)
        data.map{l => (l._1, l._2.apply(index))}.toIterator
      } else {
        var num_pgt = (n_base_detectors * dcs).toInt
        if (num_pgt < 1) num_pgt = 1
        val sorted = correlations.sortWith(_ > _)
        val mostCorrelated = Array.fill[Int](num_pgt)(elem = -1)
        for (j <- 0 until num_pgt) {
          mostCorrelated(j) = correlations.indexOf(sorted(j))
        }
        if (strategy.equals("avg")) {
          data.map { l =>
            val values = Array.fill[Double](num_pgt)(elem = Double.NegativeInfinity)
            for (j <- 0 until num_pgt) {
              values(j) = l._2(mostCorrelated(j))
            }
            (l._1, values.max)
          }.toIterator
        } else {
          data.map { l =>
            val values = Array.fill[Double](num_pgt)(elem = Double.NegativeInfinity)
            for (j <- 0 until num_pgt) {
              values(j) = l._2(mostCorrelated(j))
            }
            (l._1, values.sum / values.length)
          }.toIterator
        }
      }
    } else {
      Array[(Long, Double)]().toIterator
    }
   }
   scores.sortByKey(ascending = true).map(_._2)
 }
}
```

Реалізація XGBOD.

```
package org.apache.spark.mllib.anomaly

import ml.dmlc.xgboost4j.scala.spark.XGBoostClassifier
import org.apache.spark.ml.feature.{LabeledPoint, VectorAssembler}
import org.apache.spark.ml.linalg.{Vectors, Vector => VectorML}
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types.{LongType, StructField, StructType}
import org.apache.spark.sql.{DataFrame, Dataset, Row}
import org.apache.spark.storage.StorageLevel

/**
 * XGBOD implementation for Apache Spark
 *
 * @param data              the dataset in Dataset[features, label] format
 * @param n_base_detectors  number of base detectors for the ensemble (default = 10)
```

```
 * @param n_selected_detectors number of selected detectors (default = 5)
 * @param strategy           strategy for the selection: random ("rnd"), accuracy ("acc") (default = "rnd")
 * @param threshold          threshold for the accuracy strategy (default = 0.1)
 * @param n_cores            number of cores for XGBoost (default = 360)
 * @param seed               (default = 48151623)
 * @return an RDD[Double] with the label predicted of each instance in the dataset
 */

class XGBOD_BD(val data: Dataset[Row], val n_base_detectors: Int = 10, val n_selected_detectors: Int = 5,
val strategy: String = "rnd", val threshold: Double = 0.1, val n_cores: Int = 360, val seed: Int = 48151623)
extends Serializable {

  // Performs the XGBOD algorithm
  def fit(): RDD[Double] = {

    // Transformed Outlier Scores (TOS)
    val outlier_scores: Array[RDD[(Long, Double)]] = Array.fill[RDD[(Long,
Double)]](n_base_detectors)(data.sparkSession.sparkContext.emptyRDD)
    if (!data.storageLevel.useMemory) data.persist()

    // Calculate n_base_detectors detectors and join the results
    for (i <- 0 until n_base_detectors) {
      val n_bins_min = 100
      val n_bins_max = 1000
      val rnd = new scala.util.Random
      val n_bins = n_bins_min + rnd.nextInt((n_bins_max - n_bins_min) + 1)
      val scores = new LODA_BD(data, n_bins).fit()
      if (!scores.getStorageLevel.useMemory) scores.persist(StorageLevel.MEMORY_AND_DISK)
      outlier_scores(i) = scores.zipWithIndex().map { case (v, k) => (k, v) }
      outlier_scores(i).persist(StorageLevel.MEMORY_AND_DISK)
    }

    var transformed_outlier_scores = outlier_scores.map(_.mapValues(s => Seq(s))).reduce((a, b) =>
a.join(b).mapValues { case (s1, s2) => s1 ++ s2 }).sortByKey(ascending = true).map(_._2.toArray)
    transformed_outlier_scores.persist(StorageLevel.MEMORY_AND_DISK)

    // TOS selection
    transformed_outlier_scores = strategy match {
      // Random
      case "rnd" =>
        val tos_list = scala.util.Random.shuffle(0 to n_base_detectors - 1).take(n_selected_detectors)
        transformed_outlier_scores.map { l =>
          var final_tos: Array[Double] = Array[Double]()
          tos_list.foreach { tos =>
            final_tos = final_tos :+ l.apply(tos)
          }
          final_tos
        }
      // Accuracy
      case "acc" =>
        val accuracy = transformed_outlier_scores.map { l =>
          l.map { score =>
            if (score >= threshold) 1.0
            else 0.0
          }
```

```
      }
      val labels = data.rdd.map(_.getAs[Double](fieldName = "label"))
      val metrics_results = Array.fill[Double](n_base_detectors)(elem = 0.0)
      for (i <- accuracy.first().indices) {
       val metrics_data = accuracy.map { l => l(i) }.zipWithIndex().map { case (v, k) => (k, v)
}.join(labels.zipWithIndex().map { case (v, k) => (k, v) }).map(l => l._2)
       metrics_results(i) = new MulticlassMetrics(metrics_data).accuracy
      }
      val (addSorted, indices) = metrics_results.zipWithIndex.sortWith(_._1 > _._1).unzip
      val indexes = indices.take(n_selected_detectors)
      transformed_outlier_scores.map { l =>
       var final_tos: Array[Double] = Array[Double]()
       indexes.foreach { tos =>
        final_tos = final_tos :+ l.apply(tos)
       }
       final_tos
      }
    }

    import data.sqlContext.implicits._
    val TOS_DF = transformed_outlier_scores.map { l => LabeledPoint(0.0, Vectors.dense(l))
}.toDF().drop(colName = "label").withColumnRenamed(existingName = "features", newName = "scores")

    // Append TOS to data
    val dataIndex: DataFrame = data.sqlContext.createDataFrame(
     data.rdd.zipWithIndex.map(ln => Row.fromSeq(Seq(ln._2) ++ ln._1.toSeq)),
     StructType(Array(StructField("index", LongType, nullable = false)) ++ data.schema.fields)
    )
    val TOS_Index: DataFrame = TOS_DF.sqlContext.createDataFrame(
     TOS_DF.rdd.zipWithIndex.map(ln => Row.fromSeq(Seq(ln._2) ++ ln._1.toSeq)),
     StructType(Array(StructField("index", LongType, nullable = false)) ++ TOS_DF.schema.fields)
    )

    // Join data & scores
    val joined_data = dataIndex.join(TOS_Index, Seq("index")).sort(sortCol = "index").drop(colName =
"index")

    val assembler = new VectorAssembler()
     .setInputCols(Array("features", "scores"))
     .setOutputCol("features2")
    val expanded_data = assembler.transform(joined_data).drop(colName = "scores").drop(colName =
"features").withColumnRenamed(existingName = "features2", newName = "features")
    expanded_data.persist(StorageLevel.MEMORY_AND_DISK)

    // XGBOD learning
    val xgbParam = Map("eta" -> 0.1f,
     "missing" -> -999,
     "objective" -> "multi:softmax",
     "num_class" -> 2,
     "num_round" -> 50,
     "num_workers" -> n_cores,
     "tree_method" -> "approx")
    val xgbClassifier = new XGBoostClassifier(xgbParam).
     setFeaturesCol("features").
     setLabelCol("label")
```

```
    val xgbClassificationModel = xgbClassifier.fit(expanded_data)
    val results = xgbClassificationModel.transform(expanded_data.select(col = "features"))
    results.persist()
    results.count()

    results.rdd.map(_.getAs[Double](fieldName = "prediction"))
  }
}
```