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

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
(підпис)
«_ __ »_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра

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

на тему: Паралелізація алгоритму класифікації Random Forest для пришвидшення виявлення кібератак

Виконав: здобувач вищої освіти IV курсу, групи ФБ-82
Козачок Вячеслав Олександрович _____
(підпис)
Керівник к.т.н, доцент кафедри ІБ,
Гальчинський Леонід Юрійович _____
(підпис)
Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

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

Київ - 2022 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)
«___»_____2022 р.

ЗАВДАННЯ

1. Тема роботи: Паралелізація алгоритму класифікації Random Forest для пришвидшення виявлення кібератак.
   Керівник роботи _____
   доцент кафедри інформаційної безпеки Гальчинський Л. Ю.
   (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
   Затверджені наказом по університету від «___» _____ 2022р. №
2. Термін подання здобувачем вищої освіти роботи 10 червня 2022 р.
3. Вихідні дані до роботи: основи машинного навчання, Оцінювання методів машинного навчання для виявлення DoS/DDoS атак в IOT. Леонід Гальчинський. Микола Грайворонський.
4. Зміст роботи: огляд основної літератури за напрямком "засоби машинного навчання для виявлення атак", зокрема DoS та DDoS. Окремий аналіз алгоритму Random Forest, для знаходження варіантів пришвидшення його роботи. Реалізація програмного коду, що пришвидшує виконання алгоритму та підвищує його ефективність.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій, тощо): презентація

6. Дата видачі завдання: 1 листопада 2021

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Визначення напрямку роботи. Отримання завдання	01.11.2021	виконано
2	Загальний огляд теми. Формування мети та завдання роботи	05.11.2021 – 10.01.2022	виконано
3	Визначення основного списку літератури	22.01.2022 – 05.02.2022	виконано
4	Ознайомлення з літературою	05.02.2022 – 20.02.2022	виконано
5	Написання першого та другого розділів з теоретичним підґрунтям	21.02.2022 – 05.04.2022	виконано
6	Розробка програми із пришвидшенням роботи моделі Random Forest	06.04.2022 – 01.05.2022	виконано
7	Проходження переддипломної практики	02.05.2022 – 29.05.2022	виконано
8	Написання третього розділу диплому та висновків	29.05.2022 – 04.06.2022	виконано
9	Створення презентації до передзахисту	05.06.2022 – 11.06.2022	виконано
10	Передзахист дипломної роботи	13.06.2022	виконано
11	Доопрацювання дипломної роботи	14.06.2022 – 19.06.2022	виконано
12	Захист дипломної роботи	20.06.2022	виконано

Здобувач вищої освіти _____ Вячеслав КОЗАЧОК

(підпис) (Власне ім’я, ПРІЗВИЩЕ)

Керівник роботи _____ Леонід ГАЛЬЧИНСЬКИЙ

(підпис) (Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Дипломна робота має обсяг 60 сторінок, містить 18 рисунків, 5 таблиць, 2 додатки та 13 джерел посилань.

З кожним днем кількість даних у мережі інтернет зростає в геометричній прогресії. Зростає кількість людей та час, що ці люди проводять на сторінках, де можна знайти все що завгодно. Інтернет також став основою для багатьох бізнесів, які навіть не є можливими без під'єднання мережі Інтернет.

Проте виникає проблема, що завжди знайдуться особи, які прагнуть заволодіти інформацією, що їм не належить, або обмежити доступність деякого сервісу, та отримати від цього певну користь у вигляді грошей або інші блага. Одна з найпоширеніших атак на сьогоднішній день є атака DDoS, що може змушувати цілу систему вийти з ладу на деякий, іноді тривалий час.

Найбільш вразливими до DDoS атак є IoT(Internet of Things) сегмент, де при виявлені атаки важлива кожна секунда.

Об'єктом дослідження є боротьба з DDoS атаками.

Метою дослідження є створення методики покращення виявлення DDoS атак.

Ключові слова: АЛГОРИТМИ КЛАСИФІКАЦІЇ, DDOS, RANDOM FOREST, МАШИННЕ НАВЧАННЯ, АНАЛІЗ АЛГОРИТМІВ КЛАСИФІКАЦІЇ.

ABSTRACT

The work consists of 60 pages, has 18 illustrations, 5 tables, 2 appendices and 13 references.

Every day the amount of data on the Internet grows exponentially. The number of people and the time that these people spend on pages where you can find anything. The Internet has also become the basis for many businesses that are not even possible without an Internet connection.

However, there is a problem that there will always be people who want to take information that does not belong to them, or limit the availability of a service, and get some benefit from it in the form of money or other benefits. One of the most common attacks today is a DDoS attack, which can cause the entire system to fail for a while, sometimes for a long time.

The most vulnerable to DDoS attacks is the IoT (Internet of Things) segment, where every second counts when attacks are detected.

The object of research is the fight against DDoS attacks.

The aim of the study is to create a methodology to improve the detection of DDoS attacks.

Keywords: CLASSIFICATION ALGORITHMS, DDOS, RANDOM FOREST, MACHINE LEARNING, ANALYSIS OF CLASSIFICATION ALGORITHMS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Decision Tree – Дерево ухвалення рішень.

DoS – Denial of Service. Відмова в обслуговуванні.

DDoS – Distributed Denial of Service. Розповсюджена відмова в обслуговуванні.

IoT – Internet of Things. Інтернет речей.

ВСТУП

Актуальність роботи. В сучасному світі існує величезна кількість загроз для вашого комп’ютера або навіть цілої мережі, що надходять з інтернету: віруси, трояни, фішингові листи, відмова в обслуговуванні (DoS/DDoS) і тому подібне. Кожного дня проектуються та розроблюються нові системи запобігання кібератакам, адже вектори атак завжди доповнюються та видозмінюються. Те, наскільки вдало наші системи протистоять тій чи іншій атаці, в першу чергу залежить від виявлення цієї атаки. Саме тому є важливим пошук нових та удосконалення вже відомих методів протидії атакам.

Об’єкт дослідження – боротьба з DDoS атаками.

Предметом дослідження є паралелізація класифікації Random Forest

Метою роботи є створення методики, що зробить виявлення DDoS атак більш ефективним. Це дозволить підвищити рівень інформаційної безпеки.

Методи дослідження: опрацювання літератури по темі дипломної роботи, створення умов, що вказуватимуть на достовірність і точність отриманих даних, проведення аналізу отриманих даних. Аналіз алгоритмів машинного навчання.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ DDOS

1.1 Проблематика

На сьогоднішній день багато галузей мають потребу у використанні алгоритмів, що дозволяють класифікувати дані або будувати прогнози подальшої зміни певних значень. Безпосередньо в сфері кібербезпеки так алгоритми використовуються у переважній більшості для виявлення атак. Тобто класифікації деяких вхідних даних такими, що означають атаку певного роду. Найбільш незахищеними у даній ситуації є кінцеві точки, що відносяться до інтернету речей.

Проблема забезпечення кібербезпеки для таких пристроїв, які відносяться до IoT, полягає у особливостях цієї галузі. Наприклад використання технологій, які вже є застарілими. Такі технології зазвичай мають проблеми з конфіденційністю та загалом з безпекою, бо на таких пристроях оновлення програмного забезпечення(ПЗ) відбувається досить рідко, або взагалі не відбувається.

1.2 Класифікація мережевих атак

Мережеві атаки можна віднести до одної з нижче наведених груп:

1. Активні
2. Пасивні

Активні атаки – такі атаки змінюють систему та її працездатність безпосередньо. Наприклад вірус, що змінює інформацію збережену на комп’ютері, або атаки на веб додатки, що змінюють вміст сторінки. Також DoS атаки підпадають під цю категорію, адже через таку атаку можна порушити доступність цільової системи. Відмінністю активних атак є залишення у системі слідів виконання шкідливого програмного забезпечення(ШПЗ)

Пасивні атаки – атаки, що націлені на збір інформації або про систему, або збір даних що знаходиться на цій системі. До цієї категорії підпадає таке

ШПЗ як прослуховувачі (sniffers). В більшості випадків такі атаки так і залишаються непоміченими через велику складність їх виявлення. Пасивні атаки майже ніколи не залишають слідів.

Також доречним буде згадати цілі, що переслідує атакуючий проводячи ту чи іншу атаку:

- Атаки з метою отримання доступу до даних
- Атаки з метою відмови в обслуговуванні цільового ресурсу
- Атаки з метою отримати дані про цільову систему(розвідка).

Розвідувальні атаки є першим кроком у підготовці до атаки на будь-яку мережу. Вони використовуються зловмисниками для збору інформації, яка може надати дані про можливі вразливості системи та необхідні інструменти їх функціонування.

Атаки відмови в обслуговуванні (DoS) вважаються найпоширенішим типом атаки. Вони відрізняються від інших типів атак, оскільки їх мета — не отримати доступ до мережі чи будь-якої інформації. Їх мета – вимкнути систему або обмежити її функціонування, створюючи умови, які не дозволяють сумлінним користувачам отримати доступ до наданих ресурсів або послуг. Популярність DoS-атак виправдовується їх простотою реалізації. Якщо цей тип атаки здійснюється з використанням великої кількості пристроїв, можна говорити про розподілену атаку відмови в обслуговуванні.

1.3 Типи DDoS атак

1.3.1 Атаки прикладного рівня.

Це одна з найпоширеніших форм відмови в обслуговувані додатку на сьогоднішній день заснована на безкінечній відправці http(s)-повідомлень GET на порт 80(443) для завантаження веб-сервера, що робить його нездатним обробляти інші запити. Часто об’єктом флуду є не корінь веб-додатку, а одна з його частин, що виконує ресурсомістке завдання або та, що використовує

базу даних. Надзвичайно швидке зростання кількості логів веб-серверів означало б, що атака почалася.

Контрзаходи проти HTTP-flood включають налаштування веб-серверів і баз даних для зменшення впливу атак, а також перевірку на наявність DoS-ботів за допомогою різних методів. По-перше, потрібно збільшити максимальну кількість підключень до бази даних за раз. По-друге, встановлення легкого та ефективного nginx перед головним веб-сервером значно підвищило б відмовостійкість, адже він буде гешувати(hashing) запити та видавати статичні дані, коли головний веб сервер витрачав свої ресурси на більш важливі задачі.

Також ресурсоємні частини сайту можуть та мають бути захищені перевіркою так званої «людяності» користувача, наприклад за допомогою використання CAPTCHA.

Детектування такого роду атаки є непростим завданням, адже відрізнити легітимні HTTP запити від тих, які причетні до DDoS атаки, нетривіальна задача. Ця проблема вирішується при використанні машинного навчання, адже між трафіком зловмисника та трафіком легітимного користувача є відмінності, але не завжди помітні людиною через втому, неуважність, недосвідченість і т. п.

1.3.2 Атаки мережевого рівня.

Дуже поширений спосіб для DDoS за допомогою SYN пакетів, що надсилаються на певну машину. Мета цієї атаки це забити канал зв’язку та перевести мережевий стек операційної системи в стан, коли він більше не прийматиме нові запити на з’єднання. Заснована на спробі ініціювати велику кількість одночасних TCP-з'єднань шляхом відправки пакета SYN з неіснуючої адресою повернення.

Більшість операційних систем ставлять у чергу таке з’єднання де після кількох спроб відповісти на пакет ACK не прийшла відповідь. Тільки після n-ї спроби отримати відповідь на пакет ACK операційна система закриває з'єднання. Через великий трафік пакетів ACK, що намагаються закінчити

встановлення з’єднання, черга швидко заповнюється, і ядро відмовляється намагатися відкрити нове з’єднання. Найрозумніші DoS-боти також аналізують систему перед атакою, надсилаючи лише запити на вже відкриті порти.

1.3.3 Атаки вичерпання полоси пропускання.

UDP flood полягає в надсиланні до комп’ютера жертви великої кількості UDP пакетів, що призводить до захаращення смуги пропускання.

ICMP flood простий та ефективний метод забивання смуги через легкість реалізації.

DNS підсилення або DNS відбиття – метод що дозволяє за допомогою одного атакуючого комп’ютера та багатьох DNS серверів третіх сторін, генерувати великий об’єм трафіку. Така атака має в собі особливість невеликої кількості пропускної здатності атакуючого, заповнити набагато більшу полосу пропускання жертви. Це можливо за рахунок генерації такого запиту до DNS, відповідь до якого буде набагато більша у розмірі. Але відповідь отримає не атакуючий, а жертва, через підміну IP адреси у запиті.

Отже одна з цілей (D)DoS атаки полягає в заповненні полоси пропускання трафіку жертви. Середній та малий бізнес зазвичай використовують сервери з пропускною здатністю 1Gbps або 10 Gbps, що в свою чергу коштує в середньому 50 та 300 доларів США за місяць відповідно. Виснаження полоси пропускання зробить його непридатним до обслуговування легітимних запитів.

Іншою ціллю може бути вичерпання всіх доступних ресурсів, таких як RAM(Random Access Memory), використання всієї потужності ЦП, тощо. Наслідками такої атаки можуть бути або недоступність сервера, або настання такого стану, який є потенційно небезпечним та може відкривати нові вразливості, що притаманні саме цьому стану системи та можуть призвести до компрометації сервера або даних.

Існує достатня кількість статичних і динамічних методів протидії таким атакам:

- Створити план протистояння DDoS атакам
- Забезпечити високий рівень мережевої безпеки – використовувати брандмауери, антивіруси, засоби контролю вхідного та вихідного трафіку, сегментація мережі
- Створення надлишковості
- Використання CDN
- Постійний моніторинг за системою.

Як ми бачимо є достатня кількість засобів захисту, які можна використовувати, щоб не стати жертвою DDoS атаки, або щоб вдало їй протистояти. Однак навіть в такому випадку після початку атаки до її виявлення може пройти певний час. Все залежить від того, як налаштований процес моніторингу та реакції на інциденти. Зловмисники намагаються збільшити час для виявлення атаки різними методами, тому навіть два три інженери можуть бути не в змозі надати оцінку показникам, що вони спостерігають на екранах. Тож гостро стоїть проблема саме детектування DDoS атаки вчасно, щоб максимально зменшити вплив на інфраструктуру.

Звісно DDoS можна зафіксувати дивлячись на графік, наприклад такий як видно на рисунку.

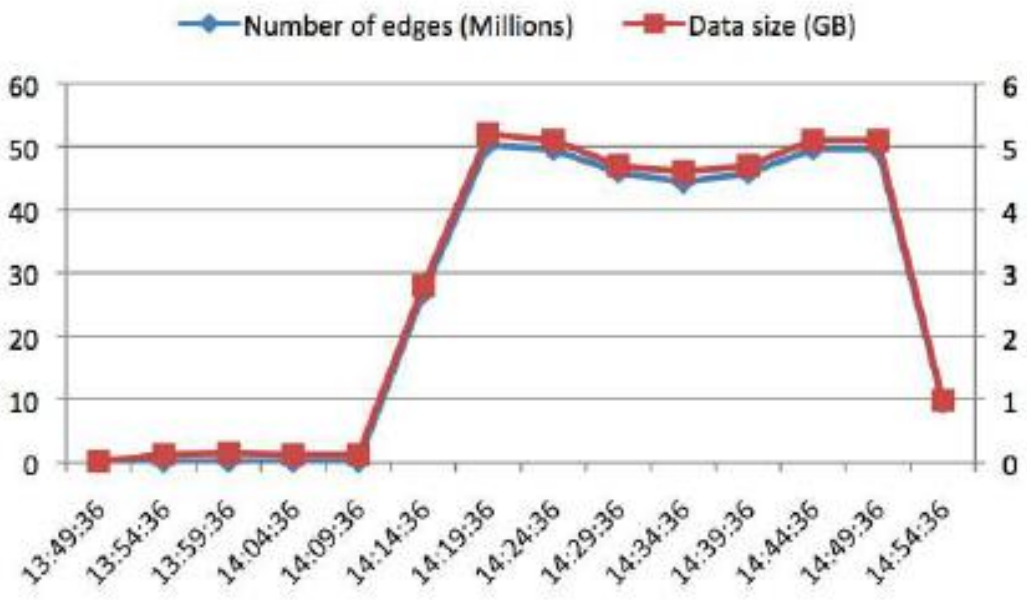

Рисунок 1.1 – Приклад графіку DDoS атаки.

Проте крок дискретизації тут у 5 хвилин, це означає що таку атаку можуть розпізнати лише в проміжку від 1 до 5 хвилин, коли трафік збільшився в 30-50 разів. Тому використання систем, що

У випадках, коли даних дуже багато і треба швидко дати їм оцінку, найбільш вдало справляються комп’ютери, адже вони вміють за короткий час проаналізувати надані дані. Проте аби-який метод, особливо статистичний, не дасть нам коректної оцінки.

Статистичний метод, що створений для детектування атак може бути корисним і надавати правильну інформацію, але при зміні природи атаки, чи певних її складових такий метод нічого не помітить, а переписати його на льоту достатньо нетривіальна задача. Також слід враховувати людський фактор, що допускає помилку і тому подібне. Цю проблему статистичного виявлення та його обмеження гарно висвітлено в статті [13]

Проблему з детектуванням різних видів атак причому з класифікацією до потоків пакетів може вирішити машинне навчання. Цей метод гнучкий, може використовуватись для багатьох сфер діяльності, в тому числі і у виявлені атак.

ВИСНОВКИ ДО РОЗДІЛУ 1

Отже DDoS становить загрозу, яка може спричинити серйозні збитки через виведення з ладу або пошкодження роботи деяких серверів.

Існує певний перелік атак, що набули популярності та регулярно використовуються по всьому світу. Саме ці атаки варто розглядати в першу чергу. Яскравими прикладами є такі атаки як HTTP-flood, SYN-flood, DNS-reflection.

Також було встановлено, що статистичні методи розпізнавання атак не є панацеєю і не мають можливості настільки точно ідентифікувати атаку або приналежність деяких даних до атаки, як методи машинного навчання.

2 ТЕОРЕТИЧНІ ОСНОВИ МАШИННОГО НАВЧАННЯ

2.1 Машинне навчання

Машинне навчання – це один з видів штучного інтелекту який дозволяє навчити програмне забезпечення класифікувати чи передбачувати деякі дані на основі інших даних, тобто дійти логічного висновку, як це робить людина. Змусити комп'ютер «думати» як людина і вирішувати проблеми так, як це робимо ми, є однією з основ машинного навчання.

Машинне навчання ділиться на такі типи

- Машинне навчання з вчителем(Supervised ML) – навчання вже на класифікованих даних, коли алгоритм може чітко поєднати вихідні дані із результатом. Коли модель навчилась на вже готових даних, вона може класифікувати нові дані, що ще не були класифіковані. Це найпростіший вид машинного навчання.
- Машинне навчання без вчителя(Unsupervised ML) – основна відмінність цього підходу, що він може працювати і без заздалегідь класифікованих даних, що робить можливим працювати із значно більшою кількістю даних.
- Посилене машинне навчання (Reinforced ML) – це навчання на особистих помилках. Тобто алгоритм виконує деяку послідовність кроків до того моменту поки не досягне цілі, або не провалить поставлену задачу. За цим слідує або «винагорода», або «покарання». Виходячи з цього алгоритм розуміє як потрібно далі діяти.

Для вирішення завдань цієї дипломної роботи варіант використання машинного навчання з учителем дуже гарно підходить

Припустимо ми вже маємо деяку модель, що може класифікувати дані, тоді формальна постановка задач виглядає наступним чином:

Маємо деяку матрицю даних, нехай X. Цим даним відповідає інша матриця Y. Між результатами матриці X та Y є залежність, яку можно дослідити.

$$X = \begin{pmatrix} x_{11} & \dots & x_{1n} \\ \dots & \dots & \dots \\ x_{m1} & \dots & x_{mn} \end{pmatrix};\ Y = \begin{pmatrix} y_{11} \\ \dots \\ y_{m1} \end{pmatrix}$$

Якщо є залежність, що може перетворити одну матрицю на іншу. Нехай такою функцією перетворення X на Y буде функція f. Тепер введемо матриці X1 та Y1 такі, що містять дані, за якими модель буде вчитися. Матриці X2 та Y2 це матриці за якими перевірятиметься наскільки вдало модель може класифікувати дані.

$$X_1 = \begin{pmatrix} x_{11} & \dots & x_{n1} \\ \dots & \dots & \dots \\ x_{1m} & \dots & x_{nm} \end{pmatrix};\ Y_1 = \begin{pmatrix} y_{11} \\ \dots \\ y_{1m} \end{pmatrix}$$

$$X_2 = \begin{pmatrix} x'_{11} & \dots & x'_{n1} \\ \dots & \dots & \dots \\ x'_{1m} & \dots & x'_{nm} \end{pmatrix};\ Y_2 = \begin{pmatrix} y'_{11} \\ \dots \\ y'_{1m} \end{pmatrix}$$

Задача машинного навчання знайти таку f для якої справджується наступне твердження: $f(X_1) = Y_1$. Показником наскільки вдало модель працює на інших даних є коефіцієнт відхилення Y' від Y_2, де $Y' = f(X_2)$. При чому тестові дані, тобто X_2 по більшій мірі не мають співпадати із X_1. Нехай коефіцієнт точності(accuracy) буде $a \in [0; 1]$. Тоді коефіцієнт відхилення розраховується наступним чином:

$$a = \frac{\text{Кількість правильно класифікованих даних}}{\text{Загальна кількість даних}}$$

Відповідно чим менше коефіцієнт точності, тим більше помилок робить модель при класифікації даних, а чим це число більше, то тим краще.

Це один з найпростіших варіантів оцінки моделі, але існують також і інші методи оцінки у науковій спільноті. Наприклад таке поняття як f-міра, точність, влучність та повнота.

$$Recall = \text{Повнота} = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$Precision = \text{Влучність} = \frac{True\ Positives}{True\ Positives + False\ Positieves}$$

$$f_{measure} = f_{\text{міра}} = 2\frac{\text{Влучність} * \text{Повнота}}{\text{Влучність} + \text{Повнота}}$$

True Positives (істинно позитивне) – кількість даних, що позначені як позитивні, і справді є позитивними.

True Negatives (істинно негативне) – кількість даних, що позначені як негативні, і справді є негативними.

False Positives (хибно позитивне) – кількість даних, що позначені як позитивні, але насправді негативні.

False Negatives (хибно негативне) – кількість даних, що позначені як негативні, але насправді є позитивними.

Повнота – надає інформацію про кількість знайдених позитивних результатів з усієї кількості позитивних результатів. Влучність в свою чергу показує скільки даних було класифіковано як позитивні і при цьому вони і справді є позитивними. f-міра в свою чергу поєднує два поняття влучність і точність, що надає змогу оцінити модель по одному показнику.

Існує достатня кількість алгоритмів машинного навчання, які можуть виконувати поставлену задачу методами. У дослідженні Гальчинського та Грайворонського [11] проведена велика робота, та досліджено наступні алгоритми машинного навчання:

- Метод k-найближчих сусідів (KNN);
- Наївний Баєсів класифікатор (NB);
- Випадковий ліс (Random Forest);
- Логістична регресія (Logistic Regression);
- Дерево ухвалення рішень (Decision Tree).

Нижче представлена таблиця 1 порівнянь різних алгоритмів за точністю, повнотою, влучністю на різних даних.

Таблиця 1.1 – Оцінка алгоритмів на різних даних

Критерій		KNN	Random Forest	Logistic Regression	NB	Decision Tree
час на 100 прогнозів		73,1	2,87	1,5	1,7	1,45
DoS UDP	f-міра	0,99	**1**	0,04	0,5	0,94
	повнота	0,99	1	0,66	0,96	0,98
	влучність	0,98	1	0,02	0,34	0,9
DDoS UDP	f-міра	**0,986**	0,983	0,99	0,817	0,967
	повнота	0,99	0,988	1	0,77	0,975
	влучність	0,982	0,979	0,99	0,87	0,96
DDoS TCP	f-міра	0,99	**1**	0,96	0,65	0,94
	повнота	0,98	1	0,93	0,53	0,9
	влучність	0,99	1	0,99	0,85	0,91
DoS TCP	f-міра	0,97	0,983	**0,99**	0,92	0,99
	повнота	0,98	0,988	0,99	0,84	0,99
	влучність	0,972	0,979	1	0,99	0,98
DoS HTTP	f-міра	0,98	0,994	0,985	**0,999**	0,962
	повнота	0,98	0,991	0,99	0,999	0,953
	влучність	0,98	0,998	0,98	0,999	0,971
DDoS HTTP	f-міра	0,964	**1**	0,545	0,364	0,726
	повнота	0,99	1	0,67	0,27	0,57
	влучність	0,94	1	0,46	0,56	0,998
Normal	f-міра	0,979	**0,989**	0,758	0,618	0,853
	повнота	0,96	0,99	0,702	0,46	0,901
	влучність	0,998	0,999	0,823	0,95	0,81

2.2 Дерева ухвалення рішень

Одним з методів машинного навчання є використання дерев ухвалення рішень(Decision Trees). Їх варто розглянути, так як з них складається ліс дерев (Forest). Алгоритм найкраще розглянути на прикладі. Припустимо ми маємо деякий ряд кущів з різними ознаками. Наприклад маємо 7 кущів з ягодами. Базуючись на цих ознаках ми можемо створити питання, на які можна чітко відповісти «так» чи «ні». Зі схеми нижче можна зрозуміти, що послідовність

дій, що виконує алгоритм, люди виконують кожен день, обираючи між столовими приборами ніж або виделку.

Уявіть, що нам надали набір даних, що складається з кущів різних видів. Як і належить видам в них є вимірювані відмінності, що можна порівняти між собою і за ними класифікувати. Алгоритму, як і звичайній людині, не обов'язково знати що перед ним знаходиться, щоб розбити деякі об'єкти, в даному випадку кущі, по групах.

Одна з перших ознак, на яку дивиться людина в першу чергу – колір. В нашому наборі присутні червоні та сині ягоди, тому одне з можливих питань це: «Кущі мають червоні ягоди?». Така постановка запитання дозволяє нам розбити кущі по двом очевидним категоріям.

Роздивимось категорію праворуч. Всі кущі що залишились там, повністю ідентичні, тому на цьому можна завершити.

Категорія ліворуч навпроти має ще одну ключову відмінність – розмір. Тобто поглянувши на дані кущі і точно вимірявши діаметри ягід взяти між ними середнє і враховувати це як норму, за якою проводити подальшу класифікацію. Зазвичай люди навіть не задумуються, що для них є нормою, але несвідомо обирають середнє арифметичне серед наданих варіантів. Наразі задача розділення кущів з червоними ягодами є доволі простою, адже наявні тільки два розміри. Кущі з більшими ягодами збираємо в одну групу, а кущі з меншими ягодами – в другу.

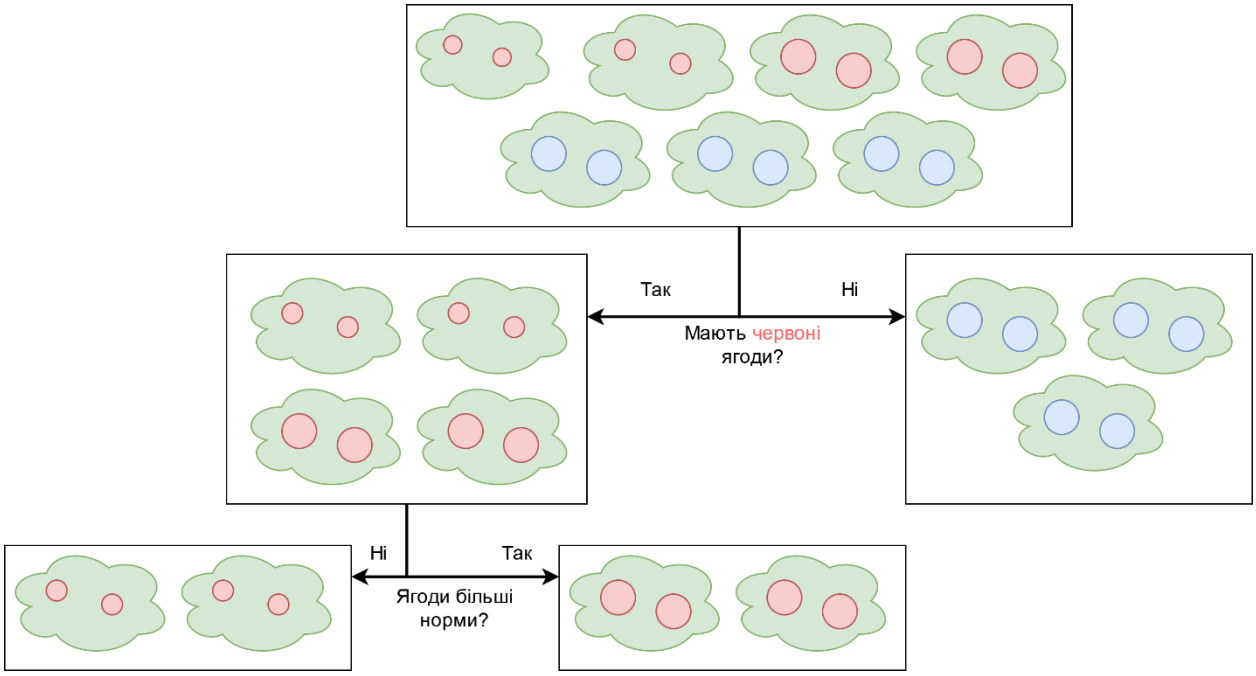

Рисунок 2.1 - Схема дерева прийняття рішень

2.3 Основи Random Forest. Як працює алгоритм.

Основою алгоритму Random Forest є множина дерев, що виконують алгоритм описаний вище, але з певними особливостями.

Як випливає з назви, випадкові ліси складаються з великої кількості незалежних дерев рішень, які діють як єдине ціле. Кожне дерево у випадковому лісі дає прогноз класу, а клас з найбільшою кількістю голосів стає прогнозом нашої моделі. Візуалізацію цього підходу представлено на рисунку 2.2.

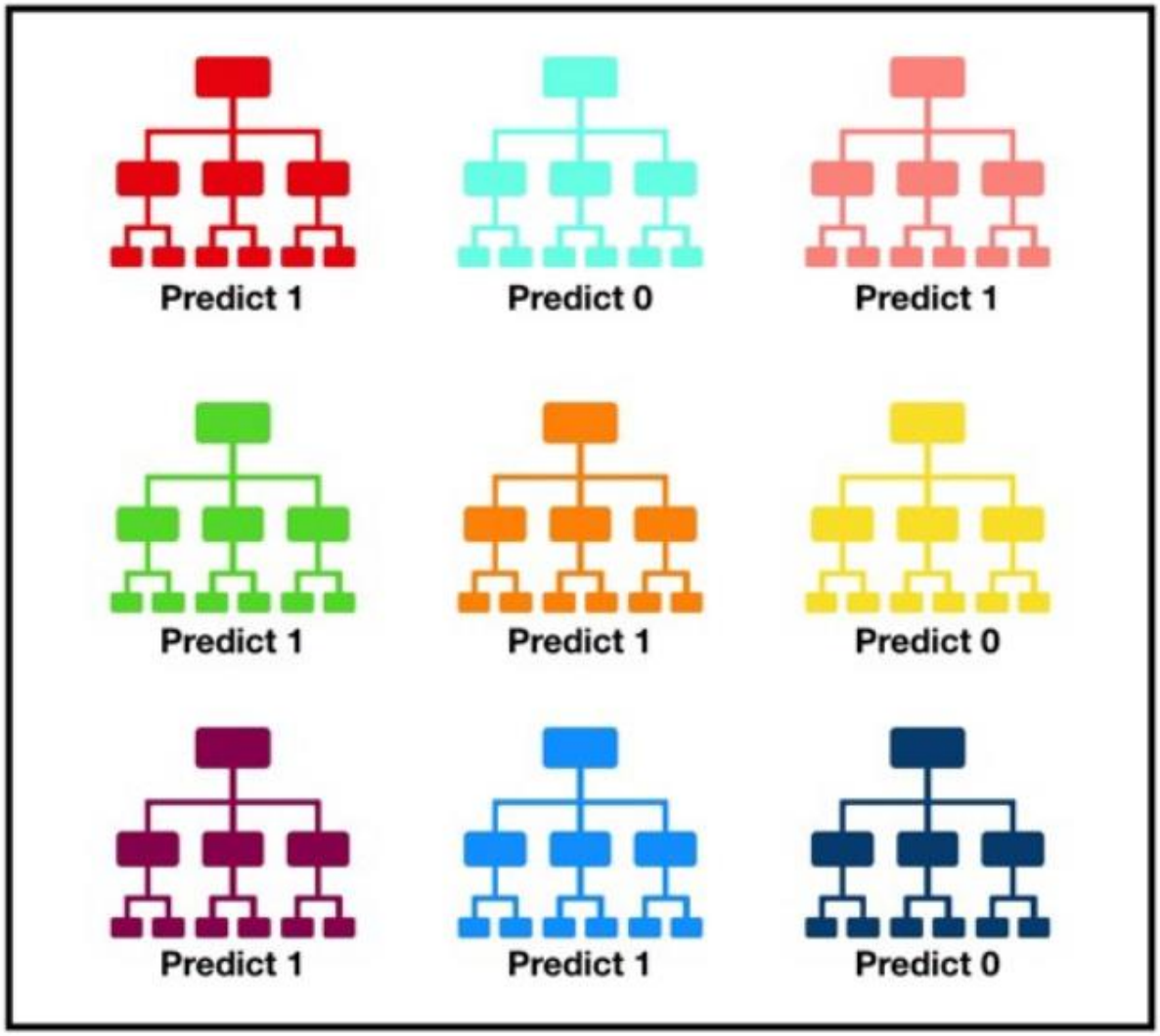

Рисунок 2.2 – Голосування дерев

Основна концепція випадкових лісів проста, але потужна - мудрість натовпу. Згідно з наукою про дані, моделі випадкових лісів настільки хороші з наступної причини: велика кількість відносно не пов’язаних моделей (дерев), що діють як комітет, перевищить ефективність будь-якого окремого компонента моделі.

Низька кореляція між моделями є ключовим фактором. Подібно до того, як низькокорельовані інвестиції, такі як акції та облігації, об’єднуються, щоб утворити портфель, який перевищує суму їх частин, некорельована модель може створити загальний прогноз, більш точний, ніж будь-який індивідуальний прогноз. Причина цього чудового ефекту полягає в тому, що дерева захищають одне одного від помилок один одного (поки вони

продовжують йти неправильно в одному напрямку). Хоча деякі дерева можуть бути “неправильними”, багато інших можуть бути “правильними”, оскільки набір дерев може рухатися в правильному напрямку. Тому передумовами ефективної роботи випадкових лісів є наступна: наші функції повинні мати певний фактичний сигнал про те, що моделі, створені за допомогою цих функцій, ефективніші, ніж випадкові припущення. Кореляція між прогнозами (і, отже, помилками), зробленими окремими деревами, повинна бути низькою.

Тепер ознайомимось з методами, що дозволяють створювати некорельовані дерева ухвалення рішень.

1. Тренування на випадкових частинах одних і тих самих даних.
2. Випадковість критеріїв поділу.

Перший метод дозволяє нам створювати безліч різних дерев, що можуть дуже сильно різнитися між собою, але видавати схожі результати. Як ми вже бачили при огляді дерева ухвалення рішень, вхідні дані дуже впливають на структуру дерева. Проте дані, що подаються до дерева мають бути завжди однієї і тієї ж самої довжини. Наприклад, якщо ми хочемо навчити нашу модель на такому масиві даних [1,2,3,4,5,6], то кожному окремому дереву може надаватися один з наступних масивів даних [1,2,4,4,4,6], [1,2,2,3,4,5], [1,1,3,3,4,6], і так далі. Зверніть увагу, що дані заміняються на ті, які вже наявні у масиві. Заповнення цих даних відбувається випадковим чином

Другий метод дозволяє нам маніпулювати критеріями, які дерево використовує при поділі даних. Тобто кожному окремому дереву надається різний набір критеріїв. Наприклад, якщо наш ліс має містити всього два дерева, і у нас є наприклад 4 критерії, ми можемо їх розподілити таким чином, що першому дереву дістануться критерії з номерами 2 та 3, а другому дереву дістануться критерії розподілу 1 та 4 як показано на рисунку 2.3.

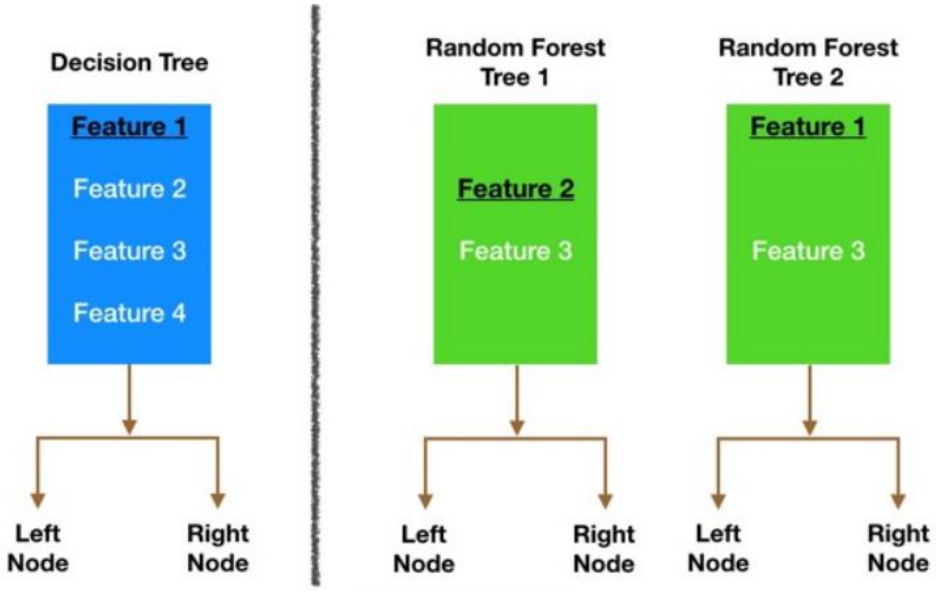

Рисунок 2.3. Вибір критеріїв для окремих дерев

З дерев, що підпадають під опис вище можна створювати ліс, що містить в собі дерева відносно не пов’язані один з одним, де кожне дерево може приймати особисте рішення стосовно вхідних даних.

2.4 Ідея паралельного обчислення. Недоліки та переваги

Паралельне обчислення передбачає спроможність комп’ютера виконувати такий вид обчислень, на що спроможні сьогодні будь-які комп’ютери з хоча б двома ядрами.

Переваги:

- Зменшення часу на обробку даних і навчання моделі.

Недоліки:

- Збільшує кількість ресурсів, що використовуються одночасно.
- Складність узгодження потоків.
- Складність створення багато-потокової програми

Існує два етапи які потрібно виконувати паралельно, а саме побудову моделі, та передбачення даних. Побудова моделі це процес що може займати десятків хвилин, адже даних може надаватися досить багато, тому з більшою кількістю даних збільшується і час.

На рисунку 2.4 зображено діаграму на якій показано, яким чином будується модель. В даному випадку показано 3 потоки, які беруть тренувальні дані, та кожен окремо за одним і тим самим алгоритмом створюють стільки випадкових лісів, скільки потоків було використано. Так як ці окремі ліси створюються за одними і тими ж даними, але з різним випадковим станом, то в результаті дерева, що знаходять в кожному з лісів мають однакову природу, працюють з такими самими стовпцями та типами даних. Отже можна припустити, що поєднавши ці в даному випадку три ліси, ми отримаємо один ліс, де кожне дерево є унікальним та

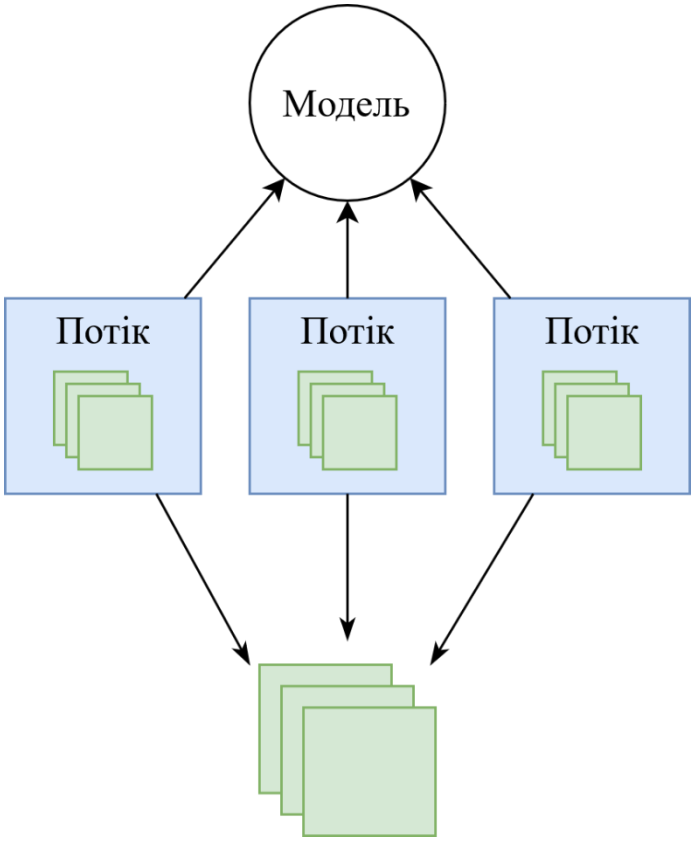

Рисунок 2.4 – Побудова моделі паралельно

Проте якщо побудова моделі не є для нас пріоритетом, адже це зазвичай виконується заздалегідь на вивірених даних. Більший інтерес представляє швидкість класифікації певних даних, так як саме цей процес є важливим для нас для своєчасного виявлення атаки. Реалізація паралельного виконання можлива принаймні двома методами.

Перший метод полягає у використанні однієї і тієї ж моделі для класифікації всіх даних частинами в такому випадку потоки живуть ззовні моделі і використовують її та частини даних(Рис 2.5). Тобто дані поділяються на рівні частини, що передаються у потоки. Результат виконання кожного з цих потоків об’єднується в одне ціле після успішного виконання кожного потоку і подається у вигляді стовпчика відповідей.

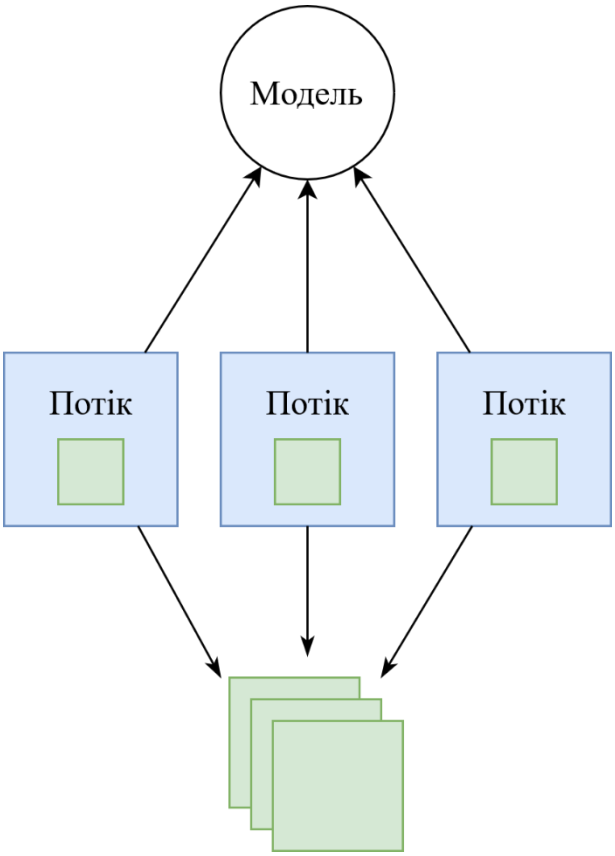

Рисунок 2.5 – Класифікація даних. Спосіб 1

Другий метод передбачає дописування коду у бібліотеку проекту і реалізація богатопотоковсті всередині моделі класифікатору. Можна уявити таку реалізацію чином, що зображений на рисунку 2.6. Тобто для кожного дерева створюється потік і якому воно виконує поставлену задачу одночасно з іншими деревами. Кожне дерево опрацьовує кожний шматок даних. Реалізувати можна таким чином, що кожне з дерев в один і той самий момент опрацьовує один і той самий шматок даних. Коли кожне дерево закінчить класифікацію цієї частини даних, то відповіді звіряються і методом голосування вирішується як в кінцевому варіанті класифікувати.

У такого способу є мінус в тому, що під час виконання купа дерев може чекати, поки одне з них завершить класифікацію, що може значно сповільнити модель.

Іншим варіантом підходу є такий, що кожне дерево створює свій особистий вектор незалежно від інших а після того, як всі дані всіма дерева проаналізовані, відбувається злиття векторів відповідей Y_i , що є результатами кожного окремого потоку.

Такий метод може збільшити використання оперативної пам’яті, проте теоретично має виконуватись швидше за попередньо розглянуту реалізацію.

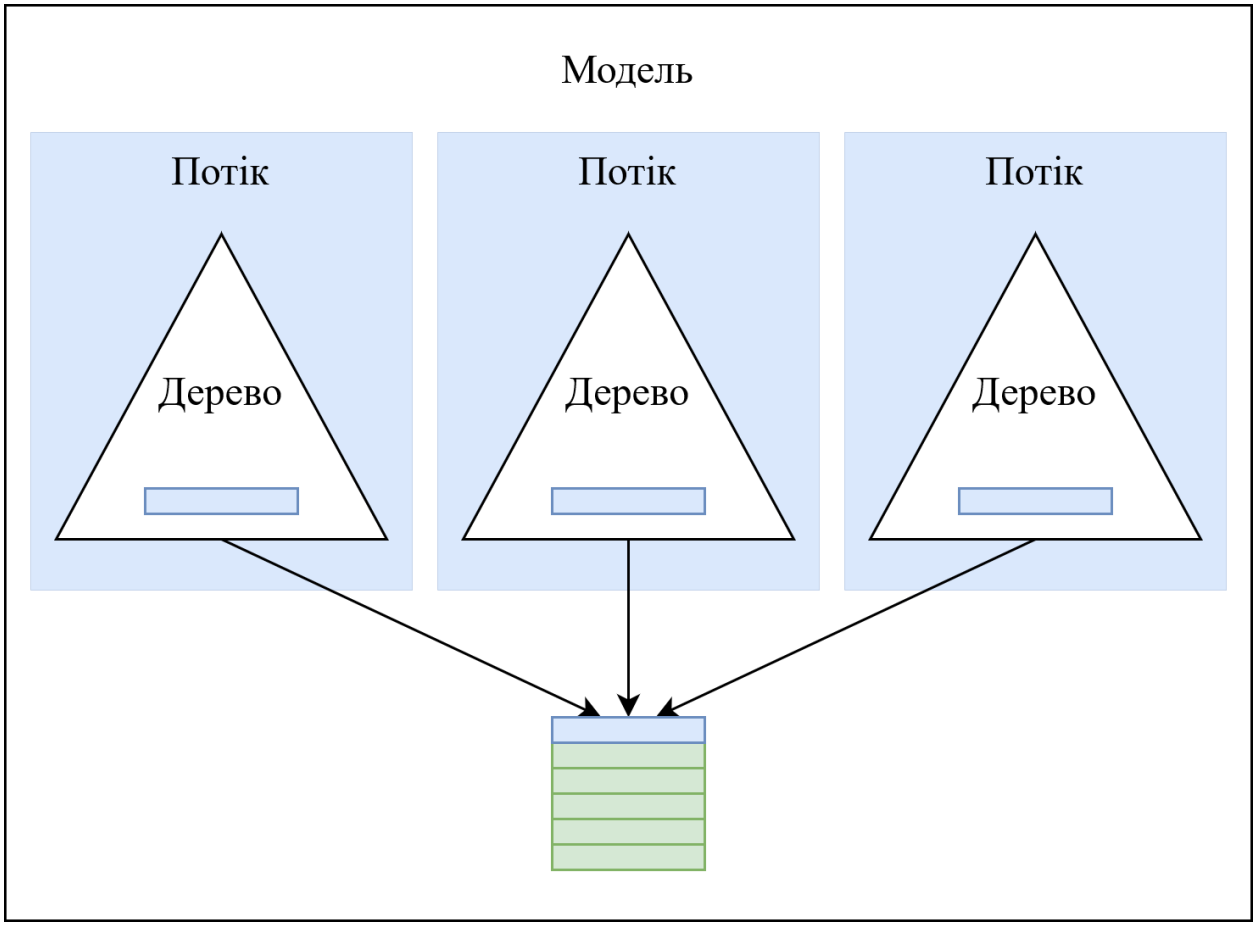

Рисунок 2.6 – Багатопотокова класифікація.

Ну і на останок треба згадати, що не завжди вигідно створювати 100 потоків для 100 дерев із лісу, інколи набагато ефективніше зробити 10 потоків в кожному з яких по 10 дерев(рис 2.7). Кількість потоків, яка буде оптимальною для використання у даній ситуації, ще потрібно дослідити.

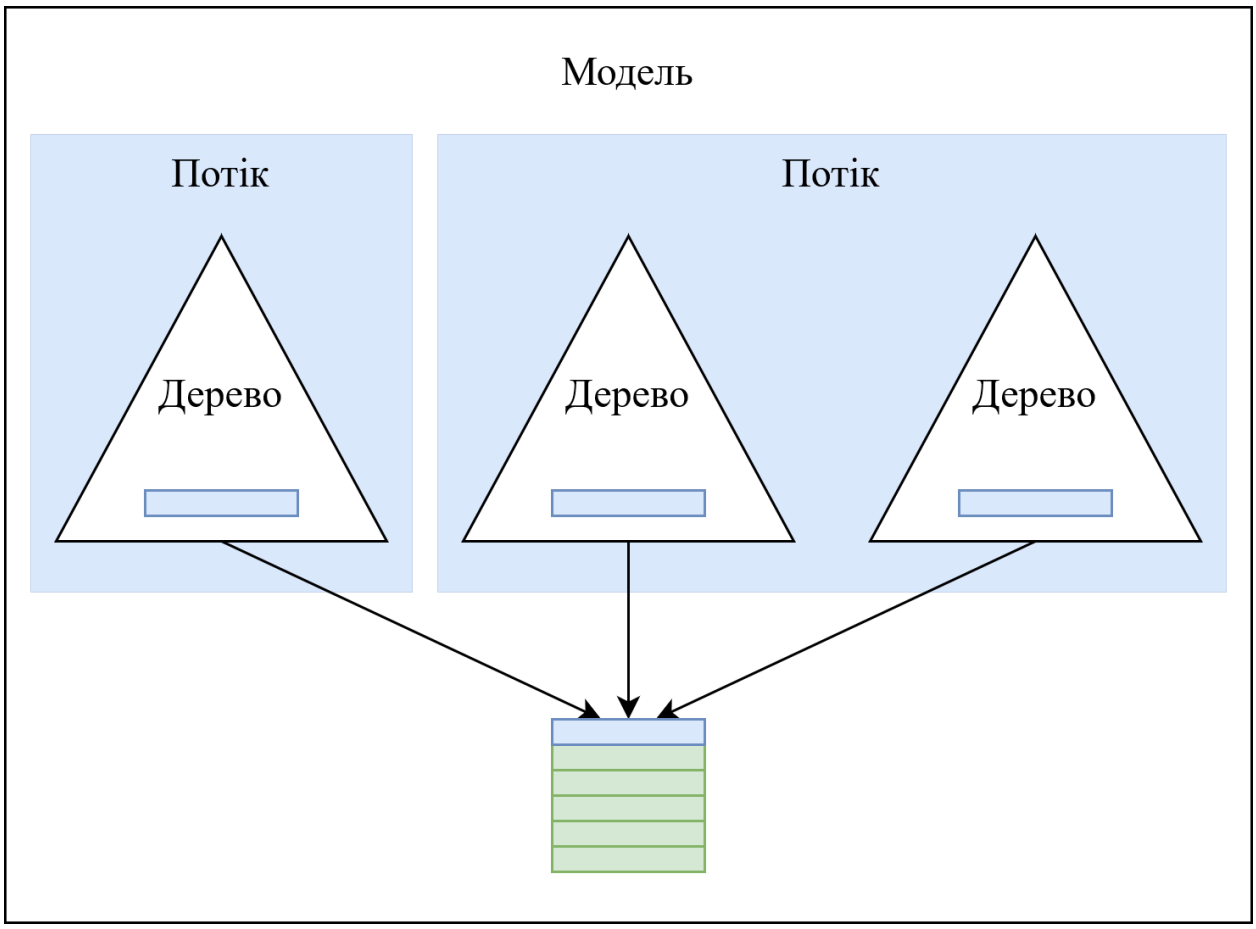

Рисунок 2.7 – Багатопотокова класифікація(2)

2.5 Ідея розподіленого обчислення. Недоліки та переваги

Розподілене обчислення передбачає наявність принаймні декількох різних машин, що працюють над виконанням однієї глобальної задачі. На даний момент реалізація RandomForest у бібліотеці scikit-learn для мови програмування Python передбачає паралельні обчислення, але для надвеликих масивів даних було б чудово мати змогу ще пришвидшити виконання цього алгоритму використовуючи декілька машин.

Цей метод має наступні переваги:

- Більша швидкість обчислення
- Збільшення обсягів обробки даних через додання нових серверів до мережі
- Відсутність єдиної точки відмови.
- Дані реплікують у кластерах, що забезпечує їх надлишковість.

Недоліки розподілених обчислень:

- Підвищена складність підтримки та діагностування помилок.
- Складність розробки.

Система, що працює в режимі реального часу та відстежує зміни в трафіку, що надходять до мережі повинна бути встановлена inline(в лінію), для того, щоб мати можливість автоматично реагувати на пакети, що мають характер атаки, або має бути під’єднана до іншого мережевого пристрою, що буде приймати результати системи класифікації і відповідно реагувати.

Незалежно як система встановлена до неї має надходити трафік у реальному часі, цей трафік має бути розподілений. Існує різні підходи як розподілити трафік по різним робітникам, щоб це було ефективно. Одним з варіантів є використання окремої одиниці мережі, що балансує навантаження між вказаними робітниками за налаштованими критеріями. Критерії можуть бути наступними:

- Видавати блоки інформації для опрацювання кожному робітникові по черзі
- Видавати блоки інформації для опрацювання залежно від кількості оброблених пакетів кожним робітником.
- Видавати блоки інформації для опрацювання за швидкістю оброблення інформації

Кожний з методів має ті чи інші переваги та недоліки. Одні з найкращих рішень по балансуванню трафіку можна знайти у вендора F5.

Нижче на рисунку 2.8 приведено умовну схему, по якій дані потрапляють до робітників. На ній можна виділити основні(ключові) фігури: дані, що надходять до системи із зовні, керуючий комп’ютер, що спілкується з робітниками, та безпосередньо самі робітники.

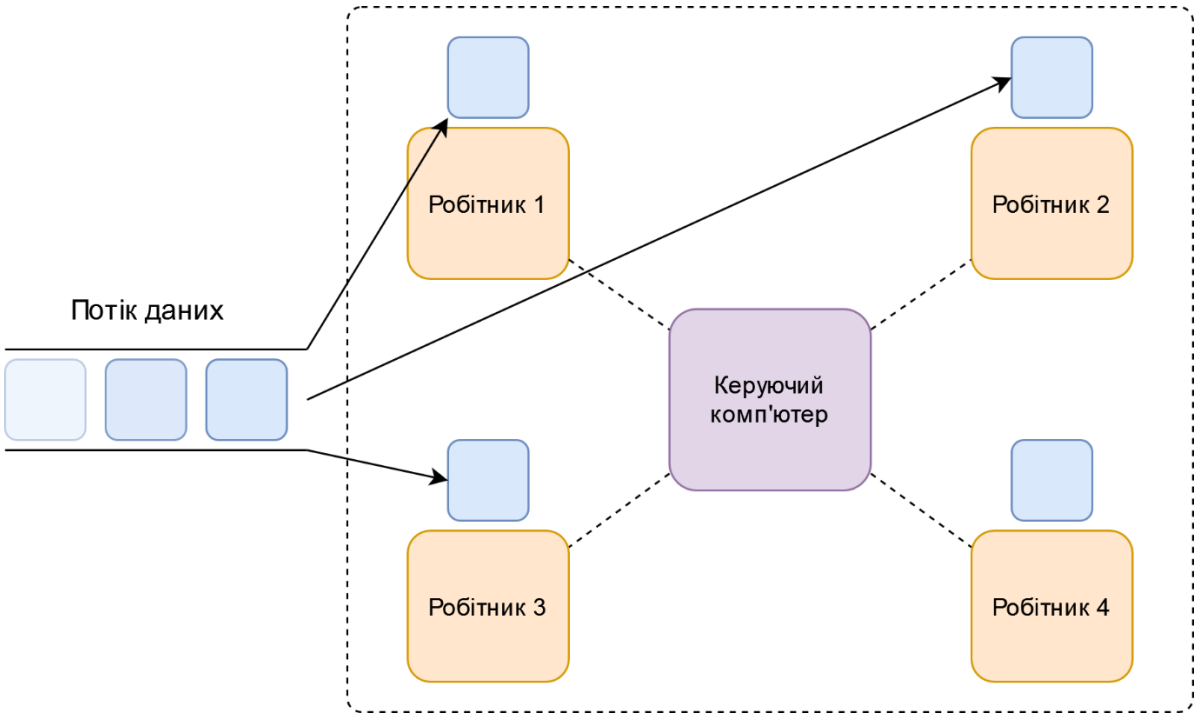

Рисунок 2.8 - Розподілення інформації у розподіленій системі

Даний варіант розглядається як можливий але не буде виконаний в цій дипломній роботі. Однак є можливість розглянути дану реалізацію у науковій дисертації магістерського рівня.

ВИСНОВКИ ДО РОЗДІЛУ 2

Було розкрито поняття машинного навчання та формалізовано за допомогою математичної моделі. Описано таку модель як дерево ухвалення рішень, як спосіб класифікації. Зрозуміли на яких принципах створено модель Random Forest: мудрість більшості та створення не корельованих дерев у лісі.

Створена формальна модель, що описує можливі способи створення декількох потоків, що одночасно будуть модель або класифікують дані. Дані моделі містять в собі варіанти реалізації потоків «всередині» моделі, та «ззовні» неї.

3 РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНИХ ОБЧИЛЕНЬ ДЛЯ АЛГОРИТМУ RANDOM FOREST

Для реалізації поставленої задачі буде використовуватись мова програмування Python версії 3.10.0. Python чудова мова до якої можна встановити безліч бібліотек, що значно розширює можливості розробки. Бібліотека scikit-learn чудово підходить під завдання, що ставить перед собою дана дипломна робота.

3.1 Вхідні дані та їх обробка

Вхідні дані що використовуються у даній дипломній роботі узяті з сайту Канадського інституту з кібербезпеки [4] та використовуються дослідниками та ентузіастами по всьому світу через максимально велику схожість з атаками, які здійснюються на справжні інфраструктури.

Дані у наборах, що можна знайти на їх сайті мають різні представлення від PCAP файлів, до вже готових CSV для аналізу.

Ці дані цікаві тим, що в них є всі необхідні(і навіть більше) полів, за якими можна виявити DDoS атаку. Повний перелік полів, що є у цьому наборі даних можна побачити у додатку[1].

Перша проблема, яка зустрічається на шляху, це обробка даних певним чином, щоб алгоритм міг з ними працювати. Як вже згадувалось, дані взяті з офіційного сайту канадського інституту кібербезпеки [4], а саме дані DDoS Evaluation Dataset (CIC-DDoS2019).

1. Необхідно прибрати поля, що є унікальними для даної конкретної атаки, наприклад IP адреси.
2. Потрібно визначати значущість тих чи інших даних при ідентифікації атаки, щоб залишити тільки необхідні стовпці.
3. Звірити однорідність даних у колонці, тобто знайти і виправити такі колонки, які містять строкові дані, хоча не мали б, дані що містять пусті стрічки, значення NULL, +infinity, -infinity.

4. Якщо у деяких колонках присутні такі дані, що мають рядки, то закодувати їх відповідно. Наприклад якщо ми маємо різні типи атак, то в стовпці замість назв “DNS”, “SYN”, “UDP”, “NORMAL” треба вказати умовні числові позначення у вигляді 1, 2, 3, 4.

Насамперед треба визначитись які дані збиратимуться для порівняльного аналізу моделі, що працює в один потік та в багато потоків.

1. Час, що витрачається на побудову;
2. Час, що витрачається на класифікацію даних;
3. Розміри даних, на яких тренується(будується) модель;
4. Кількість стовпців та рядків на який проводяться тести та будується модель.

Таблиця 3.1 – Список файлів та їх розміри

Назва файлу	Розмір
Portmap.csv	76 769 КБ
Small-Syn.csv	286 751 КБ
Syn.csv	1 833 372 КБ
AppDDos.pcap_Flow.csv	144015 КБ

3.2 Аналіз побудови моделі класифікації для файлів

Створену програму, що вимірює необхідні метрики застосуємо на по черзі на кожний з цих файлів.

Почнемо по порядку. Першим на черзі для аналізу в нас такий файл як Portmap.csv і отримаємо певні статистичні дані. Рис 3.1 та 3.2.

```
(env) C:\University\4course\Diploma\code>python paralel_forest.py 03-11\Portmap.csv
INFO:root:Using 03-11\Portmap.csv
INFO:root:Column 'Unnamed: 0' dropped successfully
INFO:root:Column 'Flow ID' dropped successfully
INFO:root:Column 'Source IP' dropped successfully
INFO:root:Column 'Destination IP' dropped successfully
INFO:root:Column 'Timestamp' dropped successfully
INFO:root:Column 'SimillarHTTP' dropped successfully
WARNING:root:column 'Dst IP' was not deleted.
WARNING:root:column 'Src IP' was not deleted.
INFO:root:Column 'Fwd Header Length.1' dropped successfully
INFO:root:Column 'Inbound' dropped successfully
WARNING:root:'Label' Failed fill 'None' value with median
Training Data:
(153355, 79)
(153355, 1)
Testing Data:
(38339, 79)
(38339, 1)
INFO:root:---Creating Forest Synchronycally---
INFO:root:Forest is ready. Trees:100
INFO:root:Score: 0.9997913351939278
INFO:root:time measurment: 4.679509700003109
INFO:root:---Creating Forest in Paralel---
INFO:root:Forest is ready. Trees:100
INFO:root:Score: 0.9997913351939278
INFO:root:time measurment: 2.687598699998489
INFO:root:---Predicting Forest Synchronycally---
INFO:root:time measurment: 0.16387070000200765
INFO:root:---Predicting Forest in Paralel---
Threads: 2. INFO:root:time measurment: 0.09658160000253702
Threads: 3. INFO:root:time measurment: 0.0863482999993721
Threads: 4. INFO:root:time measurment: 0.09712289999879431
Threads: 5. INFO:root:time measurment: 0.10727729999780422
Threads: 6. INFO:root:time measurment: 0.12418249999973341
Threads: 7. INFO:root:time measurment: 0.149307100000442
Threads: 8. INFO:root:time measurment: 0.163321600000927
Threads: 9. INFO:root:time measurment: 0.1742164999996021
Threads: 10. INFO:root:time measurment: 0.18539559999771882
Threads: 11. INFO:root:time measurment: 0.19602260000101523
Threads: 12. INFO:root:time measurment: 0.20802630000252975
Threads: 13. INFO:root:time measurment: 0.22149889999855077
Threads: 14. INFO:root:time measurment: 0.23020259999975679
Threads: 15. INFO:root:time measurment: 0.2536050000016985
Threads: 16. INFO:root:time measurment: 0.26074979999975767
Threads: 17. INFO:root:time measurment: 0.27403469999990193
Threads: 18. INFO:root:time measurment: 0.28499290000036126
Threads: 19. INFO:root:time measurment: 0.30726030000005267
Threads: 20. INFO:root:time measurment: 0.3100025999992795
```

Рисунок 3.1 – Аналіз Portmap.csv(1)

```
--- Analyze Synchronical ---
Accuracy: 0.99684 (+/- 0.01772)
Precision: 0.99981 (+/- 0.00042)
Recall: 0.99695 (+/- 0.01829)
F-measure: 0.99841 (+/- 0.00926)
col_0    0      1
row_0
0      942      1
1        7  37389
--- Analyze Paralel ---
Accuracy: 0.99684 (+/- 0.01772)
Precision: 0.99981 (+/- 0.00042)
Recall: 0.99695 (+/- 0.01829)
F-measure: 0.99836 (+/- 0.00923)
col_0    0      1
row_0
0      942      1
1        7  37389
INFO:root:Synchronically predicted:
Portmap    37390
BENIGN       949
dtype: int64
INFO:root:Predicted in Paralel:
Portmap    37390
BENIGN       949
dtype: int64
```

Рисунок 3.2 – Аналіз Portmap.csv(2)

Так як ми бачимо вивід програми перший раз, давайте більш детально ознайомимось з ним.

В першу чергу при відкритті будь якого файлу він оброблюється методами, що були описані на початку розділу: видалення непотрібних стовпчиків, заповнення однорідними даними, і т. п. Інформація про успішність або не успішність виконаних дій виводиться на екран. Можна побачити, що деякі стовпчики не були видалені, так як програма розроблена для коректної роботи з даними сайту канадського інституту кібербезпеки за 2019 рік та з більш новими даними. Різниця в них виникає через різні версії програми, що створювала CSV файли з PCAP. Різні версії мають трохи різні назви полів, та ще додатково до того нові поля.

Наступним кроком є розбиття набору даних на частини по 80% та 20%. Більша їх частина вважається як тренувальна, на якій модель навчається класифікувати одні пакети, від інших. Менша частина – контрольні або перевірочні дані, на яких потрібно впевнитись, чи правильно алгоритм відпрацював та яка точність моделі. Відповідно на екран виводиться

інформація, що у наборі тренувальних даних присутні 153355 рядків даних, в кожному з яких є по 79 різних параметрів. У тренувальних даних 38339, 79. Для майбутньої моделі це є X, який треба за допомогою ще невідомої функції відобразити у Y. Математичне підґрунтя було описане у розділі 2.1. За Y ми беремо останню колонку, що позначає колонку відповідей для нашої моделі.

У наступних рядках можна бачити інформацію про те як створюється модель, послідовно чи паралельно та скільки часу на це витрачається. Також вказується скільки часу займає класифікувати тренувальні дані. Якщо у послідовному варіанті все це відбувається в одному потоці, то при паралельних обчисленнях можуть використовуватись безліч потоків, від 2 до «нескінченності». В даному випадку модель передбачує дані для 2, 3, 4 і так до 20 потоків.

Коли дані класифіковані, треба дізнатись як вдало з цим впорались моделі. Для цього використаємо наступні величини: точність, повноту, влучність та f-міру. Одразу біля цих значень розміщається матриця невідповідностей за значеннями якої і вираховуються веще зазначені величини.

Таблиця 3.2 – Матриця невідповідностей

		Actual values	
		Positive	Negative
Predicted values	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

На останок виводяться значення, що були передбачені, в даному випадку це Portmap та BENIGN(звичайний трафік).

Тепер, коли ми знаємо які саме дані виводить програма, можна зібрати дані стосовно всіх загаданих до цього файлів.

Результати виконання програми для файлу Syn.csv на рисунках 3.3 та 3.4

```
Training Data:
(3456433, 79)
(3456433, 1)
Testing Data:
(864108, 79)
(864108, 1)
INFO:root:---Creating Forest Synchronycally---
INFO:root:Forest is ready. Trees:100
INFO:root:Score: 0.9999930564235027
INFO:root:time measurment: 615.3841919999995
INFO:root:---Creating Forest in Paralel---
INFO:root:Forest is ready. Trees:100
INFO:root:Score: 0.9999930564235027
INFO:root:time measurment: 518.966300600001
INFO:root:---Predicting Forest Synchronycally---
INFO:root:time measurment: 4.274096799999825
INFO:root:---Predicting Forest in Paralel---
Threads: 2. INFO:root:time measurment: 2.4233595999976387
Threads: 3. INFO:root:time measurment: 1.8610959999969054
Threads: 4. INFO:root:time measurment: 1.5890063999977428
Threads: 5. INFO:root:time measurment: 1.4659147000020312
Threads: 6. INFO:root:time measurment: 1.4113854999995965
Threads: 7. INFO:root:time measurment: 1.3718101999984356
Threads: 8. INFO:root:time measurment: 1.3732081000016478
Threads: 9. INFO:root:time measurment: 1.3355514999966545
Threads: 10. INFO:root:time measurment: 1.3214842000015778
Threads: 11. INFO:root:time measurment: 1.392355400002998
Threads: 12. INFO:root:time measurment: 1.3983320000006643
Threads: 13. INFO:root:time measurment: 1.3364268000004813
Threads: 14. INFO:root:time measurment: 1.379425099999935
Threads: 15. INFO:root:time measurment: 1.2753653999970993
Threads: 16. INFO:root:time measurment: 1.2824383999977726
Threads: 17. INFO:root:time measurment: 1.3181368000005023
Threads: 18. INFO:root:time measurment: 1.3536008999981277
Threads: 19. INFO:root:time measurment: 1.352648500000214
Threads: 20. INFO:root:time measurment: 1.354097700001148
```

Рисунок 3.3 – Аналіз Syn.csv(1)

```
--- Analyze Synchronical ---
Accuracy: 0.99990 (+/- 0.00053)
Precision: 1.00000 (+/- 0.00001)
Recall: 0.99990 (+/- 0.00053)
F-measure: 0.99995 (+/- 0.00027)
col_0     0       1
row_0
0      7256       3
1         3  856846
--- Analyze Paralel ---
Accuracy: 0.99989 (+/- 0.00053)
Precision: 1.00000 (+/- 0.00002)
Recall: 0.99991 (+/- 0.00047)
F-measure: 0.99995 (+/- 0.00027)
col_0     0       1
row_0
0      7256       3
1         3  856846
INFO:root:Synchronically predicted:
Syn       856849
BENIGN      7259
dtype: int64
INFO:root:Predicted in Paralel:
Syn       856849
BENIGN      7259
dtype: int64
```

Рисунок 3.4 – Аналіз Syn.csv(2)

В ході виконання роботи були отримані дані зазначені в таблиці 3.3, що стосуються саме побудови моделей.

Таблиця 3.3 – Отримані дані побудови.

		Паралельне виконання			
Назва файлу	Кількість рядків	Час побудови	Повнота	Влучність	f-міра
Syn.csv	3456433	380с	0,99991	1,00000	0,99995
Small-Syn.csv	691286	35,4с	0,99952	0,99999	0,99977
Portmap.csv	153355	2,81с	0,99695	0,99981	0,99836
		Послідовне виконання			

Назва файлу	Кількість рядків	Побудова моделі:	Повнота	Влучність	f-міра
Syn.csv	3456433	624с	0,99990	1,00000	0,99995
Small-Syn.csv	691286	82с	0,99950	0,99684	0,99975
Portmap.csv	153355	4,679с	0,99695	0,99981	0,99841

З таблиці 3.3 видно, що побудова моделі в один потік, тобто послідовно, займає більше часу ніж побудова моделі паралельно. Залежність часу побудови до кількості даних відображено на рисунку 3.5 .

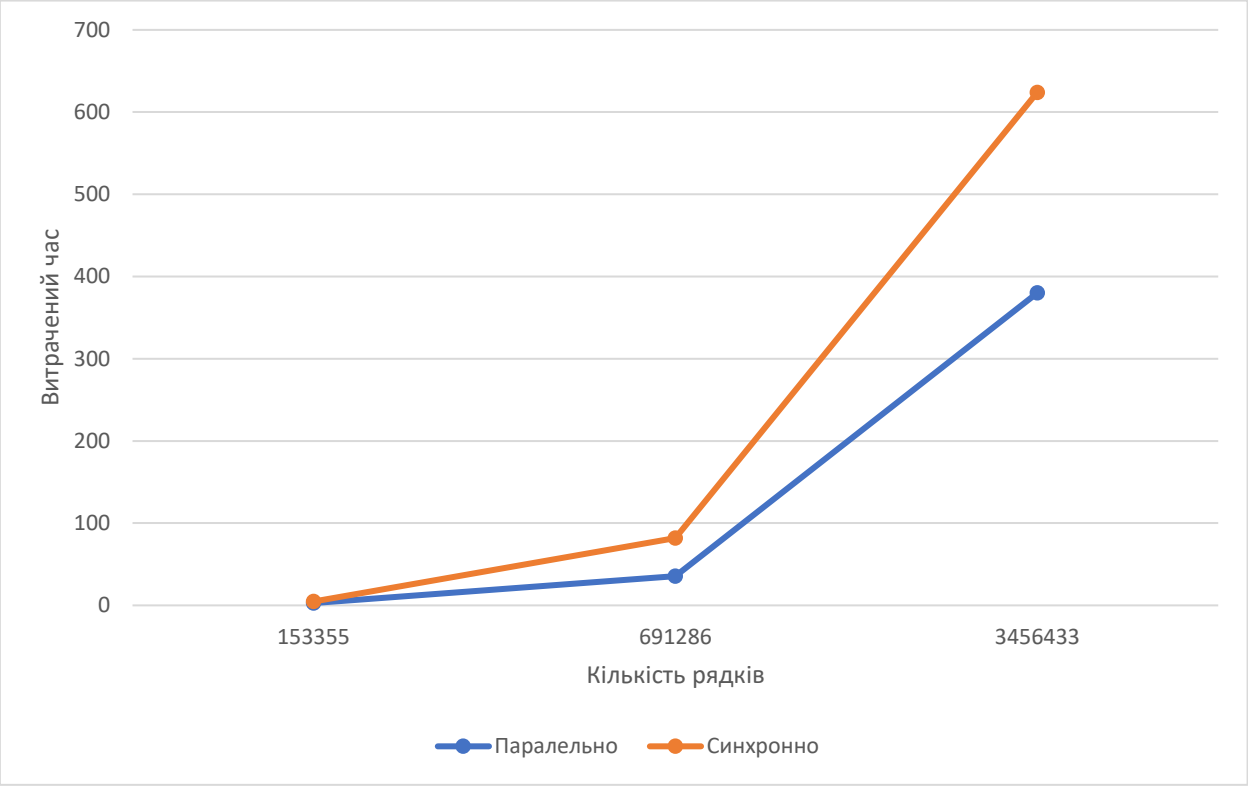

Рисунок 3.5 -- Залежність витраченого часу на побудову моделі до кількості рядків у даних

На рисунку 3.6 можна побачити той самий графік але з логарифмічною шкалою. Можна побачити, що кількість часу витраченого на класифікацію

даних з багатьма потоками, а саме 10 штук, приблизно в середньому у 1.8 рази швидше від алгоритму який робить це в один потік. Тао

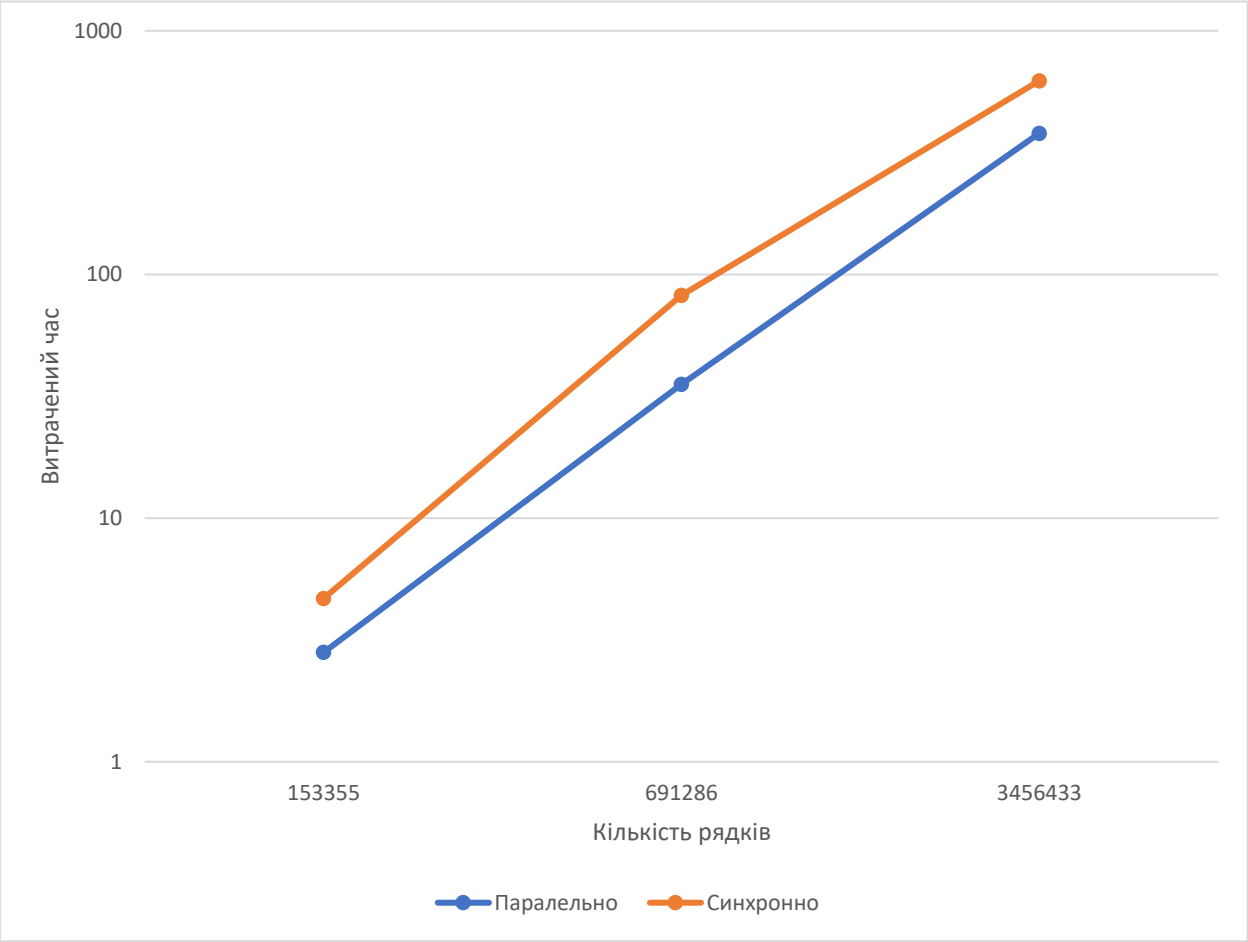

Рисунок 3.6 -- Залежність витраченого часу на побудову моделі до кількості рядків у даних у логарифмічному вигляді.

3.3 Аналіз класифікації даних за допомогою побудованої моделі

Так як задача класифікації даних за допомогою вже побудованої моделі є більш пріоритетною задачею, то цю частину розглянемо більш детально.

Як раніше було представлено рисунках 3.1-4, ми маємо час класифікації даних з різною кількістю потоків. Об’єднаємо цей час у таблиці 3.4. З таблиці видно що час виконання з одним потоком куди менший ніж з декількома, але

і тут є виключення. Для наочності побудовано графік для першого файлу Portmap на рисунку 3.7.

Таблиця 3.4 – Час на класифікацію тренувальних даних різних файлів з різною кількістю потоків

Кількість потоків	Час		
	Portmap	Small-Syn	Syn
1	0,167113	1,0370759	4,274097
2	0,095513	0,6407790	2,423360
3	0,081043	0,4217110	1,861096
4	0,087132	0,3941999	1,589006
5	0,103936	0,3111512	1,465915
6	0,119567	0,3048949	1,411385
7	0,150357	0,3484710	1,371810
8	0,156191	0,3308401	1,373208
9	0,166884	0,3339860	1,335551
10	0,185908	0,3689111	1,321484
11	0,197642	0,3421322	1,392355
12	0,211195	0,3592047	1,398332
13	0,230281	0,3687290	1,336427
14	0,246230	0,3787009	1,379425
15	0,250359	0,4078883	1,275365
16	0,259985	0,4069255	1,282438
17	0,272923	0,4367646	1,318137
18	0,286298	0,4522401	1,353601
19	0,299016	0,4853544	1,352649
20	0,311642	0,7501998	1,354098

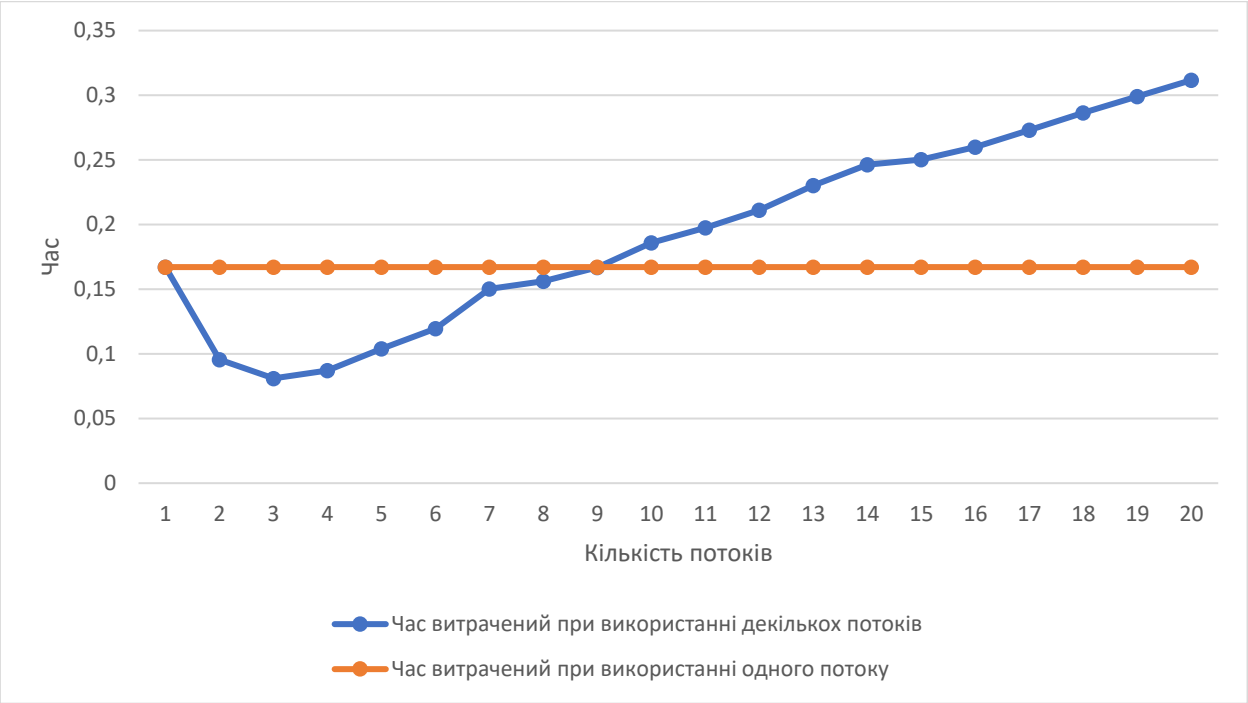

Рисунок 3.7 – Витрати часу на класифікацію до кількості потоків

Portscan.csv

Portmap це найменший файл з нашої вибірки. Яку ж поведінку ми бачимо на графіку? Найбільш швидко класифікація проходить при кількості потоків у 3 одиниці. Десь близько 9 потоків швидкість класифікації компенсується кількістю додаткових операцій, які необхідно зробити з даними після класифікації, тож тоді використання багатопотоковості немає значення, адже час виконання один і той самий. Далі при збільшенні кількості потоків кількість часу на їх класифікації тільки зростатиме. Тобто можна зробити висновок, що для невеликої кількості даних 3-4 потоки є оптимальним вирішенням.

Наступним файлом для тестів ми обрали файл Small-Syn.csv. Результати графічно відображені на рисунку 3.8

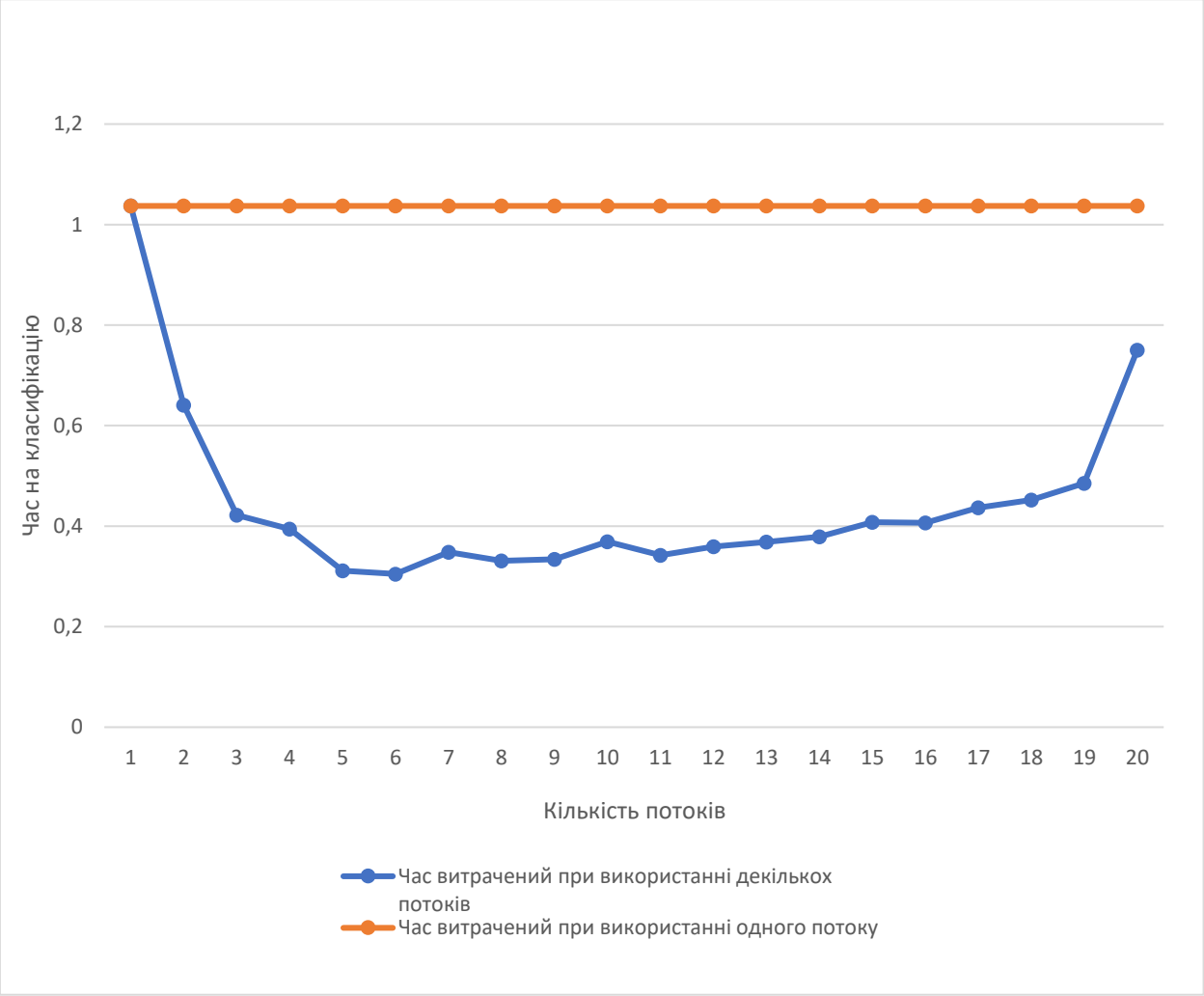

Рисунок 3.8 – Витрати часу на класифікацію до кількості потоків Small-Syn.csv

Як видно з рисунка найменше часу було витрачено при використанні 6 потоків, тобто це є оптимальною кількістю для файлів такого розміру.

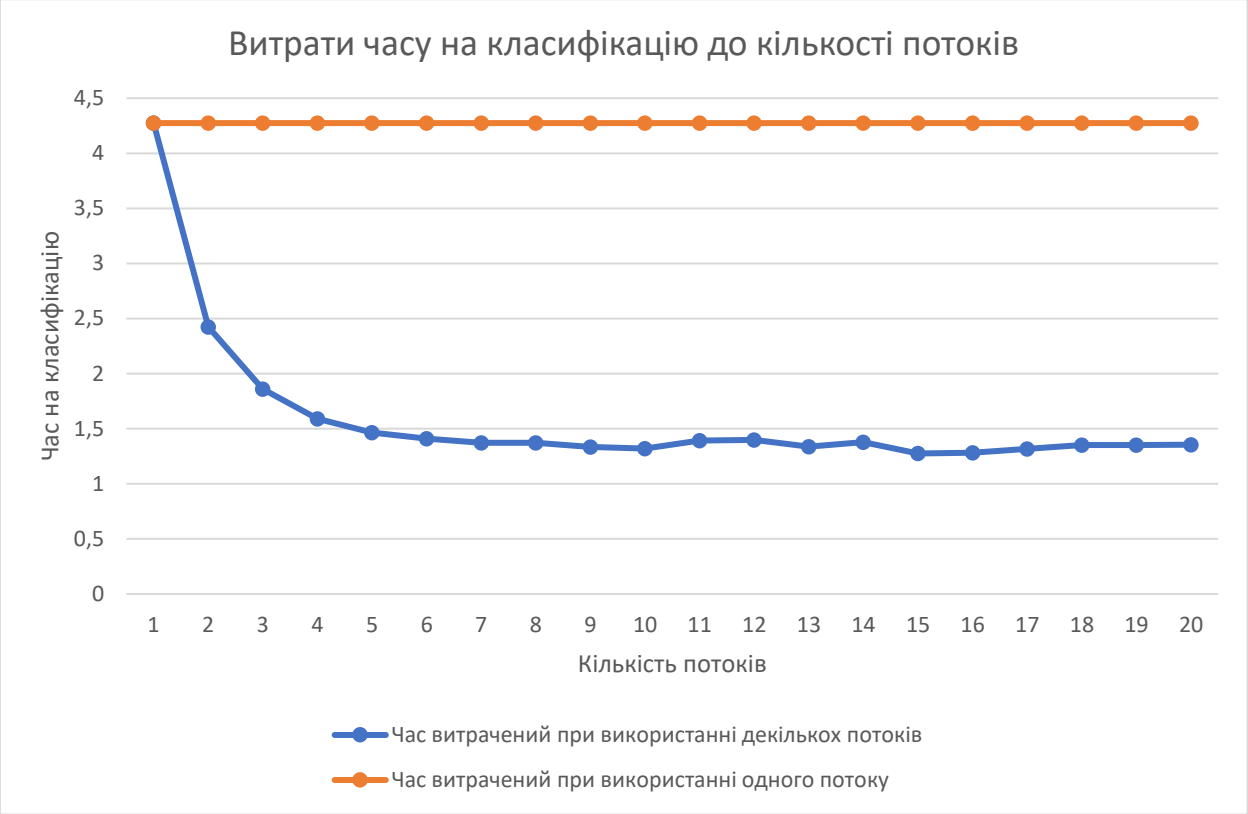

Рисунок 3.9 – Витрати часу на класифікацію до кількості потоків Syn.csv

Результат отриманий на найбільшому файлі (рисунок 3.9) надає найбільш цікаві результати. При класифікації великої кількості час, що необхідний для виконання класифікації швидко зменшується, після чого довго тримається на одному і тому самому рівні. Для різної кількості потоків. Можна зробити висновок, що від 8 до 20 потоків час на достатньо великому об’ємі даних сильно не знижується і тримається приблизно одного і того самого рівня.

Проте, зважаючи на графіки попередніх результатів можна зробити припущення що рано чи пізно графік все таки піде вгору і теж перетнеться з часом роботи одного потоку.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі було розглянуто роботу створеної програми, що виконує поставлену перед собою задачу, а саме зменшує час класифікації даних моделі Random Forest, а також побудову цієї моделі.

Також розкривається тема попередньої обробки даних для успішної та ефективної роботи з ними.

На графіках чітко представлено скільки часу необхідно для класифікації даних потрібно з одним потоком та з більшою кількістю. Знайдено оптимальну кількість потоків для кожного з випадків. Виявлено, що через мірно багато потоків може призвести до збільшення часу класифікації, що є неприйнятним.

За рахунок паралелізації час на класифікацію даних зменшився відповідно на такий відсоток:

- Portmap: 51.5%
- Small-Syn: 67%
- Syn: 67.4%

ВИСНОВКИ

На сьогоднішній день атаки відмови в обслуговуванні не новизна, а навіть звичайна буденність, адже вони трапляються усюди. Спеціалісти з кібербезпеки завжди мають бути напоготові, щоб впоратися з ними своєчасно, адже дана робота і ставить на меті своєчасність виявлення та попередження можливих наслідків такої атаки.

У даній дипломній роботі було розглянуто такі поняття як DDoS атаки, яку природу вони мають, які типи атак бувають та як ним протистояти, чому вони мають місце та є небезпечними. Було описано чому статистичні методи виявлення атак не є панацеєю і в якому напрямку треба рухатись, щоб позбутися проблем, які виникають перед спеціалістами в боротьбі з кібератаками.

Машинне навчання як ключ до вирішення певних проблем, а саме точності класифікації, точності прогнозу початку DDoS, виявлення атаки в загальному. Саме цех пунктів бракує статистичному аналізу. Запропоновано використовувати такий алгоритм як Random Forest через його високу ефективність за загальноприйнятими метриками.

Аналіз роботи алгоритму випадкового лісу докорінно дозволив зробити деякі заключення, що дозволили запропонувати модель, що привносить новизну та сприяє покращенню кібербезбезпеки в цілому. В третьому розділі ця модель паралельного алгоритму класифікації була реалізована.

Через зазначені вище критерії я вирішив створити заявку на авторське право у патентному бюро. Наразі заявка знаходиться у процесі виконання.

Також було показано, що існує потенціал до обробки масивних об’ємів даних за допомогою розподілених обчислень. Така тема може бути використана для майбутньої наукової дисертації.

СПИСОК ПОСИЛАНЬ

1. Метод виявлення мережевих атак в комп'ютеризованих системах управління. URL: http://konkurs.khnu.km.ua/wp-content/uploads/sites/25/2019/04/DP3_Eugen.pdf
2. Network Attacks and Their Detection Mechanisms. URL: https://www.researchgate.net/publication/263052997_Network_Attacks_and_Their_Detection_Mechanisms_A_Review
3. Random Forest Modeling for Network Intrusion Detection System. URL: https://www.sciencedirect.com/science/article/pii/S1877050916311127
4. Canadian Institute for Cybersecurity. URL: https://www.unb.ca/cic/datasets/
5. Understanding Random Forest. How the Algorithm Works and Why it Is So Effective. URL: https://towardsdatascience.com/understanding-random-forest-58381e0602d2
6. Intrusion-Detection-using-Machine-Learning-on-NSL--KDD-dataset. URL: https://github.com/Deepthi10/Intrusion-Detection-using-Machine-Learning-on-NSL--KDD-dataset/blob/master/IDS.ipynb
7. Distributed Data Processing 101 – The Only Guide You'll Ever Need. URL: https://www.scaleyourapp.com/distributed-data-processing-101-the-only-guide-youll-ever-need/
8. Методи боротьби з DoS або DDoS атаками. URL: https://wiki.tntu.edu.ua/Методи_боротьби_з_Dos_або_DDos_атаками
9. Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets. URL: https://arxiv.org/pdf/1702.03681.pdf
10. Statistical Analysis Driven Optimized Deep Learning System for Intrusion Detection. URL: https://www.researchgate.net/publication/327110465_Statistical_Analysis_Driven_Optimized_Deep_Learning_System_for_Intrusion_Detection

11. L. Galchynsky and M. Grayvoronsky, “Evaluation of machine learning methods to detect DoS / DDoS attacks in IoT,” Theoretical And Applied Cybersecurity Vol. 5 No. 1 (2022)
12. When Accuracy Isn’t Enough, Use Precision and Recall to Evaluate Your Classification Model. Will Koehrsen. URL: https://builtin.com/data-science/precision-and-recall
13. Deep learning approaches for detecting DDoS attacks: a systematic review. Meenakshi Mittal, Krishan Kumar, Sunny Behal. 2022. URL: https://link.springer.com/article/10.1007/s00500-021-06608-1

ДОДАТКИ

ДОДАТОК А

Поля в наборі даних

Source Port
Destination Port
Protocol
Flow Duration
Total Fwd Packets
Total Backward Packets
Total Length of Fwd Packets
Total Length of Bwd Packets
Fwd Packet Length Max
Fwd Packet Length Min
Fwd Packet Length Mean
Fwd Packet Length Std
Bwd Packet Length Max
Bwd Packet Length Min
Bwd Packet Length Mean
Bwd Packet Length Std
Flow Bytes/s
Flow Packets/s
Flow IAT Mean
Flow IAT Std
Flow IAT Max
Flow IAT Min
Fwd IAT Total
Fwd IAT Mean
Fwd IAT Std
Fwd IAT Max
Fwd IAT Min
Bwd IAT Total
Bwd IAT Mean
Bwd IAT Std
Bwd IAT Max
Bwd IAT Min
Fwd PSH Flags
Bwd PSH Flags
Fwd URG Flags
Bwd URG Flags
Fwd Header Length
Bwd Header Length
Fwd Packets/s
Bwd Packets/s
Min Packet Length
Max Packet Length
Packet Length Mean
Packet Length Std
Packet Length Variance
FIN Flag Count
SYN Flag Count
RST Flag Count
PSH Flag Count
ACK Flag Count
URG Flag Count
CWE Flag Count
ECE Flag Count
Down/Up Ratio
Average Packet Size
Avg Fwd Segment Size
Avg Bwd Segment Size
Fwd Avg Bytes/Bulk
Fwd Avg Packets/Bulk
Fwd Avg Bulk Rate
Bwd Avg Bytes/Bulk
Bwd Avg Packets/Bulk
Bwd Avg Bulk Rate
Subflow Fwd Packets
Subflow Fwd Bytes
Subflow Bwd Packets
Subflow Bwd Bytes
Init_Win_bytes_forward
Init_Win_bytes_backward
act_data_pkt_fwd
min_seg_size_forward
Active Mean
Active Std
Active Max
Active Min
Idle Mean
Idle Std
Idle Max
Idle Min
Label

ДОДАТОК Б

Код програми

```
from functools import reduce
import math
import sys
from threading import Thread
from typing import Dict, List, Tuple
import joblib
from sklearn.preprocessing import LabelEncoder
from tqdm import tqdm
import numpy as np
import pandas as pd
from pandas import DataFrame
import requests
from sklearn.ensemble import RandomForestClassifier
import warnings
import logging
from timeit import default_timer as timer
from sklearn.model_selection import cross_val_score
from sklearn import metrics


logging.basicConfig(stream=sys.stdout, level=logging.INFO)


CLF_FILE = "RandomForestTrained.joblib"
TEST_DATA_URL =
"http://205.174.165.80/CICDataset/CICDDoS2019/Dataset/CSVs/CSV-03-11.zip"


def merge_classifiers(clf1, clf2):
    n1, n2 = clf1.n_estimators, clf2.n_estimators
    clf1.estimators_ += clf2.estimators_
```

```
    clf1.n_estimators = len(clf1.estimators_)
    logging.debug(f"Combined two forests: {n1} + {n2} = {clf1.n_estimators}")
    return clf1


def strip_dataset_columns(dataset):
    "eliminate spaces before and after in column names: ' Label ' -> 'Label'"
    logging.debug("Stripping dataset")
    dataset.columns = [x.strip() for x in dataset.columns]


def drop_columns(labels: List[str], dataset):
    for label in labels:
        try:
            logging.debug(f"Trying to drop column {label}")
            dataset = dataset.drop(label, axis=1)
            logging.info(f"Column '{label}' dropped successfully")
        except KeyError as e:
            logging.warning(f"column '{label}' was not deleted.")

    return dataset


def fill_none(dataset):
    #mitigate None values in dataset
    for col in dataset.columns:
        try:
            logging.debug(f"Trying to fill None values in {col}")
            dataset[col].fillna(dataset[col].median().round(1), inplace=True)
            logging.debug(f"Successfully filled Nones in {col}")
```

```
        except:
            logging.warning(f"'{col}' Failed fill 'None' value with median")


def replace_infinity(dataset):
    for col in dataset.columns:
        #Elimintaion of Infinities with the largest real number
        logging.debug(f"replacing infinities in column {col} dataset")
        largest_number_in_col = dataset[col][dataset[col] != np.inf].max()
        dataset[col] = dataset[col].replace(np.inf, largest_number_in_col)


def split_test_training_data(dataset, frac=0.8) -> Tuple[DataFrame, DataFrame]:
    """Splint DataFrame to smaller datasets
    Default sizes:
     training dataset -- 80%
     training dataset -- 20%
    return: Tuple[training dataset, testing dataset]"""
    logging.debug(f"Splitting test and training data with {frac=}")
    training_data = dataset.sample(frac=frac, random_state=25)
    testing_data = dataset.drop(training_data.index)
    return (training_data, testing_data,)


def X_Y_data_split(dataset, y_col) -> Tuple[DataFrame, DataFrame]:
    logging.debug("Trying to separate X and Y")
    x = dataset.drop(y_col, axis=1)
    y = pd.DataFrame(dataset[y_col], columns=[y_col])
    logging.debug("X and Y separated successfully")
    return (x, y)
```

```
def _fit_clf(clf: RandomForestClassifier, X, Y):
    clf.fit(X,Y)


def _predict_clf(clf: RandomForestClassifier, X: DataFrame, thread_id: int, output:
Dict):
    output[thread_id] = list(clf.predict(X))


def time_measure(func):
    def wrapper(*args, **kwargs)   :
        start = timer()
        ret = func(*args, **kwargs)
        end = timer()
        logging.info(f"time measurment: {end - start}")
        return ret
    return wrapper


@time_measure
def create_clf_synchronycaly(X, Y, X_test, Y_test):
    clf = RandomForestClassifier(n_estimators=100, max_depth=20)
    clf.fit(X, Y)
    logging.info('Forest is ready. Trees:' + str(len(clf.estimators_)))
    logging.info(f"Score: {clf.score(X_test, Y_test)}")
    return clf
```

```
@time_measure
def create_clf_paralelly(X, Y, X_test, Y_test, n_jobs=10) ->
RandomForestClassifier:
    small_forest_clfs = [RandomForestClassifier(n_estimators=10, max_depth=20)
for _ in range(n_jobs)]
    threads: List[Thread] = [Thread(target=_fit_clf,args=[clf, X, Y]) for clf in
small_forest_clfs]

    for thread in threads:
        thread.start()

    for thread in threads:
        thread.join()

    logging.debug(f"Merging '{n_jobs}' classifiers")
    clf = reduce(merge_classifiers, small_forest_clfs)

    logging.info('Forest is ready. Trees:' + str(len(clf.estimators_)))
    logging.info(f"Score: {clf.score(X_test, Y_test)}")

    return clf


def split_dataset(data, parts):
    rows_count = math.ceil(len(data) / parts)
    return [data[rows_count*i:rows_count*(i+1)] for i in range(parts)]


def paralel_predict(clf, X, n_jobs=10):
    data_parts = split_dataset(X, n_jobs)
    predicted = {}
```

```
    threads = [Thread(target=_predict_clf, args=[clf, data_parts[i], i, predicted]) for i
in range(n_jobs)]

    @time_measure
    def _start_threads(threads):
        for thread in threads:
            thread.start()

        for thread in threads:
            thread.join()

    _start_threads(threads=threads)

    out = []
    for i in range(len(predicted)):
        out += predicted[i]

    return out


@time_measure
def synchronical_predict(clf: RandomForestClassifier, X):
    return clf.predict(X)


def preprocess_dataset(dataset, labels_to_drop=None):
    "Preprocessing data"
    if labels_to_drop is None:
        labels_to_drop = ['Unnamed: 0', 'Flow ID', 'Source IP', 'Destination IP',
            'Timestamp', 'SimillarHTTP', 'Dst IP', 'Src IP',
```

```
        'Fwd Header Length.1', 'Inbound']
    strip_dataset_columns(dataset)
    dataset = drop_columns(labels_to_drop, dataset)
    fill_none(dataset)
    replace_infinity(dataset)
    return dataset


def question():
    return input(f"Download Test Data from \"{TEST_DATA_URL}\"?(y/N): ")


def download_data():
    local_filename = TEST_DATA_URL.split('/')[-1]
    logging.info(f"Downloading file from \"{TEST_DATA_URL}\"")
    with requests.get(TEST_DATA_URL, stream=True) as r:
        r.raise_for_status()
        total_size_in_bytes= int(r.headers.get('content-length', 0))
        block_size = 128*1024
        progress_bar = tqdm(total=total_size_in_bytes, unit='iB', unit_scale=True)
        with open(local_filename, 'wb') as f:
            for chunk in r.iter_content(chunk_size=block_size):
                progress_bar.update(len(chunk))
                f.write(chunk)

    logging.info(f"File {local_filename} has been downloaded. Please unarchive.
Exitting...")


def confusion_matrix(Y_test, Y_predicted):
```

```
    print(pd.crosstab(Y_test, Y_predicted))


def score(clf, X, Y):
    accuracy = cross_val_score(clf, X, Y, cv=10, scoring='accuracy')
    print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))

    precision = cross_val_score(clf, X, Y, cv=10, scoring='precision')
    print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))

    recall = cross_val_score(clf, X, Y, cv=10, scoring='recall')
    print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))

    f = cross_val_score(clf, X, Y, cv=10, scoring='f1')
    print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

    warnings.filterwarnings("ignore")
    logging.basicConfig(stream=sys.stdout, level=logging.INFO)


def main():
    try:
        file = sys.argv[1]
        logging.info(f"Using {file}")
    except IndexError:
        file = '03-11/Small-Syn-.csv'
        logging.info(f"Not given any file to the program. Using {file}.")

    try:
        dataset = pd.read_csv(file, low_memory=False)
```

```
except FileNotFoundError:
    logging.error(f"Not found default file: {file}. Exitting...")
    if question().lower() == 'y':
        download_data()

    exit()

dataset = preprocess_dataset(dataset)

#Split data
training_data, testing_data = split_test_training_data(dataset)

X_testing_data, Y_testing_data = X_Y_data_split(testing_data, 'Label')
X_training_data, Y_training_data = X_Y_data_split(training_data, 'Label')

print("Training Data:")
print(X_training_data.shape)
print(Y_training_data.shape)

print("Testing Data:")
print(X_testing_data.shape)
print(Y_testing_data.shape)

# Encode 'Y' column
le = LabelEncoder()
Y_testing_data = le.fit_transform(Y_testing_data['Label'])
Y_training_data = le.fit_transform(Y_training_data['Label'])

logging.info("---Creating Forest Synchronycally---")
```

```
    clf = create_clf_synchronycaly(X_training_data, Y_training_data,
X_testing_data, Y_testing_data)
    logging.info("---Creating Forest in Paralel---")
    clf_paralel = create_clf_paralelly(X_training_data, Y_training_data,
X_testing_data, Y_testing_data, 10)


    logging.info("---Predicting Forest Synchronycally---")
    predicted = synchronical_predict(clf, X_testing_data)


    logging.info("---Predicting Forest in Paralel---")
    for i in range(2, 51):
        print(f'Threads: {i}. ', end='')
        predicted_paralel = paralel_predict(clf_paralel, X_testing_data, i)


    print("--- Analyze Synchronical ---")
    score(clf, X_testing_data, Y_testing_data)
    confusion_matrix(Y_testing_data, predicted)
    print("--- Analyze Paralel ---")
    score(clf_paralel, X_testing_data, Y_testing_data)
    confusion_matrix(Y_testing_data, predicted_paralel)


    predicted = le.inverse_transform(predicted)
    predicted_paralel = le.inverse_transform(predicted_paralel)
    logging.info("Synchronically predicted:\n" +
str(DataFrame(predicted).value_counts()))
    logging.info("Predicted in Paralel:\n" +
str(DataFrame(predicted_paralel).value_counts()))


    joblib.dump(clf_paralel, CLF_FILE, compress=3)
```

```
if __name__ == '__main__':
    main()
```