ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ

Пояснювальна записка

до магістерської роботи
на тему:

«ТЕХНОЛОГІЯ ВИЯВЛЕННЯ СКАНУВАННЯ МЕРЕЖІ НА БАЗІ МЕТОДІВ МАШИННОГО НАВЧАННЯ»

Виконав студент 6 курсу, групи БСДМ-62
спеціальності 125 Кібербезпека
освітньо-професійної програми «Інформаційна та
кібернетична безпека»
(шифр і назва спеціальності)
Кузьмінський В.С.
(прізвище та ініціали)
Керівник Гайдур Г.І.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)
Нормоконтролер Чумак Н.С.
(прізвище та ініціали)

КИЇВ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ПК – операційна система

ACK – Acknowledgement Field Significant

BIRCH – Balanced Iterative Reducing and Clustering Using Hierarchies

DNS – Domain Name System

DoS – Denial of Service

DT – Decision Tree

FIN – Choose Your Own Device

GUI – Graphic User Interface

ICMP – Internet Control Message Protocol

IP – Internet Protocol

MAC – Media Access Control

NASL – Nessus Attack Scripting Language

NSE – Nmap Scripting Engine

NULL – Not in List

OSINT – Open Source intelligence

PSH – Push Function

RST – Reset the Connection

RF – Random Forest

SNMP – Simple Network Management Protocol

SYN – Synchronize Sequence Numbers

SVM – Support Vector Machine

UDP – User Datagram Protocol

URG – Urgent Point Field Significant

ВСТУП

Актуальність дослідження. Кількість кібератак на елементи комп'ютерних мереж зростає з кожним роком. Недосконаліть програмного забезпечення мережних пристроїв, операційних систем, веб-додатків та мережних сервісів створює величезну різноманітність кібератак. Для проведення більшості кібернетичних атак першим етапом визначають мережну розвідку.

Одним з популярних методів проведення розвідки є сканування активних елементів комп'ютерної мережі. Зловмисники виявляють відкриті порти та з'ясовують, які сервіси працюють в цільовій системі. Тобто, визначають подальший вектор атаки, підготовують і проводять цілеспрямовану атаку проти виявлених сервісів і їх вразливостей.

Для протидії вторгненням до мережі важливо виявляти в масштабах реального часу процес сканування з метою запобігання можливих кіберінцидентів.

……

Вищенаведені аргументи актуалізують тему даної магістерської роботи, зміст якої становить дослідження щодо технології виявлення процесу сканувань мережі на базі методів машинного навчання.

Об'єкт дослідження – процес виявлення сканування мережі на базі методів машинного навчання.

Предмет дослідження – технологія виявлення процесу сканування мережі на базі методів машинного навчання.

Мета роботи – розробити варіант системи виявлення процесу сканувань та рекомендації щодо застосування технології машинного навчання.

Наукові завдання:

дослідити сутність проблеми сканування мережних хостів;

проаналізувати існуючі технології сканування комп'ютерної мережі;

встановити сутність технології машинного навчання;

проаналізувати методи та алгоритми машинного навчання;

проаналізувати способи використання машинного навчання для аналізу мережного трафіку.

Методи дослідження – опрацювання літератури за даною темою, аналіз експлуатаційної документації, міжнародних стандартів та їх порівняння, моделювання процесу виявлення сканування мережі на базі методів машинного навчання.

Практичне значення одержаних результатів полягає в розробці варіанту системи виявлення сканувань на базі методів машинного навчання та рекомендації щодо застосування технології машинного навчання для захисту від сканувань.

1 СУЧАСНІ ПІДХОДИ ПРОВЕДЕННЯ РОЗВІДКИ СТРУКТУРИ КОМП'ЮТЕРНОЇ МЕРЕЖІ

1.1. Сканування мережних хостів

Розвиток та поширення обчислювальної техніки передбачає ведення розвідки в інформаційному просторі. Класифікаційна ознака відносно нового виду технічної розвідки – комп'ютерной розвідки – спосіб добування інформації. Основним способом добування інформації в даному випадку є перехоплення або отримання інформації в комп'ютерах та мережах. З огляду на те, що комп'ютери стають основним засобом обробки і зберігання інформації, її можливісті та актуальність безперервно зростають. Комп'ютерна розвідка є одним з найбільш ефективних засобів протидії і боротьби з найрізноманітнішими комп'ютерними злочинами, оскільки в задачу комп'ютерної розвідки входить і пошук зловмисників в комп'ютерних мережах.

Комп'ютерна розвідка – це добування інформації з комп'ютерних систем та мереж, характеристик їх програмно-апаратних засобів і користувачів.

Ґрунтуючись на особливостях обробки, передачі і зберігання інформації та архітектури сучасних комп'ютерних систем, виділимо джерела інформації в комп'ютерній розвідці [1]:

1. дані, відомості та інформація, що обробляється, передається і зберігається, в комп'ютерних системах і мережах (електронні документи, бази даних, дані, що проходять через комутаційне обладнання);
2. технічні характеристики програмних, апаратних і програмно-апаратних комплексів (у тому числі засобів захисту інформації), що функціонують в інформаційних системах;
3. характеристики користувачів комп'ютерних систем і мереж (особиста інформація користувача, електронна пошта, паролі, IP-адреса);
4. методи і механізми захисту в комп'ютерних системах і мережах.

За масштабом (рис 1.1.) комп'ютерна розвідка поділяється на інтернет-розвідку, мережну розвідку та цільову розвідку [2]:

Рис 1.1 Класифікація комп’ютерної розвідки за масштабом

1. Інтернет розвідка – комп'ютерна розвідка в відкритих інформаційних ресурсах, здійснюється методами інтелектуального пошуку даних у відкритих джерелах. Інтернет величезне сховище корисної інформації і сьогодні всі без винятку можуть знайти інформацію, зодовільнить їхні інтереси, головне – обрати потужні пошукові системи і правильно сформувати запит.

2. Окремо можна виділити автоматизацію інтернет-розвідки. Технологія OSINT (Open Source Intelligence) відноситься до найбільш перспективних напрямків використання загальнодоступних джерел для збору інформації. Сюди входять газети, Інтернет, книги, телефонні довідники, наукові журнали, радіомовлення, телебачення, урядові звіти, технічна документація, керівні-інструкції і т.д.

3. Цільова розвідка – комп'ютерна розвідка в захищених інформаційних системах, що здійснюється шляхом дестабілізуючого впливу (комп'ютерних атак), подолання захисних механізмів і т.д.

4. Мережна розвідка – комп'ютерна розвідка в мережах передачі даних, яка здійснюється, як правило методами перехоплення і аналізу трафіку, а також шляхом ідентифікації доступних мережних хостів, сервісів і программного

забезпечення. Мережна розвідка перший крок при підготовці до тестування на вразливість та проведення атаки на мережні хости.

Серед основних способів реалізації розвідки мережі відзначають:

1. аналіз і перехоплення трафіку (сніффінг);
2. підміна довіреного вузла мережі (спуфинг);
3. атаки людина по середині (MITM).

Мережна розвідка спрямована на добування даних з комп'ютерних мереж, шляхом реалізації зондування мережі, ідентифікації віддалених служб і аналізу вразливостей ресурсів мережі. Перше що потрібно виконати це – ідентифікувати працюючі хости. Після чого необхідно визначити, які служби та додатки запущені на віддалених хостах мережі.

Дослідження мережі за допомогою проколу ICMP. ICMP відноситься до службових протоколів (рис 1.2.) і використовується для виявлення помилок на мережевому рівні. Як правило, в локальних мережах його не блокують, так як він часто використовується системними адміністраторами для пошуку несправності в мережі. Завдяки цьому за допомогою ICMP можливе дослідження працюючих в мережі пристроїв.

Type (Тип)	Code (Код)	Checksum (Контрольна сума)
Дані		

Рис 1.2 Структура протоколу ICMP

Повідомлення ICMP передаються у вигляді IP-датаграми, до них додається заголовок IP. Існують кілька типів повідомлень ICMP. Кожен має свій формат, при цьому всі вони містять такі три поля 8-бітного цілого числа, що позначає тип повідомлення (TYPE); 8-бітного поля коду (CODE), який конкретизує призначення повідомлення; 16- бітного поля контрольної суми (CHECKSUM) [3].

Всі типи повідомлень ICMP умовно ділиться на дві групи [4]: повідомлення про помилки (наприклад, Destination unreachable) та запити і відповіді (наприклад, Echo Request і Echo Reply).

Повідомленя про помилки містять заголовок, і перші 64 біта даних пакета IP, при передачі яких виникла помилка. Для більшості повідомлень помилки виводяться в поле коду. Для дослідження мережі використовуються Echo Reply, Echo Request і Timestamp.

В основу ідентифікації закладений наступний принцип. Хост, що відправляє ICMP-запит, встановлює значення полів Identifier, ці значення дозволяють визначити відповіді, які прийшли від різних хостів. А для того, щоб відрізнити кілька відповідей, які прийшли від одного хосту, використовується поле Sequence Number. В поле Code записується нуль, поле даних довільне (наприклад, алфавіт). Сторона, що відповідає повинна замінити значення поля Type на 0 і відправити датаграмму назад.

Одним з способів визначення доступності хоста є відправлення повідомлення ICMP Echo (Type 8). Якщо система працює і відсутня фільтрація трафіку даного типу, то у відповідь прийде повідомлення ICMP Echo Reply (Type 0).

Звернення відразу до декількох хостів (діапазону) з використанням ICMP-запитів (Echo) називається ICMP Sweep або Ping Sweep. Для дослідження великої мережі буде потрібно утиліта, здатна посилати ICMP – запити паралельно.

UDP Дослідження мережі. Метод визначення доступності вузла з використанням протоколу UDP називається UDP Discovery. Особливість цього протоколу – при передачі даних неможливо впевнитись в доставці пакету отримувачу, датаграми передаються без встановлення з’єднання. Якщо у відповідь на пакет було отримано повідомлення ICMP Destination Unreachable (Port Unreachable), це означає, що хост недоступний (і порт, зазначений в UDP- пакеті, закритий).

У випадку неотримання відповіді хосту можливі наступні варіанти: хост вимкнений або недоступний, на цільовому хосту ввімкнена фільтрація трафіку або зазначений в UDP-пакеті порт відкритий.

Використання протоколу UDP (рис.1.3) для виявлення пристроїв неефективно в зв’язку з наступними факторами. По-перше, це висока ступінь фільтрації UDP-трафіку. Так як через архітектурні особливості протоколу

(відсутність механізму підтвердження доставки) UDP використовують мало служб (найвідомішими є DNS, SNMP, Syslog), цей протокол часто фільтрується в мережі. Другим недоліком використання UDP є непередбачувана поведінка системи при отриманні UDP-пакета на відкритий порт. Більшість сканерів відправляють при скануванні порожні пакети. Припустимо, такий пакет відправили на 53-й порт сервера DNS. Отримавши його на відкритий порт, система спробує прочитати вміст. При цьому на рівні додатків будуть очікуватися дані певного формату (команди, параметри). Але так як там ніяких даних немає, сервіс цілком може повести себе не коректно або, що ще гірше для зловмисника, записати даний інцидент в журнал подій.

Порт відправника	Порт отримувача
Довжина	Контрольна сума
Дані	

Рис 1.3. Структура UDP протоколу

Деякі розробники змогли обійти другий недолік UDP-сканування. На дані порти відправляється не порожній UDP – пакет, а осмислений запит, на який повинна прийти відповідь. Таким чином, відповідь на запит буде в будь-якому випадку (відкритий порт або закритий), що підвищує достовірність цього методу при ідентифікації об'єктів мережі.

TCP Дослідження мережі. Метод визначення доступності хоста з використанням протоколу TCP називається TCP Ping. Протокол TCP використовується набагато частіше різними службами і додатками, що вимагають гарантованої передачі даних. Це основний інструмент дослідження мережі, так як засоби міжмережного екранування і системи виявлення вторгнень, як правило, дозволяють відправку пакетів на найбільш поширені порти [4].

При проведенні сканування важливим є вибір значень полів Source Port, Destination Port, поєднання флагів (поле Flags таблиця 1.1).

Вибір порту джерела залежить від фільтрації трафіку різного типу, а правильний вибір порту отримувача необхідний також з міркувань можливої фільтрації (зазвичай, це найбільш поширені порти 21, 22, 23, 25, 80).

Таблиця 1.1

Флаги (управляючі біти протоколу TCP)[2]

Поля флагів	Значення біта, якщо він дорівнює 1
URG	Ознака важливої інформації
ACK	Звертати увагу на поле "Номер підтвердження"
PSH	Якнайшвидше передати дані програмі застосування
RST	Перервати зв’язок
SYN	Ознака синхронізації номерів сегментів (використовують для налагодження сполучення)
FIN	Відправник закінчив надсилання пакетів

Відправка TCP-пакета з встановленим флагом SYN може бути використана для визначення доступності хоста наступним чином: якщо у відповідь на такий запит прийшов пакет з встановленими в заголовку флагом SYN-ACK або RST, то хост доступний. Якщо ж відповідь не приходить, то хост або недоступний, або даний тип трафіку фільтрується.

1.2. Програмні засоби сканування комп’ютерної мережі

Програмні засоби сканування структури та елементів мережі під час своєї роботи використовують два основних механізми. Зондування – механізм

активного аналізу, що імітує атаку, тим самим перевіряючи вразливості. Цей механізм працює повільно, але дозволяє підтвердити наявність вразливості і виявити «дірки» у захисті структури та елементів мережі. Сканування – механізм пасивного аналізу, що значно швидше зондування, але дає менш точні результати. Сканер шукає вразливості без підтвердження її наявності, використовуючи непрямі ознаки. За допомогою сканування визначаються відкриті порти, збираються пов'язані з ними заголовки та порівнюються з таблицею правил визначення пристроїв мережі, ОС і можливих «дір» [5].

Більшість сучасних сканерів безпеки мережі мають такі функції:

1. збір інформації про мережу, ідентифікація всіх активних пристроїв і сервісів, запущених на них;
2. виявлення потенційних вразливостей;
3. підтвердження знайдених вразливостей, для чого використовуються специфічні методи і моделюються атаки;
4. створення звітів;
5. автоматичне усунення вразливостей. Не завжди цей етап реалізується в мережевих сканерах безпеки, але часто зустрічається в сканерах системних. Існує можливість створення резервного сценарію, який може скасувати зроблені зміни, – наприклад, якщо після усунення вразливості буде порушено повноцінне функціонування мережі.

Одним з головних вимог до сучасних мережних сканерів вразливостей є підтримка різних операційних систем. Більшість популярних сканерів – кросплатформні (включаючи мобільні та віртуальні ОС).

Сканери мережі досліджують відразу кілька портів, що знижує час на перевірку. Сканер перевіряє не тільки ОС, а й програмне забезпечення, особливу увагу приділяючи популярним в хакерському середовищі додаткам: Adobe Flash Player, Outlook та браузерам.

Вони можуть перевіряти роздроблену мережу, що позбавляє адміністратора від необхідності оцінювати кожен хост окремо і кілька разів задавати параметри сканування.

Сучасні засоби сканування прості у використанні, їх роботу можна налаштувати відповідно до потреб мережі. Наприклад, вони дозволяють задати перелік перевірених пристроїв і типів вразливостей, вказати дозволені для автоматичного оновлення програми, встановити періодичність перевірки і отримання звітів. Отримавши детальний звіт про уразливість, його відразу можна виправити[6].

З додаткових можливостей варто виділити аналіз «історичних» даних. Збережена історія кількох сканувань дозволяє оцінити безпеку хоста на певному часовому інтервалі, оптимально налаштувати роботу програмного і апаратного забезпечення.

Найбільш популярні засоби сканування, що зарекомендували себе як професіональні продукти для виявлення вразливостей: Nessus, Masscan та Nmap.

Nessus

Запатентований сканер портів, призначений для автоматизації тестування та виявлення вразливостей шляхом сканування портів та служб, які їх використовують. Доступний у трьох версіях Nessus Home, Nessus Professional та Nessus Manager/Nessus Cloud. Безкоштовна версія Nessus Home дозволяє просканувати 16 IP, вартість Nessus Professional та Nessus Manager досить висока. Ця програма підтримує лише активні сканування, оновлення бази даних вразливостей сканування вимагають передплати[7].

Головні особливості Nessus:

1. клієнт-серверна архітектура. Серверна частина забезпечує проведення атак та збір інформації. Клієнтська – представлена як веб-інтерфейс, що забезпечує представлення зібраної інформації та зміну конфігурацій програми. Використовує архітектуру на основі сценаріїв для виявлення вразливостей;

2. написання власних сценаріїв атак NASL (Nessus Attack Scripting Language);

3. інтеграції з іншими засобами тестування безпеки для виявлення шкідливого програмного забезпечення, сканування веб-додатків, перевірку конфігурації системи;

4. генерування звітів сканування, підтримка проксі-аутентифікації;

Masscan

Це інструмент з відкритим вихідним кодом, який дозволяє виконувати сканування мережі асинхронно. Теоретично Masscan може обробляти до 10 мільйонів пакетів в секунду. Це забезпечується через власні мережний драйвер RF_DING DNA та TCP стек, в обхід ядра ОС. Один з найшвидших сканерів, що дозволяє розділити навантаження на кілька пристроїв. Може спричинити велику навантежність на мережу, що призводить до DoS, щоб уникнути цього сканер примусово виконує рандомізацію сканованих хостів.

Головні особливості Masscan:

1. консольний додаток з простим синтаксисом команд (рис. 1.3.);
2. дозволяє використовувати довільні діапазони ip-адрес та портів для сканування та може просканувати всю глобальну мережу Інтернет;
3. кросплатформений інструмент, але версія ОС Windows потребує додаткових налаштувань для підтримки всіх методів сканування.
4. утиліта повністю безкоштовна, вихідний код відкритий.

Nmap

Безкоштовна утиліта з відкритим вихідним кодом, призначена для дослідження та перевірки захищеності об'єктів мережі. Вважається еталоном серед сканерів портів. Сканер застосовує більше десяти методів сканування мережі, багатопоточний режим з динамічною зміною кількістю портів на потік, та може використовувати кілька ядер процесора для оптимізації швидкості сканувань.

Nmap – кросплатформне програмне забезпечення, для зручності та візіуалізації взаємодії представлене розширення Zenmap.

Широкий функціонал Nmap дозволяє ідентифікувати [8]:

1. порти;
2. мережні служби;
3. операційні системи;
4. програмне забезпечення служб мережі;

5. політику безпеки міжмережних екранів;
6. імена користувачів, від імені яких запущені служби.

Для обходу міжмережних екранів та систем виявлення вторгнень у Nmap використовуються: фрагментація пакетів, посилання пакетів з фіктивними сумами, зміна MAC та IP адрес, встановлення time-to-live пакету та маскування сканування за допомогою фіктивних хостів.

Nmap Scripting Engine (NSE) дозволяє користувачам писати скрипти, використовуючи мову програмування Lua та C, для взаємодії та автоматизації задач сканеру. Крім стандартної перевірки відкритих портів сканером Nmap, NSE дозволяє отримати також розширену інформацію про сервіси, запущених на них. За допомогою скриптів NSE можливе проведення аналізу сервісів мережі, аудит налаштування, перевірка, яка інформація доступна, чи не використовуються слабкі облікові дані. Фактично, скрипти NSE перетворюють Nmap в гнучку платформу для взаємодії з сервісами мережі[8].

1.3. Методи сканування Nmap.

Утиліта Nmap в процесі сканування мережі перебирає доступний діапазон портів і намагається підключитися до кожного з них. Для агресивного сканування з метою отримання максимально можливої інформації про цільові хости необхідно правильно задати параметри сканування. Для того щоб визначити, які комп'ютери працюють в мережі Nmap дозволяє проводити сканування мережі наступними методами: TCP сканування, Приховане сканування, FIN сканування, Null сканування, UDP сканування та XMAS сканування[9].

TCP сканування

Для TCP сканування хоста використовується команда:

```
$ nmap -sT 10.10.10.10
```

Відправка TCP-пакета з флагом SYN може бути використана для визначення доступності хоста наступним чином у Nmap (рис 1.4.):

Рис. 1.4. TCP сканування відкритого порту

Якщо зловмиснику у відповідь на запит прийшов пакет з встановленим флагом SYN-ACK, він відправляє ACK та RST-ACK пакети у відповідь – хост доступний

Якщо зловмисник у відповідь на запит прийшов пакет RST-ACK – хост закритий (рис 1.5.).

Рис. 1.5. TCP сканування закритого порту

Приховане сканування

Приховане сканування не завершує TCP з'єднання для пришвидшення сканування портів хоста. Вважається найпопулярнішим варіантом сканування, здатне проходити брандмауери цілі.

Для прихованого сканування хоста використовується команда:

```
$ nmap -sS 192.168.0.1
```

Зловмисник відправляє пакет з флагом SYN та чекає відповіді, якщо у відповідь на запит прийшов SYN, ACK – хост відкритий (рис 1.6.).

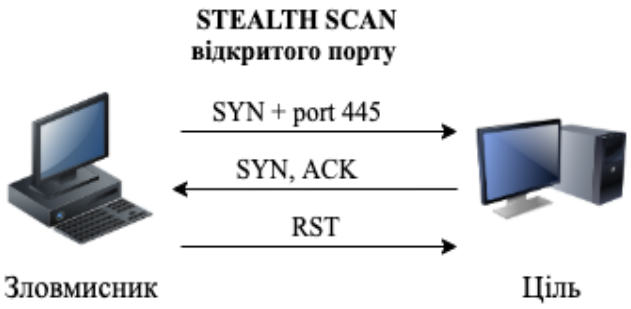


Рис. 1.6. Приховане сканування відкритого порту

У випадку, якщо порт закритий пакети зловмисника та цільового хоста відправляються так само як у випадку з TCP скануванням.

FIN Scan скануваннння

Пакет FIN використовується для няершення TCP-з'єднання між зловмисником і цільовим портом, як правило, після завершення передачі даних. Замість пакета SYN Nmap розпочинає сканування FIN за допомогою пакета FIN. Якщо порт відкритий, відповідь не надходитиме з порту призначення (Рис. 1.7.).

Рис. 1.7. FIN сканування відкритого порту

Для FIN сканування хоста використовується команда:

```
$ nmap -sF 192.168.0.1
```

Зловмисник відправляє пакет з флагом FIN та чекає відповіді, якщо у відповідь на запит прийшов RST, ACK – хост закритий (рис 1.8.).

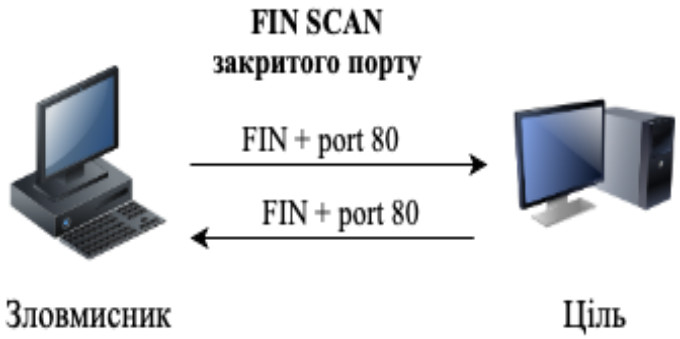


Рис. 1.8. FIN сканування закритого порту

Null Scan сканування

Null сканування - це серія TCP-пакетів, які містять порядковий номер «нулів» (0000000), а оскільки жоден прапорець не встановлений, адресат не знатиме, як відповісти на запит. Він відкине пакет, і відповідь не буде надіслана, це вказує на те, що порт відкритий (Рис. 1.9.).

Рис. 1.9. NULL сканування відкритого порту

Для NULL сканування хоста використовується команда:

```
$ nmap -sF 192.168.0.1
```

Зловмисник відправляє пакет з флагом NULL та чекає відповіді, якщо у відповідь на запит прийшов RST, ACK – хост закритий (рис 1.10.).

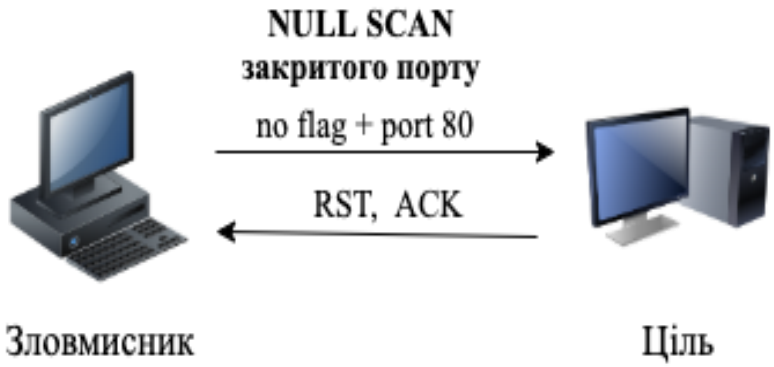


Рис. 1.10. NULL сканування закритого порту

UDP scan сканування

Для деяких загальних портів, таких як 53 та 161, для збільшення швидкості відповіді надсилається корисне навантаження, специфічне для протоколу, цільовий хост відповість пакетом UDP, якщо що він відкритий. У випадку після повторних передач відповіді не отримано, порт класифікується як відкритий або відфільтрований (рис 1.11).

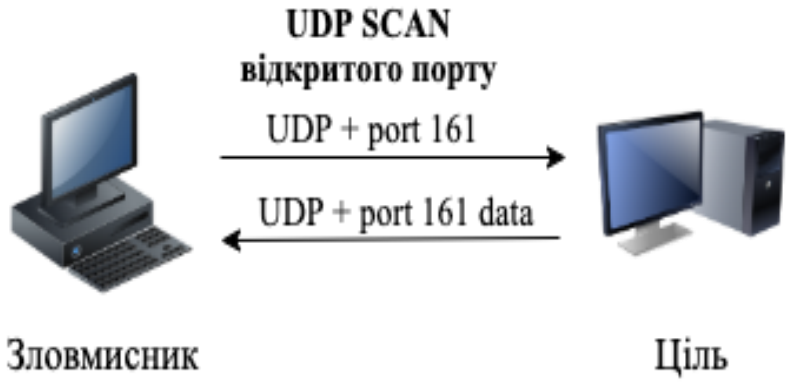


Рис. 1.11. UDP сканування відкритого порту

Для UDP сканування хоста використовується команда:

```
$ nmap -sU 192.168.0.1
```

Зловмисник відправляє UDP пакет, якщо у відповідь на запит прийшов ICMP пакет з інформацією, що порт недоступний – хост закритий (рис 1.12.).

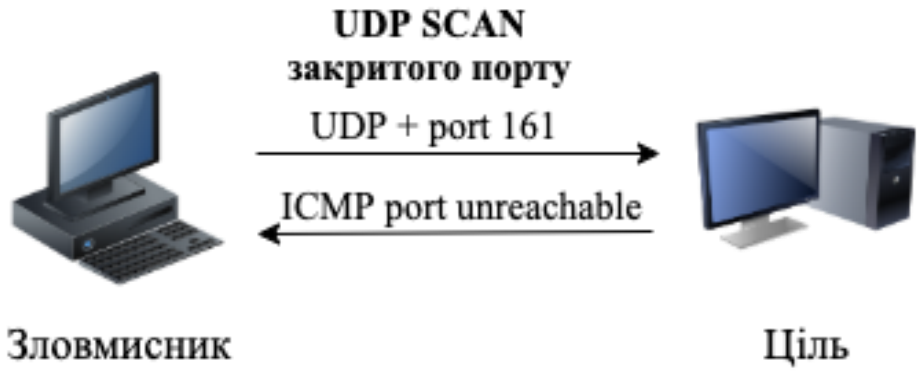


Рис. 1.12. UDP сканування закритого порту

XMAS Scan сканування

Ці скани призначені для маніпулювання прапорцями PSH, URG та FIN заголовка TCP, встановлює прапори FIN, PSH та URG, висвітлюючи пакет, як ялинку. Коли зловмисник надіслав пакет FIN, PUSH та URG до порту, і якщо порт відкритий, то адресат відкине пакети і не надішле жодної відповіді (рис. 1.13).

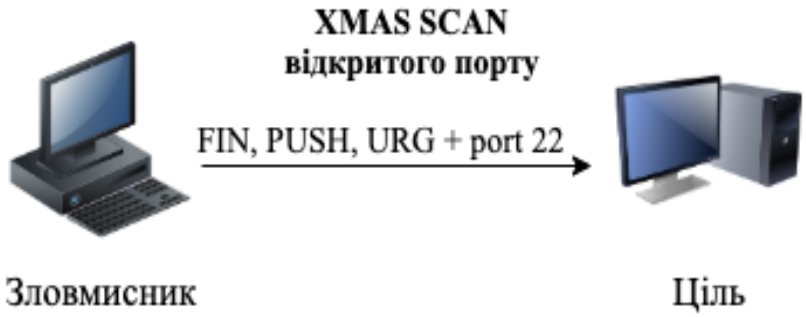


Рис. 1.13. XMAS сканування відкритого порту

Для UDP хоста використовується команда:

```
$ nmap -sX 192.168.0.1
```

Зловмисник відправляє пакет з флагом FIN, PUSG, URG та чекає відповіді, якщо у відповідь на запит прийшов RST, ACK – хост закритий (рис 1.14.).

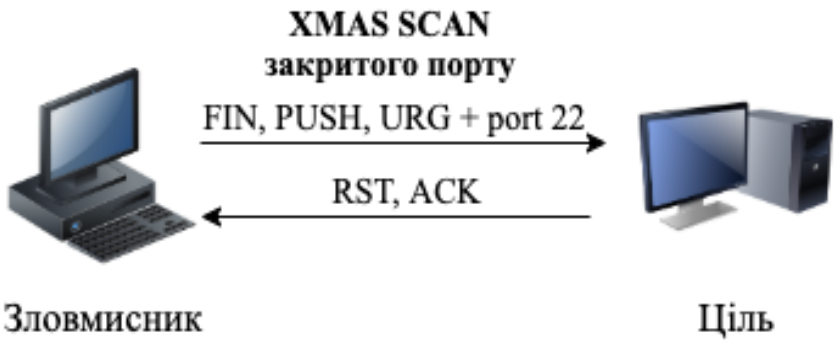


Рис. 1.14. XMAS сканування закритого порту

Висновок.

Отже, сканування служб і сервісів, запущених на комп'ютерах мережі є першим етапом підготовки і проведення атаки на комп'ютерні системи. Для цих цілей зловмисники використовують мережні сканери. Для виявлення сканувань використовують брандмауери, журнали сервісів та системи виявлення вторгнень.

Утиліта Nmap – вважається кращим сканером, що використовує різні методи сканування за допомогою флагів або надсилання корисного навантаження. Крім того Nmap підтримує використання скриптів з метою автоматизації процесу сканування та пошуку вразливостей.

2 АНАЛІЗ ЗАСТОСУВАННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ

2.1. Методи прийняття рішень на основі нейронних мереж

Машинне навчання – це використання математичних моделей даних, які допомагають комп'ютеру навчатися без безпосередніх інструкцій. Вважається однією з форм штучного інтелекту. За допомогою алгоритмів машинного навчання виявляються закономірності в даних. На основі цих закономірностей створюється модель даних для прогнозування. Чим більше даних обробляє така модель і чим довше вона використовується, тим точніше стають результати. Це дуже схоже на те, як людина відточує навички на практиці[10].

Адаптивний характер машинного навчання підходить для сценаріїв, в яких дані постійно змінюються, властивості запитів або завдань нестабільні або написати код для вирішення завдання фактично неможливо.

Машинне навчання має широке застосування на практиці, а не тільки аналітичний і теоретичний зміст. Перевірка різних методів і алгоритмів на реальних даних досягається за рахунок емпіричного досвіду. Для досягнення ефективної роботи запропонованого рішення, застосовуються додаткові методи, які відображають помилки прогнозованих значень від реальних даних.

Машинне навчання базується на ідеї про те, що аналітичні системи можуть вчитися виявляти закономірності і приймати рішення з мінімальним втручанням людини.

Нейронні мережі зробили важливий прорив в технологіях машинного навчання, автоматизувавши процеси векторизації складно структурованих об'єктів. Кожна нейронна мережа містить безліч параметрів, які неможливо налаштувати вручну, і тому ми навчаємо її на масиві даних: в процесі навчання і перенавчання вагові коефіцієнти нейронів безперервно змінюються і налаштовуються так, щоб результат обчислень і обробки сигналів став осмисленим [11].

Вперше математична модель штучного нейрону була запропонована нейрофізіологом Уореном Маккалохом (Warren McCulloch) та математи- ком Уолтером Пітсом (Уолтер Піттс) разом із моделлю нейронної мережі в 1943 році.

Математична модель елементарного нейрону (рис.2.1) описується за допомогою системи рівнянь та має вираз (2.1):

$$\begin{cases} y = \int(x) \\ s = \sum_{i=1}^{n} x_i * w_i + T \end{cases} \qquad (2.1)$$

де: x_i – вхідні сигнали; w_i – вагові коефіцієнти; T – пороговий рівень нейрону; $\int$ – функція активація нейрону.

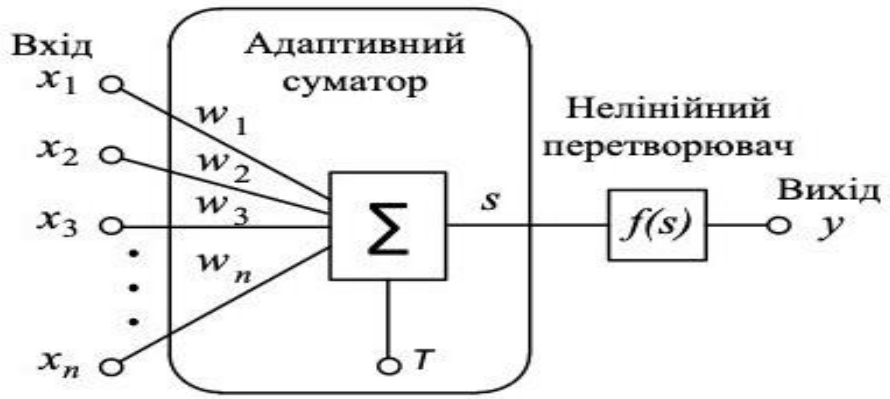


Рис.2.1 Графічне зображення нейрону

Тобто штучний нейрон складається з двох частин — адаптивного суматора, який сумує вхідні сигнали та нелінійного перетворювача, який ще називають функцією активації. Навчання такого нейрону в більшості випадків зводиться до корекції вагових коефіцієнтів, так як одночасна корекція вагових коефіцієнтів і параметрів функції активації призводить до різкого зростання часу навчання.

Останній компонент штучного нейрона – функція активації. В теорії побудови нейронних мереж застосовують велику кількість функцій активації, серед них слід виокремити, які використовуються найчастіше.

Функція Хевісайда

Значення виходу нейрона рівне нулю до тих пір, доки на виході суматора не буде значення, яке перевищує пороговий рівень. Як тільки це сталося – нейрон переходить в збуджений стан і на виході з'являється одиниця:

$$\int(s) = \begin{cases} 0 & s < T \\ 1 & s \geq T \end{cases} \qquad (2.2)$$

де s – значення виходу нейрона, T – пороговий рівень та має вираз (2.2).

До переваг функції(рис.2.2) слід віднести простоту описання та швидкість розрахунку. З недоліків є непридатність для роботи з неперервними сигналами (вихід приймає значення або 0, або 1), а відсутність першої похідної не дозволяє використовувати деякі з методів навчання. Крім того при використанні функції Хевісайда наявність прихованих шарів у нейронній мережі є надлишковою, так як вони не впливають на роздільну здатність нейронної мережі та на швидкість навчання [12].

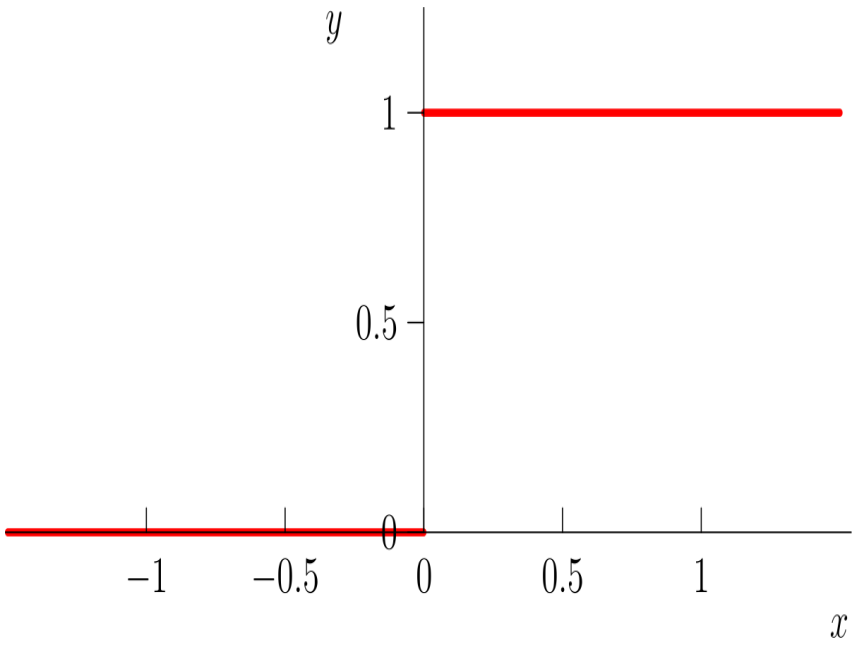


Рис.2.2 Функція Хевісайда

Лінійна функція активації з насиченням

Лінійна функція з насиченням(рис.2.3) має проміжні значення в діапазоні від 0 до 1, що дає змогу більш широкого використання в системах класифікації образів, також таку функцію інтерпретують, як апроксимаційну характеристику нелінійного підсилювача.

$$\int(s) = \begin{cases} -1 & s < -T \\ 1 & s > T \\ s & -T < s < T \end{cases} \qquad (2.2)$$

де s – значення виходу нейрона, T – пороговий рівень та має вираз (2.3).

Недоліком функції: відсутність першої похідної.

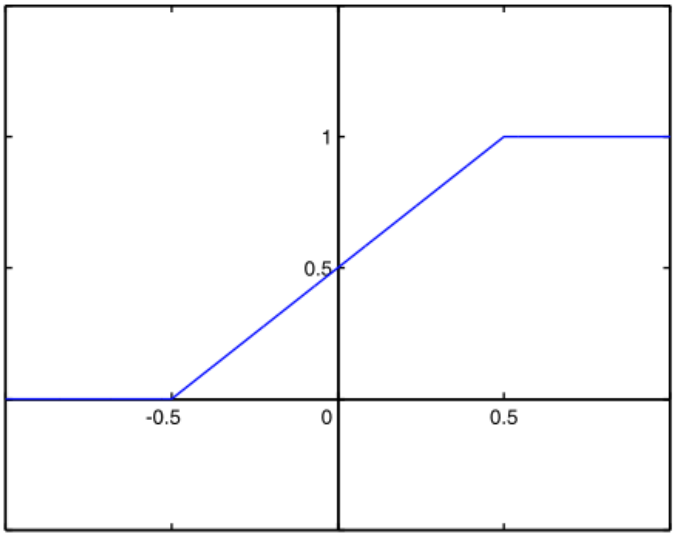


Рис. 2.3. Лінійна функція активації з насиченням

Сигмоїдальна функція активації

Сигмоїдальні функції активації відносяться до стискаючих нелінійних функцій. Введення нелінійності в роботу нейронної мережі дозволяє будувати ефективні багатошарові нейронні мережі. На відміну від порогової, сигмоїдальні функції(рис 2.4.) диференційовані на всій числовій осі та мають властивість до підсилювання слабких сигналів краще, ніж великих, і тому запобігають насиченню нейронної мережі великими вхідними сигналами [12].

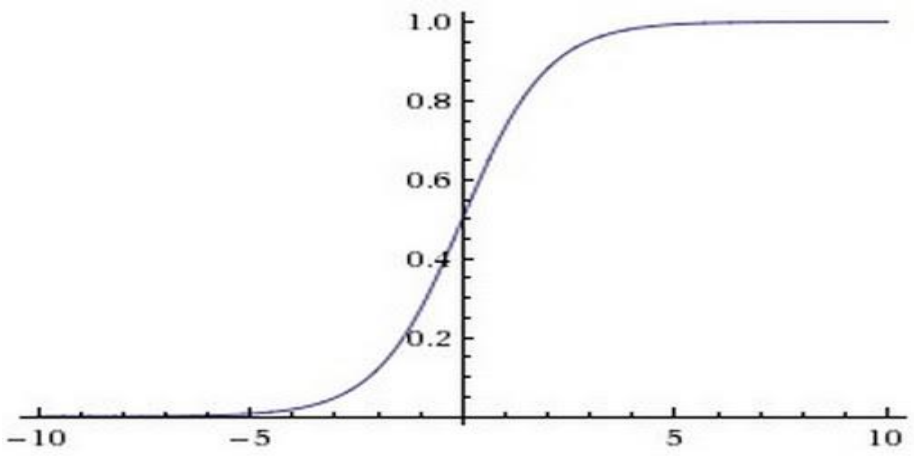


Рис 2.4. Сигмоїдальна функція

Використання сигмоїдальних функції дозволило перейти від дискретних сигналів до неперервних.

До сигмоїдальних функцій відноситься логістична функція, яка в залежності від значення а приймає різний кут нахилу, що можна викори- стовувати для регулювання підсилення слабких сигналів та зміни швидкості навчання мережі:

$$\int (s) = \frac{1}{1+e^{-as}} \qquad (2.4)$$

де а – коефіцієнт крутизни логістичної функції та має вираз (2.4).

Крім логістичної функції до сигмоїдальних відноситься і гіперболічний тангенс, який на відміну від попередньої функції має діапазон вихідних значень, який лежить в інтервалі (-1;1), що дозволяє нейронній мережі працювати не лише з додатними, але й з від'ємними величинами. Від вибору функції активації залежить метод навчання нейронної мережі.

Вибір функції активації та її параметрів залежить від діапазону вхідних та вихідних значень мережі і методу навчання. На сьогодні ще не розроблено способу визначення необхідної функції активації . Особливо це стосується задач, які не відносяться до класичних (з огляду на використання нейронних мереж) [12].

2.2. Методи машинного навчання

Нейронні мережі, на відміну від статистичних методів багатовимірного класифікаційного аналізу, базуються на паралельній обробці інформації і мають здатність до самонавчання, тобто отримувати обґрунтований результат на підставі даних, які не зустрічалися в процесі навчання.

Перевага використання нейронних мереж, як інструменту оцінки стану об’єкта полягає в тому, що тотожність між величинами заздалегідь не встановлюються, оскільки метод передбачає вивчення існуючих взаємозв’язків на готових моделях.

Для нейронних мереж також не потрібно ніяких припущень щодо основного розподілу сукупності, а також, на відміну від багатьох традиційних статистичних методів, вони можуть працювати з неповними даними. Нейронні мережі представляють нову і дуже перспективну обчислювальну технологію. Здатність до моделювання нелінійних процесів, роботи з зашумленими даними і адаптивність дають можливість застосовувати нейронні мережі для вирішення широкого класу завдань

Для ефективного використання нейронних мереж необхідна наявність достатнього обсягу навчальної вибірки, використовуючи яку нейронну мережу можна навчити.

Нейронні мережі навчають на прикладах. Розробник нейронної мережі підбирає представницькі дані, а потім запускає алгоритм навчання, який автоматично сприймає структуру даних. При цьому від розробника потрібно набір евристичних знань про те, як слід відбирати і готувати дані, вибирати потрібну архітектуру мережі та інтерпретувати результати[13].

За методами навчання нейронні мережі класифікуються:

1. навчання з вчителем, або контрольоване навчання;
2. напівавтоматичне навчання або часткове навчання ;
3. навчання без вчителя;
4. навчання з підкріпленням.

5. глибоке навчання

Навчання з вчителем

Навчання з вчителем – категорія машинного навчання, об’єднуючі алгоритми та методи побудови моделей на основі множини прикладів та містять пари «відомий вхід – відомий вихід». До алгоритмів навчання з учителем належать класифікація та регресія.

Мета класифікації полягає в тому, щоб спрогнозувати мітку класу (class label) із заздалегідь визначеного списку можливих варіантів за набором пов’язаних компонентів.

Класифікація іноді розділяється на бінарну класифікацію (binary classification), і мультикласову (multiclass classification), коли в класифікації бере участь більше двох класів. Бінарну класифікацію можна представити як ціле число в форматі «0» та «1», в мультикласову відбувається перетворення мітки в числове значення.

Мета регресії полягає в тому, щоб спрогнозувати безперервне число або число з плаваючою точкою якщо висловлюватись у термінах програмування, або дійсне число якщо говорити мовою математичних термінів. Прогнозоване значення доходу являє собою суму і може бути будь-яким числом в заданому діапазоні.

Алгоритми регресії моделюють залежність міток від пов’язаних компонентів, щоб визначити закономірність змін міток при різних значеннях компонентів

Напівавтоматичне навчання

Напівавтоматичне навчання – категорія машинного навчання, різновид навчання з учителем, яке використовує, як невелику кількість розмічених даних і велику кількість нерозмічених даних.

Напівавтоматичне навчання займає проміжну позицію між навчанням без вчителя та навчанням з учителем, використовується щоб значно поліпшити точність навчання.

Навчання без вчителя

Навчання без вчителям – категорія машинного навчання, об’єднуюча алгоритми коли відповідь невідома і відсутній учитель, який вказує відповідь алгоритму.

У машинному навчанні без учителя є лише вхідні дані і алгоритму необхідно знайти закономірності у цих даних. До таких алгоритмів належать задачі кластеризації, зменшення розмірності та пошуку правил.

Неконтрольовані перетворення (unsupervised transformations) – це алгоритми, що створюють нове уявлення даних, яке на відміну від вихідного уявлення людині або алгоритму машинного навчання буде обробити легше. Застосування неконтрольованих перетворень – скорочення розмірності даних що складається з безлічі ознак. Мета якого полягає в знаходженні нового спосібу представлення даних, узагальнюючи основні характеристики і отримуючи меншу кількість ознак. Загальнопоширене застосування скорочення розмірності – це отримання двовимірного простору з метою візуалізації. Ще одне застосування неконтрольованих перетворень буде пошук компонент, з яких «складаються» дані.

Мета кластеризації полягає в тому, щоб розбити задану вибірку об’єктів на підмножини, які називаються кластерами, так, щоб кожен кластер складався з схожих об’єктів, а об’єкти різних кластерів істотно відрізнялися. Кластеризація дозволяє виділити нетипові об’єкти, які не вдається зіставити з жодним кластером та спростити обробку даних, розбиваючи на групи схожі об’єкти.

Навчання з підкріпленням.

Навчання з підкріпленням – це категорія машинного навчання, що передбачає навчання на практиці. У той час як інші методи машинного навчання припускають пасивну передачу вхідних даних і виявлення в них структур, для навчання з підкріпленням використовуються агенти навчання, щоб забезпечити активне прийняття рішень і навчання на власних результатах.

Глибоке навчання.

Глибоке навчання – це алгоритми машинного навчання, які використовують великі об’єми розмічених даних та архітектуру нейронної мережі, що містять

велику кількість прихованих шарів. Крім того, глибоке навчання виконує «наскрізне навчання» – мережі надаються необроблені дані та завдання для виконання, наприклад класифікація, і вона навчається робити це автоматично.

2.3. Класичні алгоритми машинного навчання

Машинне навчання включає великий спектр підходів і методів збору, обробки, аналізу даних будь-якого розміру. Окремим практично важливим напрямком даної форми штучного інтелекту є робота з великими даними за допомогою нових принципів математичного та обчислювального моделювання, коли класичні методи перестають працювати через неможливість їх масштабування.

Кластеризація K-MEANS

Алгоритм роботи k-Means складається з чотирьох кроків:

1. Задається число кластерів k, яке повинно бути сформовано з об’єктів вихідної вибірки.

2. Випадковим чином вибирається k записів вихідної вибірки, які будуть використовуватись як початкові центри кластерів. Початкові точки, з яких потім «виростає» кластер, часто називають «насінням» (від англ. Seeds – насіння).

3. Для кожного запису вихідної вибірки визначається найближчий до неї центр кластера.

4. Виконується обчислення центроїдів – центрів тяжіння кластерів. Для цього шляхом простого визначення середнього для значень кожної ознаки для всіх записів в кластері. Наприклад, якщо в кластер увійшли три записи з наборами ознак (x_1, y_1), (x_2, y_2), (x_3, y_3), то координати його центроїда будуть розраховуватися наступним чином (2.5):

$$(x, y) = \left(\frac{x_1+x_2+x_3}{3}, \frac{x_1+x_2+x_3}{3}\right) \qquad (2.5)$$

Після чого старий центр кластера зміщується в його центроїд. Таким чином, центроїди стають новими центрами кластерів для наступної ітерації алгоритму.

Кроки 3 і 4 повторюються до тих пір, поки виконання алгоритму не буде розірвано або не буде виконана умова відповідно до деякого критерію збіжності.

Зупинка алгоритму відбувається тоді, коли кордони кластерів і розташування центроїдів не перестануть змінюватися від ітерації до ітерації, тобто на кожній ітерації в кожному кластері буде залишатися один і той же набір записів. На практиці алгоритм k-Means зазвичай знаходить набір стабільних кластерів за кілька десятків ітерацій.

Що стосується критерію збіжності, то найчастіше використовується критерій суми квадратів помилок між центроїдом кластера і всіма записами, тобто має вираз (2.6):

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} (p - m_i)^2 \qquad (2.6)$$

Де $p \in C_i$ – випадкова точка з даних, що належить кластеру C_i, m_i –центроїд цього кластера. Іншими словами, алгоритм зупиниться тоді, коли помилка E досягне досить малого значення.

До головних властивостей цього алгоритму відноситься:

1. Помірні обчислювальні витрати, які зростають лінійно зі збільшенням числа записів вихідної вибірки даних. Обчислювальна складність алгоритму визначається як (2.7):

$$O(n) = k \times n \times l \qquad (2.7)$$

де k - число кластерів, n - число записів, l - число ітерацій.

Оскільки k і l задані, то обчислювальні витрати зростають пропорційно числу записів вихідної множини;

2. Результати його роботи не залежать від порядку проходження записів у вихідній вибірці, а визначаються тільки вибором вихідних точок.

Недоліком, що властивий цьому алгоритмом, є відсутність чітких критеріїв вибору числа кластерів, цільової функції їх ініціалізації і модифікації. Крім цього, він є дуже чутливим до шумів і аномальних значенням в даних, вони здатні значно вплинути на середнє значення, яке використовується при обчисленні положень центроїд[14].

Приклад роботи алгоритму k-Means, що розбиває об'єкти на 3 кластери, наведено на рис 2.5.

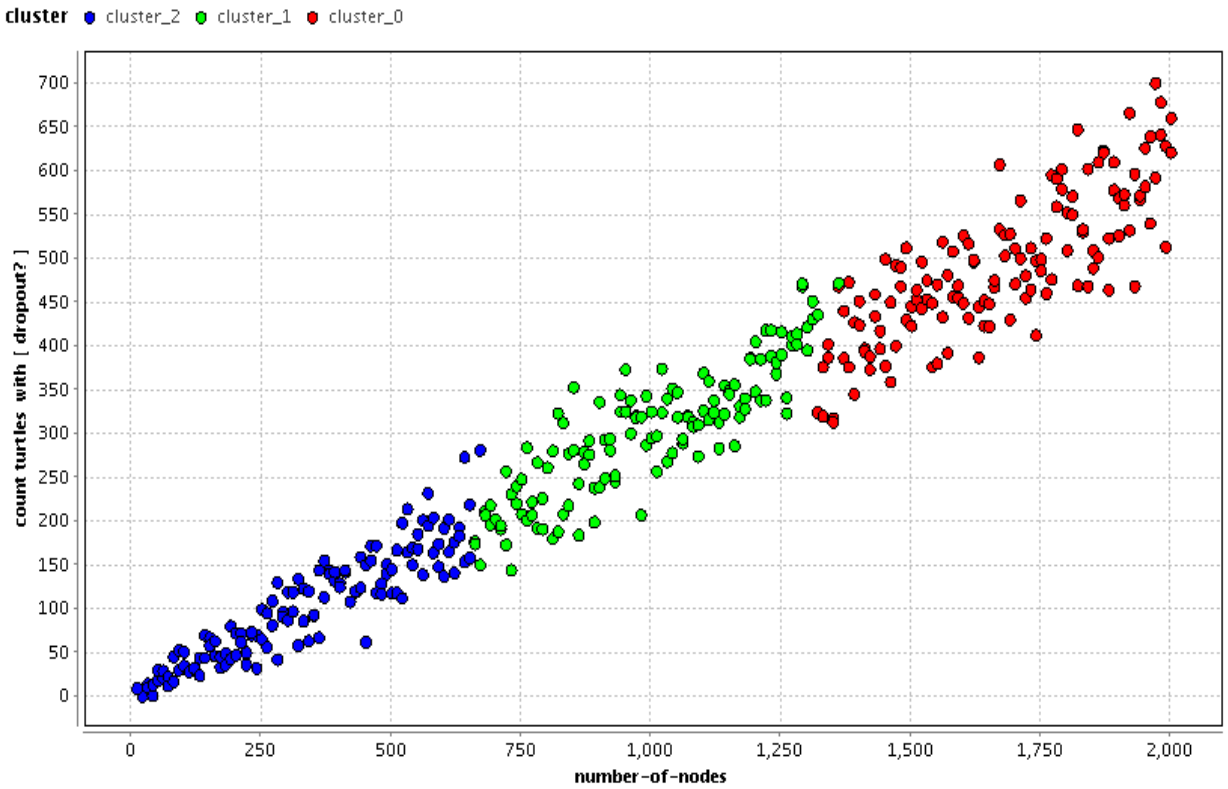


Рис. 2.5. Кластеризація алгоритму k-Means

Кластеризація GAUSSIAN MIXTURE

Алгоритм Gaussian Mixture починається з кроку ініціалізації, який призначає параметри моделі до прийнятних значень на основі даних. Виконання алгоритму

продовжуться поки не буде виконана умова відповідно до деякого критерію збіжності [15].

Алгоритм роботи складається з трьох кроків:

1. Випадково ініціалізуються значення об’єктів вихідної вибірки
2. Визначається дисперсія всіх компонентів за формулою (2.8):

$$\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \qquad (2.8)$$

Де x_i −параметр об’єкту вихідної вибірки, $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$

3. Рівномірно розподіляються всі компоненти вибірки має вираз (2.9):

$$\frac{1}{K} \qquad (2.9)$$

Алгоритм використовується у випадках, коли кількість компонентів вибірки невідома. Приклад роботи алгоритму Gaussian Mixture, що розбиває об'єкти на кластери, наведено на рис 2.6.

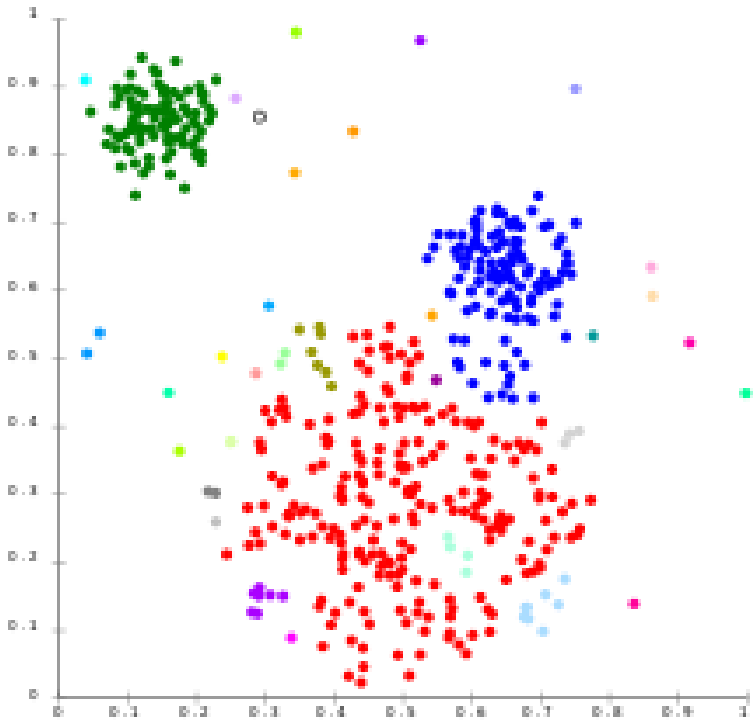


Рис. 2.6. Кластеризація алгоритму Gaussian Mixture

Кластеризація BIRCH

Алгоритм інтелектуального аналізу даних без вчителя, що представляє із себе ієрархічну кластеризацію великого розміру даних. Головною перевагою методу виступає динамічна кластеризація даних у міру їх надходження з метою отримання оптимального результату для наявних ресурсів: тимчасових рамок і пам'яті. Даний алгоритм часто вимагає одного проходу по базі даних [16]. Кластеризація BIRCH передбачає двоетапний процес кластеризації. Даний метод призначений для кластеризації даних великих розмірів, здатний обробляти тільки числові дані.

Алгоритм роботи складається з чотирьох кроків:

1. Наявні дані завантажуються в пам'ять. Первісна побудова в пам'яті кластерного дерева по елементам, отриманих в результаті первинного сканування набору даних.

2. При необхідності виконується стиснення даних (ущільнення). Дані стискаються до прийнятних розмірів за допомогою зменшення і перебудови кластерного дерева зі збільшенням граничної величини.

3. Виконання глобальної кластеризації. На листкових компонентах кластерного дерева застосовується обраний алгоритм кластеризації. Після цього кроку отримуємо набір кластерів, які містять основні схеми розподілу в даних. Однак можуть існувати невеликі локальні неточності, які можуть бути оброблені необов'язковим кроком 4.

4. При необхідності виконується поліпшення кластерів. Отримані в 3 етапі центру ваги кластерів використовуються як основа і відбувається перерозподіл даних між схожими (близькими) кластерами. Даний етап забезпечує потрапляння однакових елементів в один кластер.

Приклад роботи алгоритму BIRCH, що розбиває об'єкти на кластери, наведено на рисунку 2.7.

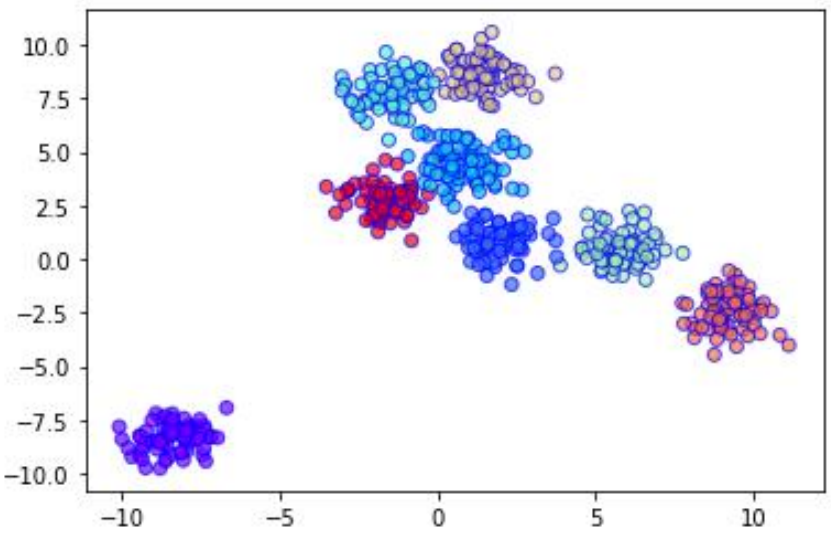


Рис. 2.7. Кластеризація алгоритму BIRCH

Класифікація SVM

Метод опорних векторів (англ. SVM, support vector machine) - набір схожих алгоритмів навчання з учителем, що використовуються для задач класифікації та регресивного аналізу. Особливою властивістю методу опорних векторів є невпинне зменшення емпіричної помилки класифікації і збільшення відстані (margin), тому метод також відомий як метод класифікатора з максимальною відстанню. Принцип роботи наступний: Враховуючи набір навчальних прикладів, кожен з яких позначається як такий, що належить до тієї чи іншої з двох категорій, алгоритм навчання SVM створює модель, яка призначає нові приклади однієї категорії або іншої, що робить його неімовірнисним бінарним лінійним класифікатором (хоча методи наприклад, масштабування Platt існує для використання SVM в імовірнісному класифікації). SVM-модель(рис 2.8.) являє собою представлення прикладів як точок в просторі, закріплені таким чином, що приклади окремих категорій ділиться явним розділенням, яке є максимально можливим[17].

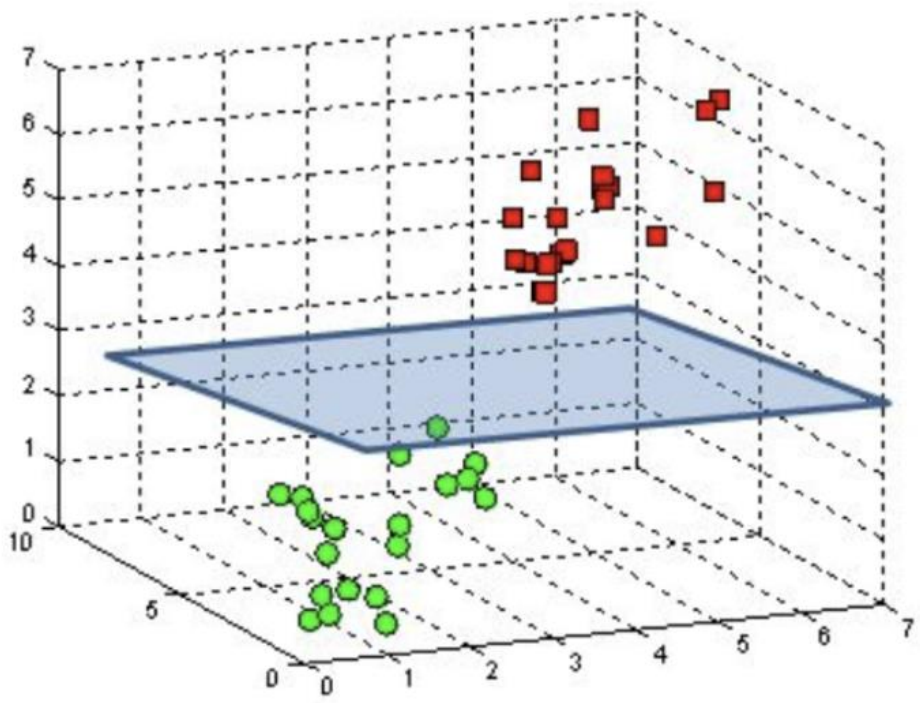


Рис. 2.8. Візуалізація SVM

Нові дані потім вкладаються в той самий простір і передбачають, що вони належать до категорії, на підставі яких була сформована тренувальна вибірка. Крім виконання лінійної класифікації, SVM може ефективно виконувати нелінійну класифікацію, використовуючи те, що називається ”трюком з ядром” (kernel trick), що неявно відображає їхні входи у високорозмірних просторах.

Коли дані не є розміченими, навчання з учителем неможливе, і виникає необхідність у застосування спонтанного навчання, яке намагається відшукати природну кластеризацію даних груп, а потім класифікувати нові дані до цих сформованих груп. Тобто, фактично, вирішується спочатку задача кластеризації одним із методів (метод к-середніх, метод ієрархічного кластерування), а вже потім безпосередньо задача класифікації. Алгоритм векторної кластеризації, застосовує статистику векторів підтримки, розроблених в алгоритмі векторних машин підтримки, для класифікації нерозмічених даних і є одним з найбільш широко використовуваних алгоритмів кластеризації в промислових цілях.

Гіперплощина H1 не є роздільною. Гіперплощина H2 є роздільною, але не з максимальним розділенням. H3 є роздільною гіперплощиною із максимальним розділенням(рис. 2.9.).

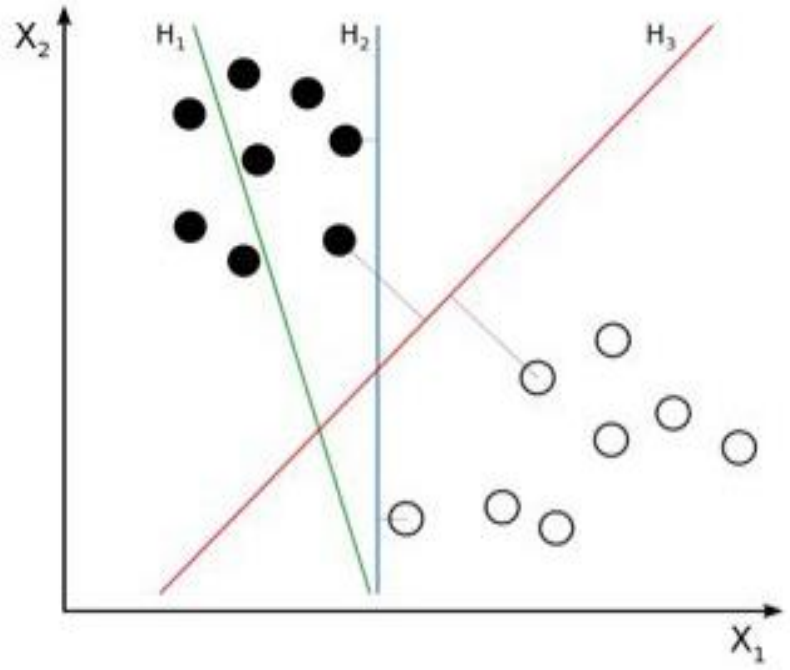


Рис. 2.9. Візуалізація класифікатора

Класифікація Random Forest

Алгоритм собою ансамбль численних незміщених, але чутливих до навчальної вибірки алгоритмів (дерев рішень). Кожен з цих класифікаторів будується на випадковій підмножині об'єктів і ознак.

Навчальна вибірка складається з N прикладів, розмірність простору ознак дорівнює M, і заданий параметр m. Розглянемо покроково алгоритм Random Forest.

Всі дерева ансамблю будуються незалежно один від одного за таким принципом:

1. Згенеруємо випадкову підвибірку з повторенням розміру n з навчальної вибірки.

2. Побудуємо вирішуюче дерево, що класифікує приклади такої підвибірки, причому в ході створення чергового вузла дерева будемо вибирати ознакт, на основі якого проводиться розбиття, не з усіх M ознак, а лише з m випадково обраних.

3. Дерево будується до повного вичерпання підвибірки і не піддається процедурі прунінга (англ. Pruning - відсікання гілок)

Класифікація об'єктів проводиться шляхом голосування: кожне дерево ансамблю відносить клисфікуємий об'єкт до одного з класів, і перемагає клас, за який проголосувало найбільше число дерев.

Алгоритм Random Forest може бути використаний в завданні оцінки важливості ознак. Для цього необхідно навчити алгоритм на вибірці і під час побудови моделі для кожного елемента навчальної вибірки вирахувати out-of-bag-помилку (2.10).

$$OOB = \sum_{i=1}^{l} L(y_i, \frac{1}{\sum_{n=1}^{N}[x_i \not\subset X_n^l]} \sum_{n=1}^{N}[x_i \not\subset X_n^l]\, b_n(x_i)) \qquad (2.10)$$

Де X_n^l – бутстрапована вибірка дерева b_n, L(y,z) – функція втрат, тоді out-of-bag помилка вираховується по формулі:

Для кожного об'єкта така помилка усереднюється по всьому випадковому лісі. Щоб оцінити важливість ознаки, його значення перемішуються для всіх об'єктів навчальної вибірки і out-of-bag-помилка викликається знову. Важливість ознаки оцінюється шляхом усереднення по всіх деревах різниці показників out-of-bag-помилок до і після перемішування значень. При цьому значення таких помилок нормалізуються на стандартне відхилення [18].

Крім того, випадковий ліс має ще деякі переваги для використання в якості алгоритму відбору ознак: він має дуже мало параметрів, що налаштовуються, відносно швидко і ефективно працює, що дозволяє знаходити інформативність ознак без значних обчислювальних витрат.

2.4. Використання машинного навчання для аналізу мережного трафіку

Технології машинного навчання, особливо алгоритми глибокого навчання, серед найпопулярніших технологій для обробки мережного трафіку. Це пояснюється тим, що сучасні комунікаційні системи та мережі, в тому числі Інтернет Речей та мобільні мережі, мають відмінні ознаки, що відповідають алгоритмам глибокого навчання. До цих ознак відносяться:

1. генерація великих даних;
2. складність;
3. мультимодальні дані;
4. масштабованість;
5. велика кількість протоколів у різних інформаційних системах.

Традиційні методи аналізу трафіку не завжди точні та сильно залежать від кваліфікованості спеціаліста. Відносно цього методи машинного навчання мають певні переваги:

1. Моделі глибоко навчання не потребують значних людських зусиль і не залежать від вибору ознак. Моделі ГН можуть використовувати різні репрезентативні шари та ефективні алгоритми для вилучення прихованих знань із величезних обсягів даних трафіку без розробки особливостей. Ця перевага моделей дуже ефективна для методів аналізу мережного трафку, оскільки більшість даних керування мережею є немаркованими або напівмаркованими [19].

2. Моделі глибоко навчання дозволяють обробляти часово-просторові дані, фіксуючи їх ознаки. Зібрані дані, що включають часово-просторові, дозволяють покращити точність алгоритмів глибокого навчання. Розгортання точних і ефективних методів аналізу трафіку є надзвичайно важливим. Точне передбачення мобільного трафіку є важливим для проектування трафіку (розподіл ресурсів за вимогою), економії енергії та аналізу мобільності користувачів у стільникових мережах (прогнозування руху).

3. Реалізація моделей глибокого навчання з вдосконаленням парадигм машинного навчання дозволяє навчати кожу модель окремо на іншому пристрої. Враховуючи, що для аналізу трефіку необхідно зібрати велику кількість обіг інформації мережею з різних пристроїв до головного центру обробки інформації, використовуючи методи розподіленого машинного навчання, моделі глибокого навчання можна навчати окремо на кожній машині, зменшуючи накладні витрати на мережу та загрозу безпеці та конфіденційності.

Класифікація мережного трафіку

Класифікація трафіку дозволяє покращити процес управління мережею. В найширшому значенні класифікація мережного трафіку опирається на систему, в якій програма призначає потоки трафіку джерелам(програми або протоколи), що їх створюють. Крім того класифікація трафіку широко застосовується в QoS цілях, ціноутвореннях інтернет-провайдерів та виявлення аномалій. Кількість пристроїв та додатків постійно зростає, що зумовлює необхідність ефективного аналізу мережного трафіку. Методи аналізу трафіку класифікуються[20] (рис 2.10):

Рис. 2.10. Класифікація трафіку

1. по порту; Методи на основі порту одні з найперших методів класифікації трафіку. Ці методи пов’язують служби/програми із зареєстрованими номерами портів та класифікуйте трафік відповідно до номера порту, що використовується. Перевагами даного методу є: простота в реалізації, але у випадку розгортання нових комунікаційних методів, таких як тунелювання та методи випадкових

призначень портів, ускладнює цей метод та впливає на продуктивність та можливість його застосування.

2. перевірка мережних пакетів; Методи на основі перевірки даних мережних пакетів, особливо, що пов’язані з прикладним рівнем моделі OSI, для того щоб зв’язати пакет з певною службою чи програмою. Щоб спрогнозувати, ці методи зазвичай використовують попередньо визначені підписи або шаблони для кожного протоколу мережі, щоб у подальшому виявляти їх та відрізняти в потоці трафіку один від одного. Відповідно до традиційної мережної парадигми недоліками даного методу є:

- стикаються з труднощами класифікації зашифрованого трафіку;
- політика конфіденційності може обмежувати доступ до даних;
- методи накладають великі обчислювальні витрати на систему.

3. класифікатори на основі потоку; Здатні обробляти будь-який трафік, так як кожна програма та сервіс має унікальні часові та статистичні характеристики. Методи на основі потоку використовують моделі машинного навчання: дерево рішень, логістична регресія та SVM для класифікації трафіку.

Безпека мережі

Будь-яка комунікаційна система чи мережа мають дедалі більший вплив на життя людей та потребує захисту. Кіберзахист забезпечується впровадженням політик, підходів, технологій та процесів, що тісно зв’язані для забезпечення захисту комп’ютерних систем, мереж та даних від атак, спроб несанкціонованого доступу та зловмисних змін. Основні інструменти кіберзахисту: брандмауери, антивірусне програмне забезпечення та IDS. Вони спрямовані на захист комунікаційних систем та мереж від внутрішніх та зовнішніх загроз. Ці інструменти спрямовані на захист комунікаційних систем і мереж від внутрішніх і зовнішніх загроз. IDS є одним із базових механізмів для запобігання атакам і виявлення порушень безпеки в мережах. IDS – це апаратне або програмне забезпечення, яке відстежує мережу та системи на наявність зловмисних дій, атак, порушень політики безпеки. Існує багато видів IDS, варто виділити NIDS –

мережна система виявлення вторгнень та HIDS – хостова система виявлення вторгнень.

NIDS – апаратне чи програмне забезпечення, який використовуються щоб перевіряти та аналізувати мережний трафіку для захисту систем від атак і можливих загроз. NIDS класифікуються за двома основними категоріями: виявлення аномалій та зловмисного використання (рис 2.11). Методи виявлення зловживань, які ще називаються сигнатурними, дозволяють виявляти раніше відомі атаки шляхом зіставлення сигнатур цих атак з проаналізованими даними [22]. В свою чергу методи виявлення аномалій спрямовані на те, щоб відрізнити аномальні шаблони трафіку від нормальних шляхом моніторингу мережних даних.

HIDS – відноситься до систем, які використовують поведінку мережі через різні файли журналу на локальному хост-комп'ютері для виявлення атак. Хостова система виявлення зосереджена на виявленні внутрішніх атак, в першу чергу – виявлення зловмисного програмного забезпечення та ботнетів (рис 2.11).

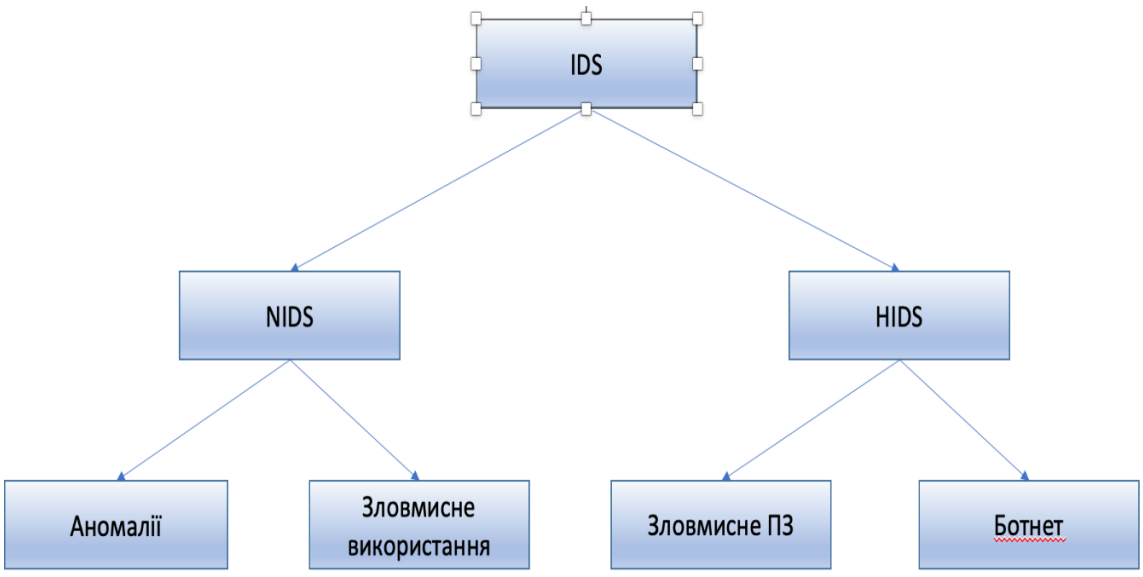


Рис. 2.11. Класифікація IDS

При аналізі досліджень інших авторів щодо застосування технології МН було виявлено, що більша частина досліджень зосереджена на використанні методів глибокого навчання для виявлення вторгнень та аномалій. Дослідивши літературу

інших авторів, щодо створення додатків на основі алгоритмів методів машинного навчання, було класифіковано дані роботи наступним чином: ті, що відносяться до систем NIDS (виявлення (виявлення неправильного використання та виявлення аномалій) та ті, що відносяться до HIDS (виявлення зловмисного програмного забезпечення та виявлення ботнетів).

Застосування технологій машинного навчання набуває поширення в питаннях захисту мережі, особливо в мережних IDS. Насеєр та інші[22] досліджували придатність до застосування моделей глибоко навчання для систем виявлення аномалій. Науковці використали різні технології МН: класичні – K-means, Random Forest, SVM і дерево рішень та глибоко навчання – CNN, RNN, AE. Відповідно до результатів тестування моделі, зазначено, що моделі на основі ГН можуть використовуватись в реальних системах для виявлення аномалій. Автори порівнювали використання класичних та моделей ГН та зробили висновки, що моделі ГН дають більшу точність та ймовірність визначення аномалії, але в свою чергу витрачають більше часу на навчання та тестування ніж класичні. Малая та інші[23] зробили подібні дослідження, як і вищезазначені автори, однак використовували наступні моделі ГН – FCN, VAE та Seq2Seq. Зазначено, що через нелінійність даних мережного трафіку, класичні методи навчання не можуть належно виконувати виявлення аномалій. Наукова робота Юзуфі-Азару та інших[24] висвітляє модель МН – AE, як засіб виявлення аномалій та шкідливих програм для додатків кіберзахисту. Автори використали модель AE для виявлення аномалій і зловмисного програмного забезпечення та досягли багатообіцяючих результатів, оскільки AE є неконтрольованою генеративною моделлю та може вивчати оригінальне приховане представлення даних трафіку. Запропонована модель AE дав 83,34% точності та перевершив своїх конкурентів класичного МЛ, у тому числі Gaussian Naive Bayes = 82,02%, Fuzzy classifier = 82,74% та Дерево Рішень = 80,14%.

Шкідливе програмне забезпечення може значно впливати на величезну кількість користувачів та підключених пристроїв. Оскільки кількість і різноманітність зловмисного програмного забезпечення зростає, методи виявлення

потребують удосконалення для захисту комунікаційних систем і мереж. З цією метою можуть використовуватись системи виявлення ЗПЗ на основі ГН. Як приклад Вей Чжун та Фен Гу[25] запропонували багаторівневу систему на основі ГН, що може легко масштабуватись та обробляти більш складні набори даних. Автори розглядали три системи на основі моделей CNN, LSTM та RNN. Кожна модель відповідає за вивчення конкретного розподілу даних певного класу шкідливих програм. Було порівняно точність виявлення запропоновано системи з іншими на основі CNN і RNN, та SVM з деревом рішень. Результати продемонстрували перевагу наданої системи відносно її аналогів. Харді та інші автори[26] розробили архітектуру на основі моделі SAE. Використовуючи методи ГН, після цього корегуючи результати налаштуванням параметрів, зменшують помилку передбачення алгоритму. Запропонована архітектура забезпечила точність 0.95%, методи Naive Bayes, SVM і дерево рішень – дають такі ж результати по точності, на тренувальних даних але хиблять при виявленні шкідливих програм не в синтетичних умовах.

Враховуючи, що виявлення ботнетів – клас виявлення HIDS, варто розглянути детальніше даний вид атаки на структуру мережі. Ботнет — це мережа приватних комп’ютерних систем, інфікованих шкідливим програмним забезпеченням, що контролюється без відома користувача, наприклад, для здійснення розсилки спаму або інших зловмисних дій[27]. Для виявлення ботнетів використовуються різні підходи, які в основному грунтуються на пасивному моніторингу мережного трафіку. Ці підходи використовують алгоритми МН для категоризації мережного трафіку конструювання та вибору ознак. Що б краще працювати з патернами мережного трафіку і отримувати більш якісні результати роботу алгоритму МН варто використовувати методи ГН, що здатні автоматично виявляти важливі ознаки та ієрархічно доставати їх. Ці переваги створити значний інтерес щодо імплементації алгоритмів ГН для виявлення ботнетів. Рузмелен та інші[28] використовували методи ГН для виявлення потоку ботнет трафіку. Зокрема для цього використовували потоки пакетів TCP/UDP/IP, автоматично

витягували унікальні ознаки використовуючи SDA, після чого для аналізу використовували DNN, досягнувши 99,7% точності.

Таблиця 2.1.

Аналіз робіт кіберзахисту мереж з використанням алгоритмів МН

Автори	Категорія	Модель МН	Ключові тези
Насеєр та інші[22]	Системи виявлення аномалій	CNN, RNN та AE	Проектування та впровадження моделей ГН. Порівняння класичних методів МН
Малая та інші[23]	Виявлення аномалій	FCN, VAE та Seq2Seq	Аналіз кількох методів ГН.
Юзуфі-Азару та інші[24]	Виявлення аномалій та зловмисного використання	Автоенкодери	Використання алгоритмів МН без вчителя на основі AE.
Вей Чжун та Фен Гу[25]	Системи виявлення ЗПЗ	CNN, RNN і LSTM	Впровадження багаторівнених моделей ГН для виявлення ЗПЗ.
Харді та інші[26]	Виявлення ЗПЗ	SAE	Впровадження фреймворку для виявлення ЗПЗ.
Рузмелен та інші[28]	Виявлення ботнетів	SDA та DNN	Реалізація додатку, використовуючи методи ГН, для виявлення ботнетів, що сканує потоки TCP/UDP/IP пакетів

Висновок.

Отже, технології машинного навчання дозволяють обробляти великий об’ємом даних, та вирішувати велику кількість нетрадиційних завдань за короткий проміжок часу.

За допомогою проаналізованих алгоритмів та розглянутих методів машинного навчання можна прогнозувати значення, виявляти незвичайні сценарії та закономірності, визначати структуру або створювати категорії.

Автори наукових досліджень зазначають перевагу використання методів глибоко навчання для реалізації систем виявлення вторгнень. Необхідність класичних методів у більшості випадків зазначена лише для порівняння точності моделей досліджень. Системи виявлення вторгнень можуть бути легко імплементовані для виявлення сканувань та виконують функції захисту від ШПЗ, ботнетів, аномалій та зловмисного використання.

Розглянуті класичні алгоритми машинного навчання можуть бути використані для розробки системи виявлення сканувань.

3 РОЗРОБЛЕННЯ ВАРІАНТУ ВИЯВЛЕННЯ СКАНУВАННЯ МЕРЕЖІ НА БАЗІ МЕТОДІВ МАШИННОГО НАВЧАННЯ

3.1. Розроблення алгоритму виявлення сканування на базі методів машинного навчання

Реалізація програмного додатку для виявлення процесу сканування передбачає підтримку кросплатформеності, простоту взаємодії процесів та впровадження технології машинного навчання. Для написання програмному модуля була обрана мова програмування Python версія інтерпретатору – 3.8, оскільки не всі бібліотеки сумісні з останньою верією інтерпретатору.

Python – високорівнева інтерпретована багатоцільова мова програмування, яка дозволяє писати код, що легко читається. Відносний лаконізм мови дозволяє створити програму, яка буде набагато коротше свого аналога, написаного на іншій мові. Програми на Python можна запускати в різних операційних системах без будь-яких змін. Стандартна бібліотека, яка встановлюється разом з Python містить готові інструменти для роботи з операційною системою, веб-сторінками, базами даних, різними форматами даних, для побудови графічного інтерфейсу програм. Програми, написані на цій мові програмування можуть бути як невеликими скриптами, так і складними системами[29].

Переваги Python для розробки програмного забезпечення з машинним навчанням:

1. Простий синтаксис, що дозволяє зосередитись на вирішення завдання, а не написанні коду;
2. Висока швидкість компіляції коду при обробці великих масивів даних;
3. Великий вибір бібліотек та фреймворків, що пришвидшують написання програмного забезпечення з машинним навчанням.

Життєвий цикл будь-якої моделі машинного навчання (рис 3.1) починається з аналізу завдання. Відповідно до попередньо проаналізованих робіт різних авторів, попередньо були зроблені наступні висновки:

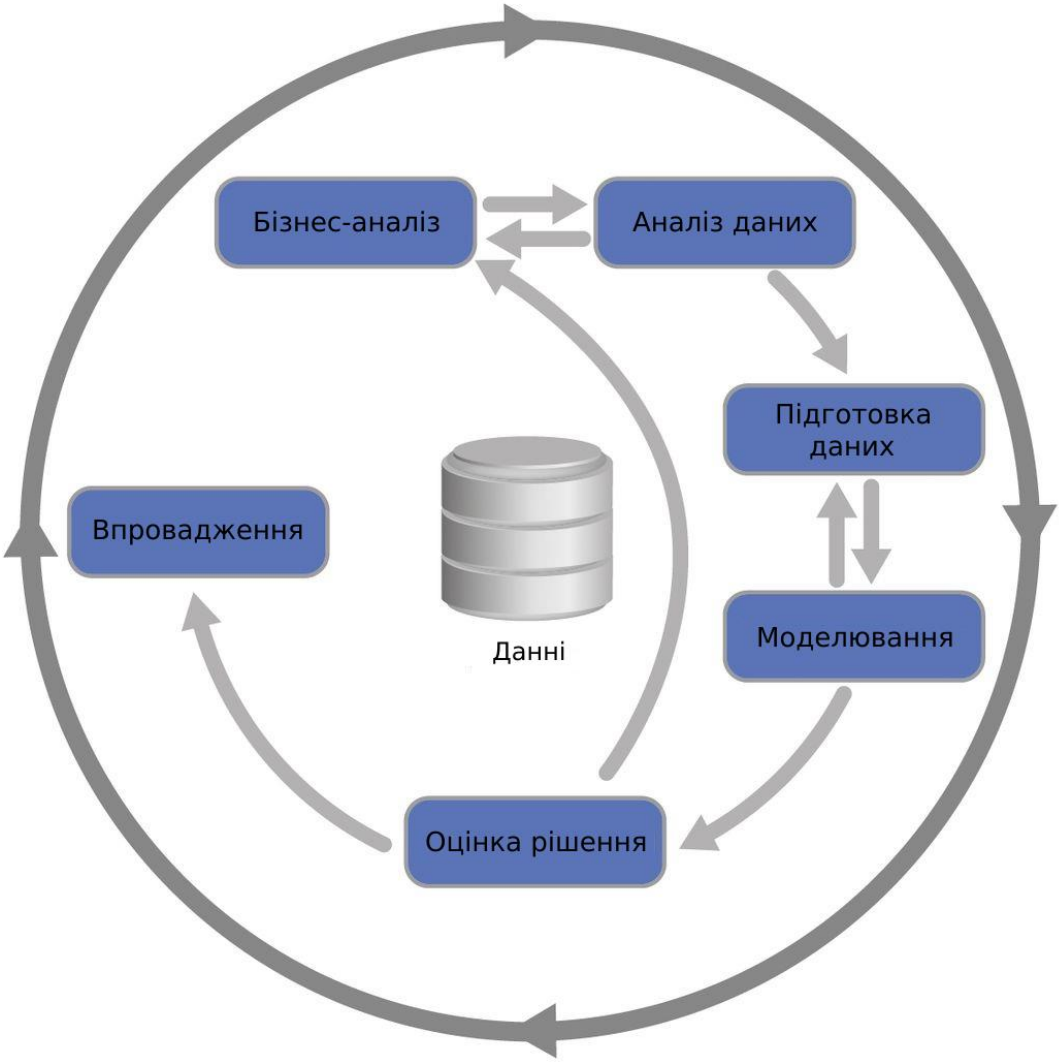


Рис. 3.1. Життєвий цикл моделі МН

1. Велика кількість наукових досліджень проводиться на синтетичних даних;

2. Імплементація класичних моделей МН впроваджуються в більшості випадків не як рішення завдання, а для множинного порівняння з іншими моделями;

3. Ототожнення аналізу результатів алгоритмів у локальній(LAN) та глобальній(Інтернет) мережі.

Аналіз завдання

Реалізація системи виявлення сканувань потребує велику кількість даних для навчання класичної моделі МН та аналіз результатів, щодо вибору алгоритму,

котрий буде впроваджено для тестування при аналізі трафіку у мережі. Метою завдання є: реалізація додатку виявлення сканувань.

Аналіз даних

Дизбаланс даних, необхідних для навчання моделей МН, може призвести до зниження ефективності детектування сканування мережі. У випадку недостатньої кількості даних зловмисного трафіку, можлива велика похибка точності моделей МН. Збалансованість набору даних перевірятиметься на двох наборах даних, що цикрулюють у локальній(LAN) та глобальній(Інтернет) мережі. Для реалізації завдання необхідно два набори даних:

1. набір даних без зловмисного трафіку;
2. набір даних зловмисного трафіку.

Для аналізу даних зловмисного трафіку використаний набір даних з наукової статі [30], інфраструктура згенерованих даних створюється через хмарні екземпляри, географічно розподілені по світу та представлені на рис 3.2.

Рис. 3.2. Хости для генерування набору даних

Підготовка даних

Щоб згенерувати даний набір даних, автор використовував наступний підхід: для кожної атаки, автор спочатку надсилає UDP пакет, щоб запустити tcpdump. Після цього виконується сканування портів, а потім через UDP надсилається повідомлення про зупинку. Дану комункікацію зображено на рис 3.3. Цей датасет містить 455 503 пакети, наведених у таблиці.

Таблиця 3.1.

Набір даних зловмисного трафіку

Тип сканування	Кількість пакетів
zmap	74613
nmap_connect	45882
hping_syn	43750
unicorn_syn	43039
nmap_syn	40642
unicorn_conn	39170
masscan	21138
nmap_ack	20497
nmap_window	18851
nmap_null	12511
nmap_xmas	12505
nmap_fin	12504
nmap_ack	11344
nmap_maimon	10493
unicorn_ack	8494
hping_xmas	7344
hping_null	7344
unicorn_null	4690
unicorn_xmas	4466
unicorn_fxmas	4444
unicorn_fin	4438

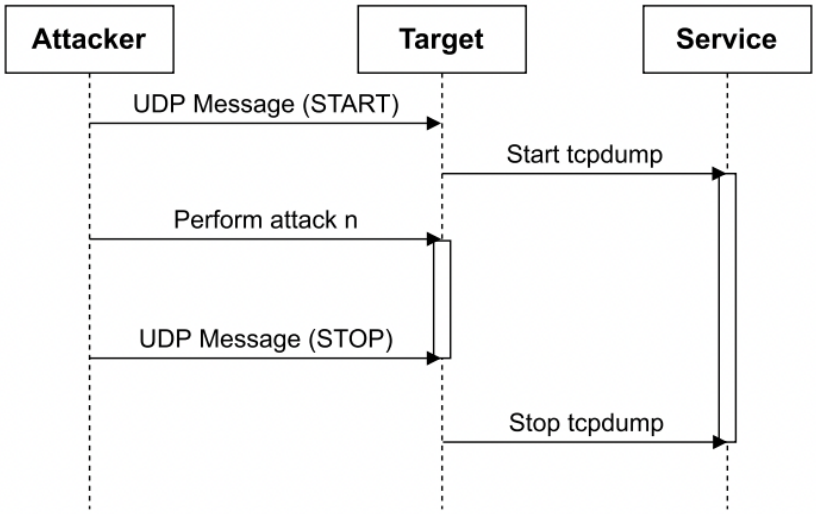


Рис. 3.3. Генерування даних

Для аналізу даних без зловмисного трафіку в LAN мережі використано утиліту iPerf3, що б згенерувати TCP/IP трафік протягом 20 хвилин. Для генерації трафіку в глобальній мережі було використано попереднього набору даних та з MAWLAB[31]. Після поділу, вибірки та фільтрації лише трафіку TCP/IP отримано 328 261 пакет. Загальна кількість даних на рис 3.4.

```
1    455503
0    328261
Name: label, dtype: int64
```

Рис. 3.3. Загальна кількість даних для тренування моделі НМ

Моделювання

Для навчання моделей використані наступні класичні алгоритми МН:

1. k-Nearest Neighbours – (kNN)
2. Random Forest – (RF)
3. Naive Bayes – (NB)
4. Support Vector Machine – (SVM)
5. Logistic Regression – (LR)
6. Decision Tree – (DT)

В результаті очищення вхідних наборів даних, отримали наступну кількість пакетів представлених у таблиці 3.2:

Таблиця 3.2.

Набори даних

Набір даних	Загальна кількість	Зловмисні у %	Без зловмисних у %
Локальний (LAN)	113759	20%	80%
Глобальний (Інтернет)	677879	54%	46%

Наявні набори даних містять схожу вибірку, але різні за кількістю та відсотковим співвідношенням. Це необхідно для тестування алгоритмів машинного навчання. У випадку перетренування моделі на одному наборі даних, зможемо визначити точність алгоритму на іншому. Для тренування моделей машинного навчання використано мову програмування Python та бібліотеку scikit-learn для підрахування точності детектування TCP пакетів. Вибірку навчання після обробки розбили на дві групи, де:

«0» – пакети без зловмисного трафіку;

«1» – пакети зловмисного трафіку.

В даному випадку було спрощено класифікацію до бінарної. Для аналізу результатів навчання моделі використовували показники F1 та співвідношення ROC/AUC.

Показник F1 – середнє співвідношення точності до відклику, та вираховується за формулою 3.1:

$$F1 = 2\times \frac{\text{Точність}\times\text{Відклик}}{\text{Точність}+\text{Відклик}} \qquad (3.1)$$

Точність вираховується за формулою 3.2:

$$\text{Точність} = \frac{\text{True Positive}}{\text{True Positive+False Positive}} \qquad (3.2)$$

Відклик вираховується за формулою 3.3:

$$\text{Точність} = \frac{\text{True Positive}}{\text{True Positive+False Negative}} \qquad (3.2)$$

З наборів даних видалено всі ознаки, які не застосовуються, надлишкові та неваріативні. Оскільки до перед обробки, набори даних були з pcap файлу, потрібно було прибрати низькорівневі протоколи, які підходили для тренувань та тестувань. Після чого видаляєм з IP заголовку версію та протокол, а також поля джерела і призначення IP адреси для узагальнення навчання набору даних. Крім того було видалено наступні поля: довжина заголовку, тип служби, зміщення фрагменту, тип служби та зарезервовані бітові фрагменти.

Усі алгоритми перед тренуванням були визначені в стратифікованій k-вибірці, де k=10.

Тренування та тестування алгоритмів буде відбуватись на пристрої, характеристики якого наведено у таблиці 3.3.

Таблиця 3.3.

Характеристика пристрою

Компонент	Опис
CPU	1xIntel(R) Xeon(R) CPU E5-2690v3
RAM	2GB DDR4
Тип підключення	SSH
Мережна карта	Gigabit Ethernet
Ядро Linux	6.0.16
Дистриб’ютор	Ubuntu 20.04

3.2. Реалізація алгоритму виявлення сканування мережі на базі методів машинного навчання

Після тренування моделей машинного навчання на локальному (LAN) наборі даних отримали наступні результати (рис 3.4)

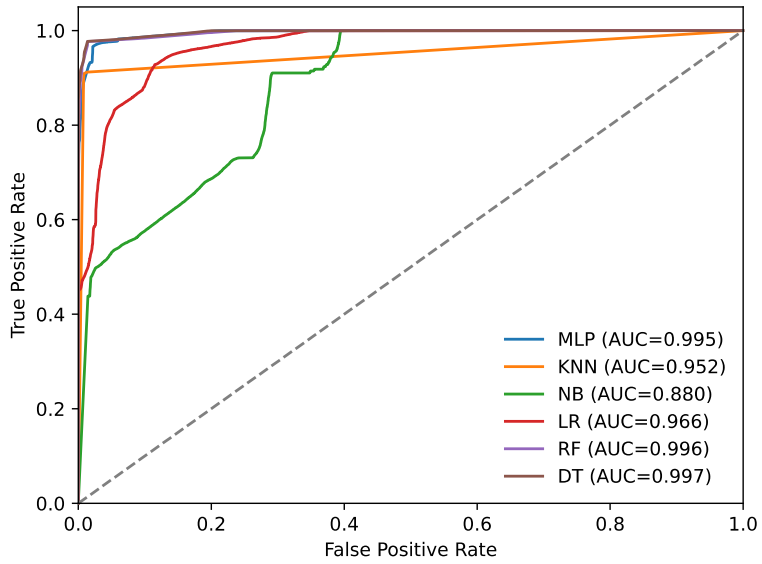


Рис 3.4. Співвідношення ROC/AUC у LAN наборі даних

Після тренування моделей машинного навчання на глобальному (Інтернет) наборі даних отримали наступні результати (рис 3.5)

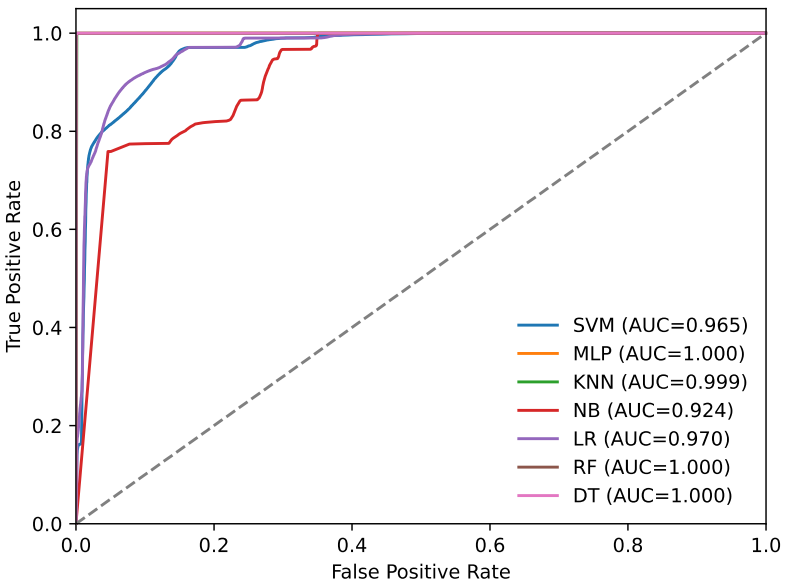


Рис 3.5. Співвідношення ROC/AUC у Інтернет наборі даних

Відповідно до загального співвідношення True Positive Rate до False Positive Rate у локальному наборі даних важливо зазначити (рис 3.6), що деякі з моделей класичного машинного навчання після тренувань врешті-решт було перетреновані, що робить неможливим використанням певних алгоритмів в зв’язку з тим, що дані моделі «перенавчені» та подальша їх імплементація може збільшити помилку виявлення системи.

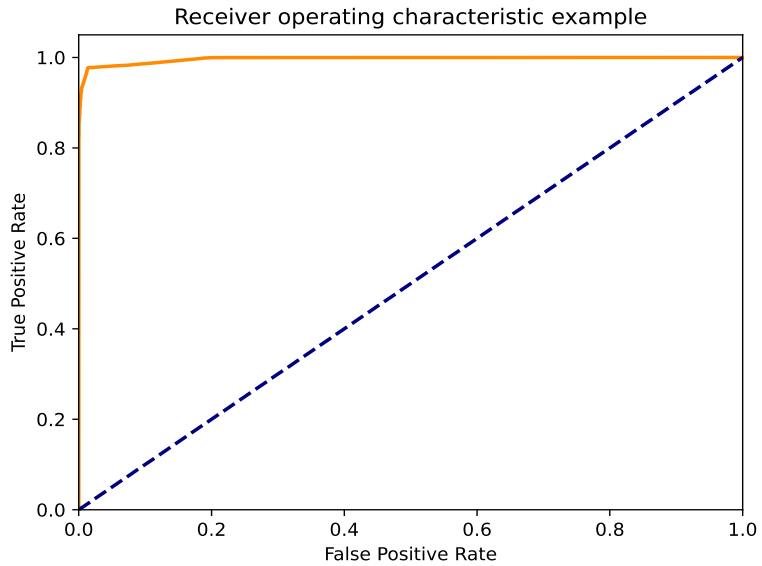


Рис 3.6. Загальне співвідношення True Positive Rate до False Positive Rate у Інтернет наборі даних

На рис 3.7 зображено виконання grid-search. Відносно цього графіку можна відстежити значення оптимальних параметрів алгоритму. Як зазначено на рисунку, обведеній в точці відбулось перетренування алгоритму. Відповідно подальше використання може призвести до збільшення кількості неправильних спрацювань. Бібліотека scikit-learn, дозволяє зберегти найкращі параметри моделі та відкинути ті, що були перетреновані. Значення F1 в даному випадку приблизно рівне 0.96.

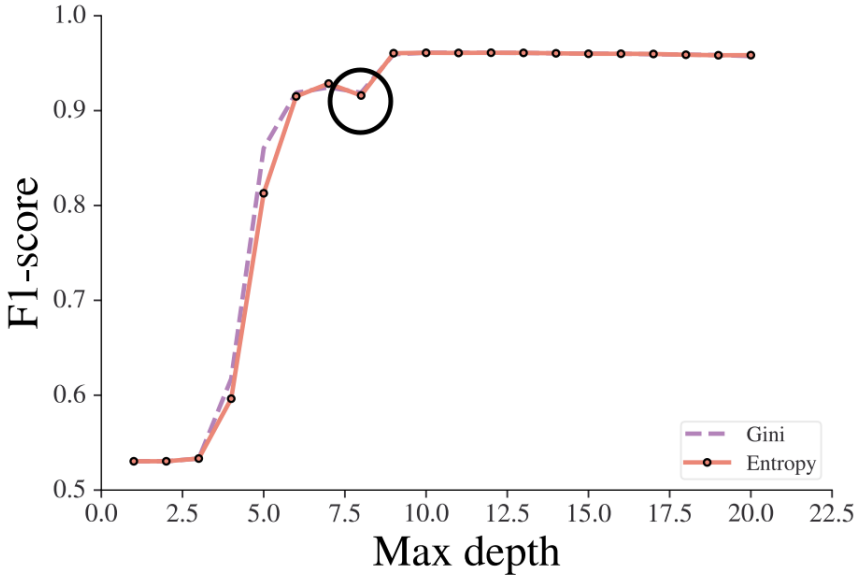


Рис 3.7. Виконання grid-search

Реалізована система виявлення вторгнень буде взаємодіяти з утилітою iptables використовуючи модуль Netfilter NFQUEUE, однойменна бібліотека підтримується в Python та може бути легко імплементована в додаток для виявлення вторгнень. В даному алгоритмі реалізації (рис 3.8) всі пакети надходять з простору ядра до користувача, а потім розпізнаються алгоритмом машинного навчання заданим при ініціалізації додатку.

Розгортання системи виявлення вторгнень у цьому випадку передбачає перетворення моделей машинного навчання на «фінальні моделі», концепція з вибором найкращих моделей по grid-search. Кожна модель – це набір байтів, котрий викликається в Python додатку і розпізнає пакети, що передаються з простору користувача. Розгортання системи передбачає наявність привілейованих прав для взаємодії з Netfilter NFQUEUE. Реалізація програми наявна у додатку роботи.

Для оцінки ефективності додатку на пристрої, характеристики якого наведено у таблиці 3.3 необхідно встановити правила iptables з Netfilter NFQUEUE та розгорнути систему виявлення сканувань з певним алгоритмом на пристрої. Після чого дані продуктивності та інші метрики будуть зібрані за допомогою утиліти sysstat. Для кожної моделі протягом 20-хвилинного тесту були виміряні

наступні показники: використання CPU та RAM. Дані графіки зображені на рис. 3.9 та 3.10.

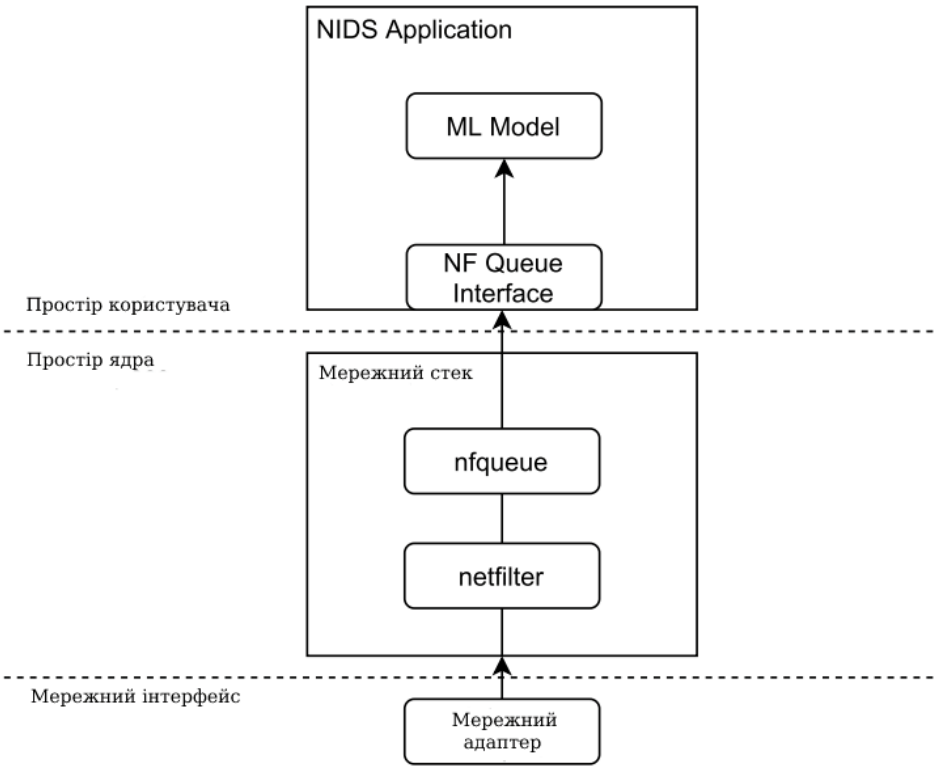


Рис 3.8. Алгоритм реалізації системи виявлення вторгнень

З графіку (рис 3.9) використання CPU, найбільш ресурсозатратними алгоритмами є: RF та KNN.

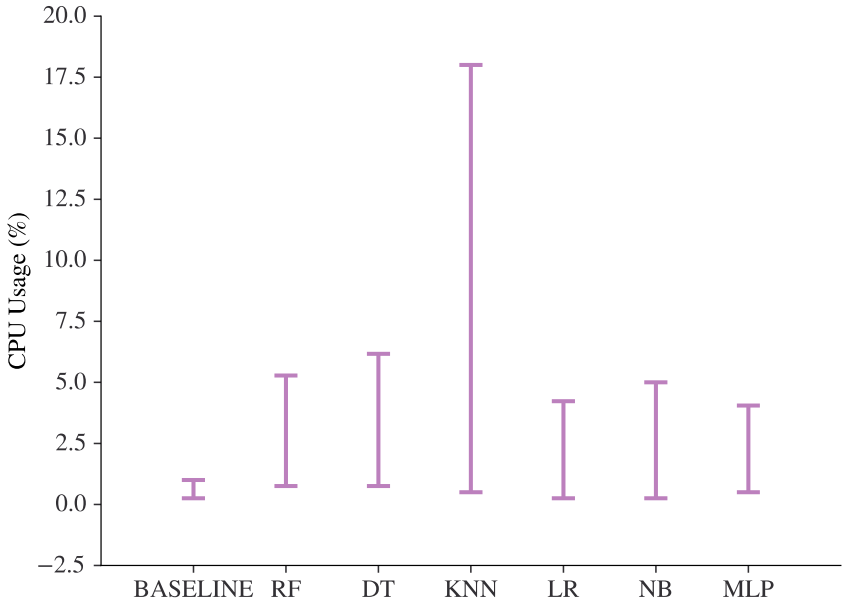


Рис 3.9. Використання CPU

З графіку (рис 3.10.) використання RAM, найбільш ресурсозатратними є: DT та KNN. Крім цього введемо ще одну метрику – час на розпізнавання. Для цього необхідно було отримати з робочого циклу, що складається зі створення пакетів із випадковими атрибутами (20 разів) разом з бібліотекою timeit Python для усереднення 100 висновків виконання розпізнавання випадкових пакетів. Для пришвидшення обробки даних моделі було протестовані на Apple M2 процесорі. В таблиці 3.4 наведено тестування часу на розпізнавання.

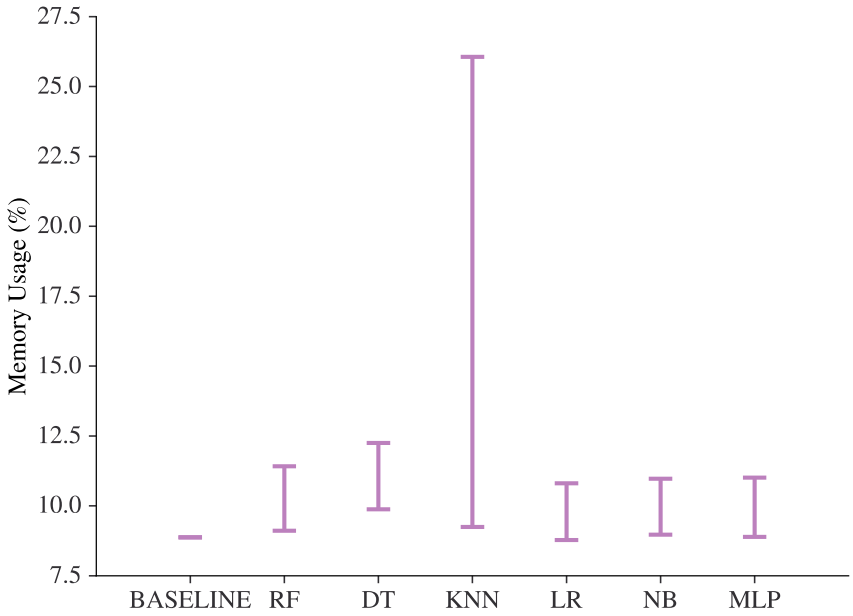


Рис 3.10. Використання RAM

Таблиця 3.4.

Середній час на розпізнавання

Алгоритм	Час на розпізнавання у мілісекундах
KNN	7.13
DT	0.33
LR	0.32
MLP	0.33
NB	0.34
RF	1.8

Відповідно до значень наведених у таблиці 3.4, необхідно зазначити, що алгоритми RF та KNN є досить повільними, при використанні більш слабких процесорів це може призвести до переповнення буферу та розпізнавання пакетів буде займати великий об’єм часу, що унеможливить використання системи виявлення вторгнень. В свою чергу більш прості моделі такі як: MLP, DT, LR з легкістю зможуть обробити великий обсяг даних та спрогнозувати результат. Це варто враховувати при використанні даної системи.

Рекомендації щодо розробки системи виявлення сканувань на базі методів машинного навчання

Розглянемо основні рекомендації щодо розробки системи виявлення сканувань:

- необхідно чітко визначити поставлене завдання та необхідність використання алгоритмів машинного навчання;
- набори даних без зловмисного трафіку можуть бути використані з відкритих джерел;
- необхідно великі варіативні набори даних для тестувань та тренувань алгоритмів машинного навчання;
- варто впроваджувати підтримку всіх мережних протоколів для збільшення кількості успішного виявлення загроз;
- необхідно відслідковувати процес тренування для аналізу стану навчання моделі машинного навчання;
- деякі з ознак мережних пакетів необхідно знеособити для стабільної роботи моделі машинного навчання у справжньому середовищі;
- необхідно відстежувати середній час на розпізнавання алгоритмів для уникнення переповнення буферу;
- високе навантаження використання CPU та RAM може призвести до неможливості розгортання системи виявлення сканувань на пристрої;
- будь-яку готову систему виявлення сканувань необхідно протестувати у справжньому середовищі перед розгортанням;

- методи глибоко навчання можуть успішно використовуватись для аналізу трафіку.

Висновок.

Отже, розроблена система виявлення сканувань може бути впроваджена та виконувати свої функції. З переваг використання варто зазначити швидкість розгортання на пристроях, що підтримують Linux та незначне використання GPU та RAM більшості алгоритмів машинного навчання.

Система виявлення вторгнень була протестована на двох наборах даних, що в свою чергу забезпечує її використання як в локальній так і в глобальних мережах. Точність алгоритмів на синтетичних даних була більше ніж 90%, що в свою чергу дає можливість до тестування та впровадження її для аналізу виявлення сканувань.

В подальшому можливе використання цієї системи на пристроях, що мають невеликий об’єм CPU та RAM. У випадку необхідності, можливе легке впровадження нових моделей машинного навчання та перенавчання наявних.

Наступним етапом розвитку системи можливе впровадження алгоритмів глибоко навчання для збільшення об’єму аналізу даних та більш точне розпізнавання процесу сканувань.

ВИСНОВКИ

В роботі проведено аналіз проблеми виявлення сканувань на базі методів машинного навчання, визначено його мету та завдання.

1. Проведено аналіз існуючих технологій сканування мережі. Сканування служб і сервісів, запущених на комп'ютерах мережі є першим етапом підготовки і проведення атаки на комп'ютерні системи. Для цих цілей зловмисники використовують мережні сканери. Для виявлення сканувань використовують брандмауери, журнали сервісів та системи виявлення вторгнень.

2. Досліджено методи та засоби сканування структури мережі. Nmap – вважається кращим сканером, що використовує різні методи сканування за допомогою флагів або надсилання корисного навантаження. Крім того Nmap підтримує використання скриптів з метою автоматизації процесу сканування та пошуку вразливостей.

3. Визначено методи, основні функції та алгоритми застосування машинного навчання. Технології машинного навчання можуть прогнозувати значення, виявляти незвичайні сценарії та закономірності, визначати структуру або створювати категорії.

4. Розглянуто способи використання методів глибоко навчання для реалізації систем виявлення вторгнень. Необхідність класичних методів у більшості випадків зазначена лише для порівняння точності моделей досліджень. Системи виявлення вторгнень можуть бути легко імплементовані для виявлення сканувань та виконують функції захисту від ШПЗ, ботнетів, аномалій та зловмисного використання.

5. Розроблено варіант системи виявлення сканувань на базі методів машинного навчання з використанням класичних алгоритмів.

6. Розроблено рекомендації щодо розробки системи виявлення сканувань на базі методів машинного навчання.

Таким чином, правильна системи виявлення сканувань має протидіяти вторгненням до мережі та виявляти в режимі реального часу процес сканування з метою запобігання можливих кіберінцидентів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Полікарпов Е. С. КОМПЬЮТЕРНАЯ РАЗВЕДКА / Е. С. Полікарпов.. – 198 с.

2. Пахомова А.С., Пахомов А.П., Юрасов В.Г., Об использовании классификации известных компьютерных атак в интересах разработки структурной модели компьютерной разведки/ А.С. Пахомова, О.А. Оступенко // Информация и безопасность. - 2013. - С. 115-118.

3. Протокол обмена управляющими сообщениями ICMP [Електронний ресурс] – Режим доступу до ресурсу: http://citforum.ck.ua/nets/ip/glava_7.shtml.

4. СТЕК ПРОТОКОЛІВ TCP/IP [Електронний ресурс] – Режим доступу до ресурсу: http://wad00m.narod.ru/index/0-11.

5. Бирюков А.А. Информационная безопасность. Защита и нападение. – 2-е изд., перераб. и доп. – М.: ДМК Пресс, 2017. – 434 с.

6. Beecroft A. J. PASSIVE FINGERPRINTING OF COMPUTER NETWORK RECONNAISSANCE TOOLS [Електронний ресурс] / Beecroft // MONTEREY, CALIFORNIA. – 2009. – Режим доступу до ресурсу: https://apps.dtic.mil/dtic/tr/fulltext/u2/a509167.pdf.

7. Introduction to Nessus [Електронний ресурс] – Режим доступу до ресурсу: http://apachepersonal.miun.se/~janjon/oldcourse/dtab80/lab/lab2/nessus_1.pdf.

8. Руководство по работе со скриптами Nmap Scripting Engine [Електронний ресурс] – Режим доступу до ресурсу: https://networkguru.ru/nmap-scripting-engine-rukovodstvo/.

9. Справочное руководство Nmap (Man Page) [Електронний ресурс] – Режим доступу до ресурсу: https://nmap.org/man/ru/index.html.

10. Машинное обучение (machine learning): основные методы, нейронные сети [Електронний ресурс] – Режим доступу до ресурсу: https://www.it.ua/ru/knowledge-base/technology-innovation/machine-learning.

11. Что такое машинное обучение | Microsoft Azure [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/ru-ru/overview/what-is-machine-learning-platform/#uses.

12. Адаменко В. О. Штучні нейронні мережі в задачах реалізації матеріальних об'єктів [Електронний ресурс] / В. О. Адаменко – Режим доступу до ресурсу: https://kivra.kpi.ua/wp-content/uploads/file/science/adamenko/134-210-1-SM.pdf.

13. Гафаров Ф. М. ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ И ИХ ПРИЛОЖЕНИЯ [Електронний ресурс] / Ф. М. Гафаров, А. Ф. Галимянов – Режим доступу до ресурсу: https://kpfu.ru/staff_files/F1493580427/NejronGafGal.pdf.

14. Алгоритм K-Means [Електронний ресурс] – Режим доступу до ресурсу: https://algowiki-project.org/ru/Алгоритм_k_средних_(k-means).

15. Gaussian Mixture Model [Електронний ресурс] – Режим доступу до ресурсу: https://brilliant.org/wiki/gaussian-mixture-model/.

16. ML | BIRCH Clustering [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/ml-birch-clustering/.

17. Support Vector Machines [Електронний ресурс] – Режим доступу до ресурсу: https://scikit-learn.org/stable/modules/svm.html.

18. Understanding Random Forest [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

19. Mobile big data analytics using deep learning and apache spark / M. A. Alsheikh та ін. IEEE Netw.2016. Т. 30, № 3. С. 22–29.

20. Dainotti A., Pescape A., C. Claffy K. Issues and future directions in traffic classification. IEEE. 2012. Т. 26, № 1. С. 35–40.

21. Acha K., Modi C. N. Virtualization layer security challenges and intrusion detection/prevention systems in cloud computing: a comprehensive review. The Journal of Supercomputing. 2017. № 73. С. 1192–1234.

22. Muhammad Munwar Iqbal, Kijun Han, Enhanced network anomaly detection based on deep neural networks / S. Naseer та ін. IEEE Access. 2018. Т. 6.

23. An empirical evaluation of deep learning for network anomaly detection / R. K. Malaiya та ін. International Conference on Computing, Networking and Communications (ICNC). 2018. IEEE. С. 893–898.

24. Autoencoder-based feature learning for cyber security applications / M. Yousefi-Azar та ін. International Joint Conference on Neural Networks (IJCNN). 2017. IEEE. С. 3854–3861.

25. Zhong W., Gu F. A multi-level deep learning system for malware detection. Expert Syst.2019. Appl., № 133. С. 151–162.

26. DL4MD: A deep learning framework for intelligent malware detection / W. Hardy та ін. Proceedings of the International Conference on Data Mining (DMIN). 2016. The Steering Committee of The World Congress in Computer Science. С. 61.

27. Савенко О., Лисенко С., Крищук А. Модель ботнет мереж. Вісник Вінницького політехнічного інституту. 2013. Т. 3. С. 76–81.

28. Roosmalen J. v., Vranken H., Eekelen M. v. , Applying deep learning on packet flows for botnet detection. Proceedings of the 33rd Annual ACM Symposium on Applied Computing. 2018. С. 1629–1636.

29. Путівник мовою програмування Python [Електронний ресурс] – Режим доступу до ресурсу: https://pythonguide.rozh2sch.org.ua.

30. An End-to-End Framework for Machine Learning-Based Network Intrusion Detection System / G. D. C. BERTOLI та ін. IEEE Access. 2021.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ
КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ

Магістерська робота
на тему:

«ТЕХНОЛОГІЯ ВИЯВЛЕННЯ СКАНУВАННЯ МЕРЕЖІ НА БАЗІ МЕТОДІВ МАШИННОГО НАВЧАННЯ»

Виконав: КУЗЬМІНСЬКИЙ Віктор Сергійович

Керівник: ГАЙДУР Галина Іванівна,
к.т.н., доцент

2

Об'єкт дослідження – процес виявлення сканування мережі на базі методів машинного навчання

Предмет дослідження – технологія виявлення процесу сканування мережі на базі методів машинного навчання

Мета роботи – розробити варіант системи виявлення процесу сканувань та рекомендації щодо застосування технології машинного навчання

Наукові завдання:

- дослідити сутність проблеми сканування мережних хостів;
- проаналізувати існуючі технології сканування комп'ютерної мережі;
- встановити сутність технології машинного навчання;
- проаналізувати методи та алгоритми машинного навчання;
- проаналізувати способи використання машинного навчання для аналізу мережного трафіку.

Дослідження проблеми сканування мережних хостів

3

Розвиток та поширення обчислювальної техніки передбачає ведення розвідки в інформаційному просторі. Класифікаційна ознака відносно нового виду технічної розвідки – **комп'ютерной розвідки** – спосіб добування інформації.
Комп'ютерна розвідка – це добування інформації з комп'ютерних систем та мереж, характеристик їх програмно-апаратних засобів і користувачів.

За масштабом комп'ютерна розвідка поділяється на інтернет-розвідку, мережну розвідку та цільову розвідку:

Мережна розвідка спрямована на добування даних з комп'ютерних мереж, шляхом реалізації зондування мережі, ідентифікації віддалених служб і аналізу вразливостей ресурсів мережі. Перше що потрібно виконати це – ідентифікувати працюючі хости. Після чого необхідно визначити, які служби та додатки запущені на віддалених хостах мережі.

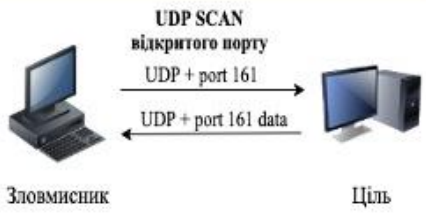
Аналіз підходів технології сканування комп'ютерної мережі

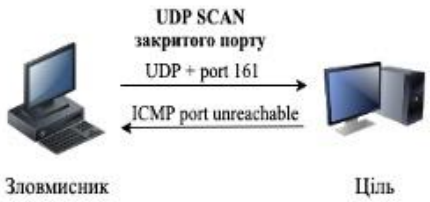
4

Програмні засоби сканування структури та елементів мережі під час своєї роботи використовують два основних механізми. **Зондування** – механізм активного аналізу, що імітує атаку, тим самим перевіряючи вразливості. Цей механізм працює повільно, але дозволяє підтвердити наявність вразливості і виявити «дірки» у захисті структури та елементів мережі. **Сканування** – механізм пасивного аналізу, що значно швидше зондування, але дає менш точні результати.
Утиліта Nmap в процесі сканування мережі перебирає доступний діапазон портів і намагається підключитися до кожного з них, приклади деяких з видів сканувань показано на рисунках.

Більшість сучасних сканерів безпеки мережі мають такі функції:

1. збір інформації про мережу, ідентифікація всіх активних пристроїв і сервісів, запущених на них;
2. виявлення потенційних вразливостей;
3. підтвердження знайдених вразливостей, для чого використовуються специфічні методи і моделюються атаки;
4. створення звітів;
5. автоматичне усунення вразливостей.

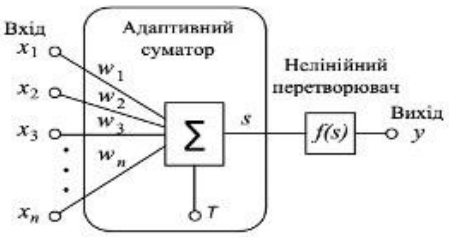




Дослідження технології машинного навчання

5

Машинне навчання – це використання математичних моделей даних, які допомагають комп'ютеру навчатися без безпосередніх інструкцій. Вважається однією з форм штучного інтелекту. За допомогою алгоритмів машинного навчання виявляються закономірності в даних. На основі цих закономірностей створюється модель даних для прогнозування.



Машинне навчання має широке застосування на практиці, а не тільки аналітичний і теоретичний зміст. Перевірка різних методів і алгоритмів на реальних даних досягається за рахунок емпіричного досвіду. Для досягнення ефективної роботи запропонованого рішення, застосовуються додаткові методи, які відображають помилки прогнозованих значень від реальних даних.

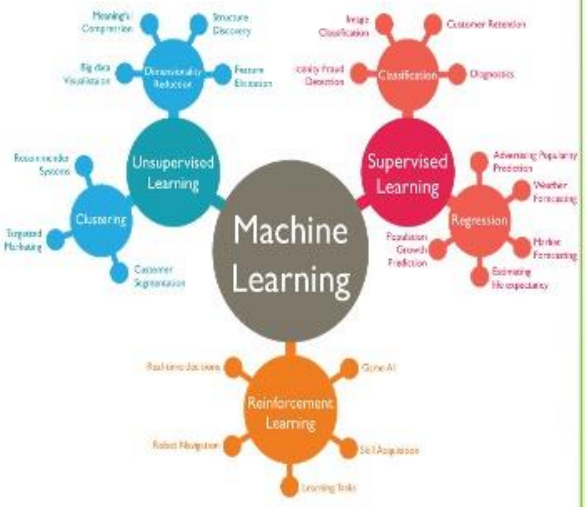
Базується на ідеї про те, що аналітичні системи можуть вчитися виявляти закономірності і приймати рішення з мінімальним втручанням людини.

Штучний нейрон складається з двох частин — адаптивного суматора, який сумує вхідні сигнали та нелінійного перетворювача, який ще називають функцією активації. Навчання такого нейрону в більшості випадків зводиться до корекції вагових коефіцієнтів, так як одночасна корекція вагових коефіцієнтів і параметрів функції активації призводить до різкого зростання часу навчання. компонент штучного нейрона – функція активації. В теорії побудови нейронних мереж застосовують велику кількість функцій активації.

Методи та алгоритми машинного навчання

6

Нейронні мережі, на відміну від статистичних методів багатовимірного класифікаційного аналізу, базуються на паралельній обробці інформації і мають здатність до самонавчання, тобто отримувати обгрунтований результат на підставі даних, які не зустрічалися в процесі навчання.



Нейронні мережі навчають на прикладах. Розробник нейронної мережі підбирає представницькі дані, а потім запускає алгоритм навчання, який автоматично сприймає структуру даних. При цьому від розробника потрібно набір евристичних знань про те, як слід відбирати і готувати дані, вибирати потрібну архітектуру мережі та інтерпретувати результати.

За методами навчання нейронні мережі класифікуються:

1. навчання з вчителем, або контрольоване навчання;
2. напівавтоматичне навчання або часткове навчання;
3. навчання без вчителя;
4. навчання з підкріпленням.
5. глибоке навчання

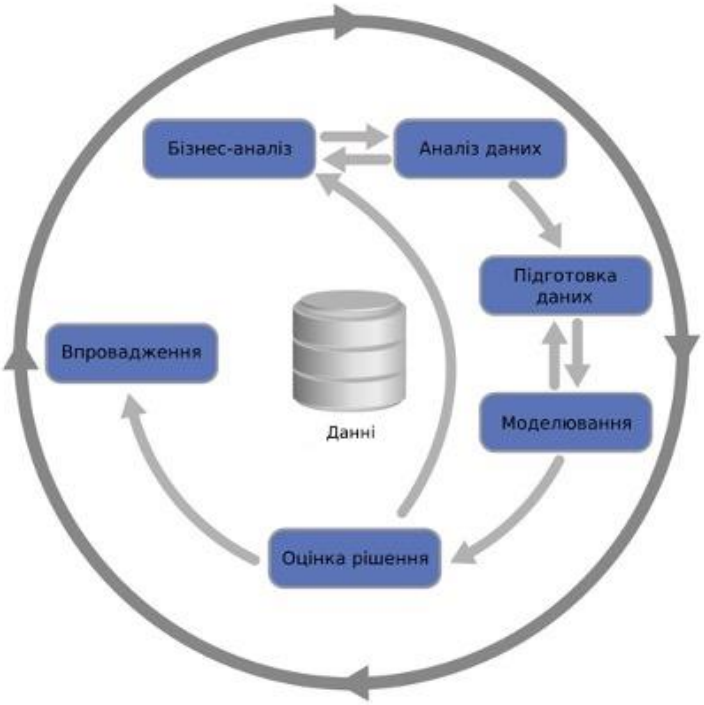
Способи використання машинного навчання для аналізу мережного трафіку

6

Автори	Категорія	Модель МН	Ключові тези
Насеєр та інші	Системи виявлення аномалій	CNN, RNN та AE	Проектування та впровадження моделей ГН.
Малая та інші	Виявлення аномалій	FCN, VAE та Seq2Seq	Аналіз кількох методів ГН.
Юзуфі-Азару та інші	Виявлення аномалій та зловмисного використання	Автоенкодери	Використання алгоритмів МН без вчителя на основі AE.
Вей Чжун та Фен Гу	Системи виявлення ЗПЗ	CNN, RNN і LSTM	Впровадження багаторівнених моделей ГН для виявлення ЗПЗ
Харді та інші	Виявлення ЗПЗ	SAE	Впровадження фреймворку для виявлення ЗПЗ.
Рузмелен та інші	Виявлення ботнетів	SDA та DNN	Реалізація додатку, використовуючи методи ГН, для виявлення ботнетів.

Життєвий цикл моделі машинного навчання

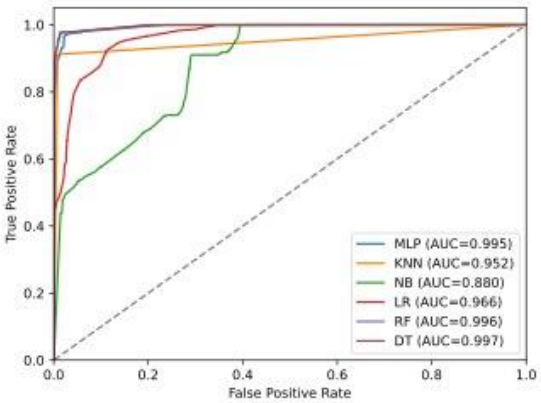
7



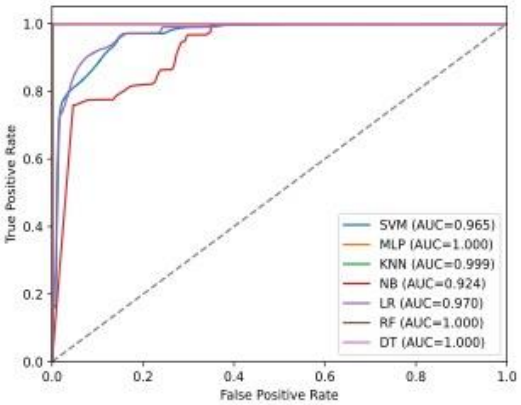
Тренування алгоритмів машинного навчання

8

Після тренування моделей машинного навчання на локальному (LAN) наборі даних отримали наступні результати:



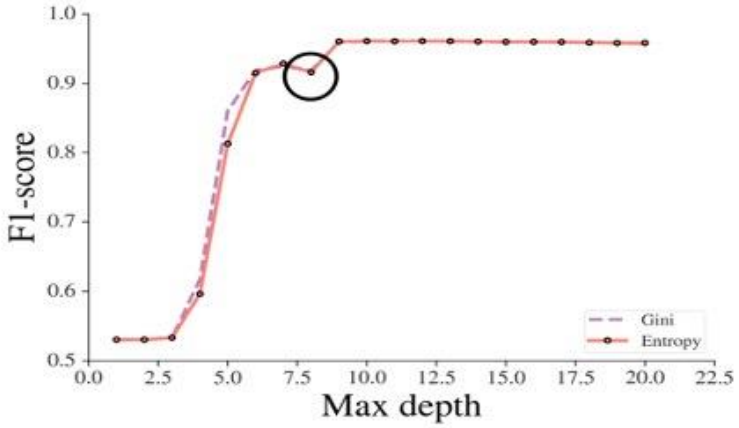
Після тренування моделей машинного навчання на глобальному (Інтернет) наборі даних отримали наступні результати:



Оптимізація гіперпараметрів за допомогою функції grid-search

9

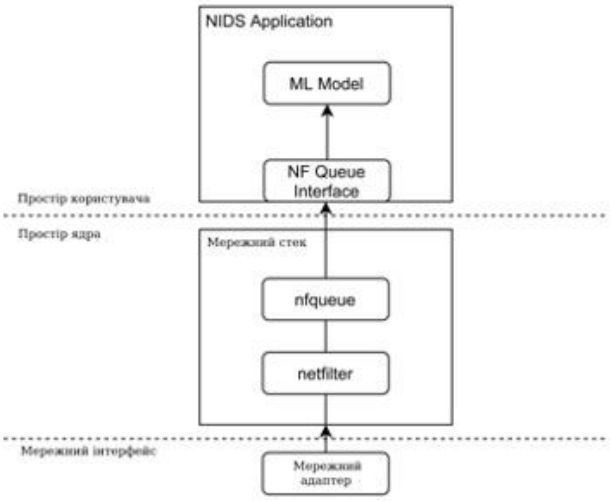
Відносно графіку зображеного на рисунку можна відстежити значення оптимальних параметрів алгоритму. Як зазначено на рисунку, обведеній в точці відбулось перетренування алгоритму. Відповідно подальше використання може призвести до збільшення кількості неправильних спрацювань. Бібліотека scikit-learn, дозволяє зберегти найкращі параметри моделі та відкинути ті, що були перетреновані. Значення F1 в даному випадку приблизно рівне 0.96



Алгоритм реалізаціїї системи виявлення вторгнень

10

Реалізована система виявлення вторгнень буде взаємодіяти з утилітою iptables використовуючи модуль Netfilter NFQUEUE, однойменна бібліотека підтримується в Python та може бути легко імплементована в додаток для виявлення вторгнень. В даному алгоритмі реалізації зображеному на рисункувсі пакети надходять з простору ядра до користувача, а потім розпізнаються алгоритмом машинного навчання заданим при ініціалізації додатку.

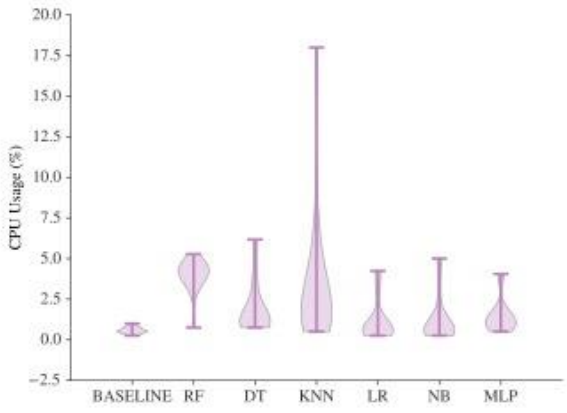


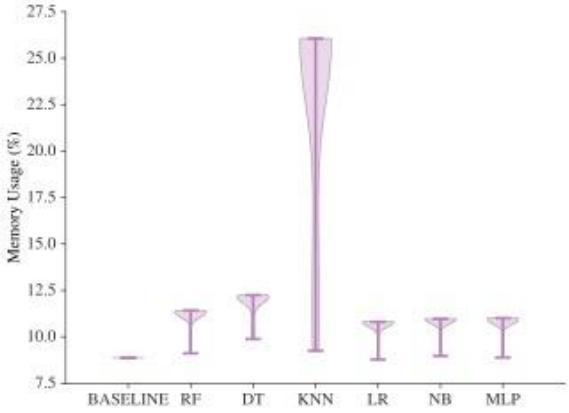
Продуктивність системи виявлення вторгнень

10

З графіку використання CPU, найбільш ресурсозатратними алгоритмами є:
RF та KNN.

З графіку використання RAM, найбільш ресурсозатратними алгоритмами є:
DT та KNN





Рекомендації щодо розробки системи виявлення сканувань на базі методів машинного навчання

11

Розглянемо основні рекомендації щодо розробки системи виявлення сканувань:

- необхідно чітко визначити поставлене завдання та необхідність використання алгоритмів машинного навчання;
- набори даних без зловмисного трафіку можуть бути використані з відкритих джерел;
- необхідно великі варіативні набори даних для тестувань та тренувань алгоритмів машинного навчання;
- варто впроваджувати підтримку всіх мережних протоколів для збільшення кількості успішного виявлення загроз;
- необхідно відслідковувати процес тренування для аналізу стану навчання моделі машинного навчання;
- деякі з ознак мережних пакетів необхідно знеособити для стабільної роботи моделі машинного навчання у справжньому середовищі;
- необхідно відстежувати середній час на розпізнавання алгоритмів для уникнення переповнення буферу;
- високе навантаження використання CPU та RAM може призвести до неможливості розгортання системи виявлення сканувань на пристрої;
- будь-яку готову систему виявлення сканувань необхідно протестувати у справжньому середовищі перед розгортанням;
- методи глибоко навчання можуть успішно використовуватись для аналізу трафіку.

ВИСНОВКИ

12

В роботі проведено аналіз проблеми виявлення сканувань на базі методів машинного навчання, визначено його мету та завдання.

1. Проведено аналіз існуючих технологій сканування мережі. Сканування служб і сервісів, запущених на комп'ютерах мережі є першим етапом підготовки і проведення атаки на комп'ютерні системи. Для цих цілей зловмисники використовують мережні сканери. Для виявлення сканувань використовують брандмауери, журнали сервісів та системи виявлення вторгнень.
2. Досліджено методи та засоби сканування структури мережі. Nmap – вважається кращим сканером, що використовує різні методи сканування за допомогою флагів або надсилання корисного навантаження. Крім того Nmap підтримує використання скриптів з метою автоматизації процесу сканування та пошуку вразливостей.
3. Визначено методи, основні функції та алгоритми застосування машинного навчання. Технології машинного навчання можуть прогнозувати значення, виявляти незвичайні сценарії та закономірності, визначати структуру або створювати категорії.
4. Розглянуто способи використання методів глибоко навчання для реалізації систем виявлення вторгнень. Необхідність класичних методів у більшості випадків зазначена лише для порівняння точності моделей досліджень. Системи виявлення вторгнень можуть бути легко імплементовані для виявлення сканувань та виконують функції захисту від ШПЗ, ботнетів, аномалій та зловмисного використання.
5. Розроблено варіант системи виявлення сканувань на базі методів машинного навчання з використанням класичних алгоритмів.
6. Розроблено рекомендації щодо розробки системи виявлення сканувань на базі методів машинного навчання.

Таким чином, правильна системи виявлення сканувань має протидіяти вторгненням до мережі та виявляти в режимі реального часу процес сканування з метою запобігання можливих кіберінцидентів.

14

Дякую за увагу!
Доповідь закінчено

ЛІСТИНГ МОДУЛІВ ДОДАТКУ

Лістинг модулю train_local_py.py

```
import os

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC

from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold, GridSearchCV, cross_val_predict,
cross_val_score
from sklearn.metrics import make_scorer, f1_score, roc_auc_score, roc_curve
from sklearn.tree import export_graphviz
import graphviz
import emlearn

from sklearn import tree

pd.set_option("display.max_columns", 200)
import warnings
warnings.filterwarnings('ignore')

#read dataset and labels
df = pd.read_csv("data/attack_dataset.csv") # attack dataset
attack_classes = pd.read_csv(".data/attack_labels_sbseg.csv") # labels
bonafide = pd.read_csv('data/bonafide_dataset_20191121.csv.gz') # bonafide dataset
print(df.shape, bonafide.shape)

#create attack dataset with labels defined by ip_src
df_labeled = df.merge(attack_classes, how='inner', left_on='ip.src', right_on='ip')
df_labeled.drop(['ip'], axis=1, inplace=True)

#create column label on bonafide dataset
bonafide['label'] = "bonafide"

#comparing bonafide and attack datasets
if (df_labeled.columns == bonafide.columns).all():
    examples_attack = df_labeled.shape[0]
    examples_bonafide = bonafide.shape[0]
    total = examples_attack+examples_bonafide
    print('Total examples of {0} with {1:0.2f} of attack and {2:0.2f} bonafide packets'.format(total,
examples_attack/total, examples_bonafide/total))

#pre-processing
```

```
fields = ['eth.type', 'ip.id', 'ip.flags', 'ip.checksum', 'ip.dsfield', 'tcp.flags', 'tcp.checksum']

for field in fields:
   df_labeled[field] = df_labeled[field].apply(lambda x: int(str(x), 16))

bonafide = bonafide.fillna(0)
for field in fields:
   bonafide[field] = bonafide[field].apply(lambda x: int(str(x), 16))

#create AB-TRAP dataset with all packets (bonafide and attack)
full_data = pd.concat([bonafide, df_labeled])

#protocol check
wrong_proto = full_data[full_data['ip.proto'] != 6]['label'].value_counts().values
full_data = full_data[full_data['ip.proto'] == 6]
print("It was found and removed", wrong_proto,"packets.")

#features not applicable to this study
full_data.drop(columns=['frame_info.time', 'frame_info.encap_type', 'frame_info.time_epoch',
'frame_info.number',
                  'frame_info.len', 'frame_info.cap_len', 'eth.type', 'ip.flags', 'ip.src', 'ip.dst',
                  'ip.version', 'ip.proto', 'tcp.flags'], axis=1, inplace=True)

#remove columns with zero variance
full_data.drop(columns=['ip.hdr_len', 'ip.tos', 'ip.flags.rb',
                  'ip.flags.mf', 'ip.frag_offset'], axis=1, inplace=True)

#plot box_plot
_columns = 3
lines = int(full_data.shape[1]/_columns)+1

plt.figure(figsize = [100, 100])

i = 1
for column in full_data.columns.values:
   if column != "label":
      plt.subplot(lines, _columns, i)
      full_data.boxplot([column]);
      i += 1
plt.clear()

#bivariate Analysis - Linear correlation (absolute threshold of 0.5)
high_corr = full_data.corr().abs().round(2)
high_corr_var = high_corr[high_corr>0.5]
plt.figure(figsize = (20,16))
sns.heatmap(high_corr_var, xticklabels=high_corr_var.columns, yticklabels=high_corr_var.columns,
annot=True)

#check counts
print(full_data['label'].value_counts())

full_data.label[full_data.label == "bonafide"] = 0 # convert bonafide label to 0
```

```
full_data.label[full_data.label != 0] = 1 # convert attack labels to 1
full_data['label'].value_counts()

# It is removed ttl because previous attempt shows that it is learning the LAN architecture TTL=62
# (from scan tools TTL=64 minus 2 routers in the infrastructure)

# sequence, checksum and acknowledge features because they are random

# removed source and destination ports to be agnostic regarding the service ports

# removed tcp.options.mss_val because it is dificult to be retrieved as LKM

full_data.drop(columns=["ip.checksum", "ip.ttl", "tcp.checksum", "tcp.dstport", "tcp.seq",
"tcp.srcport",
                       "tcp.ack", "tcp.options.mss_val"], axis=1, inplace=True)

algorithms = {
    "MLP" : (MLPClassifier(random_state=17), {
        "hidden_layer_sizes" : (10, 10),
    }),
    "SVM" : (LinearSVC(random_state=17), {}),
    "KNN" : (KNeighborsClassifier(n_jobs=-1), {
        "n_neighbors" : [1, 3, 5]
    }),
    "NB" : (GaussianNB(), {}),
    "LR" : (LogisticRegression(random_state=17, n_jobs=-1), {}),
    "RF" : (RandomForestClassifier(random_state=17, n_jobs=-1), {
        "n_estimators" : [10, 15, 20],
        "criterion" : ("gini", "entropy"),
        "max_depth": [5, 10],
        "class_weight": (None, "balanced", "balanced_subsample")
    }),
    "DT" : (DecisionTreeClassifier(random_state=17), {
        "criterion": ("gini", "entropy"),
        "max_depth": [5, 10, 15],
        "class_weight": (None, "balanced")
    }),
}

full_data = full_data.fillna(0)
X = full_data.drop(columns = ["label"])
y = full_data.label

print(X.shape, y.shape)

#remove mistake of train(change variable)
X = X.astype(int)

#grid Search
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=17) # Train, Test
gskf = StratifiedKFold(n_splits=3, shuffle=True, random_state=17) # Validation
perf = f1_score # can be used roc_auc_score for binary classification
```

```
score = {}
for algorithm in algorithms.keys():
    score[algorithm] = []

for algorithm, (clf, parameters) in algorithms.items():
    print(algorithm)
    for train, test in kf.split(X, y):
        prep = StandardScaler()
        #prep = MinMaxScaler()
        prep.fit(X.iloc[train])
        best = GridSearchCV(clf, parameters, cv=gskf, scoring=make_scorer(perf))
        best.fit(prep.transform(X.iloc[train]), y.iloc[train])
        score[algorithm].append(perf(best.predict(prep.transform(X.iloc[test])), y.iloc[test]))

# f1-scores
pd.DataFrame.from_dict(score)

#ROC/AUC evaluation
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=17)  # Train, Test
gskf = StratifiedKFold(n_splits=3, shuffle=True, random_state=17)  # Validation
perf = roc_auc_score

results = {}
for algorithm in algorithms.keys():
    results[algorithm] = {'expected': [], 'predicted': []}

for algorithm, (clf, parameters) in algorithms.items():
    print(algorithm)
    for train, test in kf.split(X, y):
        prep = StandardScaler()
        # prep = MinMaxScaler()
        prep.fit(X.iloc[train])
        best = GridSearchCV(clf, parameters, cv=gskf, scoring=make_scorer(perf))
        best.fit(prep.transform(X.iloc[train]), y.iloc[train])

        results[algorithm]['expected'].extend(y.iloc[test])

results[algorithm]['predicted'].extend(best.predict_proba(prep.transform(X.iloc[test])).transpose()[1])

# ROC/AUC scores for the best set of parameters from the Grid Search above (for each k-fold)
for algorithm in algorithms.keys():
    auc = roc_auc_score(results[algorithm]['expected'], results[algorithm]['predicted'])
    print('AUC(%s) = %.4f' % (algorithm, auc))

#draw TP FP
plt.clear()

plt.figure()

index = 0
for model_key, result in results.items():
```

```
    fpr, tpr, thresholds = roc_curve(result['expected'], result['predicted'])
    AUC = auc(fpr, tpr)

    plt.plot(fpr, tpr, label="{} (AUC={:.3f})".format(model_key, AUC))
    index = index + 1
    # save ROC data
    filename = 'data/LAN_' + model_key + '_ROC_data.csv'
    pd.DataFrame.from_dict(data={'fpr': fpr, 'tpr': tpr, 'thresholds': thresholds}).to_csv(filename,
index=False)

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# plt.title('Receiver operating characteristic (ROC)')
plt.legend(loc="lower right")

plt.savefig(os.path.join('images/', 'roc_lan_models.pdf'), dpi=300, bbox_inches="tight")
plt.clear()

#results from Grid Search
dados = []
for i in range(0,len(best.cv_results_['params'])):
    print(best.cv_results_['params'][i], best.cv_results_['mean_test_score'][i],
          best.cv_results_['std_test_score'][i])
    dados.append([best.cv_results_['params'][i]['criterion'],
                  best.cv_results_['params'][i]['max_depth'], best.cv_results_['mean_test_score'][i]])

gini = {}
entropia = {}
for valores in dados:
    if valores[0] == "gini":
        gini.update({valores[1] : valores[2]})
    else:
        entropia.update({valores[1] : valores[2]})

#F1 Score/Max depth
lists1 = sorted(gini.items())
lists2 = sorted(entropia.items())
x_gini, y_gini = zip(*lists1)
x_entropia, y_entropia = zip(*lists2)

#plt.figure(figsize=(10,8))
#plt.rcParams.update({'font.size': 15})

#plt.title('Performance according to grid-search parameters')
plt.ylabel('F1-score', fontsize=20)
plt.xlabel('Max depth', fontsize=20)
plt.plot(x_gini, y_gini, '--', label='Gini')
plt.plot(x_entropia, y_entropia, '.-', label='Entropy')
plt.legend(loc="lower right");
```

```
import os
plt.savefig(os.path.join('images/', 'grid_search_dt.pdf'), dpi=300, bbox_inches = "tight")
plt.clear()

#ROC Analysis

kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=17)

clf = DecisionTreeClassifier(criterion='gini', max_depth=11, class_weight="balanced") # {0: 0.01,
1:0.99}

predicted = cross_val_predict(clf, X, y, cv=kf, method='predict_proba')
print(predicted.transpose()[1]) # Probability of the positive class

fpr, tpr, thr = roc_curve(y, predicted.transpose()[1])
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',lw=lw)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
plt.clear()

#threshold
goal = 0.99
print('Threshold to achieve FPR < %.2f: %.2f' % (1-goal, thr[np.where(fpr < 1 - goal)[0][-1]]))
print('Threshold to achieve TPR > %.2f: %.2f' % (goal, thr[np.where(tpr > goal)[0][0]]))

#threshold
goal = 0.95
print('Threshold to achieve FPR < %.2f: %.2f' % (1-goal, thr[np.where(fpr < 1 - goal)[0][-1]]))
print('Threshold to achieve TPR > %.2f: %.2f' % (goal, thr[np.where(tpr > goal)[0][0]]))

#threshold
goal = 0.90
print('Threshold to achieve FPR < %.2f: %.2f' % (1-goal, thr[np.where(fpr < 1 - goal)[0][-1]]))
print('Threshold to achieve TPR > %.2f: %.2f' % (goal, thr[np.where(tpr > goal)[0][0]]))

for thresh in [.1, .2, .3, .4, .5, .6, .7, .8, .9]:
    print('Using threshold = %.1f, we have FPR=%.2f and TPR=%.2f' %
          (thresh, fpr[np.where(thr <= thresh)[0][0]], tpr[np.where(thr <= thresh)[0][0]]))

#Feature Importance Evaluation
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=17)

clf = DecisionTreeClassifier(criterion='gini', max_depth=11, class_weight="balanced") # {0: 0.01,
1:0.99}
```

```
scores = cross_val_score(clf, X, y, cv=kf, scoring='f1') # recall

print("F1-Score: %0.3f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

#DecisionTreeClassifier
clf.fit(X, y)


#feature importance
feature_importance = np.array(clf.feature_importances_)
feature_names = np.array(X.columns)

data = {'feature_name': feature_names,
      'feature_importance': feature_importance}

fi_df = pd.DataFrame(data)

fi_df.sort_values(by=['feature_importance'], ascending=False, inplace=True)

relevantes = fi_df[fi_df.feature_importance > 0]

#plt.figure(figsize=(10,8))
g=sns.barplot(x=relevantes['feature_importance'], y=relevantes['feature_name'])

plt.xlabel('Importance', fontsize=20)
plt.ylabel('Feature', fontsize=20);
i=0
for index, row in relevantes.iterrows():
   g.text(row.feature_importance+0.03, i, round(row.feature_importance, 4), color='black',
ha="center", va="center", fontsize=9)
   i+=1
plt.savefig(os.path.join('images/', 'feature_importance_dt.pdf'), dpi=300, bbox_inches = "tight")



#Random forest
fn = X.columns
cn = ['normal', 'scan']
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(30, 30), dpi=300)
tree.plot_tree(clf,
            feature_names=fn,
            class_names=cn,
            filled=True)


graphviz.Source(export_graphviz(clf, out_file=None, feature_names=X.columns.tolist()))

#Inference model
cmodel = emlearn.convert(clf)
```

Лістинг модулю train_internet_py.py

```
import os

import time


import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.linear_model import LogisticRegression

from sklearn.neural_network import MLPClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import LinearSVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import StratifiedKFold, GridSearchCV

from sklearn.metrics import make_scorer, f1_score, roc_auc_score, roc_curve

from joblib import dump


pd.set_option("display.max_columns", 200)

import warnings

warnings.filterwarnings('ignore')



#read dataset and labels

df = pd.read_csv("Internet/attack_dataset.csv.gz") # attack dataset

bonafide = pd.read_csv('data/bonafide_dataset_20191121.csv.gz') # bonafide traffic from mawilab

bonafide = pd.concat([bonafide, pd.read_csv('data/bonafide_dataset_20201110.csv.gz')])

bonafide = pd.concat([bonafide, pd.read_csv('data/bonafide_dataset_20201129.csv.gz')])

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

```
#label column in the bonafide dataset
bonafide['label'] = "bonafide"


#comparison of datasets
if (df.columns == bonafide.columns).all():
    examples_malicious = df.shape[0]
    examples_legitim = bonafide.shape[0]
    total = examples_malicious+examples_legitim
    print('Total examples of {0} with {1:0.2f} of attack and {2:0.2f} bonafide packets'.format(total, examples_malicious/total, examples_legitim/total))


#pre-processing
fields = ['eth.type', 'ip.id', 'ip.flags', 'ip.checksum', 'ip.dsfield', 'tcp.flags', 'tcp.checksum']


for field in fields:
    df[field] = df[field].apply(lambda x: int(str(x), 16))


bonafide = bonafide.fillna(0)
for field in fields:
    bonafide[field] = bonafide[field].apply(lambda x: int(str(x), 16))


#create a dataset with all packets (bonafide and attack)
full_data = pd.concat([bonafide, df])


#check if there are packets with protocol field different than TCP (value 6)
wrong_proto = full_data[full_data['ip.proto'] != 6]['label'].value_counts().values
full_data = full_data[full_data['ip.proto'] == 6]
print("Found and removed", wrong_proto,"packets from the original dataset.")


#features not applicable to this work
```

```
full_data.drop(columns=['frame_info.time', 'frame_info.encap_type', 'frame_info.time_epoch',
'frame_info.number',
                'frame_info.len', 'frame_info.cap_len', 'eth.type', 'ip.flags', 'ip.src', 'ip.dst',
                'ip.version', 'ip.proto', 'tcp.flags'], axis=1, inplace=True)


# check features with zero variance, they do not support the learning task
(full_data.var() == 0)


#remove columns with variance zero
full_data.drop(columns=['ip.hdr_len', 'ip.tos', 'ip.flags.rb',
                'ip.flags.mf', 'ip.frag_offset'], axis=1, inplace=True)


#Bivariate Anaysis
high_corr = full_data.corr().abs().round(2)
high_corr_var = high_corr[high_corr>0]
plt.figure(figsize = (20,16))
sns.heatmap(high_corr_var, xticklabels=high_corr_var.columns, yticklabels=high_corr_var.columns,
annot=True)


# replace "normal" labels to 0
# replace all scan labels to 1
full_data.label[full_data.label == "bonafide"] = 0
full_data.label[full_data.label != 0] = 1
full_data['label'].value_counts()


#Remove more columns
#checksum and acknowlegde are random
#tcp.dstport will tend to learn the testbed (some tools were targeted to specific services)
full_data.drop(columns=["ip.checksum", "tcp.checksum",
                "tcp.ack", "tcp.dstport"], axis=1, inplace=True)
```

```
#drop duplicates
full_data.drop_duplicates(inplace=True, ignore_index=True)



#create algorithm
algorithms = {
    "MLP" : (MLPClassifier(random_state=17), {
        "hidden_layer_sizes" : (10, 10),
    }),
    #"SVM" : (LinearSVC(random_state=17), {}),
    "KNN" : (KNeighborsClassifier(n_jobs=-1), {
        "n_neighbors" : [1, 3, 5]
    }),
    "NB" : (GaussianNB(), {}),
    "LR" : (LogisticRegression(random_state=17, n_jobs=-1), {}),
    "RF" : (RandomForestClassifier(random_state=17, n_jobs=-1), {
        "n_estimators" : [10, 15, 20],
        "criterion" : ("gini", "entropy"),
        "max_depth": [5, 10],
        "class_weight": (None, "balanced", "balanced_subsample")
    }),
    "DT" : (DecisionTreeClassifier(random_state=17), {
        "criterion": ("gini", "entropy"),
        "max_depth": [5, 10, 15],
        "class_weight": (None, "balanced")
    }),
}

full_data = full_data.fillna(0)
X = full_data.drop(columns = ["label"])
```

```
y = full_data.label



#remove mistake of train(change variable)
X = X.astype(int)

#Grid Search for Machine Learning Algorithms
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=17) # Train, Test
gskf = StratifiedKFold(n_splits=3, shuffle=True, random_state=17) # Validation
perf = f1_score # could be considered roc_auc_score for binary classification

score = {}
best_params = {}
for algorithm in algorithms.keys():
    score[algorithm] = []

for algorithm, (clf, parameters) in algorithms.items():
    print(algorithm)
    for train, test in kf.split(X, y):
        prep = StandardScaler()
        dump(prep, open('Models/preprocessor.pkl', 'wb'))
        #prep = MinMaxScaler()
        prep.fit(X.iloc[train])
        best = GridSearchCV(clf, parameters, cv=gskf, scoring=make_scorer(perf), n_jobs=-1)
        best.fit(prep.transform(X.iloc[train]), y.iloc[train])
        dump(best, open('Models/{}.pkl'.format(algorithm), 'wb'))
        best_params[algorithm] = best.best_params_
        score[algorithm].append(perf(best.predict(prep.transform(X.iloc[test])), y.iloc[test]))

#f1-scores for the best set of parameters from the Grid Search above (for each k-fold)
pd.DataFrame.from_dict(score)
```

```
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=17)  # Train, Test
gskf = StratifiedKFold(n_splits=3, shuffle=True, random_state=17)  # Validation
perf = roc_auc_score

results = {}

for algorithm in algorithms.keys():
    results[algorithm] = {'expected': [], 'predicted': []}

for algorithm, (clf, parameters) in algorithms.items():
    print(algorithm)
    for train, test in kf.split(X, y):
        prep = StandardScaler()
        prep.fit(X.iloc[train])

        best = GridSearchCV(clf, parameters, cv=gskf, scoring=make_scorer(perf), n_jobs=-1)
        best.fit(prep.transform(X.iloc[train]), y.iloc[train])
        results[algorithm]['expected'].extend(y.iloc[test])

results[algorithm]['predicted'].extend(best.predict_proba(prep.transform(X.iloc[test])).transpose()[1])

# ROC/AUC scores for the best set of parameters from the Grid Search above (for each k-fold)
for algorithm in algorithms.keys():
    auc = roc_auc_score(results[algorithm]['expected'], results[algorithm]['predicted'])
    print('AUC(%s) = %.4f' % (algorithm, auc))

#True positive and False positive
plt.clf() #release memory
plt.close()
plt.figure()
```

```
try:
    for model_key, result in results.items():
        fpr, tpr, thresholds = roc_curve(result['expected'], result['predicted'])
        AUC = auc(fpr, tpr)
        plt.plot(fpr, tpr, label="{} (AUC={:.3f})".format(model_key, AUC))
        # save ROC data
        filename = 'data/Internet_' + model_key + '_ROC_data.csv'
        pd.DataFrame.from_dict(data={'fpr': fpr, 'tpr': tpr, 'thresholds': thresholds}).to_csv(filename,
index=False)
except:pass


plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# plt.title('Receiver operating characteristic (ROC)')
plt.legend(loc="lower right")


plt.savefig(os.path.join('images/', 'roc_internet_models.pdf'), dpi=300, bbox_inches="tight")



dados = []
for i in range(0,len(best.cv_results_['params'])):
    print(best.cv_results_['params'][i], best.cv_results_['mean_test_score'][i],
          best.cv_results_['std_test_score'][i])
    dados.append([best.cv_results_['params'][i]['criterion'],
                  best.cv_results_['params'][i]['max_depth'], best.cv_results_['mean_test_score'][i]])


gini = {}
entropia = {}
```

```
for valores in dados:
    if valores[0] == "gini":
        gini.update({valores[1] : valores[2]})
    else:
        entropia.update({valores[1] : valores[2]})


lists1 = sorted(gini.items())
lists2 = sorted(entropia.items())
x_gini, y_gini = zip(*lists1)
x_entropia, y_entropia = zip(*lists2)


plt.clf() #release memory
plt.close()
plt.style.use('plot_style.txt')
plt.figure(figsize=(10,8))
plt.rcParams.update({'font.size': 15})


#plt.title('Performance according to grid-search parameters')
plt.ylabel('F1-score', fontsize=20)
plt.xlabel('Max depth', fontsize=20)
plt.plot(x_gini, y_gini, '--', label='Gini')
plt.plot(x_entropia, y_entropia, '.-', label='Entropy')
plt.legend(loc="lower right");
plt.savefig(os.path.join('images/', 'grid_search_dt.pdf'), dpi=300, bbox_inches = "tight")
```

Лістинг модулю generate_model.py

```
import time

from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier

from joblib import dump
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import pandas as pd
import numpy as np
pd.set_option("display.max_columns", 200)
import warnings
warnings.filterwarnings('ignore')


df = pd.read_csv("Internet/attack_dataset.csv.gz") # attack dataset
bonafide = pd.read_csv('data/bonafide_dataset_20191121.csv.gz') # bonafide traffic from mawilab
bonafide = pd.concat([bonafide, pd.read_csv('data/bonafide_dataset_20201110.csv.gz')])
bonafide = pd.concat([bonafide, pd.read_csv('data/bonafide_dataset_20201129.csv.gz')])
print(df.shape, bonafide.shape)

bonafide['label'] = "bonafide"

fields = ['eth.type', 'ip.id', 'ip.flags', 'ip.checksum', 'ip.dsfield', 'tcp.flags', 'tcp.checksum']

for field in fields:
    df[field] = df[field].apply(lambda x: int(str(x), 16))

bonafide = bonafide.fillna(0)
for field in fields:
    bonafide[field] = bonafide[field].apply(lambda x: int(str(x), 16))

full_data = pd.concat([bonafide, df])

wrong_proto = full_data[full_data['ip.proto'] != 6]['label'].value_counts().values
full_data = full_data[full_data['ip.proto'] == 6]
print("Found and removed", wrong_proto,"packets from the original dataset.")

full_data.drop(columns=['frame_info.time', 'frame_info.encap_type', 'frame_info.time_epoch',
'frame_info.number',
                        'frame_info.len', 'frame_info.cap_len', 'eth.type', 'ip.flags', 'ip.src', 'ip.dst',
                        'ip.version', 'ip.proto', 'tcp.flags'], axis=1, inplace=True)

full_data.drop(columns=['ip.hdr_len', 'ip.tos', 'ip.flags.rb',
                        'ip.flags.mf', 'ip.frag_offset'], axis=1, inplace=True)

full_data.label[full_data.label == "bonafide"] = 0 # replace "normal" labels to 0
```

```
full_data.label[full_data.label != 0] = 1 # replace all scan labels to 1
full_data['label'].value_counts()

full_data.drop(columns=["ip.checksum", "tcp.checksum",
                        "tcp.ack", "tcp.dstport"], axis=1, inplace=True)

full_data.drop_duplicates(inplace=True, ignore_index=True)

full_data = full_data.fillna(0)
X = full_data.drop(columns = ["label"])
y = full_data.label
X = X.astype(int)
X.columns.values

prep = StandardScaler()
prep.fit(X)


dump(prep, open('Models/preprocessor.pkl', 'wb'))

mlp = MLPClassifier(hidden_layer_sizes=(10, 10), random_state=17)
svm = LinearSVC(random_state=17)
knn = KNeighborsClassifier(n_neighbors=1, n_jobs=-1)
nb = GaussianNB()
rf = RandomForestClassifier(class_weight="balanced_subsample", criterion="entropy",
max_depth=10, n_estimators=15, n_jobs=-1, random_state=17)
dt = DecisionTreeClassifier(class_weight=None, criterion="entropy", max_depth=15,
random_state=17)

mlp.fit(X, y)
svm.fit(X, y)
knn.fit(X, y)
nb.fit(X, y)
rf.fit(X, y)
dt.fit(X, y)

dump(mlp, open('Models/mlp.pkl', 'wb'))
dump(svm, open('Models/svm.pkl', 'wb'))
dump(knn, open('Models/knn.pkl', 'wb'))
dump(nb, open('Models/nb.pkl', 'wb'))
dump(rf, open('Models/rf.pkl', 'wb'))
dump(dt, open('Models/dt.pkl', 'wb'))
```

Лістинг модулю data_analysis.py

```
 import pandas as pd
import matplotlib.pyplot as plt
import os
import numpy as np
import scipy.stats

models = ['baseline', 'rf', 'dt', 'knn', 'lr', 'nb', 'mlp']
features = ['%sys', '%memused']

data = {}

for model in models:
    df = pd.read_csv("results/{0}.csv".format(model), sep=';')
    data[model] = {
                "sys_mean" :
df[features].replace(',','.',regex=True).astype(float).describe()['%sys']['mean'],
                "sys_std"  : df[features].replace(',','.',regex=True).astype(float).describe()['%sys']['std'],
                "sys_max"  : df[features].replace(',','.',regex=True).astype(float).describe()['%sys']['max'],
                "sys_min"  : df[features].replace(',','.',regex=True).astype(float).describe()['%sys']['min'],
                "mem_mean" :
df[features].replace(',','.',regex=True).astype(float).describe()['%memused']['mean'],
                "mem_std"  :
df[features].replace(',','.',regex=True).astype(float).describe()['%memused']['std'],
                "mem_max"  :
df[features].replace(',','.',regex=True).astype(float).describe()['%memused']['max'],
                "mem_min"  :
df[features].replace(',','.',regex=True).astype(float).describe()['%memused']['min']
               }

m = df.var() > 0
df_filtered = df.loc[:, m.reindex(df.columns, axis=1, fill_value=False)]
df_filtered.columns.values

x_pos = np.arange(len(models))
sys_mean = [value['sys_mean'] for value in data.values()]
sys_std = [value['sys_std'] for value in data.values()]

plt.style.use('plot_style.txt')


fig, ax = plt.subplots()
ax.bar(x_pos, sys_mean, yerr=sys_std, align='center', alpha=0.5, ecolor='black', capsize=10)
ax.set_ylabel('CPU Usage (%)')
ax.set_xticks(x_pos)
ax.set_xticklabels(model.upper() for model in models)
for i in range(0, len(sys_mean)):
    plt.text(i, sys_mean[i]+sys_std[i]+0.2, str(round(sys_mean[i],2))+" ± "+str(round(sys_std[i],2)),
ha="center", va="bottom", rotation=90)
plt.savefig(os.path.join('images/', 'cpu_usage_ml_models.pdf'), dpi=300, bbox_inches = "tight")
plt.show()
plt.clear()
```

```
x_pos = np.arange(len(models))
mem_mean = [value['mem_mean'] for value in data.values()]
mem_std = [value['mem_std'] for value in data.values()]

fig, ax = plt.subplots()
ax.bar(x_pos, mem_mean, yerr=mem_std, align='center', alpha=0.5, ecolor='black', capsize=10)
ax.set_ylabel('Memory Usage (%)')
ax.set_xticks(x_pos)
ax.set_xticklabels(model.upper() for model in models)
for i in range(0, len(mem_mean)):
   plt.text(i, mem_mean[i]+mem_std[i]+0.4, str(round(mem_mean[i],2))+" ±
"+str(round(mem_std[i],2)), ha="center", va="bottom", rotation=90)
plt.savefig(os.path.join('images/', 'mem_usage_ml_models.pdf'), dpi=300, bbox_inches = "tight")
plt.show()
plt.clear()

data = {}

for model in models:
   df = pd.read_csv("results/{0}.csv".format(model), sep=';')
   data[model] = {
                  "sys" : df[features]['%sys'].replace(',','.',regex=True).astype(float),
                  "mem" : df[features]['%memused'].replace(',','.',regex=True).astype(float),
                 }


plot_sys = []

x_pos = np.arange(len(models))


fig, ax = plt.subplots()
for model in models:
   plot_sys.append(data[model]['sys'])

ax.set_ylabel('CPU Usage (%)')
#ax.set_xticks(x_pos+1)
#ax.set_xticklabels(model.upper() for model in models)


ax.violinplot(plot_sys);
plt.xticks(x_pos+1, [model.upper() for model in models])
plt.savefig(os.path.join('images/', 'cpu_usage_ml_models_violin.pdf'), dpi=300, bbox_inches = "tight")
plt.show()
plt.clear()


plot_sys = []

x_pos = np.arange(len(models))
```

```
fig, ax = plt.subplots()
for model in models:
    plot_sys.append(data[model]['sys'])

ax.set_ylabel('CPU Usage (%)')
#ax.set_xticks(x_pos+1)
#ax.set_xticklabels(model.upper() for model in models)



ax.boxplot(plot_sys, showfliers=False);
plt.xticks(x_pos+1, [model.upper() for model in models])
plt.savefig(os.path.join('images/', 'cpu_usage_ml_models_boxplot.pdf'), dpi=300, bbox_inches =
"tight")
plt.show()
plt.clear()

[model.upper() for model in models]

plot_mem = []

x_pos = np.arange(len(models))


fig, ax = plt.subplots()
for model in models:
    plot_mem.append(data[model]['mem'])

ax.set_ylabel('Memory Usage (%)')
#ax.set_xticks(x_pos+1)
#ax.set_xticklabels(model.upper() for model in models)

ax.violinplot(plot_mem);
plt.xticks(x_pos+1, [model.upper() for model in models])
plt.savefig(os.path.join('images/', 'mem_usage_ml_models_violin.pdf'), dpi=300, bbox_inches =
"tight")
plt.show()


plot_mem = []

x_pos = np.arange(len(models))


fig, ax = plt.subplots()
for model in models:
    plot_mem.append(data[model]['mem'])

ax.set_ylabel('Memory Usage (%)')
#ax.set_xticks(x_pos+1)
#ax.set_xticklabels(model.upper() for model in models)
```

```
ax.boxplot(plot_mem, showfliers=False, notch=True);
plt.xticks(x_pos+1, [model.upper() for model in models])
plt.savefig(os.path.join('images/', 'mem_usage_ml_models_boxplot.pdf'), dpi=300, bbox_inches =
"tight")
plt.show()


def mean_confidence_interval(data, confidence=0.95):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), scipy.stats.sem(a)
    h = se * scipy.stats.t.ppf((1 + confidence) / 2., n-1)
    return m, m-h, m+h


for metric in ['sys', 'mem']:
    for model in models:
        mean, bottom, top = mean_confidence_interval(data[model][metric])
        print(metric, model, mean, bottom, top, (mean>bottom and mean<top))


def adjacent_values(vals, q1, q3):
    upper_adjacent_value = q3 + (q3 - q1) * 1.5
    upper_adjacent_value = np.clip(upper_adjacent_value, q3, vals.iloc[-1]) # replaced [-1] by iloc[-1]
for being Pandas Series

    lower_adjacent_value = q1 - (q3 - q1) * 1.5
    lower_adjacent_value = np.clip(lower_adjacent_value, vals[0], q1)
    return lower_adjacent_value, upper_adjacent_value


def set_axis_style(ax, labels):
    ax.xaxis.set_tick_params(direction='out')
    ax.xaxis.set_ticks_position('bottom')
    ax.set_xticks(np.arange(1, len(labels) + 1))
    ax.set_xticklabels(labels)
    ax.set_xlim(0.25, len(labels) + 0.75)
    ax.set_xlabel('Sample name')


plot_sys = []

x_pos = np.arange(len(models))

fig, ax = plt.subplots()

for model in models:
    plot_sys.append(data[model]['sys'])

ax.set_ylabel('CPU Usage (%)')
```

```
parts = ax.violinplot(
    plot_sys, showmeans=False, showmedians=False,
    showextrema=False)

# for pc in parts['bodies']:
# pc.set_facecolor('#D43F3A')
# pc.set_edgecolor('black')
# pc.set_alpha(1)

quartile1, medians, quartile3 = np.percentile(plot_sys, [25, 50, 75], axis=1)

whiskers = np.array([adjacent_values(sorted_array, q1, q3)
                     for sorted_array, q1, q3 in zip(plot_sys, quartile1, quartile3)])
# whiskers_min, whiskers_max = whiskers[:, 0], whiskers[:, 1]

whiskers_min = [i.min() for i in plot_sys]
whiskers_max = [i.max() for i in plot_sys]

inds = np.arange(1, len(medians) + 1)
ax.scatter(inds, medians, marker='.', color='white', s=10, zorder=3)
ax.vlines(inds, quartile1, quartile3, linestyle='-', lw=5)
ax.vlines(inds, whiskers_min, whiskers_max, linestyle='-', lw=1)

for i, model in enumerate(models):
    if model == "baseline":
        models[i] = "none"

plt.xticks(x_pos + 1, [model.upper() for model in models])
plt.savefig(os.path.join('images/', 'cpu_usage_ml_models_violin_plus_box.pdf'), dpi=300, bbox_inches
= "tight")
plt.show()

plot_mem = []

for i, model in enumerate(models):
    if model == "none":
        models[i] = "baseline"

x_pos = np.arange(len(models))

fig, ax = plt.subplots()

for model in models:
    plot_mem.append(data[model]['mem'])

ax.set_ylabel('RAM Memory Usage (%)')

parts = ax.violinplot(
    plot_mem, showmeans=False, showmedians=False,
    showextrema=False)

# for pc in parts['bodies']:
```

```
# pc.set_facecolor('#D43F3A')
# pc.set_edgecolor('black')
# pc.set_alpha(1)

quartile1, medians, quartile3 = np.percentile(plot_mem, [25, 50, 75], axis=1)

whiskers = np.array([adjacent_values(sorted_array, q1, q3)
                     for sorted_array, q1, q3 in zip(plot_mem, quartile1, quartile3)])
# whiskers_min, whiskers_max = whiskers[:, 0], whiskers[:, 1]

whiskers_min = [i.min() for i in plot_mem]
whiskers_max = [i.max() for i in plot_mem]

inds = np.arange(1, len(medians) + 1)
ax.scatter(inds, medians, marker='.', color='white', s=10, zorder=3)
ax.vlines(inds, quartile1, quartile3, linestyle='-', lw=5)
ax.vlines(inds, whiskers_min, whiskers_max, linestyle='-', lw=1)

for i, model in enumerate(models):
    if model == "baseline":
        models[i] = "none"

plt.xticks(x_pos + 1, [model.upper() for model in models])
plt.savefig(os.path.join('images/', 'mem_usage_ml_models_violin_plus_box.pdf'), dpi=300,
bbox_inches = "tight")
plt.show()
```

Лістинг модулю scanner.py

```
import fnfqueue
from joblib import load
from pandas import DataFrame
from scapy.all import *
import sys

if len(sys.argv) < 2:
   print("[Error] Not found model")
else:
   accept_model = ['knn', 'rf', 'dt', 'lr',  'mlp', 'nb', 'svm']
   if sys.argv[1] in accept_model:
      model_name = sys.argv[1] + ".pkl"
      queue = 1

      conn = fnfqueue.Connection()
      preprocess = load(open("Models/preprocessor.pkl", "rb"))
      model = load(open("Models/"+model_name, "rb"))
         cols = model.get_booster().feature_names
      except:
         #not a xgboost model
         xgboost = 0

      features = ['ip.id', 'ip.flags.df', 'ip.ttl', 'ip.len', 'ip.dsfield', 'tcp.srcport', 'tcp.seq', 'tcp.len',
'tcp.hdr_len',
               'tcp.flags.fin', 'tcp.flags.syn', 'tcp.flags.reset', 'tcp.flags.push', 'tcp.flags.ack', 'tcp.flags.urg',
               'tcp.flags.cwr', 'tcp.window_size', 'tcp.urgent_pointer', 'tcp.options.mss_val']

      try:
         q = conn.bind(queue)
         q.set_mode(0xffff, fnfqueue.COPY_PACKET)
      except PermissionError:
         print("Access denied; Do I have root rights or the needed capabilities?")
         sys.exit(-1)

      while True:
         try:
            for packet in conn:
               pkt = []
               layer3 = IP(packet.payload)
               layer4 = TCP(packet.payload)

               pkt.append(layer3.id)    #ip_id
               pkt.append(layer3.flags.DF & 1)  #ip_flag_df
               pkt.append(layer3.ttl)   #ip_ttl
               pkt.append(layer3.len)   #ip_len
               pkt.append(layer3.tos)     #ip_dsfield

               pkt.append(layer4.sport) #tcp_dport
               pkt.append(layer4.seq)   #tcp_seq
               pkt.append(len(layer4.payload))  #tcp_len
               pkt.append(len(layer4))  #tcp_hdr_len
```

```
                # bitwise operation on the flags to retrieve the value
                pkt.append(layer4.flags.F & 1)   #tcp_flag_fin
                pkt.append(layer4.flags.S & 1)   #tcp_flag_syn
                pkt.append(layer4.flags.R & 1)   #tcp_flag_rst
                pkt.append(layer4.flags.P & 1)   #tcp_flag_push
                pkt.append(layer4.flags.A & 1)   #tcp_flag_ack
                pkt.append(layer4.flags.U & 1)   #tcp_flag_urg
                pkt.append(layer4.flags.C & 1)   #tcp_flag_cwr
                pkt.append(layer4.window)        #tcp_window_size
                pkt.append(layer4.urgptr)        #tcp_urgent_pointer

                pkt.append(next((int.from_bytes(x[1], byteorder=sys.byteorder) for x in layer4.options if
x[0] == "MSS"), 0)) #tcp_options_mss_val

                df = DataFrame([pkt], columns=features)
                X = preprocess.transform(df)

                X = DataFrame(X, columns=features)
                predict = model.predict(X)

                if predict == 0:
                    packet.accept()
                else:
                    packet.drop()

            except fnfqueue.BufferOverflowException:
                print("buffer error")

        conn.close()
    else:
        print("[Error]")
```