

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

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
(підпис)

« _____ » _____ 2024 р.

**Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи, технології та
математичні методи кібербезпеки»
спеціальності 125 «Кібербезпека»**

на тему: Використання ML для детектування веб-атак та інтеграція алгоритму з SIEM
системами

Виконав: здобувач вищої освіти IV курсу, групи ФБ-02
(шифр групи)

Сапегін Валентин Ігорович
(прізвище, ім'я, по батькові) (підпис)

Керівник к.ек.н., доцент каф. ІБ Ткач В.М.
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент к.ф.-м.н., доцент каф. ММЗІ Южакова Г. О.
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Здобувач вищої освіти _____
(підпис)

Київ – 2024 року

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

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Дмитро ЛАНДЕ
(підпис)
«__» _____ 2024 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти
Сапегін Валентин Ігорович

(прізвище, ім'я, по батькові)

1. Тема роботи: Використання ML для детектування веб-атак та інтеграція алгоритму з SIEM системами,

керівник роботи к.е.н., доцент кафедри ІБ Ткач Володимир Миколайович

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31» травня 2024 р. № 2251-с

2. Термін подання здобувачем вищої освіти роботи « 14 » червня 2024 р.

3. Вихідні дані до роботи: наукова література та відкриті джерела, попередні дослідження на тему виявлення веб-атак, дані для тренування моделей (CSE-CIC-IDS2018).

4. Зміст роботи: Огляд видів веб-атак та їх характеристик, традиційні методи виявлення веб-атак, переваги та обмеження використання машинного навчання для виявлення веб-атак, збір та попередня обробка даних, вибір алгоритмів машинного навчання для виявлення веб-атак, навчання та валідація моделей,

вибір найкращої моделі, огляд SIEM систем та їх роль у кібербезпеці, процес інтеграції моделей з SIEM системами, використання ELK стеку для обробки та аналізу даних, тестування інтегрованої системи та аналіз результатів.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): Презентація.

6. Дата видачі завдання 07.04.2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Отримання завдання	07.04.2024	Виконано
2	Огляд та опрацювання інформаційних джерел	08.04.2024 – 15.04.2024	Виконано
3	Огляд наявних датасетів та вибір даних для роботи	16.04.2024 – 27.04.2024	Виконано
4	Обробка даних	28.04.2024 – 06.05.2024	Виконано
5	Написання програмного коду для тренування та тестування моделей машинного навчання	07.05.2024 – 18.05.2024	Виконано
6	Порівняння та вибір найкращої моделі	19.05.2024 – 22.05.2024	Виконано
7	Огляд SIEM систем	23.05.2024 – 25.05.2024	Виконано
8	Написання програмного коду для парсера	26.05.2025 – 29.05.2021	Виконано
9	Тестування системи та аналіз отриманих результатів	29.05.2024 – 04.06.2024	Виконано
10	Оформлення дипломної роботи	05.06.2024 – 12.06.2024	Виконано

Здобувач вищої освіти _____
(підпис)

Керівник роботи _____
(підпис)

Валентин САПЕГІН
(Власне ім'я, ПРІЗВИЩЕ)

Володимир ТКАЧ
(Власне ім'я, ПРІЗВИЩЕ)

Реферат

Обсяг дипломної роботи 56 сторінок, 21 ілюстрації, 2 таблиці, 3 додатки та 10 джерел літератури.

Об'єкт дослідження: веб-атаки.

Предмет дослідження: використання методів машинного навчання для побудови ефективної системи виявлення веб-атак.

Мета дослідження: впровадження моделі машинного навчання для виявлення веб-атак з інтеграцією отриманої моделі в SIEM системи для підвищення ефективності захисту інформаційних систем.

Методи дослідження: аналіз літературних джерел для визначення сучасних методів виявлення веб-атак. Попередня обробка даних. Навчання моделей машинного навчання для виявлення веб-атак. Порівняння ефективності моделей за допомогою метрик точності та вибір найкращої. Інтеграція обраної моделі в SIEM систему та тестування її продуктивності

Отримані результати: у результаті дослідження було визначено, що модель XGBoost забезпечує найкращу ефективність виявлення веб-атак на заданих даних. Отримана модель була інтегрована в SIEM систему ELK, що має на меті дозволити автоматизувати процес завантаження нових логів, їх аналіз та візуалізацію результатів в Kibana.

Ключові слова: машинне навчання, веб-атаки, виявлення вторгнень, SIEM, ELK, кібербезпека, захист даних.

ABSTRACT

The volume of the thesis is 56 pages, 21 illustrations, 2 tables, 3 annexes and 10 sources of literature.

Object of research: web attacks.

Subject of research: the use of machine learning methods to build an effective web-attack detection system.

Purpose of the study: implementation of machine training model to detect a web-attack with the integration of the resulting model into SIEM systems to improve the efficiency of information systems.

Research Methods: analysis of literary sources for determining modern detection methods of web attacks. Pre-processing of data. Training of machine training models to detect web-attacks. Comparison of the effectiveness of models with the help of metrics of accuracy and the choice of the best. Integration of the chosen model into the SIEM system and testing its performance.

Results: the study determined that the XGBOOST model provides the best efficiency in detecting web attacks. The resulting model was integrated into the SIEM system ELK, which aims to automate the process of downloading new logs, their analysis and visualization of results in Kibana.

Keywords: machine learning, web attacks, intrusion detection, SIEM, ELK, cybersecurity, data protection.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ	8
ВСТУП.....	9
1 ПОНЯТТЯ ВЕБ-АТАК. ОГЛЯД МЕТОДІВ ВИЯВЛЕННЯ. МЕТОДИ МАШИННОГО НАВЧАННЯ. АЛГОРИТМ ВПРОВАДЖЕННЯ	11
1.1 Веб-атаки	11
1.2 Традиційні методи виявлення веб-атак	12
1.3 Переваги та обмеження використання машинного навчання для виявлення веб-атак	14
1.4 Алгоритм впровадження машинного навчання для виявлення веб-атак	16
Висновки до розділу 1	17
2 ПІДГОТОВКА НАБОРУ ДАНИХ. НАВЧАННЯ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ ТА ЇХ ПОРІВНЯННЯ	18
2.1 Збір та попередня обробка даних.....	18
2.2 Вибір алгоритмів машинного навчання	22
2.3 Метрики оцінки ефективності моделей.....	24
2.4 Порівняння результатів моделей.....	29
Висновки до розділу 2	45
3 SIEM СИСТЕМИ. ІНТЕГРАЦІЯ З МОДЕЛЛЮ.....	47
3.1 Огляд SIEM систем та їх роль	47
3.2 Процес інтеграції моделі з ELK	48
3.3 Огляд результатів	50
Висновки до розділу 3	52
ВИСНОВКИ	53

	7
ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ	55
Додаток А	57
Додаток Б.....	78
Додаток В	81

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IDS (Intrusion Detection System) – система виявлення вторгнень.

TP (True Positive) – справжнє позитивне спрацювання.

FP (False Positive) – хибно позитивне спрацювання.

TN (True Negative) – справжнє негативне спрацювання.

FN (False Negative) – хибно негативне спрацювання.

SIEM (Security Information and Event Management) – система управління інформацією та подіями безпеки.

ELK (Elasticsearch, Logstash, Kibana) - стек технологій для збору, зберігання та візуалізації даних.

DT (Decision Tree) - метод машинного навчання який використовує дерево рішень.

RF (Random Forest) - ансамблевий метод машинного навчання, який використовує множину дерев рішень.

XGBoost (Extreme Gradient Boosting) - алгоритм градієнтного бустингу, що використовується для класифікації та регресії.

LightGBM (Light Gradient Boosting Machine)- градієнтний бустинг на основі дерев рішень.

CatBoost (Categorical Boosting) - алгоритм градієнтного бустингу, розроблений для обробки категоріальних ознак.

KNN (K-Nearest Neighbors) - метод класифікації та регресії, який використовує близькість до найближчих сусідів для прогнозування міток.

ВСТУП

Розвиток інформаційних технологій та збільшення обсягів даних вимагає підвищеної уваги до їх захисту. Традиційні системи виявлення веб-атак часто мають високий рівень FP та FN спрацьовувань, через складність їх правильного налаштування, що знижує ефективність детектування. В умовах зростання кількості веб-атак та їх складності, актуальним є використання методів машинного навчання для підвищення точності виявлення загроз.

Мета і завдання дослідження:

Метою даного дослідження є впровадження моделі машинного навчання для виявлення веб-атак з інтеграцією отриманої моделі в SIEM системи для підвищення ефективності захисту інформаційних систем.

Завданнями дослідження є:

1. Тренування моделей машинного навчання, порівняння їх результатів та вибір оптимальної моделі.
2. Розробка алгоритму інтеграції моделі з SIEM системами та інтеграція обраної моделі з ELK стеком.

Об'єкт дослідження: Об'єктом дослідження є веб-атаки.

Предмет дослідження: використання методів машинного навчання для побудови ефективної системи виявлення веб-атак.

Методи дослідження:

Аналіз літературних джерел для визначення сучасних методів виявлення веб-атак. Попередня обробка даних. Навчання моделей машинного навчання для виявлення веб-атак. Порівняння ефективності моделей за допомогою метрик точності та вибір найкращої. Інтеграція обраної моделі в SIEM систему та тестування її продуктивності.

Практичне застосування:

В процесі створення роботи було сформовано загальний алгоритм впровадження машинного навчання в інфраструктуру організацій, що є в своїй основі гнучким та може бути адаптованим для використання з різними SIEM системами та особливостями даних інфраструктур. Було навчено, протестовано та інтегровано з SIEM системою ELK модель машинного навчання, яка може використовуватись для підвищення ефективності систем кібербезпеки шляхом детектування веб-атак, забезпечуючи комплексний захист.

1 ПОНЯТТЯ ВЕБ-АТАК. ОГЛЯД МЕТОДІВ ВИЯВЛЕННЯ. МЕТОДИ МАШИННОГО НАВЧАННЯ. АЛГОРИТМ ВПРОВАДЖЕННЯ

1.1 Веб-атаки

Веб-атака – це тип кіберзагрози, при якій зловмисники цілеспрямовано використовують вразливості в веб-сервісах, спрямовані на порушення нормальної роботи веб-ресурсів, крадіжку конфіденційних даних або розповсюдження шкідливого коду серед пристроїв чи інфраструктури компанії. Ці атаки можуть бути виконані через різні вразливості у веб-застосунок та серверних конфігураціях та є однією з найпоширеніших загроз для інформаційних систем в інтернеті.

Веб-атаки можуть бути класифікованими за різними критеріями. Основні види, які буду розглядатись в подальшій роботі включають в себе:

- Атаки «грубою силою» (Brute Force): це метод проб і помилок, що використовується для підбору облікових чи зашифрованих даних;
- Атаки типу відмова в обслуговуванні (DoS): ціллю таких атак є перевантаження інфраструктури веб-сервісу, з ціллю зробити його недоступним для справжніх користувачів;
- Розподілені атаки типу відмова в обслуговуванні (DDoS) та Botnet: ціль цих атак однакова з DoS, проте для її досягнення використовується розподілений трафік, що збільшує складність їх детектування та блокування;
- SQL-ін'єкції (SQL injection): одна з найпоширеніших атак на веб-сайти, використовує ввід SQL коду в форму на веб-сторінці.
- Інфільтрація (Infiltration): отримання несанкціонованого доступу до систем, використовує різноманітні вразливості в системі, від погано захищених портів до слабких паролей;

- Heartbleed: дана атака використовує вразливості OpenSSL для доступу до пам'яті серверів, де може зберігатись конфіденційна інформація, наприклад, паролі або ключі шифрування.

Актуальність проблеми захисту від веб-атак зростає пропорційно збільшенню кількості пристроїв, які мають доступ до мережі інтернет, та ростом обсягу даних, що передаються. Через використання веб-застосунків у всіх сферах бізнесу, навіть незначна вразливість може нести за собою великі втрати. Основними наслідками від успішних веб-атак для компаній та окремих користувачів можуть бути фінансові та репутаційні збитки, або ж втрата конференційної інформації.

З огляду на ці проблеми, необхідно визначити ефективний підхід до виявлення веб-атак, який зможе швидко адаптуватися до нових загроз та зменшити кількість FP та FN помилок. Використання методів машинного навчання для виявлення веб-атак є перспективним напрямком для подальшого розвитку систем виявлення, оскільки може значно підвищити загальну ефективність систем захисту.

1.2 Традиційні методи виявлення веб-атак

Виявлення веб-атак включає в себе ідентифікацію та реагування на спроби злоумисників порушити нормальну роботу веб-сервісів або ж отримати несанкціонований доступ. З моменту появи перших інтернет-ресурсів у 1990-х роках, веб-сервіси почали свій активний розвиток та процес впровадження у бізнес-процеси, тоді питання захисту інформаційних систем стало першочерговим. Перші системи виявлення веб-атак базувалися на простих механізмах, таких як базові міжмережеві екрани та системи контролю доступу.

З часом, складність та масштаб інтернет-ресурсів збільшувалась, пропорційно з цим збільшувалась і складність та кількість веб-атак, вони активно використовували нові вразливості та технології, тому виникла необхідність у

більш просунутих методах захисту. Це призвело до розробки більш складних систем виявлення вторгнень.

Основні методи, які використовуються сьогодні:

- **Сигнатурні методи** базуються на порівнянні мережевого трафіку з попередньо визначеними шаблонами. Система спрацьовує при виявленні збігу з однією із зазначених сигнатур. Цей підхід є дуже ефективним для виявлення відомих атак, але має погано виявляє нові або навіть модифіковані атаки, які більше не відповідають існуючим сигнатурам.
- **Евристичні методи** використовують правила та евристики для детектування. Вони можуть включати аналіз поведінки, відстеження аномалій у трафіку тощо для виявлення аномальних патернів. Хоча ці методи можуть бути ефективними для виявлення нових атак, вони часто генерують багато FP спрацьовувань, що може призводити до додаткових перевірок та витрат часу.
- **Міжмережеві екрани (firewall)** є першою лінією захисту, яка допомагає запобігти багатьом загрозам. Вони обмежують трафік за попередньо визначеними правилами. Мережеві екрани можуть бути окремими апаратними або ж програмними і використовуються для захисту мережі від несанкціонованого доступу. Проте міжмережеві екрани зачасту складні в налаштуванні і потребують регулярного оновлення для ефективного захисту від нових загроз.
- **Системи виявлення вторгнень (IDS)** моніторять мережевий трафік для виявлення потенційних загроз. В цій роботі розглядатись будуть саме NIDS (мережеві системи виявленні вторгнень). NIDS моніторять весь мережевий трафік на визначеній частині мережі, ідентифікуючи підозрілі дії та атаки.
- **Системи запобігання вторгненням (IPS)** є розвитком IDS та мають можливість не лише детектувати, але й блокувати підозрілий трафік. За аналогії з IDS також можуть бути мережевими (NIPS) або на основі хоста (HIPS). IPS може стати вузьким місцем у мережі, що сповільнить трафік та зашкодить бізнес-процесам, а велика кількість FP у IPS може призвести взагалі до зупинки певних процесів компанії.

1.2.1 Обмеження та недоліки традиційних методів виявлення веб-атак.

		Справжні значення	
		Позитивні (1)	Негативні (0)
Визначені значення	Позитивні (1)	TP	FP
	Негативні (0)	FN	TN

Рисунок 1.1 – Матриця невідповідності значень IDS

Системи IDS мають чотири основні стани (рис.1.1):

- **TP:** IDS ідентифікує трафік як атаку, яка дійсно є атакою.
- **TN:** IDS ідентифікує трафік як нормальний, яким він дійсно є.
- **FP:** IDS ідентифікує трафік як атаку, проте він є нормальним для системи.
- **FN:** IDS ідентифікує трафік як нормальний, але він є атакою.

Основними проблемами методів виявлення веб-атак, які використовуються сьогодні, є високий рівень помилок першого та другого роду - FP та FN спрацювання. FP призводять до зайвих перевірок зі сторони аналітиків і суттєвих витрат часу, тоді як FN залишає атаки непоміченими та веде до компрометації системи. Окрім цього, традиційні методи часто не встигають за еволюцією веб-атак і можуть стати неефективними, при появі нових або складніших атак.

1.3 Переваги та обмеження використання машинного навчання для виявлення веб-атак

Враховуючі розглянуті недоліки актуальних методів, постає питання підвищення ефективності виявлення веб-атак за допомогою нових технології та підходів. Використання машинного навчання дозволяє створювати моделі, які можуть адаптуватися до нових загроз та вдосконалюватись з часом що може

вирішити поточні проблеми. Розглянемо детальніше переваги та недоліки цього підходу.

1.3.1 Переваги використання машинного навчання

Завдяки навчанню на великих обсягах даних та побудові неочевидних закономірностей та подальшого донавчання, алгоритми машинного навчання здатні виявляти атаки, які ще не внесені до сигнатур або правил.

Завдяки можливості навчання на справжніх логах компанії, алгоритм може адаптуватись під трафік конкретної системи, що підвищує релевантність та ефективність детектування в реальних умовах, суттєво зменшуючи кількість FP та FN спрацювань.

Алгоритми машинного навчання можуть автоматично аналізувати величезні обсяги даних у реальному часі, виявляючи аномалії швидше та ефективніше, ніж класичні системи IDS, які покладаються на завчасно визначені правила і сигнатури.

1.3.2 Обмеження та виклики використання машинного навчання

Алгоритми машинного навчання потребують великого обсягу якісних даних для навчання. Також при потраплянні в навчальні дані достатньої кількості трафіку з не визначеною веб-атакою, алгоритм може спотворитись та в подальшому тільки збільшити кількість FN помилок. Саме через це правильний підбір навчального датасету є таким важливим.

Модель потрібно правильно навчати, запобігаючи перенавчанню, оскільки це суттєво знизить точність при роботі з новими даними.

Значні зміни в інфраструктурі компанії можуть вплинути на трафік, який вона генерує, та змінити його так, що навчена на попередньому трафіку модель потребуватиме додаткового перенавчання.

1.4 Алгоритм впровадження машинного навчання для виявлення веб-атак

Після проведення аналізу літератури та відкритих джерел було визначено загальний алгоритм впровадження машинного навчання для визначення веб-атак. Цей алгоритм охоплює всі ключові етапи, необхідні для створення ефективної системи виявлення веб-атак:

- **Збір даних:** Перший крок полягає у зборі даних мережевого трафіку компанії. Дані можуть бути зібрані з різних джерел, включаючи журнали веб-додатків, систем моніторингу трафіку та інших джерел логування.
- **Попередня обробка даних:** Після збору даних необхідно обробити їх для подальшої можливості використання для навчання. Цей етап включає в себе очищення даних, нормалізацію, трансформацію, їх класифікацію, видалення непотрібних даних та перетворення у формат, придатний для аналізу.
- **Вибір та налаштування алгоритмів:** Після підготовки даних необхідно визначитись з алгоритмами машинного навчання, які будуть розглядатись та порівнюватись для задачі виявлення веб-атак.
- **Навчання моделі:** На даному етапі моделі машинного навчання навчаються на підготовлених даних. Це потребує часу, великих обсягів обчислюваних даних та налаштування моделей.
- **Тестування та валідація:** Після закінчення навчання необхідно провести тестування моделей та валідацію на окремому наборі тестових даних. Оцінивши ефективність моделі та її здатність до виявлення веб-атак визначається найкращий алгоритм для інфраструктури компанії, який і буде використовуватись в подальшому.
- **Інтеграція з SIEM системою:** Для використання моделі в реальних умовах, інтеграція з системою SIEM є ключовою, для забезпечення моніторингу та аналізу мережевого трафіку у реальному часі. Це включає налаштування стеку вибраної SIEM, написання парсера для автоматичного завантаження нових логів та передачу результатів роботи алгоритму аналізу до системи візуалізації.

- **Тестування інтегрованої системи:** Після інтеграції необхідно провести тестування всієї системи, як і самої моделі машинного навчання, так і правильної та безперебійної роботи в парі з SIEM. Це включає перевірку коректності завантаження логів, точності аналізу та візуалізації результатів.
- **Моніторинг та оптимізація:** Після завершення тестування необхідно постійно моніторити її роботу, проводити додаткове навчання моделі, щоб залишати ефективність на найвищому рівні та доналаштування моделі в подальшому. Також цей етап може включати визначення мінімальної кількості обчислювальних даних для коректної роботи тощо.

Висновки до розділу 1

Веб-атаки становлять серйозну загрозу для інформаційних систем сьогодні через зростання кількості підключених до інтернету пристроїв та обсягів даних, що передаються мережою. Було розглянуто основні види атак: Brute Force, DoS, DDoS, SQL-ін'єкції, Інфільтрація та Heartbleed.

Традиційні методи виявлення веб-атак, мають значні обмеження, вони ефективні для виявлення вже відомих загроз, проте часно не справляються з новими та модифікованими атаками, також можуть генерувати багато FP та FN спрацьовувань.

Використання алгоритмів машинного навчання це перспективний підхід для виявлення веб-атак, який може допомогти значно зменшити кількість FP та FN спрацьовувань, а також може швидко адаптуватися до нових загроз.

Було розглянуто загальний алгоритм впровадження машинного навчання, який дозволить створити ефективну систему виявлення веб-атак.

2 ПІДГОТОВКА НАБОРУ ДАНИХ. НАВЧАННЯ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ ТА ЇХ ПОРІВНЯННЯ

2.1 Збір та попередня обробка даних.

Збір та попередня обробка даних є першими та ключовими етапами у розробці моделей машинного навчання, оскільки від цих пунктів найбільше залежить якість кінцевих моделей. У цьому підрозділі буде розглянуто процес збору та обробки даних на прикладі датасету CSE-CIC-IDS2018.

2.1.1 Збір даних

Для задачі навчання моделей машинного навчання був обраний датасет CSE-CIC-IDS2018. Він був створений Канадським інститутом кібербезпеки (Canadian Institute for Cybersecurity) у співпраці з Центром передового досвіду в області кібербезпеки (CSE - Communications Security Establishment) на основі AWS. Датасет містить детальні дані, що спеціально згенеровані максимально наближеними до справжнього трафіку, які включають в себе як нормальний трафік, так і різні типи атак.

Для максимального наближення даних до справжніх, при створенні цього датасету, була створена архітектура, яка включала в себе робочі станції на базі Windows та Linux.

Атаки, які включає в себе датасет: Brute-force, Heartbleed, Botnet, DoS, DDoS, Web attacks та Інфільтрацію. Дані збережені в декількох csv файлах, кожен з яких зберігає в собі певні атаки, здійснені в різні дати.

Цей датасет був обраний для даного дослідження через його високу якість та реалістичність, що дозволить ефективно натренувати та протестувати моделі машинного навчання для задачі виявлення різних типів веб-атак.

2.1.2 Обробка та підготовка даних

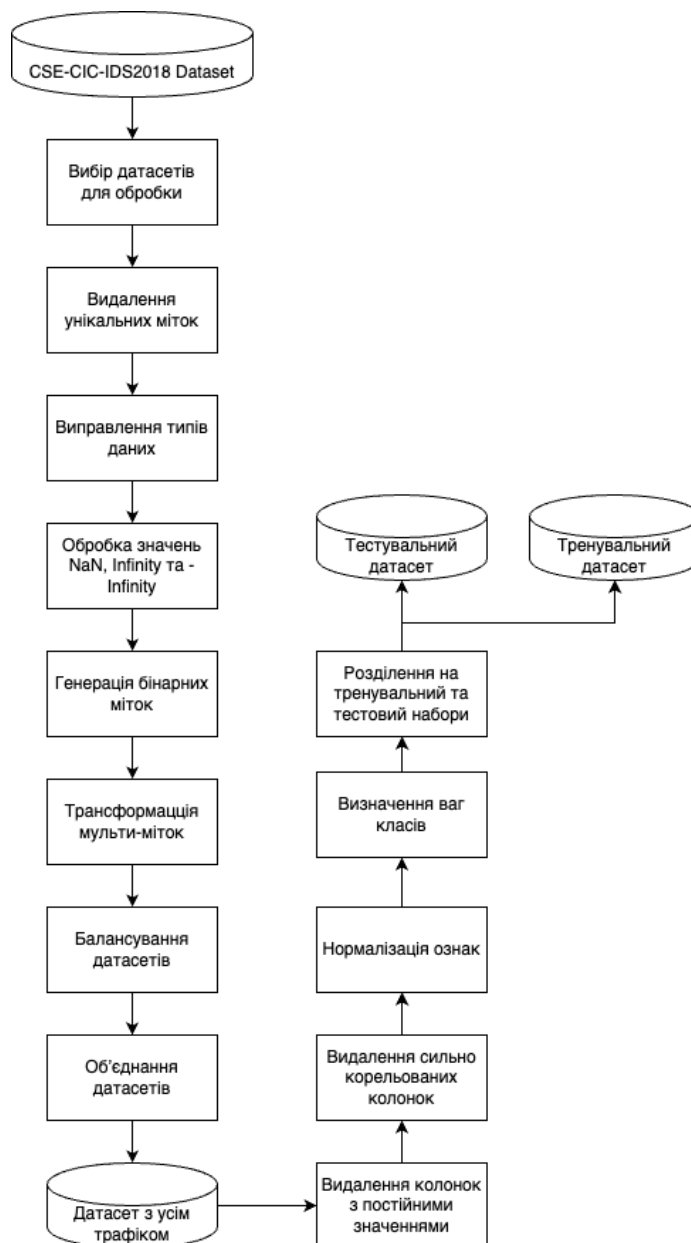


Рисунок 2.1 – Схематичне представлення процесу обробки даних

Попередня обробка даних дозволяє трансформувати дані, зменшити кількість ознак та загалом оптимізувати набір даних для покращення результатів навчання.

На рисунку 2.1 зображена схема проведеної обробки даних. Було зроблено наступні кроки:

1. **Видалення унікальних міток:** На першому етапі обробки було здійснено порівняння колонок у всіх частинах датасету та уніфіковано їх шляхом видалення унікальних. Були видалені такі колонки з 4 датасету “Dataset/02-20-2018.csv”: ‘Src Port’, ‘Flow ID’, ‘Src IP’, ‘Dst IP’.
2. **Виправлення типів даних:** Всі колонки були переведені у відповідні типи даних (integer або float) для коректної їх інтерпретації та зменшення можливості появи помилок.
3. **Обробка Nan та Infinity значень:** Всі значення типу “Infinity”, “-Infinity” та “NaN” було замінені на допустимі або ж видалені для забезпечення цілісності навчальних даних та уникнення подальших помилок.
4. **Генерація бінарних міток:** На цьому етапі трафік класифікується як нормальний (Benign) або шкідливий (Malicious). Це дозволяє спростити початковий процес навчання моделей машинного навчання.
5. **Трансформація мульти-міток:** Було здійснено трансформацію мульти-міток у 7 класів: “Benign, Brute-force, Web attack, DoS attack, DDoS attack, Botnet, Infiltration”. Це дозволило об’єднати схожі типи атак, що дозволило отримати кращу точність та спростити задачу класифікації без втрати наочності результату. Об’єднання міток було виконано згідно до таблиці 2.1

Таблиця 2.1 – Класифікація атак за категоріями

Brute-force	SSH-Bruteforce
	FTP-Bruteforce
Web attack	Bruteforce-XSS
	Bruteforce-WEB
	SQL Injection
DoS attack	DoS attack-slowHTTPTest
	DoS attack-Slowloris
	DoS attack-GoldenEye
DDoS attack	DDos attack-HOIC
	DDos attack-LOIC-UDP
	DDoS attack-LOIC-HTTP
Botnet	Bot
Infiltration	Infiltration
Benign	Benign

6. **Балансування датасету:** Був використаний метод рандомізованого підбору даних (Random UnderSampling) для балансування кількості даних у кожному класі. Це вирішило проблему суттєвого переважування одних класів над іншим, що також позитивно повпливало на процес навчання моделі.
7. **Об'єднання датасету:** Після завершення попередніх обробок, всі датасети були об'єднанні в один для зручності подальшої роботи.
8. **Визначення та видалення колонок з постійними значеннями:** Було виявлено колонки з нульовою дисперсією, їх видалення дозволило позбутись даних, які мають постійні значення та не несуть корисної інформації.
9. **Видалення сильно корельованих колонок:** Для запобігання мультиколінеарності та ризику перенавчання, а також додатково зменшення розмірності датасету було здійснено аналіз кореляції між колонками за тепловою картою (Додаток Б). Було виявлено та видалено сильно корельовані колонки (поріг кореляції встановлений в 0.90). Дані, що залишились після видалення корельованих колонок, зменшили кількість ознак з 72 до 38, що в свою чергу покращило ефективність та точність моделей машинного навчання.
10. **Нормалізація ознак:** Для забезпечення рівнозначності впливу всіх особливостей на модель, всі дані були нормалізовані за допомогою методу MinMaxScaler, та преведено в діапазон від 0 до 1.
11. **Визначення ваг класів:** Ваги класів дозволяють краще навчатися на менш представлених класах та запобігнути їх ігноруванню під час навчання. Ваги визначенні за допомогою бібліотеки `sklearn.utils.class_weight` за формулою:

$$weight_i = \frac{N}{n_i \times k} \quad 2.1$$

де N – кількість зразків у датасеті;

n_i – кількість зразків у i класі;

k – кількість класів у датасеті.

12. Розділення на тренувальний та тестовий набори: Датасет було розділено на тренувальний та тестовий набори даних для подальшого навчання та тестування моделей.

2.2 Вибір алгоритмів машинного навчання

Наступним ключовим етапом після збору та обробки даних є вибір підходящого алгоритму машинного навчання. Для виконання цієї роботи було обрано шість алгоритмів: Decision Tree (DT), Random Forest (RF), XGBoost, LitghtGBM, CatBoost, K-Nearest Neighbors (KNN).

2.2.1 Decision Tree

DT – це базовий алгоритм, структура якого схожа на блок-схему, що використовується для вирішення задач класифікації та регресії. Він має структуру дерева та складається з внутрішніх вузлів, що відповідають за перевірку конкретної умови, де кожна гілка відповідає результату перевірки та зовнішніх (кінцевих) вузлів, які відповідають певному класу або постійному значенню. Цей алгоритм є частиною більш складних, таких як RF та Gradient Boosting.

Перевагами DT є його легкість інтерпретації, що в свою чергу дозволяє визначити, які саме характеристики даних впливають на результат класифікації найбільше, а також має високу швидкість роботи.

До недоліків можна віднести схильність до перенавчання (в більшості на невеликих наборах даних) та чутливість до змін у даних, що може вплинути на зміну результатів.

DT забезпечує базову точку для порівняння з більш складними алгоритмами, які використовуються в даній роботі. Він легко інтерпретується, через що дозволяє наглядно порівнювати його та визначати вплив різних характеристик даних.

2.2.2 Random Forest

RF – ансамблевий метод, що використовує велику кількість дерев рішень (DT). Для кожного дерева використовуються різні підмножини даних та ознак. Кінцевий результат визначається за «голосуванням» дерев.

RF зазвичай має доволі високу точність, він стійкий до перенавчання та здатен ефективно працювати на наборі з великою кількістю даних, а також мають велику кількість ознак.

Проте, складність моделі RF потребує багато ресурсів та її складніше інтерпретувати, порівняно із одиночним DT.

2.2.3 XGBoost

XGBoost – це алгоритм градієнтного бустингу дерева рішень, що складається з ансамблю більш слабких моделей DT, який використовується для побудови сильної моделі.

До переваг XGBoost відносяться його загальна висока ефективність та швидкість, він гарно обробляє великі набори даних і наявність вбудованих механізмів захисту від перенавчання.

Недоліками є складність в інтерпретуванні та в налаштуванні гіперпараметрів, що може зайняти багато часу.

2.2.4 LightGBM

LightGBM – алгоритм градієнтного бустингу, який використовує метод листового росту дерева, що полягає у виборі одного листка на кожному кроці, що має найбільше зменшення втрат, і потім він розбивається на два нових листки.

Перевагами LightGBM є висока швидкість роботи, низьке споживання ресурсів та ефективна робота з великими наборами даних.

Проте, LightGBM має високий ризик перенавчання та високу складність налаштування для ефективної роботи.

2.2.5 CatBoost

CatBoost також є алгоритмом градієнтного бустингу, що розроблений спеціально для обробки категоріальних задач без потреби додаткової їх трансформації.

Окрім ефективної роботи з категоріальними даними, цей алгоритм має зазвичай високу точність, швидкість, а також невеликий ризик перенавчання. На додаток, він менш чутливий до параметрів у порівнянні з іншими алгоритмами градієнтного бустингу.

CatBoost також має свої недоліки, а саме виский рівень споживання ресурсів та складність у налаштуванні.

2.2.6 K-Nearest Neighbors

KNN – доволі простий, проте ефективний алгоритм класифікації, робота якого полягає в пошуку найближчих сусідів кожної точки даних у всьому просторі ознак. Результат класифікації базується на значенні класів найближчих сусідів.

KNN простий та легкий для розуміння та інтерпретації алгоритм, який може працювати добре на великому наборі даних.

Недоліками ж є висока складність обчислення на великих наборах даних та чутливість до дисбалансу класів.

2.3 Метрики оцінки ефективності моделей

- **Точність (Accuracy):** точність є однією з базових та найпоширеніших метрик оцінки якості роботи моделей. Вона показує, наскільки добре модель виконує задачу класифікації, а саме частку правильно класифікованих зразків відносно їх загальної кількості.

Високе значення точності означає, що модель правильно класифікує більшість зразків. Однак, потрібно зважати, що точність може показувати необ'єктивні дані при роботі з незбалансованими даними.

Формула для обчислення точності:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \quad 2.2$$

- **Влучність (Precision):** влучність вимірює, наскільки добре модель скільки з позитивних передбачень є насправді позитивними. Тобто, показує, наскільки добре модель уникає FP помилок.

Чим вище значення, тим менше модель помиляється при передбаченні атак, проте влучність не враховує FN результати, тому не дає побачити повної картини.

Формула для обчислення влучності:

$$\text{Precision} = \frac{TP}{TP + FP} \quad 2.3$$

- **Повнота (Recall):** вимірює здатність моделі виявляти TP результати та показує частку істинно позитивних серед всіх реальних позитивних зразків. Високе значення recall означає, що модель ефективно виявляє більшість реальних загроз.

Формула обчислення повноти:

$$\text{Recall} = \frac{TP}{TP + FN} \quad 2.4$$

- **Специфічність (Specificity):** визначає частку правильно визначених FN випадків серед усіх негативних випадків. Тобто, специфічність показує кількість помилкових спрацювань на нешкідливому трафіку.

Формула для обчислення специфічності:

$$\text{Specificity} = \frac{TN}{TN + FP} \quad 2.5$$

- **Хибно позитивний рейтинг (FNR):** визначає частку позитивних випадків, які були класифіковані, як негативні. Високий рівень означає, що модель пропускає велику кількість атак.

Формула для обчислення FNR:

$$\text{FNR} = \frac{FN}{FN + TP} \quad 2.6$$

- **F1-міра (F1-score):** це гармонічне середнє влучності та повноти. Ця метрика дозволяє, за допомогою балансування між двома метриками, побачити повну картину роботи алгоритму класифікації. F1-міра може набувати значення від 0 до 1. Високе значення вказує на те, що модель добре збалансована та буде мінімально генерувати помилки типу FP та FN.

Формула обчислення F1-міри:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad 2.7$$

- **Коефіцієнт кореляції Метьюса (MCC):** особливістю метрики MCC є те, що вона враховує всі 4 значення матриці невідповідності (TP, FP, TN, FN). Це забезпечує збалансоване оцінювання моделі, навіть за умови нерівномірного розподілення класів. MCC також корисний у випадках, коли важливі не тільки значення влучності і повноти, але й узгодженість між позитивними та негативними передбаченнями зразків.

Формула для обчислення MCC:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad 2.8$$

- **Збалансована точність:** дозволяє якісно оцінити класифікацію моделі машинного навчання на незбалансованих класах. Це середнє арифметичне чутливості (вимірює частку істинно позитивних значень) та специфічності (вимірює частку істинно негативних значень).

Формула обчислення збалансованої точності:

$$\text{Balanced Accuracy} = \frac{1}{2} \left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right) \quad 2.9$$

- **Cohen's Карра:** це метрика, що оцінює рівень згоди між двома наборами класифікацій, враховуючи при цьому можливість випадкового збігу. Це дає можливість оцінити узгодження між передбаченням моделі та визначеними справжніми мітками, додатково враховуючи можливість випадкового збігу, тому є доволі надійною метрикою для мультикласифікації.

Cohen's Карра може набувати значень від -1 до 1, де значення менше 0 вказує на результат, гірший за випадковий збіг, 0 – що не перевищує випадковий збіг, 1 – повне співпадіння результатів.

Формули для обчислення Cohen's Карра:

$$\kappa = 1 - \frac{1 - P_0}{1 - P_e} \quad 2.10$$

де P_0 – очікувана згода;

P_e – спостережна згода.

P_0 та P_e обчислюються за формулами:

$$P_o = \frac{\sum_{i=1}^k c_{ii}}{N}, P_e = \sum_{i=1}^k \left(\frac{\sum_{j=1}^k c_{ij} \sum_{j=1}^k c_{ji}}{N^2} \right) \quad 2.11, 2.12$$

де c_{ii} – кількість випадків, коли модель та реальні мітки співпадають для класу i ;
 c_{ij} - кількість випадків, коли реальна мітка – i , а модель передбачає j ;
 N – кількість випадків загалом.

- **Логарифмічна втрата (крос-ентропія):** показує ймовірність віднесення значення до певного класу, визначену моделлю. Логарифмічна втрата враховує не лише точність класифікації, але й впевненість, з якою модель робить передбачення.

Формула обчислення логарифмічної втрати для задачі мультикласифікації:

$$P_e = \sum_{i=1}^k \left(\frac{\sum_{j=1}^k c_{ij} \sum_{j=1}^k c_{ji}}{N^2} \right) \quad 2.13$$

- **ROC-AUC:** Ця метрика дозволяє оцінити, наскільки добре модель розрізняє позитивні та негативні класи, враховуючи всі можливі порогові значення.

ROC-крива – це візуальне представлення продуктивності моделі, яке демонструє співвідношення між показниками повноти та хибно негативним рейтингом (FPR) для різних порогів класифікації, що дає побачити, наскільки добре модель розрізняє позитивні та негативні класи на всьому діапазоні порогів.

AUC – числовий показник, який демонструє загальну якість моделі класифікації. Значення AUC надає кількісну оцінку того, наскільки добре модель відокремлює класи. AUC обчислюється як площа під кривою ROC. При значення $AUC = 0.5$, модель просто вгадує класи і чим більшого набуває значення, тим краще має здатність класифікувати позитивно.

- **Precision-Recall:** є графічним відображенням залежності між влучністю і повнотою при різних порогових значеннях для двокласових класифікаторів. Для

мультикласової класифікації, Precision-Recall крива може бути обчислена окремо для кожного класу. Кожен окремий клас розглядається як позитивний, а всі інші – як негативні. Це дозволяє побудувати криву для кожного класу та оцінити продуктивність моделі для кожного з них. Крива Precision Recall є корисною для оцінки ефективності моделей, особливо при незбалансованих даних.

- **Матриця невідповідностей:** це графічне (або табличне) представлення класифікації, що дозволяє оцінити кількість правильно класифікованих значень відносно неправильно класифікованих і також наочно побачити, з яким іншим класом модель плутає визначений. Загалом, матриця невідповідностей надає повну картину про правильні та неправильні передбачення моделі.

2.4 Порівняння результатів моделей

2.4.1 Проведення експериментів

Для порівняння моделей було використано попередньо оброблений і розділений на тренувальний та тестовий набір даних з датасету CSE-CIC-IDS2018. Основна мета експериментів полягала в тому, щоб оцінити ефективність обраних алгоритмів машинного навчання для вирішення задачі мультикласифікації веб-атак. Кожна модель була навчена на тренувальних даних з використанням відповідних гіперпараметрів декілька разів для зменшення впливу даних на об'єктивність оцінки точності отриманих результатів. Метрики були обчислені з використанням відповідних бібліотек Python, таких як sklearn, seaborn, matplotlib та відповідних формул.

2.4.2 Аналіз отриманих результатів

На основі проведених експериментів було отримано наступні результати роботи моделей:

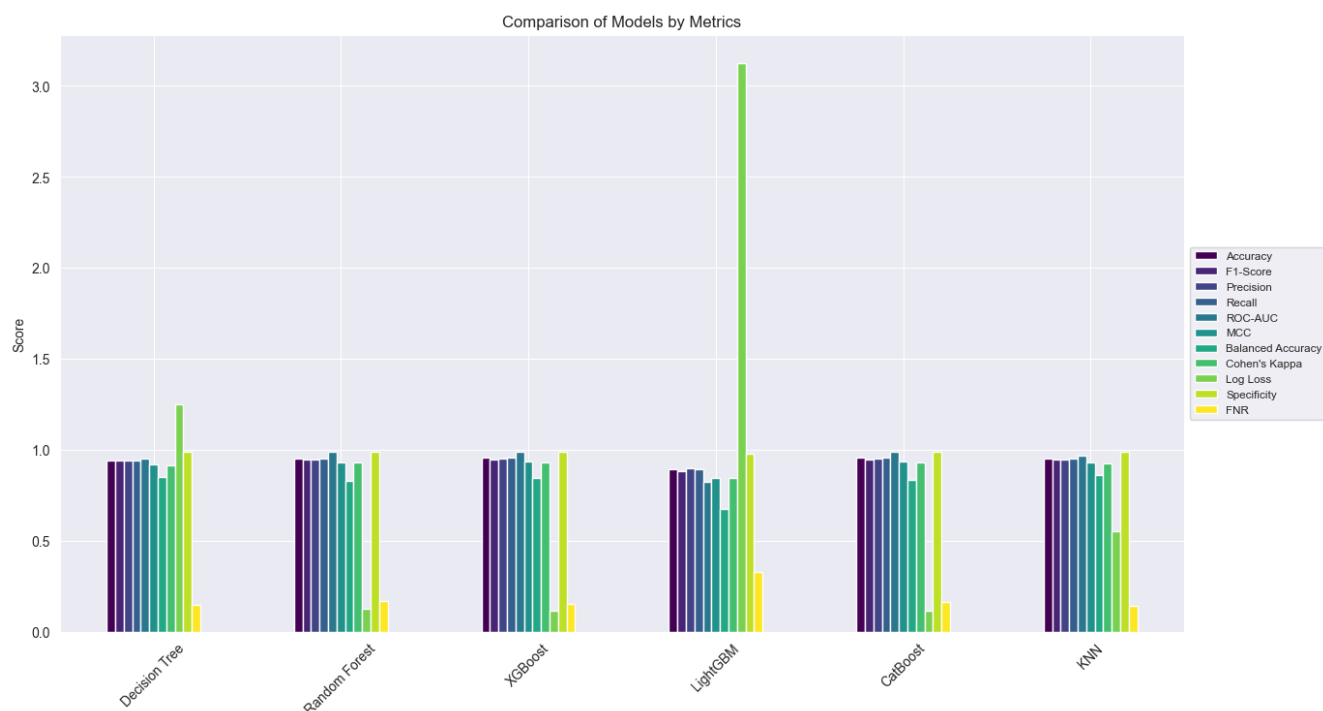


Рисунок 2.3 – Порівняння результатів моделей

Таблиця 2.2 – Результати роботи моделей

	DT	RF	XGBoost	LightGBM	CatBoost	KNN
Точність	0.93	0.94	0.95	0.89	0.95	0.94
Влучність	0.92	0.94	0.95	0.89	0.94	0.92
Повнота	0.92	0.95	0.95	0.89	0.95	0.94
Специфічність	0.98	0.98	0.98	0.97	0.98	0.98
Хибно позитивний рейтинг	0.14	0.17	0.15	0.32	0.16	0.14
F1-міра	0.92	0.94	0.94	0.88	0.94	0.93
Коефіцієнт кореляції Метьюса	0.91	0.92	0.93	0.84	0.93	0.92
Збалансована точність	0.85	0.82	0.84	0.67	0.83	0.86
Cohen's Kappa	0.91	0.92	0.93	0.84	0.93	0.92

Кінець таблиці 2.2

Логарифмічна втрата	1.25	0.12	0.11	3.12	0.11	0.55
ROC-AUC	0.95	0.98	0.99	0.82	0.99	0.96

- Аналіз таблиці з результатами:**

Найвищу точність показала модель XGBoost (0.95) та CatBoost (0.95), що показує здатність цих моделей найефективніше класифікувати більшість даних. DT (0.93), RF (0.94) та KNN (0.94) також показали високий результат. Найгірший результат показав LightGBM (0.89).

Влучність показує, скільки з передбачених позитивних випадків є дійсно позитивними. XGBoost має найвищу точність (0.95), що свідчить про здатність мінімізувати FP передбачення цією моделлю. Схожі результати має CatBoost і RF (0.95 і 0.94 відповідно). LightGBM знову має найнижчий показник (0.89).

Повнота показує, скільки з реальних позитивних випадків були правильно передбачені. Модель XGBoost має найвищий показник повноти (0.954429), разом і з CatBoost (0.95) та RF (0.95) що означає, що вони успішно ідентифікують більшість позитивних випадків. LightGBM має найнижчий показник (0.89), що свідчить про те, що вона пропускає більше позитивних випадків.

Специфічність показує, скільки з реальних негативних випадків були правильно передбачені. Усі моделі мають високі показники (0.98), що свідчить про їх здатність мінімізувати FN передбачення. LightGBM має трохи нижчий показник (0.97).

Хибно позитивний рейтинг показує, скільки з реальних позитивних випадків були пропущені моделлю. XGBoost має один з найнижчих показників FNR (0.15), що свідчить про її здатність мінімізувати пропуски позитивних випадків. Найкраще показали себе DT та KNN (0.14). CatBoost і Random Forest також мають доволі низькі показники (0.16 і 0.17 відповідно). LightGBM має найвищий

показник (0.32), що показує меншу ефективність передбачення позитивних випадків.

Моделі CatBoost, XGBoost та RF показали найвищий результат F1-міри (0.94), що показує гарний баланс між повнотою та специфічністю. DT (0.92) та KNN (0.93) також показали гарний результат. LightGBM має найнижчий F1-Score (0.88).

Коефіцієнт кореляції Метьюса вимірює якість класифікаторів. XGBoost та CatBoost мають найвищі MCC (0.93), що показує високу якість класифікації. RF, KNN, DT також мають високі значення (0.92, 0.92 і 0.91 відповідно). LightGBM знову показує гірший результат (0.84).

Збалансована точність враховує збалансованість між класами. KNN, DT та XGBoost мають найвищі показники (0.86, 0.85 та 0.84 відповідно). LightGBM має найнижчий показник (0.67), що свідчить про проблеми з класифікацією окремих класів.

Cohen's Kappa вимірює узгодженість між передбаченнями моделей і реальними значеннями. XGBoost і CatBoost мають найвищі показники (0.93). RF та KNN також мають високий показник (0.92). LightGBM має найнижчий (0.84).

Логарифмічна втрата показує ефективність моделі у передбаченні ймовірностей і її впевненість у своїх передбаченнях. XGBoost і CatBoost мають найнижчу втрату (0.11), що свідчить про їх здатність до точного передбачення ймовірностей. DT та LightGBM мають найбільш високий показник логарифмічної втрати (1.25 та 3.12 відповідно), що їх меншу впевненість у своїх передбаченнях у порівнянні з іншими моделями, що розглядаються.

ROC-AUC показує загальну здатність моделі розрізняти класи. XGBoost та CatBoost знову показали себе найкраще (0.99), що показує їх можливість дуже добре розрізняти класи між собою між класами. Інші моделі також показали гарний результат, окрім LightGBM, що має найнижчий показник (0.82), який свідчить про її меншу ефективність.

- Аналіз матриць невідповідності:

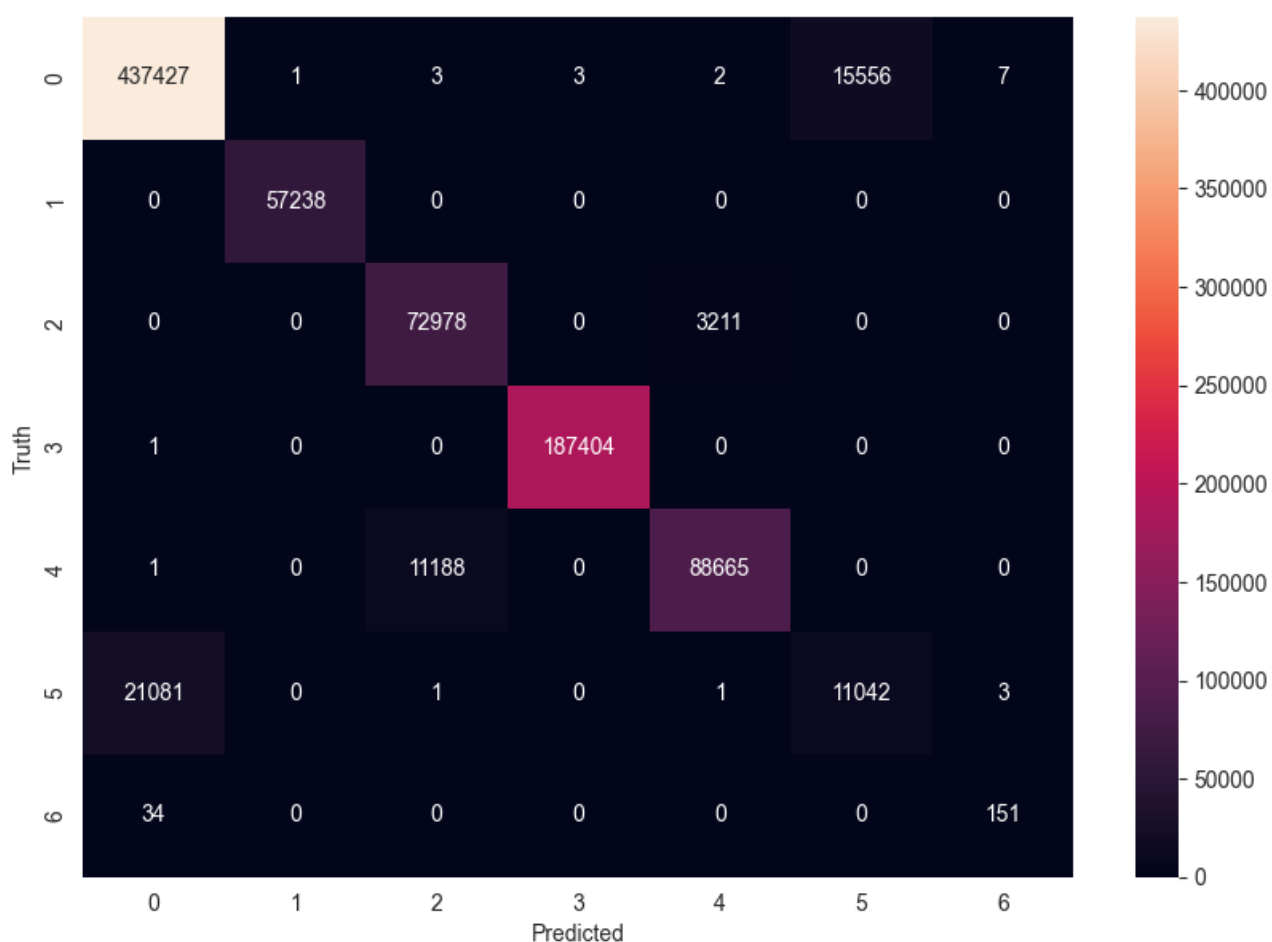


Рисунок 2.4 – Матриця невідповідностей для DT

На матриці невідповідностей для DT видно, що модель показує гарний результат класифікації, особливо для класів Benign, Brute-force, DoS attack та DDoS attack, з незначною кількістю помилкових класифікацій. Більшість труднощів спостерігаються з класифікацією атаки типу Infiltration, що змішується з Benign. Загалом, модель демонструє хороший рівень ефективності.

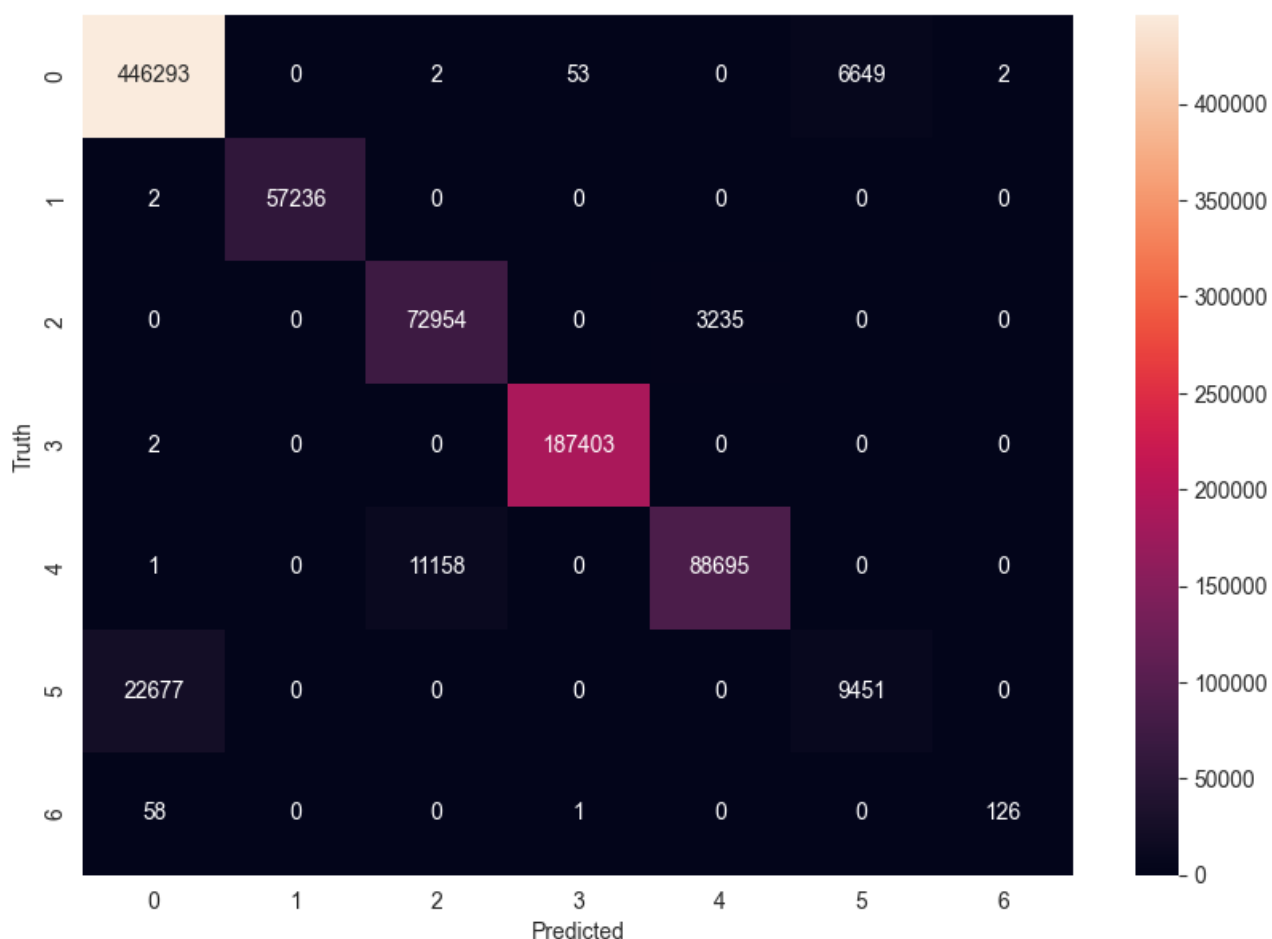


Рисунок 2.5 – Матриця невідповідностей для RF

У випадку з RF можемо бачити також гарний результат класифікації. Модель гарно справляється з більшість класів, проте також має проблеми з Infiltration. Незважаючи на це, RF показав покращену продуктивність у порівнянні з DT, маючи меншу кількість помилкових класифікацій у загальному випадку.

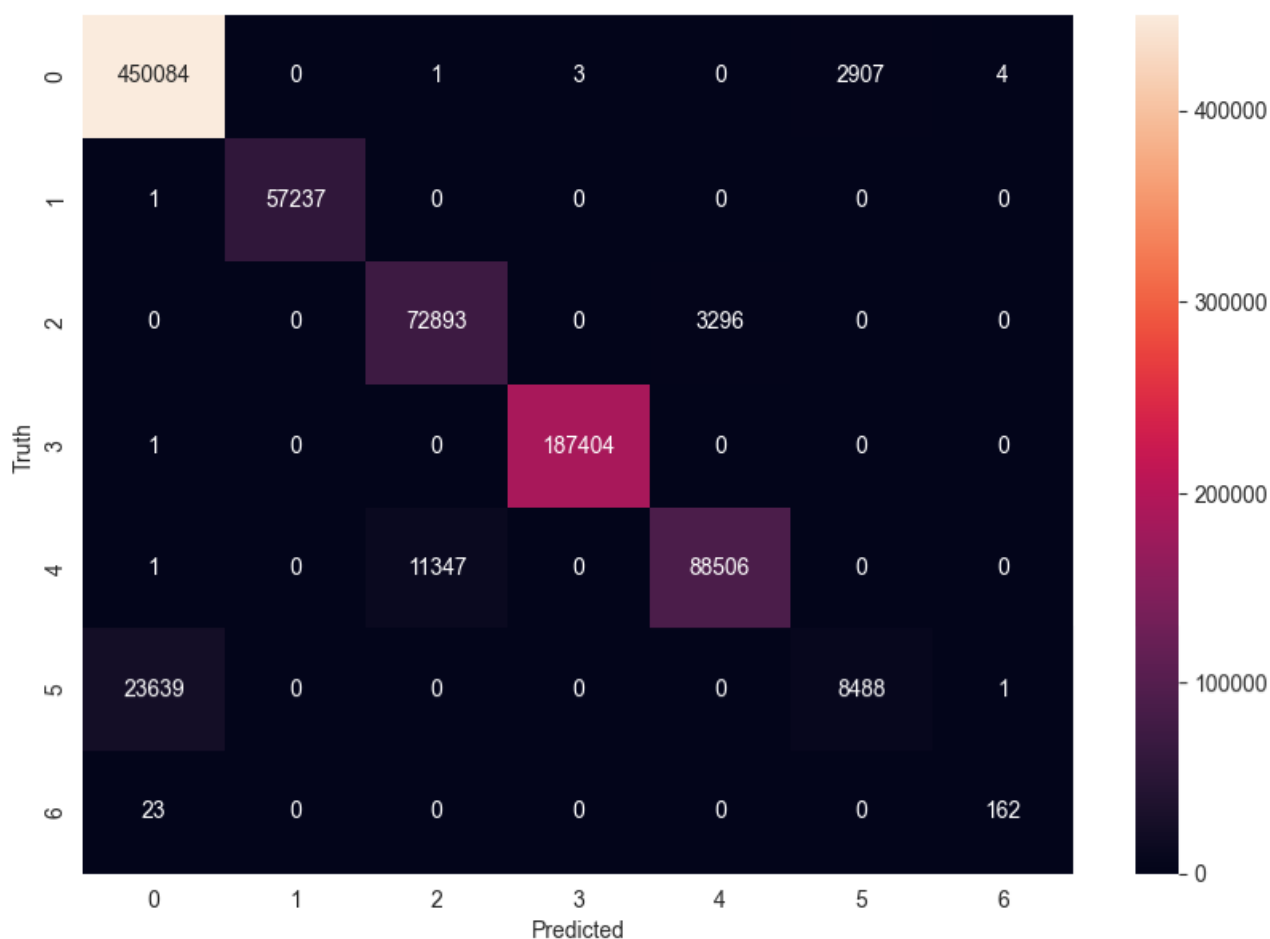


Рисунок 2.6 – Матриця невідповідностей для XGBoost

На матриці невідповідностей для XGBoost видно, що модель демонструє високу точність у класифікації, в більшості з мінімальною кількістю помилок. Помилкові класифікації для класу Infiltration знижені порівняно з попередніми моделями, хоча все ще присутні в достатньо великій кількості. Загалом, XGBoost показує покращену продуктивність, особливо у класифікації важких класів, таких як Web attack та Infiltration.



Рисунок 2.7 – Матриця невідповідностей для LightGBM

Для випадку LightGBM видно, що модель має значні проблеми з класифікацією класів Brute-force та Infiltration, показуючи велику кількість помилок. Зокрема, багато зразків з класу Brute-force неправильно класифікуються як DoS attack та Web attack. Клас Infiltration також має значну кількість хибних спрацьовувань у класах Benign та DoS attack. Це свідчить про те, що LightGBM менш ефективний у розрізненні цих класів у порівнянні з іншими моделями. Незважаючи на це, модель добре справляється з класифікацією класів Benign, DDoS attack та DoS attack, що показує наявність сильний сторін у задачі виявлення певних типів атак.

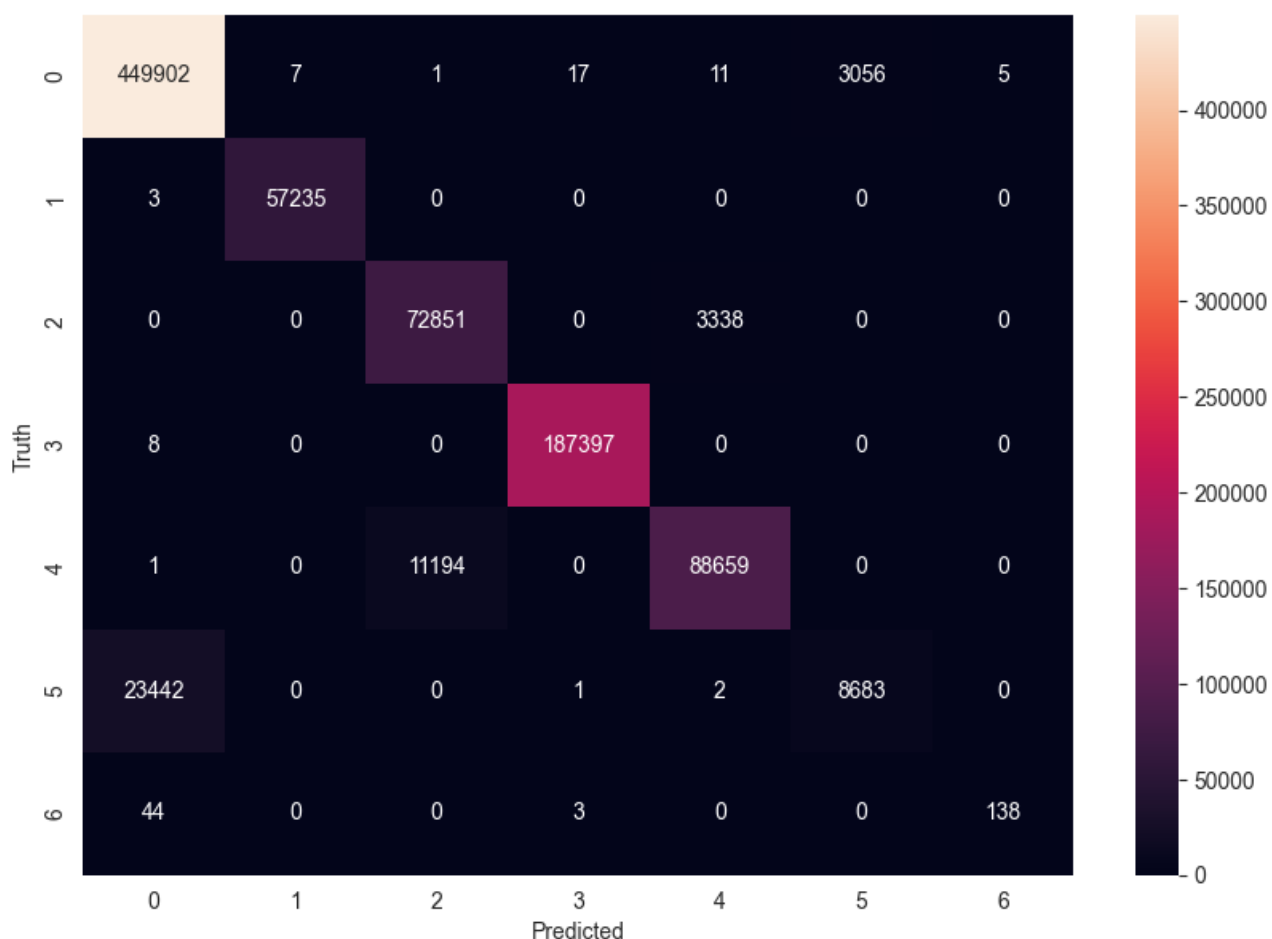


Рисунок 2.8 – Матриця невідповідностей для CatBoost

CatBoost має добрі результати у класифікації для більшості випадків. Особливо добре CatBoost справляється з виявленням класів Benign, DDoS attack та DoS attack. Однак, помітні певні труднощі з класифікацією класів Infiltration та Web attack, де значна частка зразків була помилково класифікована як Benign. Крім того, модель має проблеми з розпізнаванням зразків з класу Brute-force, деяка кількість з яких були класифіковані як Benign або DoS attack. В цілому, CatBoost показує високий рівень точності та добрі результати у більшості класів, але все ще має деякі труднощі з певними типами атак.

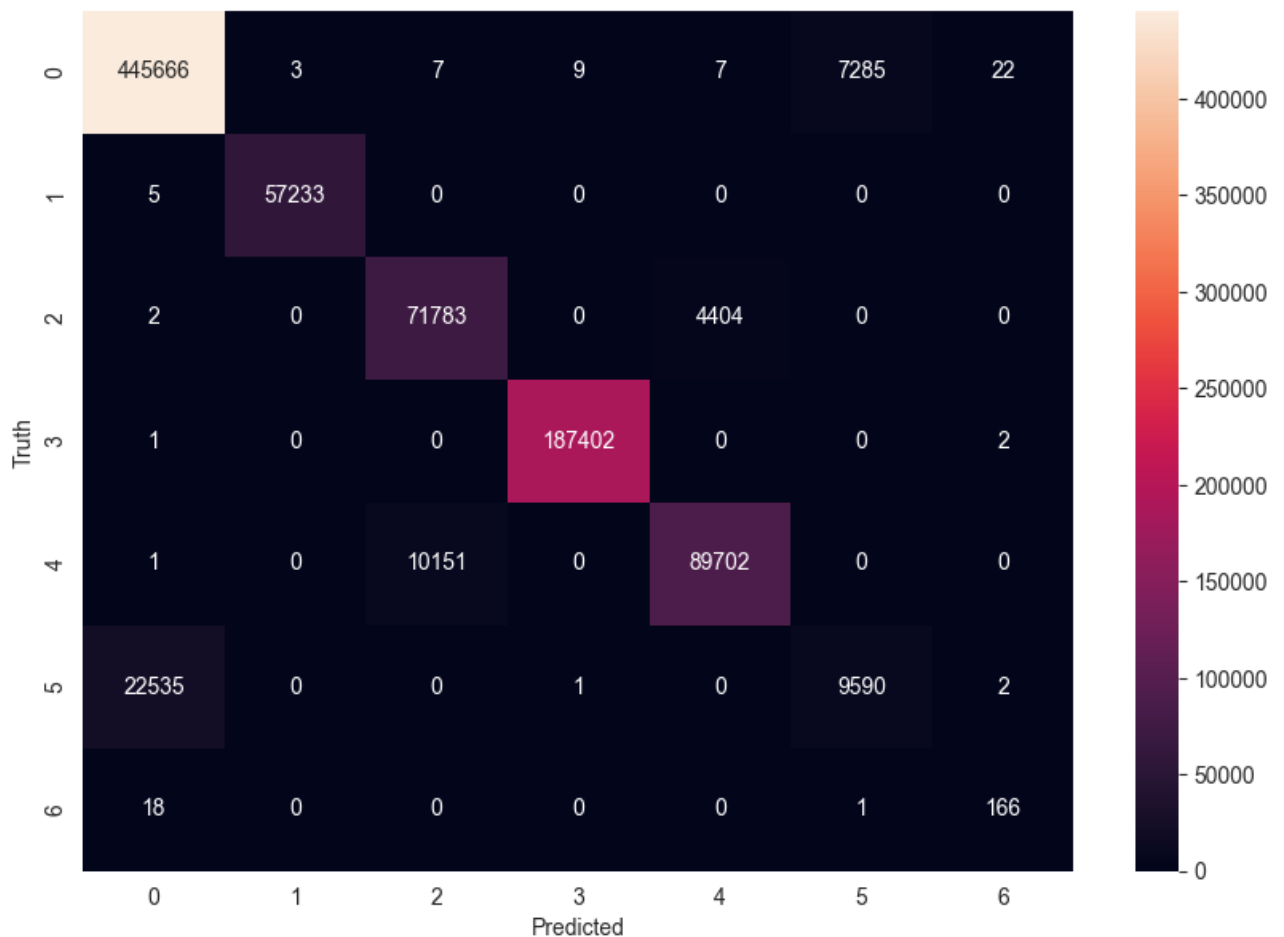


Рисунок 2.9 – Матриця невідповідностей для KNN

Загалом, KNN показує гарні результати, проте значна кількість зразків з класу Brute-force була помилково класифікована як Benign або DoS attack, що вказує на труднощі KNN з розпізнаванням цього типу атаки. В цілому, KNN показує високу точність у розпізнаванні основних класів, але має деякі проблеми з деякими типами атак.

- Аналіз ROC-AUC:

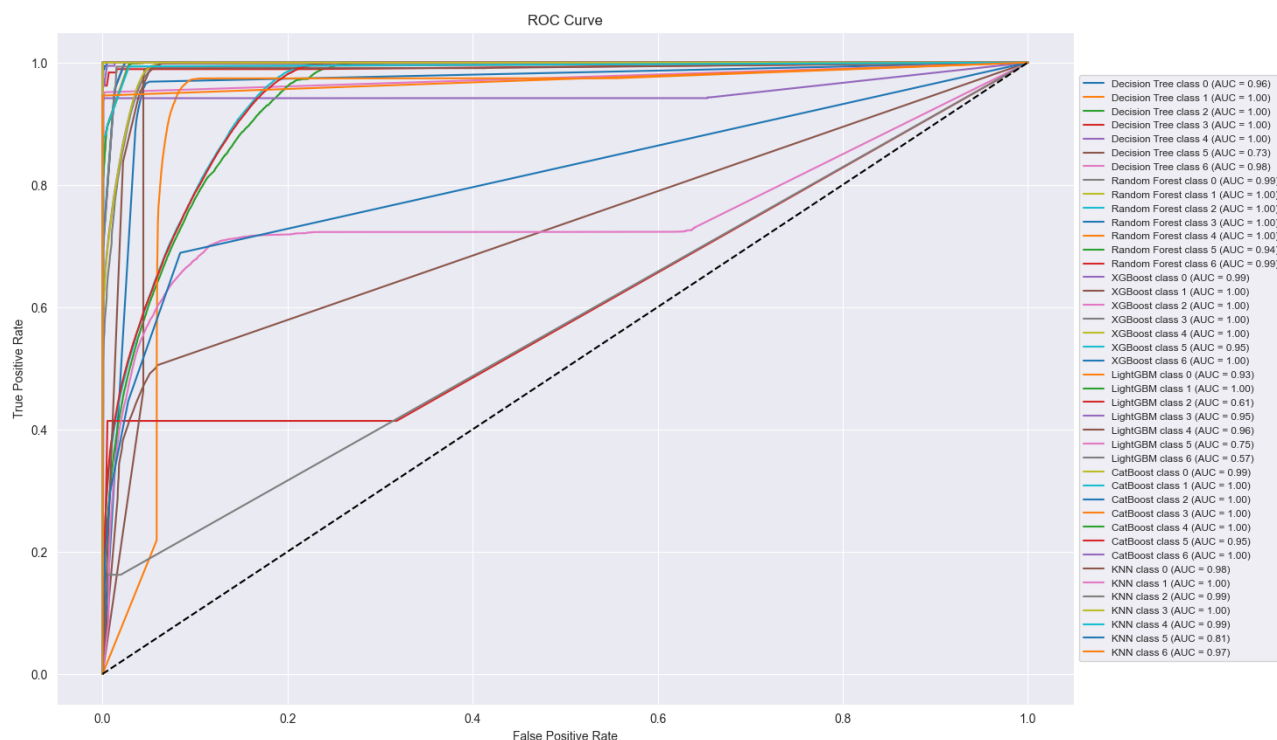


Рисунок 2.10 – ROC-AUC крива

На графіку ROC-кривої площі під AUC можемо бачити, що моделі XGBoost і CatBoost показують особливо високі значення для всіх класів, що підтверджує їх ефективність. Модель LightGBM має нижчі значення у порівнянні з іншими моделями, особливо для класів Brute-force та Web attack. Інші моделі, такі як DT, RF та KNN, також показують високі значення, хоча і з деякими відмінностями між класами.

- Аналіз Precision-Recall:

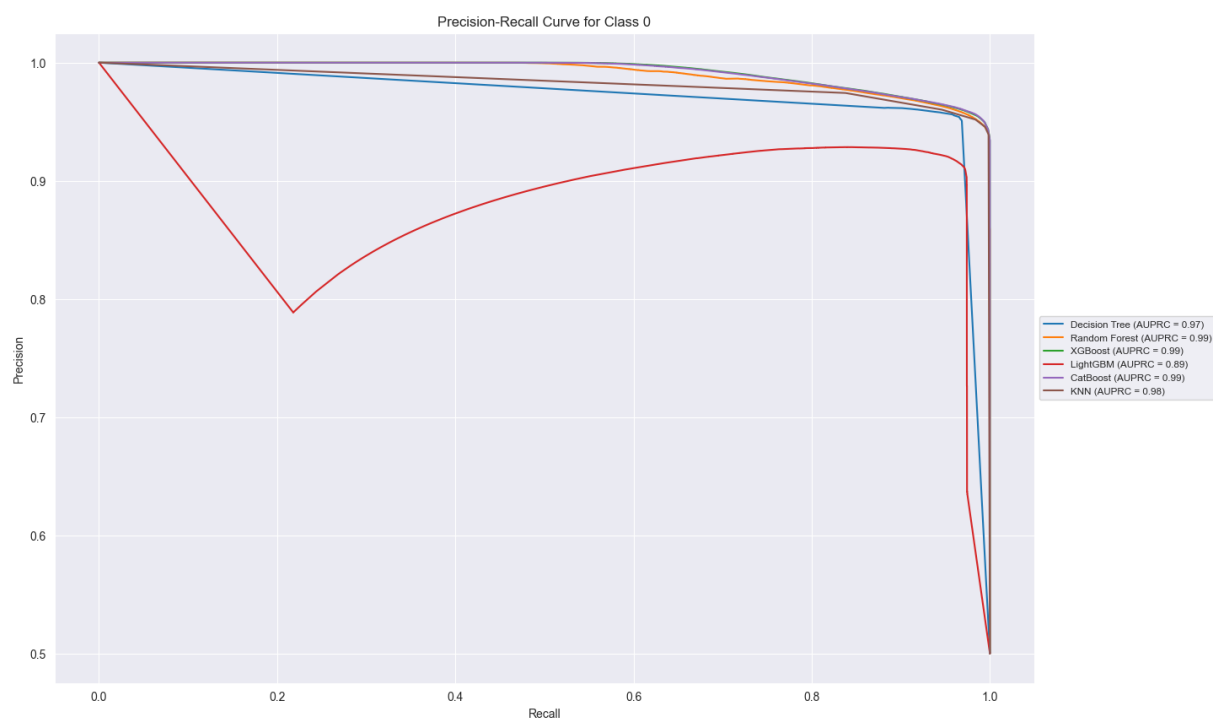


Рисунок 2.11 – Precision-Recall для класу 0

На графіку Precision-Recall для класу Benign (0) видно, що моделі XGBoost, CatBoost, і RF мають високі значення площі під кривою, що свідчить про їх високу точність та ефективність для виявлення звичайного трафіку. DT та KNN також демонструють високі результати. Модель LightGBM має нижчий показник, що вказує на дещо гіршу, але все ще достатню здатність класифікувати безпечний трафік.

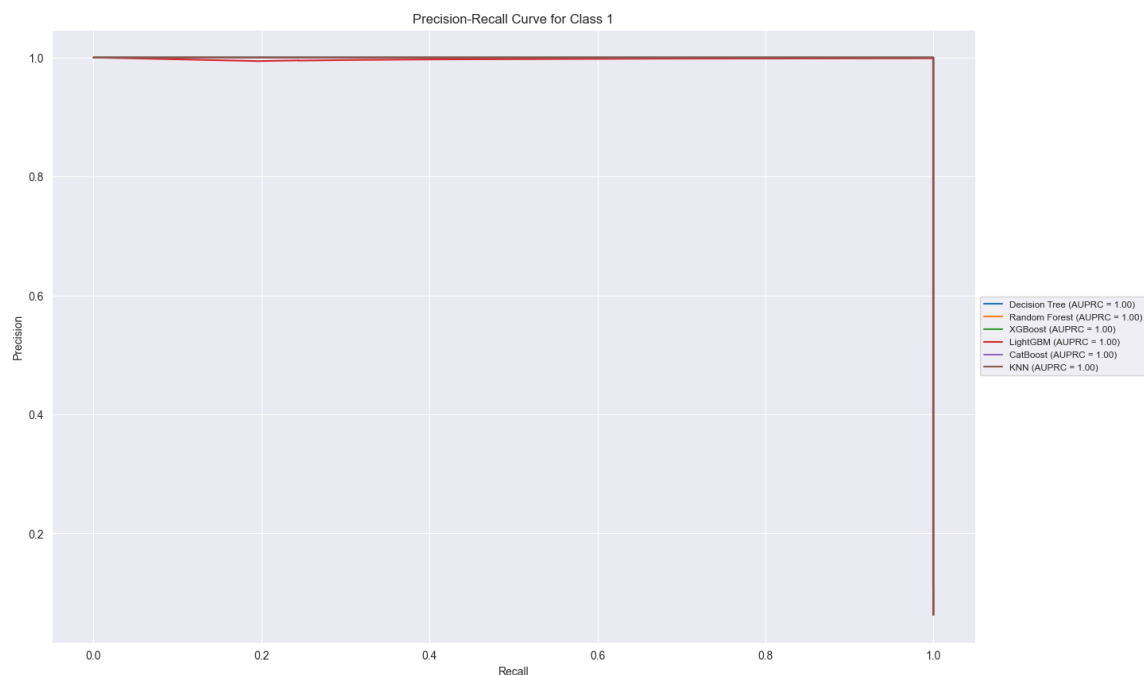


Рисунок 2.12 – Precision-Recall для класу 1

Для класу Botnet (1) всі моделі мають ідеальні значення площі (AUPRC = 1.00). З цього випливає, що всі моделі мають максимальну точність та ідеально класифікували ботнет атаки на визначеному трафіку. Це є ідеальним результатом і показує, що моделі добре розпізнають ботнет атаки в даному наборі даних.

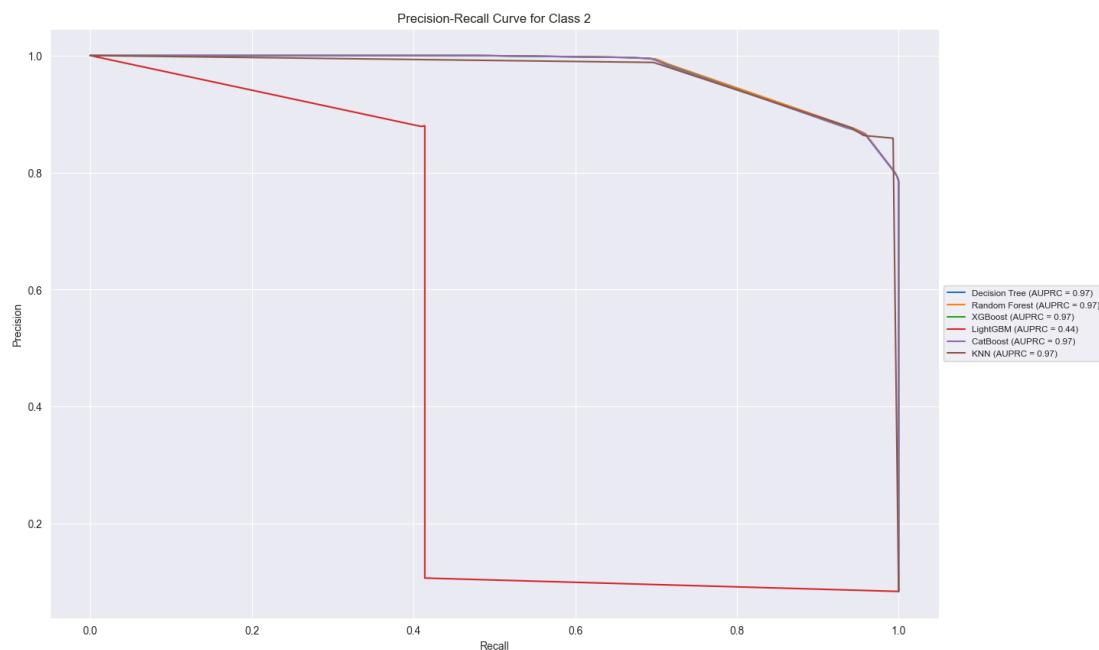


Рисунок 2.13 – Precision-Recall для класу 2

На графіку для класу Brute-force (2) майже всі моделі демонструють дуже високі значення, за винятком LightGBM, яка має значно нижчий показник ($AUPRC = 0.44$). Моделі, крім LightGBM, здатні з високою точністю і відгуком ідентифікувати атаки цього класу.

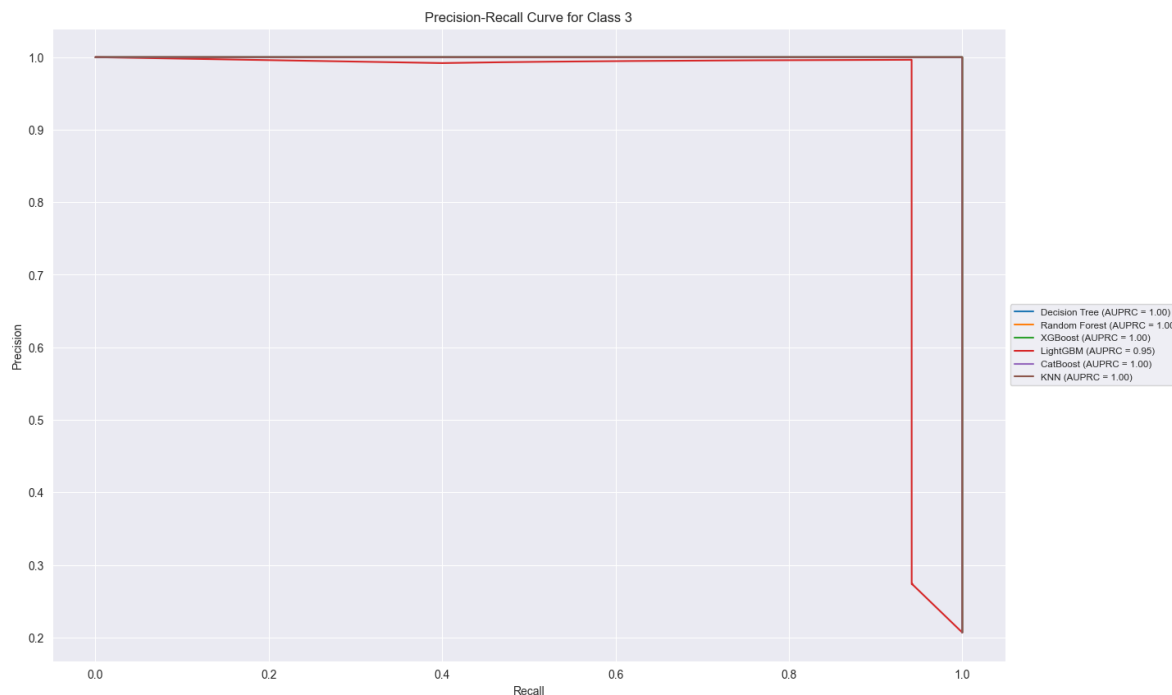


Рисунок 2.14 – Precision-Recall для класу 3

Графік для класу DDos attack (3) показує, що всі моделі мають майже ідеальні значення, за винятком LightGBM, яка має трохи нижче значення ($AUPRC = 0.95$). Всі моделі, здатні дуже ефективно ідентифікувати атаки DDoS з високою точністю. LightGBM, хоча й дещо відстає, все ще показує дуже хороші результати для цього класу.

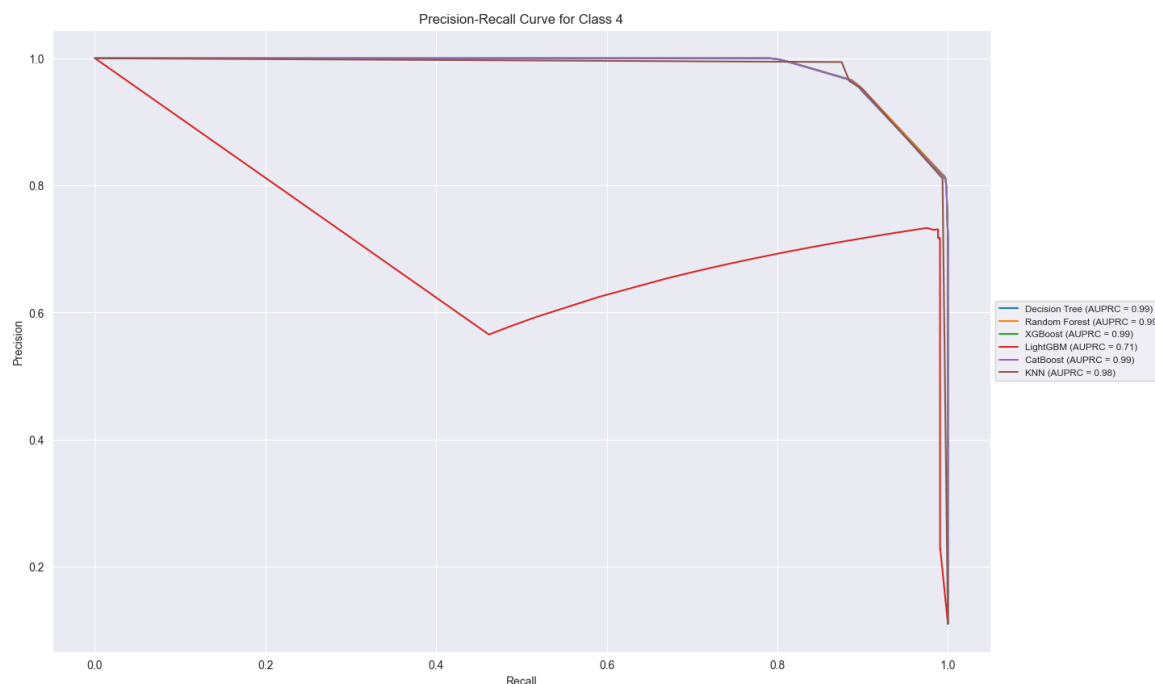


Рисунок 2.15 – Precision-Recall для класу 4

Графік Precision-Recall для класу DoS attack (4) показує, що всі моделі, за винятком LightGBM, демонструють високі показники, наближаючись до значення 0.99. LightGBM, з AUPRC = 0.71, відчутно відстає від інших моделей. LightGBM має значно нижчу точність відгуку для цього класу порівняно з іншими моделями.

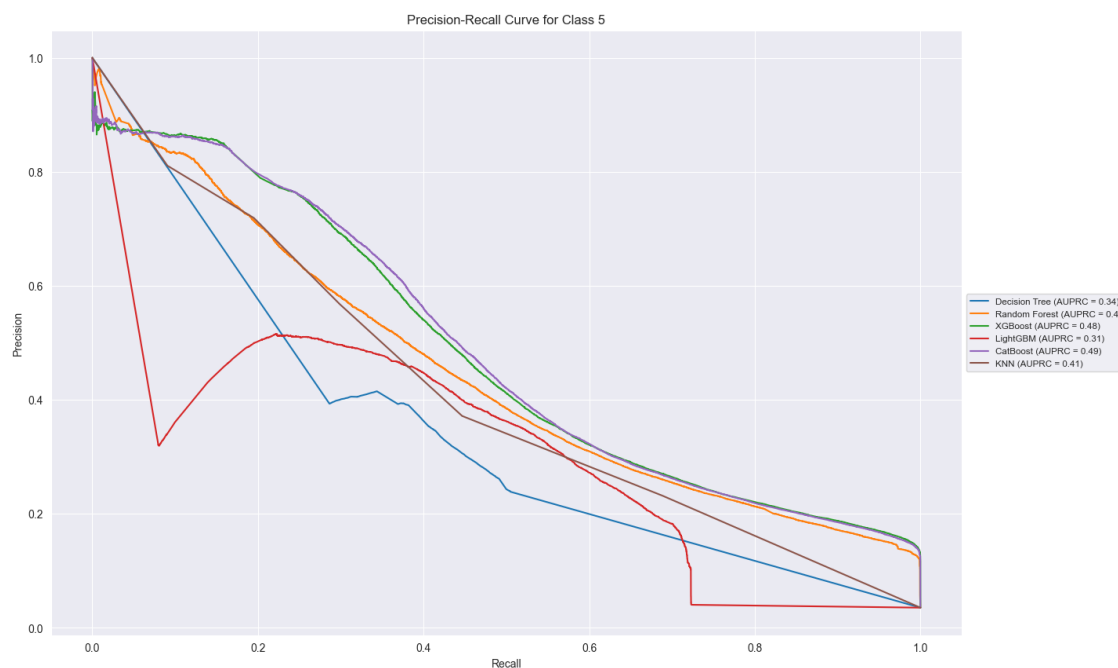


Рисунок 2.16 – Precision-Recall для класу 5

Клас Infiltration (5) є найбільш проблемним для всіх моделей. Графік показує, що жодна з них не демонструє високих значень. DT має найнижче значення AUPRC = 0.34, що свідчить про слабку точність для цього класу. RF (AUPRC = 0.45), XGBoost (AUPRC = 0.48), CatBoost (AUPRC = 0.49), LightGBM (AUPRC = 0.31), і KNN (AUPRC = 0.41) також показують порівняно низьку продуктивність для виявлення атак типу Infiltration. Моделі мають труднощі з правильною класифікацією цього типу атак.

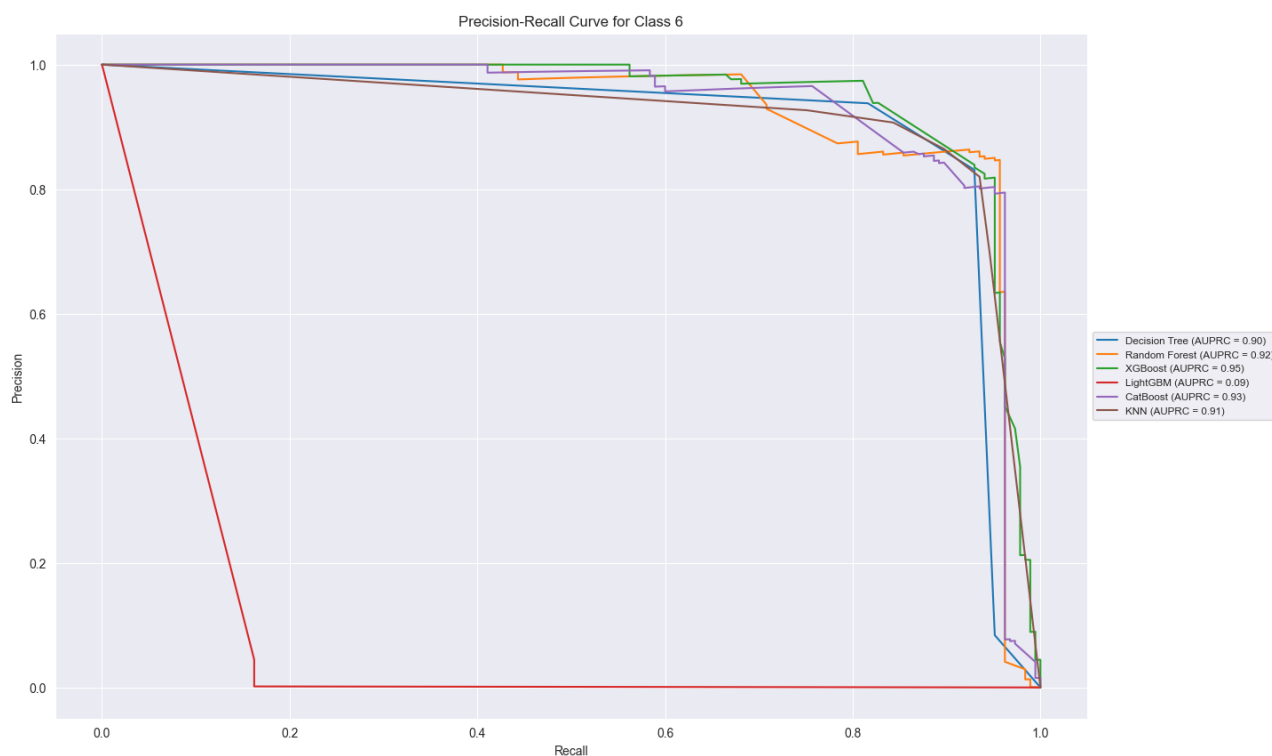


Рисунок 2.17 – Precision-Recall для класу 6

У випадку із Web attack (клас 6) Precision-Recall графік показує, що моделі мають високу ефективність при виявленні цього типу атак. Decision Tree має AUPRC = 0.90, Random Forest = 0.92, XGBoost = 0.95, LightGBM = 0.09, CatBoost = 0.93, і KNN = 0.91. Найкращу продуктивність демонструють XGBoost та CatBoost. LightGBM показує дуже низьку ефективність для цього класу, що свідчить про великі складнощі у виявленні класу Web attack для цієї моделі.

2.4.3 Вибір моделі

За результатами тестування моделей, найкраще показали себе XGBoost та CatBoost. Обидві моделі мають доволі схожі результати для вирішення задачі мультикласифікації веб-атак.

Проте, при додатковому тестуванні моделей на невеликих обсягах даних (було вибрано 5000 записів), продуктивність CatBoost відносно XGBoost значно знизилась. Зокрема, точність CatBoost падала до значень від 0.92 до 0.89, при цьому XGBoost залишався на стабільному рівні 0.95-0.96.

Саме стабільність роботи XGBoost стала вирішальним фактором для вибору саме цієї моделі для реалізації системи виявлення веб-атак на цих даних.

Висновки до розділу 2

У 2 розділі було детально розглянуто процес підготовки даних для навчання на них моделей машинного навчання, що включало в себе вибір необхідних ознак, їх нормалізацію та підготовку до подальшого аналізу. Вибраний був датасет CSE-CIC-IDS2018 через його репрезентативність та різноманітність представлених атак.

Для визначення найбільш ефективної моделі було розглянуто кілька метрик оцінювання, що включали точність, F1-міру, влучність, повноту, ROC-AUC, MCC, збалансовану точність, Cohen's Карра, логарифмічна втрата, специфічність та FNR.

За результатами експериментів – моделі XGBoost та CatBoost продемонстрували найвищі показники за більшістю метрик. Обидві моделі мають високу здатність правильно класифікувати різні типи атак. Однак під час експерименту з невеликою кількістю записів (5000) продуктивність CatBoost значно знизилася, на відміну від XGBoost, через що його і було вибрано як модель для реалізації системи виявлення.

Цей розділ підкреслює важливість ретельної обробки та аналізу даних, а також розглядає основні метрики, які дозволяють якісно оцінити модель машинного навчання, та вибрати підходящу для класифікації веб-атак на конкретних даних.

3 SIEM СИСТЕМИ. ІНТЕГРАЦІЯ З МОДЕЛЛЮ

SIEM є ключовими інструментами для забезпечення захисту інформаційних систем. Вони дозволяють збирати та аналізувати дані, за допомогою чого аналітики мають можливість оперативно виявляти та реагувати на інциденти безпеки. Інтеграція моделі машинного навчання з SIEM є заключним етапом створення ефективної системи виявлення вторгнень.

3.1 Огляд SIEM систем та їх роль

Система управління подіями безпеки (SIEM) є центральним елементом сучасного підходу для забезпечення інформаційної безпеки. Ці системи існують вже більше 10 років та дозволяють збирати, аналізувати, встановлювати кореляцію логів з подіями безпеки з різноманітних джерел, таких як мережеві пристрої, сервери, бази даних тощо. Ці системи дозволяють організаціям комплексно підходити до питання захисту своїх активів.

Існує доволі велика кількість SIEM системи, одними ж з найпопулярнішими є:

- **ELK (Elasticsearch, Logstash, Kibana):** SIEM система з відкритим кодом. ELK стек є гнучким, легко масштабується та має можливість тонкого налаштування для потреб організації, що і зробило це рішення одним з найпопулярніших виборів.
- **Splunk:** це потужне рішення, яке також користується великим попитом, через гарну інтеграцію з джерелами логів та можливістю легкого масштабування.
- **IBM QRadar:** пропонує розширені можливості кореляції подій та інтеграцію з іншими інструментами безпеки, що робить його ефективним для комплексного захисту.
- **ArcSight:** забезпечує високу продуктивність та детальний аналіз подій, завдяки розширеним можливостям кореляції та аналізу, що зробило його популярним вибором серед великих підприємств.

Для інтеграції була обрана система ELK. Вибір на її користь був наданий через її гнучкість, можливість детального налаштування та відкриту архітектуру. Ця SIEM система дозволяє легко інтегрувати нові джерела даних та налаштовувати обробку логів під специфічні потреби. Завдяки своїй масштабованості ELK може обробляти великі обсяги даних, що також робить її логічним вибором для використання в рамках цього дослідження.

Інтеграція моделі машинного навчання з SIEM системою є критично важливим шляхом для побудови ефективної системи виявлення веб-атак. З одного боку, навчені моделі машинного навчання здатні ефективно виявляти веб-атаки завдяки можливості самонавчання та адаптації до нових видів атак. З іншого ж боку, сама модель не може забезпечити постійний моніторинг та реагування на інциденти. Саме тут на допомогу приходять SIEM системи, використання яких є вкрай важливим для побудови працюючої системи виявлення, адже дозволяє аналізувати великі обсяги даних аналітикам та забезпечує більш глибоке розуміння безпекових інцидентів, будуючи кореляцію подій з різних джерел.

3.2 Процес інтеграції моделі з ELK

3.2.1 Вибір моделі

Архітектура для інтеграції з SIEM системою ELK має в собі елементи, кожен з яких виконує функції для забезпечення ефективного збору, аналізу та візуалізації даних.

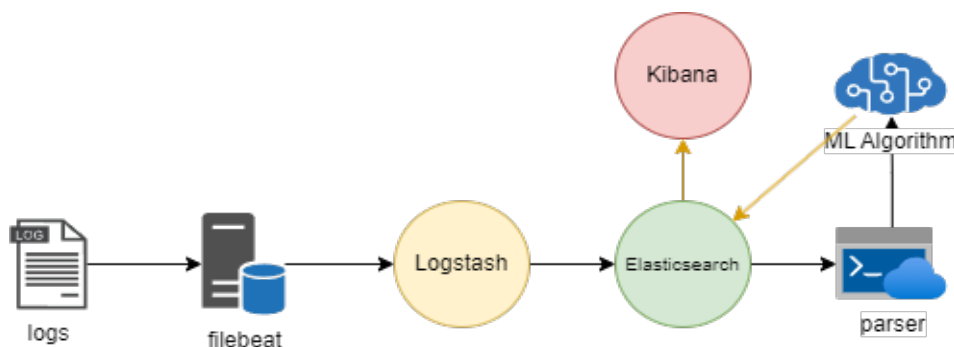


Рисунок 3.1 – Архітектура роботи ELK з алгоритмом машинного навчання

1. Джерелами логів можуть бути сервери, веб-додатки, бази даних та інші пристрої, що використовують журнали подій. Ці джерела є початковими точками для збору даних, які будуть аналізуватись надалі.
2. Filebeat постійно моніторить файли логів на наявність нових записів і передає такі до Logstash, де вони проходять подальшу обробку.
3. Logstash в свою чергу трансформує логи. Приймаючи дані з різних джерел, Logstash структурує їх в один формат, що вкрай необхідно для коректної подальшої роботи.
4. Elasticsearch зберігає логи та дозволяє працювати з ними. Саме сюди вони потрапляються після форматування в Logstash. На цьому етапі логи стають доступними для проведення подальшого аналізу.
5. Парсер періодично робить запити до Elasticsearch, перевіряючи наявність нових логів, за наявності таких – додатково форматує дані за потреби та передає на аналіз до алгоритму машинного навчання. Також парсер та відсилає вже оброблені логи назад до Elasticsearch.
6. Модель аналізує логи на наявність аномалій, шаблонів атак тощо. Після аналізу – додає висновок до кожного з логів.
7. Після отримання звіту від парсеру, Kibana візуалізує дані та дозволяє ефективно оцінити тренди атак та їх критичність для аналітиків безпеки.

Особливу увагу слід приділяти на правильне налаштування обробки даних, адже саме це є початковим, але критичним етапом аналізу даних, який дозволяє алгоритму машинного навчання ефективно працювати.

3.2.4 Огляд роботи парсера

Парсер є ключовим компонентом, що забезпечує автоматизацію процесу збору, аналізу та збереження логів.

Для підключення використовується API клієнта Elasticsearch. Після ініціалізації клієнта, парсер починає перевіряти наявність нових логів у Elasticsearch за допомогою запитів кожні 2 хвилини. При виявленні нових даних,

вони завантажуються у вигляд, прийнятний для аналізу алгоритмом та передаються навчені моделі машинного навчання, в нашому випадку – XGBoost.

Модель, в свою чергу, класифікує дані та визначає веб-атаки. Цей процес включає надання логам міток (Benign для нешкідливого трафіку або клас веб-атаки) та визначає рівень загрози в залежності від типу атаки (Low, Medium, High).

Після проведення аналізу, результат роботи моделі, а саме класифіковані дані з визначеними веб-атаками та рівнем небезпеки зберігаються в окремий індекс, попередньо приводяться парсером у вигляд, придатний для роботи з Elasticsearch. Для цього використовується Bulk API.

3.3 Огляд результатів

Після отримання нових логів, аналізу їх алгоритмом та надання класифікації і рівня загрози логи повертаються в Elasticsearch та зберігаються в окремому індексі. Це дозволяє використовувати саме проаналізовані логи для побудови ефективної системи моніторингу за допомогою інструментів Elasticsearch та Kibana, зокрема створенням дашбордів та візуалізації даних.

Для представлення даних у зручному форматі та можливості комплексної оцінки стану системи, трендів трафіку організації та додаткової інформації було створено дашборд, який візуалізував ці дані в зручній для сприйняття формі.

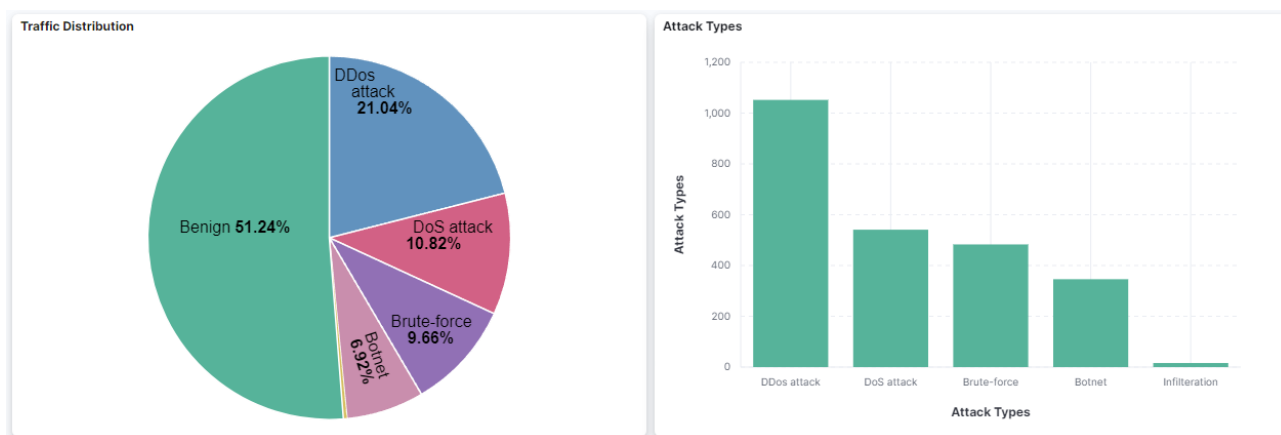


Рисунок 3.2 – Діаграми розподілу трафіку та тип атак

Розподіл трафіку дозволяє швидко оцінити загальну картину та визначити частоту різних атак. Типи атак дозволяють більше детально оцінити саме найбільш поширені атаки на систему.

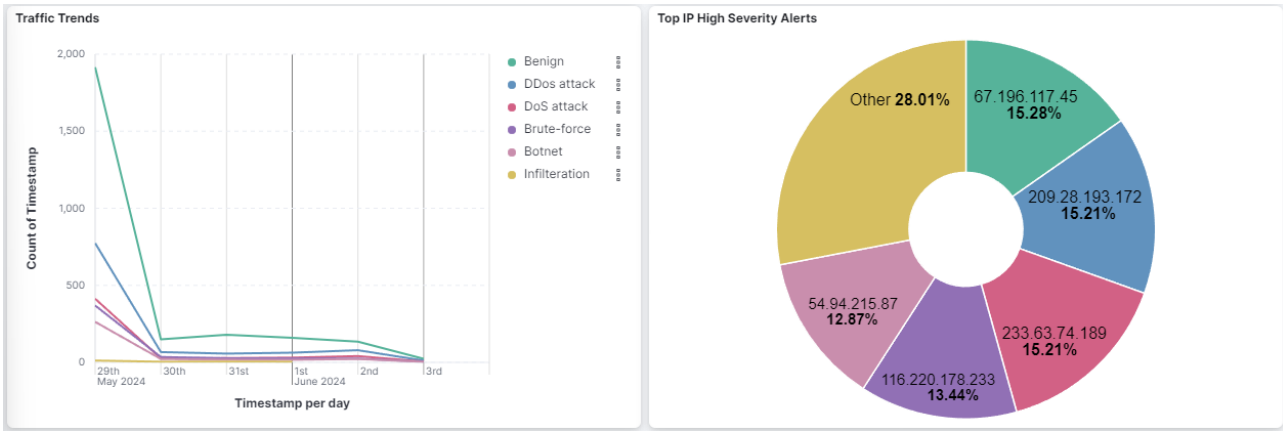


Рисунок 3.3 – Діаграми трендів трафіку та IP адрес з високим рівнем загроз

Тренди трафіку дозволяють побачити часову його зміну та наглядне співвідношення безпечного трафіку до атак. IP адресу з високим рівнем показують адреси, які найбільше спричиняли високорівневі загрози. Їх ідентифікація дозволяє вживати заходів їх блокування.

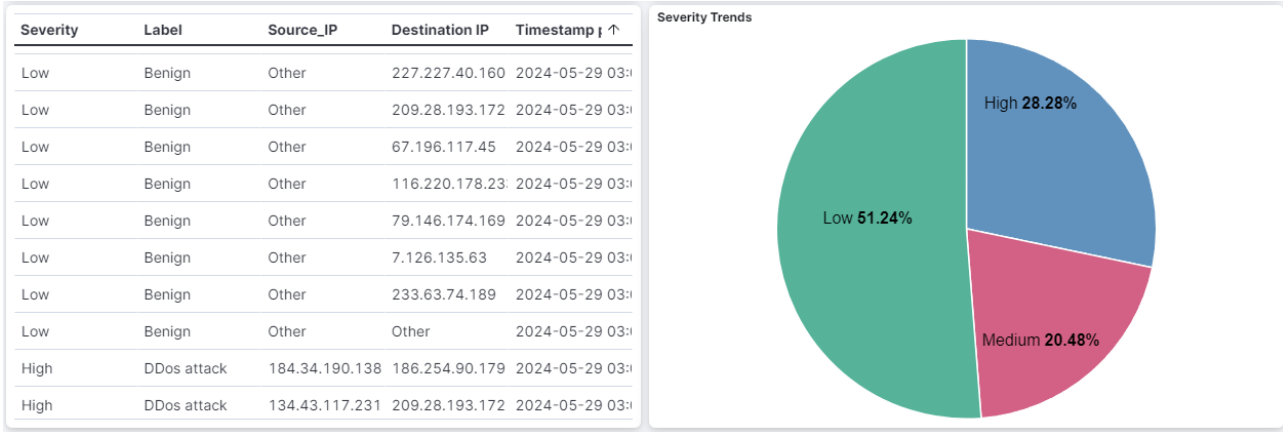


Рисунок 3.4 – Діаграми розподілу даних за рівнем загроз та таблиця з відображенням логів

Розподіл за рівнем загроз дає змогу оцінити загальний стан системи, таблиця з логами дозволяє більш детально ознайомитись з інформацією конкретного альорту для аналізу безпекової ситуації.

Таким чином, реалізація цих графіків у Kibana дозволяє проводити моніторинг та аналіз трафіку у реальному часі, виявляти аномалії, ідентифікувати джерела загроз та приймати оперативні заходи для їх нейтралізації.

Висновки до розділу 3

Для реалізації системи виявлення веб-атак була вибрана SIEM система ELK, через відкриту архітектуру, масштабованість та гнучкість. ELK стек взяв на себе збір даних, трансформовування та збереження.

Парсер, за наявності надходження нових логів, передавав їх у підходящому форматі до моделі машинного навчання. Було розглянуто процес аналізу логів та формування звітів, що передаються в Elasticsearch.

Розробка дашбордів у Kibana дозволив візуалізувати дані та отримати систему, що дозволяє проводити ефективну оцінку трендів атак, критичність, стан системи, розподілу трафіка та можливість оперативного реагування на інциденти.

Ця інтеграція дозволила перейти від просто навченого алгоритму машинного навчання до комплексної системи виявлення веб-атак, що забезпечує постійний моніторинг, аналіз та візуалізацію даних у реальному часі. Завдяки поєднанню можливостей машинного навчання з потужністю SIEM системи, вдалося створити інструмент для виявлення та аналізу веб-атак.

ВИСНОВКИ

У даній роботі розглянуто використання методів машинного навчання для вирішення задачі детектування веб-атак та подальшої інтеграції отриманої моделі з системами SIEM системами. Було розглянуто різні типи веб-атак, їх особливості. Оглянуті традиційні методи виявлення, включаючи сигнатурні та евристичні підходи, а також міжмережеві екрани та IDS системи. Основною метою дослідження було розробка ефективної моделі машинного навчання, яка здатна знижувати рівень хибно позитивних та хибно негативних спрацьовувань при виявленні веб-атак та налагодження її роботи з SIEM системою.

На початковому етапі виконання роботи було зібрано та оброблено дані з датасету CSE-CIC-IDS2018. Цей датасет містить детальні та наближені до реальних умов дані про різні типи атак. Дані були очищені, нормалізовані, перетворені та підготовлені для подальшого аналізу.

Було обрано та протестовано кілька моделей машинного навчання, що включали DT, RF, XGBoost, LightGBM, CatBoost та KNN. Для оцінки ефективності роботи моделей використовувалися метрики, такі як точність, влучність, повнота, F1-міра, ROC-AUC, логарифмічна втрата та інші. Найкращі результати показали моделі XGBoost та CatBoost.

Після додаткового тестування моделей на невеликих обсягах даних при імітації роботи в реальних умовах, було встановлено, що XGBoost показує стабільніші результати та постійно високу точність, що і зробило його оптимальним вибором для інтеграції в систему виявлення вую-атак.

На заключному етапі роботи було проведено інтеграцію обраної моделі з SIEM системою ELK, що дозволило автоматично завантажувати нові логи, аналізувати їх та візуалізувати результати в Kibana.

Важливо зазначити, що розроблений підхід та архітектура є гнучкими та можуть бути адаптовані для використання не тільки з ELK та вибраним

датасетом, але і з іншими популярними SIEM-системами. Це дозволяє організаціям вибирати найбільш підходящий інструмент для своїх потреб, зберігаючи високу ефективність роботи та плюси виявлення веб-атак алгоритмами машинного навчання.

Таким чином, у цій роботі було продемонстровано перспективність використання методів машинного навчання для виявлення веб-атак та їх інтеграції з SIEM-системами, що дозволяє зменшити обмеження і недоліки класичних систем виявлення атак та забезпечити надійний захист інформаційних систем.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ

1. Cloudflare. DDoS Threat Report Q4 2023 [Електронний ресурс]. – Режим доступу: <https://blog.cloudflare.com/ddos-threat-report-2023-q4>.
2. Verizon. 2022 Data Breach Investigations Report (DBIR) [Електронний ресурс]. – Режим доступу: <https://www.verizon.com/business/resources/reports/dbir/2022-data-breach-investigations-report-dbir.pdf>.
3. OWASP. Intrusion Detection [Електронний ресурс]. – Режим доступу: https://owasp.org/www-community/controls/Intrusion_Detection.
4. C. -Y. Ho, Y. -C. Lai, I. -W. Chen, F. -Y. Wang and W. -H. Tai, "Statistical analysis of false positives and false negatives from real traffic with intrusion detection/prevention systems," in IEEE Communications Magazine, vol. 50, no. 3, pp. 146-154, March 2012. [Електронний ресурс] – Режим доступу: <https://doi.org/10.1109/MCOM.2012.6163595>.
5. Rogers, Jeremy & Gunn, Steve. (2005). Identifying Feature Relevance Using a Random Forest. Subspace, Latent Structure and Feature Selection. [Електронний ресурс] – Режим доступу: <https://doi.org/10.48550/arXiv.2307.11544>.
6. Pulyala, S. R. . (2023). The Future of SIEM in a Machine Learning-Driven Cybersecurity Landscape. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 14(03), 1309–1314. [Електронний ресурс] – Режим доступу <https://doi.org/10.61841/turcomat.v14i03.14392>.
7. Adabi Raihan Muhammad, Parman Sukarno, Aulia Arif Wardana, Integrated Security Information and Event Management (SIEM) with Intrusion Detection System (IDS) for Live Analysis based on Machine Learning. Procedia Computer Science, Volume 217, 2023, Pages 1406-1415, ISSN 1877-0509. [Електронний ресурс] – Режим доступу: <https://www.sciencedirect.com/science/article/pii/S1877050922024243>.

8. Gutierrez-Garcia, J.L., Sanchez-DelaCruz, E., Pozos-Parra, M.d.P. (2023). A Review of Intrusion Detection Systems Using Machine Learning: Attacks, Algorithms and Challenges. In: Arai, K. (eds) Advances in Information and Communication. FICC 2023. Lecture Notes in Networks and Systems, vol 652. Springer, Cham. [Электронный ресурс] – Режим доступа: https://doi.org/10.1007/978-3-031-28073-3_5
9. Y. Xin et al., "Machine Learning and Deep Learning Methods for Cybersecurity," in IEEE Access, vol. 6, pp. 35365-35381, 2018. [Электронный ресурс] – Режим доступа: <https://doi.org/10.1109/ACCESS.2018.2836950>.
10. Zamani, Mahdi. (2013). Machine Learning Techniques for Intrusion Detection [Электронный ресурс]. – Режим доступа: <https://doi.org/10.48550/arXiv.1312.2177>

Додаток А

Програмний код підготовки датасету, навчання та тестування моделей:

```
import pandas as pd

import numpy as np

import json

from matplotlib import pyplot as plt

import seaborn as sns

from sklearn.metrics import (

    roc_auc_score, matthews_corrcoef, balanced_accuracy_score,

    cohen_kappa_score, confusion_matrix, precision_recall_curve, auc,

    classification_report, accuracy_score, f1_score, precision_score, recall_score,

    roc_curve, log_loss)

from sklearn.model_selection import train_test_split, StratifiedKFold,

cross_validate

from sklearn.preprocessing import MinMaxScaler

from sklearn.feature_selection import SelectFromModel

from imblearn.under_sampling import RandomUnderSampler

from sklearn.tree import DecisionTreeClassifier

from sklearn.utils import class_weight

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

from lightgbm import LGBMClassifier
```

```

from catboost import CatBoostClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import IsolationForest

from sklearn.neural_network import MLPClassifier

import joblib

import time

# Зчитування даних

df1 = pd.read_csv("Dataset/02-14-2018.csv", low_memory=False)

df2 = pd.read_csv("Dataset/02-15-2018.csv", low_memory=False)

df3 = pd.read_csv("Dataset/02-16-2018.csv", low_memory=False)

df4 = pd.read_csv("Dataset/02-20-2018.csv", low_memory=False)

df5 = pd.read_csv("Dataset/02-21-2018.csv", low_memory=False)

df6 = pd.read_csv("Dataset/02-22-2018.csv", low_memory=False)

df7 = pd.read_csv("Dataset/02-23-2018.csv", low_memory=False)

df8 = pd.read_csv("Dataset/02-28-2018.csv", low_memory=False)

df9 = pd.read_csv("Dataset/03-01-2018.csv", low_memory=False)

df10 = pd.read_csv("Dataset/03-02-2018.csv", low_memory=False)

def compare_columns(dfs):

    # Extract column sets for each DataFrame

    column_sets = {f'df{i + 1}': set(df.columns) for i, df in enumerate(dfs)}

```

```

# Dictionary to store unique columns for each DataFrame

unique_columns = {key: set() for key in column_sets}


# Compare columns

for name1, columns1 in column_sets.items():

    for name2, columns2 in column_sets.items():

        if name1 != name2:

            unique_columns[name1].update(columns1 - columns2)


# Filter out DataFrames with no unique columns

unique_columns = {k: v for k, v in unique_columns.items() if v}


# Output

if unique_columns:

    for df_name, cols in unique_columns.items():

        print(f"Unique columns in {df_name}: {list(cols)}")

else:

    print("All DataFrames have the same columns.")


# Compare columns

```

```
dataframes = [df1, df2, df3, df4, df5, df6, df7, df8, df9, df10]
```

```
compare_columns(dataframes)
```

```
del dataframes
```

```
# Drop columns that are not common in all DataFrames
```

```
df4.drop(columns=['Flow ID', 'Src IP', 'Src Port', 'Dst IP'], axis=1, inplace=True)
```

```
def fix_data_type(df):
```

```
    # Create a copy of the DataFrame to ensure all operations are performed on it
    directly.
```

```
    df = df[df['Dst Port'] != 'Dst Port'].copy()
```

```
    # Using .loc to safely modify the DataFrame directly
```

```
    int_columns = ['Dst Port', 'Protocol', 'Flow Duration', 'Tot Fwd Pkts', 'Tot Bwd
Pkts', 'TotLen Fwd Pkts', 'TotLen Bwd Pkts', 'Fwd Pkt Len Max', 'Fwd Pkt Len
Min', 'Bwd Pkt Len Max', 'Bwd Pkt Len Min', 'Flow IAT Max', 'Flow IAT Min',
'Fwd IAT Tot', 'Fwd IAT Max', 'Fwd IAT Min', 'Bwd IAT Tot', 'Bwd IAT Max',
'Bwd IAT Min', 'Fwd PSH Flags', 'Bwd PSH Flags', 'Fwd URG Flags', 'Bwd URG
Flags', 'Fwd Header Len', 'Bwd Header Len', 'Pkt Len Min', 'Pkt Len Max', 'FIN
Flag Cnt', 'SYN Flag Cnt', 'RST Flag Cnt', 'PSH Flag Cnt', 'ACK Flag Cnt', 'URG
Flag Cnt', 'CWE Flag Count', 'ECE Flag Cnt', 'Down/Up Ratio', 'Fwd Byts/b
Avg', 'Fwd Pkts/b Avg', 'Fwd Blk Rate Avg', 'Bwd Byts/b Avg', 'Bwd Pkts/b
Avg', 'Bwd Blk Rate Avg', 'Subflow Fwd Pkts', 'Subflow Fwd Byts', 'Subflow
Bwd Pkts', 'Subflow Bwd Byts', 'Init Fwd Win Byts', 'Init Bwd Win Byts', 'Fwd
```

Act Data Pkts', 'Fwd Seg Size Min', 'Active Max', 'Active Min', 'Idle Max', 'Idle Min']

```
float_columns = ['Fwd Pkt Len Mean', 'Fwd Pkt Len Std', 'Bwd Pkt Len Mean',
                 'Bwd Pkt Len Std', 'Flow Byts/s', 'Flow Pkts/s', 'Flow IAT Mean', 'Flow IAT
                 Std', 'Fwd IAT Mean', 'Fwd IAT Std', 'Bwd IAT Mean', 'Bwd IAT Std', 'Fwd
                 Pkts/s', 'Bwd Pkts/s', 'Pkt Len Mean', 'Pkt Len Std', 'Pkt Len Var', 'Fwd Seg
                 Size Avg', 'Bwd Seg Size Avg', 'Active Mean', 'Active Std', 'Idle Mean', 'Idle
                 Std']
```

```
# Convert integer columns
```

```
for column in int_columns:
```

```
    df.loc[:, column] = df[column].astype(int)
```

```
# Convert float columns
```

```
for column in float_columns:
```

```
    df.loc[:, column] = df[column].astype(float)
```

```
return df
```

```
df_d1 = fix_data_type(df1)
```

```
df_d2 = fix_data_type(df2)
```

```
df_d3 = fix_data_type(df3)
```

```
df_d4 = fix_data_type(df4)
```

```
df_d5 = fix_data_type(df5)
```

```
df_d6 = fix_data_type(df6)
```

```
df_d7 = fix_data_type(df7)
```

```
df_d8 = fix_data_type(df8)
```

```
df_d9 = fix_data_type(df9)
```

```
df_d10 = fix_data_type(df10)
```

```
pd.set_option('future.no_silent_downcasting', True) # Raise errors when
downcasting
```

```
def drop_infinite_null(df, df_index):
```

```
    print(df_index, df.shape)
```

```
    # Replace "Infinity" and "-Infinity" strings directly with np.nan
```

```
    df.replace(["Infinity", "infinity", np.inf, -np.inf], np.nan, inplace=True)
```

```
    # Explicitly handle object conversion if data type consistency is needed
```

```
    df = df.infer_objects()
```

```
    # Drop all rows with any NaN values
```

```
    df.dropna(inplace=True)
```

```
    print(df_index, df.shape)
```

```
    return df
```

```
# Drop infinite and null values
```

```
df_d1 = drop_infinite_null(df_d1, 1)
```

```

df_d2 = drop_infinite_null(df_d2, 2)

df_d3 = drop_infinite_null(df_d3, 3)

df_d4 = drop_infinite_null(df_d4, 4)

df_d5 = drop_infinite_null(df_d5, 5)

df_d6 = drop_infinite_null(df_d6, 6)

df_d7 = drop_infinite_null(df_d7, 7)

df_d8 = drop_infinite_null(df_d8, 8)

df_d9 = drop_infinite_null(df_d9, 9)

df_d10 = drop_infinite_null(df_d10, 10)


# Timestamp column is dropped

columns_to_drop = ["Timestamp"]

for df in [df_d1, df_d2, df_d3, df_d4, df_d5, df_d6, df_d7, df_d8, df_d9, df_d10]:

    df.drop(columns=columns_to_drop, inplace=True)


def generate_binary_label(df):

    df['Bin_Label'] = df['Label'].apply(lambda x: "Benign" if x == 'Benign' else
"Malicious")

    print(df['Bin_Label'].value_counts())

    return df

```

```

# Create binary labels

df_d1 = generate_binary_label(df_d1)

df_d2 = generate_binary_label(df_d2)

df_d3 = generate_binary_label(df_d3)

df_d4 = generate_binary_label(df_d4)

df_d5 = generate_binary_label(df_d5)

df_d6 = generate_binary_label(df_d6)

df_d7 = generate_binary_label(df_d7)

df_d8 = generate_binary_label(df_d8)

df_d9 = generate_binary_label(df_d9)

df_d10 = generate_binary_label(df_d10)


mapping = {'SSH-Bruteforce': 'Brute-force',

           'FTP-BruteForce': 'Brute-force',

           ##### Brute-force

           'Brute Force -XSS': 'Web attack',

           'Brute Force -Web': 'Web attack',

           'SQL Injection': 'Web attack',

           ##### Web attack

           'DoS attacks-Hulk': 'DoS attack',

           'DoS attacks-SlowHTTPTest': 'DoS attack',

```

```

'DoS attacks-Slowloris': 'DoS attack',

'DoS attacks-GoldenEye': 'DoS attack',

##### DoS attack

'DDOS attack-HOIC': 'DDoS attack',

'DDOS attack-LOIC-UDP': 'DDoS attack',

'DDoS attacks-LOIC-HTTP': 'DDoS attack',

##### DDoS attack

'Bot': 'Botnet',

##### Botnet

'Infiltration': 'Infiltration',

##### Infiltration

'Benign': 'Benign',

'Label': 'Benign',

##### Benign

}

```

```

def transform_multi_label(df):

    print(df['Label'].value_counts())

    df['Label'] = df['Label'].map(mapping)

    return df

```

```

# Transform multi-class labels

df_d1 = transform_multi_label(df_d1)

df_d2 = transform_multi_label(df_d2)

df_d3 = transform_multi_label(df_d3)

df_d4 = transform_multi_label(df_d4)

df_d5 = transform_multi_label(df_d5)

df_d6 = transform_multi_label(df_d6)

df_d7 = transform_multi_label(df_d7)

df_d8 = transform_multi_label(df_d8)

df_d9 = transform_multi_label(df_d9)

df_d10 = transform_multi_label(df_d10)


def balance_data(df):

    X = df.drop(["Label"], axis=1) # Features

    y = df["Label"] # Target variable

    rus = RandomUnderSampler()

    X_balanced, y_balanced = rus.fit_resample(X, y) # resampling

    df = pd.concat([X_balanced, y_balanced], axis=1) # Concatenate the balanced
    data

    del X, y, X_balanced, y_balanced # Delete the variables to free up memory

```

```
print(df.shape)

print('_____')

print(df['Label'].value_counts())

return df
```

```
# Balance data
```

```
df_d1 = balance_data(df_d1)
```

```
df_d2 = balance_data(df_d2)
```

```
df_d3 = balance_data(df_d3)
```

```
df_d4 = balance_data(df_d4)
```

```
df_d5 = balance_data(df_d5)
```

```
df_d6 = balance_data(df_d6)
```

```
df_d7 = balance_data(df_d7)
```

```
df_d8 = balance_data(df_d8)
```

```
df_d9 = balance_data(df_d9)
```

```
df_d10 = balance_data(df_d10)
```

```
# Data aggregation
```

```
df_all = pd.concat([df_d1, df_d2, df_d3, df_d4, df_d5, df_d6, df_d7, df_d8,
df_d9, df_d10], axis=0)
```

```
df_all.reset_index(drop=True, inplace=True)
```

```
# Delete the variables to free up memory

variances = df_all.var(numeric_only=True)

constant_columns = variances[variances == 0].index

df_all = df_all.drop(constant_columns, axis=1)


# Heatmap of correlation matrix

plt.figure(figsize=(70, 70))

corr = df_all.corr(numeric_only=True)

sns.heatmap(corr, annot=True, cmap='RdBu', vmin=-1, vmax=1, square=True)

plt.show()


# Remove highly correlated columns

threshold = 0.90

correlated_col = set()

for i in range(len(corr.columns)):

    for j in range(i):

        if abs(corr.iloc[i, j]) >= threshold:

            correlated_col.add(corr.columns[j])

df_all.drop(correlated_col, axis=1, inplace=True)

# Heatmap of correlation matrix after removal

plt.figure(figsize=(70, 70))
```

```
corr = df_all.corr(numeric_only=True)

sns.heatmap(corr, annot=True, cmap='RdBu', vmin=-1, vmax=1, square=True)

plt.show()
```

```
# Save the cleaned data
```

```
feature_cols = list(df_all.columns)
```

```
feature_cols.remove("Bin_Label")
```

```
feature_cols.remove("Label")
```

```
# train-test split
```

```
train_df, test_df = train_test_split(df_all, test_size=0.2, random_state=2,
                                     shuffle=True, stratify=df_all["Label"])
```

```
del df_all
```

```
# Normalizing the data
```

```
minmax_scaler = MinMaxScaler()
```

```
train_df[feature_cols] = minmax_scaler.fit_transform(train_df[feature_cols])
```

```
test_df[feature_cols] = minmax_scaler.transform(test_df[feature_cols])
```

```
# Label encoding
```

```
labels_list = list(np.unique(train_df["Label"]))
```

```
label_order = np.array(labels_list)
```

```

class_weights = class_weight.compute_class_weight('balanced',
classes=label_order, y=train_df["Label"].values)

class_weights_dict = {k: float(v) for k, v in enumerate(class_weights)}

with open("class_weights.json", "w") as outfile:

    json.dump(class_weights_dict, outfile)


label_dict = {v: v for v in labels_list}

with open("label_dict.json", "w") as outfile:

    json.dump(label_dict, outfile)


y_train = [labels_list.index(k) for k in train_df["Label"]]
y_test = [labels_list.index(k) for k in test_df["Label"]]


# Dictionary to store the accuracy of each model

models_accuracy = {}


# Function to print the scores

def print_score(y_test, y_pred, y_prob, model_name, elapsed_time):

    metrics = {}


    # Calculate and print time

    print(f'Elapsed time: {elapsed_time:.2f} ms')

```

```

print(f'Results for {model_name}:')

metrics['Accuracy'] = accuracy_score(y_test, y_pred)

metrics['F1-Score'] = f1_score(y_test, y_pred, average='weighted')

metrics['Precision'] = precision_score(y_test, y_pred, average='weighted')

metrics['Recall'] = recall_score(y_test, y_pred, average='weighted')

metrics['ROC-AUC'] = roc_auc_score(y_test, y_prob, multi_class='ovr')

metrics['MCC'] = matthews_corrcoef(y_test, y_pred)

metrics['Balanced Accuracy'] = balanced_accuracy_score(y_test, y_pred)

metrics['Cohen\'s Kappa'] = cohen_kappa_score(y_test, y_pred)

metrics['Log Loss'] = log_loss(y_test, y_prob)


# Calculate and print specificity and FNR for each class

cm = confusion_matrix(y_test, y_pred)

specificity_list = []

fnr_list = []

for i in range(len(cm)):

    tp = cm[i, i]

    fp = np.sum(cm[:, i]) - tp

    fn = np.sum(cm[i, :]) - tp

    tn = np.sum(cm) - (tp + fp + fn)

```

```

specificity = tn / (tn + fp)

specificity_list.append(specificity)

fnr = fn / (fn + tp)

fnr_list.append(fnr)

metrics['Specificity'] = np.mean(specificity_list)

metrics['FNR'] = np.mean(fnr_list)

# Print the metrics

for metric_name, metric_value in metrics.items():

    print(f'{metric_name}: {metric_value}')

print("\n")

# Plot confusion matrix

plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt="d")

plt.xlabel('Predicted')

plt.ylabel('Truth')

plt.show()

models_accuracy[model_name] = metrics['Accuracy']

return metrics

# Function to evaluate model

def evaluate_model(model, model_name):

```

```

start_time = time.time()

model.fit(train_df[feature_cols], y_train)

elapsed_time = (time.time() - start_time) * 1000 # Time in milliseconds

y_pred = model.predict(test_df[feature_cols])

y_prob = model.predict_proba(test_df[feature_cols])

return print_score(y_test, y_pred, y_prob, model_name, elapsed_time)


# Initialize models

models = {

    "Decision Tree": DecisionTreeClassifier(random_state=2),

    "Random Forest": RandomForestClassifier(random_state=2),

    "XGBoost": XGBClassifier(random_state=2, use_label_encoder=False,
eval_metric='mlogloss'),

    "LightGBM": LGBMClassifier(random_state=2),

    "CatBoost": CatBoostClassifier(verbose=0, random_state=2),

    "KNN": KNeighborsClassifier()

}


# Store results

results = {}

# Train and evaluate each model

for model_name, model in models.items():

```

```

print(f'Training {model_name}...')

metrics = evaluate_model(model, model_name)

results[model_name] = metrics


# Print summary of all models

summary_df = pd.DataFrame(results).T

print("Summary of all models:")

print(summary_df)

# Visualization of the metrics

summary_df.plot(kind='bar', figsize=(15, 8), colormap='viridis')

plt.title("Comparison of Models by Metrics")

plt.xticks(rotation=45)

plt.ylabel("Score")

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), fontsize='small')

plt.show()


# ROC Curve

def plot_roc_curves(models, X_test, y_test, labels_list):

    plt.figure(figsize=(15, 10))

    for model_name, model in models.items():

        y_prob = model.predict_proba(X_test)

```

```

for i in range(len(labels_list)):

    fpr, tpr, _ = roc_curve(y_test, y_prob[:, i], pos_label=i)

    plt.plot(fpr, tpr, label=f'{model_name} class {i} (AUC = {auc(fpr,
tpr):.2f})')

plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), fontsize='small')

plt.show()

# Precision-Recall Curve

def plot_precision_recall_curves(models, X_test, y_test, labels_list):

    plt.figure(figsize=(15, 10))

    for model_name, model in models.items():

        y_prob = model.predict_proba(X_test)

        for i in range(len(labels_list)):

            precision, recall, _ = precision_recall_curve(y_test, y_prob[:, i], pos_label=i)

            plt.plot(recall, precision, label=f'{model_name} class {i} (AUPRC =
{auc(recall, precision):.2f})')

plt.xlabel('Recall')

plt.ylabel('Precision')

```

```

plt.title('Precision-Recall Curve')

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), fontsize='small')

plt.show()

# Calling functions to plot the graphs

plot_roc_curves(models, test_df[feature_cols], y_test, labels_list)

plot_precision_recall_curves(models, test_df[feature_cols], y_test, labels_list)

# Individual Precision-Recall Curve

def plot_individual_precision_recall_curves(models, X_test, y_test, labels_list):

    for i in range(len(labels_list)):

        plt.figure(figsize=(15, 10))

        for model_name, model in models.items():

            y_prob = model.predict_proba(X_test)

            precision, recall, _ = precision_recall_curve(y_test, y_prob[:, i],
                pos_label=i)

            plt.plot(recall, precision, label=f'{model_name} (AUPRC = {auc(recall,
                precision):.2f})')

        plt.xlabel('Recall')

        plt.ylabel('Precision')

        plt.title(f'Precision-Recall Curve for Class {i}')

        plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), fontsize='small')

```

```
plt.show()
```

```
# Calling functions to plot the graphs
```

```
plot_individual_precision_recall_curves(models, test_df[feature_cols], y_test,  
labels_list)
```

```
# Save best model
```

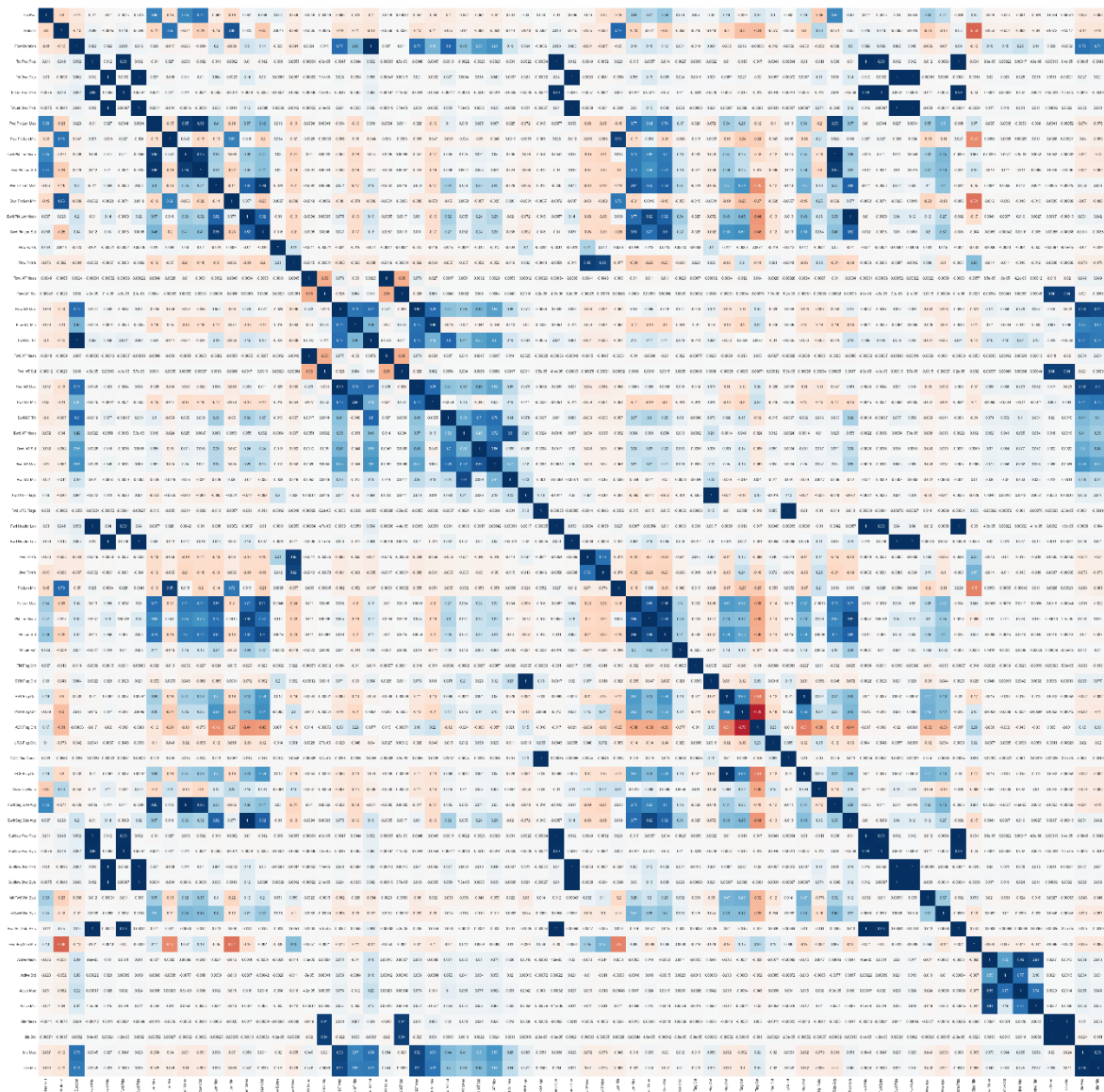
```
joblib.dump(model, "xgboost_model.pkl")
```

```
# Save feature columns
```

```
with open("feature_cols.json", "w") as outfile:
```

```
    json.dump(feature_cols, outfile)
```

Рисунок Б.1 – Теплова карта до видалення сильно корельованих стовпців





Додаток В

Програмний код роботи парсера:

```
import pandas as pd

from elasticsearch import Elasticsearch, helpers

import joblib

import json

import time


ELASTICSEARCH_HOST = 'localhost'

ELASTICSEARCH_PORT = 9200

ELASTICSEARCH_INDEX = 'sampled_logs'

ELASTICSEARCH_USER = 'elastic'

ELASTICSEARCH_PASSWORD = 'MHSkTC7ThvtVu9MenQFh'

# Ініціалізація клієнта Elasticsearch

es = Elasticsearch(

    [{'host': ELASTICSEARCH_HOST, 'port': ELASTICSEARCH_PORT,

      'scheme': 'http'}],

    basic_auth=(ELASTICSEARCH_USER, ELASTICSEARCH_PASSWORD)

)

# load XGBoost model

xgboost_model = joblib.load("xgboost_model.pkl")
```

```

# load feature columns

with open("feature_cols.json", "r") as infile:

    feature_cols = json.load(infile)


def predict_label(df_loaded, feature_cols, xgboost_model):

    print("\nAnalyzing data...")

    y_pred = xgboost_model.predict(df_loaded[feature_cols])

    # y_prob = xgboost_model.predict_proba(df_loaded[feature_cols])


    labels_list = ['Benign', 'Botnet', 'Brute-force', 'DDos attack', 'DoS attack',
'Infiltration', 'Web attack']

    predicted_labels = [labels_list[i] for i in y_pred]

    df_loaded['Predicted_Label'] = predicted_labels

    df_loaded['Severity'] = df_loaded['Predicted_Label'].apply(

        lambda x: 'Low' if x == 'Benign' else 'Medium' if x == 'DoS attack' or x ==
'Brute-force' else 'High')

    print("Analysis complete")

    return df_loaded


def generate_elasticsearch_docs(dataframe, index_name):

    for record in dataframe.to_dict(orient='records'):

```

```

yield {

    "_index": index_name,

    "_source": record

}

while True:

    response = es.search(index=ELASTICSEARCH_INDEX, body={

        "query": {

            "range": {

                "timestamp": {

                    "gte": "now-2m"

                }

            }

        },

        "size": 5000

    })

    documents = [hit['_source'] for hit in response['hits']['hits']]

    if len(documents) > 0:

        print("New logs found")

        df = pd.DataFrame(documents)

        df_predicted = predict_label(df, feature_cols, xgboost_model)

```

```
df_predicted.to_csv('classified_data_test_new.csv', index=False)
```

```
# print the first 5 rows of label and severity
```

```
print("\nPredicted Label and Severity:")
```

```
print(df_predicted[['Predicted_Label', 'Severity']].head())
```

```
# Send data to Elasticsearch using the bulk API
```

```
helpers.bulk(es, generate_elasticsearch_docs(df_predicted, 'reported_logs'))
```

```
print("\nReports has been sent to Elasticsearch")
```

```
# wait 2 minutes before checking for new logs
```

```
time.sleep(120)
```

```
continue
```