МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Аліна САВЧЕНКО
«_____» _____ 2024р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ **"БАКАЛАВР"**

Тема: "Система автоматичного розпізнавання та фільтрація спаму в коментарях веб-сайтів"

Виконавець: студент групи УС-411 Дорогань Владислав Анатолійович

Керівник: к.т.н., доцент Колісник Олена Василівна

Нормоконтролер: _____ Олександр ШЕВЧЕНКО
(підпис)

Київ – 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«____»_____2024р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи (проєкту)

Дороганя Владислава Анатолійовича

1. Тема роботи: «Система автоматичного розпізнавання та фільтрація спаму в коментарях веб-сайтів» затверджена наказом ректора від "05" квітня 2024 р. за № 517/ст.

2. Термін виконання роботи: 06.05.2024 – 07.06.2024

3. Вихідні данні до роботи: проблема спаму на веб-ресурсах; існуючі методи розпізнавання спаму; набори даних з прикладами спам-повідомлень та легітимного контенту; завдання створення системи виявлення спаму на основі машинного навчання; необхідність аналізу підходів до фільтрації спаму.

4. Зміст пояснювальної записки: вступ, огляд методів та алгоритмів фільтрації повідомлень, аналіз найпопулярніших методів розпізнавання спаму, вибір технологій для розробки, архітектура проекту, опис роботи програми, висновки.

5. Перелік обов'язкового графічного матеріалу: скріншоти роботи спам-фільтру та веб-сайту, схеми та графіки роботи методів розпізнавання спаму**.**

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу, створення плану кваліфікаційної роботи та побудова плану-графіку виконання робіт.	06.05.2024 – 07.05.2024	
2.	Розроблення та затвердження календарного плану виконання кваліфікаційної роботи.	07.05.2024 – 08.05.2024	
3.	Проведення консультації з науковим керівником.	09.05.2024	
4.	Огляд та аналіз наукової літератури по темі кваліфікаційної роботи та написання Розділу 1.	10.05.2024 – 16.05.2024	
5.	Написання Розділу 2 кваліфікаційної роботи.	17.05.2024 – 22.05.2024	
6.	Написання Розділу 3 кваліфікаційної роботи.	23.05.2024 – 30.05.2024	
7.	Завершення створення пояснювальної записки кваліфікаційної роботи.	31.05.2024 – 02.06.2024	
8.	Оформлення та друк пояснювальної записки. Створення презентації, доповіді та підготовка до захисту кваліфікаційної роботи.	03.06.2024 – 05.06.2024	
9.	Підготовка матеріалів кваліфікаційної роботи для передачі секретарю ДЕК (папка, конверт, диск із файлом диплому, рецензія, відгук).	06.06.2024 – 07.06.2024	

7. Дата видачі завдання: «6» травня 2024 р.

Керівник кваліфікаційної роботи _____ Олена КОЛІСНИК
(підпис)

Завдання прийняв до виконання _____ Владислав ДОРОГАНЬ
(підпис)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Система автоматичного розпізнавання та фільтрація спаму в коментарях веб-сайтів»: 80 с., 32 рис., 28 літературних джерел.

Об'єкт дослідження: Процес створення автоматичної системи фільтрації спаму в коментарях.

Предмет дослідження: різні методи та алгоритми розпізнавання спаму, їх переваги, недоліки та особливості застосування.

Мета роботи: Метою даної роботи є розробка ефективної системи для виявлення спаму в коментарях веб-ресурсів із застосуванням передових алгоритмів фільтрації спам-повідомлень.

Методи дослідження: аналіз вимог до фільтрації повідомлень, порівняння ефективності методів розпізнавання, тренування моделі розпізнавання, тестування та реалізація програмного прототипу.

Наукова новизна полягає у розробці системи для виявлення спаму на веб-ресурсах з використанням моделі штучного інтелекту на основі порівняльного аналізу різних методів та алгоритмів розпізнавання спаму, що дозволяє підвищити ефективність захисту веб-ресурсів від спаму.

В результаті виконання роботи створено та натреновано модель штучного інтелекту для виявлення спаму, проаналізовано переваги і недоліки різних підходів до фільтрації спаму. Також реалізовано програмну систему з інтеграцією натренованої моделі, що може бути використана для захисту веб-ресурсів від спаму та підвищення ефективності їх роботи.

Результати кваліфікаційної роботи рекомендується використовувати під час створення систем захисту веб-сайтів.

СПАМ, ФІЛЬТРАЦІЯ ПОВІДОМЛЕННЬ, РОЗПІЗНАВАННЯ СПАМУ, МАШИННЕ НАВЧАННЯ, ВЕБ-САЙТ, КОМЕНТАРІ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ 7
ВСТУП 9
РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО СИСТЕМИ ФІЛЬТРАЦІЇ СПАМУ 10
1.1. Визначення та значення систем фільтрації спаму 10
1.2. Вплив спаму на продуктивність та безпеку інтернет-ресурсів 12
1.3. Огляд алгоритмів та підходів до розпізнавання спаму 13
1.3.1. Машинне навчання 13
1.3.2. Статистичні методи 15
1.3.3. Фільтрація на основі правил 17
1.4. Класифікація методів фільтрації спаму 18
1.4.1. Фільтрація на основі контенту 18
1.4.2. Фільтрація на основі списків 20
1.4.3. Гібридні методи фільтрації спаму 21
1.5. Висновки до першого розділу 23
РОЗДІЛ 2. АНАЛІЗ МЕТОДІВ РОЗПІЗНАВАННЯ СПАМУ 25
2.1. Алгоритм наївного Байєса 25
2.2. Метод опорних векторів 32
2.3. Метод нейронних мереж 38
2.4. Висновки до другого розділу 44
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ 45
3.1. Опис використаних інструментів при розробці 45
3.1.1. HTML та CSS 45
3.1.2. JavaScript 47

3.1.3. TensorFlow.js 48
3.1.4. Node.js та Socket.io 50
3.2. Перетренування моделі розпізнавання спаму 53
3.3. Створення веб-сайту та підключення його до серверу 55
3.4. Створення можливості залишати коментарі на веб-сайті 58
3.5. Завантаження та тестування моделі на основі штучного інтелекту 60
3.6. Реалізація роботи спам фільтру в реальному часі 68
3.7. Висновки до третього розділу 75
ВИСНОВКИ 77
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ 78

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ANN	Artificial Neural Network - штучна нейронна мережа, обчислювальна модель, натхненна біологічними нейронними мережами.
AJAX	Asynchronous JavaScript and XML - набір технологій, що дозволяє веб-додаткам асинхронно обмінюватися даними з сервером.
CNN	Convolutional Neural Network - згорткова нейронна мережа, клас глибоких нейронних мереж, що застосовується для аналізу візуальних образів.
CSS	Cascading Style Sheets
DMC	Dynamic Markov Compression - метод стиснення даних, заснований на Марковських моделях.
DNS	Domain Name System - система доменних імен, розподілена система для присвоєння імен вузлам мережі.
DOM	Document Object Model - об'єктна модель документа, програмний інтерфейс для HTML, XML та SVG документів.
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol - протокол передачі гіпертексту, основний протокол для передачі даних у всесвітній павутині.
IDE	Integrated Development Environment - інтегроване середовище розробки, програмне забезпечення для розробки додатків.
IoT	Internet of Things - інтернет речей, концепція обчислювальної мережі фізичних об'єктів.
JSON	JavaScript Object Notation - текстовий формат обміну даними, заснований на JavaScript.
OCR	Optical Character Recognition - оптичне розпізнавання символів, технологія розпізнавання тексту на зображеннях.
PPM	Prediction by Partial Matching - метод стиснення даних на основі передбачення частини вхідного потоку.

RBL	Real-time Blackhole List
RFC	Random Forest Classifier
RNN	Recurrent Neural Network - рекурентна нейронна мережа, для обробки послідовних даних.
SVM	Support Vector Machine
Tf-idf	Term Frequency-Inverse Document Frequency - числове представлення важливості слова у документі.
WASM	WebAssembly

ВСТУП

Системи фільтрації спаму відіграють надзвичайно важливу роль у сучасному інтернет-просторі та є невід'ємною складовою сучасних веб-сервісів та поштових систем, забезпечуючи безпеку, продуктивність та дотримання стандартів у сфері інформаційної безпеки. Їх значення полягає у багатоаспектному підході до захисту користувачів і поштових систем від небажаного контенту. Подивимось детальніше на основні аспекти цієї проблеми.

Перш за все, системи фільтрації спаму є невід'ємною частиною захисту репутації веб-сайтів та поштових сервісів. Наявність спаму може суттєво підірвати довіру користувачів до певного ресурсу, що впливає на активність взаємодії з ним та можливості розвитку.

Важливим аспектом є забезпечення продуктивності користувачів шляхом зменшення обсягу непотрібної інформації в їхніх поштових скриньках. Час, який витрачається на фільтрацію спаму та видалення його, може бути використаний більш ефективно.

Також, фільтрація спаму має велике значення для запобігання поширенню шкідливого програмного забезпечення та фішингових атак. Вміст спаму часто містить посилання на шкідливі веб-сайти або віруси, що може призвести до серйозних проблем з безпекою.

Важливою є і економічна сторона питання. Періодичне видалення спаму допомагає зберігати дисковий простір на серверах і економити ресурси компаній та провайдерів[4].

Крім того, у світлі регуляційних вимог і стандартів безпеки, фільтрація спаму стає необхідністю для багатьох організацій та державних установ, що відповідають за збереження конфіденційності та безпеку даних.

Не можна забувати й про те, що ефективна фільтрація спаму допомагає зменшити навантаження на мережеві ресурси та підтримувати стабільну пропускну здатність каналів зв'язку[5].

РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО СИСТЕМИ ФІЛЬТРАЦІЇ СПАМУ

1.1. Визначення та значення систем фільтрації спаму

Системи фільтрації спаму є критично важливими для забезпечення безпеки, продуктивності та нормального функціонування інтернет-ресурсів, зокрема веб-сайтів з можливістю коментувати.

Спам – це будь-який небажаний масовий контент, що розсилається через електронну пошту, текстові повідомлення, месенджери, соціальні мережі чи інші канали комунікації без згоди одержувачів. "Спамом" спочатку називалася консерва зі свинини та шинки зі спеціями. Завдяки англійській комічній групі Monty Python, відомій своїм незвичайним гумором, через їх постійне повторення у жартах одного й того ж слова, «спам» став синонімом масової небажаної електронної пошти[6].

Спам може містити рекламу, шахрайські пропозиції, шкідливі посилання або вкладення з метою нелегального заробітку, крадіжки даних чи поширення шкідливого програмного забезпечення.

Захист від спаму — це комплекс заходів, спрямованих на боротьбу з небажаними масовими розсилками. Часто електронні адреси потрапляють до списків розсилки спаму внаслідок їх придбання або викрадення хакерами. Відправники спаму розраховують, що навіть якщо їхні повідомлення спрацюють для невеликої частки одержувачів, маркетингова кампанія або шахрайська схема все одно будуть успішними[7].

Системи фільтрації спаму (Спам-фільтр) — це програма, призначена для виявлення вхідних небезпечних електронних листів та коментарів від зловмисників або маркетологів. Зловмисники часто використовують електронні листи або коментарі на сторінках, які видаються корисними або попереджають про уявну загрозу, але насправді є лише приманкою, щоб змусити користувачів натиснути посилання для завантаження шкідливого програмного забезпечення або перенаправлення на небезпечні сайти[4].

Вони захищають як поштові скриньки користувачів, так і розділи коментарів на веб-сайтах від спам-повідомлень. Основне значення таких систем полягає в наступному:

- Захист репутації та авторитетності веб-ресурсів шляхом недопущення публікації спаму в коментарях, що створює враження неякісного або небезпечного контенту;
- Підвищення продуктивності користувачів і співробітників за рахунок зменшення часу на обробку непотрібної інформації у вхідних повідомленнях;
- Запобігання поширенню шкідливого програмного забезпечення, фішингових атак та крадіжок даних через спам-листи або спам-коментарі;
- Економія дискового простору на серверах шляхом своєчасного видалення великих обсягів спаму;
- Дотримання нормативних вимог і стандартів безпеки в певних галузях чи юрисдикціях завдяки фільтрації небезпечного контенту;
- Збереження конфіденційності даних користувачів від витоку через спам-повідомлення;
- Зменшення навантаження на мережеві ресурси й інфраструктуру веб-сайтів внаслідок обробки величезних потоків спаму.

Ефективні системи фільтрації спаму є життєво необхідними для забезпечення безпроблемної роботи інтернет-ресурсів, захисту клієнтських даних, дотримання вимог безпеки та підвищення задоволеності користувачів. Вони допомагають створювати безпечне та продуктивне середовище для комунікації, обміну контентом і ведення бізнесу в онлайн-просторі[5].

1.2. Вплив спаму на продуктивність та безпеку інтернет-ресурсів

Спам у коментарях веб-сайтів може мати руйнівний вплив на продуктивність та безпеку інтернет-ресурсів, що призводить до значних фінансових втрат та шкоди репутації[4].

По-перше, велика кількість спам-повідомлень сильно навантажує сервери сайту, оскільки їх необхідно обробляти, зберігати та відфільтровувати. Це може суттєво сповільнити швидкість завантаження сторінок, призвести до збоїв у роботі сайту та погіршити користувацький досвід відвідувачів. Як наслідок, люди можуть відмовитися від подальшого відвідування ресурсу, що негативно позначиться на трафіку, конверсії та прибутковості бізнесу.

По-друге, спам-коментарі часто містять шкідливі посилання, вкладення або коди, які можуть бути використані зловмисниками для поширення вірусів, шахрайства, крадіжки даних чи здійснення інших кібератак. Це ставить під загрозу безпеку як самого сайту, так і його відвідувачів, особливо якщо спам не виявляється та не видаляється вчасно. Наслідки можуть бути катастрофічними, включаючи витік конфіденційної інформації, втрату клієнтської бази, фінансові збитки та репутаційні ризики[8].

Крім того, значна присутність спаму на сайті може ввести в оману пошукові системи та драматично знизити його ранжування в результатах пошуку. Оскільки пошукові роботи розглядають коментарі як частину контенту, велика кількість спаму може бути інтерпретована як ознака низької якості ресурсу. Це призведе до зниження видимості сайту в інтернеті та, як наслідок, до втрати потенційних відвідувачів та клієнтів.

Також постійна модерація та видалення спам-коментарів вимагає значних зусиль, часу та ресурсів від адміністраторів сайту. Це може відволікати їх від виконання інших важливих завдань, пов'язаних з розвитком та підтримкою ресурсу, що негативно позначається на загальній продуктивності роботи. Великі компанії можуть наймати додатковий персонал для боротьби зі спамом, що збільшує

операційні витрати, тоді як малі підприємства можуть не мати такої можливості, що ставить їх у невигідне становище[5].

Вплив спаму у коментарях на продуктивність та безпеку веб-сайтів є надзвичайно серйозним і може мати катастрофічні наслідки для бізнесу. Саме тому впровадження ефективних систем виявлення та фільтрації спаму є критично важливим завданням для підтримки стабільної та безпечної роботи будь-якого інтернет-ресурсу[5,8].

1.3. Огляд алгоритмів та підходів до розпізнавання спаму

1.3.1. Машинне навчання

Машинне навчання відіграє ключову роль у сучасних системах розпізнавання та фільтрації спам-повідомлень. Замість використання статичних правил, алгоритми машинного навчання можуть самостійно виявляти складні закономірності та ознаки спаму на основі аналізу великих наборів даних[9].

Ось деякі основні підходи та алгоритми машинного навчання, що застосовуються для виявлення спаму:

Наївний байєсовський класифікатор

Це один з найпопулярніших алгоритмів для фільтрації спаму. Він ґрунтується на байєсовській теоремі та використовує частоту появи окремих слів або токенів у спам- та легітимних повідомленнях для обчислення ймовірності того, що нове повідомлення є спамом. Незважаючи на його простоту, наївний байєсовський класифікатор демонструє високу точність виявлення спаму (до 98%) і добре масштабується. Але він вразливий до атак на основі введення випадкових слів (байєсовського отруєння)[1].

Дерева рішень

Дерева рішень є потужними моделями машинного навчання для вирішення завдань класифікації. Для виявлення спаму дерево рішень будує ієрархічну структуру

правил на основі різних ознак повідомлень (наприклад, слова, заголовки, відправник тощо). Це дозволяє врахувати складні комбінації ознак. Більш просунуті версії дерев рішень, такі як випадкові ліси та градієнтний бустинг, підвищують точність і стійкість до перенавчання[1].

Нейронні мережі

Глибокі нейронні мережі стали потужним інструментом для багатьох завдань обробки природної мови, включно з виявленням спаму. Вони можуть ефективно виявляти складні закономірності в текстових даних, враховуючи контекст і семантику. Популярними архітектурами нейронних мереж для класифікації спаму є рекурентні нейронні мережі (RNN), згорткові нейронні мережі (CNN) та трансформери[9].

Метод опорних векторів (SVM)

SVM є потужним алгоритмом машинного навчання для класифікації та регресії. Він намагається побудувати гіперплощину, яка максимально розділяє спам і легітимні повідомлення, спираючись на їхні ознаки (наприклад, tf-idf вектори слів). SVM часто демонструє високу точність на завданнях фільтрації спаму завдяки своїй стійкості до перенавчання та здатності ефективно працювати з високовимірними даними[2].

Навчання з підкріпленням

Підхід навчання з підкріпленням дозволяє моделям машинного навчання самостійно виявляти оптимальні стратегії для досягнення певної мети, отримуючи винагороди або штрафи за свої дії. Це робить його перспективним для боротьби зі спамерами, які постійно адаптують свої методи для обходу фільтрів. Моделі навчання з підкріпленням можуть динамічно коригувати свої стратегії розпізнавання спаму на основі зворотного зв'язку від користувачів[2].

Крім того, машинне навчання застосовується для виявлення спаму не лише за текстовим вмістом, але й за візуальними ознаками, використовуючи методи комп'ютерного зору для аналізу зображень та відео у повідомленнях[7].

Незважаючи на свою ефективність, підходи машинного навчання не є досконалими і мають певні недоліки. Вони можуть бути вразливими до спеціально

розроблених атак, що вводять випадкові шуми для обходу фільтрів. Крім того, для досягнення високої точності моделі машинного навчання часто потребують великих наборів якісних тренувальних даних, які не завжди доступні[9].

1.3.2. Статистичні методи

Статистичні методи є потужними інструментами для фільтрації спаму в коментарях веб-сайтів. Вони використовують ймовірнісні моделі та теорію інформації для виявлення спаму на основі статистичних властивостей тексту. Ці методи не покладаються на специфічні правила чи ключові слова, а натомість аналізують загальні статистичні характеристики тексту для визначення його легітимності.

Один із найпоширеніших статистичних підходів до фільтрації спаму - це стиснення даних. Ідея полягає в тому, що легітимний текст, як правило, має більш передбачувану структуру і може бути ефективніше стиснутий за допомогою алгоритмів стиснення даних. Натомість спам часто містить більш хаотичний і непередбачуваний текст, що робить його менш стисливим[3].

Процес фільтрації спаму за допомогою стиснення даних відбувається таким чином:

1. Створюються дві статистичні моделі стиснення: одна для легітимних коментарів, інша - для спаму. Ці моделі навчаються на наборі тренувальних даних;
2. Коли новий коментар надходить, він стискається за допомогою обох моделей, і обчислюється ступінь стиснення для кожної моделі;
3. Коментар класифікується як спам чи легітимний, залежно від того, яка модель забезпечує кращий ступінь стиснення[3].

Одним із найвідоміших алгоритмів стиснення даних, що застосовується для фільтрації спаму, є алгоритм динамічного стиснення Маркова (DMC). Він моделює текст як скінченний автомат, де стани представляють контексти, а переходи між станами визначають ймовірності символів на основі їх контекстів. DMC адаптивно

будує цю модель під час обробки тексту, що дозволяє йому ефективно стискати тексти з різними статистичними властивостями[1].

Іншим популярним алгоритмом є прогнозування за частковим збігом (PPM). Він використовує статистику частоти зустрічальності символів у різних контекстах для прогнозування наступного символу в послідовності. На основі цих прогнозів алгоритм PPM може ефективно стискати текст.

Статистичні методи фільтрації спаму мають кілька переваг:

- *Гнучкість*. Вони можуть адаптуватися до нових типів спаму без необхідності оновлення правил чи переналаштування;
- *Стійкість до шумів*. Статистичні методи є менш чутливими до шумів і намагань спамерів приховати спам, оскільки вони покладаються на загальні статистичні властивості тексту, а не на специфічні ключові слова чи формат;
- *Висока точність*. При належному навчанні статистичні методи можуть досягати високої точності класифікації спаму.

Проте статистичні методи також мають певні обмеження:

- *Потреба в навчальних даних*. Для ефективного навчання моделей потрібні великі набори даних для легітимного контенту та спаму;
- *Обчислювальна складність*. Деякі статистичні алгоритми, такі як DMC або PPM, можуть бути обчислювально складними, особливо для довгих текстів;
- *Вразливість до змін у статистичних властивостях*. Якщо статистичні властивості спаму або легітимного контенту суттєво змінюються, ефективність моделей може знизитися, доки вони не будуть перенавчені.

Статистичні методи фільтрації спаму є доволі потужним інструментом для боротьби зі спамом у коментарях веб-сайтів, проте мають і недоліки, на які потрібно зважати. Вони часто використовуються в комбінації з іншими підходами, такими як фільтрація на основі правил або машинне навчання, для досягнення максимальної ефективності фільтрації[3].

1.3.3. Фільтрація на основі правил

Фільтрація спаму на основі правил є одним із найпопулярніших та найбільш гнучких підходів для боротьби зі спамом у коментарях веб-сайтів. На відміну від інших методів, які покладаються на машинне навчання чи статистичний аналіз, фільтрація за правилами дозволяє адміністраторам вручну визначати конкретні критерії, за якими буде оцінюватися вміст коментарів.

Цей метод ґрунтується на створенні набору правил, що визначають, які елементи вважатимуться ознаками спаму. Правила можуть бути різноманітними та охоплювати широкий спектр аспектів, таких як присутність певних ключових слів, посилань, форматування тексту, частота публікації коментарів від одного користувача чи IP-адреси та багато іншого[10].

Основні переваги використання фільтрації на основі правил для боротьби зі спамом у коментарях:

- *Гнучкість та можливість налаштування*. Адміністратори можуть створювати власні правила, спеціально підлаштовані під специфічні потреби веб-ресурсу та типи спаму, з якими вони стикаються. Це дозволяє забезпечити високу точність фільтрації та мінімізувати помилкові спрацьовування;
- *Легкість інтерпретації*. На відміну від деяких складних методів машинного навчання, правила фільтрації є досить простими та зрозумілими для людини. Це полегшує налагодження та оновлення системи фільтрації;
- *Цільове блокування*. Фільтрація за правилами дозволяє адміністраторам створювати спеціальні правила для блокування певних типів спаму, наприклад, реклами конкретних продуктів чи послуг, недоречного або образливого контенту тощо;
- *Висока швидкість обробки*. Порівняно з деякими складнішими методами, перевірка коментарів на відповідність набору правил є відносно швидким процесом, що дозволяє ефективно фільтрувати спам у режимі реального часу.

Разом з тим, фільтрація на основі правил має й певні недоліки:

- *Необхідність постійного оновлення*. Оскільки спамери постійно змінюють свої тактики, правила фільтрації потребують регулярного оновлення та доповнення новими критеріями для забезпечення ефективності;
- *Можливість обходу*. Правила фільтрації можуть бути обійдені спамерами, якщо вони достатньо обізнані зі способом роботи системи та навмисно уникають використання триггерів, прописаних у правилах;
- *Складність налаштування*. Для створення ефективного набору правил потрібно глибоке розуміння характеристик спаму та специфіки веб-ресурсу.

Неправильно сконфігуровані правила можуть призводити до пропускання значної кількості спаму або блокування легітимного контенту.

Фільтрація на основі правил є важливим інструментом у боротьбі зі спамом у коментарях веб-сайтів. Часто цей підхід використовується в комбінації з іншими методами, такими як машинне навчання чи статистичний аналіз, для досягнення максимальної ефективності[9].

1.4. Класифікація методів фільтрації спаму

1.4.1. Фільтрація на основі контенту

На сьогоднішній день існує велика кількість різноманітних мов Фільтрація на основі контенту є одним з найпоширеніших підходів до виявлення спам-повідомлень. Ці методи аналізують сам зміст електронного листа або коментаря з метою ідентифікації ймовірного спаму[4].

Існує кілька різновидів та технік фільтрації на основі контенту:

Фільтри на основі слів (Word-based filters)

Це найпростіший тип фільтрів контенту. Вони блокують будь-які повідомлення, що містять певні заздалегідь визначені слова чи фрази, характерні для спаму (наприклад, "знижка", "розпродаж" тощо). Однак такі фільтри можуть

генерувати помилкові спрацьовування для легітимних листів чи коментарів, якщо ті випадково містять заблоковані слова. [4]

Евристичні фільтри (Heuristic filters)

На відміну від фільтрів на основі слів, евристичні фільтри враховують декілька термінів та фраз у повідомленні. Вони присвоюють оцінки (бали) підозрілим словам, що часто зустрічаються в спамі, після чого підсумовують ці бали. Якщо загальна оцінка перевищує визначений поріг, повідомлення класифікується як спам[11].

Фільтри на основі "мішка слів" (Bag of words)

Це модель, яка представляє текст у вигляді multiplicity (мішка) його слів, ігноруючи при цьому граматику та навіть порядок слів. Частота появи слів у документі використовується як ознака для машинного навчання алгоритмів класифікації спам/неспам[11].

Фільтри на основі OCR

Ця група методів застосовує оптичне розпізнавання символів (OCR) для витягування тексту з вкладених у повідомлення зображень і подальшого його аналізу на предмет спаму[6].

Фільтри на основі аналізу зображень

Окрім аналізу текстового вмісту, деякі фільтри використовують технології комп'ютерного зору для виявлення спаму в самих зображеннях за допомогою аналізу їх візуальних ознак, текстур, країв тощо[6].

Фільтри на основі аналізу заголовків

Ці методи фокусуються виключно на аналізі полів заголовків (від-кого, тема та інші) електронних листів для пошуку ознак спаму[12].

Переваги фільтрації на основі контенту:

- Здатність виявляти нові види спаму, оскільки не покладається лише на списки;
- Можливість адаптуватися до змін у шаблонах та техніках поширення спаму;
- Гнучкість у застосуванні до різних типів контенту (текст, зображення тощо).

Недоліки фільтрації на основі контенту:

- Ускладнення обробки зашифрованого або мультимедійного контенту;

- Потенційні помилкові спрацьовування;
- Можлива необхідність періодичного переналаштування та оновлення моделей.

Фільтрація на основі контенту є потужним і різноманітним класом методів протидії спаму, що активно розвивається та удосконалюється завдяки новим досягненням у сфері штучного інтелекту та машинного навчання[11,12].

1.4.2. Фільтрація на основі списків

Фільтрація на основі списків є одним з найдавніших і найпоширеніших підходів до виявлення та блокування спаму. Ці методи покладаються на використання заздалегідь підготовлених переліків (списків) відомих джерел спаму, таких як IP-адреси, домени, адреси електронної пошти тощо[4,9].

Існують різні типи списків для фільтрації спаму:

Чорні списки (Blacklists)

Чорні списки містять перелік об'єктів (IP-адрес, доменів, адрес електронної пошти), які вважаються джерелами спаму або іншої небезпечної активності. Будь-які повідомлення від джерел, включених до чорного списку, автоматично блокуються або відхиляються. Для наповнення чорних списків використовуються різні методи, наприклад, пастки для спаму, ручне позначення користувачами, виявлення порушень правил RFC тощо[11].

Білі списки (Whitelists)

Білі списки діють протилежно чорним - вони містять перелік довірених та дозволених джерел. Система дозволяє отримувати повідомлення лише від об'єктів, включених до білого списку, а все інше блокується. Білі списки можуть формуватися вручну адміністраторами або на основі репутаційних даних від третіх сторін[9].

Сірі списки (Greylists)

Сірі списки використовуються для тимчасової блокування невідомих джерел, що не входять ані до білих, ані до чорних списків. Повідомлення від таких джерел тимчасово відхиляються, а якщо вони надсилаються повторно протягом певного

періоду часу (як це характерно для легітимної пошти), вони пропускаються. Такий підхід дозволяє виявляти спам від одноразових джерел[12].

Публічні списки (RBL)

RBL - це публічні списки IP-адрес та доменів, які ідентифіковані як джерела спаму. Системи перевіряють вхідні повідомлення відносно RBL і блокують їх, якщо відправник присутній у цих списках[9].

DNSBL (DNS-based Blackhole List)

DNSBL є різновидом RBL, де чорні списки реалізовані на рівні DNS. Система виконує зворотний пошук DNS для IP-адреси відправника і блокує повідомлення, якщо ця IP присутня в DNSBL[12].

Переваги фільтрації на основі списків:

- Простота реалізації та використання;
- Висока швидкість фільтрації з мінімальним навантаженням на ресурси;
- Ефективність проти відомих джерел спаму.

Недоліки фільтрації на основі списків:

- Неможливість виявити нові, невідомі джерела спаму;
- Потреба в регулярному оновленні списків;
- Потенційні помилкові спрацьовування на легітимні джерела;
- Складність підтримки власних локальних списків.

Фільтри на основі списків часто використовуються у поєднанні з іншими типами фільтрів, такими як фільтрація на основі контенту, для досягнення більш високої ефективності виявлення спаму. Також їх застосування має бути гнучким та налаштованим відповідно до специфічних вимог системи чи організації[12].

1.4.3. Гібридні методи фільтрації спаму

Гібридні методи фільтрації спаму поєднують різні підходи та алгоритми для підвищення точності розпізнавання спам-повідомлень. Деякі з популярних гібридних методів включають:

Мішок слів (Bag of Words) з машинним навчанням

Це поєднання методу "мішка слів" для представлення текстових даних з алгоритмами машинного навчання, такими як наївний байєсовський класифікатор, логістична регресія або метод опорних векторів. Спочатку текст перетворюється на числові вектори за допомогою "мішка слів", а потім ці вектори використовуються для навчання моделі машинного навчання розпізнавати спам[1].

Правила на основі знань з машинним навчанням

Цей підхід включає ручне визначення набору правил або евристик для ідентифікації спам-повідомлень (наприклад, виявлення спам-слів, підозрілих посилань тощо). Потім додатково застосовується модель машинного навчання для кращого розпізнавання складніших випадків спаму[7].

Об'єднання контентних та списочних фільтрів

Цей метод поєднує переваги фільтрів, заснованих на контенті (аналіз тексту повідомлень), з перевагами списочних фільтрів (чорні, білі та сірі списки). Спочатку застосовуються списочні фільтри, а потім контентні фільтри для повідомлень, які не були класифіковані списками[12].

Підходи з використанням візуальних ознак

Деякі гібридні методи спираються на візуальні ознаки спам-повідомлень, наприклад, аналіз вбудованих зображень за допомогою методів комп'ютерного зору. Ці візуальні ознаки потім поєднуються з текстовими для підвищення точності класифікації[6].

Колективна фільтрація

Цей підхід поєднує методи колективної фільтрації, де повідомлення класифікуються на основі відгуків великої кількості користувачів, із традиційними методами фільтрації спаму, заснованими на контенті чи списках[12].

Переваги гібридних методів фільтрації спаму:

- Вища точність розпізнавання спаму порівняно з окремими методами завдяки комбінуванню різних підходів та джерел даних;

- Компенсація недоліків одних методів перевагами інших, наприклад, поєднання контентного аналізу з використанням списків дозволяє одночасно виявляти відомі джерела спаму та новий спам;
- Гнучкість та можливість адаптації до різних типів спаму та сценаріїв використання шляхом варіювання компонентів гібридного підходу;
- Підвищена стійкість до спроб обходу спамерами окремих методів фільтрації за рахунок комплексного підходу.

Недоліки гібридних методів:

- Збільшена обчислювальна складність та потреба в більших обчислювальних ресурсах порівняно з окремими методами;
- Складність налаштування та оптимізації різних компонентів гібридного підходу для ефективної співпраці;
- Можливість конфліктів або протиріч між окремими компонентами, що може призвести до помилкових спрацьовувань;
- Потенційно вища вартість розробки та обслуговування складніших гібридних систем.

Гібридні методи фільтрації спаму широко застосовуються завдяки їхній підвищеній ефективності та здатності адаптуватися до мінливої природи спаму. Вони особливо корисні для захисту критично важливих інтернет-ресурсів, де точність виявлення спаму має вирішальне значення. Водночас їх впровадження вимагає ретельного проектування, налаштування та оптимізації з урахуванням конкретних вимог і обмежень системи[4].

1.5. Висновки до першого розділу

Системи фільтрації спаму відіграють критично важливу роль у забезпеченні ефективної роботи веб-ресурсів, підвищенні продуктивності користувачів та співробітників, а також дотриманні стандартів інформаційної безпеки. Вони захищають від потенційних загроз, таких як поширення шкідливого програмного

забезпечення, фішингових атак та крадіжки даних через спам-повідомлення чи спам-коментарі. Дослідження виявило, що існує багато різних методів фільтрації спаму, кожен з власними перевагами та недоліками.

Класифікація цих методів включає фільтрацію на основі контенту, фільтрацію на основі списків та гібридні методи. Фільтрація на основі контенту застосовує аналіз самого змісту повідомлення, використовуючи різні підходи, такі як аналіз слів, евристики, методи машинного навчання та інші. З іншого боку, фільтрація на основі списків покладається на заздалегідь визначені переліки відомих джерел спаму, таких як IP-адреси, домени чи адреси електронної пошти. Гібридні методи поєднують переваги обох цих підходів для досягнення більшої ефективності.

Для розпізнавання спаму застосовуються різноманітні алгоритми та підходи, включно з машинним навчанням, статистичними методами та фільтрацією на основі правил. Алгоритми машинного навчання, такі як наївний байєсовський класифікатор, дерева рішень, нейронні мережі та метод опорних векторів, демонструють високу точність виявлення спаму завдяки здатності виявляти складні закономірності в даних. Статистичні методи, такі як стиснення даних, покладаються на аналіз статистичних властивостей тексту для визначення його легітимності. Фільтрація на основі правил дозволяє адміністраторам вручну визначати критерії для класифікації спаму, забезпечуючи гнучкість та можливість цільового блокування певних видів спаму.

Незважаючи на різноманітність методів, жоден з них не є ідеальним, і часто вимагається комбінування різних підходів для досягнення максимальної ефективності. Також важливо враховувати специфічні вимоги кожного веб-ресурсу та адаптувати систему фільтрації спаму відповідно до цих потреб.

РОЗДІЛ 2
АНАЛІЗ МЕТОДІВ РОЗПІЗНАВАННЯ СПАМУ

2.1. Алгоритм наївного Байєса

Алгоритм наївного Байєса (або наївний байєсовський класифікатор) - це алгоритм класифікації, заснований на теоремі Байєса з допущенням про незалежність ознак. Це означає, що цей алгоритм передбачає, що наявність певної ознаки в класі не пов'язана з наявністю будь-якої іншої ознаки.

Наприклад, повідомлення може вважатися спамом, якщо в ньому містяться певні слова або фрази, характерні для спамових листів, незалежно від того, чи пов'язані ці слова один з одним. Хоча ці ознаки можуть бути залежними одна від одної або від інших факторів, алгоритм припускає, що кожна з них вносить незалежний внесок у ймовірність того, що повідомлення є спамом.

Через це спрощення алгоритм називається «наївним». Моделі на основі алгоритму наївного Байєса є простими у реалізації та надзвичайно корисними при роботі з великими наборами даних. Незважаючи на свою простоту, НБА може перевершувати навіть деякі складні алгоритми класифікації у задачах фільтрації спаму[13].

Завдяки цьому класифікатор наївного Байєса є популярним статистичним методом для фільтрації електронної пошти. Він використовує функції місць слів для ідентифікації спамових листів, застосовуючи підхід, який зазвичай використовується при класифікації тексту. Наївні класифікатори Байєса працюють шляхом співвіднесення маркерів (зазвичай слів, а іноді й інших характеристик) зі спамом та не спамом, а потім використовують теорему Байєса для обчислення ймовірності того, що електронне повідомлення є спамом чи ні.

Окремі слова мають особливу ймовірність виникнення у спам-пошті та в нормальних електронних листах. Фільтр не знає цих ймовірностей заздалегідь, і його потрібно навчити для їх визначення. Для цього користувач повинен вручну вказати, чи новий електронний лист є спамом чи ні.

Навчання алгоритму проходить так:

1. Для всіх слів у кожному навчальному повідомленні фільтр відрегулює ймовірність того, що кожне слово з'явиться у спамі або нормальному електронному листі;
2. Байєсовські спам-фільтри визначатимуть, що певні слова мають високу ймовірність появи у спамі, тоді як інші слова (наприклад, імена друзів або членів сім'ї) матимуть низьку ймовірність появи в повідомленнях від шахраїв.

Таким чином, на основі цих ймовірностей, алгоритм наївного Байєса ефективно класифікує електронні листи як спам або не спам, забезпечуючи надійну фільтрацію небажаної пошти.

Після навчання, ймовірності слів (також відомі як функції правдоподібності) використовуються для обчислення ймовірності того, що електронний лист з певним набором слів належить до однієї з категорій. Кожне слово у листі вносить свій внесок у ймовірність того, що лист є спамом. Цей внесок називається «апостеріорною ймовірністю» і обчислюється за допомогою теореми Байєса. Потім ймовірність того, що лист є спамом, обчислюється на основі всіх слів у листі, і якщо загальна ймовірність перевищує певний поріг (наприклад, 95%), фільтр позначає лист як спам.

Як і в будь-якій іншій техніці фільтрації спаму, електронні листи, позначені як спам, можуть автоматично переміщуватися в папку "Небажана пошта" або навіть видалятися. Деяке програмне забезпечення реалізує механізми карантину, які визначають період, протягом якого користувач може переглянути рішення програмного забезпечення.

Початкове навчання зазвичай можна вдосконалити, коли виявляються неправильні рішення програмного забезпечення (хибні спрацьовування або пропуски). Це дозволяє програмному забезпеченню динамічно адаптуватися до постійно мінливого характеру спаму.

Деякі фільтри спаму поєднують результати як Байєсової фільтрації спаму, так і інших евристик (попередньо визначених правил щодо вмісту, аналізу заголовка повідомлення тощо), що забезпечує ще вищу точність фільтрації, іноді за рахунок зменшення адаптивності[14].

Для того, щоб зрозуміти як цей алгоритм працює, спочатку потрібно розібратись що таке теорема Байєса.

Теорема Байєса — це фундаментальна теорема ймовірності, яка описує ймовірність події з урахуванням наявної інформації про інші події. Вона названа на честь англійського математика Томаса Байєса.

По суті своїй, вона є рівнянням, що описує взаємозв'язок умовних ймовірностей статистичних величин. У байєсовській класифікації ми зацікавлені в знаходженні ймовірності мітки за деякими спостережуваними ознаками, а також у наявності певних спостережуваних ознак, що можна записати як P(**A|B**). Теорема Байєса показує, як виразити це у термінах величин, які ми можемо обчислити більш безпосередньо:

$$\mathrm{P}(\boldsymbol{A}|\boldsymbol{B}) = \frac{\mathrm{P}(\boldsymbol{B}|\boldsymbol{A})\mathrm{P}(\boldsymbol{A})}{\mathrm{P}(\boldsymbol{B})} \tag{2.1}$$

де:

- P(A|B) - Умовна ймовірність настання події A за умови, що подія B вже відбулась;
- P(B|A) - Умовна ймовірність настання події B за умови, що подія A вже відбулась;
- P(A) - Апріорна (початкова) ймовірність настання події A без жодних умов;
- P(B) - Апріорна (початкова) ймовірність настання події B без жодних умов.

Якщо ми намагаємося вибрати між двома мітками — назвемо їх A_1 та A_2, то один зі способів прийняття цього рішення полягає в обчисленні відношення апостеріорних ймовірностей для кожної мітки:

$$\frac{P(\boldsymbol{A_1}|\boldsymbol{B})}{P(\boldsymbol{A_2}|\boldsymbol{B})} = \frac{\mathrm{P}(\boldsymbol{B}|\boldsymbol{A_1})\mathrm{P}(\boldsymbol{A_1})}{\mathrm{P}(\boldsymbol{B}|\boldsymbol{A_2})\mathrm{P}(\boldsymbol{A_2})} \quad (2.2)$$

Все, що нам тепер потрібно, це деяка модель, за допомогою якої ми можемо обчислити $\mathrm{P}(\boldsymbol{B}|\boldsymbol{A_i})$ для кожної мітки. Така модель називається генеративною моделлю, оскільки вона визначає гіпотетичний випадковий процес, який генерує дані. Визначення цієї генеративної моделі для кожної мітки є основним завданням навчання такого Байєсовського класифікатора. Загальна версія такого навчального кроку є дуже складним завданням, але його можна спростити за допомогою деяких спрощувальних припущень щодо форми цієї моделі[15].

Тут і з'являється наївність у «наївному Байєсі»: якщо ми робимо дуже наївні припущення щодо генеративної моделі для кожної мітки, ми можемо знайти грубе наближення генеративної моделі для кожного класу, а потім перейти до Байєсовської класифікації. Різні типи наївних Байєсовських класифікаторів базуються на різних наївних припущеннях щодо даних[13].

У контексті виявлення спаму ця теорема застосовується декілька разів:

1. Для розрахунку ймовірності того, що повідомлення є спамом, за наявності певної ознаки, наприклад, конкретного слова чи фрази;
2. Для визначення загальної ймовірності того, що повідомлення є спамом, ураховуючи всі його ознаки або певну їх підмножину;
3. Для опрацювання рідкісних ознак, які зустрічаються нечасто в навчальних даних.

Припустимо, підозріле повідомлення містить слово "акція". Для більшості людей це слово може асоціюватися зі спамом, що пропонує виграші, які потребують зазначення приватної інформації (дані банківської карточки чи для входу в акаунт) або пропонують нелегітимні призи. Однак програма для виявлення спаму не має такої інтуїції і має обчислити ймовірність того, що повідомлення зі словом "нагорода" є спамом, використовуючи теорему Байєса.

Формула, яка використовується програмою для визначення цього базується на формулі, яка була описана вище, має такий вигляд:

$$P(\boldsymbol{S}|\boldsymbol{W}) = \frac{P(\boldsymbol{W}|\boldsymbol{S}) * P(\boldsymbol{S})}{P(\boldsymbol{W}|\boldsymbol{S}) * P(\boldsymbol{S}) + P(\boldsymbol{W}|\boldsymbol{H}) * P(\boldsymbol{H})} \quad (2.3)$$

де:

- **P(S|W)** - Ймовірність того, що повідомлення є спамом, за умови, що воно містить певне слово;
- **P(S)** - Загальна апріорна ймовірність того, що повідомлення є спамом;
- **P(W|S)** - Ймовірність появи цього слова у спам-повідомленнях;
- **P(H)** - Апріорна ймовірність того, що повідомлення не є спамом (тобто є "хемом" або легітимним повідомленням);
- **P(W|H)** - Ймовірність появи цього слова у легітимних (не спам) повідомленнях.

Поточні дослідження говорять, що серед усього щоденного трафіку листів, 49% повідомлень можна класифікувати як спам-лист[16].

Фільтри, які використовують це припущення, вважаються "неупередженими", що означає відсутність будь-яких упереджень щодо вхідних повідомлень. Це припущення дозволяє спростити загальну формулу:

$$P(\boldsymbol{S}|\boldsymbol{W}) = \frac{P(\boldsymbol{W}|\boldsymbol{S})}{P(\boldsymbol{W}|\boldsymbol{S}) + P(\boldsymbol{W}|\boldsymbol{H})} \quad (2.4)$$

По суті, дане рівняння намагається відповісти на питання : «Який відсоток появ певного слова зустрічається у спам-листах?».

Ця величина називається "спамністю" слова і може бути обчислена. Число P(**W|S**), що використовується у формулі, наближається до частоти повідомлень, що містять певне слово у повідомленнях, ідентифікованих як спам під час навчальної фази. Аналогічно, P(W|H) наближається до частоти появи цього слова у повідомленнях, визначених як легітимні (не спам) протягом навчання. Для коректності цих наближень набір навчальних повідомлень має бути достатньо великим і репрезентативним. Також бажано, щоб навчальні набори спам- і легітимних повідомлень були приблизно однакового розміру.

Звісно, визначати, чи є повідомлення спамом лише на основі присутності одного слова є ризикованим, тому байєсовські системи виявлення спаму намагаються враховувати кілька слів та комбінувати їхню "спамність" для оцінки загальної ймовірності того, що повідомлення є спамом[14].

Доволі простим для розуміння наївним байєсівським класифікатором є гаусівський наївний байєсовський класифікатор. У цьому класифікаторі припускається, що дані з кожної мітки беруться з простого гаусівського розподілу.

Один з надзвичайно швидких способів створити просту модель - припустити, що дані описуються гаусівським розподілом без коваріації (тобто дві випадкові величини не змінюватимуться, якщо їх порівнювати одна з одною) між вимірами. Цю модель можна побудувати, просто знайшовши середнє значення і стандартне відхилення точок у межах кожної мітки - це все, що потрібно для визначення такого розподілу (рис. 2.1).

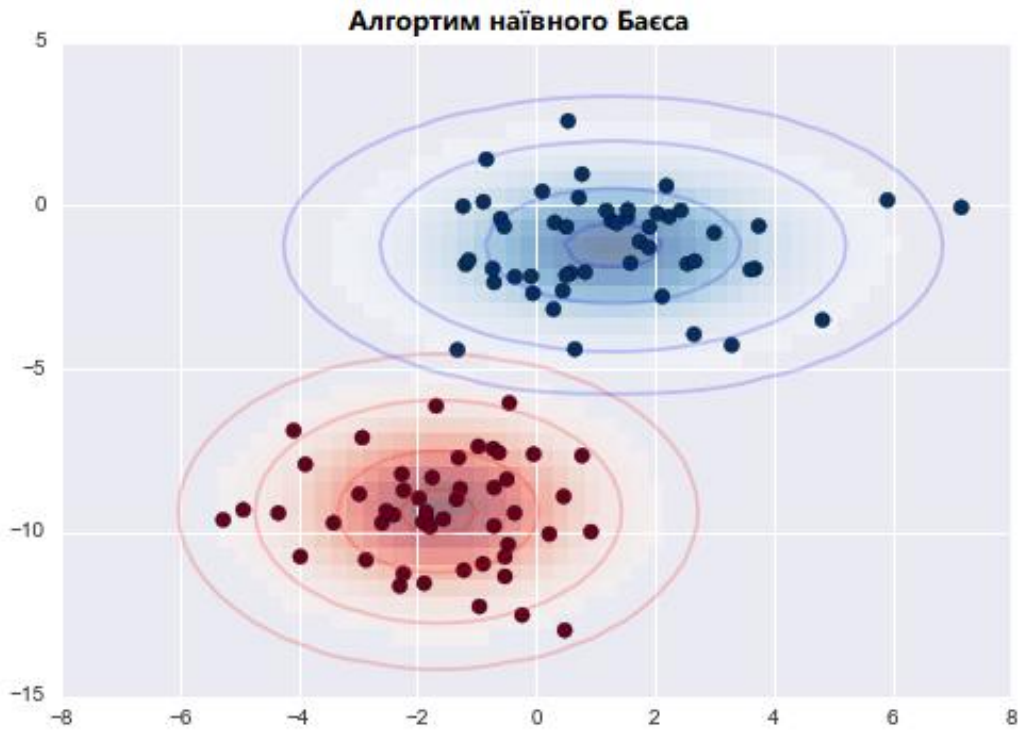


Рис. 2.1. Результат наївного гауссівського припущення

Еліпси тут представляють генеративну модель Гауса для кожної мітки, з більшою ймовірністю до центру еліпсів. Маючи таку генеративну модель для

кожного класу, ми маємо простий рецепт для обчислення ймовірності для будь-якої точки даних, і таким чином можемо швидко обчислити апостеріорне відношення і визначити, яка мітка є найбільш вірогідною для даної точки.

Звичайно, остаточна класифікація буде настільки точною, наскільки точними є припущення моделі, що призвели до неї, і саме тому гауссівський наївний Байєс часто дає не дуже гарні результати. Проте в багатьох випадках - особливо коли кількість ознак стає великою - це припущення не є достатньо шкідливим, щоб завадити гауссівському наївному Байєсу бути корисним методом.

Проте описане щойно припущення Гауса - далеко не єдине просте припущення, яке можна використати для визначення генеративного розподілу для кожної мітки. Іншим корисним прикладом є мультиноміальний наївний Байєс, де передбачається, що ознаки генеруються з простого багаточленного розподілу. Мультиноміальний розподіл описує ймовірність спостереження підрахунків серед низки категорій, і тому мультиноміальний наївний Байєс є найбільш придатним для ознак, які представляють підрахунки або частоту підрахунків.

Ідея точно така ж, як і раніше, за винятком того, що замість того, щоб моделювати розподіл даних за допомогою найкращого гауссівського, ми моделюємо розподіл даних за допомогою найкращого мультиноміального розподілу[13].

Серед переваг алгоритму наївного Байєса можна виділити наступне:

- Він простий та легкий у впровадженні;
- Не потребує великої кількості навчальних даних;
- Може опрацьовувати як неперервні, так і дискретні дані;
- Добре масштабується з кількістю предикторів (ознак) та точок даних;
- Швидкий, може використовуватися для прогнозів у реальному часі;
- Не чутливий до незначущих ознак;
- Забезпечує хороші результати навіть за наявності припущення про незалежність ознак;
- Легко інтерпретувати результати та побачити вплив кожної ознаки;
- Може бути корисним як базова модель для порівняння з іншими, складнішими моделями;

- При належному налаштуванні здатен забезпечити точність, порівнянну зі складнішими методами.

Проте цей алгоритм також має і недоліки. Найкритичнішим з них є байєсовське отруєння — техніка, яку використовують спамери для зниження ефективності фільтрів. Спамери вставляють у повідомлення великі фрагменти легітимного тексту або випадкові слова, щоб знизити оцінку спаму.

Також метод ґрунтується на припущенні про незалежність ознак, що не завжди відповідає дійсності. Це припущення може призвести до втрати точності в разі, якщо є сильна кореляція між ознаками.

Ефективність методу може знижуватися, якщо спамери застосовують трансформації слів, заміняючи їх на схожі варіанти. Це ускладнює процес навчання фільтра на таких видозмінених словах.

Фільтр доволі обмежений у опрацюванні графічного контенту, і якщо текст повідомлення замінено на зображення, звичайний байєсовський фільтр не зможе проаналізувати його зміст.

Самі результати може бути складно інтерпретувати та пояснити, особливо для складних наборів даних з великою кількістю ознак[14].

2.2. Метод опорних векторів

Метод Опорних Векторів (далі SVM) - це потужний алгоритм машинного навчання, який використовується для лінійної або нелінійної класифікації, регресії та навіть виявлення викидів, хоча найкраще він підходить саме для класифікації. SVM можна застосовувати для різноманітних задач, таких як класифікація тексту, класифікація зображень, виявлення спаму, розпізнавання рукописного тексту, аналіз генної експресії, виявлення облич та виявлення аномалій.

SVM є гнучкими та ефективними алгоритмом, оскільки здатен працювати з багатовимірними даними та нелінійними залежностями. Цей алгоритм дуже ефективний, коли ми намагаємося знайти максимальну розділяючу гіперплощину між різними класами у цільовій ознаці та має властивість ігнорувати викиди і знаходити

найкращу гіперплощину, яка максимізує відстань. Хоча ми кажемо і про задачі регресії, він найкраще підходить саме для класифікації[17].

SVM - це алгоритм навчання з учителем. Щоб зрозуміти в чому різниця, розглянем навчання з учителем та без.

У навчанні з учителем алгоритму надаються розмічені навчальні дані, де для кожного прикладу відома правильна відповідь або цільова змінна. Завданням алгоритму є знайти залежність між вхідними даними та цільовою змінною на основі цих прикладів, щоб потім можна було робити точні передбачення для нових, ще не розмічених даних. Таким чином, алгоритм навчається на прикладах, наданих "учителем".

З іншого боку, у навчанні без учителя алгоритму надаються лише вхідні дані без будь-яких міток або цільових змінних. Завданням у цьому випадку є виявлення прихованих структур або закономірностей безпосередньо у самих даних. Це може включати кластеризацію даних на групи схожих прикладів, виявлення аномалій або просто знаходження компактних уявлень для ефективного опису даних[18].

Основна мета алгоритму SVM - знайти оптимальну гіперплощину в N-вимірному просторі, яка зможе розділити точки даних різних класів у просторі ознак. Гіперплощина намагається максимізувати відстань між найближчими точками різних класів. Вимірність гіперплощини залежить від кількості ознак. Якщо кількість вхідних ознак дорівнює двом, то гіперплощина - це лінія. Якщо ознак три, то гіперплощина стає двовимірною площиною. Коли кількість ознак перевищує три, уявити гіперплощину стає складно.

Розглянемо наступний приклад. Припустимо, у нас є набір даних, і ми хочемо класифікувати й розділити червоні квадрати від синіх кіл (припустімо, позитивне й негативне). Основною метою в цьому завданні буде знайти "ідеальну" лінію, яка розділить ці два класи (Рис. 2.2).

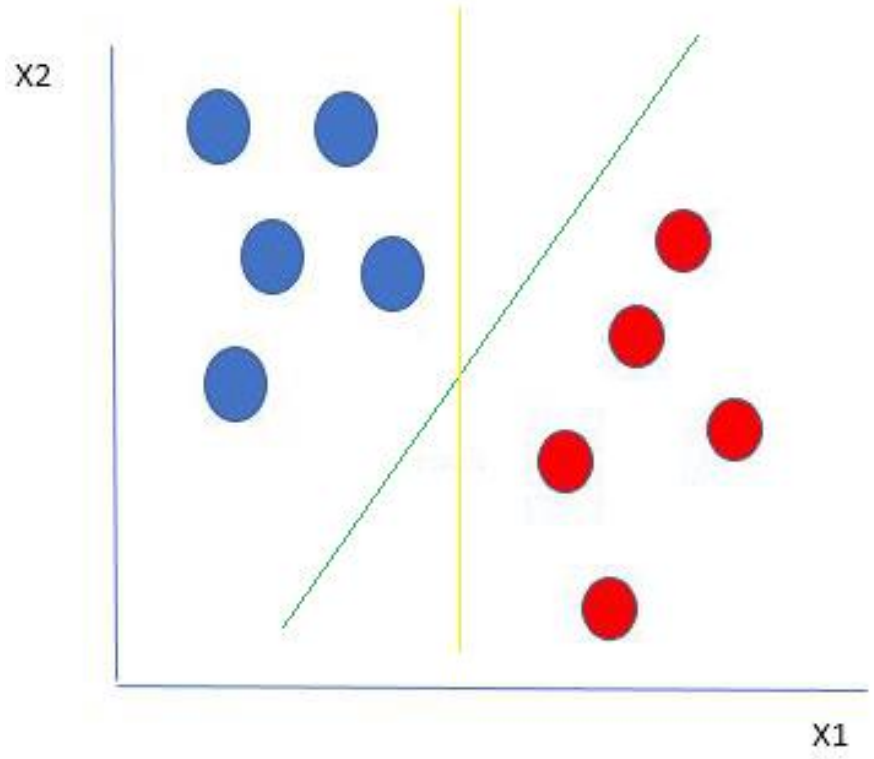


Рис. 2.2. Набір даних з двома варіантами лінійного розподілу класів

Одним із розумних виборів найкращої гіперплощини є та, що забезпечує найбільше відокремлення або відстань між двома класами. Тому ми обираємо гіперплощину, відстань від якої до найближчої точки з обох боків є максимальною. Якщо така гіперплощина існує, вона називається гіперплощиною з максимальним відокремленням або жорсткою межею.

У випадку із жовтою лінією - вона розташована занадто близько до синього класу. Незважаючи на те, що було правильно класифіковано всі об'єкти поточного набору даних, така лінія не буде генералізованою - не буде так само добре поводитися з незнайомим набором даних. Завдання знаходження генералізованої роздільної лінії двох класів є одним з основних завдань у машинному навчанні.

Алгоритм SVM влаштований таким чином, що він шукає точки на графіку, які розташовані найближче до розділювальної лінії. Ці точки називаються опорними векторами. Потім алгоритм обчислює відстань між опорними векторами та розділювальною площиною. Ця відстань називається зазором. Основна мета

алгоритму — максимізувати відстань зазору. Найкращою гіперплощиною вважається та, для якої цей зазор є найбільшим (Рис. 2.3).

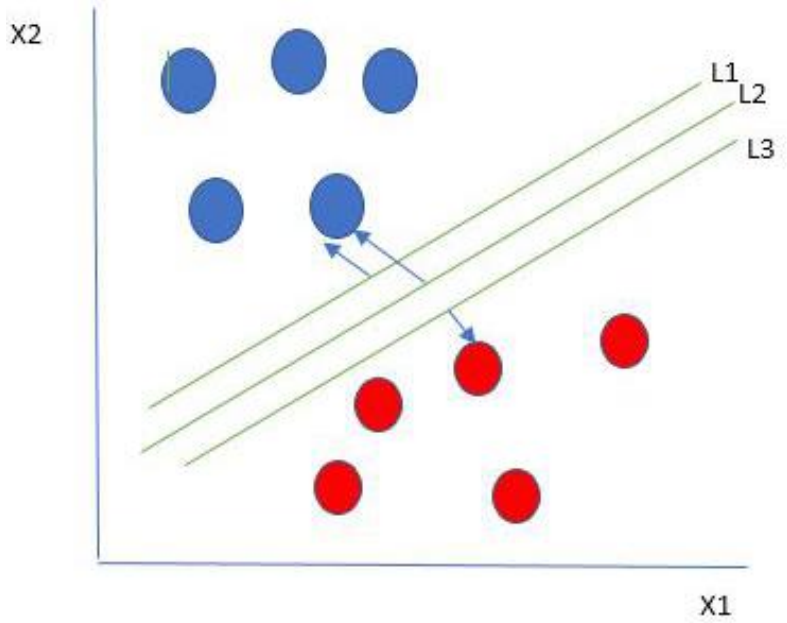


Рис. 2.3. Кілька гіперплощин розділяють дані з двох класів

Проте ось ми маємо одну синю кульку на межі червоних кульок. Як SVM класифікує дані? За допомогою «викидів» (Рис. 2.4).

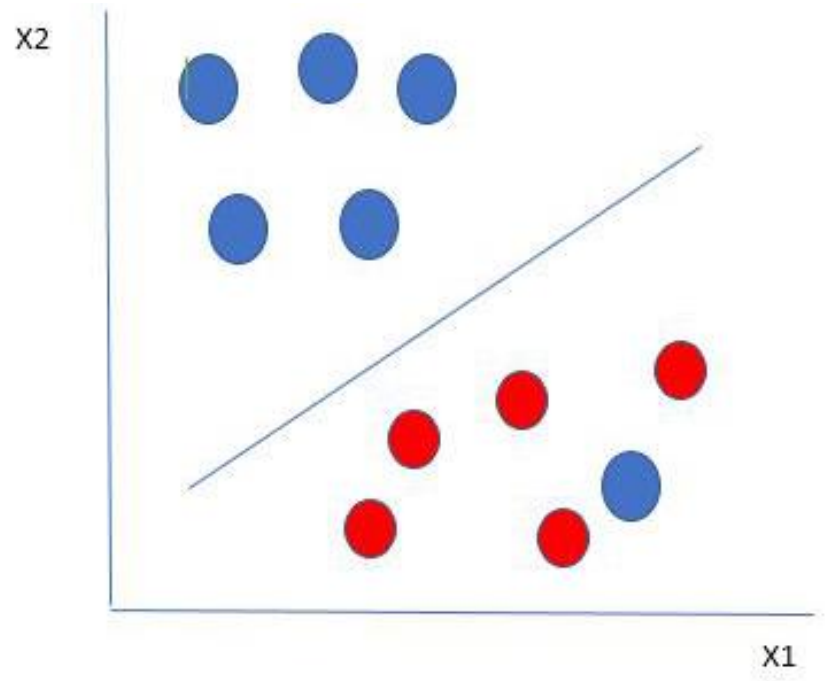


Рис. 2.4. Набір даних з викидом, розділений гіперплощиною

Синя кулька на межі червоних є викидом серед синіх кульок. Алгоритм SVM має властивість ігнорувати викиди і знаходити найкращу гіперплощину, яка максимізує зазор.

Отже, у такому типі даних SVM знаходить максимальний зазор, як і з попередніми наборами даних, та додає штраф щоразу, коли точка перетинає межу. Такі зазори називаються м'якими зазорами. Коли у наборі даних присутній м'який зазор, SVM намагається мінімізувати $(\frac{1}{\text{зазор}} + \Lambda(\sum \text{штраф})$. Втрата зазорів (hinge loss) є загальновживаним штрафом. Якщо немає порушень, то немає і втрати зазорів. Якщо є порушення, то втрата зазорів пропорційна відстані порушення[17].

Ця задача не здається такою складною, якщо ми працюємо з лінійним класом даних, проте якщо дані не є лінійно відокремлюваними потрібно застосовувати інші методи.

Розглянемо задачу бінарної класифікації з двома класами, позначеними як +1 та -1. Маємо навчальний набір даних, що складається з векторів ознак X і відповідних їм міток класів Y.

Рівняння лінійної гіперплощини матиме вигляд:

$$w^T x + b = 0 \tag{2.5}$$

Вектор W представляє нормальний вектор до гіперплощини, тобто напрямок, перпендикулярний до гіперплощини. Параметр b у рівнянні представляє зсув або відстань гіперплощини від початку координат уздовж нормального вектора w.

Відстань між точкою даних x_i та розділювальною межею можна обчислити як:

$$d_i \frac{w^T x + b_i}{||w||} \tag{2.6}$$

Де ||w|| представляє евклідову норму вектора ваг w. Евклідова норма нормального вектора W.

Для лінійного класифікатора SVM:

$$\hat{y} = \begin{cases} 1 : w^T x + b \geq 0 \\ 0 : w^T x + b \leq 0 \end{cases} \quad (2.7)$$

Для поліноміального класифікатора SVM:

$$\hat{y} = \begin{cases} 1 : (\gamma w^T x + b)^N \geq 0 \\ 0 : (\gamma w^T x + b)^N \leq 0 \end{cases} \quad (2.8)$$

Залежно від природи границі рішення, машини опорних векторів (SVM) можна розділити на дві основні частини:

- Лінійні SVM використовують лінійну границю рішення для розділення точок даних різних класів. Якщо дані можуть бути точно лінійно розділені, лінійні SVM дуже добре підходять. Це означає, що одна пряма лінія (у двовимірному або ж 2D просторі) або гіперплощина (у вищих вимірах) може повністю розділити точки даних на відповідні класи. Гіперплощина, яка максимізує відстань між класами, є межею рішення.
- Нелінійне SVM можна використовувати для класифікації даних, коли вони не можуть бути розділені на два класи прямою лінією (у випадку 2D). Використовуючи функції ядра, нелінійні SVM можуть обробляти дані, що нелінійно розділяються. Вихідні вхідні дані перетворюються за допомогою цих функцій ядра у вимірний простір ознак, де точки даних можуть бути лінійно розділені. Лінійна SVM використовується для знаходження нелінійної межі рішення в цьому модифікованому просторі.

Ядро SVM - це функція, яка бере низьковимірний вхідний простір і перетворює його у вимірний простір, тобто перетворює невідокремлювані задачі на відокремлювані. Здебільшого це корисно у нелінійних задачах розділення. Простіше кажучи, ядро виконує деякі надзвичайно складні перетворення даних, а потім знаходить процес розділення даних на основі визначених міток або виходів[17].

Перевагами SVM є:

- Метод ефективний у випадках з високою розмірністю;
- Економний у використанні пам'яті, оскільки використовує підмножину навчальних точок у функції прийняття рішень, яка називається опорними векторами;
- Для функцій прийняття рішень можна вказувати різні функції ядра та можливе використання користувацьких ядер.

Проте він має і декілька недоліків, основним з яких є обчислювальна складність. Навчання SVM може бути обчислювально затратним, особливо при роботі з великими наборами даних. Також може виникнути складність у виборі моделі: іноді важко вибрати правильну функцію ядра та оптимальні параметри для моделі SVM.

Обмежена ймовірнісна інтерпретація теж може стати проблемою. На відміну від інших алгоритмів, SVM не надають ймовірнісної інтерпретації прогнозів, що може бути недоліком для деяких застосувань[19].

2.3. Метод нейронних мереж

Нейронна мережа (ANN) - це машинна модель навчання, яка приймає рішення подібно до людського мозку, використовуючи процеси, що імітують спосіб роботи біологічних нейронів для ідентифікації явищ, зважування варіантів і формування висновків.

Ключовими концепціяии нейронних мереж є:

- Нейрони - основні обчислювальні одиниці, які отримують вхідні дані, обробляють їх за допомогою ваг і функції активації та передають результат іншим нейронам;
- Зв'язки (синапси) - з'єднання між нейронами, через які передаються сигнали. Кожне з'єднання має певну вагу, яка визначає силу зв'язку;
- Ваги - числові параметри, які регулюють силу зв'язків між нейронами і змінюються в процесі навчання мережі;

- Функція активації - нелінійна функція, яка визначає вихід нейрона на основі зваженої суми його входів;
- Архітектура - спосіб організації та з'єднання нейронів у мережі, наприклад прямі зв'язки, рекурентні або згорткові мережі.

Кожна нейронна мережа складається з шарів вузлів або штучних нейронів: вхідного шару, одного або декількох прихованих шарів та вихідного шару. Кожен вузол з'єднаний з іншими і має свою власну вагу та поріг (Рис. 2.5). Якщо вихід будь-якого окремого вузла перевищує задане порогове значення, цей вузол активується, передаючи дані до наступного шару мережі. Інакше дані не передаються до наступного шару.

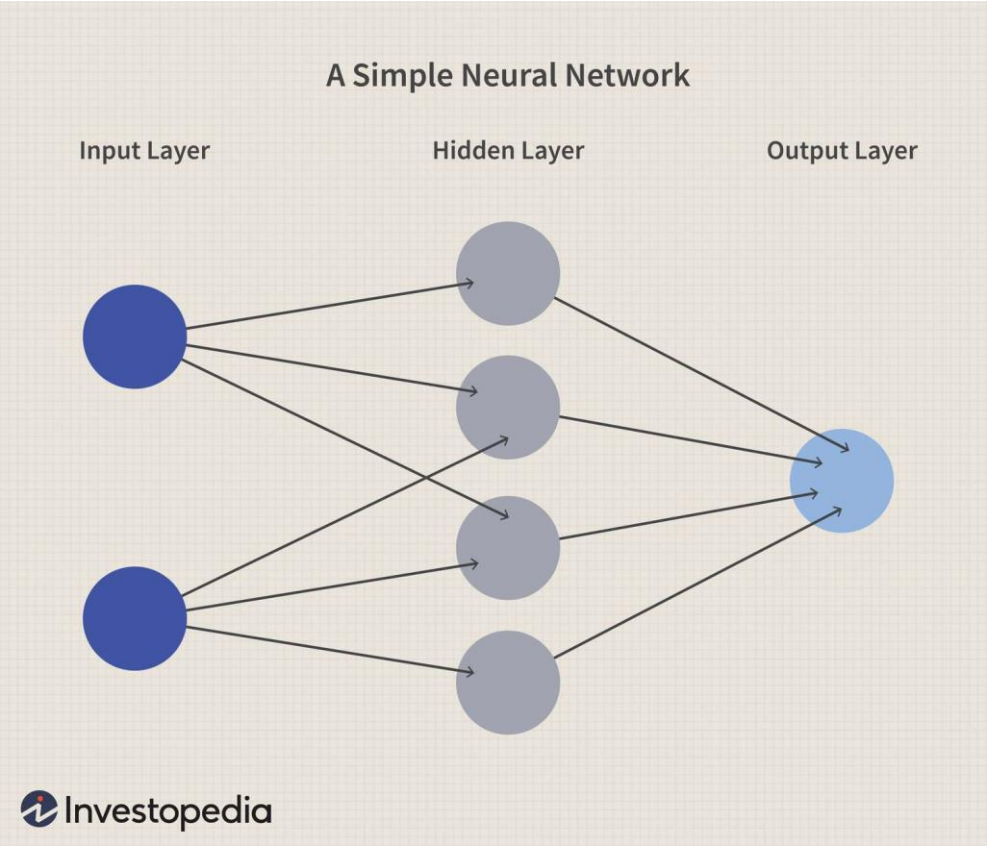


Рис. 2.5. Проста нейронна мережа

Нейронні мережі покладаються на навчальні дані для покращення точності з часом. Після налаштування на точність, вони стають потужним інструментом у

комп'ютерних науках та штучному інтелекті, дозволяючи класифікувати та кластеризувати дані з високою швидкістю.

Нейронні мережі іноді називають штучними нейронними мережами або симульованими нейронними мережами. Вони є підмножиною машинного навчання і лежать в основі моделей глибокого навчання[20].

Щоб зрозуміти, як працюють нейронні мережі, уявімо кожен окремий вузол як власну лінійну регресійну модель, що складається з вхідних даних, ваг, зміщення (або порогу) та виходу. Формула матиме такий вигляд:

$$\sum w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$
$$output = f(x) = 1\ if \sum w_1 x_1 + b \geq 0; if \sum w_1 x_1 + b < 0 \quad (2.9)$$

Після визначення вхідного шару присвоюються ваги. Ці ваги допомагають визначити важливість будь-якої заданої змінної, причому більші ваги роблять більш значний внесок у вихідний результат порівняно з іншими входами. Всі входи потім перемножуються на їхні відповідні ваги та сумуються. Після цього вихід проходить через функцію активації, яка визначає вихідне значення. Якщо це значення перевищує заданий поріг, воно "спрацьовує" (або активується) у вузлі, передаючи дані до наступного шару мережі (Рис. 2.6). Це призводить до того, що вихід одного вузла стає входом для наступного вузла. Цей процес передачі даних від одного шару до іншого визначає цю нейронну мережу як згорткову[21].

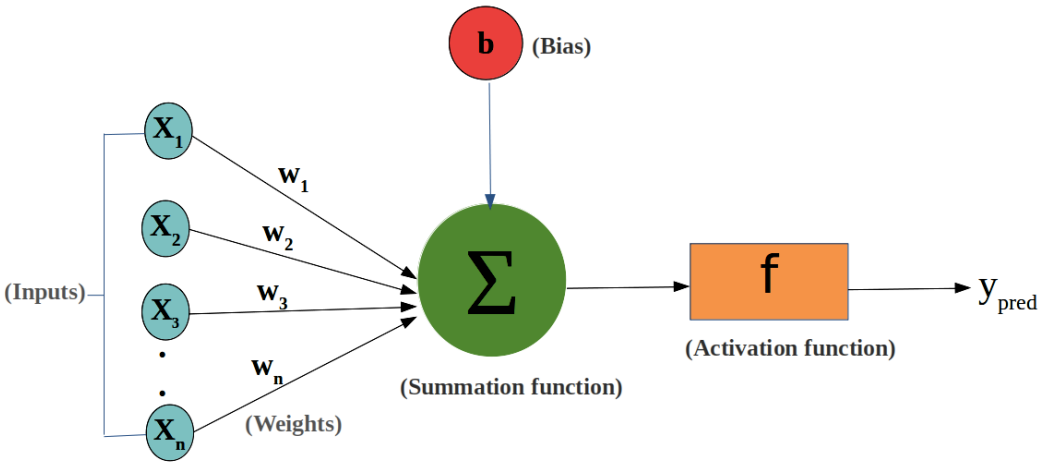


Рис. 2.6. Процеси, що проходять всередині кожного вузла

Нейронні мережі типово навчаються через мінімізацію емпіричного ризику. Цей метод базується на ідеї оптимізації параметрів мережі для мінімізації різниці, або емпіричного ризику, між передбаченим виходом та фактичними цільовими значеннями в заданому наборі даних. Методи на основі градієнтів, такі як зворотне поширення, зазвичай використовуються для оцінювання параметрів мережі. Під час фази навчання нейронні мережі вчаться на позначених навчальних даних шляхом ітеративного оновлення своїх параметрів для мінімізації визначеної функції втрат. Це дозволяє мережі узагальнювати невідомі дані.

Загалом, ризик R(h) не можна обчислити, оскільки розподіл P(x,y) невідомий алгоритму навчання (ця ситуація називається агностичним навчанням). Однак, маючи вибірку із незалежних однаково розподілених навчальних точок даних, ми можемо обчислити оцінку, яка називається емпіричним ризиком, обчисливши середнє значення функції втрат по навчальній вибірці; більш формально, обчисливши математичне сподівання з урахуванням емпіричного розподілу:

$$R_{emp}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(x_i), y_i \tag{2.10}$$

Іншими словами, під час тренування нейронна мережа намагається знайти оптимальні значення ваг та зміщень, які дозволять мінімізувати відхилення між передбаченими результатами та правильними позначеними значеннями в навчальному наборі даних. Використовуючи методи оптимізації на основі обчислення градієнтів, мережа ітеративно корегує свої параметри в напрямку, що зменшує функцію втрат. Цей процес повторюється, допоки мережа не досягне прийнятної продуктивності на тренувальному та перевірочному наборах. Завдяки цьому нейронна мережа набуває здатності узагальнювати та правильно обробляти невідомі, раніше не представлені дані[22].

Розглядаючи більш практичні випадки використання нейронних мереж, такі як розпізнавання зображень або класифікація тексту для розпізнавання спаму, застосовують навчання з учителем або позначені набори даних для навчання алгоритму. При навчанні моделі потрібно буде оцінити її точність за допомогою функції втрат (або вартості). Це також часто називають середньоквадратичною помилкою (MSE). У рівнянні нижче:

- i - індекс зразка;
- y-шапка - прогнозований результат;
- y - фактичне значення;
- m - кількість зразків.

Функція вартості:

$$MSE = \sum_{i}^{m} \frac{1}{2}(y - \hat{y})^2 \tag{2.11}$$

Зрештою, метою є мінімізація функції витрат, щоб забезпечити правильність підбору для будь-якого спостереження. Коли модель коригує свої ваги та зміщення, вона використовує функцію вартості та навчання з підкріпленням, щоб досягти точки збіжності, або локального мінімуму. Процес, в якому алгоритм налаштовує свої ваги, відбувається за допомогою градієнтного спуску, що дозволяє моделі визначити напрямок, в якому потрібно рухатися, щоб зменшити помилки або мінімізувати

функцію вартості (Рис. 2.7). З кожним навчальним прикладом параметри моделі підлаштовуються так, щоб поступово збігатися до мінімуму[23].

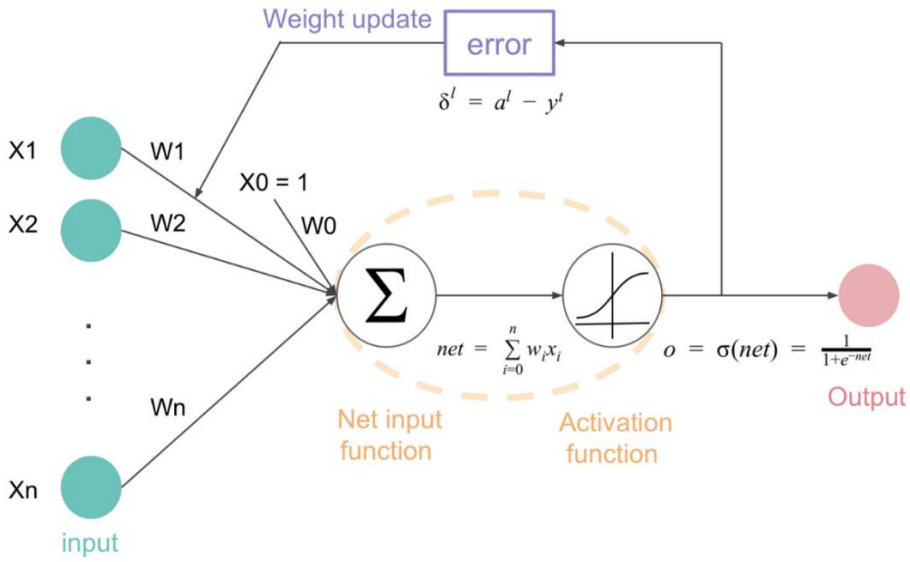


Рис. 2.7. Приклад використання середньоквадратичної помилки

Перевагами нейронних мереж для фільтрації спаму є:

- *Висока точність*. Нейронні мережі, особливо глибокі, можуть досягати високої точності у визначенні спаму, небажаного контенту або неприйнятних повідомлень завдяки здатності виявляти складні патерни в даних;
- *Адаптивність*. Нейронні мережі можуть навчатися на нових даних, що дозволяє їм адаптуватися до нових типів повідомлень і змін у мовленні або поведінці користувачів;
- *Автоматизація*. Після навчання модель може працювати автоматично, мінімізуючи необхідність людського втручання та забезпечуючи швидке реагування на нові повідомлення;
- *Можливість роботи з великим об'ємом даних*. Нейронні мережі можуть ефективно обробляти великі масиви даних, що є важливим для масштабованих систем фільтрації повідомлень;
- *Мультимодальність*. Сучасні нейронні мережі можуть працювати не лише з текстовими даними, але й з зображеннями, аудіо та відео, що дозволяє фільтрувати повідомлення з різними типами контенту.

Проте серед недоліків можна особливо виділити потребу в значних обчислювальних ресурсів, що може бути дорогим і ресурсномістким процесом. Також проблемою може бути те, що нейронні мережі часто функціонують як "чорні ящики", і важко зрозуміти, чому вони приймають ті чи інші рішення.

Загалом моделі можуть добре працювати на даних, схожих на ті, що використовувалися для навчання, але можуть бути менш ефективними на нових або значно відмінних даних що знижує здатність нейронних мереж виявляти нові види спаму, тому вона час від часу потребує оновлень[24].

2.4. Висновки до другого розділу

Проаналізовано три основні методи розпізнавання спаму: байєсовський метод, метод опорних векторів (SVM) та нейронні мережі. Кожен з цих підходів має свої переваги та недоліки.

Байєсовський метод є простим і легким у впровадженні, не потребує великої кількості навчальних даних та швидко працює. Проте він базується на припущенні про незалежність ознак, що може знизити точність за наявності кореляції між ними. Також цей метод вразливий до байєсовського отруєння та трансформацій слів спамерами.

Метод опорних векторів ефективний у випадках з високою розмірністю та економно використовує пам'ять. Однак він може бути обчислювально витратним, особливо для великих наборів даних, і вимагає ретельного вибору функції ядра та параметрів моделі.

Нейронні мережі здатні досягати високої точності у визначенні спаму, адаптуватися до нових даних та працювати з великими обсягами інформації різних типів. Проте вони потребують значних обчислювальних ресурсів, можуть бути "чорними ящиками" та менш ефективними на даних, відмінних від навчальних

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Опис використаних інструментів при розробці

Ключовим компонентом системи автоматичного розпізнавання та фільтрації спаму в коментарях веб-сайтів буде модель машинного навчання, реалізована за допомогою JavaScript та бібліотеки TensorFlow.js. Вибір JavaScript як мови програмування обумовлений його перевагами у веб-розробці, здатністю інтегруватися з HTML та CSS, а також можливістю запускати моделі машинного навчання безпосередньо на клієнтській стороні веб-додатку. Бібліотека TensorFlow.js надає зручний інструментарій для імпорту попередньо навчених моделей штучного інтелекту та їх використання для класифікації коментарів як спаму чи неспаму. Окрім того, JavaScript має велику і активну спільноту розробників, що забезпечує постійний розвиток необхідних для даної роботи алгоритмів та бібліотек машинного навчання.

3.1.1 HTML та CSS

Для того щоб система розпізнавання спаму в коментарях могла працювати, нам потрібно створити веб-сторінку, на якій можна буде залишати коментарі, які б модель могла б розпізнавати та надавати вердикт щодо вмісту коментаря. Одним з найпростіших та найпопулярніших рішень для виконання цього завдання є використання мови розмітки HTML та CSS.

HTML – це стандартна мова розмітки документів для створення веб-сторінок. Вона використовується для структурування та організації вмісту веб-сторінок за допомогою елементів, таких як заголовки, абзаци, списки, посилання, зображення та інші елементи. Основна функція HTML полягає у створенні структури веб-сторінки, визначенні її основних блоків і взаємозв'язків між ними.

CSS – це мова стилів, що використовується для опису зовнішнього вигляду і форматування веб-документів, написаних мовою HTML. CSS дозволяє визначати стилі для різних HTML-елементів, задаючи кольори, шрифти, розміри, відступи, вирівнювання та інші стилістичні характеристики. Використання CSS допомагає відокремити візуальне оформлення від структури документу, що робить код більш організованим і зручним для підтримки.

Для створення секції коментарів на веб-сторінці ми скористаємося засобами HTML для структурування елементів (Рис. 3.1).

```
<section id="comments" class="comments">
  <div id="comment" class="comment" contenteditable></div>
  <button id="post" type="button">Comment</button>

  <ul id="commentsList">
    <li>
      <span class="username">NotASpammer</span>
      <span class="timestamp">20.05.2024, 17:12:02</span>
      <p>Impressive work, deserved a like.</p>
    </li>
    <li>
      <span class="username">SomeUser</span>
      <span class="timestamp">20.05.2024, 18:01:56</span>
      <p>Nice blog. Pretty amazing videos!</p>
    </li>
</section>
```

Рис. 3.1. Секція коментарів

У цьому коді ми створюємо базову структуру коментарів. Секція коментарів позначена як section з атрибутами id="comments" та class="comments". Усередині

секції розміщено заголовок, форму для введення коментарів та контейнер для виведення списку коментарів (Рис. 3.2).

```
.comments button {
    float: right;
    margin: 5px 0;
}

.comments button, .comment {
    transition: opacity 500ms ease-in-out;
}

.comments ul {
    clear: both;
    margin-top: 60px;
}

.comments ul li {
    margin-top: 5px;
    padding: 10px;
    transition: background 500ms ease-in-out;
    background: #212121;
}
```

Рис. 3.2. Присвоєння стилів секції коментарів

У цьому CSS-коді секцію коментарів стилізовано так, щоб вона виглядала привабливо і зручно для користувачів. Визначено стилі для основного контейнера, заголовка, форми для введення коментарів та кнопки відправлення.

Ця секція є основою для подальшої інтеграції з JavaScript та бібліотекою TensorFlow.js, які будуть обробляти введені коментарі, передбачати наявність спаму та забезпечувати відповідні повідомлення користувачам.

3.1.2 JavaScript

В даній роботі переважно використана мова програмування JavaScript, оскільки вона є незамінною для розробки клієнтської частини веб-додатків. JavaScript забезпечує можливість динамічної взаємодії та маніпуляції з елементами веб-сторінки, що є ключовим для реалізації функціональності коментування.

Переваги використання JavaScript для реалізації проекту:

- JavaScript тісно пов'язаний з HTML та CSS, що дозволяє легко маніпулювати структурою та стилями веб-сторінки, спрощуючи процес відображення та обробки коментарів користувачів у веб-інтерфейсі;
- JavaScript надає можливість обробляти такі клієнтські взаємодії, як натискання клавіш, рух миші та інші дії користувача, що дозволяє створити зручний інтерфейс для введення та відправлення коментарів;
- За допомогою технології AJAX, JavaScript може відправляти асинхронні запити на сервер, не перезавантажуючи всю сторінку, забезпечуючи плавну роботу системи коментарів, коли користувач може відправляти та отримувати коментарі без перезавантаження сторінки;
- Існує багато потужних бібліотек та фреймворків JavaScript, таких як React, Angular чи Vue.js, що спрощують розробку складних веб-додатків, включаючи системи коментарів;
- JavaScript може взаємодіяти з серверними технологіями, такими як Node.js, що дозволяє створювати повноцінні веб-додатки з єдиною мовою програмування на клієнтській та серверній сторонах.

3.1.3 TensorFlow.js

Для того щоб реалізувати систему автоматичного розпізнавання та фільтрації спаму в коментарях, використано TensorFlow.js – потужну бібліотеку для машинного навчання, яка дозволяє виконувати операції машинного навчання безпосередньо в браузері за допомогою JavaScript.

TensorFlow.js – це відкрита бібліотека машинного навчання, яка працює скрізь, де може виконуватися JavaScript. Вона побудована на основі оригінальної бібліотеки TensorFlow, написаної мовою Python, і має на меті відтворити цей досвід розробки та набір API для екосистеми JavaScript. TensorFlow.js дозволяє

виконувати машинне навчання на різних платформах, включаючи використання в клієнтській частині у веб-браузері за допомогою звичайного JavaScript

Для проекту автоматичного розпізнавання спаму в коментарях ми використовуємо TensorFlow.js для завантаження та виконання попередньо навченої моделі машинного навчання безпосередньо у веб-браузері. Це забезпечує низку переваг, включаючи швидкість обробки, зниження витрат та збереження конфіденційності даних користувачів.

TensorFlow.js підтримує кілька бекендів у кожному з цих середовищ щоб забезпечити сумісність і високу швидкість роботи:

1. **WebGL** – використання графічного процесора (GPU) для виконання більших моделей (понад 3 МБ) з апаратним прискоренням;
2. **Web Assembly (WASM)** – виконання на CPU для покращення продуктивності на різних пристроях, включаючи старі мобільні телефони. Підходить для менших моделей (менше 3 МБ), які можуть працювати швидше на CPU за допомогою WASM[25].

   Переваги використання TensorFlow.js

   Клієнтська частина:

- *Конфіденційність*. Можна як навчати, так і класифікувати дані на клієнтському пристрої без необхідності передавати дані на сторонній сервер. Це важливо для дотримання місцевих законів, таких як GDPR, та забезпечення безпеки даних користувача;
- *Швидкість*. Відсутність необхідності відправляти дані на віддалений сервер прискорює процес класифікації. Також можна безпосередньо отримати доступ до сенсорів пристрою, таких як камера, мікрофон, GPS тощо;
- *Масштабованість*. Будь-хто в світі може відкрити веб-сторінку з вашим додатком у своєму браузері без необхідності встановлювати складні серверні налаштування.

Серверна частина:

- *Підтримка CUDA*. На серверній стороні для графічного прискорення необхідно встановити драйвери NVIDIA CUDA, що дозволяє повністю використовувати можливості графічного процесора, забезпечуючи швидший час навчання та прогнозування;
- *Розмір моделі*. Великі моделі, які не можуть працювати в браузері через обмеження пам'яті, можуть виконуватися на сервері з необхідними апаратними специфікаціями;
- *IoT*. Node.js підтримується на популярних одноплатних комп'ютерах, таких як Raspberry Pi, що дозволяє виконувати моделі TensorFlow.js на таких пристроях;
- *Швидкість*. Node.js написаний на JavaScript і підтримує Just-In-Time компіляцію, що може забезпечити приріст продуктивності під час виконання[25].

3.1.4 Node.js та Socket.io

Для реалізації серверної частини системи автоматичного розпізнавання та фільтрації спаму в коментарях веб-сайтів використано Node.js та Socket.io.

Node.js - це кросплатформене середовище виконання JavaScript з відкритим вихідним кодом, яке використовує асинхронну, керовану подіями архітектуру для створення масштабованих мережевих застосунків (Рис. 3.3). Node.js дозволяє розробляти швидкі веб-сервери на JavaScript, використовуючи подієво-орієнтоване програмування.

Рис. 3.3. Логотип Node.Js

Node.js працює на одному потоці з подієвою петлею (event loop), використовуючи неблокуючі виклики вводу-виводу. Це означає, що замість блокування процесу при очікуванні вводу-виводу, Node.js ефективно обробляє десятки тисяч одночасних з'єднань без необхідності контекстного перемикання потоків, забезпечуючи високу продуктивність та масштабованість. Проте, довготривалі обчислювальні операції можуть заморозити всю подієву петлю до їх завершення.

Node.js побудований на основі високопродуктивного рушія V8 від Google, що забезпечує швидке виконання JavaScript коду. Він надає вбудовану підтримку фундаментальних мережевих протоколів, таких як HTTP, TCP, UDP та DNS, що полегшує створення мережевих застосунків. Крім того, Node.js підтримує WebAssembly, що дозволяє використовувати коди на інших мовах, крім JavaScript.

Ключові переваги Node.js:

- Асинхронність та неблокуючий ввід-вивід для високої продуктивності та масштабованості;
- Використання JavaScript як для клієнтської, так і для серверної частини веб-додатків;
- Безліч інструментів для розробки, включаючи IDE, редактори коду та інші утиліти.

Node.js офіційно підтримується на Linux, macOS, Windows та деяких інших операційних системах, що забезпечує кросплатформеність застосунків. Він також

має підтримку на різних хмарних платформах, таких як Amazon Web Services, Google Cloud Platform, Microsoft Azure та інших.

Також Node.js має потужну екосистему модулів, доступних через npm (Node Package Manager), які розширюють його функціональність. Існує велика кількість відкритих бібліотек та фреймворків, таких як Express.js, Socket.IO, Sails.js, Next.js та Meteor, що прискорюють розробку веб-додатків[26].

Один з них, а саме Socket.IO буде використаним для розробки системи автоматичного розпізнавання та фільтрації спаму в коментарях.

Socket.io є потужною бібліотекою, яка дозволяє організувати двонаправлений обмін повідомленнями між клієнтом та сервером у режимі реального часу на основі подій. Вона складається з серверної частини, що працює на Node.js, та клієнтської бібліотеки, яку можна використовувати у браузері або на сервері (Рис. 3.4).

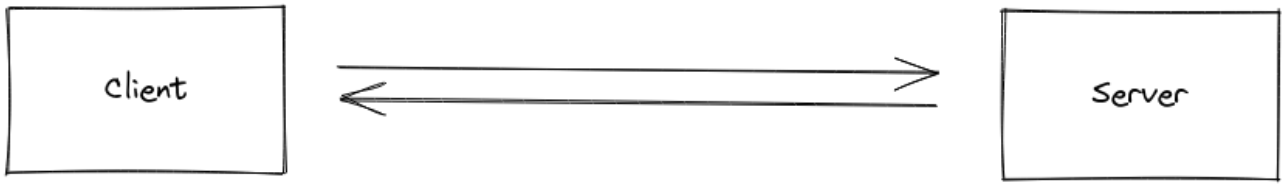


Рис. 3.4. Схема взаємодії за допомогою Socket.io

Однією з ключових особливостей Socket.io є надійність з'єднання. Воно встановлюється незалежно від наявності проксі-серверів, балансувальників навантаження, брандмауерів чи антивірусних програм. У разі роз'єднання Socket.io автоматично здійснює спробу повторного з'єднання з експоненціальним збільшенням затримки, щоб не перевантажувати сервер. Бібліотека також використовує механізм heartbeat для періодичної перевірки стану з'єднання та виявлення роз'єднань.

Важливою перевагою Socket.io є буферизація пакетів. Пакети автоматично буферизуються, коли клієнт від'єднаний, і надсилаються відразу після повторного з'єднання. Бібліотека також надає зручний спосіб відправляти події та отримувати

відповіді з можливістю встановлення часового проміжку очікування. На стороні сервера ви можете відправляти події всім під'єднаним клієнтам або підмножині клієнтів, навіть у разі масштабування на кілька вузлів.

Для передачі даних Socket.io автоматично вибирає найкращий доступний спосіб залежно від можливостей браузера та мережі, зокрема: HTTP Long Polling, WebSocket або WebTransport. Бібліотека дозволяє передавати як текстові, так і бінарні дані, що робить її ідеальним рішенням для розробки додатків, що працюють з мультимедіа. Крім того, Socket.io підтримує мультиплексування, що дозволяє встановлювати кілька з'єднань в одному TCP-сокеті[27].

3.2. Перетренування моделі розпізнавання спаму

Існують вже деякі готові натреновані моделі для виявлення спам-коментарів, але їх ефективність недостатня сама по собі. Ці моделі можуть не враховувати специфічні випадки або нові тенденції у спам-коментарях, що з'являються з часом. Тому, для збільшення ефективності, перетренуємо ці моделі під наш власний набір даних, щоб покращити їх точність та ефективність у нашому конкретному випадку.

Для даної кваліфікаційної роботи використано попередньо створену модель, яка була розроблена третьою стороною за допомогою Model Maker і використовує метод "average word embedding". Ця модель була створена для виявлення спам-коментарів, але в процесі її використання виявлено ряд граничних випадків, які вона не змогла правильно класифікувати.

Для перетренування моделі нам потрібно:

Зібрати нові дані:

- Знайти коментарі, які модель неправильно класифікувала (пропустила спам або помилково позначила як спам);
- Використовувати ручне позначення від користувачів та модераторів для ідентифікації таких коментарів;
- Для кращих результатів додати різні варіації нових речень.

Використати Python і TensorFlow Lite Model Maker:

Model Maker підтримує експорт у формат TensorFlow.js. Це дозволить використовувати перетреновану модель у веб-середовищі.

Перетренувати модель

Для цього ми використаємо середовище Google Colab, яке надає безкоштовне віртуальне середовище на 100 Гб для тренування нашої моделі.

Щоб почати перетренування моделі, нам необхідно (Рис. 3.5):

```
import numpy as np
import os
from tflite_model_maker import configs
from tflite_model_maker import ExportFormat
from tflite_model_maker import model_spec
from tflite_model_maker import text_classifier
from tflite_model_maker.text_classifier import DataLoader
import tensorflow as tf
assert tf.__version__.startswith('2')
tf.get_logger().setLevel('ERROR')

# Задавання параметрів
spec = model_spec.get('average_word_vec')
spec.num_words = 2000
spec.seq_len = 20
spec.wordvec_dim = 7

# Завантаження навчальних даних
data_file = tf.keras.utils.get_file(fname='comment-spam-extras.csv', origin='https://storage.go
data = DataLoader.from_csv(
      filename=data_file,
      text_column='commenttext',
      label_column='spam',
      model_spec=spec,
      delimiter=',',
      shuffle=True,
      is_training=True)

train_data, test_data = data.split(0.9)

# Компіляція та тренування
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model = text_classifier.create(train_data, model_spec=spec, epochs=50)

# Експорт моделі в TensorFlow.js формат
tf.saved_model.save(model, 'path/to/tfjs_model')
```

Рис. 3.5. Перетренування готової моделі розпізнавання спаму

- Підключити необхідні бібліотеки для роботи з TensorFlow, Model Maker для обробки даних;
- Налаштувати специфікації моделі "average word vec" з визначенням розміру словника, довжини послідовності та розмірності векторів слів.
- Надати тренувальний CSV-файли та завантажити його за допомогою DataLoader з визначенням колонок для тексту і міток, а також налаштувань для перемішування та розділення даних.
- Створити та зробити компіляцію моделі з використанням оптимізатора Adam, функції втрат binary_crossentropy та метрики accuracy, а потім почати її тренування на даних протягом 50 епох.

Експортувати модель

Необхідно зберегти перетреновану модель у форматі, сумісному з TensorFlow.js, для подальшого використання в веб-середовищі.

3.3. Створення веб-сайту та підключення його до серверу

Для перевірки системи автоматичного розпізнавання та фільтрації спаму в коментарях створено основну структуру веб-сторінки, яка має вигляд блогу про комп'ютерні клуби в Україні. Веб-сторінка включає різні секції, які дозволяють користувачам переглядати вміст, взаємодіяти з відео та залишати коментарі. Нижче наведено детальний опис того, що зроблено в цьому веб-документі.

- Встановлено базові теги HTML, такі як <html>, <head> та <body>, для визначення структури документа;
- У секції <head> встановлено заголовок сторінки, кодування символів, мета-теги для сумісності з браузерами та підключено зовнішній CSS файл style.css для стилізації сторінки;
- Щоб надати сайту вигляду блогу, додано заголовок <h1> із назвою "Blog about Ukrainian computer club" та два посилання для входу та реєстрації користувачів.

- Далі в <h2> та <p> тегах йде текст, який описує зміст блогу;
- Для демонстрації змісту блогу, вбудовано три YouTube відео у контейнер з класом iframe-container, використовуючи теги <iframe>;

Також створено секцію коментарів <section id="comments" class="comments">, яка містить:

- Поле для введення нового коментаря <div id="comment" class="comment" contenteditable> .
- Кнопку для публікації коментаря <button id="post" type="button">Comment</button>.
- Список існуючих коментарів <ul id="commentsList">, де кожен коментар представлений як <li> з відображенням імені користувача, часової позначки та тексту коментаря.

Щоб надати сайту вигляду блогу, використано простий та зрозумілий макет з шапкою, основним контентом та секцією коментарів. Заголовки та текстові блоки структуровані так, щоб відвідувачі могли швидко зрозуміти тему блогу та знайти цікаву інформацію.

У підсумку, веб-сайт виглядає наступним чином(Рис. 3.6).

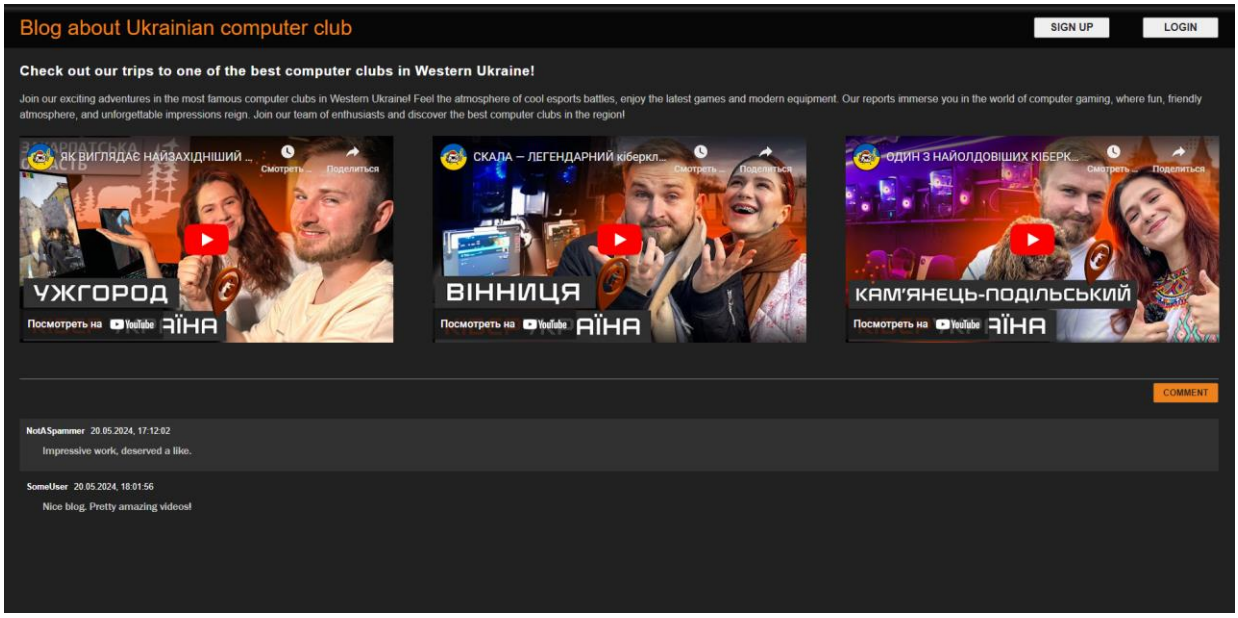


Рис. 3.6. Фінальний вигляд веб-сторінки

Загалом, цей HTML файл встановлює базову структуру та підключає необхідні ресурси для створення інтерактивного блогу з системою коментарів, де коментарі будуть фільтруватися за допомогою моделі машинного навчання TensorFlow.js та передаватися на сервер за допомогою Socket.IO для подальшої обробки та зберігання.

Для того, щоб системи автоматичного розпізнавання та фільтрації спаму могла взаємодіяти з коментарями, перш за все, потрібно, щоб ця веб-сторінка запускалася не локально, а на сервері. Це буде реалізовано за допомогою Node.js (Рис. 3.7).

```
const http = require('http');
const express = require("express");
const app = express();
const server = http.createServer(app);

// Запускаємо веб-сервер.
const listener = server.listen(process.env.PORT, () => {
  console.log("Ваш додаток слухає на порту " + listener.address().port);
});
```

Рис. 3.7. Підключення веб-сторінки до серверу

Спочатку підключаються два основні модулі Node.js: http та express. Модуль http надає функціональність для створення HTTP-сервера, а express - це фреймворк, який полегшує створення веб-додатків та API на Node.js.

Створюється новий екземпляр Express-додатку за допомогою виклику express() та присвоєння його константі app. Для створення HTTP-сервера викликається метод createServer з модуля http, передаючи як аргумент створений екземпляр Express-додатку app. Це забезпечує маршрутизацію запитів та обробку від Express у межах HTTP-сервера. Створений HTTP-сервер присвоюється константі server.

Для запуску сервера викликається метод listen об'єкта server. Цей метод приймає два аргументи: номер порту, на якому сервер буде слухати вхідні

з'єднання, та колбек-функцію, яка виконується після успішного запуску сервера. У наданому коді номер порту встановлюється як значення змінної оточення process.env.PORT. Це зручно для розгортання додатку на хмарних платформах, де порти часто динамічно призначаються середовищем.

Колбек-функція, яка передається методу listen, викликається після успішного запуску сервера. У цій функції виводиться повідомлення у консоль, яке вказує, що додаток слухає на певному порту. Номер порту отримується викликом listener.address().port, де listener - це об'єкт, який повертається методом listen.

Після виконання цього коду сервер почне слухати вхідні HTTP-запити на зазначеному порту. Коли надходить запит, Express-додаток обробляє його відповідно до визначених маршрутів та middlewares, а потім відправляє відповідь назад клієнту.

3.4. Створення можливості залишати коментарі на веб-сайті

Створивши секції коментарів за допомогою HTML та CSS, тепер необхідно реалізувати можливість користувачам самостійно залишати коментарі. Спершу, нам потрібно отримати доступ до ключових частин сторінки, якими потрібно буде маніпулювати аби звертатися до них пізніше в коді, а також визначити деякі константи класу (Рис. 3.8):

```
const POST_COMMENT_BTN = document.getElementById('post');
const COMMENT_TEXT = document.getElementById('comment');
const COMMENTS_LIST = document.getElementById('commentsList');
// CSS-клас для стилізації, що вказує на обробку коментаря при відправленні,
// щоб надати користувачеві візуальний зворотний зв'язок.
const PROCESSING_CLASS = 'processing';

// Зберігаємо ім'я користувача, який увійшов. Зараз у нас немає автентифікації,
// тому за замовчуванням це "Анонім" до тих пір, поки не буде відомо.
var currentUserName = 'Anonymous';
```

Рис. 3.8. Маніпулювання DOM елементами JavaScript

У даному коді ми отримуємо доступ до HTML-елемента з ідентифікатором 'post' (кнопка для відправлення коментаря) за допомогою методу document.getElementById. Присвоюємо отриманий елемент константі POST_COMMENT_BTN.

Повторюємо наступні дії для інших необхідних HTML-елементів з ідентифікаторами 'comment' (текстове поле для введення коментаря) та з ідентифікатором 'commentsList' (список коментарів). Присвоюємо отримані елемент константам COMMENT_TEXT та COMMENTS_LIST відповідно.

Визначаємо константу PROCESSING_CLASS зі значенням рядка 'processing'. Ця константа використовуватиметься як назва CSS-класу для візуального відображення стану обробки коментаря.

Також оголошуємо змінну currentUserName зі значенням рядка 'Anonymous'. Ця змінна, використовуватиметься для зберігання імені поточного користувача. Поки немає авторизації, ім'я користувача встановлюється як 'Anonymous' за замовчуванням.

Таким чином, ці рядки коду отримують доступ до ключових елементів DOM.

Далі нам потрібно перевірити, чи отримуємо ми дані з тексту коментаря. Для цього створимо функцію, яка буде викликатися при натисканні на кнопку відправлення коментаря та перевірятиме, чи вже відбувається процес обробки, додає відповідний CSS-клас для візуального відображення стану обробки та отримує текст коментаря (Рис. 3.9):

```
function handleCommentPost() {
    if (! POST_COMMENT_BTN.classList.contains(PROCESSING_CLASS)) {
      POST_COMMENT_BTN.classList.add(PROCESSING_CLASS);
      COMMENT_TEXT.classList.add(PROCESSING_CLASS);
      let currentComment = COMMENT_TEXT.innerText;
      console.log(currentComment);
    }
  }

  POST_COMMENT_BTN.addEventListener('click', handleCommentPost);
```

Рис. 3.9. Перевірка отримання коментаря через натискання кнопки

Для перевірки, додаємо виведення в консоль та напишемо простий коментар та натиснемо кнопку «Comment» (Рис. 3.10):

Рис. 3.10. Демонстративне повідомлення

Відкриваємо консоль та переконуємся, що ми отримали текст коментарю(Рис. 3.11):

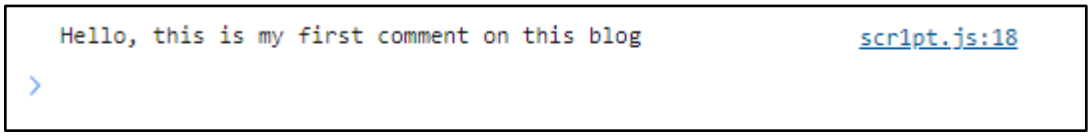


Рис. 3.11. Відображення повідомлення в консолі після відправлення

3.5. Завантаження та тестування моделі на основі штучного інтелекту

Тепер ми можемо завантажити нашу експортовану модель. Для цього ми помістимо її в окрему директорію та додамо в структуру проекту (Рис. 3.12).

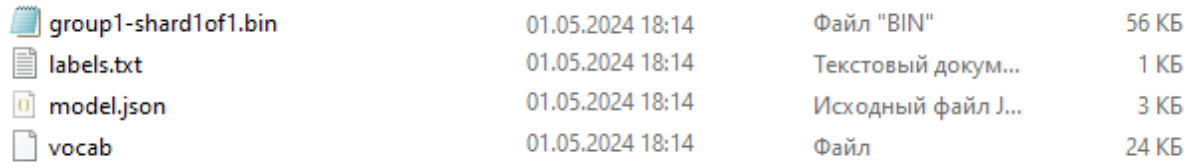


Рис. 3.12. Файли навченої моделі

Розберемо кожен файл окремо:

- **group1-shard1of1.bin** — Це бінарний файл, що містить навчені «ваги» (по суті, це числові параметри, які визначають поведінку та результати моделі при виконанні завдань, наприклад при виконанні завдання класифікації) моделі TensorFlow.js. Для завдання класифікації текстів, ваги налаштовуються так, щоб модель могла розпізнавати певні патерни та особливості у вхідних даних і відповідно присвоювати їм правильні класи або мітки. Цей файл зберігає ці оптимізовані ваги у бінарному форматі, що дозволяє ефективно завантажувати їх у пам'ять під час використання моделі.
- **labels.txt** — Цей .txt файл містить результуючі назви класів, які передбачить модель. Для цієї моделі, якщо відкрити цей файл у текстовому редакторі, він містить "false" та "true", що вказує на "не спам" або "спам" як результат прогнозування роботи.
- **model.json** — Цей файл JSON містить опис архітектури моделі, навченої за допомогою Keras та збереженої у форматі TensorFlow.js. В ньому містяться метадані моделі, такі як формат моделі, версії Keras та TensorFlow.js, які використовувалися для створення та конвертації відповідно, а також опис топології моделі, включаючи тип моделі (Sequential), список «шарів» та їхні конфігурації. Саме він буде використовуватись для завантаження архітектури навченої моделі.
- **vocab** — цей файл містить вокабуляр (словник) для моделі, що використовується в контексті обробки природної мови. У файлі model.json є шар Embedding, який потребує вакабуляру та містить всі можливі слова, що можуть зустрічатися на вході моделі, де кожному слову присвоєно

унікальний числовий індекс для конвертації вхідних слів, які потім подаються на вхід моделі для навчання або передбачення.

Після того, як ми завантажили нашу модель, потрібно перетворити масив чисел у тензор.

Тензори - це багатовимірні масиви з однакового типу даних. По суті своїй, тензори можна розглядати масив будь-якої розмірності, який має дещо інші функції, вбудовані в об'єкт тензорів, корисні для машинного навчання. Слід зазначити, що тензори зберігають дані лише одного типу, наприклад, всі цілі числа або всі числа з плаваючою комою, і після створення ви ніколи не зможете змінити вміст тензора тільки створити новий[28].

Тепер, можна перейти до тестування нашої моделі (Рис. 3.13):

```
async function loadAndPredict(inputTensor) {
  if (model === undefined) {
    model = await tf.loadLayersModel(MODEL_JSON_URL);
  }
  var results = await model.predict(inputTensor);
  results.print();
}
loadAndPredict(tf.tensor([[1,3,7,51,406,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]));
```

Рис. 3.13. Тестування моделі через задане заздалегідь слово

Спочатку ми визначаємо дві константи: MODEL_JSON_URL та SPAM_THRESHOLD. MODEL_JSON_URL містить URL-адресу файлу model.json, а SPAM_THRESHOLD визначає мінімальний поріг впевненості, за якого коментар буде розпізнаватись як спам. Значення 0.75 означає, що якщо модель визначає імовірність спаму на рівні 75% або вище, то коментар буде вважатися спамом.

Далі оголошується змінна model, яка спочатку має значення undefined. Ця змінна буде використовуватись для зберігання завантаженої моделі.

Визначається асинхронна функція loadAndPredict, яка приймає inputTensor - тензор вхідних даних для моделі. Ця функція виконує наступні дії:

Перевіряється, чи змінна model не є undefined. Якщо так, то викликається метод tf.loadLayersModel з MODEL_JSON_URL як аргументом. Цей метод завантажує модель, описану в файлі model.json, а також відповідні бінарні файли з вагами моделі. Оскільки це асинхронна операція, використовується ключове слово await.

Після завантаження моделі викликається метод model.predict з inputTensor як аргументом. Цей метод виконує передбачення моделі на основі вхідних даних. Результат зберігається у змінній results.

Виводиться результат передбачення у консоль за допомогою методу results.print().

Наприкінці коду викликається функція loadAndPredict з прикладом вхідного тензору tf.tensor([[1,3,12,18,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]). Це дозволяє протестувати роботу функції з цим вхідним тензором. Зупинимося на цій функції детальніше.

Оскільки моделі машинного навчання можуть приймати на вхід лише набір чисел, то постає проблема того, як перетворити речення на числа для використання з завантаженою моделлю. Для нашої моделі використано рішення через використання вакабуляра у файлі vocab.

По-суті своїй, він являє собою пошукову таблицю того, як перетворити слова, які вивчила модель, у зрозумілі для неї числа (Рис. 3.14):

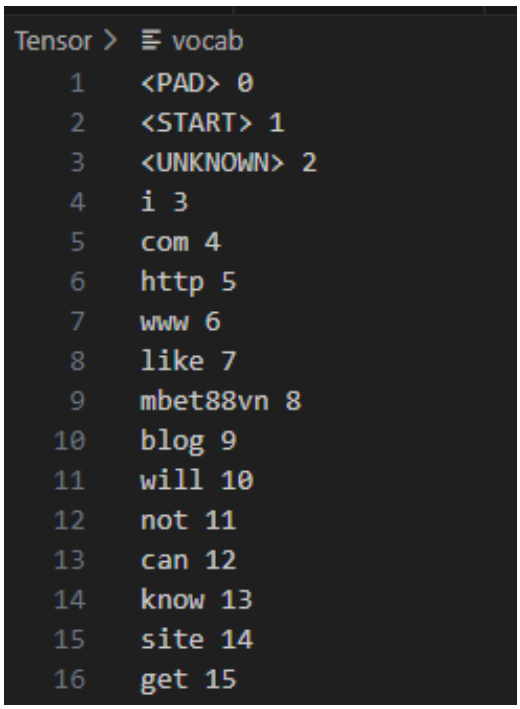


Рис. 3.14. Вміст файлу vocab

У верхній частині файлу є також деякі особливі випадки <PAD>, <START> і <UNKNOWN>:

- **<PAD>** - це скорочення від "padding". Моделі машинного навчання люблять мати фіксовану кількість вхідних даних, незалежно від того, наскільки довгим може бути речення. Використовувана модель очікує, що на вході завжди буде 20 чисел. Отже, якщо взяти фразу на кшталт "Я люблю відео", буде відбуватися заповнення решти пробілів у масиві нулями, які представляють токен <PAD>. Якщо речення містить більше 20 слів, речення потрібно буде розбити на частини, щоб воно відповідало цій вимозі, і замість цього виконати кілька класифікацій на багато менших речень.
- **<START>** - це просто перша мітка, яка Використовується для позначення початку тексту. Кожен вхідний текст починається з цього токена і масив чисел теж починається з "1".
- **<UNKNOWN>** - даний токен використовується, якщо слово не існує у цьому пошуку слів. Він призначений для слів, які відсутні у словнику. Це дозволяє

моделі обробляти нові або рідкісні слова. Якщо слово відсутнє в словнику, у масиві чисел воно заміняється на «2».

Тобто, якщо використовувати наш вакабуляр, то наступний масив чисел [1,3,7,51,406,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]. можна розшифрувати як “I like this video”.

Ініціалізувавши цей код, в консолі нашого веб-сайту з’являється наступне повідомлення (Рис. 3.15):

Рис. 3.15. Виведення значень в консоль

В ній присутні два числа:

1. 0.9540924;
2. 0.0459075.

Ці числа представляють ймовірності того, як модель класифікує введені дані. У файлі labels.txt також міститься два поля:

- False;
- True.

Таким чином, у цьому випадку модель стверджує, що з ймовірністю 95.40924% (показано у результаті як 0.9540924) введені дані (а саме [1,3,7,51,406,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]) НЕ є спамом (тобто, False).

False є першою міткою у файлі labels.txt і відповідає першому числу у виведенні консолі, що дозволяє зрозуміти, до чого відноситься передбачення. Друге число, 0.0459075, вказує на 4.59075% ймовірність того, що введені дані є спамом (тобто, True).

Система успішно протестована та працює, але для того, щоб вона могла працювати з системою коментарів, нам потрібно змінити сам файл вакабуляру. У такому вигляді використання цього файлу є доволі недоцільним в JavaScript. Його

необхідно перетворити у формат JavaScript об'єкта, який можна легко імпортувати та використовувати.

Для цього створимо на основі vocab новий файл під назвою dictionary.js (Рис. 3.16):

```
export const PAD = 0;
export const START = 1;
export const UNKNOWN = 2;
export const LOOKUP = {
"i": 3,
"com": 4,
"http": 5,
"www": 6,
"like": 7,
"mbet88vn": 8,
"blog": 9,
"will": 10,
"not": 11,
"can": 12,
"know": 13,
"site": 14,
"get": 15,
```

Рис. 3.16. Перетворення файлу vocab в об’єкт JavaScript

Визначаємо наші три унікальні токени як три константи: PAD, START та UNKNOWN, які присвоюються значенням 0, 1 та 2 відповідно.

Визначається константа LOOKUP, яка є об'єктом, що буде містити відображення слів на відповідні числові індекси. Цей об'єкт LOOKUP буде основою вакабуляру.

Дане перетворення запобігає необхідності виконувати це перетворення та парсинг при кожному завантаженні сторінки, що економить ресурси процесора. Крім того, об'єкти JavaScript мають наступні властивості:

"Ім'я властивості об'єкта може бути будь-яким допустимим рядком JavaScript або будь-чим, що можна перетворити на рядок, включаючи порожній рядок. Однак, будь-яке ім'я властивості, яке не є дійсним ідентифікатором JavaScript (наприклад,

ім'я властивості, яке має пробіл або дефіс, або яке починається з числа) може бути доступне лише за допомогою квадратних дужок".

Таким чином, використовуючи квадратну нотацію, можна створити досить ефективну таблицю пошуку через це просте перетворення. Це дозволяє швидко і зручно отримувати індекси слів під час роботи з текстовими даними в моделі машинного навчання.

Після того як ми успішно перенесли вакабуляр у формат об’єкт JavaScript, його тепер можна імпортувати в наш основний script.js файл (Рис. 3.17):

```
import * as DICTIONARY from '/dictionary.js';

const ENCODING_LENGTH = 20;


function tokenize(wordArray) {
  let returnArray = [DICTIONARY.START];

  for (var i = 0; i < wordArray.length; i++) {
    let encoding = DICTIONARY.LOOKUP[wordArray[i]];
    returnArray.push(encoding === undefined ? DICTIONARY.UNKNOWN : encoding);
  }

  while (i < ENCODING_LENGTH - 1) {
    returnArray.push(DICTIONARY.PAD);
    i++;
  }

  console.log([returnArray]);

  return tf.tensor([returnArray]);
}
```

Рис. 3.17. Імпорт об’єкту JavaScript в файл script.js

Щоб використовувати словник у файлі script.js, код починається з імпорту модуля dictionary.js. Також визначається додаткова константа ENCODING_LENGTH, яка вказує на необхідну довжину кодування для вхідних даних моделі. Це допоможе визначити, наскільки необхідно доповнити вхідні дані пізніше у коді.

Далі визначається функція tokenize, яка приймає масив слів (wordArray) і повертає тензор, що може бути використаний як вхідні дані для моделі. Функція виконує наступні кроки:

1. Створюється новий масив returnArray, який починається з токену START зі словника (DICTIONARY.START);
2. Для кожного слова у wordArray:
   - Шукається відповідний числовий індекс у словнику DICTIONARY.LOOKUP;
   - Якщо слово знайдено у словнику, відповідний індекс додається до returnArray;
   - Якщо слово не знайдено у словнику, додається токен UNKNOWN (DICTIONARY.UNKNOWN).
3. Після обробки всіх слів, масив returnArray доповнюється токеном PAD (DICTIONARY.PAD) до досягнення довжини ENCODING_LENGTH – 1;
4. Вміст масиву returnArray виводиться у консоль за допомогою console.log([returnArray]);
5. Масив returnArray перетворюється на тензор за допомогою tf.tensor([returnArray]) і повертається як результат функції.

У підсумку функція tokenize перетворює масив слів у відповідний числовий тензор, використовуючи словник DICTIONARY для відображення слів на індекси. Вона також забезпечує доповнення вхідних даних до необхідної довжини за допомогою токену PAD. Результуючий тензор (який був отриманий з коментарю) буде використаний як вхідні дані для нашої моделі.

3.6. Реалізація роботи спам фільтру в реальному часі

Тепер, коли ми маємо працюючу анти спам модель останнім етапом буде використання Node.js з деякими веб-сокетами для комунікації в режимі реального

часу і оновлення в реальному часі всіх коментарів, які додаються і не є спамом. Для цього буде використана інтегрована бібліотека Socket.io.

Спочатку ми оновимо файл нашого веб-сайту, додавши до нього рядок коду з імпортом скриптів (Рис. 3.18):

```
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@3.11.0/dist/tf.min.js" type="text/javascript"></script>
<script src="/socket.io/socket.io.js"></script>
<script src="scr1pt.js" type="module"></script>
```

Рис. 3.18. Імпорт усіх необхідних .js файлів

1. Перший імпортує бібліотеку TensorFlow.js;
2. Другий імпортує socket.io (це потрібно для підключення клієнтської частини socket.io, щоб забезпечити двосторонній зв'язок у реальному часі між клієнтом і сервером.);
3. Третій безпосередньо імпортує JavaScript файл, який відповідатиме за логіку нашої моделі.

Далі редагуємо server.js, щоб налаштувати socket.io всередині Node.js і створити простий бекенд для передачі отриманих повідомлень всім підключеним клієнтам (Рис. 3.19):

```
var io = require('socket.io')(server);
app.use(express.static("Tensor"));

app.get("/", (request, response) => {
  response.sendFile(__dirname + "/Tensor/index.html");
});

io.on('connect', socket => {
  console.log('Клієнт підключений');

  socket.on('comment', (data) => {
    socket.broadcast.emit('remoteComment', data);
  });
});
```

Рис. 3.19. Використання веб-сокетів для обслуговування статичних файлів

У цій частині коду підключається бібліотека socket.io та створюється її екземпляр "io", передаючи об'єкт server (який був створений під час налаштування серверу) як аргумент.

Встановлюється шлях до статичних файлів у директорії "Tensor" (основної директорії проекту) за допомогою app.use(express.static("Tensor")). Це дозволяє серверу обслуговувати статичні файли, такі як HTML, CSS та JavaScript, з цієї директорії.

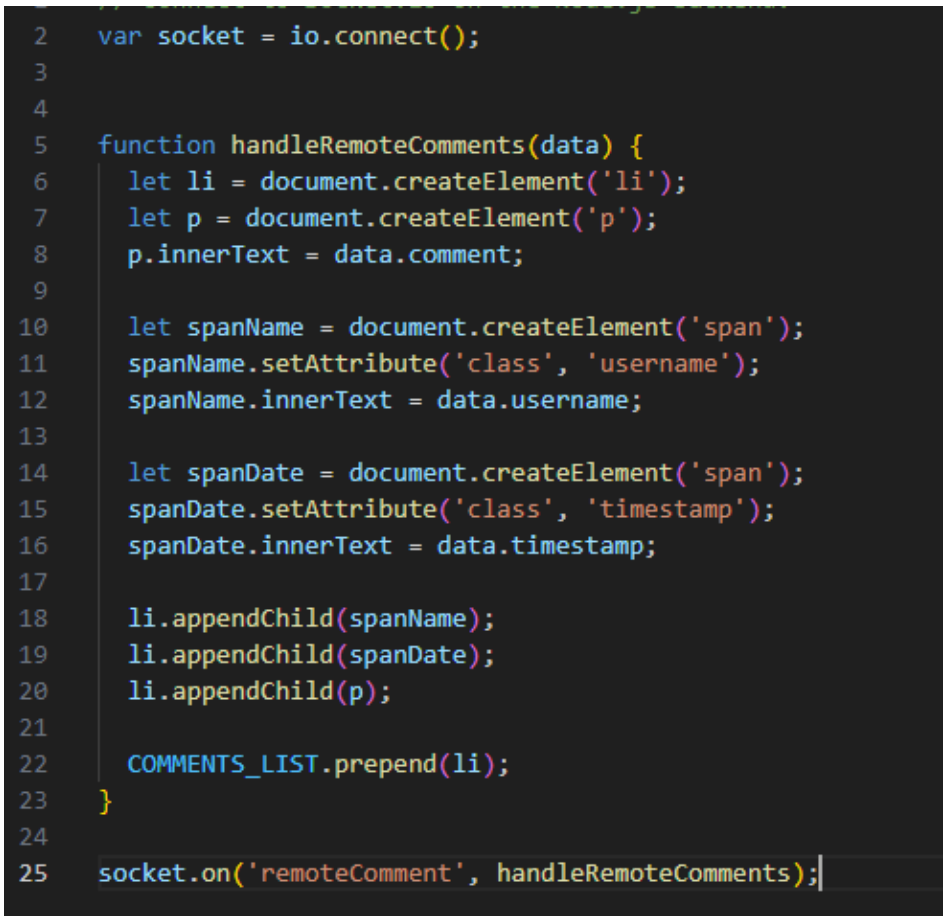
Визначається маршрут для кореневого шляху за допомогою app.get("/", ...). Коли клієнт робить запит на кореневий шлях, сервер відправляє файл index.html з директорії "/Tensor" за допомогою методу response.sendFile(__dirname + "/Tensor/index.html").

Встановлюється обробник події connect на об'єкті "io", який спрацьовує, коли новий клієнт встановлює сокет-з'єднання з сервером. У цьому обробнику виводиться повідомлення "Клієнт підключений" у консоль.

Всередині обробника події connect встановлюється додатковий обробник події comment на об'єкті socket. Цей обробник спрацьовує, коли клієнт надсилає подію comment з даними. Коли це відбувається, сервер передає ці дані всім іншим підключеним клієнтам за допомогою socket.broadcast.emit('remoteComment', data), де remoteComment є назвою події, а data - це самі дані, які були надіслані.

Таким чином, цей код встановлює сервер, який обслуговує статичні файли з директорії "Tensor", а також обробляє сокет-з'єднання. Коли клієнт підключається, сервер виводить повідомлення у консоль. Коли клієнт надсилає подію comment з даними, сервер передає ці дані всім іншим підключеним клієнтам за допомогою події remoteComment, щоб модель могла знати, як відобразити віддалений коментар.

Нарешті, додаємо логіку socket.io до коду на стороні клієнта, щоб генерувати та обробляти ці події (Рис. 3.20):



```
var socket = io.connect();


function handleRemoteComments(data) {
  let li = document.createElement('li');
  let p = document.createElement('p');
  p.innerText = data.comment;

  let spanName = document.createElement('span');
  spanName.setAttribute('class', 'username');
  spanName.innerText = data.username;

  let spanDate = document.createElement('span');
  spanDate.setAttribute('class', 'timestamp');
  spanDate.innerText = data.timestamp;

  li.appendChild(spanName);
  li.appendChild(spanDate);
  li.appendChild(p);

  COMMENTS_LIST.prepend(li);
}

socket.on('remoteComment', handleRemoteComments);
```

Рис. 3.20. Додавання логіки socket.io в файл script.js

Створюється змінна socket за допомогою io.connect(), що встановлює сокет-з'єднання з сервером.

Визначається функція handleRemoteComments, яка приймає об'єкт data як параметр. Ця функція виконується, коли сервер надсилає подію remoteComment з даними коментаря.

Всередині функції handleRemoteComments виконуються такі кроки:

1. Створюється новий елемент li за допомогою document.createElement('li');
2. Створюється новий елемент p за допомогою document.createElement('p'), і текст коментаря з об'єкта data присвоюється його властивості innerText;
3. Створюється новий елемент span для відображення імені користувача за допомогою document.createElement('span'). Йому присвоюється клас username за допомогою spanName.setAttribute('class', 'username'), і ім'я користувача з об'єкта data присвоюється його властивості innerText;

4. Створюється ще один елемент span для відображення часової позначки за допомогою document.createElement('span'). Йому присвоюється клас timestamp за допомогою spanDate.setAttribute('class', 'timestamp'), і часова позначка з об'єкта data присвоюється його властивості innerText;
5. Елементи spanName, spanDate та p додаються як дочірні елементи до li за допомогою методу li.appendChild;
6. Новостворений елемент li вставляється на початок списку коментарів. COMMENTS_LIST за допомогою методу COMMENTS_LIST.prepend(li).

Після визначення функції handleRemoteComments встановлюється обробник події remoteComment на об'єкті socket за допомогою socket.on('remoteComment', handleRemoteComments). Це означає, що коли сервер надсилає подію remoteComment з даними коментаря, функція handleRemoteComments викликається з цими даними як аргументом.

Останнє, що потрібно зробити, додати код до функції loadAndPredict, щоб згенерувати подію socket.io, якщо коментар не є спамом. Це дозволить оновити інших підключених клієнтів новим коментарем, оскільки вміст цього повідомлення буде передано їм через код server.js (Рис. 3.21).

```
async function loadAndPredict(inputTensor, domComment) {

  if (model === undefined) {
    model = await tf.loadLayersModel(MODEL_JSON_URL);
  }
  var results = await model.predict(inputTensor);

  results.print();

  results.data().then((dataArray)=>{
    if (dataArray[1] > SPAM_THRESHOLD) {
      domComment.classList.add('spam');
    } else {

      socket.emit('comment', {
        username: currentUserName,
        timestamp: domComment.querySelectorAll('span')[1].innerText,
        comment: domComment.querySelectorAll('p')[0].innerText
      });
    }
  })
}
```

Рис. 3.21. Реалізація обробки коментаря на сервері

У фінальній частині коді визначена асинхронна функція loadAndPredict, яка виконує наступні дії. Функція приймає два параметри: inputTensor (тензор вхідних даних для моделі) та domComment (елемент DOM, який представляє коментар, який класифікується).

Якщо змінна model не визначена, функція завантажує модель TensorFlow.js з URL, вказаного в MODEL_JSON_URL, за допомогою tf.loadLayersModel. Оскільки це асинхронна операція, використовується ключове слово await.

Після завантаження моделі функція викликає метод model.predict з inputTensor як аргументом, щоб отримати результат передбачення моделі. Результат зберігається у змінній results, а результат передбачення виводиться у консоль за допомогою results.print().

Функція використовує метод results.data(), який повертає проміс з масивом даних результату передбачення. Коли проміс виконується, функція обробляє отримані дані.

Якщо друге значення в масиві даних перевищує значення SPAM_THRESHOLD, до елемента domComment додається клас CSS spam;

Якщо друге значення в масиві даних не перевищує SPAM_THRESHOLD, функція емітує подію comment через socket.emit, надсилаючи об'єкт з наступними властивостями:

- username: значення змінної currentUserName;
- timestamp: текст другого елемента span всередині domComment;
- comment: текст першого елемента p всередині domComment.

Таким чином, функція loadAndPredict завантажує модель TensorFlow.js, отримує результат передбачення для вхідного тензору та виконує відповідні дії залежно від результату. Якщо коментар класифікується як спам, до відповідного елемента DOM додається клас CSS spam. В іншому випадку, функція емітує подію comment через сокет, надсилаючи дані коментаря, імені користувача та часову позначку для подальшої обробки на сервері або віддалених клієнтах.

Для демонстрації роботи відкриємо веб-сторінку з двох різних браузерів, щоб зімітувати підключення двох різних клієнтів (Рис. 3.22):

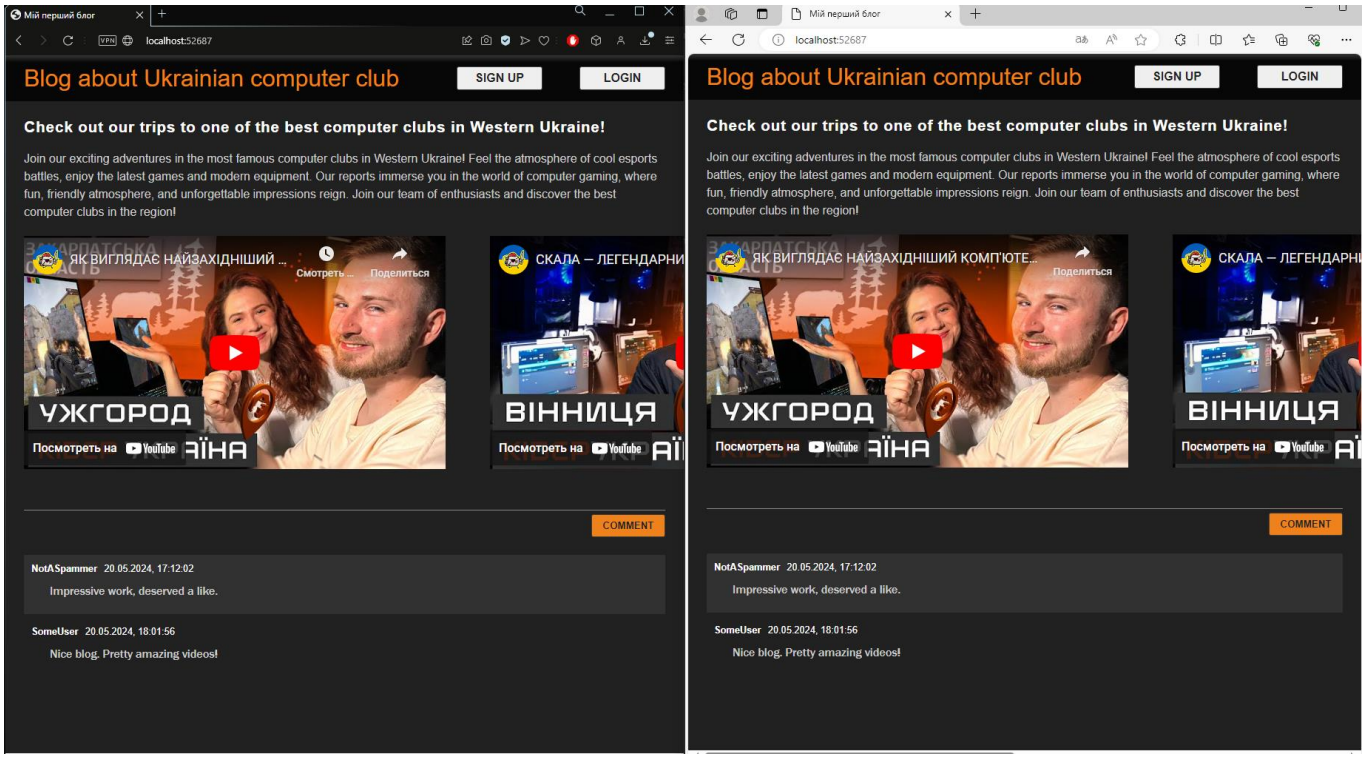


Рис. 3.22. Імітація двох відкритих клієнтів на сервері

Напишемо якийсь коментар, який об’єктивно не буде спамом (Рис. 3.23):

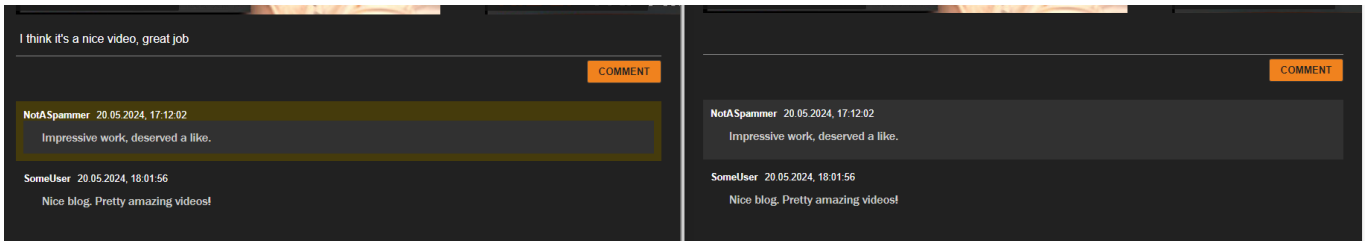


Рис. 3.23. Фіксація початкового вигляду коментарів до відправки повідомлення

Як бачимо, коментар опублікувався для двох клієнтів одночасно (Рис. 3.24).

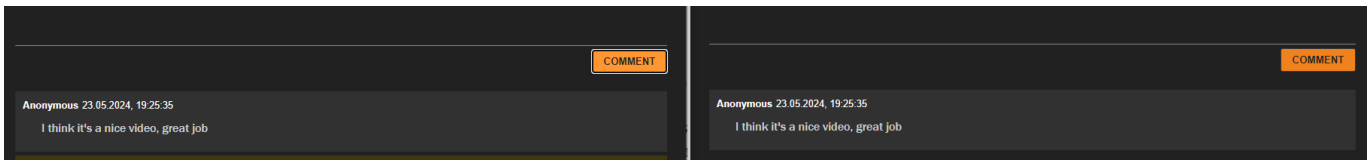


Рис. 3.24. Результат відправки коментаря, що пройшов перевірку

Тепер напишемо очевидний спам-коментар, який буде запрошувати відвідувачів блогу перейти на інший веб-сайт (Рис. 3.25):

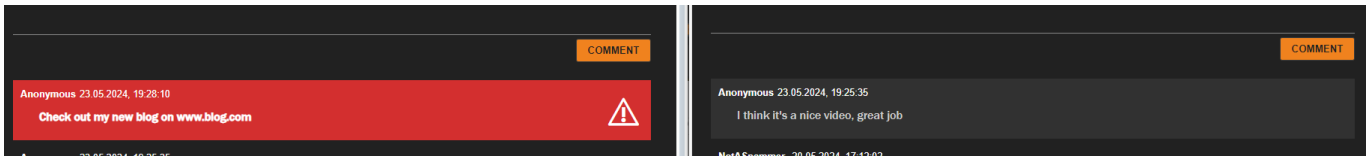


Рис. 3.25 Результат відправки коментаря, що не пройшов перевірку

В результаті бачимо, що клієнту, з якогось відбувалося написання коментаря, відобразилось, що його повідомлення помічене як спам, тим часом коли іншому клієнту не відображається нічого, оскільки повідомлення першого клієнта є спамом і не публікується.

3.7. Висновки до третього розділу

У ході розробки створено програмну реалізацію системи для виявлення спаму в коментарях на веб-сайті з використанням моделі штучного інтелекту на основі TensorFlow.js.

Спочатку представлені інструменти, що використовувалися при розробці: HTML та CSS для створення структури та стилізації веб-сторінки, JavaScript для програмної логіки на клієнтській стороні, TensorFlow.js для завантаження та використання навченої моделі штучного інтелекту, Node.js та Socket.io для створення серверної частини й забезпечення взаємодії в режимі реального часу.

Далі описано процес створення веб-сайту, підключення його до серверу Node.js та реалізацію функціоналу для написання коментарів.

Для цього розроблена клієнтська частина з використанням HTML, CSS та JavaScript, а також серверна частина на Node.js з використанням Socket.io для передачі коментарів між клієнтами в реальному часі.

Наступним кроком стало завантаження попередньо навченої моделі штучного інтелекту на основі TensorFlow.js та її тестування. Модель представлена у вигляді файлів JSON та бінарних файлів з вагами, які завантажені та ініціалізовані в JavaScript-коді.

Реалізовано роботу спам-фільтру в реальному часі. Введений коментар перетворювався на тензор вхідних даних і подавався на вхід завантаженої моделі. Модель виконувала передбачення, чи є коментар спамом, і залежно від результату, коментар або відображався на веб-сторінці, або позначався як спам за допомогою CSS-класу. У випадку, якщо коментар не був спамом, він також передавався іншим підключеним клієнтам через Socket.io для відображення в реальному часі.

Таким чином, у розділі детально описано програмну реалізацію системи для виявлення спаму в коментарях з використанням моделі штучного інтелекту, починаючи від створення веб-сайту та підключення його до серверу, реалізації функціональності коментарів, завантаження та тестування моделі штучного інтелекту, і закінчуючи інтеграцією спам-фільтру в режимі реального часу.

ВИСНОВКИ

Об'єктом цього дослідження є системи фільтрації спаму для веб-ресурсів. Метою роботи є аналіз різних методів розпізнавання спаму, а також розробка програмної реалізації системи для виявлення спаму з використанням моделі штучного інтелекту.

У даній кваліфікаційній роботі розглянуто критичну роль систем фільтрації спаму для забезпечення ефективної роботи веб-ресурсів, підвищення продуктивності користувачів та дотримання стандартів інформаційної безпеки. Проаналізовано різні методи фільтрації спаму, такі як фільтрація на основі контенту, фільтрація на основі списків та гібридні методи, а також алгоритми розпізнавання спаму, включно з машинним навчанням, статистичними методами та фільтрацією на основі правил.

Детально розглянуті три основні методи розпізнавання спаму: баєсівський метод, метод опорних векторів (SVM) та нейронні мережі, з аналізом їх переваг та недоліків.

За результатами дослідження створено програмну реалізацію системи для виявлення спаму в коментарях на веб-сайті з використанням моделі штучного інтелекту на основі TensorFlow.js. Розроблена клієнтська частина з використанням HTML, CSS та JavaScript, а також серверна частина на Node.js з використанням Socket.io для забезпечення взаємодії в режимі реального часу. Також завантажена та протестована попередньо навчена модель штучного інтелекту, і реалізована робота спам-фільтру в реальному часі, де коментарі користувачів аналізувалися моделлю, і залежно від результату, відображалися або позначалися як спам.

Таким чином, мета роботи, якою є розробка програмного забезпечення на основі веб-технологій для інтернет-сторінки, що підвищує ефективність процесу модерації повідомлень, досягнута. Кінцевим результатом стала створена система, яка успішно вирішила поставлене завдання та впроваджена у виробничий цикл, забезпечуючи вдосконалення процесу реалізації продукції.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zdziarski Jh. The end of spam: Bayesian content filtering and the art of statistical language classification / Jh. Zdziarski. — No Starch Press, 2005. — 312 p.
2. Друкер Г. Машини опорних векторів для категоризації спаму / Г. Друкер, Д. Ву, В. Вапнік; Університет Монмута. Монмут: Вид-во ун-ту Монмута, 1999. 1054 с.
3. Братко А., Філіпц Б. Фільтрація спаму з використанням статистичних моделей стиснення даних. Jožef Stefan Institute. Словенія, 2006, № 1. С. 12-18.
4. What Is a Spam Filter and How Does It Work? URL: https://emailable.com/blog/what-is-a-spam-filter-and-how-does-it-work/ (Дата звернення: 23.04.2024).
5. The Impact of Spam Emails on Businesses URL: https://texaport.co.uk/blog/the-impact-of-spam-emails-on-businesses (Дата звернення: 28.04.2024).
6. What is Spam Filtering? URL: https://www.fortinet.com/resources/cyberglossary/spam-filters (Дата звернення: 18.04.2024).
7. What is Spam Filtering and How Does It Work? URL: https://www.checkpoint.com/cyber-hub/threat-prevention/what-is-email-security/what-is-spam-filtering-and-how-does-it-work/ (Дата звернення: 23.04.2024)
8. Productivity, Security & Spam URL: https://www.linkedin.com/pulse/productivity-security-spam-andrew-james?trk=portfolio_article-card_title (Дата звернення: 29.04.2024).
9. The Whys and The Hows of Email Spam Filters URL https://mailtrap.io/blog/spam-filters/ (Дата звернення: 18.04.2024)
10. Design and Implement a Rule-Based Spam Filtering System Using Neural Network URL: https://ieeexplore.ieee.org/document/6086218 (Дата звернення: 22.04.2024).

11. What Is Spam Filtering? Types Of Spam Filters URL: https://www.wallarm.com/what/what-is-spam-filtering-types-of-spam-filters (Дата звернення: 17.04.2024).
12. Methods to combat spam URL: https://fornex.com/en/help/no-spam/ (Дата звернення: 30.04.2024).
13. In Depth: Naive Bayes Classification URL: https://cocalc.com/share/public_paths/8b892baf91f98d0cf6172b872c8ad6694d0f7204/notebooks%2F05.05-Naive-Bayes.ipynb (Дата звернення: 07.05.2024).
14. Naive Bayes spam filtering URL: https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering#cite_note-7 (Дата звернення: 07.05.2024).
15. Understanding Naive Bayes Classifier URL: https://www.simplilearn.com/tutorials/machine-learning-tutorial/naive-bayes-classifier#:~:text=The%20Naive%20Bayes%20classifier%20works,given%20by%20the%20Bayes%20theorem.&text=While%20calculating%20the%20math%20on,getting%20two%20heads%20%3D%201%2F4 (Дата звернення: 07.05.2024).
16. Spam Statistics (2024): New Data on Junk Email, AI Scams & Phishing URL: https://www.emailtooltester.com/en/blog/spam-statistics/#:~:text=162%20billion%20spam%20emails%20are,(numbers%20recorded%20for%202022) (Дата звернення: 07.05.2024).
17. Support Vector Machine (SVM) Algorithm URL: https://www.geeksforgeeks.org/support-vector-machine-algorithm/ (Дата звернення: 08.05.2024).
18. МАШИННЕ НАВЧАННЯ ПРОСТИМИ СЛОВАМИ. ЧАСТИНА 1 URL: http://www.mmf.lnu.edu.ua/en/aren/1739 (Дата звернення: 08.05.2024).
19. What are the disadvantages of support vector machines (SVMs)? URL: https://www.quora.com/What-are-the-disadvantages-of-support-vector-machines-SVMs (Дата звернення: 08.05.2024).
20. What is a neural network? URL: https://www.ibm.com/topics/neural-networks (Дата звернення: 10.05.2024).

21. What’s The Role Of Weights And Bias In a Neural Network? URL: https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f (Дата звернення: 10.05.2024).
22. Empirical risk minimization URL: https://en.wikipedia.org/wiki/Empirical_risk_minimization (Дата звернення: 10.05.2024).
23. Cost, Activation, Loss Function|| Neural Network|| Deep Learning. What are these? URL: https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de (Дата звернення: 10.05.2024).
24. Advantages and Disadvantages of Neural Network URL: https://www.linkedin.com/pulse/advantages-disadvantages-neural-network-suresh-beekhani-8bhhf#:~:text=Neural%20networks%20offer%20powerful%20capabilities,issues%2C%20and%20significant%20computational%20resources. (Дата звернення: 11.05.2024).
25. Introduction to TensorFlow. URL: https://www.tensorflow.org/learn (Дата звернення: 12.05.2024).
26. About Node.js®. URL: https://nodejs.org/en/about (Дата звернення: 13.05.2024).
27. What Socket.IO is. URL: https://socket.io/docs/v4/ (Дата звернення: 13.05.2024).
28. Tensors and operations URL: https://www.tensorflow.org/js/guide/tensors_operations (Дата звернення: 13.05.2024).