**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Чорноморський національний університет**
**імені Петра Могили**
**Факультет комп'ютерних наук**
**Кафедра інтелектуальних інформаційних систем**

**ДОПУЩЕНО ДО ЗАХИСТУ**
В.о. завідувача кафедри інтелектуальних
інформаційних систем, канд. техн. наук, доцент
_____Є. В. Сіденко
«____»_____202_ р.

**МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА**

# АНАЛІЗ ТОНАЛЬНОСТІ КОМЕНТАРІВ З ВИКОРИСТАННЯМ МАШИННОГО НАВЧАННЯ

Спеціальність 122 «Комп'ютерні науки»

**122 – МКР – 601. 21710210**

***Виконав студент 6-го курсу, групи 601***
_____ ***О. А. Єрмолаєв***
«16» лютого 2023 р.

***Керівник: к.ф.м.н., доцент***
_____ ***І. В. Кулаковська***
«16» лютого 2023 р.

**Миколаїв – 2023**

**Чорноморський національний університет ім. Петра Могили**
**Факультет комп'ютерних наук**
**Кафедра інтелектуальних інформаційних систем**

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
*(шифр і назва)*
Спеціальність **122 «Комп'ютерні науки»**
*(шифр і назва)*

**ЗАТВЕРДЖУЮ**
В.о. завідувача кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент
_____ Є. В. Сіденко
**«  »  20  р.**

# ЗАВДАННЯ
## на магістерську кваліфікаційну роботу
**Єрмолаєву Олександру Андрійовичу**
(прізвище, ім'я, по батькові)

1. Тема магістерської кваліфікаційної роботи «Аналіз тональності коментарів з використанням машинного навчання».
Керівник роботи Кулаковська Інесса Василівна, канд. фіз. мат. наук, доц.
Затв. наказом Ректора ЧНУ ім. Петра Могили від «03» листопада 2022 р. № 199
2. Строк подання студентом роботи 16 лютого 2023 р.
3. Вхідні (початкові) дані до роботи: коментарі з різних Інтернет-джерел та їх оцінка тональності; інший набір відгуків та їх оцінка спам або не спам.
Очікуваний результат: система аналізу тональності коментарів та визначення їх належності до спаму.
4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):
- огляд існуючих аналогів аналізу тональності тексту та пошуку спаму;
- розробка технічного завдання та вимог до системи;
- аналіз та вибір структуру нейронної мережі;
- створення та навчання нейронної мережи для аналізу тональності;
- створення системи для пошуку спаму на основі отриманої нейромережі;

– порівняльний аналіз результатів із застосування різних методів машинного навчання.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: забезпечення вимог охорони праці у приміщенні серверної кімнати.

7. Консультанти:

| **Розділ** | **Прізвище, ініціали та посада консультанта** | **Підпис** |
|---|---|---|
| Спеціальна частина з охорони праці | докт. біол. наук, проф. Григор'єва Л. І. | |
| Методична частина | к.ф.м.н., доцент, Кулаковська І. В. | |

Керівник роботи __к.ф.м.н., доцент, Кулаковська І. В._____
*(наук. ступінь, вчене звання, прізвище та ініціали)*

_____
*(підпис)*

Завдання прийнято до виконання ___Єрмолаєв О. А._____
*(прізвище та ініціали)*

_____
*(підпис)*

Дата видачі завдання «__09__» _____листопада____ 2022 р.

# КАЛЕНДАРНИЙ ПЛАН
## Виконання магістерської кваліфікаційної роботи

Тема: Аналіз тональності коментарів з використанням машинного навчання

| № | Найменування роботи | Початок | Закінчення | Примітки |
|---|---|---|---|---|
| 1 | Визначення керівника і теми МКР. Подання заяви на затвердження теми МКР | 01.09.2022 | 20.10.2022 | Виконано |
| 2 | Отримання завдання на виконання МКР | 21.10.2022 | 10.11.2022 | Виконано |
| 3 | Складання календарного плану на період виконання МКР | 11.11.2022 | 15.11.2022 | Виконано |
| 4 | Огляд літератури за темою дослідження | 16.11.2022 | 27.11.2022 | Виконано |
| 5 | Проходження переддипломної практики, збір та аналіз матеріалів до МКР | 28.11.2022 | 18.12.2022 | Виконано |
| 6 | Аналіз предметної області та розробка технічного завдання. Моделювання результатів | 19.12.2022 | 12.01.2023 | Виконано |
| 7 | Опис фахової частини МКР, зокрема огляд існуючих аналогів, аналіз та вибір структури нейронної мережі, її створення, навчання та використання для пошуку спаму, аналіз отриманих результатів | 13.01.2023 | 25.01.2023 | Виконано |
| 8 | Розробка спеціальної частини з охорони праці та методичної частини | 26.01.2023 | 02.02.2023 | Виконано |
| 9 | Попередній захист МКР на засіданні комісії кафедри | 03.02.2023 | 03.02.2023 | Виконано |
| 10 | Корегування роботи за результатами попереднього захисту | 04.02.2023 | 06.02.2023 | Виконано |
| 11 | Остаточне оформлення пояснювальної записки та слайдів доповіді для захисту | 07.02.2023 | 09.02.2023 | Виконано |
| 12 | Подання МКР рецензенту | 09.02.2023 | 10.02.2023 | Виконано |
| 13 | Рецензування МКР | 11.02.2023 | 12.02.2023 | Виконано |
| 14 | Подання МКР, її електронної копії та інших документів (відгуку, рецензії) до захисту | 15.02.2023 | 16.02.2023 | Виконано |
| 15 | Захист МКР перед екзаменаційною комісією (ЕК) | 22.02.2023 | 23.02.2023 | Виконано |

Розробив студент Єрмолаєв О. А. _____
*(прізвище та ініціали)* (*підпис*)

Керівник роботи к.ф.м.н., доцент, Кулаковська І. В. _____
(*наук. ступінь, вчене звання, прізвище та ініціали)* *(підпис)*

«12» листопада 2022 р.

# АНОТАЦІЯ

до магістерської кваліфікаційної роботи
студента групи 601 ЧНУ ім. Петра Могили

**Єрмолаєва Олександра Андрійовича**

на тему: **“АНАЛІЗ ТОНАЛЬНОСТІ КОМЕНТАРІВ З ВИКОРИСТАННЯМ МАШИННОГО НАВЧАННЯ”**


**Актуальність** даного дослідження полягає у необхідності підвищення точності пошуку спаму. Багато відвідувачів інтернет-магазинів дивляться на відгуки до товару перед покупкою, а користувачі відеохостингів часто орієнтуються на коментарі перед переглядом. Особливо сильно зросла кількість спам-коментарі під час повномасштабного вторгнення, коли ворог за допомогою ботів намагається посіяти паніку та заспамити Інтернет простір. Часто такі коментарі відрізняються за емоційним забарвленням від звичайних, тому існує сенс використовувати аналіз тональності для їх виявлення.

**Об’єктом** дослідження є процеси організації визначення тональності коментарів.

**Предметом** дослідження є методи організації визначення тональності коментарів.

**Метою** роботи є покращення якості пошуку спаму за допомогою визначення тональності коментарів з використанням машинного навчання.

В результаті виконання роботи було розроблено рекурентну нейрону мережу для визначення тональності коментарів. Дана нейрона мережа була використана для визначення спаму, що дало приріст точності з 88% до 93% для методу наївного байєсівського класифікатора та з 91% до 96% для методу випадкового лісу.

Дана робота складається з шести розділів. Кожен розділ відповідно присвячений: аналізу предметної області, нейромережам та LSTM, проектуванню та навчанню нейромережі для аналізу тональності, використанню створеної нейромережі для оптимізації пошуку спаму, охороні праці, методичній частині МКР. Загальний обсяг роботи – 111 сторінки. Магістерська кваліфікаційна робота містить три додатки, 49 рисунків, 6 таблиць і посилання на 55 літературних джерел.



**Ключові слова:** аналіз тональності, пошук спаму, нейромережі, аналіз тексту, Python.

**ABSTRACT**

to the master's qualification work by the student of the group 601 of Petro Mohyla Black Sea National University

**Iermolaiev Oleksandr**

**“ANALYSIS THE TONE OF COMMENTS USING MACHINE LEARNING”**

A relevance of this study lies in the need to improve the accuracy of spam detection. Many visitors of online stores look at product reviews before making a purchase, such as users of video hosting providers are often look at reviews before watching a video. The number of spam comments increased especially strongly during a full-scale invasion, when the enemy use bots to sow panic and spam the Internet. Very often such spam comments have different emotional color from ordinary ones, so it makes sense to use sentiment analysis to detect them.

An object of research is the process of organizing the determination of the sentiment analysis.

A subject of the research is the methods of organizing sentiment analysis of comments.

A purpose of the study is to improve the quality of spam detection using sentiment analysis of comments, with usage of machine learning.

As a result of the work, a recurrent neural network was developed to determine sentiment analysis of comments. This neural network was used to detect spam, which gave an increase in accuracy from 88% to 93% for the naive Bayesian classifier method and from 91% to 96% for the random forest method.

This work consists of six chapters. Each of them is devoted to: analysis of the subject area, neural networks and LSTM, design and training of a neural network for sentiment analysis, usage of this neural network to optimize the spam detection, labor protection and life safety, methodological part of the master's work.

The overall scope of the work is 111 pages. Thesis contains 3 applications, 49 figures, 6 tables and 55 sources in it.

**Key words:** sentiment analysis, spam detection, neural network, text analyze, Python.

**ЗМІСТ**

# ПЕРЕЛІК СКОРОЧЕНЬ

ДКЧП – довга короткочасна пам'ять

МКР – магістерська кваліфікаційна робота

ОС – операційна система

РНМ – рекурентна нейрона мережа

ШІ – штучний інтелект

API – application programming interface

CPU – central processing unit

GPU – graphics processing unit

LSTM – long short-term memory

SSD – solid-state drive

# Пояснювальна записка

**до магістерської кваліфікаційної роботи**

на тему:

# «АНАЛІЗ ТОНАЛЬНОСТІ КОМЕНТАРІВ З ВИКОРИСТАННЯМ МАШИННОГО НАВЧАННЯ»

Спеціальність 122 «Комп'ютерні науки»

**122 – МКР – 601.21710210**

***Виконав студент 6-го курсу, групи 601***
_____ ***О. А. Єрмолаєв***
«16» лютого 2023 р.

***Керівник: к.ф.м.н., доцент***
_____ ***І. В. Кулаковська***
«16» лютого 2023 р.

**Миколаїв – 2023**

# ВСТУП

У наш час люди все більше і більше проводять часу в Інтернеті та відвідують різноманітні сайти. Багато з цих сайтів мають коментарі, що допомагають людям приймати рішення. Так, багато відвідувачів інтернет-магазину дивиться на відгуки до товару перед покупкою, а користувачі відеохостингів часто орієнтуються на коментарі перед переглядом. Проте не всі коментарі однаково корисні, досить часто можна зустріти спам-коментарі які не несуть жодної корисної інформації. Особливо сильно зросла кількість спам-коментарі під час повномасштабного вторгнення, коли ворог за допомогою ботів намагається посіяти паніку та заспамити Інтернет простір. Часто такі коментарі відрізняються за емоційним забарвленням від звичайних, тому існує сенс використовувати аналіз тональності для їх виявлення.

Задача аналізу тональності потребує розуміння природної мови, тому часто для цього використовують методи машинного навчання та штучного інтелекту. Саме означення змісту слова “розуміти” – одна з головних задач штучного інтелекту [1]. Крім того, популярні сервіси можуть мати десятки або сотні тисяч коментарів, а таку кількість людина не зможе проаналізувати сама. Тому корисно буде мати застосунок, який визначати їх емоційне забарвлення, тобто аналізує тональність, та на основі цього робить припущення про належність коментаря до спаму.

Об’єктом є процеси організації визначення тональності коментарів, а предметом є методи організації визначення тональності коментарів.

Метою роботи є покращення якості пошуку спаму за допомогою визначення тональності коментарів з використанням машинного навчання.

Для досягнення визначеної мети необхідно вирішити такі завдання:

– розглянути та проаналізувати існуючі аналоги;
– розробити технічне завдання;
– обрати структуру нейронної мережі;

– створити та навчити нейрону мережу для аналізу тональності;

– розробити систему пошуку спаму на основі створеної нейромережі;

– проаналізувати отриману систему з використанням різних методів машинного навчання;

– створити API та задеплоїти працюючу систему на сервер.

Для розв’язання задач було використано мову програмування Python. Для створення та навчання нейромереж було використано програмне середовище Jupyter для Python та бібліотека Torch. Був використаний фреймворк FastAPI для створення API для нейромережі. Отриману систему в результаті було розгорнуто на сервері за допомогою технології контейнеризації Docker та з використання надбудови над нею – Docker Compose. Результати роботи були представлені на Всеукраїнській науково-практичній конференції молодих вчених, аспірантів і студентів “Інформаційні технології та інженерія”.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ. ПОСТАНОВКА ЗАДАЧІ

## 1.1 Опис предметної сфери

До ШІ на сьогодні відносять алгоритми, відмінною властивістю яких є те, що вони можуть “накопичувати досвід”. Таким чином вони здатні вирішувати деякі завдання с неформалізованими умовами, що не здатна зробити жодна скільки завгодно продуктивна, але жорстко запрограмована комп'ютерна система [2].

Нейромережа складається з системи з'єднаних і взаємодіючих між собою простих процесорів, які називаються нейронами. Вони зазвичай досить прості, особливо в порівнянні з процесорами, які використовують в персональних комп'ютерах. Кожен нейрон подібної мережі має справу тільки з сигналами, які він періодично отримує, і сигналами, які він періодично посилає іншим нейронам [3]. Тим не менше, будучи з'єднаними в досить велику мережу, такі окремі прості частини разом здатні виконувати досить складні завдання.

Області застосування нейронних мереж досить різноманітні – це розпізнавання тексту й мови, семантичний пошук, експертні системи і системи підтримки прийняття рішень, передбачення курсів акцій, системи безпеки, аналіз текстів тощо. У цій роботі розглядається приклад використання нейронної мережі для аналізу тональності коментарів.

Напевно, в кожній предметній області при найближчому розгляді можна знайти постановки нейромережевих завдань [4]. Ось перелік окремих областей, де рішення такого роду завдань має практичне значення вже зараз: економіка і бізнес, медицина, зв'язок та інтернет, автоматизація виробництва, політологічні і соціологічні технології, безпеку та охоронні системи, введення та обробка інформації, геологорозвідка.

Однією з задач яку може вирішувати штучний інтелект – аналіз тональності тексту (англ. sentiment analysis). Це позначає аналіз емоційного забарвлення тексту враховуючи його контекст. Тональність – це емоційне ставлення автора

висловлювання до деякого об'єкту, виражене в тексті. Емоційна складова, виражена на рівні лексеми або комунікативного фрагмента, називається лексичної тональністю (або лексичним сентиментом). Тональність всього тексту в цілому можна визначити як функцію (в найпростішому випадку суму) лексичних тональностей складових його одиниць (пропозицій) і правил їх поєднання. Основною метою аналізу тональності є знаходження думок в тексті і виявлення їх властивостей [5]. Які саме властивості будуть досліджуватися, залежить вже від поставленого завдання. У цій роботі аналіз тональності було використано для віднесення тексту до одного з п’яти класів, тобто для вирішення задачі класифікації.

Під задачами класифікації розуміються задачі на розбиття множини вхідних сигналів на попередньо задану кількість класів. Після навчання така мережа здатна визначати, до якого класу належить вхідний сигнал. В деяких різновидах нейромережа може сигналізувати про те, що вхідний сигнал не відноситься ні до одного з виділених класів – це є ознакою появи нових даних, відсутніх в навчальній вибірці, або про невірні вхідні дані [6].

## 1.2 Огляд та аналіз наявних аналогів

Сфера аналізу тональності тексту є досить нової, а нові технології у ній з’являються кожен рік. На даний момент не існує єдиного загальноприйнятого рішення для цього. Крім того, багато компаній намагають розроблювати схожі системи, які орієнтовані на свою вузьку сферу. Одним з найбільш популярних засобів аналізу тональності тексту є використання тонального словника, в якому кожне слово має оцінку, наприклад “позитивна”, “негативна” або “нейтральна”. Для отримання кінцевого результату потрібно обчислити значення двох оцінок: позитивної складової тексту і негативною. І для того щоб порахувати загальну емоційне забарвлення тексту, порахувати суму оцінок тональностей всіх слів

тексту. Незважаючи на невисоку точність наведеного способу, він використовується у деяких сервісах.

Один з найбільш популярних засобів для аналізу тональності тексту у даний момент є Cognitive Service від Microsoft Azure [7]. Це одна з багатьох хмарних технологій від Microsoft. Крім аналізу тональності тексту, він також визначає мову та надає інформацію про ключові фрази. Головна сторінка цього сервісу з прикладом використання зображено на рис 1.1.

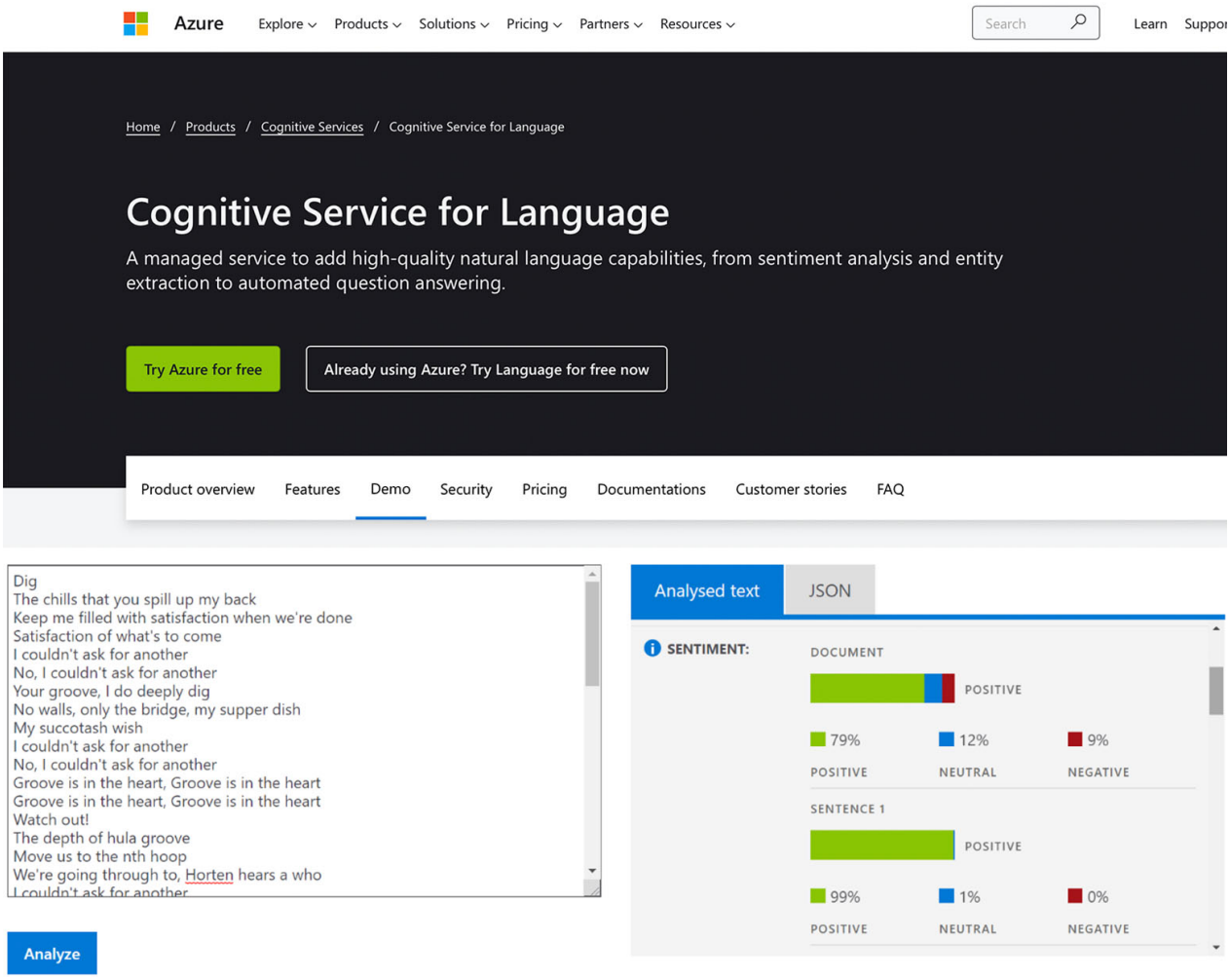


Рисунок 1.1 – Інтерфейс сервісу Microsoft Azure Cognitive Service

Проте цей сервіс має декілька недоліків. Так як він є частиною хмарної платформи Microsoft Azure, то для його використання потрібно ознайомитися з основами цієї платформи. Шлях його використання включає наступні кроки: спочатку налаштувати доступ використовуючи Azure Resource Group, далі

створити Azure Key Vault щоб отримати API ключ, потім створити Azure Storage Account щоб зберігати результати аналізу, і тільки після цього можна перейти до використання Azure Cognitive Service сервісу.

Також, нейромережі для цього сервісу були навчені на даних із текстів журналів та статей з інтернету, тому він буде давати значну похибку при аналізі текстів невеликих коментарів.

Ще одним популярним рішенням для аналізу тональності тексту є Lexalytics [8]. Цей сервіс дозволяє аналізувати не тільки простий текст, а й текст з фото. На даний момент він підтримує 20 різних мов. Цей сервіс надає вибір з різних нейромереж, кожна з яких підходить під свою задачу. Головна сторінка цього сервісу має зрозумілий інтерфейс, зображений на рис 1.2.

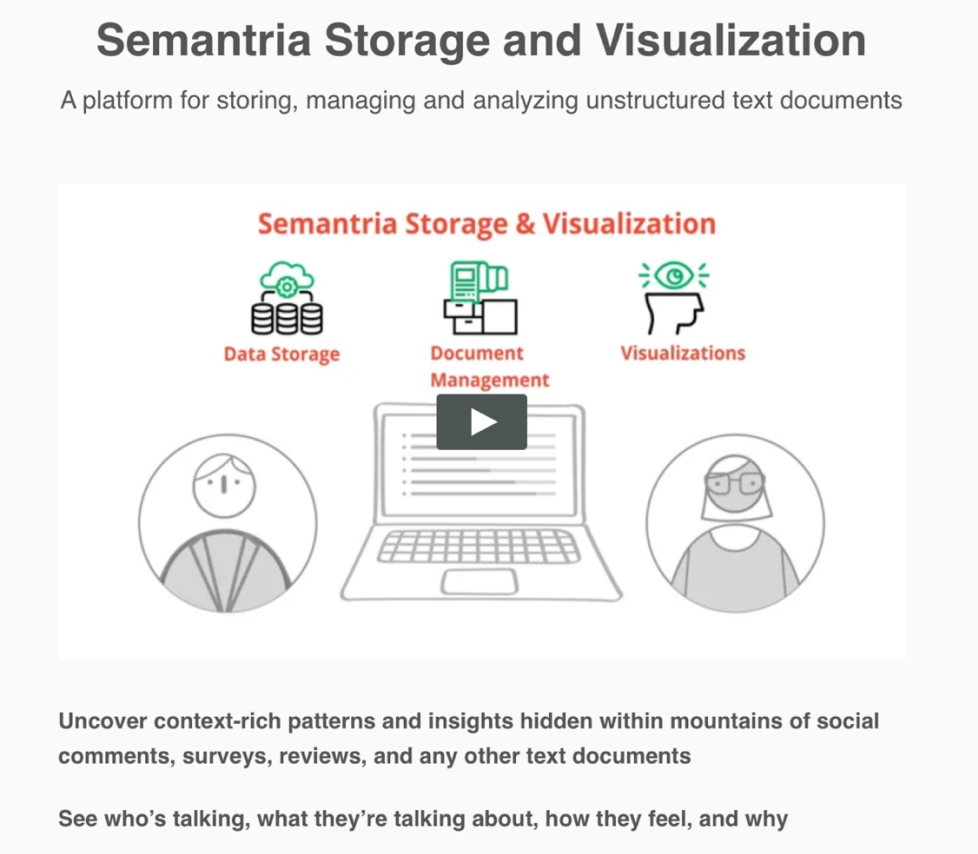


Рисунок 1.2 – Інтерфейс сервісу Lexalytics

Проте значним недоліком цього сервісу є те, що він вимагає зберігати текст та фото у їхньому сховищі даних. Крім того, вартість кожної з представлених мереж

у цьому сервісі значно відрізняється. На додаток, доведеться заплатити за тюнінг (підгонку) готової нейромережі під власні дані.

Друга частина МКР використовує аналіз тональності тексту для визначення спаму. На даний момент існує велика кількість сервісів для визначення спаму. Частіше всього знаходити спам потрібно у email повідомленнях та у коментарях. Найпопулярнішим сервісом електронної пошти є Google Gmail [9]. Він використовує комбінацію машинного навчання та обробки природної мови (Natural language processing) для ідентифікації та фільтрації спаму.

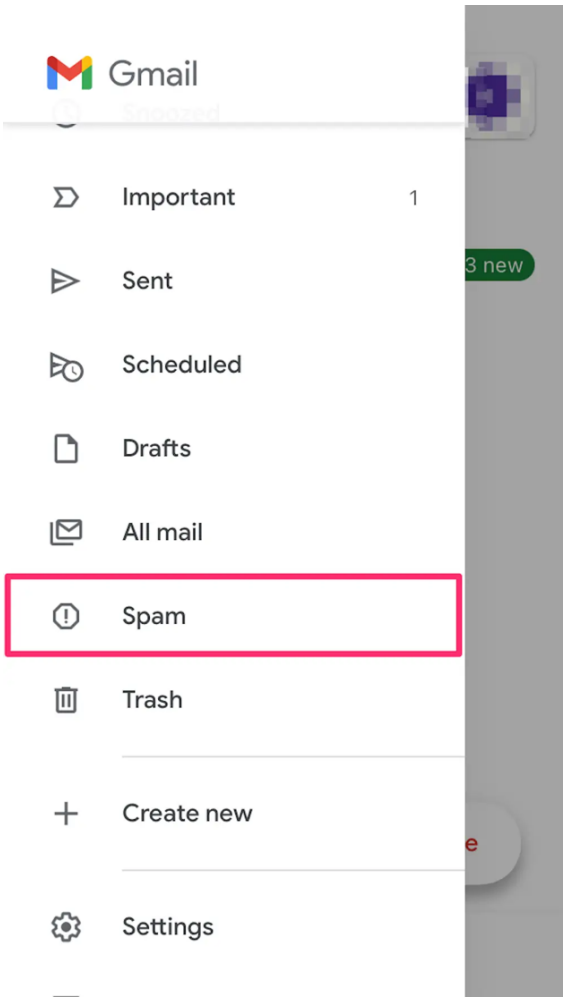


Рисунок 1.3 – Інтерфейс сервісу Google Gmail

Цей сервіс має наступні переваги:

– висока точність: Google Gmail має добре сформовану репутацію за ефективну ідентифікацію та фільтрацію спаму. Це пов’язано з використанням

передових алгоритмів машинного навчання, які постійно оновлюються та вдосконалюються;

– інтеграція з іншими службами Google. Gmail тісно інтегрований з іншими службами Google, такими як Google Drive і Google Calendar, що робить його зручним вибором для користувачів, які вже користуються цими службами;

– велика база користувачів – Gmail має велику базу користувачів, а це означає, що доступно багато даних для навчання алгоритмів машинного навчання. Це дозволяє з високою точністю ідентифікувати спам.

Проте він має деякі недоліки:

– обмежені можливості налаштування: спам-фільтри Gmail налаштовано за допомогою універсального підходу, що означає, що користувачі не можуть повністю налаштувати спам-фільтри відповідно до своїх потреб;

– відсутність конфіденційності. Оскільки Gmail є безкоштовною службою, Google використовує дані електронних листів користувачів для персоналізації реклами та іншого вмісту. Крім того, Gmail може використовувати вміст повідомлень для навчання алгоритмів.

## 1.4 Постановка задачі

Технічне завдання повинне містити в собі чіткий опис усіх елементів системи з точки зору користувацького досвіду, технічних аспектів та наявності потрібного функціоналу.

Розробка повинна складатися з двох частин:

1) створення нейромережі для аналізу тональності коментарів;

2) використання створеної нейромережі для пошуку спаму у коментарях.

Вимоги до функціональних характеристик нейромережі для аналізу тональності:

– класифікація коментарів по п’ятибальній шкалі, де 1 бал позначає емоційно-негативне забарвлення, а 5 балів – емоційно-позитивне;

– середнє квадратичне відхилення оцінки менше одного балу за шкалою вище;

– точність більше ніж 75%;

– навчальний розмір датасету більше ніж 500000 коментарів;

– швидкість передбачення нейромережі менше ніж 100 мс на одне речення.

Також було сформульовані наступні вимоги до функціональних характеристик системи пошуку спаму:

– робота без та з використанням нейромережі для аналізу тональності коментарів;

– робота з використанням декількох алгоритмів машинного навчання;

– можливість використання через API для аналізу;

– можливість аналізу групами по декілька коментарів (батчами) за один запит до API.

Навчати рекурентні нейромережі найкраще всього на відеокартах (GPU). Для того щоб процес навчання був швидшим, потрібно мати кращу відеокарту та більше оперативної пам’яті (RAM). Крім того, датасет має досить великий розмір, тому однією з вимогою буде наявність достатньої кількості дискового простору. Краще за все обрати SSD диск ніж HDD, так як при навчанні нейромережі відбувається активне читання датасету, особливо якщо він цілком не вміщується в оперативній пам’яті [10]. Швидкість інтернет з’єднання не є критично важливою у даній задачі, проте вона повинна бути достатньою для завантаження великого файлу з датасетом. Мінімальна апаратна конфігурація комп’ютера, на якому може бути запущене навчання нейромережі для аналізу тональності коментарів:

– відеокарта Tesla K80 або краща;

– процесор: Xeon 2.3 GHz або кращий;

– від 32 ГБ оперативної пам’яті;

– від 32 ГБ вільного місця на SSD;

– стабільне швидкісне інтернет з’єднання від 100 МБ/с.

Для використання отриманої навченої нейромережі можна використовувати як CPU так і GPU. Крім того, для задачі пошуку спаму вимоги до технічного забезпечення значно менші, аніж до навчання нейромережі для аналізу тональності. Наявність GPU та великого об’єму оперативної пам’яті не є обов’язковою для цього. Мінімальна апаратна конфігурація комп’ютера, на якому може відбуватися навчання пошук спаму за допомогою отриманої нейромережі:

– процесор: Xeon 2.3 GHz або кращий;
– від 8 ГБ оперативної пам’яті;
– від 16 ГБ вільного місця на SSD.

**Висновки до розділу 1**

Розглянувши найпопулярніші рішення, можна прийти до висновку, що на даний момент часу системи, яка б вирішувала всі необхідні задачі та потреби, не існує, а тому розробка такого рішення є досить актуальною. Крім того, сфера аналізу тональності тексту є досить новою, і на ринку існує невелика кількість готових рішень, на відміну від сфері пошуку спаму, де існує велика кількість промислових сервісів.

Також у даному розділі було сформульовано технічні та функціональні вимоги до застосунку. Вони описані окремо як для створення нейромережі для аналізу тональності коментарів, так і для задачі пошуку спаму на основі цієї нейромережі.

# 2 АРХІТЕКТУРА РЕКУРЕНТНИХ НЕЙРОНИХ МЕРЕЖ ТА LSTM

## 2.1 Штучні нейронні мережі та їх структура

Нейронна мережа – математична модель, яка побудована на основі принципу організації та функціонування біологічних нейронних мереж, а саме на основі нервових клітин живого організму. Ця модель з’явилась при вивченні процесів, що відбуваються в мозку живих організмів та при спробі змоделювати ці процеси. Після удосконалення алгоритмів навчання ШІ ці моделі стали використовувати в практичних цілях, наприклад в задачах прогнозування, для розпізнавання образів, в задачах управління [11].

Системи ШІ здатні виконувати завдання, які зазвичай вимагають людського інтелекту, наприклад навчання, досвіду та прийняття рішень на основі цих знань. Системи штучного інтелекту використовують різні алгоритми, включаючи машинне та глибоке навчання, щоб накопичувати досвід і покращувати свою точність з часом. Навчаючись на даних і маючи можливість визначати закономірності та робити прогнози, системи штучного інтелекту можуть надавати розуміння та вирішувати проблеми в різноманітних сферах. Нейронні мережі привабливі з інтуїтивної точки зору, бо вони засновані на примітивній біологічній моделі нервових систем. В майбутньому розвиток таких нейробіологических моделей може привести до створення дійсно розумних комп'ютерів. Зі збільшенням кількості даних, які генеруються щодня, алгоритми ШІ стали важливими для отримання інформації та цінності з даних, що робить їх критично важливими інструментами для сучасних компаній і галузей.

Потужність ШІ полягає в його здатності навчатися та адаптуватися до наданих даних, що робить його здатним виконувати складні завдання, які людям важко або неможливо виконати вручну. Це стало можливим завдяки різноманітним алгоритмам машинного навчання, таким як глибоке навчання, навчання з підкріпленням і обробка природної мови, які дозволяють системам ШІ обробляти

та розуміти великі обсяги неструктурованих даних у режимі реального часу. З точки зору машинного навчання, нейронна мережа являє собою окремий випадок методів розпізнавання образів, дискримінантного аналізу, методів кластеризації. З математичної точки зору, навчання нейронних мереж – це багатопараметричне завдання нелінійної оптимізації [4, 12].

В останні роки комп'ютери накопичили стільки обчислювальної потужності, що тепер будь-який ноутбук може створити нейромережу із сотнею нейронів і прогнати її через мільйони циклів навчання. А відкриття нових типів нейромереж дозволяє підібрати найоптимальнішу модель для кожної задачі. Крім того, кожна модель має багато гіперпараметрів з якими можна експериментувати.

Одні і ті ж види моделей машинного навчання можуть вимагати різні розміри та швидкості навчання для різних видів даних. Ці параметри називаються гіперпараметрами і їх слід налаштовувати так, щоб модель могла оптимально вирішити конкретне завдання [13]. Для цього знаходиться набір гіперпараметрів, який дає оптимальну модель, що оптимізує задану функцію (або набір функцій) втрат на заданих незалежних даних. Цільова функція бере набір гіперпараметрів і повертає розраховані функції втрат.

Найпоширеніші типи нейромереж представлено на рисунку 1.1. Серед них найпопулярнішим и є:

– перцептрон;
– багатошаровий перцептрон;
– рекурентна нейронна мережа;
– згорткова нейронна мережа;
– імпульсна нейронна мережа.

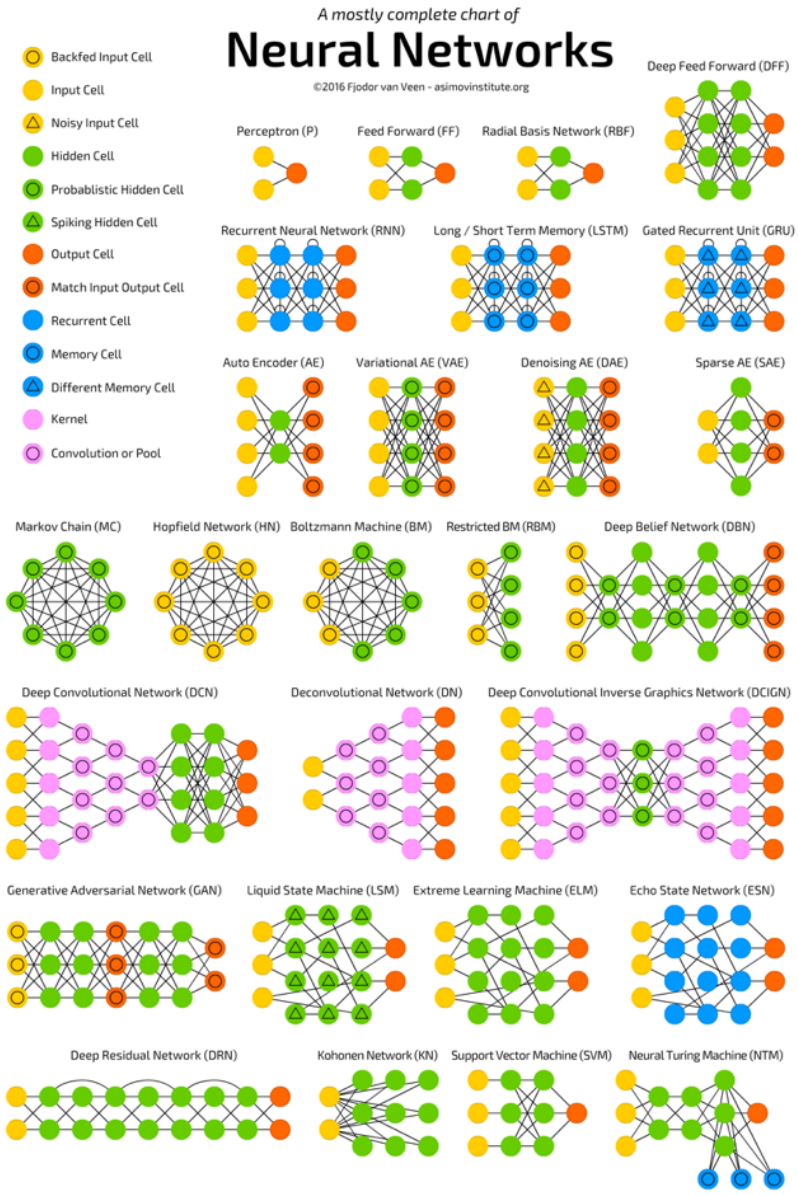


Рисунок 2.1 – Типи нейромереж

Нейронні мережі не програмуються в звичному сенсі цього слова, вони навчаються. Можливість навчання – одна з головних переваг нейронних мереж перед традиційними алгоритмами. Технічно процес навчання полягає в знаходженні коефіцієнтів між зв'язками нейронів. В процесі навчання нейронна мережа здатна виявляти складні залежності між вхідними даними і вихідними, а також робити узагальнення [14]. Це означає, що в разі успішного навчання мережа зможе повернути вірний результат для даних, які були відсутні в навчальній вибірці, а також для неповних та спотворених даних.

## 2.2 Рекурентні нейроні мережі та LSTM

Рекурентна нейрона мережа – це тип штучної нейронної мережі, яка призначеної для розпізнавання послідовних шаблонів і прогнозування наступного елемента в послідовності на основі попередніх вхідних даних. На відміну від традиційних нейронних мереж прямого зв'язку, які обробляють інформацію лише в одному прямому напрямку, РНМ має цикл зворотного зв'язку, який дозволяє передавати інформацію від одного шару до наступного. Це робить РНМ особливо корисними для обробки даних часових рядів і природної мови, де важливі контекст і порядок [15]. Програмування поведінки рекурентних нейромереж здійснюється шляхом навчання на прикладах, що не потребує формального визначення цілей. Вони здатні ефективно діяти в умовах невизначеності, зокрема, вирішувати задачі керування поведінкою складних систем у швидкозмінному оточенні, адаптуватися, приймати оперативні рішення в системах ситуаційного управління тощо. Архітектура та здатність адаптуватись до умов оточення у РНМ нагадує нервову систему живих організмів. Її входам та виходам відповідають шари рецепторних та ефекторних нейронів або просто лінії передачі даних. Між цими шарами розташовано один або кілька шарів прихованих нейронів [16]. Входи нейронів кожного шару мають прямі зв'язки з виходами нейронів попереднього шару та можуть мати зворотні зв'язки з виходами нейронів свого та наступних шарів.

Для даного типу задачі підходить архітектура нейронної мережі ДКЧП (англ. LSTM). Це різновидність архітектури рекурентних нейронних мереж, запропонована 1997 року Зеппом Хохрайтером для вирішення проблеми зникнення градієнтів, які виникають під час навчання стандартної РНМ. На відміну від стандартних РНМ, які використовують лише один шар повторюваних одиниць, мережі LSTM мають кілька шарів комірок пам'яті, які можуть зберігати інформацію протягом тривалих періодів часу [17]. Ці комірки пам'яті призначені для вибіркового забування або запам'ятовування певних фрагментів інформації на

основі набору функцій, які визначають, які фрагменти інформації треба передати на наступний часовий крок.

Здатність вибірково запам’ятовувати або забувати інформацію робить LSTM добре придатними для таких додатків, як розпізнавання мови, машинний переклад і створення субтитрів до зображень. Крім того, LSTM можна використовувати в поєднанні зі згортковими нейронними мережами для створення високоточних прогнозів на основі складних часових рядів даних, таких як ті, що знаходяться на фондовому ринку або в кліматичних моделях. Деякі з переваг LSTM включають їхню здатність фіксувати довгострокові залежності в послідовних даних, їх високу гнучкість у обробці послідовностей змінної довжини та їхню ефективність у моделюванні складних нелінійних зв’язків між вхідними функціями та вихідними цілями [18]. Однак одним із головних недоліків LSTM є те, що їх навчання може бути дорогим з обчислювальної точки зору, особливо при роботі з великими наборами даних або глибокими архітектурами.

Структура одного блоку (вузлу) нейромережі ДКЧП зображена на рисунку 2.2.

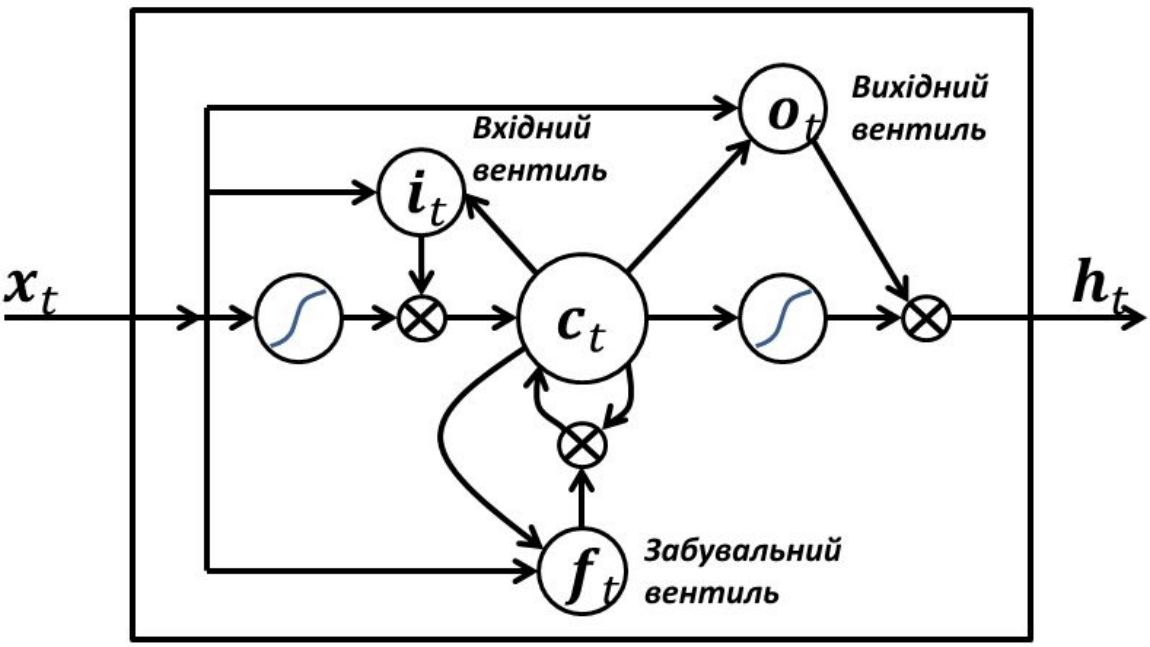


Рисунок 2.2 – Структура блоку ДКЧП

Вузол ДКЧП – це вузол рекурентної нейронної мережі, який виділяється запам'ятовуванням значень для довгих, і коротких проміжків часу. Ключем до цієї здатності є те, що він не використовує функції активації в межах своїх рекурентних складових. Таким чином, значення, що зберігається, не зменшується ітеративно з плином часу, і член градієнту не має схильності розмиватися, коли для його тренування застосовується зворотне поширення в часі.

У LSTM є три типи вентилів: вхідний, забутий та вихідний [19].

– Вхідні вентилі – вони визначають, яку інформацію з поточного введення та попереднього стану комірки слід додати до поточного стану комірки. Він відповідає за вибіркове оновлення стану клітини новою інформацією. Вхідним вентилем керує функція активації, яка виводить значення від 0 до 1.

– Забувальний вентиль – він визначає, яку інформацію з попереднього стану комірки слід відкинути. Він відповідає за вибіркове видалення інформації зі стану клітини, яка більше не актуальна. Ворота забуття також контролюються функцією активації, яка виводить значення від 0 до 1.

– Вихідний вентиль – він визначає, яка інформація з поточного стану комірки повинна бути виведена. Він відповідає за вибіркове виведення інформації зі стану комірки, яка є релевантною для поточного прогнозу. Вихідний вентиль контролюється сигмоподібною функцією та гіперболічною функцією тангенса.

Керуючи вхідними, забувальними та вихідними вентилями, LSTM може вибірково додавати, видаляти та оновлювати інформацію в комірці пам’яті, дозволяючи йому ефективно фіксувати довготривалі залежності в послідовних даних.

На додаток до цих трьох видів вентилів, деякі варіанти LSTM включають додаткові вентилі. Наприклад діагональний вентиль, який дозволяють дивитися на стан комірки безпосередньо. Інші варіанти LSTM включають зв’язаний вентиль вводу та пропуску, який об’єднує вентилі введення та пропускання в один вентиль, і мультиплікативний LSTM, який використовує мультиплікативну комбінацію вентилів замість адитивної комбінації. Існують також складніші варіанти LSTM,

наприклад деревоподібний LSTM, призначений для роботи з ієрархічними даними, і LSTM уваги, який використовує механізм уваги для вибіркового фокусування на різних частинах вхідної послідовності.

У LSTM вентилі взаємодіють один з одним складним чином. Вентилі забуття вирішують, яку інформацію викинути зі стану комірки, тоді як вхідні вентилів вирішують, яку інформацію оновити та додати до стану комірки. Вентилі забуття зазвичай негативно корелюють із вхідними воротами. Вхідних вентиль дозволяє додавати нову інформацію до стану комірки, вентиль забуття часто блокує запам’ятовування старої інформації. Подібним чином, вихідний вентиль часто позитивно корелює з вхідним вентилем. Тому коли нова інформація додається до стану комірки, вихідний вентиль часто надає цю інформацію наступному рівню. Ці взаємодії між воротами дозволяють LSTM вибірково запам’ятовувати або забувати інформацію на основі поточного введення та контролювати потік інформації через мережу [20].

LSTM продемонстрували значний успіх у різних програмах, таких як розпізнавання мови, обробка природної мови та створення підписів до зображень, серед іншого. Її здатність обробляти послідовні дані які мають довготривалі залежності зробила їх популярним засобом для вирішення багатьох задач машинного навчання.

Переваги LSTM над іншими типами рекурентних нейронних мереж включають в себе стійкість до проблем зникання та вибуху градієнту, а також нечутливість до прогалин у даних часового ряду [21]. Ці проблеми вирішуються шляхом використання комірок пам’яті, що дозволяють вибірково запам’ятовувати важливу інформацію або забувати минулі дані. Нейронні мережі з архітектурою LSTM можуть ефективно виокремлювати довгострокові часові залежності, не зазнаючи перешкод при оптимізації. Додатковий контекст прискорює процес тренування, а також покращує ефективність моделі при роботі з проблемою класифікації послідовностей.

На відміну від традиційних нейронних мереж (перцептронів), мережа LSTM добре підходить для аналізу тональності тексту, так як вона пам’ятає останні проаналізовані слова, а отже запам’ятовує контекст.

### 2.3 Навчання та явище перенавчання нейромережі

Існують три парадигми навчання: “з учителем”, “без вчителя” (самонавчання) і змішана. У першому випадку нейронна мережа має правильними відповідями (виходами мережі) на кожен вхідний приклад. Ваги налаштовуються так, щоб мережа виробляла відповіді як можна більш близькі до відомих правильних відповідей. Посилений варіант навчання з учителем передбачає, що відома тільки критична оцінка правильності виходу нейронної мережі, але не самі правильні значення виходу [5, 22]. Навчання без вчителя не вимагає знання правильних відповідей на кожен приклад навчальної вибірки. В цьому випадку розкривається внутрішня структура даних або кореляції між зразками в системі даних, що дозволяє розподілити зразки по категоріях. При змішаному навчанні частина ваг визначається за допомогою навчання з учителем, в той час як інша виходить за допомогою самонавчання.

Найвідоміший варіант алгоритму навчання нейронної мережі – алгоритм зворотного поширення (англ. back propagation). Цей алгоритм використовується для обчислення градієнта функції втрат нейромережі, який потім використовується для оновлення ваг під час навчання. Алгоритм працює, поширюючи помилку в зворотному напрямку через мережу, починаючи з вихідного рівня і повертаючись через приховані рівні до вхідного рівня. Алгоритм зворотного поширення включає дві основні фази: прямий перехід і зворотний перехід. У прямому проході вхідні дані подаються через мережу, а вихід обчислюється. У зворотному проході обчислюється похибка між прогнозованим виходом і фактичним виходом і обчислюються градієнти функції втрат щодо ваг. Потім ці градієнти використовуються для оновлення вагових коефіцієнтів мережі за допомогою

алгоритму оптимізації, такого як стохастичний градієнтний спуск. Алгоритм зворотного поширення помилки є потужним інструментом для навчання нейронних мереж, оскільки він дозволяє автоматично обчислювати градієнти, необхідні для оптимізації [23]. Це означає, що складні нейронні мережі з багатьма шарами та параметрами можна навчати без необхідності ручного розрахунку градієнтів. Однак він може бути чутливим до таких проблем, як зникнення градієнтів, коли градієнти стають дуже малими та спричиняють повільне навчання.

У машинному навчанні швидкість навчання – це гіперпараметр, який визначає розмір кроку, який виконує оптимізатор під час процесу оновлення ваг. Швидкість навчання впливає на те, наскільки швидко модель навчається та зближується до оптимального набору параметрів, які мінімізують функцію втрат.

Якщо швидкість навчання надто мала, модель потребує більше часу для зближення, а якщо швидкість навчання надто велика, модель може ніколи не збігатися і може коливатися навколо мінімуму [24]. У деяких випадках висока швидкість навчання може призвести до того, що модель перестрибне локальний мінімум та не досягне найкращого результату. Тому важливо вибрати правильну швидкість навчання.

Зазвичай алгоритм навчання видозмінюється таким чином, щоб включати доданок імпульсу. Цей член сприяє просуванню в фіксованому напрямку, тому якщо було зроблено кілька кроків в одному і тому ж напрямку, то алгоритм “збільшує швидкість”, що іноді дозволяє уникнути локального мінімуму, а також швидше проходити плоскі ділянки.

Таким чином, алгоритм діє ітеративно, і його кроки прийнято називати епохами. На кожній епосі на вхід мережі по черзі подаються всі навчальні спостереження, вихідні значення мережі порівнюються з цільовими значеннями і обчислюється помилка. Значення помилки, а також градієнту поверхні помилок використовується для коригування ваг, після чого всі дії повторюються [22, 25]. Початкова конфігурація мережі вибирається випадковим чином, і процес навчання припиняється або коли пройдено певну кількість епох, або коли помилка досягне

деякого певного рівня малості, або коли помилка перестане зменшуватися (користувач може сам вибрати необхідна умова зупинки).

Перенавчання – явище, коли побудована модель добре пояснює приклади з навчальної вибірки, але щодо погано працює на прикладах, які не брали участі в навчанні (на прикладах з тестової вибірки) [26]. Це пов'язано з тим, що при побудові моделі в процесі навчання в навчальній вибірці виявляються деякі випадкові закономірності, які відсутні в генеральної сукупності. Якщо візуалізувати явище перенавчання для двовимірного середовища, то отримуємо рис. 2.3.

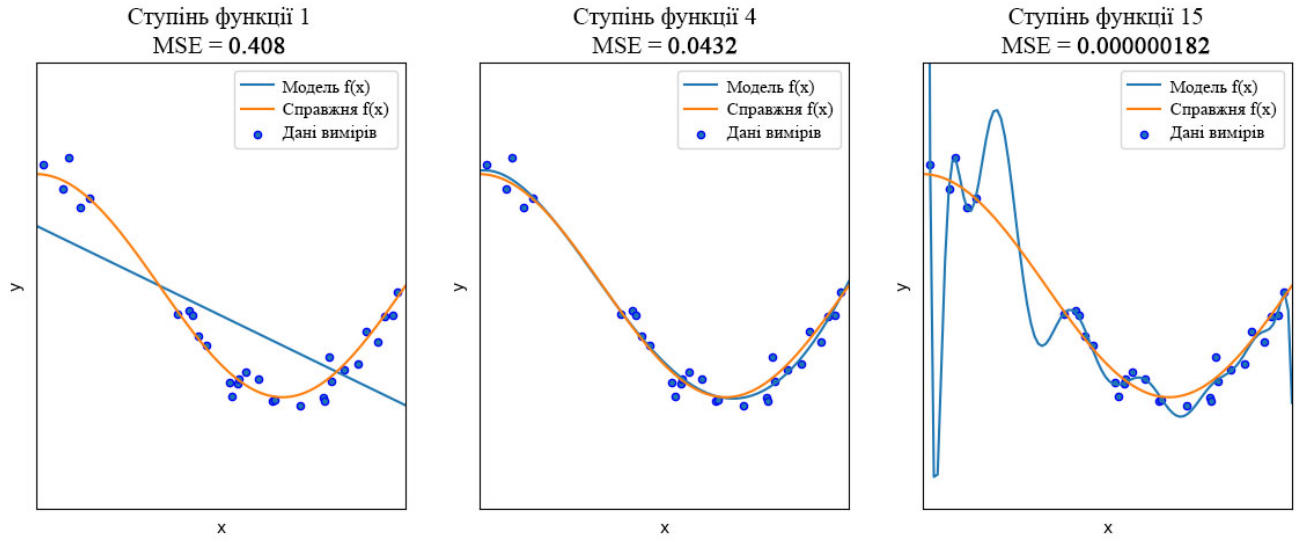


Рисунок 2.3 – Праворуч графік передбачень перенавченої моделі, ліворуч графік недонавченної моделі

Навіть тоді, коли навчена модель не має надмірної кількості параметрів, можна очікувати, що ефективність її на нових даних буде нижче, ніж на даних, що використовувалися для навчання. Зокрема, значення коефіцієнта детермінації буде скорочуватися в порівнянні з вихідними даними навчання. Способи боротьби з перенавчанням залежать від методу моделювання і способу побудови моделі. Наприклад, якщо будується дерево прийняття рішень, то можна обрізати деякі його гілки в процесі побудови.

Для запобігання перенавчення потрібно резервувати частину спостережень і не використовувати їх в навчанні за алгоритмом зворотного поширення [27, 28]. Замість цього, у міру роботи алгоритму, вони будуть використовуються для незалежного контролю результату. На самому початку роботи помилка мережі на навчальному та валідаційному датасеті буде однаковою, так як дійсно випадкове розбиття всіх спостережень на дві множини буде рівномірним для усіх класів. У міру того як мережа навчається, помилка навчання, природно, зменшується, і, поки навчання зменшує дійсну функцію помилок, помилка на валідаційному датасеті також буде спадати. Якщо ж помилка на валідаційному датасеті перестала зменшуватися або навіть стала рости, це вказує на те, що мережа почала занадто близько апроксимувати дані і навчання слід зупинити. Це явище надто точної апроксимації в процесі навчання і називається перенавчанням. Якщо таке трапилося, то зазвичай радять зменшити число прихованих елементів та шарів, бо мережа є занадто потужної для даного завдання. Якщо ж мережа, навпаки, була взята недостатньо багатою для того, щоб моделювати наявну залежність, то відбувається обернене явище – недонавчання [29]. При цьому помилки навчання на навчальному та валідаційному датасеті будуть приблизно однаково великими.

Тому для того щоб не допустити перенавчання нейромережі, потрібно слідкувати як зменшується змінюється похибки на валідаційному та навчальному датасеті [30]. Крім того, потрібно закінчити навчання коли похибка на валідаційному датасеті перестане зменшуватися. Після цього треба перевірити, чи є задовільною похибка на валідаційному датасеті.

## Висновки до розділу 2

За результатами даного розділу було досліджено та детально розглянуто моделі та алгоритми, які допоможуть розв’язати задачі, що поставлені перед системою, яка має бути результатом виконання МКР.

Для аналізу тональності тексту було використано рекурентну нейронну мережу, а саме структуру під назвою LSTM яка має довгу короткочасну пам’ять. Цей вибір було аргументовано здатністю цих нейронних мереж пам’ятати останні проаналізовані слова, а отже запам’ятовує контекст. Було детально описано можливу проблему перенавчання та описано алгоритм по запобіганню цього явища.

# 3 МОДЕЛЮВАННЯ ТА ТЕХНІЧНЕ ПРОЕКТУВАННЯ

## 3.1 Вибір датасету та його попередня обробка

Спочатку потрібно було обрати датасет на якому буде проходити навчання. Правильний датасет є дуже важливою частиною навчання нейромережі. Він повинен:

– мати достатню кількість даних;

– ці дані повинні бути актуальними;

– датасет повинен бути збалансованим, тобто мати приблизно однакову кількість даних у кожному класі;

– бути зібраним з достовірного джерела.

Крім того, датасет потрібно розділити на декілька частин. Навчальна вибірка (train) – це вибірка, за якої відбувається навчання, тобто оптимізація параметрів моделі. Кожна ітерація навчання буде обробляти цей датасет, або певну його частину. Ця частина датасету повинна бути найбільшою [25, 31]. Дуже важливо щоб вона підходила під вимоги які описані вище, так як саме ця частина датасету буде відповідати за точність моделі.

Коли модель залежності побудована за навчальною вибіркою, то оцінка якості цієї моделі, зроблена по тій же вибірці виявляється, як правило, оптимістично зміщеною. Це небажане явище називають перенавчанням та було описано у другому розділі. На практиці воно зустрічається дуже часто. Хорошу емпіричну оцінку якості побудованої моделі дає її перевірка на незалежних даних, які не використовувалися для навчання. Тому як було описано у ньому, для уникнення явище перенавчання потрібно перевіряти нейромережу на іншій частині даних, яку називають валідаційною. Ця частина датасету є значно меншою ніж навчальна вибірка, проте також повинна підходити під вимоги до датасету, які описані вище.

Тестова або контрольна вибірка (test) – це вибірка, за якої оцінюється якість побудованої моделі. Якщо навчальна та валідаційна вибірка подаються на вхід моделі багаторазово, то дуже важливо щоб тестову вибірку мережа бачила як можна меншу кількість разів [32]. Саме на основі цієї вибірки буде оцінюватися остаточна точність нейромережі.

Для вирішення задачі аналізу коментарів було обрано досить відомий датасет [33], який підходить під усі описані вище критерії. Сам датасет складається з 700000 відгуків, які були взяті з відомого американського сайту yelp.com. Крім тексту, кожен коментар має оцінку тональності від 1 до 5.

Попередню обробку та навчання відбувалося у Jupyter Notebook [34]. Це інтерактивне обчислювальне середовище, яке дозволяє користувачам писати, запускати та ділитися кодом, а також відображати та маніпулювати даними в інтерактивному та спільному режимі. Jupyter Notebook підтримує різноманітні мови програмування, включаючи Python. Однією з ключових переваг Jupyter Notebook є його здатність інтерактивне обчислювати кожен крок. Крім того, він має потужні вбудовані можливості для візуалізації даних.

Спочатку у середовище Jupyter було завантажено англійський словник з використанням бібліотеки en_core_web_sm (див. рис. 3.1).

```
[2]: DATASET_URL = 'http://hidra.lbd.dcc.ufmg.br/datasets/yelp_2015/original/yelp_review_full_csv.tar.gz'
     DATASET_ARCHIVE_NAME = 'dataset.tar.gz'
     DATASET_FOLDER_NAME = 'dataset'

     ENGLISH = en_core_web_sm.load()
```

Рисунок 3.1 – Завантаження словника англійської мови

Після цього було завантажено, розпаковано з архіву та збережено на жорсткий диск датасет за допомогою команд на рисунку 3.2.

```
[4]: %%time
     # Download dataset
     def download_file(url: str, filename: str, chunk_size: int = 2 ** 15):
         with requests.get(url, stream=True) as request:
             request.raise_for_status()
             with open(filename, 'wb') as file:
                 for chunk in request.iter_content(chunk_size=chunk_size):
                     file.write(chunk)


     def extract_archive(archive_name: str, folder_name: str):
         with tarfile.open(archive_name) as tar:
             tar.extractall(path=folder_name)


     download_file(DATASET_URL, DATASET_ARCHIVE_NAME)
     extract_archive(DATASET_ARCHIVE_NAME, DATASET_FOLDER_NAME)

     CPU times: user 5.83 s, sys: 1.52 s, total: 7.35 s
     Wall time: 36.8 s
```

Рисунок 3.2 – Завантаження та розпакування датасету

Архів цього датасету складався з двох частин – навчальної та валідаційної (у папці train) та тестової (папка test). Перша папка далі буде розділена на навчальну та валідаційну вибірку для того щоб уникнути перенавчання. Тестова частина буде використовуватися після навчання для того щоб остаточно перевірити нейромережу на незалежних даних. Обидві частини будо завантажено у Jupyter за допомогою бібліотеки pandas. На рисунку 3.3 можна побачити, перша частина містить 650000 коментарів, а друга (тестова) – 50000 коментарів.

```
[5]: %%time
     # Read dataset
     test = pd.read_csv(f'{DATASET_FOLDER_NAME}/yelp_review_full_csv/test.csv', header=None)
     train = pd.read_csv(f'{DATASET_FOLDER_NAME}/yelp_review_full_csv/train.csv', header=None)

     test = test.rename(columns={0: 'label', 1: 'review'})
     train = train.rename(columns={0: 'label', 1: 'review'})

     print(len(test), len(train))
     train.head()

     50000 650000
     CPU times: user 3.79 s, sys: 420 ms, total: 4.21 s
     Wall time: 4.2 s
```

Рисунок 3.3 Завантаження датасету у код

Після цього вхідні данні було очищено та токенізовано. Очищення даних – це самий базовий етап обробки [35]. У ньому прибирають усі не буквені символи з кожної частини тексту. Токенізація – це процес розбиття тексту на менші частини,

які називають токенами. У цьому випадку токенами будуть окремі слова. Тому далі кожен коментар розділено на окремі слова (див. рис. 3.4). Для виконання токенізації використовується словник англійської мови який був завантажений раніше. Він використовується для того щоб привести усі слова основної форми. Цей процес називається стемінгом (англ. stemming) та являється важливою частиною нормалізації тексту [36]. Так як не існує ідеального алгоритму пошуку основної форми, то було використано завантажений словник англійської мови який містить слова та різни їх форми. Цей метод стемінгу називається пошуком за таблицею. Перевагами цього методу є простота, швидкість та зручність обробки винятків для кожної мови [37]. До недоліків слід віднести те, що таблиця пошуку має містити в собі всі форми слів, а це позначає що алгоритм не буде працювати з новими словами.

```
[6]: %%time
# Tokenize in order to clean text
def tokenize(text: str) -> list:
    text = re.sub(r'[^\x00-\x7F]+', ' ', text)
    regex = re.compile('[' + re.escape(string.punctuation) + '0-9\\r\\t\\n]')
    words = regex.sub(' ', text.lower())
    words = re.sub(r'\s+', ' ', words.strip(), flags=re.UNICODE)
    return [token.text for token in ENGLISH.tokenizer(words)]


counts = Counter()
for index, row in train.iterrows():
    counts.update(tokenize(row['description']))
```

Рисунок 3.4 – Процес токенізації

Після цього описується спосіб перетворення кожного слова у вектор (набір чисел). За допомогою цієї функції кодується навчальний датасет. Цей метод називається word2vec, що можна перекласти як “перетворення слів у вектор”. Цей алгоритм приймає великий текстовий корпус в якості вхідних даних і зіставляє кожному слову вектор, видаючи координати слів на виході [38]. Отримані векторні уявлення слів можуть бути використані для обробки природної мови та машинного навчання. У цьому випадку було використано самий простий спосіб перетворення

слів у вектор, коли кожному слову зіставляється свій власний вектор, який не залежить від інших слів.

Для даної задачі словник містив 2000 слів які найчастіше зустрічалися серед усіх коментарів. Усі слова які зустрічаються рідше замінялися на ключ “UNK”. Під час аналізу тексту великі набори даних можуть містити тисячі або навіть мільйони унікальних слів, багато з яких є нерелевантними або не додають суттєвої інформації для аналізу [35]. Під час побудови моделі важливо звести кількість параметрів до мінімуму, оскільки кожен новий параметр ускладнює модель і збільшує ризик перенавчання. І навпаки, обмеживши кількість параметрів можна легше і швидше навчати модель. Це також може зменшити обчислювальні витрати на навчання моделі, а також зменшити пам’ять, необхідну для зберігання матриці ознак. Це особливо важливо під час навчання нейромережі на GPU у Google Colab так як у ньому існують ліміти на максимальний час використання обчислювальних ресурсів. Код методу векторизації показано на рисунку 3.5.

```
[9]: %%time
# Encode reviews to array of int
def encode_sentence(text: str, word2vec: dict, size: int = 150):
    tokenized = tokenize(text)
    encoded = np.zeros(size, dtype=int)
    enc1 = np.array([word2vec.get(word, word2vec['UNK']) for word in tokenized])
    length = min(size, len(enc1))
    encoded[:length] = enc1[:length]
    return encoded, length


train['encoded'] = train['review'].apply(lambda x: np.array(encode_sentence(x, word2vec), dtype=object))
```

Рисунок 3.5 – Перетворення слів у вектори

Так як нумерація оцінки коментарів у датасеті починається с 1, а нейромережа нумерує класи від 0, то потрібно зменшити значення кожної оцінки на один. Крім того, треба перевірити щоб кількість елементів у кожному класі була приблизно однаковою, інакше це може призвести до раннього перенавчання коли нейромережа спеціально обирає більш популярний клас. Для цього для кожного класу була виведена кількість коментарів у ньому. І дійсно, на рис 3.6 можна побачити що в обраному датасеті кількість коментарів у кожному класі майже

однакова. Максимальну кількість коментарів має перший клас – 129998, а мінімальну має нульовий клас – 129987 коментарів. Абсолютна різниця складає 11 коментарів, що менше ніж 0.01% від кількості коментарів у кожному класу. Тобто датасет є ідеально збалансованим.

```
[11]: # Transform labels from 1..5 to 0..4
      zero_numbering = {1: 0, 2: 1, 3: 2, 4: 3, 5: 4}
      train['label'] = train['label'].apply(lambda x: zero_numbering[x])
      Counter(train['label'])

[11]: Counter({0: 129987, 1: 129998, 2: 129998, 3: 129998, 4: 129992})
```

Рисунок 3.6 – Кількість коментарів у кожному класі

Далі першу частину датасету було розділено на навчальний та валідаційний датасети. Тут було обране стандартне відношення 80% на 20%, де менша частина піде на валідацію. Для рівномірного випадкового розділу було використано вбудовано функцію (див. рис. 3.7).

```
[12]: # Split into train and validation subsets
      x = list(train['encoded'])
      y = list(train['label'])
      x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.2)
```

Рисунок 3.7 – Розділ на навчальний та валідаційний датасети

### 3.2 Метрики оцінки нейромережі

Для того щоб оцінити якість та похибку нейромережі використовуються різні функції оцінки, які називаються метриками. Їх існує багато, кожна метрика оцінює нейромережу з певної сторони, враховуючи певні критерії.

У даній роботі для оцінки нейромережі було використано три метрики:

1) Логістична функція втрат – метрика оцінки класифікатора, що дозволяє оцінювати ймовірності приналежності об'єктів класів. Цей метод рекомендовано використовувати при навчанні для задачі класифікації [39]. Крім того, він

ефективно працює навіть при незбалансованому навчальному наборі даних. Цю метрику можна задати формулою:

$$loss = -\sum_{c=0}^{M}\sum_{i=0}^{N} y_i \log(\overline{y_i}), \quad (3.1)$$

де $loss$ – значення логістичної функції втрат;

$M$ – кількість класів;

$c$ – індекс класу;

$N$ – загальна кількість об’єктів;

$i$ – індекс об’єкту;

$y_i$ – 1 якщо $i$-й об’єкт належить класу $c$, інакше 0;

$\overline{y_i}$ – передбачена ймовірність належності $i$-го об’єкту класу $c$.

2) Точність (accuracy) – відношення кількості правильних відповідей моделі до кількості усіх даних. Досить проста метрика оцінки яка є інтуїтивно зрозуміло. Проте в задачах класифікації вона вимагає того, щоб кількість елементів кожного класу буда приблизно однаковою. Крім того, ця метрика не враховую масштаб похибки, їй вважливо лише чи є передбачення моделі вірним.

Цю метрику можна задати формулою:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \quad (3.2)$$

де $accuracy$ – значення точності;

$TP$ – кількість істинно-позитивних результатів (true positive);

$TN$ – кількість істинно-негативних результатів (true negative);

$FP$ – кількість хибно-позитивних результатів (false positive);

$FN$ – кількість хибно-негативний результатів (false negative).

3) Середньоквадратична помилка (Root Mean Square Error, скорочено RMSE) – показник різниці між середніми значеннями передбачених моделлю, та

дійсними значеннями. Він розраховується як квадратний корінь із середнього значення квадрату помилок.

Вплив кожної помилки на RMSE пропорційний до найбільшої квадратичної помилки [40]. Таким чином, більші помилки мають непропорційно великий вплив на RMSE, тому цей метод є чутливий до викидів. Значення RMSE завжди невід'ємне, а значення 0 позначає повне відсутність похибки, що майже ніколи не досягається на практиці. Отже, чим нижче значення цього параметру, тим краще навчена нейромережа. На відміну від метрики точності, цей метод враховує масштаб похибки.

Цю метрику можна задати формулою:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\overline{y_i}-y_i)^2}{n}}, \quad (3.3)$$

де $RMSE$ – значення середньоквадратичної помилки;

$n$ – кількість усіх елементів;

$\overline{y_i}$ – передбачений клас;

$y_i$ – вірний клас.

### 3.3 Структура та навчання нейромережі

Як було писано у другому розділі, у роботі було використано РНМ, а саме нейромережу із структурою LSTM [17, 18]. Створення та навчання нейромережі відбувалось у середовищу Jupyter Notebook. Для створення та роботи з нейромережою було обрано фреймворк Torch [41], який іноді називають PyTorch. Це фреймворк машинного навчання з відкритим кодом для Python, розроблений командою дослідження штучного інтелекту з компанії Facebook (FAIR). Він широко використовується для розробки та навчання нейронних мереж та інших моделей машинного навчання.

Torch відомий своїм динамічним обчислювальним графом, який забезпечує більшу гнучкість і легше налагодження порівняно з іншими фреймворками, такими як TensorFlow [42]. Це означає, що ви можете змінювати структуру нейронної мережі на льоту, а фреймворк автоматично обчислюватиме градієнти та оновлюватиме ваги. Однією з його найбільших переваг є простота використання. Він має простий та інтуїтивно зрозумілий API, і його відносно легко освоїти порівняно з іншими фреймворками глибокого навчання. Torch також пропонує хорошу продуктивність і може працювати як на процесорах, так і на графічних процесорах, що робить його придатним для широкого спектру програм.

Крім того він пропонує ряд готових модулів і утиліт, які допомагають пришвидшити процес розробки, включаючи попередньо навчені моделі, завантажувачі даних і функції втрати. Крім того, PyTorch має зростаючу спільноту розробників, які роблять свій внесок у структуру та надають підтримку через форуми та навчальні посібники. У тому числі до бібліотеки готових моделей входить одна з реалізацій LSTM.

Значною перевагою бібліотеки Torch є підтримка відеокарт (GPU) для навчання нейромереж. Саме це прискорює навчання у десятки разів, так як задача навчання нейромереж підходить для паралельного виконання на GPU. На даний момент підтримується лише програмно-апаратна архітектура CUDA, яка використовується на відеокартах від фірми Nvidia. В основі інтерфейсу програмування додатків CUDA лежить мова C з деякими розширеннями. CUDA широко використовується в різних областях, особливу у машинному навчанні [43]. Для перевірки того, чи підтримує фреймворк Torch відеокарту на певному пристрої, існує спеціальна функція. Цю функцію (див. рис. 3.8) потрібно викликати перед початком використання бібліотеки для навчання.

```
[14]: # Check cuda device (GPU)
      device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
      torch.cuda.is_available()

[14]: True
```

Рисунок 3.8 – Перевірка підтримки GPU на архітектурі CUDA

Була створена наступна нейромережа: на вхід вона приймає вектор, у якому закодовано текст коментаря. На вихід вона має п’ять ознак – по одній для кожної оцінки коментаря, ці виходи пронумеровано від 0 до 4. Вихід який набрав найбільшу кількість балів і є найбільш ймовірним номером класу. Після декількох ітерацій навчання було додано метод dropout. Це метод регуляризації, який використовується в нейронних мережах для запобігання перенавчанню [12]. Він працює шляхом випадкового вилучення (заміни на нуль) певного відсотка зв’язків нейромережі під час навчання. Це змушує мережу створювати надлишкові правильні зв’язки, роблячи її більш надійною та зменшує ймовірність перенавчання. Значення відсіву є гіперпараметром та визначається шляхом експерименту. Для даної нейромережі було обрано значення 0.2, тобто 20%.

Бібліотека Torch містить інтерфейс для роботи с нейромережами із LSTM структурою. Тому було налаштовано цей інтерфейс (див. рис 3.9) для того щоб він правильно працював із вхідними даними, мав dropout та п’ять вихідних класів які були описані вище.

[16]:
```python
# Create LSTM net class
class CustomLSTM(torch.nn.Module):
    def __init__(self, vocab_size: int, embedding_dim: int, hidden_dim: int):
        super().__init__()
        self.hidden_dim = hidden_dim
        self.embeddings = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
        self.dropout = nn.Dropout(0.2)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.linear = nn.Linear(hidden_dim, 5)
        # initialize the hidden state (see code below)
        self.hidden_dim = self.init_hidden()

    def init_hidden(self):
        """At the start of training, we need to initialize a hidden state
        there will be none because the hidden state is formed based on previously seen data
        So, this function defines a hidden state with all zeroes and of a specified size"""
        # The axes dimensions are (n_layers, batch_size, hidden_dim)
        return torch.zeros(1, 1, self.hidden_dim).to(device), \
               torch.zeros(1, 1, self.hidden_dim).to(device)

    def forward(self, x, s):
        x = self.embeddings(x)
        x = self.dropout(x)
        x_pack = pack_padded_sequence(x, s, batch_first=True, enforce_sorted=False)
        out_pack, (ht, ct) = self.lstm(x_pack)
        out = self.linear(ht[-1])
        return out
```

Рисунок 3.9 – Створення LSTM нейромережі

Для навчання нейромережі було описано функцію навчання (рис. 3.10) та функцію валідації (рис 3.11). Увесь процес навчання було розбито на епохи. У кожній епохі відбувається розбиття навчального датасету на групи по 2000 коментарів, та після аналізу кожної групи відбувається навчання – зміна ваг зв'язків у нейромережі. Так як навчальна вибірка містить 80% від 650'000 коментарів, то усього таких груп буде $\frac{0.8 \cdot 650000}{2000} = 260$.

```
[15]: # Create train and validation functions
def train_model(model: torch.nn.Module, epochs: int = 10, lr: float = 0.001):
    since = time.time()
    ep_time = time.time()
    parameters = filter(lambda p: p.requires_grad, model.parameters())
    optimizer = torch.optim.Adam(parameters, lr=lr)
    for i in range(epochs):
        model.train()
        sum_loss = 0.0
        total = 0
        for x, y, l in train_dataloader:
            x = x.long().to(device)
            y = y.long().to(device)
            y_pred = model(x, l)
            optimizer.zero_grad()
            loss = functional.cross_entropy(y_pred, y)
            loss.backward()
            optimizer.step()
            # print('train loss %.3f' % (loss))
            time_elapsed = time.time() - ep_time
            ep_time = time.time()
            # Time spent for train/eval
            # print(f'Complete in {time_elapsed // 60}m {time_elapsed % 60}s')
            sum_loss += loss.item() * y.shape[0]
            total += y.shape[0]
        val_loss, val_acc, val_rmse = validation_metrics(model, valid_dataloader)
        print('train loss %.3f, val loss %.3f, val accuracy %.3f, and val rmse %.3f' % (sum_loss / total, val_loss, val_acc, val_rmse))
```

Рисунок 3.10 – Функція навчання нейромережі

Наприкінці кожної епохи відбувається валідація. Під час валідації розраховуються усі три метрики – логістична функція втрат для навчальної та валідаційної вибірки, точність для валідаційної вибірки та середньоквадратична помилка для валідаційної вибірки.

```python
def validation_metrics(model: torch.nn.Module, dataloader: DataLoader):
    model.eval()
    correct = 0
    total = 0
    sum_loss = 0.0
    sum_rmse = 0.0
    for x, y, l in dataloader:
        x = x.long().to(device)
        y = y.long().to(device)
        y_hat = model(x, l)
        loss = functional.cross_entropy(y_hat, y)
        pred = torch.max(y_hat, 1)[1].cpu()
        correct += (pred == y.cpu()).float().sum().cpu()
        total += y.shape[0]
        sum_loss += loss.item() * y.shape[0]
        sum_rmse += np.sqrt(mean_squared_error(pred, y.cpu().unsqueeze(-1))) * y.shape[0]
    return sum_loss / total, correct / total, sum_rmse / total
```

Рисунок 3.11 – Функція валідації нейромережі

Після того як описана структура нейромережі, задано функції навчання та валідації, потрібно об’єднати усі частини (див. рис. 3.12). Як було вказано вище, навчальний датасет оброблюється групами по 2000 коментарів для зручності роботи з GPU. Спочатку було задано 10 епох, проте методом перебору було вирішено збільшити їх кількість до 20. Ще більша кількість епох давала перенавчання, а менша призводила до недонавчання нейромережі.

[17]:
```python
# Create dataloaders
batch_size = 2000
vocab_size = len(words)
train_dataloader = DataLoader(train_ds, batch_size=batch_size, shuffle=True)
valid_dataloader = DataLoader(valid_ds, batch_size=batch_size)
```

[18]:
```python
# Create model
model = CustomLSTM(vocab_size, 100, 100)
model.to(device)
```

[18]:
```
CustomLSTM(
  (embeddings): Embedding(2002, 100, padding_idx=0)
  (dropout): Dropout(p=0.2, inplace=False)
  (lstm): LSTM(100, 100, batch_first=True)
  (linear): Linear(in_features=100, out_features=5, bias=True)
)
```

[19]:
```python
# Train model
train_model(model, epochs=20, lr=0.005)
```

```
train loss 1.240, val loss 1.025, val accuracy 0.455, and val rmse 0.924
train loss 0.991, val loss 0.956, val accuracy 0.583, and val rmse 0.896
```

Рисунок 3.12 – Навчання нейромережі

Зміна значень метрик похибок під час навчання для 20 епох наведено у таблиці 3.1, а візуалізація цих даних на рисунку 3.13.

Таблиця 3.1 – Зміна значень метрик під час навчання

| **Номер епохи** | **Train loss** | **Validation loss** | **Validation accuracy** | **Validation RMSE** |
|---|---|---|---|---|
| 1 | 1.24 | 1.025 | 0.555 | 0.924 |
| 2 | 0.92163 | 0.88908 | 0.62688172 | 0.83328 |
| 3 | 0.8366 | 0.82948 | 0.66741573 | 0.77341 |
| 4 | 0.7786 | 0.7752 | 0.708235294 | 0.72505 |
| 5 | 0.74617 | 0.75613 | 0.726506024 | 0.70633 |
| 6 | 0.72209 | 0.73187 | 0.747239264 | 0.678895 |
| 7 | 0.711312 | 0.724304 | 0.751231527 | 0.67396 |
| 8 | 0.703137 | 0.725845 | 0.75215783 | 0.649611 |
| 9 | 0.69579 | 0.71847 | 0.756790123 | 0.67068 |
| 10 | 0.690077 | 0.72001 | 0.757725587 | 0.666616 |
| 11 | 0.684376 | 0.719928 | 0.758663366 | 0.668216 |
| 12 | 0.680301 | 0.716616 | 0.763320942 | 0.656091 |
| 13 | 0.674622 | 0.71331 | 0.76426799 | 0.653666 |
| 14 | 0.670565 | 0.714035 | 0.765217391 | 0.651245 |
| 15 | 0.666516 | 0.712344 | 0.764925373 | 0.654456 |
| 16 | 0.662475 | 0.712261 | 0.765877958 | 0.656051 |
| 17 | 0.658442 | 0.707364 | 0.7680798 | 0.643204 |
| 18 | 0.654417 | 0.708084 | 0.769038702 | 0.644004 |
| 19 | 0.6512 | 0.7096 | 0.76991003 | 0.644 |
| 20 | 0.6496 | 0.7088 | 0.7700102 | 0.6439 |

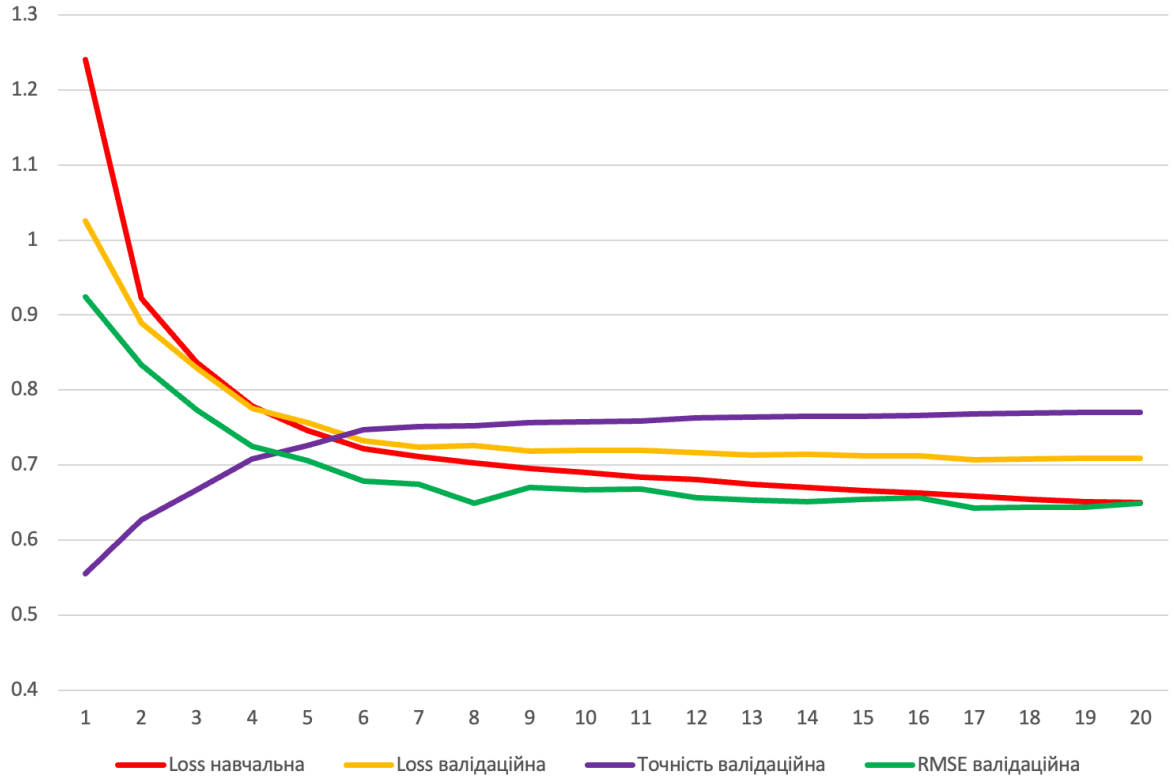


Рисунок 3.13 – Графік зміни значень метрик при навчанні

Навчання та остаточна перевірка нейромережі відбувалася на ресурсах платформи Google Colab. Google Colab [44] – це безкоштовний хмарний сервіс на основі Jupyter Notebook. Він надає все необхідне для машинного навчання прямо в браузері, дає безкоштовний доступ до CPU та неймовірно швидких GPU та TPU.

1) CPU – центральний процесор комп'ютера, який виконує операції з даними. Може швидко виконувати надскладні операції, проте має обмеження за кількість ядер. Зазвичай кількість ядер не більша ніж декілька десятків. Отже, він підходить для важких однопотокових операцій.

2) GPU – графічний процесор [45]. Обробляє дані швидше, так як завдання виконує паралельно, а не послідовно, як CPU. Він заточений виключно під графіком, тому на ньому зручніше працювати з зображенням і відео. Ідеально підходить під задачі які можна розбити на велику кількість паралельних задач. А так як сучасні фреймворки нейромереж дозволяють розбивати навчання на велику кількість паралельних процесів, то GPU часто використовують для навчання нейромереж.

3) TPU – тензорний процесор, нещодавня розробка Google. Він призначений для тренування нейромереж. У цього процесора в рази вища продуктивність при великих обсягах обчислювальних задач. Проте для ефективного використання тензорного процесору потрібно писати на спеціальних фреймворках у певному стилі. Крім того, час роботи TPU коштує набагато більше ніж час на CPU та GPU.

За допомогою Google Colab можна легко навчити моделі за лічені секунди. Він підтримує Python з коробки. Проте він має певні обмеження, а саме на час використання. Особливо обмежений час використання GPU та ще більше – час використання TPU. Так як TPU у Google Colab накладає багато обмежень, то було вирішено навчати нейромережу на GPU.

Ще одним приємним фактом є інтеграція Google Colab із Google Drive, що дозволяє відразу зберігати файли на Google Drive [45]. Ця інтеграція активно використовувалася при тюнінгу нейромережі для зберігання проміжних файлів на Google Drive.

Віртуальна машина Google Colab, де відбувалось навчання нейромережі мала наступні характеристики:

– GPU Tesla K80;
– CPU Xeon 2.3 Ghz;
– 32 ГБ оперативної пам’яті (рис 3.14);
– 108 ГБ SSD, з них 70 ГБ вільно (рис 3.15);
– стабільне швидкісне інтернет з’єднання 100 МБ/с.

```
colab > free -h
              total        used        free      shared  buff/cache   available
Mem:            12G        538M         10G        1.1M        2.1G         11G
Swap:            0B          0B          0B
colab >
```

Рисунок 3.14 – Обсяг оперативної пам’яті віртуальної машини Google Colab

```
colab > df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay         108G   39G   70G  36% /
tmpfs            64M     0   64M   0% /dev
tmpfs           6.4G     0  6.4G   0% /sys/fs/cgroup
shm             5.9G     0  5.9G   0% /dev/shm
tmpfs           6.4G   36K  6.4G   1% /var/colab
/dev/sda1       114G   40G   75G  35% /etc/hosts
tmpfs           6.4G     0  6.4G   0% /proc/acpi
tmpfs           6.4G     0  6.4G   0% /proc/scsi
tmpfs           6.4G     0  6.4G   0% /sys/firmware
```

Рисунок 3.15 – Обсяг диску віртуальної машини Google Colab

### 3.4 Перевірка нейромережі на тестовому датасеті

Для фінальної перевірки нейромережі було використано тестовий датасет. У попередньому пункті було описано процес його створення. Він містить 50’000 коментарів та їх оцінки тональності. До цього моменту тестовий датасет жодного разу не подавалася на вхід до нейромережі. Отже він є підходящим набором для кінцевої оцінки нейромережі.

Так само як і усі інші, тестовий датасет було токенізовано та перетворено на вектор (рис. 3.16). Після цього нормалізований вектор було подано на вхід нейромережі для оцінки (рис. 3.17), а її відповідь була зіставлена із реальними результатами в датасеті. У результаті точність склала 76.3%. Значення метрики логістична функція втрат та середньоквадратична помилка склали 0.7061 та 0.6478 відповідно.

```
test['encoded'] = test['review'].apply(
    lambda x: np.array(
        encode_sentence(x, word2vec),
        dtype=object
    )
)
test['label'] = test['label'].apply(lambda x: zero_numbering[x])
```

Рисунок 3.16 – Підготовка тестового датасету

```
# Create dataloader for test dataset
x_test = list(test['encoded'])
y_test = list(test['label'])
test_dataset = ReviewsDataset(x_test, y_test)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size)
```

```
# Evaluate model on test dataset
test_loss, test_acc, test_rmse = validation_metrics(model, test_dataloader)
print('test loss %.3f, test accuracy %.3f, and test rmse %.3f' % (test_loss, test_acc, test_rmse))
```

```
test loss 0.885, test accuracy 0.715, and test rmse 0.811
```

Рисунок 3.17 – Оцінка нейромережі на тестовому датасеті

## Висновки до розділу 3

За результатами даного розділу було створено, навчено та оцінено нейромережу для аналізу тональності коментарів. Було описано вимоги до датасету та обрано відповідний набір даних [33]. Його було розділено на три частини: навчальна частина датасету містила 520000 коментарів, валідаційна – 130000, а тестова – 50000. Після завантаження цей датасет було нормалізовано та перетворено у вектор.

Для створення та навчання нейромережі було використано мову Python, середовище Jupyter Notebook та фреймворк Torch. Навчання нейромережі відбувалося на ресурсах платформи Google Colab. Для того щоб навчання було швидшим було обрано віртуально машину із GPU. За результатами тестів, найкращий результат мала нейромережа після 20 епох навчання.

Для оцінки ефективності роботи нейромережі було обрано три метрики – логістична функція втрат, точність та середньоквадратична помилка. Значення похибок, які отримані при перевірці на тестовому датасеті, дуже близькі близькі до значень на похибок навчальному. Отже, можна зробити висновок, що нейромережа дійсно навчилась аналізувати тональність коментарів по шкалі від 1 до 5. Порівняти значення метрик на обох датасетах можна на таблиці 3.2.

Таблиця 3.2 – Значення похибок на навчальному та тестовому датасетах

| **Тип датасету** | **Логістична функція втрат** | **Точність** | **Середньоквадратична помилка** |
|---|---|---|---|
| Валідаційний (при навчанні) | 0.7088 | 0.77 | 0.6439 |
| Тестовий | 0.7061 | 0.763 | 0.6478 |
| Різниця у відсотках | 0.4% | 0.9% | 0.6% |

Отримана структура нейромережі та її параметри були збережені у файл для подальшого використання у системі пошуку спаму.

# 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ ЗАДАЧІ ПОШУКУ СПАМУ ТА ЇХ АНАЛІЗ

## 4.1 Використання створеної нейромережі для пошуку спаму за допомогою наївного байєсівського класифікатора

Для розв’язку задачі пошуку спаму було обрано досить відомий датасет [46] із коментарями із сайту youtube.com. Сам датасет складається з 1961 відгуку, кожен с яких відноситься до класу спам або не спам (ham). Обидва класи представлені у датасеті в однаковій пропорції (див. рис. 4.1).

```
labels = ['Spam (спам)', 'Ham (не спам)']
sizes = data['Category'].rename('Кількість').value_counts()

plt.figure(figsize=(10, 6), dpi=300)
plt.subplot(1, 2, 1)
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=['#ff7675', '#74b9ff'])
plt.subplot(1, 2, 2)
sns.barplot(x=labels, y=sizes, palette='viridis')

plt.show()
```

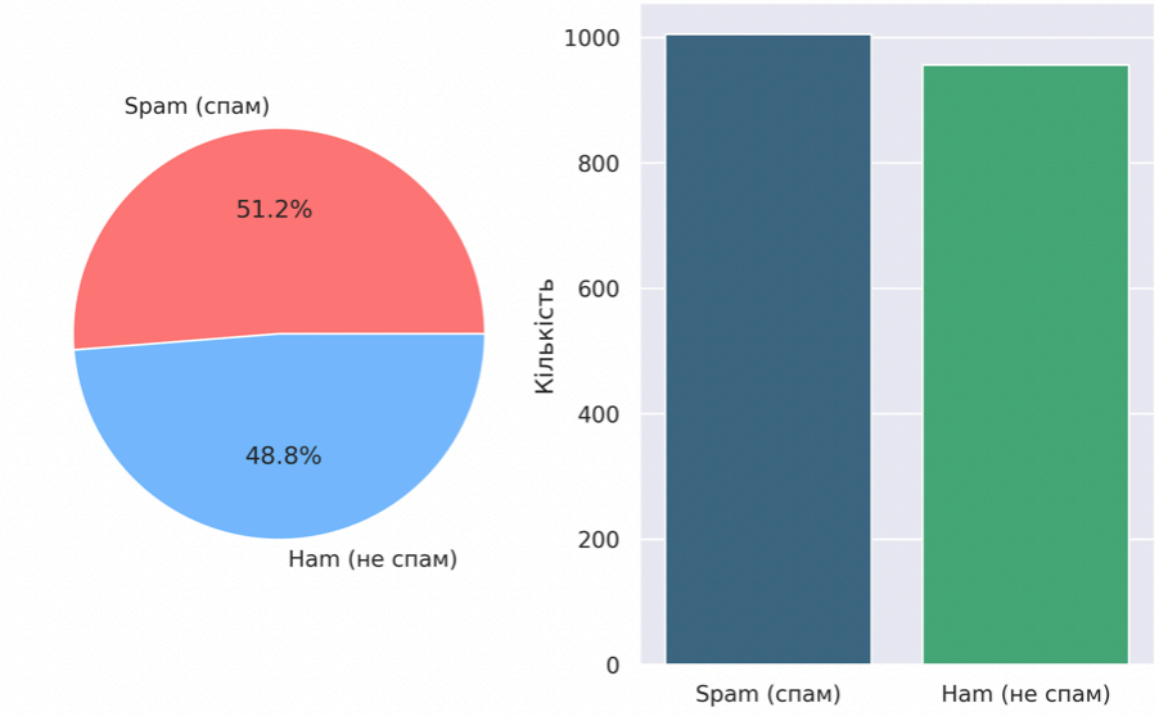


Рисунок 4.1 – Розподіл даних у датасеті

Попередню обробку та навчання було проведено у середовищі Jupyter. Текст кожного відгуку цього датасету було очищено, токенізовано та перетворено на вектор як описано у третьому розділі, тобто так само як було оброблено датасет на якому навчалася нейромережа для аналізу тональності. Крім того, із тексту було обрано усі Інтернет-посилання. Очищення даних – це самий базовий етап обробки [35]. У ньому прибирають усі не буквені символи з кожної частини тексту. Токенізація – це процес розбиття тексту на менші частини, які називають токенами. У цьому випадку токенами будуть окремі слова. Для виконання токенізації використовується функція TfidfVectorizer з бібліотеки sklearn. Вона аналізує кількість повторень кожного слова та залишає лише набір з найпопулярніших слів. Далі цей алгоритм зіставляє кожному слову певний вектор [38]. Отримані векторні уявлення слів можуть бути використані для обробки природної мови та машинного навчання. У цьому випадку було використано самий простий спосіб перетворення слів у вектор, коли кожному слову зіставляється свій власний вектор, який не залежить від інших слів.

Для даної задачі словник містив 2500 слів які найчастіше зустрічалися серед усіх коментарів з навчального датасету. Це більше ніж під час навчання нейромережі для аналізу тональності, так як задача пошуку спаму не потребує роботи GPU, а отже вільних обчислювальних ресурсів більше. Проте немає сенсу сильно збільшувати розмір словника, так як вхідний датасет містить біля двох тисяч коментарів, що не так багато за розміром як датасет для аналізу тональності. Далі цей датасет було поділено на навчальну та тестову частини у відношенні 80% на 20%, де більша частина даних відноситься до навчальної. Тобто навчальна вибірка складала 1568 коментарів, а тестова вибірка – 393 коментаря.

Для вирішення задачі пошуку спаму спочатку було використано метод наївного байєсівського класифікатора [47]. Наївний баєсів класифікатор – це імовірнісний класифікатор, заснований на застосуванні теореми Баєса з наївними припущеннями про незалежність. Цей класифікатор вважає, що всі ці властивості об’єкта незалежно сприяють його класифікації. У контексті класифікації наївний

класифікатор Байєса використовує цей принцип для оцінки ймовірності певного класу для певних вхідних даних. Існує кілька варіантів наївних байєсівських класифікаторів, у тому числі гаусівських наївних байєсів, мультиноміальних наївних байєсів та наївних байєсів Бернуллі, кожен з яких підходить для різних типів даних. Основна ідея класифікатора Наївного Байєса полягає в тому, щоб обчислити ймовірності кожного класу з урахуванням набору ознак, а потім призначити клас з найвищою ймовірністю для вибірки. Для класифікації тексту ймовірності зазвичай обчислюють за допомогою комбінації частоти слів у документі та загальної частоти слів у корпусі. Потім ці ймовірності використовуються для класифікації нових документів. Залежно від конкретної природи ймовірнісної моделі, наївні класифікатори Баєса можуть бути дуже ефективними в умовах навчання з учителем [3]. У багатьох практичних застосуваннях для оцінки параметрів наївних байєсівських моделей використовується метод максимальної правдоподібності. Незважаючи на спрощені припущення, наївні класифікатори Баєса дають високу точність в багатьох реальних ситуаціях. Ще одна перевага наївного байєсівського класифікатора полягає в тому, що для навчання достатньо невеликої кількісті даних.

Основою наївного байєсівського класифікатора є теорема Баєса. Це статистична теорема, яка описує зв’язок між попередньою ймовірністю, ймовірністю та апостеріорною ймовірністю [10]. У ньому стверджується, що ймовірність гіпотези (A) за наявності доказів (B) пропорційна відношенню ймовірності гіпотези та ймовірності доказів під гіпотезою. Математично ця теорема задається формулою:

$$P(A|B) = \frac{P(B|A)\cdot P(A)}{P(B)}, \quad (4.1)$$

де $P(A|B)$ – ймовірність події $A$ якщо відбулась подія $B$;

$P(A|B)$ – ймовірність події $B$ якщо відбулась подія $A$;

$P(A)$ – ймовірність події $A$;

$P(B)$ – ймовірність події $B$.

Для навчання було використано об’єкт MultinomialNB із бібліотеки sklearn. На вхід він приймав вектор у якому закодовано текст, а на вихід видає ймовірність належності даного тексту до спаму, від 0 до 1. Перед початком навчання того було описано функцію (рис. 4.2) яка навчає класифікатор та виводить його точність та матрицю невідповідностей.

```
def train_model(model, x_train, y_train, x_test, y_test):
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    y_prob = model.predict_proba(x_test)
    accuracy = round(accuracy_score(y_test, y_pred), 3)
    precision = round(precision_score(y_test, y_pred), 3)
    recall = round(recall_score(y_test, y_pred), 3)
    print(f'Accuracy of the model: {accuracy}')
    print(f'Precision Score of the model: {precision}')
    print(f'Recall Score of the model: {recall}')

    sns.set_context('notebook', font_scale= 1.3)
    fig, ax = plt.subplots(1, 1, figsize = (25,  8))
    plot_confusion_matrix(y_test, y_pred, ax=ax, cmap='YlGnBu')
```

Рисунок 4.2 – Функція навчання

Матриця невідповідностей – це таблиця, яка використовується для оцінки продуктивності алгоритму класифікації. Він порівнює фактичні класифікації набору даних із прогнозованими класифікаціями. Матриця зазвичай організована так, що рядки представляють фактичні класи, а стовпці представляють прогнозовані класи. Записи в матриці – це кількість даних, які належать до певного фактичного класу та, за прогнозами, належатимуть до певного прогнозованого класу. Записи вздовж діагоналі представляють правильні передбачення, а записи поза діагоналлю представляють неправильні передбачення. Матриця невідповідностей використовується для оцінки різних аспектів алгоритму класифікації [3]. Дана задача має два класи, тому матриця має вигляд таблиці з 4 різними комбінаціями прогнозованих та фактичних значень. Прогнозовані

значення описуються як позитивні та негативні, а фактичні – як справжні та хибні. Ці комбінації мають назви:

– $TP$ – кількість істинно-позитивних результатів (true positive);
– $TN$ – кількість істинно-негативних результатів (true negative);
– $FP$ – кількість хибно-позитивних результатів (false positive);
– $FN$ – кількість хибно-негативний результатів (false negative).

Результати навчання алгоритмом наївного байєсівського класифікатора наведено на рисунку 4.3. Точність складає 88.3%.

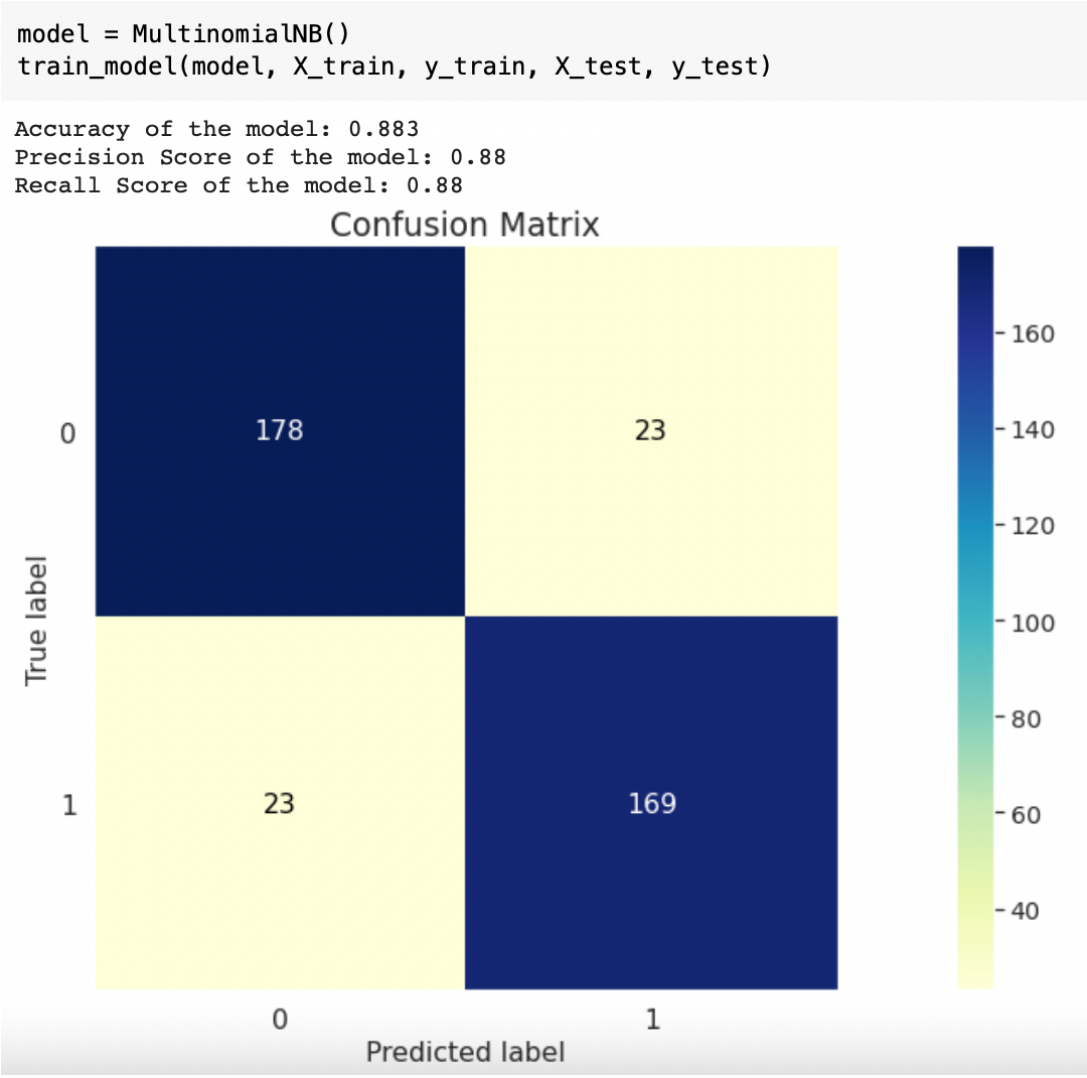


Рисунок 4.3 – Результат навчання наївного байєсівського класифікатора

Наступним кроком до параметрів навчання було додано ще один параметр, який описує тональність коментаря. Він був обчислений за допомогою раніше створеної нейромережі. Так як нейромережа для аналізу тональності видає значення в діапазоні від 1 до 5, то зробимо нормалізацію – приведемо значення до діапазону від 0 до 1 за допомогою функції на рисунку 4.4.

```
tfidf = TfidfVectorizer(max_features=2500, min_df=2)
X_train = tfidf.fit_transform(X_train_original).toarray()
X_test = tfidf.transform(X_test_original).toarray()

X_train_ext = np.column_stack(((net.predict_many(X_train) - 1) / 4, X_train))
X_test_ext = np.column_stack(((net.predict_many(X_test) - 1) / 4, X_test))
```

Рисунок 4.4 – Додавання параметру тональності до вектору коментаря

Після цього знову навчимо модель наївного байєсівського класифікатора. Для цього створимо нову модель, яку передаємо у функцію train_model разом з новими векторами, які містять параметр тональності тексту. Отримана модель має точність 93.1% (див. рис. 4.5). Отже, додавання параметру тональності допомогло підвищити точність на 4.8%.

```
model = MultinomialNB()
train_model(model, X_train_ext, y_train, X_test_ext, y_test)
```

```
Accuracy of the model: 0.931
Precision Score of the model: 0.937
Recall Score of the model: 0.922
```

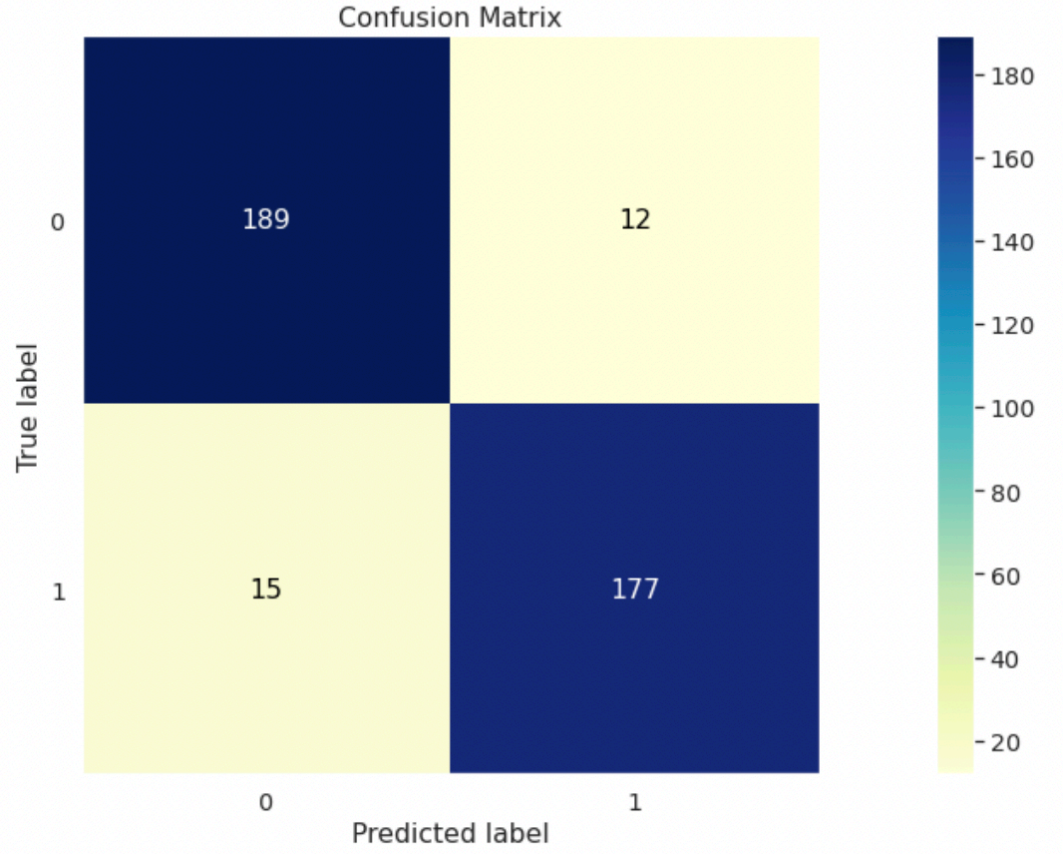


Рисунок 4.5 – Результат навчання наївного байєсівського класифікатора після додавання параметру тональності тексту

## 4.2 Використання створеної нейромережі для пошуку спаму за допомогою випадкового лісу

Випадковий ліс (Random Forest) – це ансамблевий алгоритм машинного навчання, який використовується для задач класифікації та регресії [48]. Він поєднує кілька дерев рішень для прогнозування. Дерева навчаються за допомогою початкових зразків даних, і кожне дерево робить прогноз. Остаточний прогноз робиться шляхом об’єднання прогнозів усіх дерев. Це призводить до більш надійної моделі порівняно з єдиним деревом рішень, оскільки воно зменшує перенавчання шляхом усереднення зміщень і дисперсії окремих дерев. Алгоритм

може обробляти як безперервні, так і категоріальні характеристики, відсутні дані та нелінійні зв'язки між функціями та цілями.

Однією з ключових переваг випадкового лісу є його здатність обробляти відсутні дані та дані з шумом [12]. На відміну від інших алгоритмів, на які можуть вплинути ці проблеми, випадковий ліс стійкий до таких даних. Ще одна перевага Random Forest полягає в тому, що його можна використовувати для оцінки важливості кожної функції в даних. Це корисно під час роботи з даними великої розмірності, оскільки дозволяє вибрати найважливіші характеристики для подальшого аналізу та інтерпретації. Крім того, випадковий ліс можна легко розпаралелити, що робить його швидким алгоритмом для великих наборів даних. Це робить його гарним вибором для різноманітних реальних додатків, де потрібно ефективно обробляти великі обсяги даних. Загальна структура випадкового лісу зображена на рисунку 4.6.

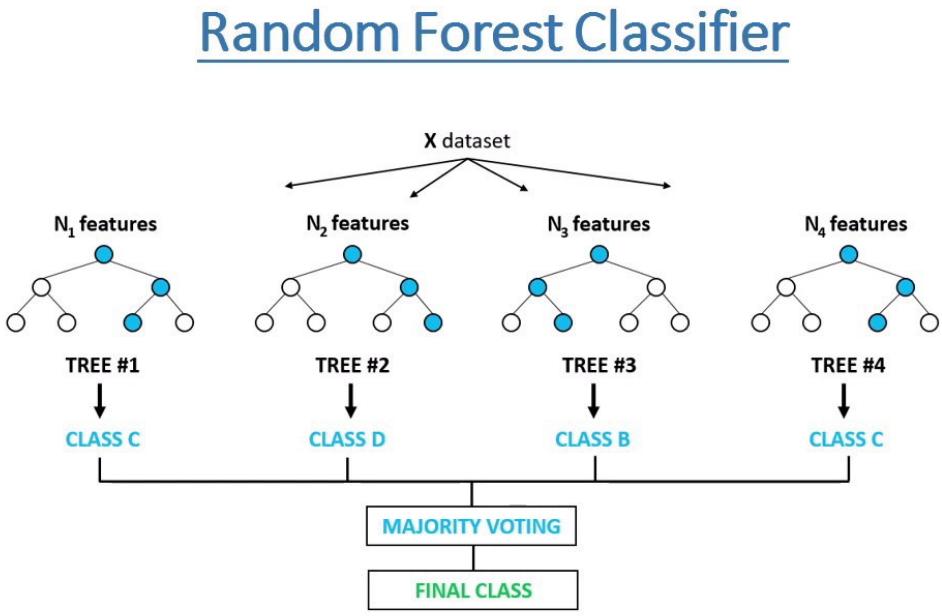


Рисунок 4.6 – Схема випадкового лісу

Як було зазначено вище, випадковий ліс складається з багатьох дерев рішень. Дерево рішень – це модель машинного навчання на основі дерева, яка

використовується для представлення рішень і можливих наслідків цих рішень. Воно складається з набору гілок і вузлів, які представляють різні рішення та результат кожного рішення. Кожен вузол у дереві представляє перевірку певної функції чи атрибута, а кожна гілка представляє можливі результати перевірки. Кінці гілок називаються листами, які представляють кінцеві результати або прогнози. Дерева рішень можна використовувати як для завдань класифікації, так і для регресії, і вони є популярним алгоритмом для прийняття рішень і розв'язання проблем у широкому діапазоні програм, включаючи інтелектуальний аналіз даних, розпізнавання образів і машинне навчання.

Дерево рішень – це структура, схожа на блок-схему, яка використовується для прийняття рішення або прогнозу щодо вхідної вибірки в машинному навчанні. Дерево будується шляхом рекурсивного поділу даних на основі значень ознак, доки не буде досягнуто бажаного рівня точності або мінімальної кількості вибірок на листковий вузол [3]. Прогноз робиться шляхом обходу дерева від кореня до кінцевого вузла та виведення відповідної мітки класу або розподілу ймовірностей. Дерева рішень можна використовувати як для задач регресії, так і для класифікації, і їх відносно легко інтерпретувати та візуалізувати, що робить їх популярним інструментом для дослідження та розуміння даних. Однак вони також схильні до перенавчання та можуть давати необ'єктивні результати, якщо певні особливості домінують у структурі дерева. Щоб усунути ці обмеження і використовується випадковий ліс.

Для пошуку спаму було обрано датасет [46] із минулої частини , який складається з 1961 коментаря. Попередню обробку та навчання було проведено у середовищі Jupyter. Текст кожного відгуку цього датасету було очищено, токенізовано та перетворено на вектор як описано у третьому розділі, тобто так само як було оброблено датасет на якому навчалася нейромережа для аналізу тональності. Перед початком навчання того було описано функцію (рис. 4.7) яка навчає класифікатор та виводить його точність та матрицю невідповідностей.

```
def train_model(model, x_train, y_train, x_test, y_test):
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    y_prob = model.predict_proba(x_test)
    accuracy = round(accuracy_score(y_test, y_pred), 3)
    precision = round(precision_score(y_test, y_pred), 3)
    recall = round(recall_score(y_test, y_pred), 3)
    print(f'Accuracy of the model: {accuracy}')
    print(f'Precision Score of the model: {precision}')
    print(f'Recall Score of the model: {recall}')

    sns.set_context('notebook', font_scale= 1.3)
    fig, ax = plt.subplots(1, 1, figsize = (25,  8))
    plot_confusion_matrix(y_test, y_pred, ax=ax, cmap='YlGnBu')
```

Рисунок 4.7 – Функція навчання

Результати навчання алгоритмом випадкового лісу наведено на рисунку 4.8. Точність складає 90.8%.

```
model = RandomForestClassifier(n_estimators=256)
train_model(model, X_train, y_train, X_test, y_test)
```

```
Accuracy of the model: 0.908
Precision Score of the model: 0.97
Recall Score of the model: 0.839
```

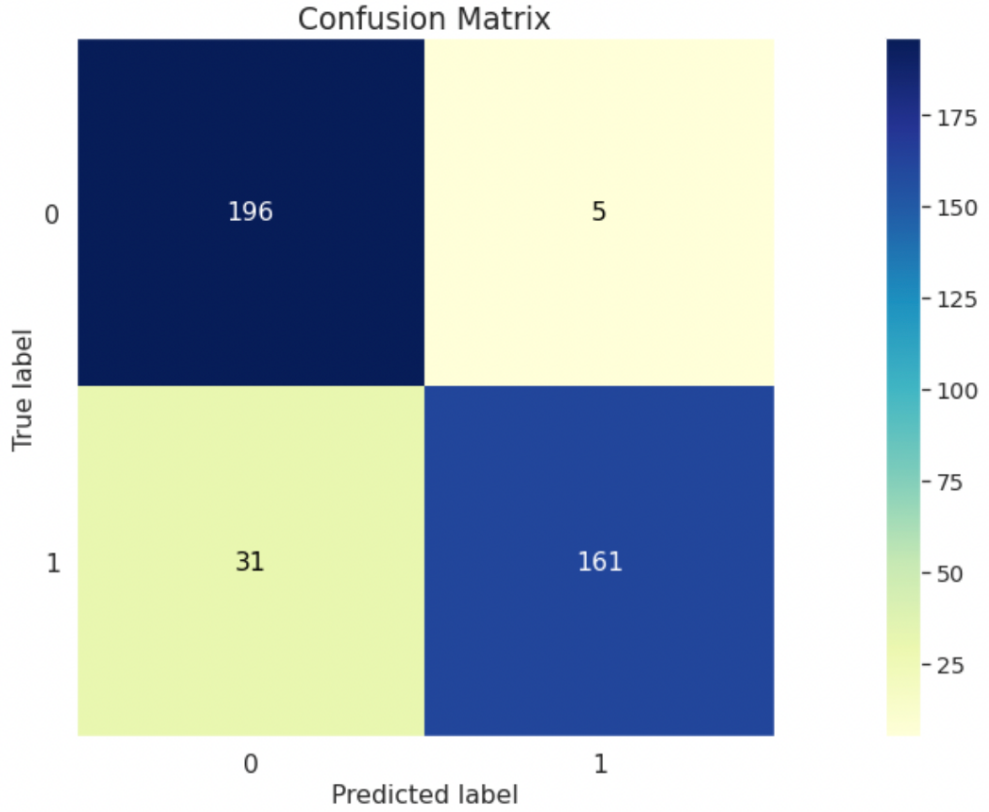


Рисунок 4.8 – Результат навчання випадкового лісу

Далі, як і в попередній частині, до параметрів навчання було додано ще один параметр, який описує тональність коментаря. Він був обчислений за допомогою раніше створеної нейромережі. Так само, оцінка тональності була нормована до проміжку від 0 до 1.

Після цього навчимо покращену модель випадкового лісу. Для цього створимо нову модель, яку передаємо у функцію train_model разом з новими векторами, які містять параметр тональності тексту. Отримана модель має точність 95.7% (див. рис. 4.9). Отже, додавання параметру тональності допомогло підвищити точність випадкового лісу на 4.9%.

Крім того, варто зазначити що точність випадкового лісу в середньому вища за точність наївного байєсівського класифікатора для задачі пошуку спаму. А додавання параметру тональності у обох випадках підвищує точність.

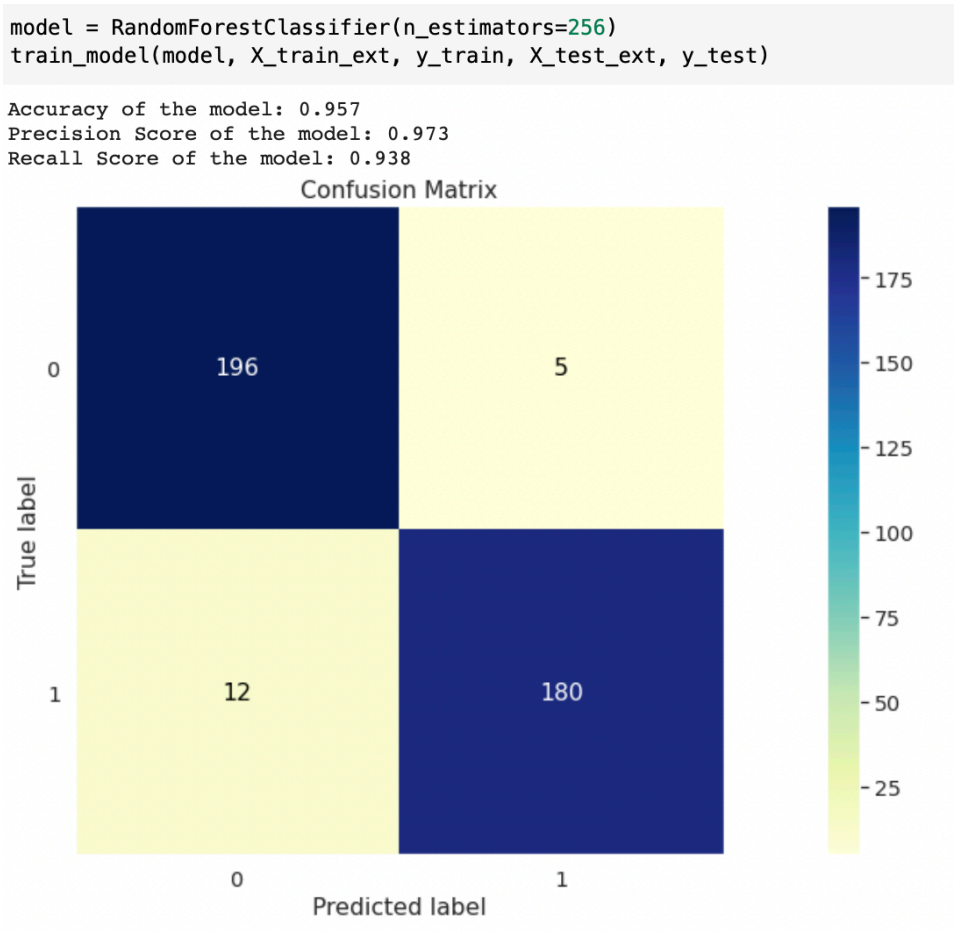


Рисунок 4.9 – Результат навчання випадкового лісу після додавання параметру тональності тексту

## 4.3 Створення API та деплой на сервері

Для зручної роботи з нейромережою для аналізу тональності було створено API на Python за допомогою фреймворку FastAPI. FastAPI – це сучасний, швидкий фреймворк для створення API за допомогою Python [49]. Він розроблений, щоб забезпечити високу продуктивність як для розробки, так і під час роботи. FastAPI побудовано на основі Starlette для веб-частин і Pydantic для частин даних, а також він використовує асинхронність для досягнення паралелізму з відносно низькими накладними витратами.

FastAPI забезпечує автоматичну перевірку вхідних даних і генерує інтерактивну документацію API за допомогою Swagger [50]. Він також підтримує протоколи WebSocket і GraphQL. FastAPI розроблено для бездоганної роботи з сучасними фронтенд фреймворками, такими як React, Angular, Vue та іншими. Основною його перевагою є можливість швидко створити HTTP сервер який буде приймати як GET, POST та інші запити.

Аналіз тексту нейромережою відбувається дуже швидко. Проте швидкість запиту через інтернет-з'єднання набагато повільніше. Тому для оптимізації всієї системи було реалізовано можливість відправляти на аналіз декілька текстів в одному запиті. При цьому відповіддю від API буде масив з оцінками для кожного із відправлених текстів. Ця оптимізація збільшила швидкість роботи усієї системи у декілька разів, та дозволила найбільш оптимальним способом використовувати ресурси. Саме цей метод було використано для аналізу великої кількості тексту під час пошуку спаму.

Однією з особливостей FastAPI є автоматична генерація документації до API на основі платформи Swagger. Згенерована документація до API нейромережі зображено на рисунку 4.10. Відповідно до даної документації, існує чотири можливих POST запитів:

– /analyze/one/ – приймає один коментар, передає його на нейромережу, яка повертає оцінку його тональності від 1 до 5;

– /analyze/one/deep – приймає один коментар, повертає оцінку його тональності від 1 до 5 та ймовірності для кожної оцінки;

– /analyze/many/ – приймає масив коментарів, повертає масив з оцінок тональності від 1 до 5;

– /analyze/many/deep – приймає масив коментарів, повертає масив з оцінок тональності від 1 до 5 разом з масивом ймовірностей для кожної оцінки.

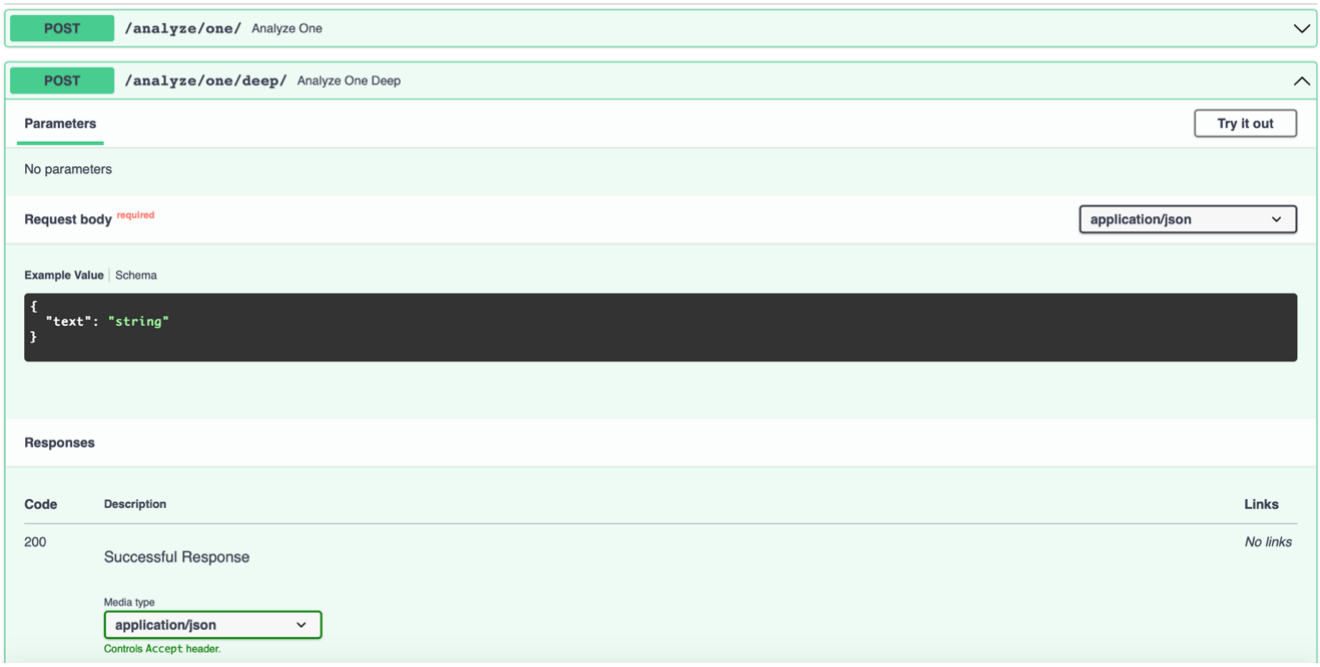


Рисунок 4.10 – Документація до API у Swagger

Крім того, документація у Swagger дозволяє відразу протестувати API та отримати відповідь. Приклад виклику для /analyze/one/deep зображено на рисунку 4.11.

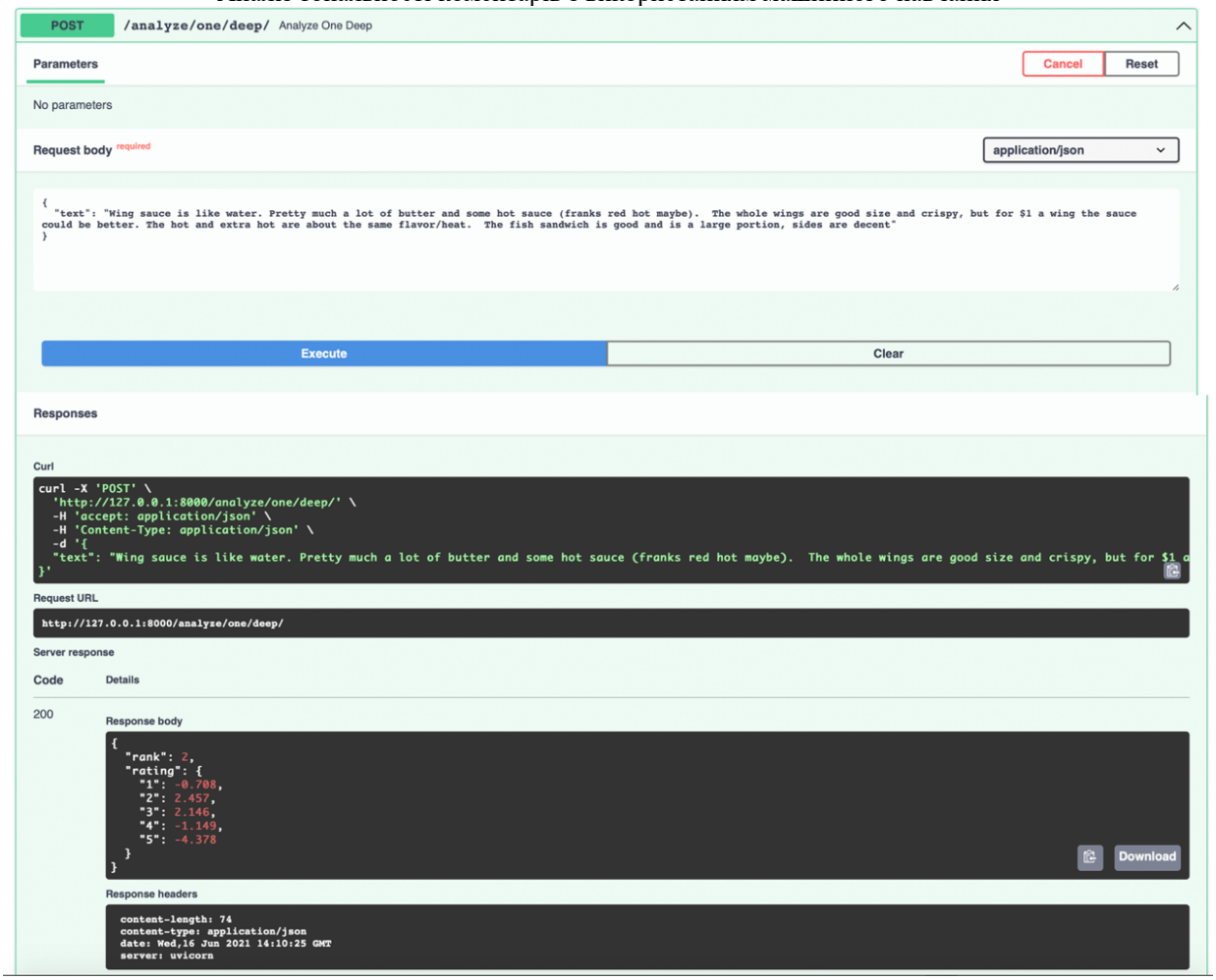


Рисунок 4.11 – Приклад виклику /analyze/one/deep через API

Для деплою системи на сервер було використано Docker. Docker – програмне забезпечення з відкритим вихідним кодом, що застосовується для розробки, тестування, доставки і запуску веб-додатків в середовищах з підтримкою контейнеризації [51]. Він потрібен для більш ефективного використання системи і ресурсів, швидкого розгортання готових програмних продуктів, а також для їх масштабування і перенесення в інші середовища з гарантованим збереженням стабільної роботи.

Основний принцип роботи Docker – контейнеризація. Цей тип віртуалізації дозволяє упаковувати програмне забезпечення по ізольованим середах – контейнерів. Кожен з цих віртуальних блоків містить всі необхідні елементи для роботи програми. Це дає можливість одночасного запуску великої кількості

контейнерів на одному хості. Ці контейнери працюють в окремих віртуальних середовищах, а отже можуть мати окремі обчислювальні ресурси.

Структура при розгортанні Docker-контейнеру зображена на рисунку 4.12.

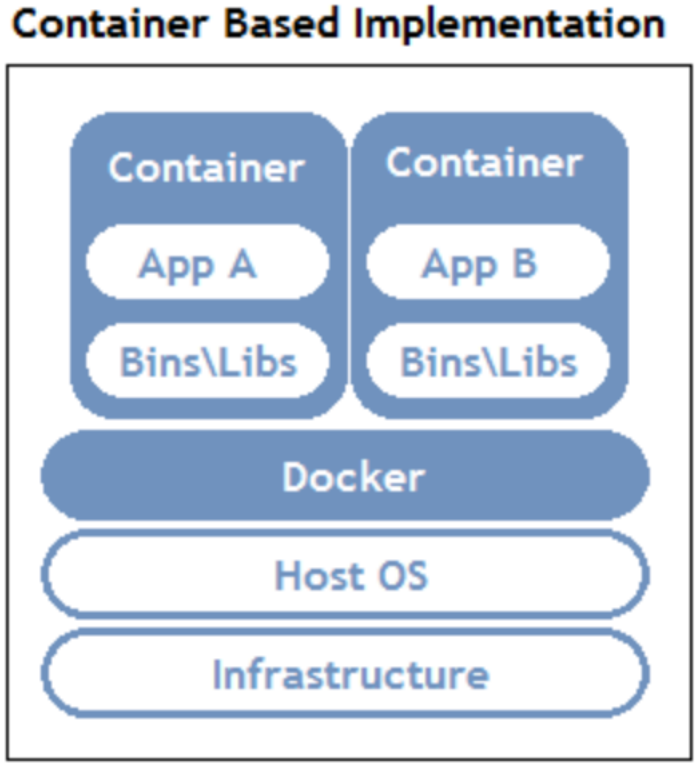


Рис. 4.12. Структура Docker образу

Переваги при використані Docker:

1) мінімальне споживання ресурсів – контейнери не віртуалізують всю ОС, а використовують ядро хоста та ізолюють програму на рівні процесу. Останній споживає набагато менше ресурсів локального комп'ютера, ніж віртуальна машина;

2) швидке розгортання – допоміжні компоненти можна не встановлювати, а використовувати вже готові docker-образи (шаблони). Наприклад, не має сенсу постійно встановлювати і налаштовувати Linux Ubuntu. Досить один раз її інсталювати, створити образ і постійно використовувати, лише оновлюючи версію при необхідності;

3) зручне приховування процесів – для кожного контейнера можна використовувати різні методи обробки даних, приховуючи фонові процеси;

4) робота з небезпечним кодом - технологія ізоляції контейнерів дозволяє запускати будь-який код без шкоди для ОС;

5) просте масштабування – будь-який проект можна розширити, запровадивши нові контейнери;

6) зручний запуск – додаток, що знаходиться всередині контейнера, можна запустити на будь-якому docker-хості;

7) оптимізація файлової системи – образ складається з шарів, які дозволяють дуже ефективно використовувати файлову систему.

Для розгортання усієї системи було описано Dockerfile (див. рис. 4.13).

```
FROM python:3.7-slim
RUN apt-get update
RUN pip install --upgrade pip
WORKDIR /app
COPY ./requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ['uvicorn', 'runner:app', '--host', '0.0.0.0', '--port', '80']
```

Рис. 4.13. Dockerfile для API з нейромережою

Для більш зручного деплою було створено docker-compose.yml файл за допомогою Docker Compose [52]. Його код наведено у додатку В. Цей файл було скопійовано на сервер за допомогою консольної команди SCP та запущено за допомогою команди docker-compose up --build -d.

**Висновки до розділу 4**

У цьому розділі було описано використання створеної раніше нейромережі для покращення результатів пошуку спаму. Було обрано датасет з коментарями від Youtube [46]. Текст кожного відгуку цього датасету було очищено, токенізовано та перетворено на вектор так само як було оброблено датасет на якому навчалася

нейромережа для аналізу тональності. При використанні алгоритму наївного байєсівського класифікатора без аналізу тональності, точність склала 88.3%, тоді як з параметром тональності тексту точність зросла до 93.1%. При використанні алгоритму випадкового лісу без аналізу тональності, точність склала 90.8%, тоді як з параметром тональності тексту точність зросла до 95.7%. Ці дані наведено на таблиці 4.1.

Таблиця 4.1 – Значення похибок на навчальному та тестовому датасетах

| **Модель** | **Точність** |
|---|---|
| Наївний байєсівський класифікатор без аналізу тональності | 88.3% |
| Наївний байєсівський класифікатор з аналізом тональності | 93.1% |
| Випадковий ліс без аналізу тональності | 90.8% |
| Випадковий ліс з аналізом тональності | 95.7% |

Можна зробити висновок, що додавання параметру тональності підвищує точність для обох моделей. Значення приросту точності становить 4.8% для наївного байєсівського класифікатора та 4.9% для випадкового лісу, а отже використання нейромережі для аналізу тональності допомагає краще вирішувати задачу пошуку спаму. Крім того, варто зазначити що точність випадкового лісу в середньому вища за точність наївного байєсівського класифікатора для даної задачі.

У третій частині цього розділу було описано створення API для нейромережі визначення спаму, та деплої його на сервер. Для створення API було використано Python фреймворк FastAPI, який автоматично згенерував Swagger документацію до API. Для деплою було використано технологію контейнеризації Docker та надбудова Docker Compose. Це дозволило запустити систему на сервері за допомогою лише декількох команд.

# ВИСНОВКИ

Під час виконання МКР було зроблено аналіз предметної області систем для аналізу тональності тексту, який показав що не існує універсального рішення. На відміну від задачі пошуку спаму, для якої існує багато готових систем, але вони мають закритий програмний код. А отже не відомо чи використовують вони аналіз тональності тексту. Крім того, у першому розділі було сформовано технічне завдання.

Наступним кроком був вибір, проектування та навчання нейромережі для аналізу тональності тексту. Було обрано LSTM нейромережу та датасет [33] для її навчання та перевірки. Було описано три метрики для оцінки якості нейромережі, а датасет було проаналізовано та розбито та навчальну, валідаційну та текстову вибірки.

Навчання нейромережі відбувалося на платформу Google Colab з використанням GPU. У результаті нейромережа змогла оцінювати тональність коментаря по шкалі від 1 до 5, де чим вище оцінка – тим більш емоційно-позитивний відгук і навпаки. Після навчання нейромережа досягла точності у 76.3% на тестовому датасеті, а серденя квадратична помилка становила 0.6478, що позначає що нейромережа помиляється менше ніж на один клас.

У четвертому розділі було використано цю нейромережу аналізу тональності для покращення результатів пошуку спаму. Для цього було обрано датасет [46], текст кожного відгуку цього датасету було очищено, токенізовано та перетворено на вектор так само як було оброблено датасет на якому навчалася нейромережа для аналізу тональності. При використанні алгоритму наївного байєсівського класифікатора без аналізу тональності, точність склала 88.3%, тоді як з параметром тональності тексту точність зросла до 93.1%. При використанні алгоритму випадкового лісу без аналізу тональності, точність склала 90.8%, тоді як з параметром тональності тексту точність зросла до 95.7%.

Можна зробити висновок, що додавання параметру тональності підвищує точність для обох моделей. Значення приросту точності становить 4.8% для

наївного байєсівського класифікатора та 4.9% для випадкового лісу. Отже, використання аналізу тональності допомагає краще вирішувати задачу пошуку спаму. Крім того, варто зазначити що точність випадкового лісу в середньому вища за точність наївного байєсівського класифікатора для даної задачі.

У методичній частині було створено дві лабораторних роботи на тему пошуку спаму. Перша лабораторна робота використовує метод наївного байєсівського класифікатора, а друга – метод випадкового лісу. Лабораторні роботи супроводжуються рисунками та прикладами коду.

У спеціальній частині з охорони праці та безпеки у надзвичайних ситуаціях було виконано аналіз умов праці в серверному приміщенні. Під час цього було перевірено забезпечення вимог охорони праці та виявлено, що оцінка умов праці на робочому місці відноситься до IV категорії, коли спостерігається робота у несприятливих умовах праці. Також було виявлено, що в даному виробничому приміщенні температура повітря для працівників є нижче норми, тобто її необхідно збільшити, а швидкість руху повітря навпаки необхідно зменшити. Для цього було розраховано припливно-витяжну вентиляцію та підібрано вентилятор з необхідною витратно-напірною характеристикою.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Hopfield J. J. Neural networks and physical systems with emergent collective computational abilities, 1984. С. 147-169.

2) Галузинський, Г. П. Сучасні технологічні засоби обробки інформації: навч. посіб. Київ : КНЕУ, 1998. 224 с

3) Тимощук П. В. Штучні нейронні мережі: навчальний посібник. Львів, 2011. 444 с.

4) Segaran T. Programming collective intelligence. LA, 2012

5) Творошенко, І. С. Особливості застосування сучасних принципів штучного інтелекту до розробки ефективних механізмів моделювання складних систем. 2018, С. 118–121.

6) Ясницкий Л.Н. Введення в штучний інтелект, 2005. 176 с.

7) Система-аналог Microsoft Azure Cognitive Service: веб-сайт. URL: https://azure.microsoft.com/services/cognitive-services (дата звернення 01.02.2023)

8) Система-аналог Lexalytics: веб-сайт. URL: https://lexalytics.com (дата звернення 01.02.2023).

9) Система-аналог Google Gmail: веб-сайт. URL: https://gmail.com (дата звернення 01.02.2023).

10) Згуровский М. З., Панкратова Н. Д. Основи системного аналізу. Київ, 2007. 544 с.

11) Deerwester, S. C., Dumais S. T., Landauer T. K. Indexing by Latent Semantic Analysis. 1990. С. 391–407.

12) Гороховатський В. О., Творошенко І. С. Методи інтелектуального аналізу та оброблення даних: навч. посібник. 2021.

13) Goodfellow I., Bengio Y., Courville A. Deep Learning, 2016. 773 с.

14) John E. Kelly I. Steve Hamm Smart Machine, 2014. 147 с.

15) Alzubi, J., Nayyar, A., Kumar, A. Machine learning from theory to algorithms: an overview, 2018. 43 с.

16) Куссуль В. М., Байдик Т. Н. Розробка архітектури нейроподібної мережі для розпізнавання форми об'єктів на зображенні, 1990, С. 56–61.

17) Gers F. A., Schmidhuber J., Cummins F. Learning to forget: continual prediction with LSTM. Великобританія, 1999. С. 850–855.

18) Illustrated Guide to LSTM: веб-сайт. URL: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 (дата звернення 01.02.2023).

19) Non-linear system modeling using LSTM neural networks: веб-сайт. URL: https://www.sciencedirect.com/science/article/pii/S2405896318310814 (дата звернення 01.02.2023).

20) Exploring different types of LSTM: веб-сайт. URL: https://medium.com/analytics-vidhya/exploring-different-types-of-lstms-6109bcb037c4 (дата звернення 01.02.2023).

21) Understanding Architecture of LSTM: веб-сайт. URL: https://www.analyticsvidhya.com/blog/2021/01/understanding-architecture-of-lstm (дата звернення 01.02.2023).

22) Клейсен М., Мур Б., Пошук гіперпараметрів у машинному навчанні, 2015 р. 1502 с.

23) Домхан Т., Спрінгенберг Дж. Т., Хаттер Ф. Прискорення автоматичної оптимізації гіперпараметрів глибоких нейронних мереж шляхом екстраполяції кривих навчання, 2015. 54 с.

24) Денисюк В. О. Проектування інструментального програмного забезпечення. Вінниця : ВНТУ, 2014. 22 с.

25) ML Classifier Performance Comparison for Spam Emails Detection: веб-сайт. URL: https://towardsdatascience.com/ml-classifier-performance-comparison-for-spam-emails-detection-77749926d508 (дата звернення 01.02.2023).

26) Алібрагім Г., Людвіг С. А. Оптимізація за гіперпараметрами: порівняння генетичного алгоритму з пошуком по сітці та байєсівською оптимізацією, 2021. 65 с.

27) Єрьоменко М. О. Аналіз існуючих програмних рішень для виявлення спаму. Харків: НТУ «ХПІ». С. 118.

28) Evaluation of over and underfeeding: веб-сайт. URL: https://pubmed.ncbi.nlm.nih.gov/23077114 (дата звернення 01.02.2023).

29) Evaluation Metrics for AI: веб-сайт. URL: https://vitalflux.com/different-success-metrics-for-ai-ml-initiatives (дата звернення 01.02.2023).

30) Катренко А. В. Системний аналіз: підручник. Львів, 2009. 396 с.

31) Пістун Є. П., Стасюк І. Д. Основи автоматики та автоматизації : навч. посіб. / Львів, 2014. 333 с.

32) Feature extraction for machine learning and deep learning: веб-сайт. URL: https://www.mathworks.com/discovery/feature-extraction.html (дата звернення 01.02.2023).

33) Yelp review full dataset: веб-сайт. URL: http://hidra.lbd.dcc.ufmg.br/datasets/yelp_2015/original/yelp_review_full_csv.tar.gz (дата звернення 01.02.2023).

34) Jupyter Notebook Documentation: веб-сайт. URL: https://jupyter.org/documentation (дата звернення 01.02.2023).

35) Yang Y., Pedersen J. O. A comparative study of feature selection in text categorization. 1997. С. 412–420.

36) Yang Y., Pedersen J. O. Feature selection in statistical learning of text categorization. – 1997.

37) Pandas: фундаментальна бібліотека Python для аналізу даних і статистики / МакКінні В. та ін. 2011. 9 с.

38) Python Word Embedding using Word2Vec: веб-сайт. URL: https://www.geeksforgeeks.org/python-word-embedding-using-word2vec (дата звернення 01.02.2023).

39) Павлік А., Сігал Дж., Шарп Х Краудсорсинг документації наукового програмного забезпечення: приклад проекту документації numpy, 2014 р. 36 с.

40) Metrics in Machine Learning: веб-сайт. URL: https://machine-learning.paperspace.com/wiki/metrics-in-machine-learning (дата звернення 01.02.2023).

41) Torch Machine Learning Framework System Description: веб-сайт. URL: https://pytorch.org/ecosystem (дата звернення 01.02.2023).

42) Tensor Flow Machine Learning Documentation: веб-сайт. URL: https://tensorflow.org (дата звернення 01.02.2023).

43) CUDA Toolkit Documentation: веб-сайт. URL: https://docs.nvidia.com/cuda/index.html (дата звернення 01.02.2023).

44) Google Colab System: веб-сайт. URL: https://colab.research.google.com/notebooks (дата звернення 01.02.2023).

45) Google Colab GPU Free Usage: веб-сайт. URL: https://www.tutorialspoint.com/google_colab/google_colab_using_free_gpu.htm (дата звернення 01.02.2023).

46) YouTube Spam Collection Data Set: веб-сайт. URL: https://archive.ics.uci.edu/ml/datasets/Youtube+Spam+Collection (дата звернення 01.02.2023).

47) Email Spam Filtering Using Naive Bayes Classifier: веб-сайт. URL: https://www.springboard.com/blog/data-science/bayes-spam-filter (дата звернення 01.02.2023).

48) AI-based smart prediction of clinical disease using random forest classifier and Naive Bayes / Джекінс В., Вімал С., Каліаппан М., Лі М. Й., 2021. 77 с.

49) FastAPI Documentation: веб-сайт. URL: https://fastapi.tiangolo.com (дата звернення 01.02.2023).

50) FastAPI in Containers – Docker: веб-сайт. URL: https://fastapi.tiangolo.com/deployment/docker (дата звернення 01.02.2023).

51) DockerHub: Build and Deploy any Application Anywhere: веб-сайт. URL: https://docs.docker.com/desktop (дата звернення 01.02.2023).

52) Docker Compose overview: веб-сайт. URL: https://docs.docker.com/compose (дата звернення 01.02.2023).

53) ДСТУ 2293-99 Охорона праці. Терміни та визначення основних понять. Київ, 1999.

54) Про затвердження Правил пожежної безпеки в Україні: Наказ міністерства внутрішніх справ України від 30.12.2014 № 1417 URL: https://zakon.rada.gov.ua/laws/show/z0252-15#Text (дата звернення 01.02.2023).

55) Москальова В. М. Основи охорони праці: підручник. Київ: ВД Професіонал, 2005. 666 с.

# ДОДАТОК А

## Лістинг коду навчання нейромережі

```
import re
import string
import tarfile
from collections import Counter

import en_core_web_sm
import numpy as np
import pandas as pd
import requests
import torch
import torch.nn as nn
import torch.nn.functional as functional
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from torch.nn.utils.rnn import pack_padded_sequence
from torch.utils.data import Dataset, DataLoader

DATASET_URL =
'http://hidra.lbd.dcc.ufmg.br/datasets/yelp_2015/original/yelp_review_full_csv.tar
.gz'
DATASET_ARCHIVE_NAME = 'dataset.tar.gz'
DATASET_FOLDER_NAME = 'dataset'

ENGLISH = en_core_web_sm.load()


def download_file(url: str, filename: str, chunk_size: int = 2 ** 15):
    with requests.get(url, stream=True) as request:
        request.raise_for_status()
        with open(filename, 'wb') as file:
            for chunk in request.iter_content(chunk_size=chunk_size):
                file.write(chunk)


def extract_archive(archive_name: str, folder_name: str):
    with tarfile.open(archive_name) as tar:
        tar.extractall(path=folder_name)


# Download dataset
download_file(DATASET_URL, DATASET_ARCHIVE_NAME)
extract_archive(DATASET_ARCHIVE_NAME, DATASET_FOLDER_NAME)

# Read dataset
test = pd.read_csv(f'{DATASET_FOLDER_NAME}/yelp_review_full_csv/test.csv',
header=None)
train = pd.read_csv(f'{DATASET_FOLDER_NAME}/yelp_review_full_csv/train.csv',
header=None)

test = test.rename(columns={0: 'label', 1: 'review'})
train = train.rename(columns={0: 'label', 1: 'review'})

print(len(test), len(train))
train.head()


# Tokenize in order to clean text
def tokenize(text: str) -> list:
```

```
    text = re.sub(r'[^\\x00-\\x7F]+', ' ', text)
    regex = re.compile('[' + re.escape(string.punctuation) + '0-9\\r\\t\\n]')
    words = regex.sub(' ', text.lower())
    words = re.sub(r'\\s+', ' ', words.strip(), flags=re.UNICODE)
    return [token.text for token in ENGLISH.tokenizer(words)]


counts = Counter()
for index, row in train.iterrows():
    if index % 100 == 0:
        percent = 100 * index // len(train)
        print(f'{percent}%')
    counts.update(tokenize(row['description']))

# Check words with spaces (must be empty)
for word in list(counts):
    if ' ' in word:
        print(word)
        print(counts[word])

# Create vocabulary
counts_most = counts.most_common(2000)
word2vec = {'': 0, 'UNK': 1}
words = ['', 'UNK']
for word, freq in counts_most:
    word2vec[word] = len(words)
    words.append(word)


# Encode reviews to array of int
def encode_sentence(text: str, word2vec: dict, size: int = 150):
    tokenized = tokenize(text)
    encoded = np.zeros(size, dtype=int)
    enc1 = np.array([word2vec.get(word, word2vec['UNK']) for word in tokenized])
    length = min(size, len(enc1))
    encoded[:length] = enc1[:length]
    return encoded, length


train['encoded'] = train['review'].apply(lambda x: np.array(encode_sentence(x,
word2vec), dtype=object))

# Find and drop reviews with zero word length
indexes = []
for index, row in train.iterrows():
    if row['encoded'][0][0] == 0:
        indexes.append(index)
train.iloc[indexes].head()
train.drop(train.index[indexes], inplace=True)

# Transform labels from 1..5 to 0..4
zero_numbering = {1: 0, 2: 1, 3: 2, 4: 3, 5: 4}
train['label'] = train['label'].apply(lambda x: zero_numbering[x])
Counter(train['label'])

# Split into train and validation subsets
x = list(train['encoded'])
y = list(train['label'])
x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.2)


# Dataset
class ReviewsDataset(Dataset):
```

```
    def __init__(self, x: list, y: list):
        self.x = x
        self.y = y

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        tensor = torch.from_numpy(self.x[idx][0].astype(np.int32))
        return tensor, self.y[idx], self.x[idx][1]


train_ds = ReviewsDataset(x_train, y_train)
valid_ds = ReviewsDataset(x_valid, y_valid)

# Check cuda device (GPU)
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
torch.cuda.is_available()


# Create train and validation functions
def train_model(model: torch.nn.Module, epochs: int = 10, lr: float = 0.001):
    parameters = filter(lambda p: p.requires_grad, model.parameters())
    optimizer = torch.optim.Adam(parameters, lr=lr)
    for i in range(epochs):
        model.train()
        sum_loss = 0.0
        total = 0
        for x, y, l in train_dataloader:
            x = x.long().to(device)
            y = y.long().to(device)
            y_pred = model(x, l)
            optimizer.zero_grad()
            loss = functional.cross_entropy(y_pred, y)
            loss.backward()
            optimizer.step()
            sum_loss += loss.item() * y.shape[0]
            total += y.shape[0]
        val_loss, val_acc, val_rmse = validation_metrics(model, valid_dataloader)
        print('train loss %.3f, val loss %.3f, val accuracy %.3f, and val rmse
%.3f' % (
            sum_loss / total, val_loss, val_acc, val_rmse))


def validation_metrics(model: torch.nn.Module, dataloader: DataLoader):
    model.eval()
    correct = 0
    total = 0
    sum_loss = 0.0
    sum_rmse = 0.0
    for x, y, l in dataloader:
        x = x.long().to(device)
        y = y.long().to(device)
        y_hat = model(x, l)
        loss = functional.cross_entropy(y_hat, y)
        pred = torch.max(y_hat, 1)[1].cpu()
        correct += (pred == y.cpu()).float().sum().cpu()
        total += y.shape[0]
        sum_loss += loss.item() * y.shape[0]
        sum_rmse += np.sqrt(mean_squared_error(pred, y.cpu().unsqueeze(-1))) *
y.shape[0]
    return sum_loss / total, correct / total, sum_rmse / total
```

```
# Create LSTM net class
class CustomLSTM(torch.nn.Module):
    def __init__(self, vocab_size: int, embedding_dim: int, hidden_dim: int):
        super().__init__()
        self.hidden_dim = hidden_dim
        self.embeddings = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
        self.dropout = nn.Dropout(0.2)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.linear = nn.Linear(hidden_dim, 5)
        # initialize the hidden state (see code below)
        self.hidden_dim = self.init_hidden()

    def init_hidden(self):
        """At the start of training, we need to initialize a hidden state
        there will be none because the hidden state is formed based on previously
seen data
        So, this function defines a hidden state with all zeroes and of a
specified size
        Hi everyone!"""
        # The axes dimensions are (n_layers, batch_size, hidden_dim)
        return torch.zeros(1, 1, self.hidden_dim).to(device), torch.zeros(1, 1,
self.hidden_dim).to(device)

    def forward(self, x, s):
        x = self.embeddings(x)
        x = self.dropout(x)
        x_pack = pack_padded_sequence(x, s, batch_first=True,
enforce_sorted=False)
        out_pack, (ht, ct) = self.lstm(x_pack)
        out = self.linear(ht[-1])
        return out


# Create dataloaders
batch_size = 2000
vocab_size = len(words)
train_dataloader = DataLoader(train_ds, batch_size=batch_size, shuffle=True)
valid_dataloader = DataLoader(valid_ds, batch_size=batch_size)

# Create model
model = CustomLSTM(vocab_size, 100, 100)
model.to(device)

# Train model
train_model(model, epochs=20, lr=0.005)

torch.save(model.state_dict(), 'model.pt')

model.load_state_dict(torch.load('model.pt'))

# Encode test dataframe
test['encoded'] = test['review'].apply(lambda x: np.array(encode_sentence(x,
word2vec), dtype=object))

indexes = []
for index, row in test.iterrows():
    if row['encoded'][0][0] == 0:
        indexes.append(index)
test.drop(test.index[indexes], inplace=True)

test['label'] = test['label'].apply(lambda x: zero_numbering[x])
```

```
# Create dataloader for test dataset
x_test = list(test['encoded'])
y_test = list(test['label'])
test_dataset = ReviewsDataset(x_test, y_test)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size)

# Evaluate model on test dataset
test_loss, test_acc, test_rmse = validation_metrics(model, test_dataloader)
print('test loss %.3f, test accuracy %.3f, and test rmse %.3f' % (test_loss,
test_acc, test_rmse))
```

# ДОДАТОК Б

## Лістинг коду моделі випадкового лісу для пошуку спаму

```
import re
import string
import warnings
from collections import Counter
from functools import reduce

import matplotlib.pyplot as plt
import nltk
import numpy as np
import pandas as pd
import seaborn as sns
from nltk import word_tokenize
from nltk.corpus import stopwords
from scikitplot.metrics import plot_confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from wordcloud import WordCloud

sns.set()
warnings.filterwarnings('ignore')

nltk.download('punkt')
nltk.download('stopwords')

df1 = pd.read_csv('yt/Youtube01-Psy.csv')
df2 = pd.read_csv('yt/Youtube02-KatyPerry.csv')
df3 = pd.read_csv('yt/Youtube03-LMFAO.csv')
df4 = pd.read_csv('yt/Youtube04-Eminem.csv')
df5 = pd.read_csv('yt/Youtube05-Shakira.csv')
data = reduce(lambda a, b: a.append(b, ignore_index=True), [df1, df2, df3, df4,
df5])
data = data.rename(columns={'CONTENT': 'Message', 'CLASS': 'Category'})
data['Category'] = data['Category'].replace({1: 'spam', 0: 'ham'})
data.head()

labels = ['Spam (спам)', 'Ham (не спам)']
sizes = data['Category'].rename('Кількість').value_counts()

plt.figure(figsize=(10, 6), dpi=300)
plt.subplot(1, 2, 1)
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=['#ff7675', '#74b9ff'])
plt.subplot(1, 2, 2)
sns.barplot(x=labels, y=sizes, palette='viridis')

plt.show()

data['Total Words'] = data['Message'].apply(lambda x: len(x.split()))


def count_total_words(text: str):
    char = 0
    for word in text.split():
        char += len(word)
    return char
```

```
data['Total Chars'] = data['Message'].apply(count_total_words)


def kdeplot_column(data: pd.DataFrame, column: str):
    plt.figure(figsize=(10, 6))
    sns.kdeplot(x=data[column], hue=data['Category'], palette='winter',
shade=True)
    plt.show()


kdeplot_column(data, 'Total Words')
kdeplot_column(data, 'Total Chars')


def to_lowercase(text: str):
    text = text.lower()
    return text


data['Message'] = data['Message'].apply(to_lowercase)


def remove_urls(text: str):
    re_url = re.compile('https?://\S+|www\.\S+')
    return re_url.sub('', text)


data['Message'] = data['Message'].apply(remove_urls)

exclude = string.punctuation


def remove_punctuations(text: str):
    return text.translate(str.maketrans('', '', exclude))


data['Message'] = data['Message'].apply(remove_punctuations)


def remove_stopwords(text: str):
    new_list = []
    words = word_tokenize(text)
    stopwrds = stopwords.words('english')
    for word in words:
        if word not in stopwrds:
            new_list.append(word)
    return ' '.join(new_list)


data['Message'] = data['Message'].apply(remove_stopwords)

data['Total Words After Transformation'] = data['Message'].apply(lambda x:
np.log(len(x.split())))


def show_wordcloud(data: pd.DataFrame, category: str):
    text = ' '.join(data[data['Category'] == category]['Message'])
    plt.figure(figsize=(15, 10))
    wordcloud = WordCloud(max_words=500, height=800, width=1500,
background_color='black', colormap='viridis') \
        .generate(text)
    plt.imshow(wordcloud, interpolation='bilinear')
```

```
    plt.axis('off')
    plt.show()


show_wordcloud(data, 'spam')
show_wordcloud(data, 'ham')


def most_used_words_barplot(data: pd.DataFrame, category: str):
    words = list()
    for sentence in data[data['Category'] == category]['Message'].to_list():
        for word in sentence.split():
            words.append(word)
    df = pd.DataFrame(Counter(words).most_common(25), columns=['Word',
'Frequency'])

    sns.set_context('notebook', font_scale=1.3)
    plt.figure(figsize=(18, 8))
    sns.barplot(y=df['Word'], x=df['Frequency'], palette='summer')
    plt.title(f'Most Commonly Used {category.title()} Words')
    plt.xlabel('Frequnecy')
    plt.ylabel('Spam Words')
    plt.show()


most_used_words_barplot(data, 'spam')
most_used_words_barplot(data, 'ham')

data['Category'] = data['Category'].replace({'spam': 0, 'ham': 1})

X = data['Message']
y = data['Category'].values
X_train_original, X_test_original, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

tfidf = TfidfVectorizer(max_features=2500, min_df=2)
X_train = tfidf.fit_transform(X_train_original).toarray()
X_test = tfidf.transform(X_test_original).toarray()

X_train_ext = np.column_stack(((net.predict_many(X_train) - 1) / 4, X_train))
X_test_ext = np.column_stack(((net.predict_many(X_test) - 1) / 4, X_test))


def train_model(model, x_train: np.ndarray, y_train: np.ndarray, x_test:
np.ndarray, y_test: np.ndarray):
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    y_prob = model.predict_proba(x_test)
    accuracy = round(accuracy_score(y_test, y_pred), 3)
    precision = round(precision_score(y_test, y_pred), 3)
    recall = round(recall_score(y_test, y_pred), 3)
    # accuracy, precision, recall = 0.957, 0.973, 0.938
    print(f'Accuracy of the model: {accuracy}')
    print(f'Precision Score of the model: {precision}')
    print(f'Recall Score of the model: {recall}')

    sns.set_context('notebook', font_scale=1.3)
    fig, ax = plt.subplots(1, 1, figsize=(25, 8))
    plot_confusion_matrix(y_test, y_pred, ax=ax, cmap='YlGnBu')
    # fake_plot(np.asarray([[196, 5], [12, 180]]))


model = MultinomialNB()
```

```
train_model(model, X_train, y_train, X_test, y_test)

model = MultinomialNB()
train_model(model, X_train_ext, y_train, X_test_ext, y_test)

model = RandomForestClassifier(n_estimators=256)
train_model(model, X_train, y_train, X_test, y_test)

model = RandomForestClassifier(n_estimators=256)
train_model(model, X_train_ext, y_train, X_test_ext, y_test)
```

# ДОДАТОК В

## Docker Compose файл для деплою API

```yaml
version: '3.1'

services:

  api:
    restart: always
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - 80:80
    volumes:
      - ./resources:/app/resources
```