

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ПРИКЛАДНОГО
СИСТЕМНОГО АНАЛІЗУ**

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«__» _____ 2024 р.

**Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Класифікація HTTP запитів за допомогою нейронних мереж»**

Виконав: студент IV курсу, групи КА-04

Свирид Нікіта Олександрович _____

Керівник: старший викладач кафедри ММСА, PhD

Левенчук Людмила Борисівна _____

Консультант з економічного розділу: доцент кафедри ЕК, PhD

Мажара Гліб Анатолійович _____

Консультант з нормоконтролю: доцент кафедри ММСА, к.ф.–м.н.

Статкевич Віталій Михайлович _____

Рецензент: доцент кафедри ШІ, PhD

Гуськова Віра Геннадіївна _____

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

Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
ІН Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Оксана ТИМОЩУК

«__» _____ 2024 р.

ЗАВДАННЯ

на дипломну роботу студенту

Свириду Нікіті Олександровичу

1. Тема роботи «Класифікація HTTP запитів за допомогою нейронних мереж», керівник роботи старший викладач кафедри ММСА, Левенчук Людмила Борисівна, затверджені наказом по університету від «31» травня 2024р. № 2240-с.

2. Термін подання студентом роботи – «11» червня 2024р.

3. Вихідні дані до роботи: HTTP запити спрямовані на веб додаток.

4. Зміст роботи: 1. Дослідження теоретичних складових мережевих протоколів, огляд існуючих рішень для боротьби з хакерськими атаками; 2. Дослідження актуальних підходів до побудови мовних моделей та класифікаторів; 3. Побудова мовної моделі та класифікаторів, аналіз та їх порівняння; 4. Функціонально-вартісний аналіз програмного продукту.

5. Перелік ілюстративного матеріалу: блок-схеми алгоритму навчання, графіки функцій активації, ілюстрації роботи методів машинного навчання, графіки порівняння якості моделей, презентація.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Мажара Г.А., доцент, к.е.н		

7. Дата видачі завдання «17» квітня 2023р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Формування тематики дослідження	23.04.2023	Виконано
2.	Попередня підготовка до написання роботи: огляд літератури за обраною тематикою, постановка задачі	30.04.2023	Виконано
3.	Підготовка першого розділу роботи: збір інформації, вибір даних, аналіз можливостей побудови програми	07.05.2023	Виконано
4.	Підготовка другого розділу роботи: теоретичний опис алгоритмів, побудова програми.	14.05.2023	Виконано
5.	Підготовка третього розділу роботи: форматування даних, підготовка мовної моделі, побудова класифікаторів, порівняння математичних моделей	21.05.2023	Виконано

6.	Підготовка четвертого розділу роботи: проведення функціонально-вартісного аналізу роботи	24.05.2023	Виконано
7.	Підготовка пояснювальної записки та презентації	28.05.2023	Виконано

Студент

_____ Нікіта СВИРИД

Керівник

_____ Людмила ЛЕВЕНЧУК

РЕФЕРАТ

Дипломна робота: 94 с., 21 рис., 9 табл., 2 додатки, 25 джерел.

ПРОГНОЗУВАННЯ, МОВНІ МОДЕЛІ, ГЕНЕРАЦІЯ ОЗНАК,
МАТЕМАТИЧНІ МОДЕЛІ КЛАСИФІКАЦІЇ, АНАЛІЗ НТТР ТРАФІКУ.

Дана робота присвячена розробці та реалізації моделі, що використовує методи мовного моделювання для вирішення задач класифікації шкідливих НТТР запитів. За результатами дослідження очікується отримати найбільш ефективну модель шляхом оцінки метрик якості.

Об'єкт дослідження – задача класифікації НТТР запитів за допомогою мовної моделі, спрямованих на веб-додаток. Дослідження охоплює різні аспекти класифікації, включаючи вибір відповідної мовної моделі, підготовку даних, навчання моделі, а також оцінку її продуктивності та точності.

Предмет дослідження – мовна модель RoBERTa для вирішення задачі генерування ознак, її математична складова, переваги та недоліки.

Мета дослідження – аналіз НТТР запитів, створення векторного простору для навчання класифікаторів, підбір моделей для класифікації.

Ключовою особливістю розробленого алгоритму є використання мовної моделі для вирішення задачі генерації ознак, які будуть подаватись на вхід математичних моделей. Мовна модель аналізує текстові дані, виявляє приховані патерни та тренди, а потім генерує нові ознаки, які можуть бути корисними для прогнозування та класифікації, що значно спрощує процес підготовки даних.

ABSTRACT

Bachelor Thesis: 94 p., 21 fig., 9 tabl., 2 append., 25 sources.

FORECASTING, LANGUAGE MODELS, FEATURES GENERATING,
CLASSIFICATION MODELS, HTTP TRAFFIC ANALYSIS.

This work is devoted to the development and implementation of a model that uses language modeling techniques to solve the problem of classifying malicious HTTP requests. Based on the results of the study, it is expected to obtain the most effective model by evaluating quality metrics.

The object of the study is the task of classifying HTTP requests using a language model directed to a web application. The research covers various aspects of classification, including selecting a suitable language model, preparing data, training the model, and evaluating its performance and accuracy.

The subject of the study is the RoBERTa language model for solving the problem of feature generation, its mathematical component, advantages and disadvantages.

The purpose of the study is to analyze HTTP requests, create a vector space for training classifiers, and select models for classification.

A key feature of the developed algorithm is the use of a language model to solve the problem of generating features that will be fed into mathematical models. The language model analyzes textual data, identifies hidden patterns and trends, and then generates new features that can be useful for forecasting and classification, which greatly simplifies the data preparation process.

ЗМІСТ

ЗМІСТ	7
ВСТУП.....	9
1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ.....	10
1.1 Огляд існуючих протоколів передачі даних	10
1.2 Хакерські атаки	17
1.3 Огляд програмного забезпечення для боротьби з хакерськими атаками	20
1.4 Висновки до першого розділу.....	22
2 ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ МОВНОЇ МОДЕЛІ ТА КЛАСИФІКАТОРІВ.....	24
2.1 Основні поняття та завдання NLP	24
2.1.1 Використання токенізатора для попередньої обробки даних	24
2.2.2 Огляд актуальних підходів до побудови мовних моделей	26
2.2 Огляд існуючих мовних моделей	30
2.3 Огляд методів класифікації текстових даних та оцінок якості моделі ..	33
2.4 Висновки до другого розділу	44
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ.	46
3.1 Опис наборів даних, використаних для навчання.....	46
3.2 Навчання моделі RoBERTa	48
3.3 Створення класифікаторів.....	50
3.4 Дослідження згенерованого векторного простору	51
3.5 Результати проведених експериментів, оцінка якості класифікаторів моделей за допомогою метрик якості	53

	8
3.6. Висновки до третього розділу.....	56
4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ	
.....	57
4.1 Постановка задачі проєктування	58
4.2 Обґрунтування функцій програмного продукту	59
4.3. Обґрунтування системи параметрів програмного продукту	62
4.4 Аналіз експертного оцінювання параметрів	64
4.5. Аналіз рівня якості варіантів реалізації функцій	67
4.6 Економічний аналіз варіантів розробки ПП	69
4.7 Вибір кращого варіанту ПП техніко-економічного рівня.....	75
4.8 Висновки до четвертого розділу.....	76
ВИСНОВКИ	77
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	79
ДОДАТОК А ЛІСТИНГ ПРОГРАМИ	81
ДОДАТОК Б ПРЕЗЕНТАЦІЯ.....	87

ВСТУП

У сучасному цифровому світі, де обмін інформацією здійснюється швидше і в більших обсягах, ніж будь-коли раніше, захист інформації стає вкрай важливим. Таким чином, розробка моделей для автоматизованої класифікації HTTP запитів є надзвичайно важливою задачею, що сприяє підвищенню рівня кібербезпеки.

Метою дослідження є аналіз HTTP запитів, створення векторного простору для навчання класифікаторів, підбір моделей для класифікації. Важливим аспектом є підбір оптимальних моделей для класифікації HTTP запитів, що включає експериментування з різними архітектурами нейронних мереж і підходами до обробки текстових даних.

Дана робота присвячена розробці та реалізації моделі, що використовує сучасні методи мовного моделювання для вирішення задач класифікації шкідливих HTTP запитів. Зокрема, буде застосована мовна модель яка дозволяє ефективно генерувати ознаки для текстових даних.

Предметом дослідження є використання мовної моделі RoBERTa для вирішення задачі генерування ознак HTTP запитів. Це включає вивчення математичної складової моделі, її архітектури та алгоритмів навчання.

Пояснювальна записка, складена до роботи, складається із чотирьох розділів. У першому розглянуто теоретичні складові протоколів передачі даних, також розглянуто актуальні види хакерських атак та розглянуті основні рішення для боротьби з ними, які існують та є актуальними сьогодні. У другому розділі глибоко розглянуто математичний підхід до побудови мовних моделей, окремо розглянуто внутрішню структуру класифікаторів. У третьому розділі наведено реалізацію алгоритму класифікації та проаналізовано отримані результати. Четвертий розділ присвячено функціонально-вартісному аналізу розробленого продукту.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд існуючих протоколів передачі даних

Будь який зв'язок між пристроями можливий завдяки протоколам передачі даних.

Загалом, протоколи можна поділити на фізичні та логічні. Перші встановлюють та регулюють можливість того, як саме та які сигнали будуть передаватися задля встановлення зв'язку між пристроями, наприклад, імпульсами сто разів щосекунди по 5 Вольт кожен або сигналами на конкретній частоті радіосигналу. Логічні ж протоколи відповідають передачу даних, коли зв'язок між пристроями вже встановлено.

Існують декілька видів протоколів. Наприклад, стандартизований протокол передачі даних дозволяє на фізичному рівні проводити розробку таких інтерфейсів, які не прив'язані до конкретної апаратної платформи чи виробника (наприклад, USB, Bluetooth).

Сигнальний протокол використовується для управління налагодженим з'єднанням, наприклад, його встановлення, розриву чи переадресації. Як приклад таких протоколів можна навести RTSP, SIP. Детальніше їх буде розглянуто нижче.

Мережевий протокол - набір правил та дій (алгоритму дій), що мають бути виконані задля втілення з'єднання та реалізації обміну даними між двома та більше пристроями, під'єднаними до мережі.

Часто різні протоколи описують лиш різні сторони одного типу зв'язку. Існує також поняття «стек протоколів», що описує та дає розуміння щодо програмного забезпечення, за допомогою якого реалізовано протокол.

У сучасному світі за існування такої незліченної кількості інформації та, відповідно, виникнення різносторонніх протоколів для їх передачі постало питання щодо їх стандартизації, визначення, запобігання коректності та класифікації. Нові протоколи у рамках усієї мережі Інтернет визначаються

IETF, інші ж - IEEE або ISO. Також телекомунікаційними протоколами та форматами поза мережею Інтернет займається ITU-T.

Найбільш поширеною системою задля класифікації мережевих протоколів є модель ISO (Open System Interconnection — взаємодія відкритих систем), у відповідності до якої протоколи поділяються за сімома рівнями за своїм призначенням, від фізичного, на якому відбувається формування та розпізнання електричних або будь-яких інших сигналів, до прикладного, де вибудовується інтерфейс програмування додатків для передачі інформації цими додатками. Протокол певного рівня визначає певне технічне правило зв'язку. Розглянемо головну модель міжнародного рівня для класифікації та визначення протоколів.

Модель OSI — семирівнева логічна модель коректної роботи мережі [1]. Дана система реалізується групою протоколів та правил вибудовування зав'язків, що організовані у декілька наступних рівнів.

1. На фізичному рівні проходить визначення фізичних характеристик ліній зв'язку (наприклад, електричних, оптичних, механічних тощо).

2. На канальному рівні визначаються правила використання попереднього, фізичного рівня, вузлами мережі.

3. На мережевому рівні відбувається адресація та доставка повідомлень.

4. На транспортному рівні контролюється послідовність проходження компонентів повідомлення.

5. На сеансовому рівні координується зв'язок між двома прикладними програмами, що працюють на різних робочих станціях.

6. На рівні представлення відбувається перетворення даних із внутрішнього формату комп'ютера у формат передачі даних.

7. Прикладний рівень є границею між прикладним додатком та іншими рівнями, тим самим, забезпечуючи зручний інтерфейс зв'язку для мережевих додатків користувача.

Але розглянута модель на даний момент не має широкого прикладного застосування користувачами у мережі. Тому розглянемо також іншу модель – стек протоколів TCP/IP, яка широко використовується та складається з більшої частини існуючих та широко використовуваних протоколів. Модель TCP/IP є 4-рівневою та вибудована наступним чином:

- канальний рівень;
- мережевий рівень;
- транспортний рівень;
- прикладний рівень.

Кожен рівень відповідно до назви виконує той же функціонал, що зазначено в описі багаторівневості моделі OSI.

Далі розглянемо найбільш відомі протоколи, які широко використовуються у мережі Інтернет.

1. HTTP (Hyper Text Transfer Protocol) – протокол передачі, що слідує з назви, гіпертексту. HTTP протокол широко використовується при пересиланні Web-сторінок між комп'ютерами, що мають бути під'єднаними до однієї мережі.

2. FTP (File Transfer Protocol) – протокол для передачі файлів зі спеціального файлового серверу на комп'ютер користувача. Цей протокол, що працює за схожим до HTTP принципом та забезпечує можливість обміну двійковими та текстовими файлами із будь-яким комп'ютером у мережі.

3. POP3 (Post Office Protocol) – стандартний протокол для поштового зв'язку. Сервери POP обробляють вхідну пошту, а протокол POP у той час займається обробкою запитів на отримання пошти від клієнтських поштових сервісів.

4. SMTP (Simple Mail Transfer Protocol) – протокол, який визначає набір правил для передачі пошти. Сервер SMTP працює просто – або повертає підтвердження про отримання, або повертає повідомлення про виникнення помилки, або ж повертає запит про необхідність отримання додаткової потрібної інформації.

5. DTN – протокол, призначений для мереж дальнього космічного зв'язку IPN, які використовуються NASA.

Як бачимо, існує велика кількість мережевих протоколів із абсолютно різними цілями та призначенням. Крім того, кожен із наведених протоколів та таких, що ще не були розглянуті, має свої переваги та недоліки. Також, на думку авторів, приділити увагу варто особливостям та властивостям кожного з протоколів, багато назв яких чув у повсякденному житті пересічний користувач. Тож, надалі, розглянемо детальніше протоколи мережевого зв'язку.

HTTP протокол є фундаментальною основою для функціонування усіх інтернет-ресурсів. Головним завданням даного протоколу є надання можливості запиту ресурсів, які можуть бути потрібними у віддаленій системі (наприклад, до файлів).

HTTP вважається клієнт-серверним протоколом, що означає що його відправка здійснюється тільки однією конкретною стороною. Головними особливостями цього протоколу можна назвати наступні:

- простота – алгоритми протоколу легко сприймаються людьми;
- розширюваність – звичайного зв'язку між клієнтами та сервером достатньо для того, щоб внести зміни у семантику;
- сесійність – за рахунок відсутності послідовного зв'язку між запитами, на кожній сесії можна ділитися визначеним контекстом.

HTTP протокол широко використовується, так як за його допомогою можна виконувати велику кількість задач, як-от, управління кешем, сесіями, аутентифікацією, послаблення обмежень джерел, тунелювання та проксі.

Однак, останнім часом, за рахунок загострення проблеми із захищеністю передачі даних, зростає тенденція до переходу сайтів із HTTP протоколів на HTTPS, що дозволяє збільшити рівень довіри користувача до інтернет-ресурсу.

IP (Internet Protocol) є маршрутизаторним протоколом на мережевому рівні TCP/IP. Завдяки IP стало можливим поєднати у всесвітню мережу різні комп'ютерні мережі. Головним завданням протоколу є доставка пакетів поміж різними маршрутизаторами. До важливих особливостей цього протоколу варто віднести:

- унікальність – для кожного комп'ютера мережі існують окремі IP-адреси, завдяки яким учасники мережі знаходять та ідентифікують один одного в Інтернеті;
- фрагментарність – IP протокол проходить через різні канали доставки та, у випадку перевищення можливостей конкретних вузлів мережі, передбачено дроблення пакетів;
- відсутність надійності – гарантію коректної доставки без помилок забезпечують на більш високих рівнях протоколів, у той час як IP протокол не виключає дублі, пошкодження, порушення порядку та навіть відсутність відправки.

Цей алгоритм реалізовано за допомогою таких видів як IPv4 та IPv6. У моделі TCP/IP цей протокол відноситься до мережевого рівня.

Протокол SSH (Secure Shell) реалізовано на рівні додатків та потрібен для того, щоб віддалено мати змогу управляти системою за допомогою захищеного каналу. Такий метод застосовується у роботі багатьох відомих технологій. Серед особливостей протоколу SSH варто виділити наступні:

- шифрування – особлива властивість SSH, що виділяє його з-поміж інших, реалізована за допомогою авторизації по ключу, тобто відбувається кодування всього можливого трафіку, у тому числі й паролів, за допомогою різних алгоритмів;

- безпечність – ця особливість слідує із попереднього пункту, так як завдяки шифруванню підвищується надійність віддаленої роботи;
- можливість стиснення – ця особливість особливо актуальна при передачі інформації.

Копіювання файлів по SSH дозволяє підвищити рівень захисту при передачі інформації. Даний протокол вважається протоколом прикладного рівня і його прямим призначенням являється забезпечення віддаленого доступу.

FTP (File Transfer Protocol) – один із найстаріших варіацій прикладного протоколу. Він забезпечує доступ до віддалених хостів та передачі програмного забезпечення. У цьому випадку, для більш точного розуміння цього протоколу, варто одразу розглянути його особливості:

- результативність – протокол гарантує або ж відправку або ж видачу помилки або ж повідомлення про необхідність додаткової інформації;
- варіативність шифрування – у різних випадках можливі різні підходи до шифрування, або анонімні підключення, або передача логінів та паролів відкритим текстом;
- вбудована аутентифікація – кожен користувач за замовчуванням обов'язково авторизується;
- множинність підключень – такий протокол застосовує щонайменше подвійне підключення.

Завдяки бінарному режиму передачі (аналогічна властивість є і у HTTP) зменшуються витрати трафіку та часу, необхідні для відправки та передачі великих з точки зору пам'яті файлів.

POP3 (Post Office Protocol Version 3) – стандартний варіант на прикладному рівні, який використовується користувачами email-сервісами. Головним завданням цього протоколу є забезпечення відправки пошти з віддаленого серверу за допомогою TCP-з'єднання. Також, окрім цього варіанту протоколу, для вилучення електронної пошти використовується

IMAP. Як правило, на сучасних поштових клієнтах реалізовано одразу обидва варіанти протоколу. Серед властивостей ж POP3 виділимо наступні:

- постійний доступ до листів, які збережені на комп'ютері, навіть за відсутності мережі;
- можливість відкривати вкладені файли одразу разом із листом;
- постійне очищення пам'яті ящика за рахунок того, що листи постійно вивантажуються локально на жорсткий диск комп'ютера.

Тим не менш, окрім очевидних переваг, також існує вагомий недолік – високий ризик зараження вірусами комп'ютерів користувачів. Також, при роботі з даним протоколом може знадобитися надто частий бекап.

MAC (Media Access Control) – протокол, що розміщено на низькому рівні. Його задача полягає у тому, щоб ідентифікувати пристрої локальних мереж. До особливостей MAC можна віднести:

- унікальність – для кожного пристрою наявна своя унікальна MAC-адреса, яка початково задана виробником пристрою;
- контроль - даний протокол контролює доступ до фізичної середовища передачі;
- захист від помилок – завдяки створенню та додатковій перевірці алгоритмів створюється додатковий захист.

Механізми управління доступом до каналу на MAC-рівні реалізовані як метод множинного доступу. Таким чином, одразу декілька станцій можуть застосовувати одну й ту ж фізичну середовище.

Таким чином, у зв'язку з швидким розвитком технологій та постійними загрозами безпеки, протоколи постійно вдосконалюються та змінюються, додавши нові функції та заходи безпеки для забезпечення конфіденційності, цілісності та доступності даних. Загалом, розуміння та ефективне використання протоколів передачі даних важливо для побудови надійних та безпечних мереж та забезпечення взаємодії між комп'ютерами та системами у сучасному цифровому світі.

1.2 Хакерські атаки

Хакерські атаки представляють собою дії, спрямовані на несанкціонований доступ до комп'ютерних систем, мереж або даних. Зловмисники використовують різні методи та інструменти для досягнення своїх цілей, які можуть включати викрадення інформації, порушення роботи систем, вимагання грошей або інші зловмисні дії. Кібератака може статися з кожним, тому розглянемо деякі з найпоширеніших типів хакерських атак та що робити для того, щоб їм запобігти.

1. Фішинг - вид атаки, яка використовує як вектор електронну пошту та обманом змушує користувачів загрузати зловмисні програми на власний компютер. Фішинг є частою проблемою до сьогоднішнього дня, особливо у робочій сфері, як у великих, так і малих компаніях. Існують наступні види фішингу:

- spear-phishing — атаки, що направлені на конкретних людей, наприклад, системних адміністраторів;
- whaling — атаки, що направлені на керівників високого рангу;
- smishing — атаки, що використовують текстові або SMS-повідомлення задля привернення уваги жертви;
- search engine phishing — атаки, що за допомогою SEO привертають увагу користувача до зловмисних сайтів методом просування таких вище під час пошуку;
- email phishing — атаки через електронну пошту;
- vishing — атака через голосову пошту.

При застосуванні фішингових атак зловмисники освідомлені як та під яку «маску» краще сховати посилання, наприклад, в електронному листі, щоб те спрацювало на користувача.

Задля ефективної профілактики фішингу недостатньо уваги кожного користувача. Має бути чітко налагодженою культура безпеки у рамках компанії, а також регулярні перевірки та тестування підготовленості працівників до подібних атак.

2. Malware-атаки - вид атак, що виконується за допомогою шкідливого програмного забезпечення, що включає віруси, троянські програми, шпигунські програми, програми-вимагачі (ransomware) тощо. Зловмисне ПЗ здатне викрадати дані, пошкоджувати системи або навіть вимагати викуп за розблокування файлів. Зазвичай таке ПЗ розповсюджується за допомогою спама чи соціальної інженерії. Важливо розуміти, що навіть при виплаті викупу зловмисникам користувач може так і не отримати ніколи доступу до своїх даних. У таких випадках, задля профілактики подібних серйозних наслідків варто створювати резервні копії даних.

3. DDoS-атаки (distributed denial-of-service attack) - вид атак, спрямованих на навмисне перевантаження серверів або мереж величезною кількістю запитів, що робить їх недоступними для дійсних користувачів. Такий ефект досягається за допомогою скоординованих дій багатьох пристроїв. Щоб запобігти подібному ураженню важливо бути певним, що використовуються хмарні веб-сервіси, які здатні перешкоджати переповненню об'єму трафіку, регулярно проводити тестування систем безпеки, оновлювати програмне забезпечення та пропускну здатність, а також співпрацювати із надійними та безпечними компаніями, що здатні запропонувати рішення для пом'якшення результатів подібних атак.

4. SQL-ін'єкції (SQL Injections) - використовують вразливості у веб-додатках, які працюють з базами даних. Зловмисники вводять шкідливі SQL-запити через веб-форми або URL-адреси, що дозволяє їм отримати несанкціонований доступ до баз даних або маніпулювати даними.

5. XSS-атаки - використовують вразливості у веб-додатках для впровадження шкідливих скриптів. Ці скрипти виконуються в браузері

користувача і можуть викрадати дані сесії, змінювати контент веб-сторінок або перенаправляти користувачів на шкідливі сайти.

6. Атаки на основі експлойтів - експлойти використовують відомі вразливості у програмному забезпеченні для виконання шкідливих дій. Це можуть бути вразливості в операційних системах, браузерях, додатках або мережевих протоколах.

7. Brute-force атаки - атаки, що спрямовані на підбір паролів шляхом систематичного перебору всіх можливих комбінацій. Якщо використовуються слабкі або передбачувані паролі, атака може бути успішною.

8. Атаки через соціальну інженерію - атака, під час якої зловмисники використовують психологічні прийоми для маніпуляції людьми, щоб змусити їх розкрити конфіденційну інформацію або виконати певні дії. Це можуть бути телефонні дзвінки, електронні листи або особисті зустрічі.

9. Внутрішні загрози (Insider Threats) - атаки, що здійснюються співробітниками або іншими особами, які мають легальний доступ до систем і даних, але використовують цей доступ у зловмисних цілях. Це може бути викрадення інформації, саботаж або шпигунство.

10. Атаки типу "людина посередині" (Man-in-the-Middle) - уатаки, під час яких зловмисник перехоплює та, можливо, змінює передані дані між двома сторонами, які не підозрюють про його присутність. Це може призвести до викрадення конфіденційної інформації або подробиць комунікацій.

Для захисту від хакерських атак необхідно використовувати комплексні підходи, які включають:

- використання антивірусного та антишпигунського програмного забезпечення;
- регулярні оновлення програмного забезпечення для закриття відомих вразливостей;

- використання сильних, унікальних паролів і багатофакторної аутентифікації;
- шифрування даних для захисту від перехоплення;
- проведення навчання співробітників з питань безпеки;
- впровадження політик безпеки та моніторинг підозрілої активності.

Хоча повністю захиститися від усіх можливих загроз неможливо, дотримання цих заходів може значно знизити ризик успішності хакерських атак.

1.3 Огляд програмного забезпечення для боротьби з хакерськими атаками

Зважаючи на зростаючу складність кіберзагроз та швидкість їх поширення, сучасні інструменти для мережевої безпеки стають все більш важливими для захисту корпоративних мереж, інформації та активів.

Одним із ключових аспектів є мережевий периметр, де використання міжмережевих екранів (firewalls) виявляється критичним. Сучасні міжмережеві екрани не лише контролюють трафік, але і здатні аналізувати його для виявлення потенційно небезпечних дій.

Значну роль у виявленні атак відіграють системи виявлення та запобігання вторгнень (IDS/IPS). Вони стежать за активністю в мережі, виявляючи аномалії, характерні для атак, і намагаються перешкодити їм. Приклад такої програми наведено на рис. 1.1.

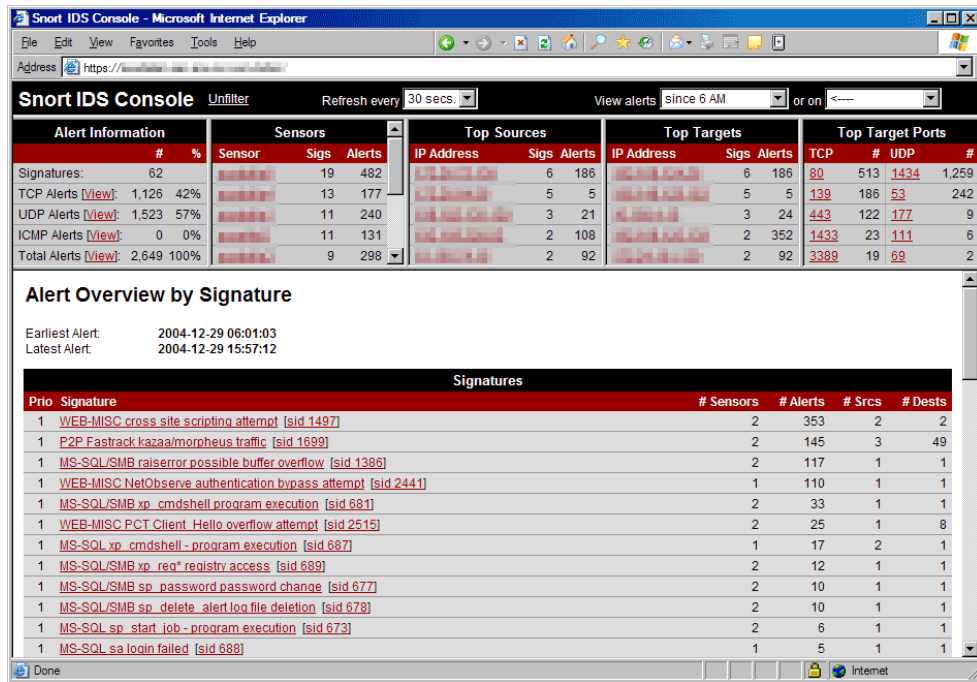


Рисунок 1.1 – Вікно програми для боротьби з вторгненнями SNORT [2]

Крім того, існують рішення для захисту від DDoS-атак, що забезпечують неперервність роботи веб-сайтів та мережесервісів під час масштабних атак. Принцип роботи програмного забезпечення для захисту від такого виду атак подано на рис. 1.2.

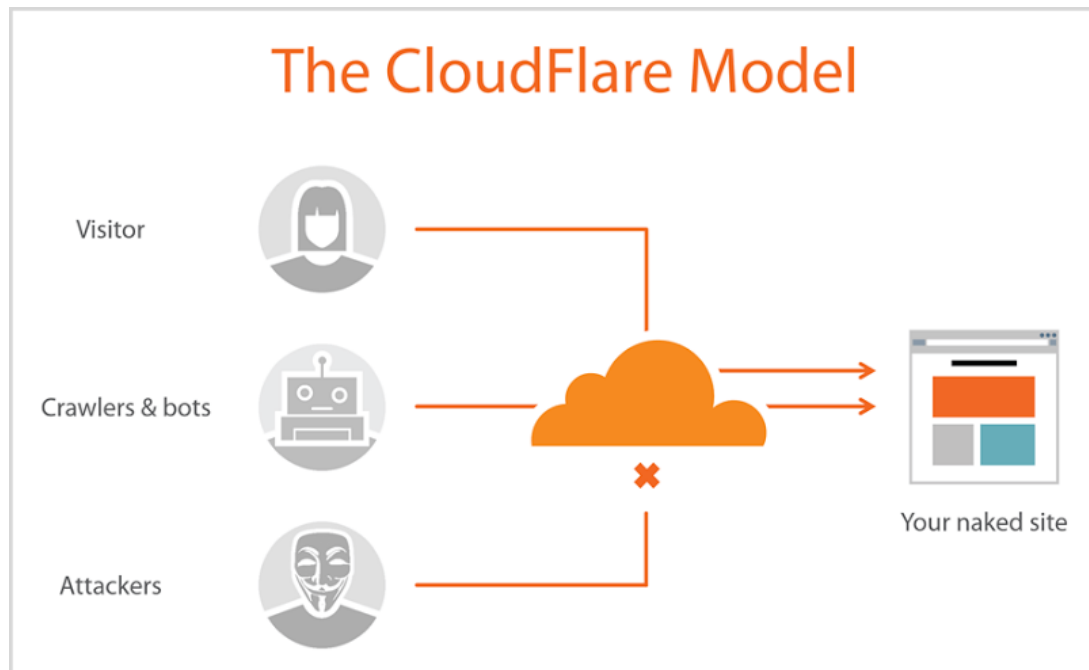


Рисунок 1.2 – Принцип роботи ПЗ для захисту від DDoS атак CloudFlare [3]

Для захисту кінцевих точок (комп'ютерів, ноутбуків, мобільних пристроїв) існують платформи захисту кінцевих точок (EPP). Вони надають широкий спектр захисту, включаючи антивірусний захист, запобігання витоку даних та контроль використання програм.

І, нарешті, системи управління вразливостями та моніторингу мережі допомагають виявляти та усувати потенційні вразливості в мережевих системах, забезпечуючи високий рівень безпеки.

1.4 Висновки до першого розділу

Отже, було розглянуто види сучасних протоколів передачі даних, преведні різні типи хакерських атак, які уснують в світі та розглянуто актуальне програмне забезпечення для боротьби з ними.

По-перше, розуміння протоколів передачі даних є ключовим для побудови безпечних та ефективних мереж. Протоколи, такі як TCP/IP, HTTP, FTP та інші, відіграють критичну роль у забезпеченні зв'язку між пристроями та системами в мережі, тому розуміння їхньої роботи та застосування відповідних заходів безпеки важливо для запобігання атакам.

По-друге, аналіз та класифікація HTTP-запитів на вредоносні та нормальні є важливим аспектом захисту веб-додатків та сервісів. Хакерські атаки часто використовують HTTP-протокол для впровадження шкідливого коду або отримання конфіденційної інформації.

Нарешті, програмне забезпечення для боротьби з хакерськими атаками відіграє критичну роль у захисті мереж та інформації. Використання сучасних інструментів допомагає виявити та відповісти на різноманітні загрози.

Таким чином, створення конкурентноспроможного програмного забезпечення для класифікації HTTP-запитів може допомогти виявляти та

блокувати потенційно небезпечний трафік, забезпечуючи високий рівень захисту.

2 ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ МОВНОЇ МОДЕЛІ ТА КЛАСИФІКАТОРІВ

2.1 Основні поняття та завдання NLP

Однією з важливих галузей штучного інтелекту є обробка природної мови (Natural Language Processing, NLP). Сутність цього підходу полягає в тому, щоб створити модель, яка здатна розуміти та обробляти людську мову.

Щоб математичні моделі могли ефективно працювати з текстовими даними, необхідно перетворити ці дані у формат, з яким модель може працювати – числові послідовності. Цей процес називається токенизацією.

2.1.1 Використання токенизатора для попередньої обробки даних

Токенізатори розбивають текст на менші одиниці – токени (слова, частини слів або символи), і присвоюють кожному токenu унікальний числовий ідентифікатор. Таким чином, текстовий вхід перетворюється в послідовність чисел, яка може бути оброблена математичною моделлю. Це забезпечує необхідний зв'язок між текстовими даними та моделлю, дозволяючи здійснювати аналіз, обробку та генерацію тексту на високому рівні.

На даний момент існують три види токенизаторів.

1. “Word-level” токенизатори – розбивають текст на окремі слова. Вони є простими у використанні, але може бути неефективним для мов з великою кількістю слів або для слів, яких модель раніше не бачила.

2. “Subword-level” токенизатори – розбивають слова на менші частини (морфеми або навіть символи). Цей підхід дозволяє моделі краще

справлятися з рідковживаними або новими словами, зберігаючи при цьому обмежений розмір словника.

3. “Character-level” токенизатори – розбивають текст на окремі символи. Найбільш гнучкий, але може призвести до довгих послідовностей токенів і ускладнити навчання моделі.

Для досягнення компромісної швидкості та ефективності будемо застосовувати Subword-level токенизацію. Цей підхід є збалансованим, оскільки зменшує довжину послідовностей порівняно з character-level токенизацією і зберігає здатність працювати з рідковживаними словами та частинами слів, що є ключовим при виявленні шкідливих HTTP запитів.

Одним з найбільш поширених алгоритмів Subword-level токенизації є Кодування пар байт (Byte Pair Encoding, BPE). Алгоритм був вперше описаний у 1994 році Філіпом Гейджем для кодування рядків тексту в табличну форму для використання в подальшому моделюванні [4]. Процес роботи BPE розпочинається з розбиття всього тексту на окремі символи. Кожен символ на початковому етапі вважається окремим токеном. Далі підраховуються всі можливі пари символів в тексті. Найбільш часті пари символів об'єднуються в новий токен. Цей процес повторюється: знову підраховуються частоти пар, знаходиться найчастіша пара і об'єднується. Критеріями зупинки для цього алгоритму можуть бути:

- досягнення заданого числа токенів;
- досягнення максимальної кількості ітерацій;
- досягнення бажаного розміру словника.

Для досягнення максимальної ефективності моделі, існує покращений метод – Byte-Level BPE. Основна ідея полягає у тому, щоб базуватись на словнику, який складається з байтів. Це дає набагато компактніший словник, ніж словник на основі символів, без втрати продуктивності. При однаковому розмірі словника BBPE сегментує речення на коротші послідовності, ніж

символьні методи, що призводить до швидкого навчання та висновків. BBPE корисний для обробки текстів, які можуть містити спеціальні символи або мовні символи, що виходять за межі стандартного набору ASCII. Для вирішення задачі класифікації HTTP запитів це може бути вирішальним, оскільки нерідко у шкідливих запитах можна зустріти такі спеціальні символи.

2.2.2 Огляд актуальних підходів до побудови мовних моделей

Сучасні підходи до побудови мовних моделей базуються на глибокому навчанні та використовують складні архітектури нейронних мереж. Довгий час найбільш популярним підходом було використання рекурентних нейронних мереж або їх вдосконалених варіантів, таких як Long Short-Term Memory (LSTM) або Gated Recurrent Unit (GRU). Ці архітектури дозволяють моделі "запам'ятовувати" попередні інформацію та використовувати її для прогнозування наступного слова в послідовності.

Сьогодні, все більш популярними стають трансформерні нейронні мережі. Трансформер – це архітектура нейронної мережі, яка вперше була представлена в статті "Attention is All You Need" в 2017 році [5]. Вона стала однією з ключових технологій у сфері обробки природної мови та інших областях, де необхідно моделювати послідовності даних.

З точки зору структури, трансформери виглядають аналогічно до енкодер-декодеру на основі рекурентних нейронних мереж. Тобто, на вхід енкодеру подається послідовність на вхідній мові. Ці послідовності представлені у вигляді векторів, які пройшовши через слої самоуваги змінюються з огляду на контекст речення загалом.

Енкодер складається з декількох ідентичних шарів (звичайно 6 або 12). Кожен шар містить два підшари.

1. Механізм багатоголової уваги (Multi-Head Self-Attention). Цей механізм дозволяє кожному слову в послідовності звертати увагу на всі інші слова. Він обчислює увагу за допомогою декількох "голів" (attention heads), що дозволяє моделі захоплювати різні типи залежностей в даних.

2. Повнозв'язаний шар (Feed-Forward Network). Після механізму уваги кожне слово проходить через двошаровий повнозв'язаний нейронний шар з активаційною функцією, що додає нелінійності до моделі.

Крім того, кожен підшар має механізм залишкового з'єднання та нормалізацію, що допомагає стабілізувати навчання та попереджає затухання градієнтів.

Декодер також складається з декількох ідентичних шарів (звичайно 6 або 12), які мають три підшари.

1. Механізм багатоголової уваги до власної послідовності (Masked Multi-Head Self-Attention). Цей механізм дозволяє кожному слову у вихідній послідовності звертати увагу на попередні слова. Маскування запобігає доступу до майбутніх слів під час тренування.

2. Механізм багатоголової уваги до виходу енкодера (Multi-Head Attention to Encoder Output). Цей механізм дозволяє кожному слову у вихідній послідовності звертати увагу на всю вхідну послідовність. Він забезпечує з'єднання між енкодером і декодером.

3. Повнозв'язаний шар (Feed-Forward Network). У цьому шарі, аналогічно енкодеру, кожне слово проходить через повнозв'язаний шар з функцією активації.

Окремо варто розглянути те, як працює механізм багатоголової уваги. Його основна задача – обчислення ваг (attention weights), які показують, наскільки важливим є кожне слово у вхідній послідовності для поточного слова у вихідній послідовності.

На практиці ми обчислюємо функцію уваги на множині запитів одночасно, упакованих разом у матрицю Q . Ключі та значення також упаковані у матриці K та V . Матрицю виходів обчислюється за наступною формулою:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

де Q – запити, K – ключі, V – значення, а d_k – розмірність запитів та ключів.

Замість того, щоб використовувати одну функцію уваги з ключами, значеннями та запитами розмірності d_{model} , більш ефективним підходом для вирішення цієї задачі є корисним лінійно спроектувати запити, ключі та значення h разів з різними, вивченими лінійними проекціями до розмірностей d_q , d_k та d_v відповідно. На кожну з цих спроектованих версій запитів, ключів і значень потім застосовується функцію уваги, отримуючи d_v -вимірні вихідні значення. Для кожного слова обчислюються ваги уваги до всіх інших слів у реченні, використовуючи окремі голови уваги. Наприклад, одна голова може фокусуватися на граматичній структурі, тоді як інша може звертати увагу на семантичні зв'язки між словами. Отримані зважені значення об'єднуються і проходять через лінійне перетворення для отримання остаточного представлення кожного слова. Це можна представити наступною формулою:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O,$$

де

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V).$$

Тут, проекції є матрицями параметрів $W_i^Q \in R^{d_{\text{model}} \times d_k}$, $W_i^K \in R^{d_{\text{model}} \times d_k}$, $W_i^V \in R^{d_{\text{model}} \times d_k}$ та $W^O \in R^{hd_v \times d_{\text{model}}}$.

Також, крім механізму багатоголової уваги кожен шар містить енкодера та декодера містить в собі повнозв'язну нейронну мережу прямого поширення зв'язку. Нейронна мережа прямого поширення зв'язку (FNN) - це тип штучної нейронної мережі, в якій інформація рухається тільки в одному напрямку, від вхідного шару через будь-які приховані шари і, нарешті, до вихідного шару.

Це найпростіший тип штучної нейронної мережі. У мережі немає циклів або петель. Термін "прямий" означає, що зв'язки між блоками не утворюють циклу, на відміну від більш складних типів мереж. В нашому випадку, мережа складається з двох прихованих шарів та функції активації ReLU.

Функція активації ReLU (Rectified Linear Unit) є однією з найпопулярніших і найбільш часто використовуваних функцій активації в сучасних нейронних мережах, особливо у глибокому навчанні. ReLU визначається наступною формулою:

$$ReLU(x) = \max(0, x).$$

Тобто, з формули зрозуміло, що ReLU "виправляє" всі негативні значення до нуля, а позитивні залишає незмінними. Обчислення ReLU є дуже простим і швидким, оскільки воно вимагає лише порівняння та максимізації. Це робить ReLU обчислювально ефективною функцією активації, що є важливим для навчання великих нейронних мереж. Графік даної функції активації наведено на рис. 2.1.

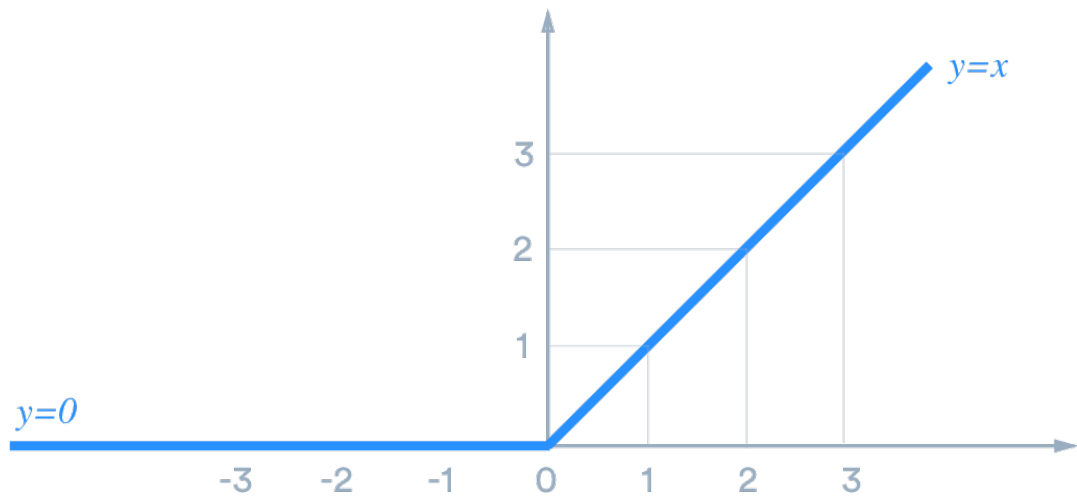


Рисунок 2.1 – Графік функції активації ReLU

2.2 Огляд існуючих мовних моделей

Переднавчені моделі є потужним інструментом, що дозволяє швидко і ефективно вирішувати складні завдання машинного навчання. Вони знижують необхідність у великих обчислювальних ресурсах і значно скорочують час, потрібний для отримання високоякісних результатів. Завдяки своїй універсальності та ефективності, переднавчені моделі знайшли широке застосування в багатьох сферах науки і техніки.

Переднавчені моделі можна використовувати двома основними способами:

1. Feature Extraction (витяг характеристик) - переднавчену модель використовують для витягу характеристик з даних, які потім передаються на вхід іншої моделі.

2. Fine-Tuning (тонке налаштування): - переднавчену модель адаптують до конкретного завдання шляхом додаткового тренування на меншому, специфічному для цього завдання наборі даних. Це дозволяє зберегти знання, отримані під час початкового тренування, і адаптувати модель до нових умов.

Завдяки появі трансформерів, сьогодні ми маємо два найбільш поширених класи переднавчених моделей для вирішення задач мовного моделювання. Розглянемо їх окремо.

BERT (Bidirectional Encoder Representations from Transformers) є передовою архітектурою нейронних мереж для обробки природної мови, розробленою Google [6]. Вона базується на трансформерах, які використовують механізм уваги для фокусування на важливих частинах вхідного тексту незалежно від їх позиції. Головною особливістю BERT є двонаправленість: модель аналізує контекст слів як зліва, так і справа, що дозволяє їй краще розуміти значення слів у різних контекстах.

Вхід до моделі складається з токенів, що включають спеціальні маркери CLS та SEP. Кожен токен представлений сумою трьох векторів: позиційного кодування, вектору токена та вектору сегменту. Основна частина BERT складається з декількох шарів енкодера трансформера з механізмами самоуваги, що дозволяють токенам взаємодіяти один з одним. У стандартній версії BERT-base використовуються 12 шарів енкодера, а в BERT-large – 24.

BERT проходить попереднє навчання за допомогою двох методів: масковане мовне моделювання (MLM), де частина токенів маскується і модель передбачає їх значення, та передбачення наступного речення (NSP), де модель визначає, чи є два речення послідовними. Після попереднього навчання BERT можна адаптувати до конкретних завдань, таких як класифікація текстів або виявлення сутностей, шляхом тонкого налаштування на відповідних наборах даних. Завдяки своїй двонаправленій природі та ефективному використанню трансформерів, BERT значно покращив точність багатьох задач обробки природної мови.

Також існують оптимізовані варіанти архітектури BERT, а саме RoBERTa, ALBERT та DistilBERT [7].

Кожен з них вносить свої покращення в оригінальну архітектуру, а саме:

- в RoBERTa оптимізовано процес навчання, збільшено обсяг даних і тривалість навчання, використовується більше міні-пакетів та динамічне маскування токенів. Відмова від завдання передбачення наступного речення (NSP). Ці зміни дозволили досягти кращих результатів у багатьох задачах обробки природної мови порівняно з оригінальним BERT;
- в ALBERT зменшено обчислювальні витрати та розмір моделі за рахунок факторизації ембедінгів та параметричного шару спільного використання між шарами трансформера. Заміна завдання NSP більш складним завданням передбачення взаємозв'язку речень. Це дозволило значно скоротити кількість параметрів, зменшити об'єм пам'яті та прискорити навчання;
- у DistilBERT створено компактнішу та швидшу версію BERT з мінімальними втратами в точності за допомогою техніки дистиляції знань. Модель має близько 60% параметрів від BERT, працює вдвічі швидше, зберігаючи близько 97% точності в задачах обробки природної мови.

Таким чином, RoBERTa покращує продуктивність, ALBERT оптимізує розмір і обчислювальні витрати, а DistilBERT забезпечує компактність і швидкість з незначними втратами точності.

Архітектура GPT (Generative Pre-trained Transformer), розроблена компанією OpenAI, базується на трансформерній архітектурі з акцентом на генерацію тексту [8]. Відмінною рисою GPT є використання однонаправлених трансформерів, де кожен токен у послідовності прогнозується на основі всіх попередніх токенів, що забезпечує ефективну генерацію текстових послідовностей через передбачення наступного слова в контексті.

Процес навчання GPT складається з двох етапів: попереднє навчання та тонке налаштування. Під час попереднього навчання модель навчається на великому корпусі текстів, передбачаючи наступне слово в реченні, що

дозволяє їй засвоїти загальні мовні закономірності. На етапі тонкого налаштування модель адаптується до конкретних задач або наборів даних, що підвищує її продуктивність у спеціалізованих завданнях.

Порівняння різних версій GPT демонструє поступове вдосконалення моделей. GPT-1 стала першою версією, яка продемонструвала ефективність підходу з попереднім навчанням та тонким налаштуванням. GPT-2 значно перевершила свого попередника завдяки збільшенню розміру моделі та обсягу даних для навчання, що дозволило їй генерувати більш зв'язний і змістовний текст. Проте, ця версія викликала занепокоєння щодо потенційного зловживання, оскільки могла генерувати реалістичний, але неправдивий контент.

GPT-3 здійснила черговий значний стрибок вперед, ставши найбільшою моделлю з понад 175 мільярдами параметрів. Це забезпечило їй здатність виконувати широкий спектр завдань без необхідності спеціального тонкого налаштування, використовуючи лише підказки, що робить її надзвичайно універсальною. Вона може відповідати на питання, писати есе, створювати програмний код і навіть брати участь у діалогах з високим рівнем контекстуального розуміння.

Основні відмінності між версіями GPT полягають у розмірах моделей та обсягах даних для навчання, що безпосередньо впливає на їх здатність генерувати якісний текст і виконувати різноманітні завдання. Кожна нова версія покращує попередню, підвищуючи точність, зв'язність.

2.3 Огляд методів класифікації текстових даних та оцінок якості моделі

Комбінований підхід, що використовує виходи мовних моделей для класифікації тексту за допомогою традиційних методів, поєднує в собі

найкращі властивості обох підходів: потужні представлення тексту від нейронних мереж та ефективність і простоту традиційних класифікаторів. Це дозволяє досягти високої точності класифікації при збереженні гнучкості, інтерпретованості та ефективності використання обчислювальних ресурсів.

Строго кажучи, в якості класифікаторів можна обрати будь яку наявну математичну модель. Але, було обрано моделі, які, на нашу думку, найбільш точно відображають векторний простір слів та речень. До таких моделей відноситься логістична регресія, випадковий ліс та метод опорних векторів. Детально розглянемо кожен з обраних моделей.

Логістична регресія є потужним інструментом для класифікації, який використовує просту лінійну комбінацію ознак та логістичну функцію для моделювання ймовірностей [9].

Лінійна комбінація ознак z пропускається через логістичну функцію, яку також називають сигмоїдою, щоб отримати ймовірність того, що подія належить до класу 1. Формула для обчислення логістичної функції наведена нижче:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Очевидно, що виходячи з основних властивостей ймовірності [10], ймовірність того, що ознака належить до класу 0 можна визначити за наступною формулою:

$$P(y = 0 | x) = 1 - \sigma(z).$$

Також, звернемо увагу на особливість функції втрат. Для навчання логістичної регресії необхідно визначити функцію правдоподібності:

$$L(w) = P(y|x; w) = \prod_{i=1}^n P(y_i|x_i; w) = \prod_{i=1}^n (\phi(z_i)^{y_i})(1 - \phi(z_i))^{1-y_i}.$$

На практиці легше максимізувати логарифм цього рівняння, який називається функцією логарифмічної правдоподібності:

$$l(w) = \log L(w) = \sum_{i=1}^n \log(\phi(z_i)) + (1 - y_i) \log(1 - \phi(z_i)).$$

З огляду на те, що наша задача полягає у мінімізації помилки нашої моделі, будемо розглядати обернену задачу оптимізації, шляхом множення функції логарифмічної правдоподібності на -1. Таким чином, ми отримаємо оцінку того, наскільки добре передбачає реальні значення.

Хочемо зауважити, що логістична регресія погано працює у випадку лінійної несепарабельності даних, оскільки основна суть цієї моделі полягає у тому, щоб розділити простір $\mathbb{R}^n$ єдиною гіперплощиною на два підпростори, які будуть відповідати класам 0 та 1 відповідно [11]. Приклад розподілення лінійно сепарабельних та несепарабельних даних наведено на рис. 2.2.

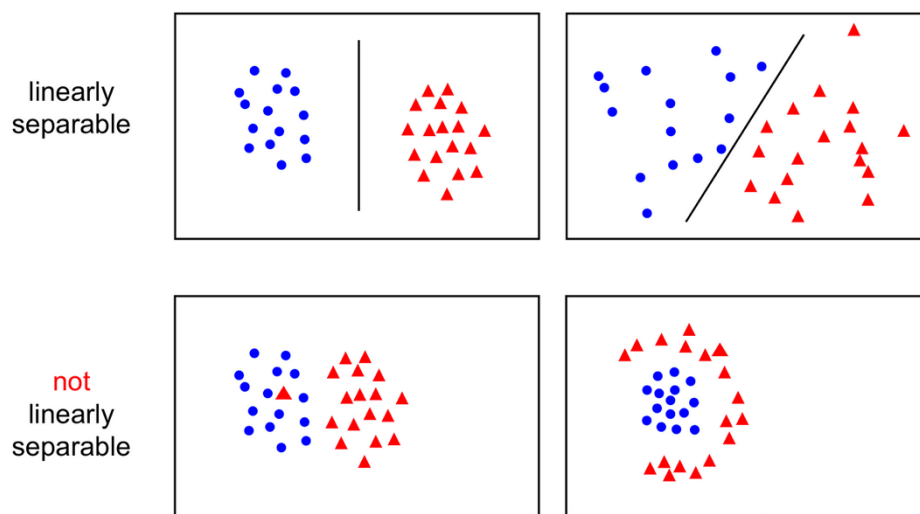


Рисунок 2.2 – Приклад лінійно сепарабельних та несепарабельних даних

Перейдемо до наступної з запропонованих моделей, а саме до випадкового лісу. Випадковий ліс відноситься до ансамблевих методів навчання. Це означає, що ця модель є поєднанням більш простих моделей, а саме з дерев прийняття рішень, що зрозуміло з назви.

Розглянемо принципи роботи дерева прийняття рішень. Починаючи з кореневого вузла, алгоритм обирає ознаку та поріг, який найкращим чином розділяє вибірку на дві частини. Модель робить це звертаючи увагу на ознаку, яка дає найбільший приріст інформації (information gain – IG). Після цього відбувається ітераційний процес розбиття у вузлів дерева, критерієм зупинки для якого буде “чистота” всіх вузлів. В даному контексті “чистота” означає те, що в кожному вузлі всі приклади будуть належати одному й тому ж самому класу. Звернемо увагу на те, як модель максимізує приріст інформації.

Визначимо information gain за допомогою наступної формули:

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^n \frac{N_j}{N_p} I(D_j),$$

де f – ознака для виконання розбиття, D_p та D_j – набори даних батьківського та j -го дочірнього вузла, I – міра “забрудненості” даних, N_p та N_j – кількість ознак батьківського та j -го дочірнього вузла відповідно.

Таким чином, ми можемо використати будь яку міру забрудненості даних, але найчастішими двома є ентропія Шеннона та критерій Джині [12].

Ентропія Шеннона визначається для системи з N можливими станами наступним чином:

$$I_e = - \sum_{i=1}^N p_i \log_2 p_i,$$

де p_i – ймовірність знаходження системи у i -тому стані. Ентропія може бути описана як ступінь хаосу в системі. Чим вища ентропія, тим менш впорядкована система, і навпаки. Таким чином, ентропія буде дорівнювати нулю, якщо всі ознаки у вузлі будуть належати одному класу та максимальною при рівномірному розподілі класів.

Забрудненість Джині можна розуміти як критерій мінімізації ймовірності неправильної класифікації та відобразити наступною формулою:

$$I_G = \sum_{i=1}^N p_i(1 - p_i) = 1 - \sum_{i=1}^N p_i^2.$$

Насправді, на практиці, інформаційна ентропія та забрудненість Джині видають приблизно однакові результати, тому, частіше за все, вони є взаємозамінними [13].

Перейдемо до поєднання випадкових дерев в ансамбль. Випадковий ліс створює кілька різних дерев прийняття рішень під час навчання. Кожне дерево T у випадковому лісі будується шляхом рекурсивного розділення вибірки на декілька підвибірок. Розділення відбувається за допомогою порогового значення v для ознак x_j . Цей алгоритм зображено на рис. 2.3.

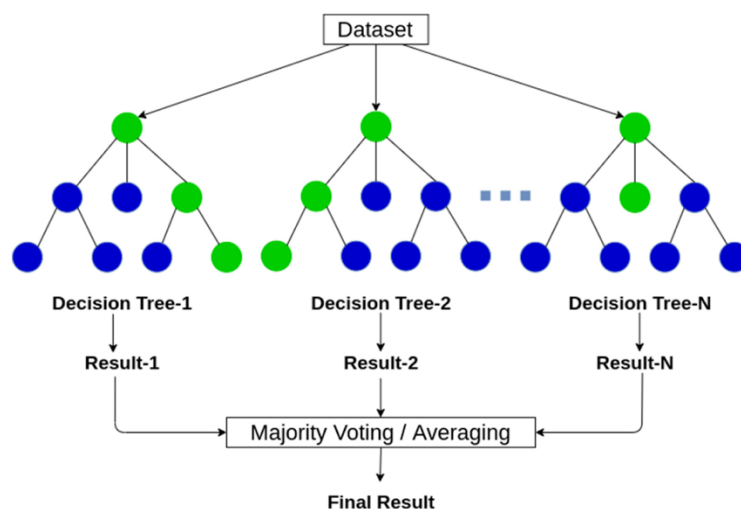


Рисунок 2.3 – Схема роботи випадкового ліс

Наведемо формулу для розділення вузла у дереві T :

$$split(m, j, v) = \{x \mid x_j \leq v\},$$

де j – індекс ознаки, v – порогове значення.

Після побудови всіх дерев в ансамблі, модель відносить об'єкт класифікації до конкретного класу шляхом визначення того, якому класу було надано більшість голосів.

Тобто, математично цей процес можна описати формулою:

$$\hat{y}(x) = mode(T(x)).$$

Хоча випадковий ліс є ефективним методом машинного навчання і має багато переваг, він також має свої недоліки:

- схильність до перенавчання: на великих наборах даних або з великою кількістю дерев в ансамблі може виникнути перенавчання. Це може призвести до того, що модель буде добре виконувати на тренувальних даних, але гірше справлятися з новими даними;
- випадковий ліс має декілька гіперпараметрів, таких як кількість дерев, глибина дерев і мінімальна кількість об'єктів у листях, які потребують налаштування для досягнення оптимальної продуктивності;
- побудова і використання великого числа дерев може вимагати значних обчислювальних ресурсів і часу, особливо якщо набір даних великий.

Хоча випадковий ліс має свої недоліки, він залишається дуже популярним і ефективним методом машинного навчання, особливо в задачах класифікації.

Наступною розглянутою моделлю буде метод опорних векторів. Основна ідея методу опорних векторів ґрунтується на пошуку гіперплощини,

яка буде найкраще розділяти класи в просторі ознак [14]. Її також називають оптимальною розділювальною гіперплощиною, приклад якої наведено на рис. 2.4.

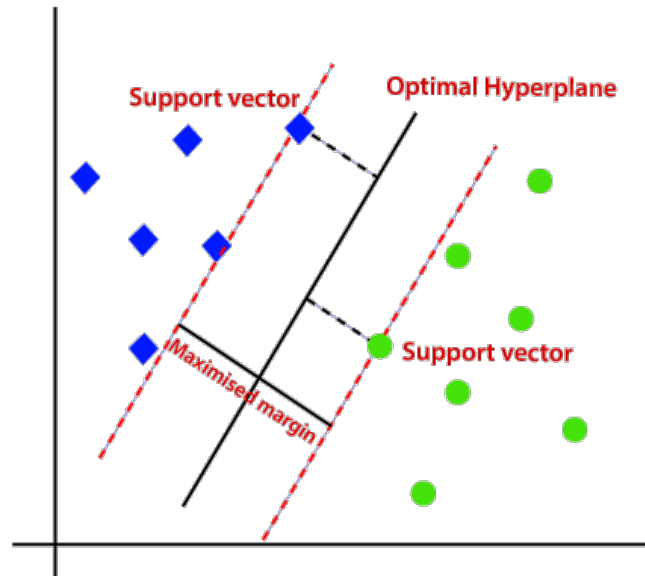


Рисунок 2.4 – Приклад розділювальної гіперплощини

Вибрана гіперплощина повинна максимально відділяти класи, тобто максимізувати відстань між найближчими точками різних класів, які називаються опорними векторами.

Для лінійно роздільних випадків пошук максимального зазору зводиться до вирішення задачі оптимізації:

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

при початкових обмеженнях:

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$

де w – вектор ваг, b – зміщення, x_i – вектор ознак для i -го зразка, y_i – мітка класу для i -го зразка (1 або -1).

Перевагою SVM є те, що ця модель може працювати з лінійно нероздільними даними. В цьому випадку застосовується ядерний трюк, приклад якого продемонстровано на рис. 2.5.

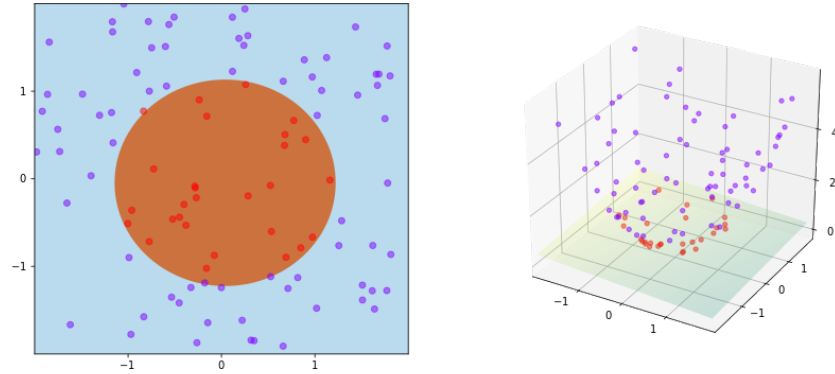


Рисунок 2.5 – Приклад застосування ядерного трюку

Основна ідея полягає у тому, щоб проектувати простір даних в простори вищих розмірностей [15]. Для цього необхідно замінити скалярний добуток $x_i^T x_j$ на, так звану, ядерну функцію:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j).$$

Одним з найбільш розповсюджених ядер є ядро на основі радіально-базисної функції (Radial Basis Function, RBF), яке називається гауссовим ядром:

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right),$$

де $\gamma = \frac{1}{2\sigma^2}$ є вільним параметром який підлягає оптимізації.

Наступним кроком при побудові математичних моделей є оцінка їх якості. Це є надзвичайно важливим для розуміння продуктивності та вибору найкращої моделі для конкретної задачі. Метрики оцінювання моделей класифікації дають кількісну оцінку ефективності моделі. Вибір метрики

оцінки залежить від конкретної задачі та важливості помилкових спрацювань і хибних спрацювань. Дуже важливо ретельно вибирати відповідну метрику (метрики) для оцінки ефективності моделі класифікації. Наведемо найбільш популярні метрики якості, які застосовуються в області машинного навчання сьогодні.

Матриця невідповідностей (Confusion Matrix) є одним з найбільш універсальних підходів оцінювання якості моделей. Вона являє собою матрицю розмірності 2x2 та дозволяє детально оцінити продуктивність класифікаційної моделі, відображаючи кількість істинно позитивних (TP), істинно негативних (TN), хибнопозитивних (FP) та хибнонегативних (FN) результатів. Зазвичай такі матриці зображують у вигляді теплової карти, приклад якої наведено на рис. 2.6.

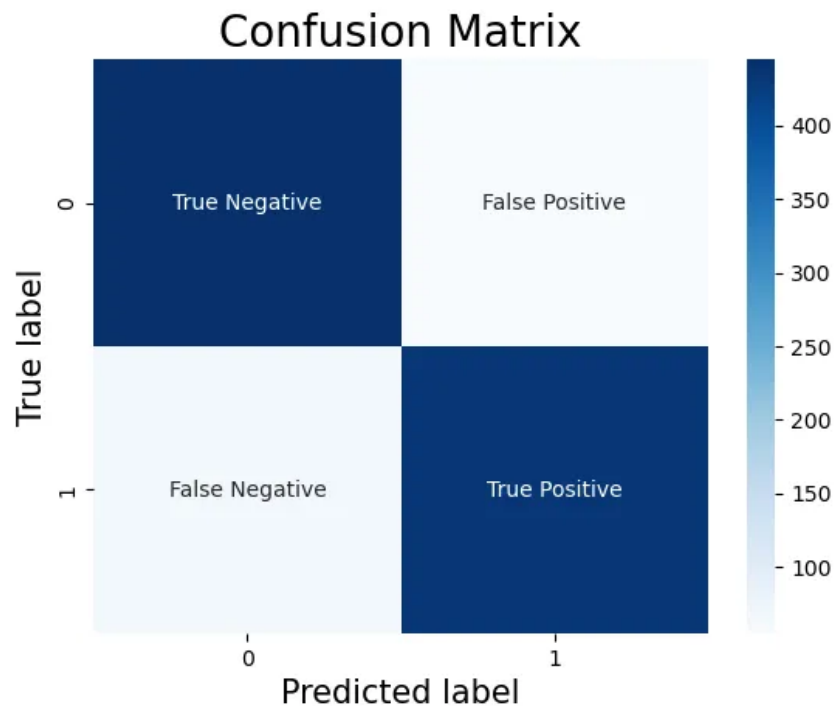


Рисунок 2.6 – Матриці невідповідностей у вигляді теплової карти

Наступною популярною метрикою якості є точність (accuracy). Вона є найбільш популярною завдяки своїй простоті. Очевидно, що ця метрика показує відсоток правильних прогнозів, зроблених моделлю серед усіх прогнозів. Висока точність свідчить про те, що модель робить велику кількість

прогнозів, які відповідають справжнім значенням прикладів. В той самий час, низька точність вказує на те, що модель робить велику кількість помилок.

Точність розраховується за формулою:

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

Також, разом з точністю, часто застосовують такі метрики як влучність (precision) та повнота (recall). Влучність показує, яку частку передбачених позитивних результатів складають істинно позитивні результати. Вона вимірює точність класифікатора у виявленні позитивних зразків. Висока точність означає, що модель має низький рівень хибнопозитивних результатів. Влучність можна розрахувати за наступною формулою:

$$\text{Precision} = \frac{TP}{TP + FP}.$$

В той самий час, повнота визначає частку позитивних результатів, які визначила модель. Вона вимірює здатність класифікатора знайти всі позитивні зразки. Висока повнота означає, що модель має низький рівень хибнонегативних результатів. Повноту можна розрахувати за наступною формулою:

$$\text{Recall} = \frac{TP}{TP + FN}.$$

Precision і Recall часто знаходяться в компромісних відносинах: підвищення одного може призвести до зниження іншого [16]. Наприклад, якщо модель прагне виявляти більше позитивних випадків (підвищуючи Recall), вона може класифікувати більше хибно позитивних випадків, що знижує Precision. І навпаки, якщо модель прагне підвищити Precision, вона може втратити деякі істинно позитивні випадки, знижуючи Recall.

Щоб збалансувати ці метрики, часто використовується F1-міра, яка є гармонійним середнім між Precision та Recall та розраховується за наступною формулою:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

F1-міра забезпечує єдину метрику, яка враховує як Precision, так і Recall, особливо в умовах незбалансованих класів.

Наступна метрика є необхідною для вимірювання продуктивності моделі бінарної класифікації. Однією з найбільш поширених метрик ефективності є крива AUC-ROC. AUC означає "площа під кривою", а ROC розшифровується як "робоча характеристика приймача" [17].

Для оцінки дискримінаційної здатності моделі, особливо коли класи незбалансовані, ROC-крива та AUC-метрика є незамінними. Вони забезпечують можливість порівняння продуктивності різних моделей при різних порогах класифікації, а також надають єдину оцінку, яка підсумовує загальну ефективність моделі. Крива ROC являє собою графічне представлення продуктивності бінарного класифікатора, яке показує компроміс між частотою істинно позитивних результатів (TPR) та частотою хибно позитивних результатів (FPR) при різних порогах класифікації. Крива AUC-ROC відображає залежність TPR від FPR при різних порогових значеннях. AUC представляє собою площу під цією кривою, значення якої варіюється від 0 до 1; чим вищий показник AUC, тим кращу продуктивність демонструє модель. Приклад ROC кривої зображено на рис. 2.7.

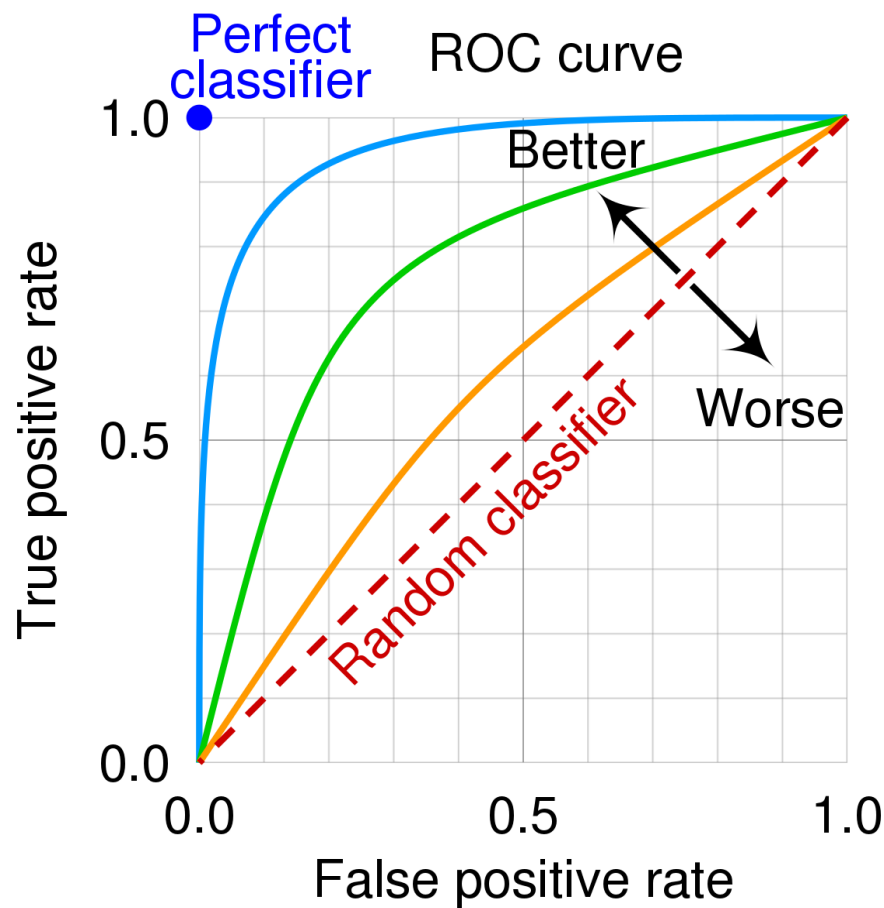


Рисунок 2.7 – ROC крива

2.4 Висновки до другого розділу

У цьому розділі було досліджено різні архітектури мовних моделей, такі як BERT і GPT, а також традиційні моделі класифікації, включаючи логістичну регресію, випадковий ліс і метод опорних векторів.

По-перше, найкращим підходом для класифікації HTTP-запитів може стати комбінація мовної моделі RoBERTa та традиційних математичних моделей класифікації. RoBERTa відома своєю здатністю до глибокого розуміння тексту та врахування семантичних зв'язків, що робить її ідеальним інструментом для векторизації текстових даних. Це дозволяє створити векторний простір, що містить інформацію про контекст і семантику запитів.

Використання традиційних математичних моделей класифікації, таких як логістична регресія, випадковий ліс та метод опорних векторів, для аналізу цього векторного простору дозволяє ефективно передбачати, чи є HTTP-запити шкідливими або нормальними. Це поєднання потужності мовних моделей з ефективністю традиційних методів класифікації може забезпечити високу точність та надійність класифікації в мережевій безпеці.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ

3.1 Опис наборів даних, використаних для навчання

Для більш об'єктивного експерименту, було прийнято рішення обрати декілька навчальних наборів даних.

Кожен набір даних, якій ми використали в ході цієї роботи було розділено на дві частини, а саме на навчальну частину (звичайні запити, що використовуються тільки для навчання представлень) та на частину для висновків (нормальний та аномальний трафік, закодований за допомогою мовної моделі). Коротко опишемо кожен з навчальних наборів.

CSIC 2010 був створений Інститутом кібернетики (Institute of Cybernetics, CSIC) для моделювання реальних атак на веб-додатки. Основною метою було забезпечення стандартного набору даних для тестування систем виявлення вторгнень (IDS) і систем захисту веб-додатків (WAF). Цей набір даних включає в себе нормальні та шкідливі HTTP запити. Кожен запис в наборі представляє собою один HTTP запит. Приклад такого запису можна побачити на рис 3.1.

```
GET http://localhost:8080/asf-logo-wide.gif.java HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=C2D7FE55801B2D1B7751B4122D7A86A7
Connection: close
```

Рисунок 3.1 – Приклад HTTP запиту з набору CSIC 2010

Нормальні запити моделюють звичайні дії користувача на веб-сайті, такі як перегляд сторінок, авторизація, пошук товарів, тощо. Шкідливі запити містить різноманітні типи атак, такі як SQL – ін'єкції, XSS, атаки на файлову

систему, тощо. Також, ми вирішили кодувати символи CR (повернення каретки) та LF (переведення рядка) у вигляді буквених рядків "\r" та "\n", оскільки це допоможе моделі відстежувати деякі типи атак.

Інший набір даних, який було використано для навчання був створений Організацією безпеки комунікацій і Канадським інститутом кібербезпеки. Він має назву CSE-CIC-IDS2018. Цей набір вважається дуже якісним і містить кілька веб-орієнтованих атак, таких як "Brute Force - Web", "Brute Force - XSS" та "SQL Injection". На жаль, цей набір даних не був спеціально створеним для задачі класифікації HTTP запитів, тому ми витягли необхідні дані з перехоплених пакетів, які містить цей набір.

Варто зауважити, що в цей набір даних має одну суттєву особливість. Для генерації аномального трафіку автори використали додаток DVWA, розміщений на одній машині. Через це, виникла необхідність внести деякі зміни в оригінальний набір даних, за для уникнення помилкових кореляцій.

1. Оскільки додаток був розміщений на одній машині, поле «Host» у HTTP запитах завжди вказувало на одну й ту саму IP адресу. Ми вирішили подолати це наступним шляхом – випадковим чином змінили значення поля “Host” на будь-яке значення з нормального набору даних.

2. Усі значення поля «URL» в запитах починалися з одного з наступних рядків: "/DVWA/vulnerabilities/xss", "DVWA/dvwa", "/DVWA". Було прийнято рішення видалити ці частини URL.

3. У кожному аномальному запиті був заголовок "Upgrade-Insecure-Requests", який ми також видалили.

Також, нормальний трафік не завжди містив достатнє інформаційне навантаження, тому ми видалили ті запити, які не відповідали правильному "Content-Type" (наприклад, "application/json").

Таким чином, було підготовлено два набори даних для обробки мовною моделлю. Наведемо поділ наборів даних у таблиці 3.1.

Таблиця 3.1 – Розподіл наборів даних

Розподіл	Набір даних	
	CSIC 2010	CSE-CIC-IDS2018
Нормальний трафік (навчальна частина)	36000	591175
Нормальний трафік (частина для висновків)	36000	13591
Аномальний трафік (частина для висновків)	25065	13591

Навчальна частина використовується тільки для моделі RoBERTa моделі RoBERTa. Частини для висновків використовуються на етапі класифікації.

3.2 Навчання моделі RoBERTa

Модель RoBERTa, яку ми використовуємо в нашому методі, базується на реалізації, доступній у пакеті HuggingFace Transformers. Ця реалізація надає інструменти для простого та ефективного використання передових мовних моделей, які були створені для обробки природної мови з використанням трансформерів. Ми вибрали модель RoBERTa завдяки її здатності до налаштування та високій точності при обробці текстових даних, що робить її ідеальним вибором для аналізу HTTP-запитів.

Для навчання нашої моделі ми використовували лише навчальні дані, які представляють нормальний HTTP трафік. Ці запити було закодовано за допомогою алгоритму Byte-Level Byte Pair Encoding. Це було зроблено з метою забезпечення високої точності моделі при розпізнаванні нормальних патернів у HTTP трафіку. Використовуючи ці дані, ми прагнули навчити

модель розуміти типові запити та відповіді веб-серверів, що, в свою чергу, дозволяє більш ефективно виявляти відхилення та потенційні атаки.

Ми вибрали розмір моделі "base", яка є збалансованим варіантом з точки зору продуктивності та ресурсів, необхідних для її навчання. Ця модель має наступні характеристики:

- максимальна довжина послідовності – 512 токенів, що дозволяє обробляти довгі запити;
- кількість шарів – 6, що забезпечує достатню глибину для моделювання складних залежностей у тексті;
- кількість голів уваги – 12, що дозволяє моделі фокусуватися на різних аспектах вхідних даних.

Модель була навчена протягом 10 епох, що є достатнім для досягнення стабільної точності уникаючи перенавчання. Розмір батча становить 32 зразки, що дозволяє ефективно використовувати обчислювальні ресурси та забезпечує швидке навчання.

Для додаткової стійкості та генералізації моделі, ми використовували техніку маскування токенів, замаскувавши 15% вхідних токенів. Це допомагає моделі краще навчитися розуміти контекст і відновлювати приховані слова, що є важливим для точного аналізу текстових даних.

Після навчання, модель використовується для генерації векторних представлень HTTP запитів. Ми отримуємо вектори довжиною 3072, які є конкатенацією останніх 4 шарів, кожен з яких має довжину 768. Цей підхід дозволяє зберігати більше інформації про контекст, оскільки кожен шар моделі захоплює різні рівні абстракції тексту. Остаточний вектор для кожного HTTP запиту є середнім значенням всіх рядків повідомлення з однаковою довжиною вектора. Такий підхід забезпечує компактне та інформативне представлення тексту, що дозволяє ефективно використовувати ці вектори для подальшої класифікації.

3.3 Створення класифікаторів

У цьому розділі ми розглянемо створення та налаштування кількох поширених класифікаторів, зокрема логістичної регресії, методу опорних векторів (SVM) та випадкового лісу. Ці моделі були обрані через їх різноманітність у підходах до класифікації, ефективність та популярність у різних прикладних задачах. Ми зосередимося на налаштуванні основних параметрів кожного з класифікаторів, що дозволить отримати високоякісні передбачення і забезпечить розуміння того, як ці параметри впливають на роботу моделей.

Логістична регресія була обрана за свою простоту і ефективність у задачах лінійної класифікації. Логістична регресія добре працює з обмеженими ресурсами та забезпечує інтерпретовані результати, що важливо для розуміння впливу кожної ознаки на передбачення. Також, при ініціалізації цієї моделі було використано наступні параметри:

- `max_iter=1000`: Цей параметр вказує максимальну кількість ітерацій для алгоритму оптимізації. Збільшення цього значення може бути необхідним, якщо алгоритм не збігається при меншій кількості ітерацій.

Випадковий ліс був обраний за його здатність обробляти великі обсяги даних та високий рівень стійкості до перенавчання. Цей алгоритм використовує ансамблевий підхід, що дозволяє покращити стабільність і точність моделі, об'єднуючи передбачення багатьох дерев рішень. Щодо параметрів, які було використано:

- `n_estimators=100`: Цей параметр визначає кількість дерев у випадковому лісі. Більша кількість дерев зазвичай покращує якість моделі, але також збільшує час навчання.

- `random_state=42`: Встановлення цього параметра забезпечує відтворюваність результатів. Це корисно для того, щоб експерименти могли бути повторені з тими ж результатами.

Нарешті, метод опорних векторів (SVM) було обрано завдяки її здатності працювати в високорозмірних просторах і забезпечувати високу точність, особливо у випадках, коли класи не лінійно роздільні. Ця модель ефективно знаходить оптимальну гіперплощину для розділення класів, що робить її потужним інструментом у багатьох задачах класифікації. Ми ініціалізували модель з наступними параметрами:

– `probability=True`: Цей параметр дозволяє оцінювати ймовірності належності до класів замість просто класових міток. Він додає можливість отримання ймовірнісних передбачень, що може бути корисним для подальшого аналізу.

3.4 Дослідження згенерованого векторного простору

Почнемо з результату алгоритму t-SNE. Графік на рисунку 3.2 показують, що класи добре розділені навіть у просторі меншої розмірності.

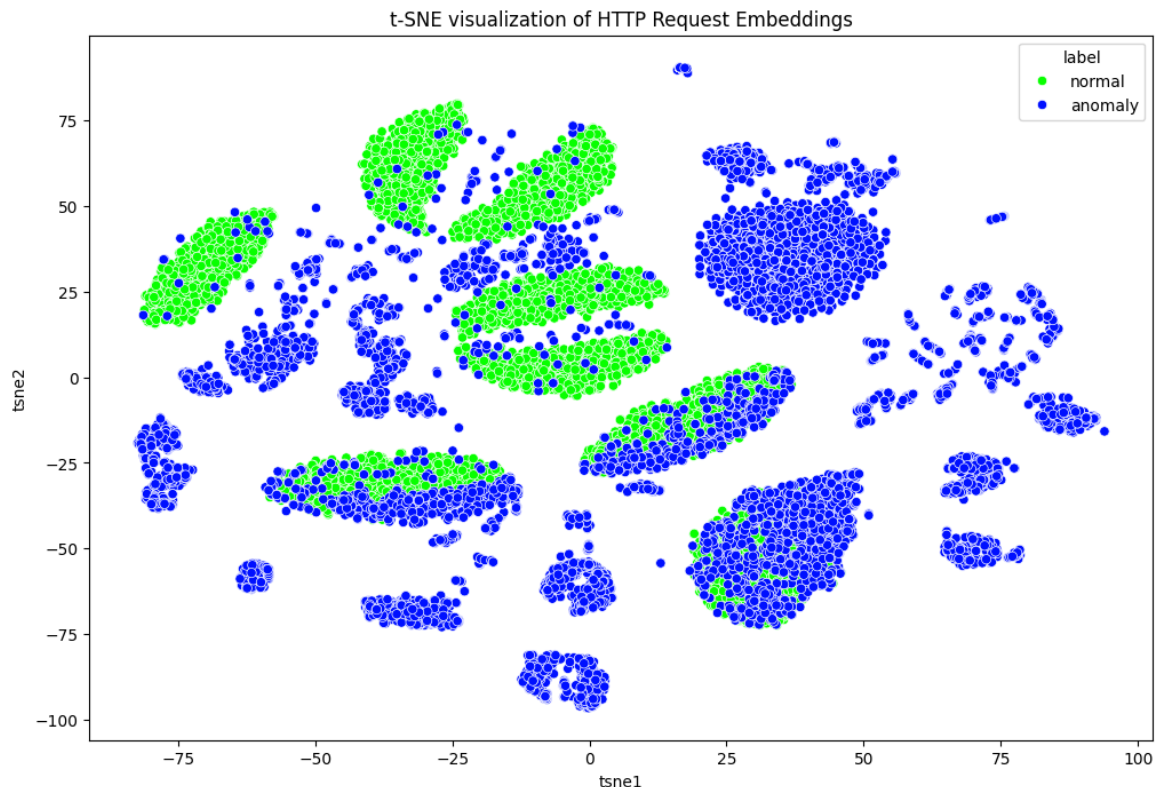


Рисунок 3.2. – t-SNE візуалізація векторного простору HTTP запитів

Це свідчить про те, що схожі HTTP-запити фактично групуються разом, що дозволяє досліджувати околиці будь-якого зразка. У таблиці 3.2 ми показуємо декілька n-тих найближчих зразків до заданого (в даному випадку було обрано запит під номером 8001).

Таблиця 3.2 – n-ті наближення до заданого зразка

1	id=2&nombre=Jam%F3n+Ib%E9rico&precio=100&cantidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al+carrito
2	id=1%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&nombre=Jam%F3n+Ib%E9rico&precio=100&cantidad=97&B1=A%F1adir+al+carrito
3	id=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=21&B1=A%F1adir+al+carrito
4	id=2&nombre=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&precio=100&cantidad=97&B1=A%F1adir+al+carrito
5	id=2&nombre=Jam%F3n+Ib%E9rico&precio=%27%3Bwaitfor+delay+%270%3A0%3A15%27%3B--&cantidad=17&B1=A%F1adir+al+carrito
6	id=1&nombre=Jam%F3n+Ib%E9rico&precio=100&cantidad=%27%29%3Bwaitfor+delay+%270%3A0%3A15%27%3B--&B1=A%F1adir+al+carrito
7	id=3%27%3Bwaitfor+delay+%270%3A0%3A15%27%3B--&nombre=Jam%F3n+Ib%E9rico&precio=39&cantidad=17&B1=A%F1adir+al+carrito
8	id=1&nombre=Jam%F3n+Ib%E9rico&precio=%3C%21--%23EXEC+cmd%3D%22ls+%2F%22--%3E&cantidad=71&B1=A%F1adir+al+carrito
9	id=%3C%21--%23EXEC+cmd%3D%22ls+%2F%22--%3E&nombre=Jam%F3n+Ib%E9rico&precio=100&cantidad=48&B1=A%F1adir+al+carrito
10	id=2%27%29%3Bwaitfor+delay+%270%3A0%3A15%27%3B--&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=73&B1=A%F1adir+al+carrito

Як видно, атаки SQL ін'єкцій насправді близькі одна до одної. Цікаво, що в подальшому оточенні зразка містяться інші типи ін'єкцій – команди операційної системи. Ми вважаємо, що цей підхід може бути використаний експертом для ідентифікації подібних спроб атак, а отже, допоможе в аналізі після інциденту.

3.5 Результати проведених експериментів, оцінка якості класифікаторів моделей за допомогою метрик якості

Після того, як HTTP запити було закодовано, ми передали вихідні дані енкодера на вхід до класифікаторів. Цей процес зображено на рисунку. 3.3.

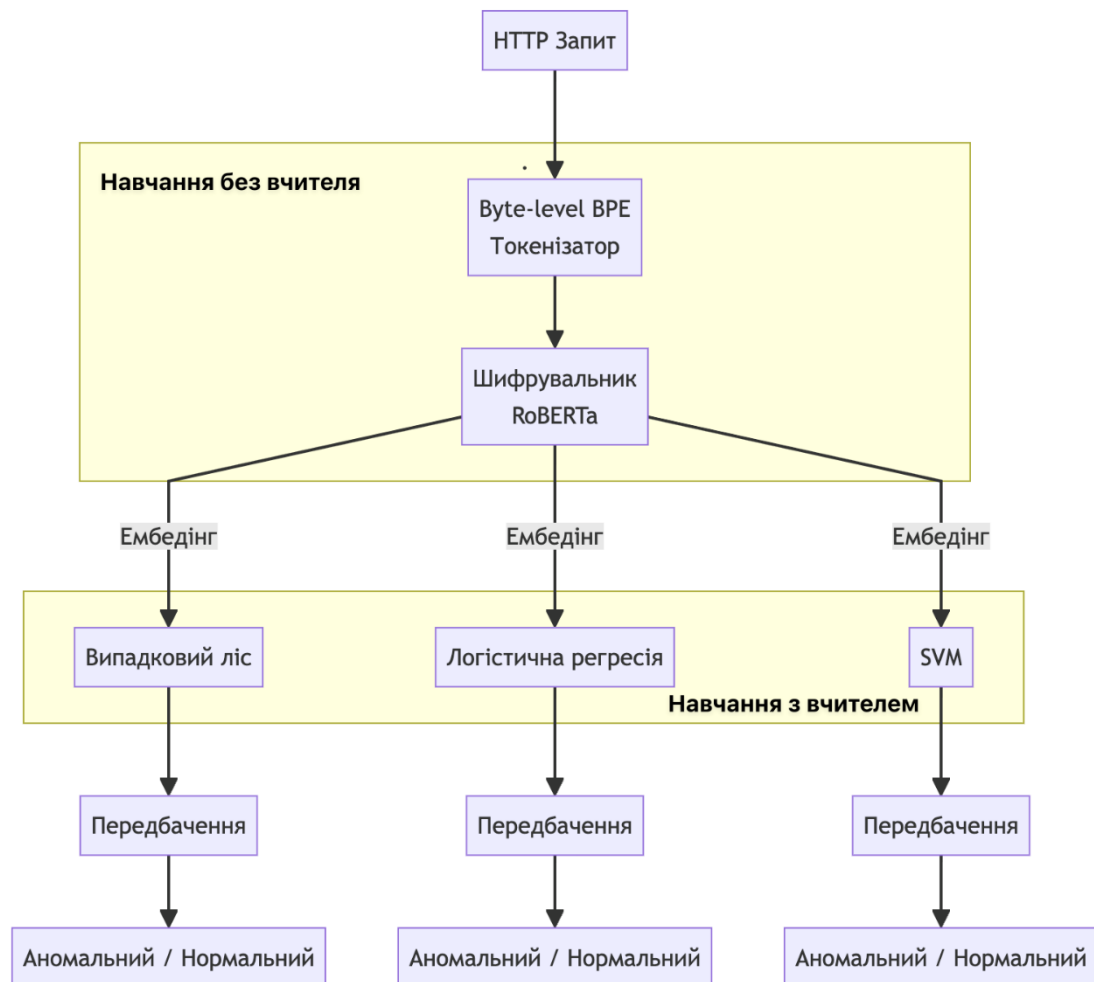


Рисунок 3.3 – Схема роботи алгоритму

Таким чином, кожен з класифікаторів отримав на вхід частину векторного простору, згенерованого RoBERTa. Цей процес значно підвищив ефективність передбачень, оскільки використання кодувальника на основі мовної моделі дало змогу відстежувати смислові зв'язки у запитах. Далі ми проаналізуємо ефективність моделей за допомогою метрик якостей та подивимось на те, яка з моделей показала себе краще у вирішенні даної задачі.

Спираючись на пораховані метрики якості, які зображені в таблиці 3.3, можна зробити висновок, що випадковий ліс показав себе найкраще в рамках виконання поставленої задачі.

Таблиця 3.3 – Порівняння класифікаторів за допомогою метрик якості

Модель	Точність (Accuracy)	Влучність (Precision)	Повнота (Recall)	F1-міра
Логістична регресія	0.9188	0.9638	0.8880	0.9244
SVM	0.7547	0.8715	0.6580	0.7499
Випадковий ліс	0.9664	0.9836	0.9557	0.9695

Це було доволі очевидним, оскільки він є найбільш гнучким з запропонованих моделей.

Також, окремо подивимось на криву ROC. Нам відомо, що чим ближче ROC крива до верхнього лівого кута, тим краще працює алгоритм. Площа під кривою (AUC) для ідеального алгоритму дорівнюватиме 1. На рисунку 3.4 ми можемо побачити, що випадковий ліс показав дуже гарні результати і в цьому випадку.

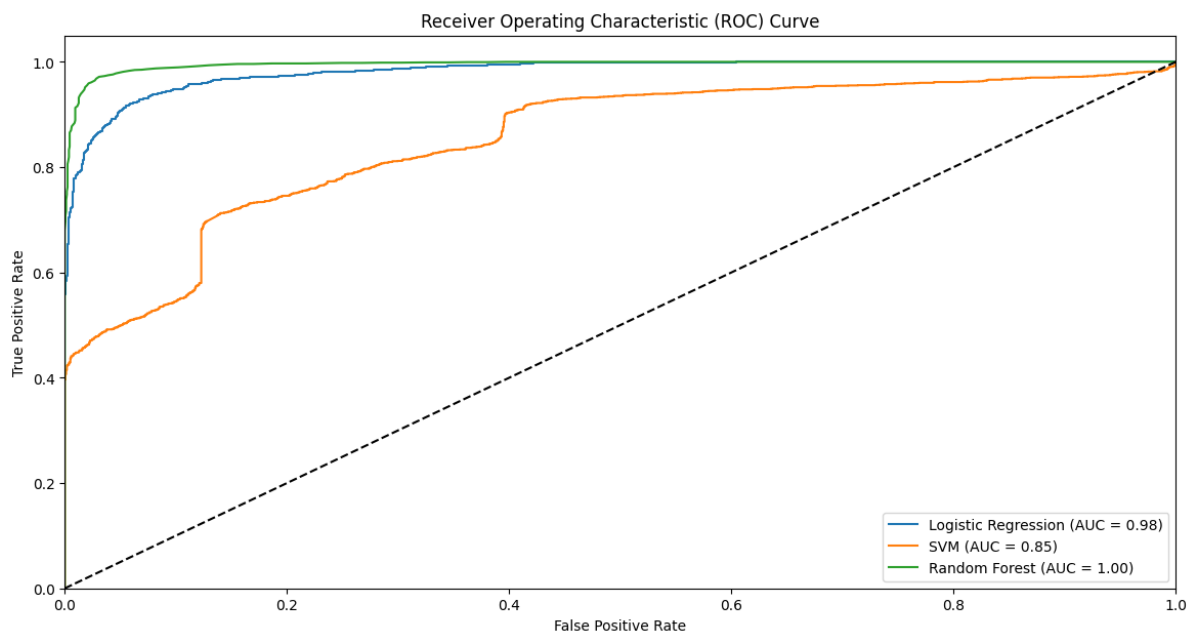


Рисунок 3.4 – ROC крива

Також, для більш якісного аналізу моделей, ми побудували матриці невідповідностей для кожної з них. Вони зображені на рисунку 3.5, 3.6 та 3.7.

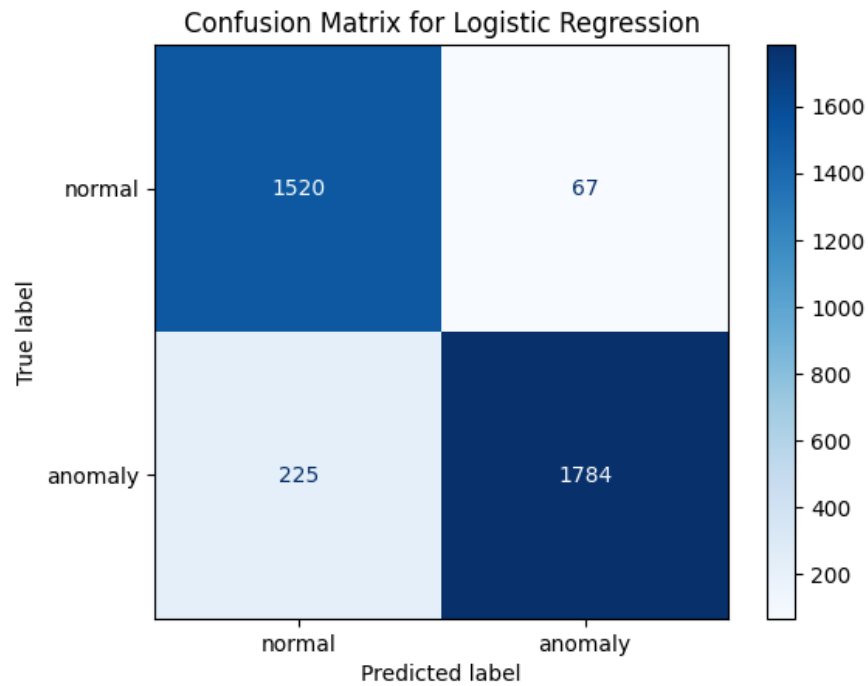


Рисунок 3.5 – Матриця невідповідностей для логістичної регресії

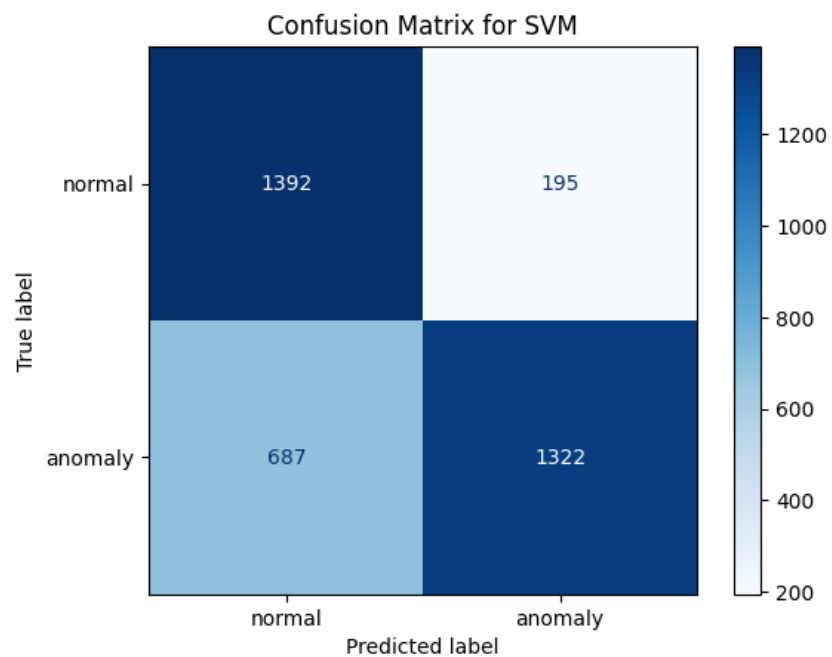


Рисунок 3.6 – Матриця невідповідностей для методу опорних векторів

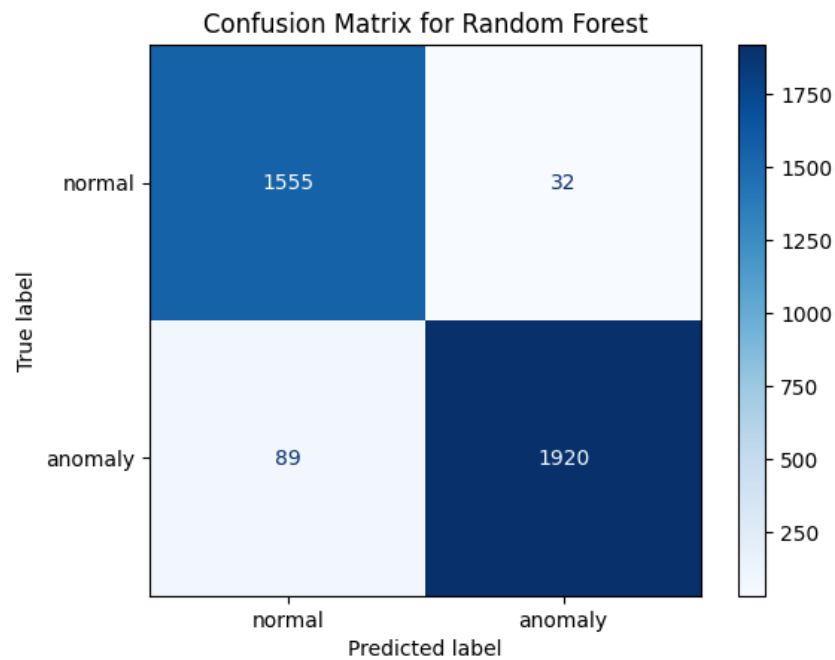


Рисунок 3.7 – Матриця невідповідностей для випадкового лісу

Аналізуючи ці метрики ми також бачимо, що випадковий ліс показує кращі результати, порівняно з логістичною регресією та методом опорних векторів, що робить цю ансамблевую модель найбільш підходящою для вирішення нашої задачі.

3.6. Висновки до третього розділу

У цьому розділі було виконано обробку даних для навчання моделей. Також, створено та застосовано мовну модель RoBERTa, яка згенерувала векторний простір ознак.

Також було побудовано три класифікатори: логістичну регресію, випадковий ліс та опорні вектори, було проаналізовано метрики якості та обрано найкращу модель для виконання задачі класифікації запитів – випадковий ліс.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

Цей розділ присвячений аналізу функціонально-вартісних аспектів програмного продукту, розробленого в рамках цього дослідження. Він дасть змогу оцінити обґрунтованість рішень щодо використаних технологій, розглядаючи їх у контексті теорії вибору опцій та оптимізації виробничих процесів. Ці елементи є ключовими для подальшого застосування результатів проекту в реальних умовах.

Всі етапи розробки піддаються ризикам, особливо початкові, які визначають межі того, як дослідник або команда можуть просуватися до досягнення мети проекту. Неправильний збір та обробка даних, вибір середовища розробки, інструментів розробки, засобів моніторингу виконання етапів та належна презентація вже виконаної роботи можуть підірвати доцільність усього програмного продукту, його надійність і, що так само важливо, економічну доцільність.

Проект, спрямований на розробку застосунку для аналізу шкідливого HTTP трафіку, має науково-дослідницький характер та не передбачає комерційної реалізації. З цієї причини важливо враховувати висновки, зроблені на основі аналізу результатів, які можуть бути в подальшому використані експертами з системними адміністраторами, DevOps/SecOps інженерами та іншими спеціалістами, що працюють у сфері інформаційної безпеки.

Все ж, економічний аспект лишається важливим у плануванні, оскільки більшість проектів обмежена в часі, а одним із найпотужніших інструментів впливу на цей фактор виступають фінанси. Важливо зазначити, що науково-дослідницькі проекти, спрямовані на пошук новаторських підходів, часто отримують належне фінансування лише за наявності видимих результатів, що ще більше підкреслює значення вартісного аналізу проведеної роботи. Таким чином, стане зрозуміло, наскільки обґрунтовано проводити розробку подібних

методів на основі вибраних технологій та отриманих результатів, які можуть бути проаналізовані при функціональному аналізі.

Зазначимо, що традиційний аналіз ринку та конкурентів у цьому випадку неможливий. Це пояснюється тим, що розробники передових рішень у галузі аналізу трафіку та інформаційної безпеки не розкривають фінансових аспектів, а їх програмні продукти не поширюються у вільному доступі з різних причин. Тому рішення не призначені для продажу і не мають на меті пряме впровадження у системи охорони здоров'я. Вони, швидше, спрямовані на поширення знань та розуміння процесів у мозку та впливу зовнішніх і внутрішніх факторів на його стан.

4.1 Постановка задачі проєктування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проєктування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який дозволяє візуалізувати та аналізувати одночасно декілька показників перфузії. Беручи за основу дану функцію, було виділено наступні головні функції виробу:

- F_1 – вибір мовної моделі;
- F_2 – вибір класифікаторів;
- F_3 – вибір мови програмування та середовища розробки.

Кожна з наведених функцій має свій варіант (або декілька) реалізації. Наведемо деякі з них.

Функція F_1 :

1. RoBERTa.
2. GPT.
3. BERT.

Функція F_2 :

1. Random Forest.
2. Логістична Регресія.
3. Метод опорних векторів .

Функція F_3 :

1. Python, PyCharm.
2. Python, Jupyter Notebook.
3. R, Visual Studio Code.
4. R, Google Colaboratory.

Також наведемо морфологічну карту на рис. 4.1, де будуть зображені реалізації наведених функцій.

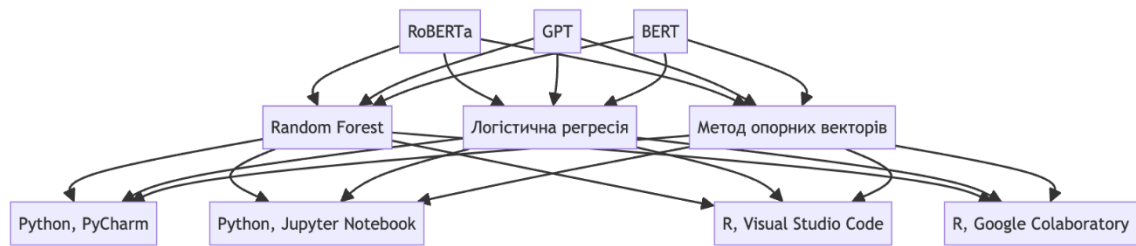


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця наведена у таблиці 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	1	Висока якість прогнозування, швидкість обробки даних	Потребує великого об'єму ресурсів
	2	Може використовуватися при задачах генерації тексту	Невисока здатність до розпізнавання тексту великого об'єму
	3	Витримує більше навантаження, обробляє більшу кількість «сміслових» даних	Слабка здатність до перебачення наступного речення
F_2	1	Має високу точність передбачення, нечутливий до викидів, добре працює з пропущеними даними	Не здатен на екстраполяцію, потребує великих об'ємів пам'яті
	2	Легко навчається, потребує невеликий об'єм обчислювальних ресурсів	Не можна застосовувати на лінійно-нероздільних даних
	3	Працює із лінійно-нерозділеними даними завдяки ядерному трюку, стійка до перенавчання	Складний у реалізації

Кінець таблиці 4.1

F_3	1	Зручна мова програмування із багатьма вбудованими бібліотеками та середовище розробки, зручне для неї	Python може потребувати певних навичок у роботі, а PyCharm використовує багато пам'яті,
	2	Зручний у використанні, особливо при написанні коду на Python	Більші потужності не є безкоштовними
	3	Зручна мова програмування для роботи зі статистичними даними, можна суміщати із іншими мовами, тим паче VS Code також підтримує багато мов	R має достатньо посередню швидкість, є складним, а із VS Code можуть виникати проблеми із дебагінгом
	4	Ті самі переваги мови R, при цьому Google Colaboratory може забезпечувати високу потужність	Google Colaboratory важко та довго працює із великими наборами даних

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1 :

Надаємо перевагу високій точності прогнозування та швидкості обробки даних, тому обираємо перший варіант.

Функція F_2 :

Так як кожен із методів має переваг, що можуть бути корисними у рамках розглядаємої задачі, то, не зважаючи на недоліки, будемо використовувати кожен із них.

Функція F_3 :

Реалізація програми на мові Python є найбільш сприйнятливою, має кращий розподіл переваг та недоліків, тому надаємо перевагу 2 варіанту.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

- $F_1 1 - F_2 1 - F_3 2$;
- $F_1 1 - F_2 2 - F_3 2$;

$$- F_1 1 - F_2 3 - F_3 2.$$

4.3. Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- $X1$ – швидкодія мови програмування;
- $X2$ – об'єм пам'яті для обчислень та збереження даних;
- $X3$ – час попередньої обробки даних;
- $X4$ – час навчання моделі.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	8000	15000	20000
Точність моделі	X2	%	65	80	95
Час попередньої обробки даних	X3	мс	100	85	70
Час навчання моделі	X4	г	20	15	8

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

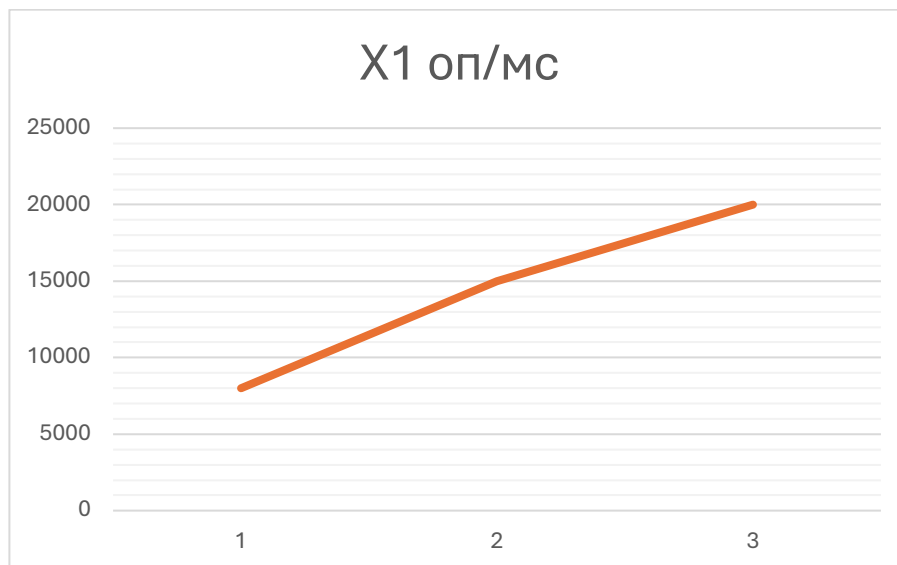


Рисунок 4.2 – X1, швидкодія мови програмування

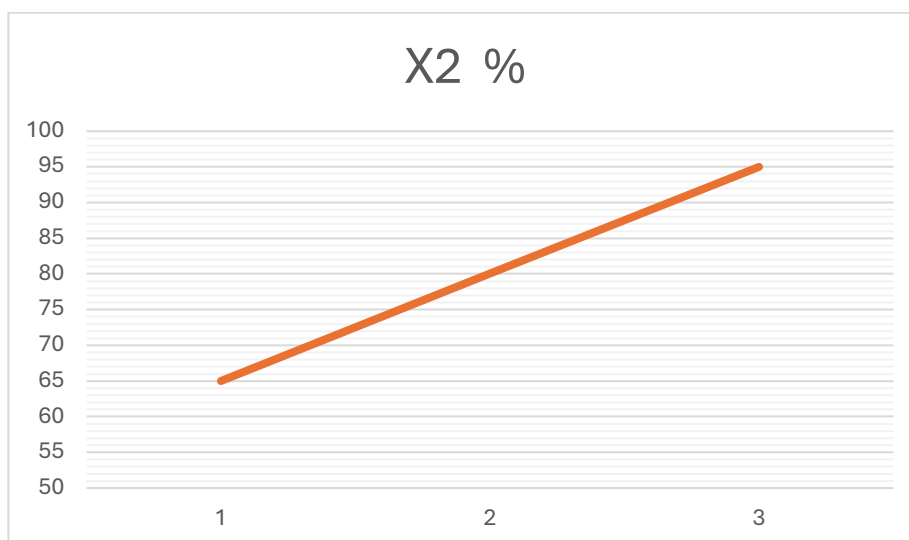


Рисунок 4.3 – X2, точність моделі

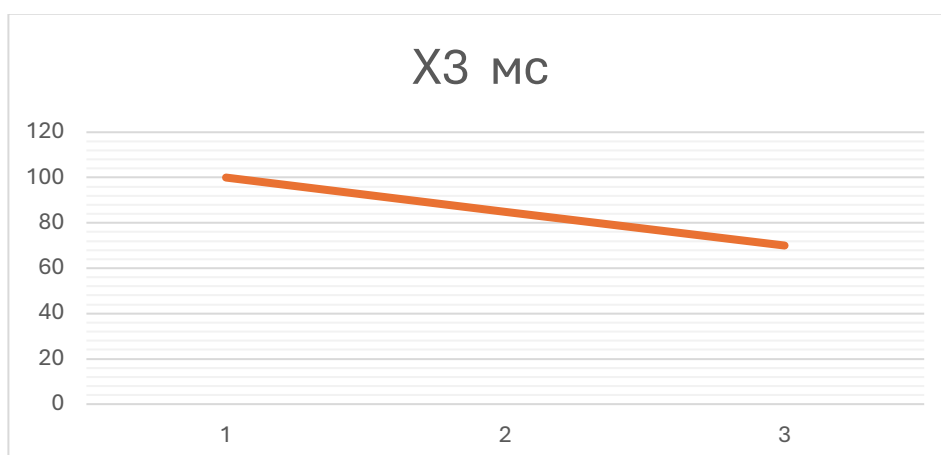


Рисунок 4.4 – X3, час попередньої обробки даних

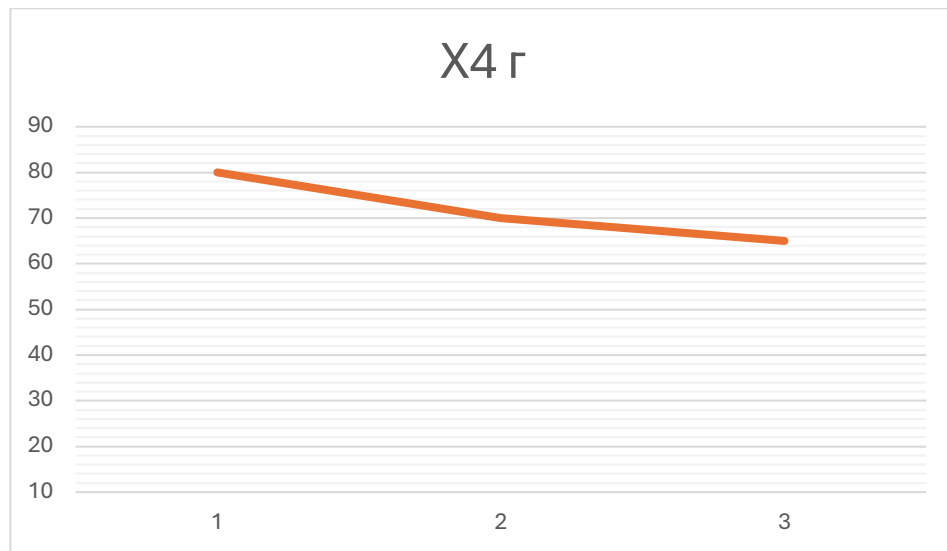


Рисунок 4.5 – X4, час навчання моделі

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Параметр	Назва параметра	Одиниц і виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхи- лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
$X1$	Швидкодія мови програмування	Оп/мс	4	3	4	3	4	4	2	26	6,5	42,25
$X2$	Точність моделі	%	2	1	1	2	1	1	1	11	-8,5	72,25
$X3$	Час попередньої обробки даних	мс	3	4	2	4	3	3	4	23	5,5	30,25
$X4$	Час навчання моделі	г	1	2	3	1	2	2	3	10	-3,5	12,25
	Разом		10	10	10	10	10	10	10	70	0	157

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

де N – число експертів та n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5. \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всіх параметрах повинна дорівнювати 0.

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^N \Delta_i^2 = 157. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 157}{7^2(4^3 - 4)} = 0,820 > W_k = 0,64. \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,64.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	>	>	>	>	>	1,5
X1 і X3	>	<	>	<	>	>	>	>	1,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	>	<	>	>	<	<	>	>	1,5
X3 і X4	>	>	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги i -го параметра над j -тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$.

Для кожного параметра зробимо розрахунок вагомості K_{bi} за наступними формулами:

$$K_{bi} = \frac{b_i}{\sum_{i=1}^n b_i} \quad (4.7)$$

$$b_i = \sum_{j=1}^N a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{bi} = \frac{b'_i}{\sum_{i=1}^n b'_i}, \quad (4.9)$$

$$b'_i = \sum_{j=1}^N a_{ij} b_j \quad (4.10)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер.	
	X1	X2	X3	X4	b_i	K_{bi}	b_i^1	K_{bi}^1	b_i^2	K_{bi}^2
X1	1	1,5	1,5	1,5	5,5	0,344	21,25	0,360	77,875	0,361
X2	0,5	1	0,5	1,5	3,5	0,219	12,25	0,208	44,875	0,208
X3	0,5	1,5	1	1,5	4,5	0,281	16,25	0,275	59,125	0,274
X4	0,5	0,5	0,5	1	2,5	0,156	9,25	0,157	34,125	0,158
Всього:					16	1	59	1	216	1

4.5. Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів $X2$ (точність моделі), $X3$ (час попередньої обробки даних) та $X4$ (час навчання моделі) відповідають технічним вимогам умов функціонування даного ПП. Абсолютне значення параметра $X1$ (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^n K_{ei,j} B_{i,j}, \quad (4.11)$$

де: n – кількість параметрів, K_{ei} – коефіцієнт вагомості i -го параметра, B_i – оцінка i -го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	1	X3	73	4	0,274	1,096
F2	1	X2	97	5	0,208	1,04
	1	X4	8	5	0,158	0,79
	2	X2	92	4	0,208	0,832
	2	X4	10	4	0,158	0,632
	3	X2	83	3	0,208	0,624
	3	X4	11	3	0,158	0,474
F3	2	X1	18000	5	0,361	1,805

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \dots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1,096 + 1,04 + 0,79 + 1,805 = 4,731;$$

$$K_{K2} = 1,096 + 0,832 + 0,632 + 1,805 = 4,365;$$

$$K_{K3} = 1,096 + 0,624 + 0,474 + 1,805 = 3,999.$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту.
2. Розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_0 = T_P \cdot K_P \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де:

- T_P – трудомісткість розробки ПП;
- K_P – поправочний коефіцієнт;
- $K_{СК}$ – коефіцієнт на складність вхідної інформації;
- K_M – коефіцієнт рівня мови програмування;
- $K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;
- $K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_P = 90$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_P = 1,8$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ} = 0,7$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1,8 \cdot 0,7 = 113,4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання використовується алгоритм третьої групи складності, степінь новизни Б, тобто $T_P = 35$ людино-днів, $K_P = 0,9$, $K_{СК} = 1$, $K_{СТ} = 0,7$:

$$T_2 = 35 \cdot 0,9 \cdot 0,7 = 22,05 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (113,4 + 22,05 + 3,7 + 22,05) \cdot 8 = 1297,3 \text{ людино-годин.}$$

$$T_{II} = (113,4 + 22,05 + 5,2 + 22,05) \cdot 8 = 1313,6 \text{ людино-годин.}$$

$$T_{III} = (113,4 + 22,05 + 6,6 + 22,05) \cdot 8 = 1315 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант III.

В розробці беруть участь два аналітики з окладом 28000 грн. та один програміст із окладом 25000. Визначимо середню зарплату за годину за формулою:

$$C_q = \frac{M}{T_m \cdot t} \text{ грн.,} \quad (4.14)$$

де:

- M – місячний оклад працівників;
- T_m – кількість робочих днів тиждень;
- t – кількість робочих годин в день.

$$C_q = \frac{28000 + 28000 + 25000}{3 \cdot 21 \cdot 8} = 160,71 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{зп} = C_q \cdot T_i \cdot K_d, \quad (4.16)$$

де:

- C_q – величина погодинної оплати праці програміста;
- T_i – трудомісткість відповідного завдання;

– K_d – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I. } C_{3П} = 160,71 \cdot 1297,3 \cdot 1,2 = 250186,90 \text{ грн.}$$

$$\text{II. } C_{3П} = 160,71 \cdot 1313,6 \cdot 1,2 = 253330,39 \text{ грн.}$$

$$\text{III. } C_{3П} = 160,71 \cdot 1315 \cdot 1,2 = 253600,38 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I. } C_{ВІД} = C_{3П} \cdot 0.22 = 250186,90 \cdot 0.22 = 55041,12 \text{ грн.}$$

$$\text{II. } C_{ВІД} = C_{3П} \cdot 0.22 = 253330,39 \cdot 0.22 = 55732,69 \text{ грн.}$$

$$\text{III. } C_{ВІД} = C_{3П} \cdot 0.22 = 253600,38 \cdot 0.22 = 55792,08 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. (C_M)

Так як одна ЕОМ обслуговує одного аналітика з окладом 28000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_{Г} = 12 \cdot M \cdot K_3 = 12 \cdot 28000 \cdot 0,2 = 67200 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{3П} = C_{Г} \cdot (1 + K_3) = 67200 \cdot (1 + 0.2) = 80640 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{ВІД} = C_{3П} \cdot 0.22 = 80640 \cdot 0,22 = 17740,8 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 30% та вартості ЕОМ – 30000 грн.

$$C_A = K_{TM} \cdot K_A \cdot C_{ПР} = 1,3 \cdot 0,3 \cdot 30000 = 11700 \text{ грн.},$$

де:

- K_{TM} – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;
- K_A – річна норма амортизації;
- $C_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{TM} \cdot C_{ПР} \cdot K_P = 1,3 \cdot 30000 \cdot 0,04 = 1560 \text{ грн.},$$

де K_P – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{EF} = (D_K - D_B - D_C - D_P) \cdot t_3 \cdot K_B = (365 - 104 - 12 - 16) \cdot 8 \cdot 0,5 = 932 \text{ години},$$

де:

- D_K – календарна кількість днів у році;
- D_B, D_C – відповідно кількість вихідних та святкових днів;
- D_P – кількість днів планових ремонтів устаткування;
- t – кількість робочих годин в день;
- K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{EL} = T_{EF} \cdot N_C \cdot K_3 \cdot C_{EH} = 932 \cdot 0,2 \cdot 0,5 \cdot 4,86 = 452,95 \text{ грн.},$$

де:

- N_c – середньо-споживча потужність приладу;
- K_3 – коефіцієнтом зайнятості приладу;
- $\Pi_{ЕН}$ – тариф за 1 кВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_H = \Pi_{ПР} \cdot 0,67 = 30000 \cdot 0,67 = 20100 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_A + C_P + C_{ЕЛ} + C_H, \quad (4.17)$$

$$C_{ЕКС} = 80640 + 17740,8 + 11700 + 1560 + 452,95 + 20100 = 132193,75 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{М-Г} = C_{ЕКС} / T_{ЕФ} = 111112,15 / 932 = 141,84 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_M = C_{М-Г} \cdot T, \quad (4.18)$$

$$\text{I. } C_M = 141,84 \cdot 1297,3 \cdot 1,2 = 220800,46 \text{ грн.}$$

$$\text{II. } C_M = 141,84 \cdot 1313,6 \cdot 1,2 = 223574,72 \text{ грн.}$$

$$\text{III. } C_M = 141,84 \cdot 1315 \cdot 1,2 = 223831,00 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{ЗП} \cdot 0,67, \quad (4.19)$$

$$\text{I. } C_H = 250186,90 \cdot 0,67 = 167625,22 \text{ грн.}$$

$$\text{II. } C_H = 253330,39 \cdot 0,67 = 169731,36 \text{ грн.}$$

$$\text{III. } C_H = 253600,38 \cdot 0,67 = 169912,25 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{\text{ПП}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{М}} + C_{\text{Н}}, \quad (4.20)$$

$$\text{I. } C_{\text{ПП}} = 250186,90 + 55041,12 + 220800,46 + 167625,22 = 693653,70 \text{ грн.}$$

$$\text{II. } C_{\text{ПП}} = 253330,39 + 55732,69 + 223574,72 + 169731,36 = 702369,15 \text{ грн.}$$

$$\text{III. } C_{\text{ПП}} = 253600,38 + 55792,08 + 223813,00 + 169912,25 = 703117,72 \text{ грн.}$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = K_{\text{К}j} / C_{\text{Ф}j}, \quad (4.21)$$

$$K_{\text{ТЕР}1} = 4,731 / 693653,70 = 0,682 \cdot 10^{-5};$$

$$K_{\text{ТЕР}2} = 4,365 / 702369,15 = 0,621 \cdot 10^{-5};$$

$$K_{\text{ТЕР}3} = 3,999 / 703117,72 = 0,568 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{\text{ТЕР}1} = 0,682 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У

нього виявився найкращий показник техніко-економічного рівня якості $K_{\text{ТЕР}} = 0,682 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- вибір кількісного методу прогнозування: лінійна регресія;
- вибір якісного методу прогнозування: метод морфологічного аналізу;
- вибір мови програмування та середовища розробки: Python, Jupyter Notebook.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

4.8 Висновки до четвертого розділу

У цьому розділі було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

У результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують. На основі проведеного аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У даній дипломній роботі було розроблено рішення для класифікації HTTP запитів за допомогою мовних моделей та методів машинного навчання. Результатом виконаної роботи є навчена та налаштована модель RoBERTa, яка використовується для створення векторного простору з текстових даних, які містять в собі HTTP запити. Також, було створено та налаштовано випадковий ліс, логістичну регресію та методи опорних векторів.

У першому розділі було розглянуто види сучасних протоколів передачі даних, приведні різні типи хакерських атак, які уснують в світі та розглянуто актуальне програмне забезпечення для боротьби з ними. У другому розділі було досліджено різні архітектури мовних моделей, такі як BERT і GPT, а також традиційні моделі класифікації, включаючи логістичну регресію, випадковий ліс і метод опорних векторів. У третьому розділі було налаштовано мовну модель, проведено аналіз векторного простору, також створено класифікатори, оцінено їх ефективність, порівняно результати. Четвертий розділ присвячено функціонально-вартісному аналізу розробленого продукту.

Поставлену задачу було повністю розв'язано, мовну модель RoBERTa було налаштовано на зібраних HTTP запитах та впроваджено алгоритм генерації векторного простору. Векторний простір успішно використовується в якості вхідних даних для класифікаторів. Покращити наведений алгоритм можна за допомогою використання більш потужних обчислювальних машин, що забезпечить більш якісний підбір гіперпараметрів для класифікаторів та навчання протягом більшої кількості епох.

Недоліками даного алгоритму можна вважати велику кількість часу, необхідну для навчання. Ця проблема вирішується шляхом використання відеокарт з більшою кількістю ядер для паралельного обчислювання, що забезпечить одночасне навчання декількох шарів мовної моделі.

Розроблений алгоритм для прогнозування може бути використаний у науково-дослідницьких цілях та потенційно бути частиною більших проєктів та програмних продуктів, а саме комплексних рішень для кібербезпеки, які використовують сучасні методи мовного моделювання для виявлення та блокування шкідливих HTTP запитів в реальному часі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Dykstra J. Essential Cybersecurity Science: Build, Test, and Evaluate Secure Systems. Los Angeles, CA: O'Reilly Media, 2015. 256 p.
2. Sherwin C. Snort ++. GitHub, 2024. URL: <https://github.com/snort3/snort3> (дата звернення: 20.05.2024).
3. Vizard M. Cloudflare CEO: 2014's Fight for the Internet and a Tighter Security Community. SiliconANGLE, 2014. URL: <https://siliconangle.com/2014/01/09/cloudflare-ceo-2014s-fight-for-the-internet-and-a-tighter-security-community/>
4. Gage P. Byte-Pair Encoding: Subword Based Tokenization Algorithm. Tbilisi: University of Computer Science, 2018. 145 p.
5. Vaswani A. Attention Is All You Need. California: NIPS, 2017. URL: <https://arxiv.org/pdf/1706.03762>
6. Devlin J., Chang M., Lee K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Hong Kong: University of Hong Kong, 2018. 120 p.
7. Liu Y. RoBERTa: A Robustly Optimized BERT Pretraining Approach. Seattle, WA: University of Washington, 2019. 201p.
8. Brown T. B. Language Models are Few-Shot Learners. Baltimore, MD: Johns Hopkins University, 2020. 75 p.
9. Raschka S., Mirjalili V. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. Birmingham: Packt Publishing, 2019. 770 p.
10. Каніовська І. Ю. Конспект лекцій з теорії ймовірностей та математичної статистики. Київ: КПІ ім. Ігоря Сікорського, 2020. URL: <https://campus.kpi.ua/student/index.php?mode=mob&show&irid=215578>

11. David H. A., Sutchliffe H. B. The Origins of Logistic Regression. *Tinbergen Institute Discussion Paper*. Amsterdam: Tinbergen Institute, 2002. Vol 1, No. 2 P. 20-25.
12. Louppe G. Understanding Random Forest from Theory to Practice. Liege: University of Liege, 2014. 223 p.
13. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge, MA: MIT Press, 2016. 775 p.
14. Schölkopf B. The Kernel Trick for Distances. *Microsoft Research*. 2000. Vol. 2, No. 1. P.30-41.
15. Cortes C., Vapnik V. Support Vector Machines. Machine Learning. Malta: University of Malta, 1995. 326p.
16. Powers D. M. W. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*. 2011. Vol. 10, No. 2. P. 29-39.
17. Fawcett T. Introduction to ROC Analysis. Pattern Recognition Letters. San Diego, CA: Institute for the Study of Learning and Expertise, 2006. 312p.
18. Недашківська Н. І. Методи і моделі інтелектуального аналізу даних. Київ: КПІ ім. Ігоря Сікорського, 2019. 71 с. URL: <https://ela.kpi.ua/handle/123456789/53764>
19. Микитишин А. Г., Митник М. М. Комп'ютерні мережі. Львів: Магнолія 2006, 2013. 256 с.
20. Сліпченко В. Г. Локальні комп'ютерні мережі. Проектування, використання та програмування. Київ: ІВЦ Політехніка, 2002. 184 с.
21. Строчніков К.С. Комп'ютерні мережі. Харків: Харківський політехнічний інститут, 2005. 304 с.
22. Тарнавський Ю. А. Організація комп'ютерних мереж. Київ: КПІ ім. Ігоря Сікорського, 2018. 259 с.
23. Фейт С. TCP/IP Архітектура, протоколи, реалізація. Київ: ІВЦ Політехніка, 2000. 424 с.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```

from google.colab import drive
drive.mount('/content/drive')

!pip install transformers
!pip install pandas
!pip install torch
!pip install datasets
!pip install -U accelerate
!pip install -U transformers

from transformers import RobertaModel, RobertaTokenizer
import torch, pandas as pd
from pathlib import Path
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

paths = [str(x) for x in
Path("/content/drive/MyDrive/HTTP2VEC/").glob("**/*.txt")]
paths

with open(paths[2], 'r') as file:
    normal_requests_roberta = file.read().split('\n\n\n')

with open(paths[1], 'r') as file:
    normal_requests_classifier = file.read().strip().split('\n\n\n')

with open(paths[0], 'r') as file:
    anomaly_requests_classifier = file.read().strip().split('\n\n\n')

normal_requests_roberta

def parse_request(request):
    try:
        lines = request.split('\n')
        if len(lines) < 1:
            return None

        request_line = lines[0]
        headers = {}
        body = None

        parts = request_line.split(' ')
        if len(parts) < 3:
            return None

        method, url, _ = parts

        for line in lines[1:]:
            if ': ' in line:
                key, value = line.split(': ', 1)
                headers[key] = value
            else:

```

```

        body = line

    return {
        'method': method,
        'url': url,
        'headers': headers,
        'body': body
    }
except Exception as e:
    print(f"Error parsing request: {e}")
    return None

def filter_none(parsed_requests):
    return [req for req in parsed_requests if req is not None]

with open(paths[2], 'r') as file:
    normal_requests_roberta = file.read().strip().split('\n\n')

with open(paths[1], 'r') as file:
    normal_requests_classifier = file.read().strip().split('\n\n')

with open(paths[0], 'r') as file:
    anomaly_requests_classifier = file.read().strip().split('\n\n')

parsed_normal_roberta = filter_none([parse_request(req) for req in
normal_requests_roberta])
parsed_normal_classifier = filter_none([parse_request(req) for req in
normal_requests_classifier])
parsed_anomaly_classifier = filter_none([parse_request(req) for req in
anomaly_requests_classifier])

import pandas as pd

df_normal_roberta = pd.DataFrame(parsed_normal_roberta)
df_normal_classifier = pd.DataFrame(parsed_normal_classifier)
df_anomaly_classifier = pd.DataFrame(parsed_anomaly_classifier)

# Define a safe unpack function for requests
def safe_unpack_requests(requests, min_length=3):
    if len(requests) >= min_length:
        return requests[:min_length]
    else:
        raise ValueError(f"Not enough requests to unpack (expected
{min_length}, got {len(requests)})")

try:
    req1, req2, req3 = safe_unpack_requests(parsed_normal_roberta)
    # Process the requests as needed
    print(req1, req2, req3)
except ValueError as e:
    print(e)

def create_feature_string(row):
    feature_str = f"{row['method']} {row['url']} "
    headers_str = ' '.join([f"{key}:{value}" for key, value in
row['headers'].items()])
    feature_str += headers_str
    if row['body']:
        feature_str += f" {row['body']}"

```

```

    return feature_str

df_normal_roberta['features'] =
df_normal_roberta.apply(create_feature_string, axis=1)

from transformers import RobertaTokenizer, RobertaModel
import torch

tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
model = RobertaModel.from_pretrained('roberta-base')

def get_roberta_embeddings(text):
    inputs = tokenizer(text, return_tensors='pt', truncation=True,
padding=True, max_length=512)
    outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).detach().numpy()

df_normal_roberta['embeddings'] =
df_normal_roberta['features'].apply(get_roberta_embeddings)

df_normal_classifier['features'] =
df_normal_classifier.apply(create_feature_string, axis=1)
df_anomaly_classifier['features'] =
df_anomaly_classifier.apply(create_feature_string, axis=1)

df_normal_classifier['embeddings'] =
df_normal_classifier['features'].apply(get_roberta_embeddings)
df_anomaly_classifier['embeddings'] =
df_anomaly_classifier['features'].apply(get_roberta_embeddings)

df_normal_classifier['label'] = 'normal'
df_anomaly_classifier['label'] = 'anomaly'

df_classifier = pd.concat([df_normal_classifier, df_anomaly_classifier])

import numpy as np
X = np.vstack(df_classifier['embeddings'].values)
y = df_classifier['label'].values

# Split the data
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_curve, auc, precision_recall_curve
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

results = {}

```

```

for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    y_prob = clf.predict_proba(X_test)[:, 1]

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label='anomaly')
    recall = recall_score(y_test, y_pred, pos_label='anomaly')
    f1 = f1_score(y_test, y_pred, pos_label='anomaly')

    fpr, tpr, _ = roc_curve(y_test == 'anomaly', y_prob)
    roc_auc = auc(fpr, tpr)

    precision_pr, recall_pr, _ = precision_recall_curve(y_test == 'anomaly',
y_prob)
    pr_auc = auc(recall_pr, precision_pr)

    cm = confusion_matrix(y_test, y_pred, labels=['normal', 'anomaly'])

    results[name] = {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'roc_curve': (fpr, tpr, roc_auc),
        'pr_curve': (precision_pr, recall_pr, pr_auc),
        'confusion_matrix': cm
    }

for name, metrics in results.items():
    print(f"{name} - Accuracy: {metrics['accuracy']:.4f}, Precision:
{metrics['precision']:.4f}, Recall: {metrics['recall']:.4f}, F1 Score:
{metrics['f1']:.4f}")

plt.figure(figsize=(14, 7))
for name, metrics in results.items():
    fpr, tpr, roc_auc = 1 - metrics['roc_curve']
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
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) Curve')
plt.legend(loc='lower right')
plt.show()

plt.figure(figsize=(14, 7))
for name, metrics in results.items():
    precision_pr, recall_pr, pr_auc = metrics['pr_curve']
    plt.plot(recall_pr, precision_pr, label=f'{name} (AUC = {pr_auc:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()

def plot_confusion_matrix(cm, title):

```

```

    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=['normal', 'anomaly'])
    disp.plot(cmap=plt.cm.Blues)
    plt.title(title)
    plt.show()

for name, metrics in results.items():
    cm = metrics['confusion_matrix']
    plot_confusion_matrix(cm, f'Confusion Matrix for {name}')

from sklearn.manifold import TSNE
import seaborn as sns
X = np.vstack(df_classifier['embeddings'].values)
y = df_classifier['label'].values

tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)

df_tsne = pd.DataFrame(X_tsne, columns=['tsne1', 'tsne2'])
df_tsne['label'] = y

plt.figure(figsize=(12, 8))
sns.scatterplot(x='tsne1', y='tsne2', hue='label',
palette=sns.color_palette("hsv", 2), data=df_tsne)
plt.title('t-SNE visualization of HTTP Request Embeddings')
plt.show()

plt.figure(figsize=(14, 7))
for name, metrics in results.items():
    fpr, tpr, roc_auc = metrics['roc_curve']
    plt.plot(1- fpr, 1- tpr, label=f'{name} (AUC = {1-roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
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) Curve')
plt.legend(loc='lower right')
plt.show()

plt.figure(figsize=(14, 7))
for name, metrics in results.items():
    precision_pr, recall_pr, pr_auc = metrics['pr_curve']
    plt.plot(recall_pr, precision_pr, label=f'{name} (AUC = {pr_auc:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()

df_anomaly_classifier.loc[0]

import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
import pandas as pd

df_classifier['embeddings'] = df_classifier['embeddings'].apply(lambda x:
np.array(x))

```

```

embedding_shapes = df_classifier['embeddings'].apply(lambda x:
x.shape).unique()
print(f"Unique embedding shapes: {embedding_shapes}")

assert len(embedding_shapes) == 1, "All embeddings must have the same shape"

df_classifier.reset_index(drop=True, inplace=True)

sql_injection_samples =
df_classifier[df_classifier['features'].str.contains('SELECT+', case=False,
na=False)]
sql_injection_samples.url.iloc[0]

sql_injection_sample_index = df_anomaly_classifier.index[8001]
sql_injection_embedding = df_classifier.loc[sql_injection_sample_index,
'embeddings']

df_classifier.loc[sql_injection_sample_index]

all_embeddings = np.vstack(df_classifier['embeddings'].values)
distances = euclidean_distances(sql_injection_embedding,
all_embeddings).reshape(-1,1)

df_classifier['distance_to_sql_injection'] = distances

closest_samples =
df_classifier.sort_values(by='distance_to_sql_injection').head(11)

for index, row in closest_samples.iterrows():
    print(f"Sample Index: {index}")
    print(f"Distance: {row['distance_to_sql_injection']:9.5f}")
    print(f"Label: {row['label']}")
    if row['label'] == 'normal':
        print(f"Request: {df_classifier.loc[index, 'features']}")
    else:
        print(f"Request: {df_classifier.loc[index, 'features']}")
    print("\\n")

```

ДОДАТОК Б ПРЕЗЕНТАЦІЯ

Слайд 1

Класифікація HTTP запитів за допомогою нейронних мереж

Виконав: студент групки Ка-04, Свирид
Нікіта

Науковий керівник: PhD, старший
викладач кафедри ММСА, Левенчук
Людмила Борисівна



Слайд 2

Актуальність

В умовах гібридної війни, що ведеться Росією проти України, кібератаки стали одним із ключових інструментів. Шкідливі HTTP запити можуть використовуватися для проведення атак на державні установи, інфраструктуру та приватні компанії. Вчасне виявлення таких запитів є критичним для захисту національної безпеки. Впровадження технологій класифікації HTTP запитів сприяє розвитку кібербезпеки в Україні, підвищуючи рівень захисту державних та приватних організацій від кіберзагроз.

Слайд 3

Об'єкт, предмет і мета дослідження

- **Об'єкт дослідження** – задача класифікації HTTP запитів за допомогою мовної моделі, спрямованих на веб-додаток.
- **Предмет дослідження** – мовна модель RoBERTa для вирішення задачі генерування ознак, її математична складова, переваги та недоліки.
- **Мета дослідження** – аналіз HTTP запитів, створення векторного простору для навчання класифікаторів, підбір моделей для класифікації.

3

Слайд 4

Постановка задачі

Дана робота присвячена розробці та реалізації моделі, що використовує методи мовного моделювання для вирішення задач класифікації шкідливих HTTP запитів. За результатами дослідження очікується отримати найбільш ефективну модель шляхом оцінки метрик якості.

4

Слайд 5

Попередня обробка даних

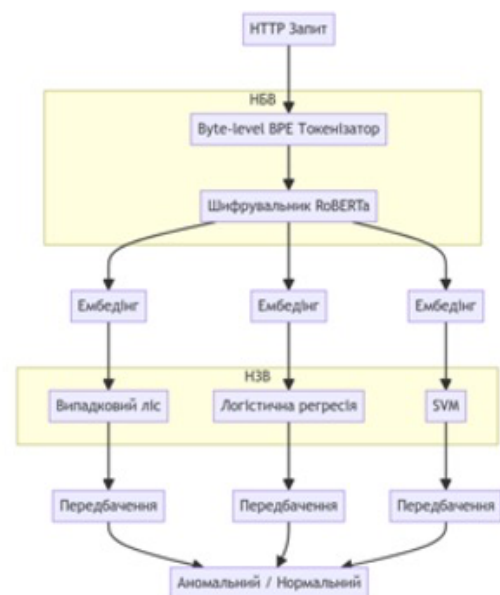
- закодували символи CR (повернення каретки) та LF (переведення рядка) у вигляді буквених рядків "\r" та "\n", оскільки це допоможе моделі відстежувати деякі типи атак.
- випадковим чином змінили значення поля "Host" на будь-яке значення з нормального набору даних.
- усі значення поля «URL» в запитих починалися з одного з наступних рядків: "/DVWA/vulnerabilities/xss", "DVWA/dvwa", "/DVWA". Було прийнято рішення видалити ці частини URL.
- у кожному аномальному запиті був заголовок "Upgrade-Insecure-Requests", який ми також видалили.

5

Слайд 6

Опис алгоритму

Для того щоб отримати найбільш якісний прогноз, ми вирішили використовувати мовну модель RoBERTa для генерування ознак. Тобто, ми використали кодувальник, який створює векторний простір ознак, який передається на вхід класифікаторам для подальшого прогнозування мітки класу.



6

Слайд 7

Мовна модель RoBERTa

Ми вибрали розмір моделі **Base**, яка є збалансованим варіантом з точки зору продуктивності та ресурсів, необхідних для її навчання. Ця модель має наступні характеристики:

- максимальна довжина послідовності – 512 токенів;
- кількість шарів – 6;
- кількість голів уваги – 12.

Модель була навчена протягом 10 епох. Розмір батча становить 32 зразки.

7

Слайд 8

Класифікатори та їх гіперпараметри

Логістична регресія

- `max_iter=1000`

Випадковий ліс

- `n_estimators=100`
- `random_state=42`

Метод опорних векторів

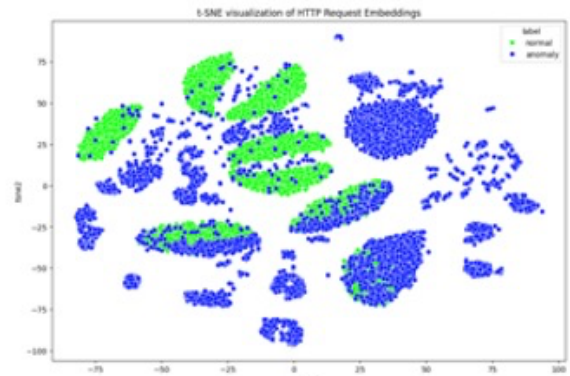
- `probability=True`

8

Слайд 9

Згенерований векторний простір

За допомогою алгоритму t-SNE, ми візуалізували утворений простір ембедингів. Класи добре розділені навіть у просторі меншої розмірності. Це свідчить про те, що схожі HTTP-запити фактично групуються разом, що дозволяє досліджувати околиці будь-якого зразка.



9

Слайд 10

Сусідні запити у просторі

Ми обрали запит № 8001 та подивились на сусідні запити (в сенсі евклідової відстані) до нього. Як видно, атаки SQL ін'єкцій насправді близькі одна до одної. Цікаво, що в подальшому оточенні зразка містяться інші типи ін'єкцій – команди операційної системи.

№	Запит
1	id=2&nombre=Jam%F3n+ib%E9rico&precio=100&cantidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al+carrito
2	id=1%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&nombre=Jam%F3n+ib%E9rico&precio=100&cantidad=97&B1=A%F1adir+al+carrito
3	id=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&nombre=Jam%F3n+ib%E9rico&precio=85&cantidad=21&B1=A%F1adir+al+carrito
4	id=2&nombre=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&precio=100&cantidad=97&B1=A%F1adir+al+carrito
5	id=2&nombre=Jam%F3n+ib%E9rico&precio=%27%3Bwaitfor+delay+%270%3A0%3A15%27%3B--&cantidad=17&B1=A%F1adir+al+carrito
6	id=1&nombre=Jam%F3n+ib%E9rico&precio=100&cantidad=%27%29%3Bwaitfor+delay+%270%3A0%3A15%27%3B--&B1=A%F1adir+al+carrito
6	id=3%27%3Bwaitfor+delay+%270%3A0%3A15%27%3B--&nombre=Jam%F3n+ib%E9rico&precio=39&cantidad=17&B1=A%F1adir+al+carrito
7	id=1&nombre=Jam%F3n+ib%E9rico&precio=%3C%21---%23EXEC+cmd%3D%22ls+%2F%22---%3E&cantidad=71&B1=A%F1adir+al+carrito

10

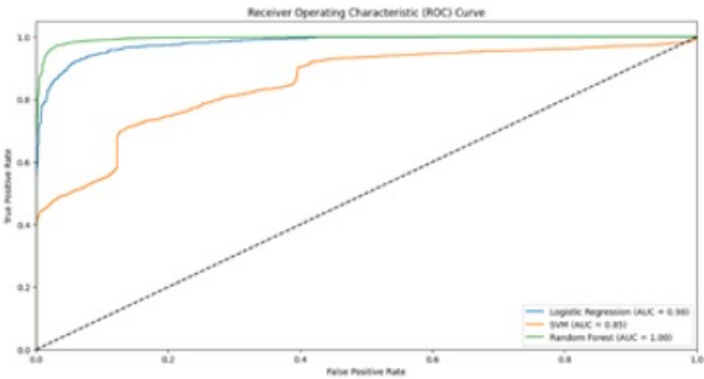
Порівняння класифікаторів

Спираючись на пораховані метрики якості, можна зробити висновок, що випадковий ліс показав себе найкраще в рамках виконання поставленої задачі.

Модель	Точність (Accuracy)	Влучність (Precision)	Повнота (Recall)	F1-міра
Логістична регресія	0.9188	0.9638	0.8880	0.9244
SVM	0.7547	0.8715	0.6580	0.7499
Випадковий ліс	0.9664	0.9836	0.9557	0.9695

ROC крива

Проаналізувавши ROC криву, ми також можемо побачити, що випадковий ліс показує себе найкраще. Коефіцієнт AUC (площа під ROC кривою) для випадкового лісу дорівнює 1, що є ідеальним результатом.



Слайд 13

Наукова новизна

- Запропонування нових методів обробки та підготовки даних, які включають спеціалізовані методи токенізації (BBPE) та виявлення аномалій (RoBERTa), які допоможуть покращити якість класифікації.
- Розробка системи, яка може бути використана в реальних умовах для захисту мереж від шкідливих запитів, а також проведення експериментів та тестування на реальних даних.
- Використання моделі для виявлення нових, раніше невідомих паттернів шкідливої активності, що можуть бути використані для подальшого покращення систем безпеки.

13

Слайд 14

Висновки

Поставлену задачу було повністю розв'язано, мовну модель RoBERTa було налаштовано на зібраних HTTP запитах та впроваджено алгоритм генерації векторного простору. Векторний простір успішно використовується в якості вхідних даних для класифікаторів. Покращити наведений алгоритм можна за допомогою використання більш потужних обчислювальних машин, що забезпечить більш якісний підбір гіперпараметрів для класифікаторів та навчання протягом більшої кількості епох.

Недоліками даного алгоритму можна вважати велику кількість часу, необхідну для навчання. Ця проблема вирішується шляхом використання відеокарт з більшою кількістю ядер для паралельного обчислювання, що забезпечить одночасне навчання декількох шарів мовної моделі.

14

Слайд 15

Подальші дослідження

Дана робота може бути використана на рівні інтернет-провайдерів для моніторингу та захисту мережевої інфраструктури від шкідливого трафіку, забезпечуючи безпеку користувачів та їх даних. Також, за допомогою даної роботи можливо створити комплексне рішення для кібербезпеки, яке використовує сучасні методи мовного моделювання для виявлення та блокування шкідливих HTTP запитів в реальному часі. Цей продукт допомагає забезпечити безпеку мережевих інфраструктур, захищаючи від широкого спектру кібератак.

15

Слайд 16

Дякую за увагу!

16