КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра технологій управління

Спеціальність 122 - Комп'ютерні науки,
освітня програма «Інформаційна аналітика та впливи»

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему:

"Розроблення моделі інтелектуального аналізу емоційного забарвлення контенту на e-commerce платформах"

Студента 2-го курсу групи ІАВ-21

Ковальчука Данила Сергійовича
(прізвище, ім'я, по батькові)

Науковий керівник:

д-р. техн. наук, професор
(науковий ступінь, вчене звання)

Заріцький Олег Володимирович
(прізвище, ім'я, по батькові)

_____ _____ _____
(підпис студента) (дата) (підпис)

Попередній захист:

_____
(Висновок: «До захисту в Екзаменаційній комісії»)

Завідувач кафедри
технологій управління _____ _____ _____
(підпис) (прізвище, ініціали) (дата)

Київ - 2025

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій

Кафедра технологій управління

Освітньо-кваліфікаційний рівень Магістр

Спеціальність 122-Комп'ютерні науки

Освітня програма Інформаційна аналітика та впливи

ЗАТВЕРДЖУЮ
Завідувач кафедри
професор Віктор МОРОЗОВ
_____
«___» _____ 20__ р.

З А В Д А Н Н Я
НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Студент Ковальчук Д. С.

Група ІАВ-21

1. Тема кваліфікаційної роботи

Розроблення моделі інтелектуального аналізу емоційного забарвлення контенту на e-commerce платформах

Затверджена наказом від «___»_____20__ р. № ___.

2. Строк подання студентом готової роботи - "___"_____ 20___ р.

3. Цільова установка та вихідні дані до роботи

Розробити модель інтелектуального аналізу емоційного забарвлення контенту на e-commerce платформах шляхом застосування сучасних методів обробки природної мови та машинного навчання. Здійснити комплексне дослідження текстових відгуків користувачів з метою виявлення полярності висловлювань, що дозволить автоматизувати аналіз настроїв споживачів. Для навчання та тестування моделі передбачено використання публічних наборів

текстових рецензій, які містять заголовки, тіла повідомлень і мітки полярності.

Вихідними даними моделі є передбачені емоційні класи, що можуть бути використані для вдосконалення рекомендаційних систем, моніторингу клієнтського задоволення та оперативного реагування на негативний зворотний зв’язок.

4. Зміст роботи

Зміст роботи охоплює аналіз наукових джерел щодо застосування сучасних методів обробки природної мови та машинного навчання для виявлення емоційної полярності текстів у сфері електронної комерції. Особливу увагу приділено підбору, попередньому опрацюванню публічних наборів текстових відгуків користувачів. У процесі дослідження реалізовано побудову та навчання моделі класифікації полярності текстового контенту, а також здійснено її оцінювання за допомогою низки метрик ефективності. Передбачено створення програмного інтерфейсу (API) для інтеграції моделі в інфраструктуру e-commerce платформ та формулювання висновків щодо ефективності запропонованого підходу.

5. Перелік графічного матеріалу (слайдів)

6. Календарний план виконання роботи:

№ п/п	Назва частин роботи	%	Виконання роботи	
			За планом	Фактично
1.	Вибір теми дипломної роботи	3	01.10.2024	01.10.2024
2.	Протокол кафедри ТУ про затвердження тем дипломних робіт та призначення наукових керівників	2	27.12.2024	27.12.2024
3.	Складання розгорнутого плану кваліфікаційної роботи	5	15.02.2025	15.02.2025
4.	Ознайомлення з іноземною та вітчизняною літературою та основними поняттями за темою	10	23.02.2025	23.02.2025

	наукового дослідження			
5.	Ознайомлення наукового керівника з розгорнутим планом кваліфікаційної роботи. Внесення змін	5	25.02.2025	25.02.2025
6.	Підготовка розділу 1 «Дослідження проблеми визначення емоційного забарвлення текстів та аналіз актуальних підходів до її розв’язання»	10	16.03.2025	16.03.2025
7.	Підготовка розділу 2 «Методи та моделі для розв’язання задачі визначення емоційного забарвлення текстового контенту»	15	31.03.2025	31.03.2025
8.	Підготовка розділу 3 «Розробка та оцінка моделі інтелектуального аналізу емоційного забарвлення текстів»	15	20.04.2025	20.04.2025
9.	Підготовка розділу 4 «Застосування розробленої моделі для обробки відгуків в сфері e-commerce»	14	26.04.2025	26.04.2025
10.	Оформлення кваліфікаційної роботи. Підготовка презентації	15	04.05.2025	04.05.2025
11.	Передача кваліфікаційної роботи науковому керівникові	2	13.05.2025	13.05.2025
12.	Передача кваліфікаційної роботи рецензенту для рецензування	2	13.05.2025	13.05.2025
13.	Попередній захист кваліфікаційної роботи	5	13.05.2025	13.05.2025

Дата видачі завдання «____» _____ 20__ р.

Керівник роботи _д-р. техн. наук, професор Заріцький Олег Володимирович
(посада, прізвище, ім’я, по батькові)

_____
(підпис)

Завдання прийняв до виконання студент групи ІАВ-21 Данило КОВАЛЬЧУК

_____
(підпис)

ЗМІСТ

АНОТАЦІЯ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра технологій управління

Спеціальність 122 - Комп’ютерні науки,

освітня програма “Інформаційна аналітика та впливи”

Дипломна робота магістра Ковальчука Д. С.

Тема роботи - «Розроблення моделі інтелектуального аналізу емоційного забарвлення контенту на e-commerce платформах».

Мета дипломної роботи магістра - розробка ефективної моделі інтелектуального аналізу емоційного забарвлення текстового контенту на платформах електронної комерції з використанням сучасних методів машинного навчання, а також створення прикладного інструменту у вигляді програмного інтерфейсу (API) для практичного застосування результатів аналізу.

Об’єкт дослідження - процеси аналізу емоційної полярності текстових відгуків користувачів в e-commerce середовищі.

Предмет дослідження - методи попередньої обробки текстових даних, машинного навчання та глибокого навчання для класифікації полярності, а також підходи до розгортання та інтеграції розробленої моделі в інфраструктуру електронної комерції.

Наукова новизна роботи полягає в удосконаленні аналізу емоційного забарвлення текстів шляхом поєднання лінгвістичного та семантичного аналізу з використанням трансформерних моделей, зокрема компактної моделі TinyBERT, яка на відміну від існуючих методів була адаптована до задачі бінарної класифікації полярності.

У роботі проведено огляд наукових джерел щодо методів аналізу емоційного забарвлення текстового контенту, а також досліджень, присвячених ефективності різних моделей у задачах обробки природної мови, зокрема в

контексті класифікації емоційної полярності. Розглянуто наукові праці, що демонструють доцільність застосування трансформерних архітектур, таких як BERT і його модифікацій, для задач емоційного аналізу. Окрему увагу приділено аналізу джерел, які підтверджують актуальність теми дослідження, зокрема динаміці зростання обсягів текстового контенту в інтернет-магазинах та стрімкому розвитку e-commerce як галузі. Здійснено збір і обробку текстових даних, реалізовано процес побудови моделі, її тренування та оцінювання за допомогою метрик точності, влучності, повноти та F1-міри. Створено API для забезпечення практичного застосування розробленої системи в e-commerce контексті.

Дипломна робота складається зі вступу, чотирьох розділів, висновків і списку використаних джерел. Загальний обсяг – 115 сторінок, кількість використаних джерел - 30.

Ключові слова: сентимент-аналіз, e-commerce, відгуки користувачів, машинне навчання, трансформери, BERT, бінарна класифікація емоційного забарвлення, текстові дані, API.

ПЕРЕЛІК ВИКОРИСТАНИХ СКОРОЧЕНЬ

BERT - Bidirectional Encoder Representations from Transformers (двостороння трансформерна модель від Google для контекстного розуміння тексту)

CNN - Convolutional Neural Network (згорткова нейронна мережа)

CRISP-DM - Cross-Industry Standard Process for Data Mining (заг-галузевий стандартний процес добування знань з даних, що охоплює етапи від розуміння бізнесу до впровадження моделі)

FFN - Feedforward Neural Network (нейронна мережа прямого розповсюдження сигналу, без зворотних зв'язків, що використовується в задачах класифікації та регресії)

GPT - Generative Pre-trained Transformer (генеративна трансформерна модель, що виконує завдання обробки мови шляхом автогенерації тексту)

GRU - Gated Recurrent Unit (альтернативна архітектура до LSTM, спрощена модель RNN з ефективним механізмом збереження контексту)

KDD - Knowledge Discovery in Databases (процес виявлення корисних знань із великих обсягів даних шляхом добування, аналізу та інтерпретації закономірностей)

k-NN - k-Nearest Neighbors (алгоритм класифікації та регресії, який класифікує об'єкти на основі схожості з найближчими сусідами у вибірці)

LSTM - Long Short-Term Memory (вид RNN, що здатен зберігати довгострокові залежності у послідовностях завдяки спеціальній структурі комірок пам'яті)

NLP - Natural Language Processing (обробка природної мови, галузь штучного інтелекту, що займається взаємодією між комп'ютерами та людською мовою)

OSEMN - Obtain, Scrub, Explore, Model, and Interpret (модель процесу роботи з даними, що включає отримання, очищення, дослідження, моделювання й інтерпретацію результатів)

RNN - Recurrent Neural Network (рекурентна нейронна мережа)

RoBERTa - Robustly Optimized BERT Approach (модифікація BERT з покращеним тренуванням, що забезпечує вищу точність на NLP-завданнях)

SEMMA - Sample, Explore, Modify, Model, Assess (модель аналітичного процесу

від SAS, що включає відбір даних, дослідження, трансформацію, моделювання та оцінку)

T5 - Text-To-Text Transfer Transformer (модель від Google, яка формулює всі NLP-завдання як задачі перетворення одного тексту в інший)

ШНМ - штучна нейронна мережа (математична модель, натхненна будовою людського мозку, яка використовується для розв’язання задач класифікації, прогнозування тощо)

ВСТУП

Аналіз емоційного забарвлення відгуків користувачів є важливим інструментом для сучасних e-commerce платформ, оскільки дозволяє автоматично виявляти настрої клієнтів, оперативно реагувати на їхні потреби та покращувати якість сервісу. Зі зростанням обсягів текстових даних, які генеруються щодня у вигляді коментарів, відгуків та оцінок, традиційні методи аналізу, ручна модерація чи ключові слова, стають недостатньо ефективними, трудомісткими та суб'єктивними. У цьому контексті актуальним є використання методів штучного інтелекту, зокрема глибокого навчання, що дозволяє автоматизувати обробку великої кількості неструктурованих текстів, виявляти приховані емоційні закономірності та забезпечувати точні прогнози щодо настроїв споживачів. Таким чином, застосування сучасних NLP-моделей для аналізу полярності відгуків відкриває нові можливості для підвищення ефективності бізнес-рішень у сфері електронної комерції.

Мета дослідження полягає у розробці технології автоматичної класифікації полярності текстових відгуків із використанням методів глибокого навчання для підвищення ефективності аналізу клієнтських коментарів на e-commerce платформах.

Завдання дослідження включають аналіз сучасних підходів до класифікації емоційної полярності текстів та можливостей науки про дані у цьому контексті, виявлення ключових особливостей текстових відгуків у сфері електронної комерції, побудову моделі для автоматичної класифікації полярності, оцінку її ефективності, розробку засобів для інтеграції моделі в прикладні системи та обґрунтування доцільності впровадження запропонованого рішення в реальні бізнес-процеси.

Об'єктом дослідження є процеси автоматичної класифікації емоційного забарвлення текстових відгуків користувачів у сфері електронної комерції.

Предметом дослідження є методи аналізу текстових даних, зокрема алгоритми машинного навчання, що застосовуються для автоматичної класифікації емоційної полярності відгуків у сфері електронної комерції.

Методи дослідження включали використання трансформерів для автоматичної класифікації текстів, застосування бібліотеки Matplotlib для візуалізації аналітики, а також розробку API на Flask і контейнеризацію додатку за допомогою Docker для забезпечення зручного розгортання.

Наукова новизна роботи полягає в удосконаленні аналізу емоційного забарвлення текстів шляхом поєднання лінгвістичного та семантичного підходів із використанням трансформерних моделей, зокрема компактної моделі TinyBERT, яка була адаптована до задачі бінарної класифікації полярності. Вперше здійснено інтеграцію цієї моделі в веб-сервіс, реалізований із використанням фреймворку Flask та контейнеризований за допомогою Docker з метою забезпечення портативності, масштабованості та спрощення процесу розгортання. Також проведено аналіз економічної доцільності хмарного розгортання, що дало змогу оцінити витрати на впровадження та подальшу експлуатацію системи.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРОБЛЕМИ ВИЗНАЧЕННЯ ЕМОЦІЙНОГО ЗАБАРВЛЕННЯ ТЕКСТІВ ТА АНАЛІЗ АКТУАЛЬНИХ ПІДХОДІВ ДО ЇЇ РОЗВ'ЯЗАННЯ

1.1 Аналіз проблеми

Сентимент-аналіз, або аналіз емоційного забарвлення тексту, є підзадачею обробки природної мови, яка включає як бінарну класифікацію емоцій на негативні та позитивні, так і багатокласову класифікацію на конкретні емоції - радість, сум, злість тощо. Застосування сентимент-аналізу охоплює широке коло сфер: від моніторингу громадської думки до вдосконалення сервісів підтримки клієнтів. Основна цінність таких систем полягає в здатності автоматично обробляти великі масиви неструктурованого тексту та виявляти настрої користувачів, чого неможливо досягти при обробці вручну на масштабах сучасних платформ [1].

Разом з тим, ця задача супроводжується низкою викликів: лексичною варіативністю, контекстуальною неоднозначністю, іронією, сарказмом, а також мовними спотвореннями (неформальні конструкції, скорочення, орфографічні помилки). Висока якість класифікації можлива лише за умови використання моделей, здатних враховувати контекст та структуру висловлювань, зокрема моделей глибокого навчання.

Однією з важливих сфер застосування сентимент-аналізу є електронна комерція. У 2025 році обсяг глобальної онлайн-торгівлі сягне 7,4 трильйона доларів США, що становить близько 24% усіх роздрібних продажів [2]. Понад 2,77 мільярда користувачів регулярно залишають відгуки та оцінки [3], що генерує значний масив даних, релевантний для аналізу споживчого досвіду.

Вивчення тональності таких відгуків дозволяє компаніям своєчасно реагувати на проблеми, адаптувати маркетингові стратегії та оптимізувати взаємодію з клієнтами. При цьому прості методи, як-от ручна перевірка чи підрахунок позитивних слів, є малоефективними на великих обсягах тексту. Крім того, ускладнення додає зростання ролі соціальної комерції, де відгуки

стають більш емоційно забарвленими, розмовними та скороченими.

У цьому контексті задача бінарної класифікації емоційного забарвлення англомовних текстових відгуків на e-commerce платформах є не лише науково цікавою, але й має практичну значущість. Її успішне вирішення забезпечує бізнесу актуальні інсайти, підвищує якість обслуговування та формує конкурентну перевагу на основі точного аналізу настроїв клієнтів [4].

1.2 Аналіз існуючих методів вирішення проблеми

1.2.1 Попередня обробка текстових даних

Ефективне вирішення задачі класифікації полярності текстів неможливе без ретельного попереднього опрацювання текстових даних. Природна мова, з її лексичними, синтаксичними і семантичними особливостями, є складним об'єктом для формального аналізу. Відтак, однією з ключових передумов побудови якісної системи класифікації є приведення текстової інформації до форми, придатної для обробки обчислювальними моделями [5].

Основними етапами обробки текстових даних є:

- нормалізація;
- токенізація;
- видалення стоп-слів;
- лематизація та стемінг;

Зрештою, варто зазначити, що правильна побудова конвеєру обробки тексту є важливою для досягнення високої точності класифікації полярності, тому розуміння кожного етапу є необхідним, а отже є потреба ознайомитись з кожним із них та розглянути відповідні методи вирішення поставленої задачі.

1. Нормалізація тексту

Нормалізація тексту є початковим етапом попередньої обробки в задачах NLP, що спрямована на зменшення лексичної варіативності та уніфікацію вхідних даних. Основні її кроки включають перетворення всіх символів у нижній регістр, видалення пунктуації, спеціальних символів, чисел, HTML-тегів та емодзі, а також очищення від надлишкових пробілів, символів нового

рядка та інших нерелевантних для аналізу символів. Часто нормалізація охоплює розширення скорочень (“don’t” => “do not”), виправлення орфографічних помилок та перетворення сленгових або неформальних скорочень (“btw” => “by the way”).

2. Токенізація

Токенізація є одним з найважливіших етапів попередньої обробки тексту, що полягає в розбитті суцільного текстового потоку на окремі одиниці - токени, які можуть бути словами, частинами слів, символами або фразами залежно від контексту задачі. У традиційних підходах вона базується на розділенні за пробілами та пунктуацією, однак у сучасних системах, особливо при використанні трансформерних моделей, таких як BERT, застосовуються більш гнучкі методи - зокрема, subword-токенізація за допомогою алгоритмів WordPiece або Byte-Pair Encoding (BPE). Ці підходи дозволяють зменшити кількість незнайомих слів (OOV) та краще працювати з морфологічною варіативністю, зберігаючи при цьому значущі мовні структури.

3. Видалення стоп-слів

Видалення стоп-слів - це один з можливих етапів попередньої обробки тексту, що спрямований на усунення службових лексем, які не несуть значущого семантичного навантаження (наприклад, “the”, “and”, “is”) і лише ускладнюють класифікацію, збільшуючи розмір словника та рівень шуму. Ця процедура сприяє зменшенню розмірності векторного простору й підвищенню ефективності моделей, однак не є універсальною: у задачах сентимент-аналізу окремі стоп-слова, як-от “not”, можуть бути критично важливими для точного розуміння полярності. Тому їх видалення варто адаптувати до контексту задачі, мови та предметної галузі, наприклад створюючи свої списки стоп-слів замість використання стандартних з бібліотек NLP.

4. Лематизація та стемінг

У попередній обробці тексту важливо звести слова до їхньої базової форми, що знижує варіативність лексичних одиниць і сприяє точнішому аналізу. Два основні методи для цього - лематизація та стемінг.

Лематизація перетворює слова на їхні лексичні корені, враховуючи граматичний контекст, що дає точніші результати, але вимагає більше обчислювальних ресурсів. Наприклад, слова “running”, “ran” і “runs” будуть зведені до леми “run”. Це підходить для завдань, де важлива точність семантики, таких як аналіз емоційного забарвлення текстів. Для задачі лематизації використовуються такі інструменти, як WordNet Lemmatizer, SpaCy Lemmatizer, Stanford Lemmatizer, Morfessor тощо.

Стемінг, з іншого боку, використовує простіші евристики для відсічення закінчень та зведення слова до кореня, що може призвести до втрати семантичної точності, але є швидшим і менш ресурсозатратним. Наприклад, слова “fishing” і “fisher” будуть зведені до “fish”, але інколи стемінг може призводити до неправильної нормалізації, наприклад “nationality” до “nation”, що призведе до неправильного розуміння семантичного значення слова. Найпоширенішими алгоритмами стемінгу є Porter Stemmer та Snowball Stemmer.

Вибір між стемінгом та лематизацією залежить від вимог до точності, швидкості та доступних ресурсів.

Завершуючи розгляд етапу попередньої обробки текстових даних, слід наголосити, що вибір конкретних процедур значною мірою залежить від типу застосовуваної моделі машинного навчання, цільової задачі та характеру вхідних даних. У класичних підходах, які ґрунтуються на векторизації тексту (наприклад, TF-IDF чи Bag-of-Words), доцільним є максимальне зниження лексичної варіативності, включно з видаленням стоп-слів, лематизацією тощо.

Однак при використанні сучасних моделей глибокого навчання, зокрема трансформерних архітектур на кшталт BERT, потреба в деяких традиційних кроках обробки суттєво зменшується, а іноді їх застосування може навіть зашкодити. Наприклад, видалення стоп-слів або лематизація можуть призвести до втрати важливого контексту, який модель здатна опрацювати самостійно. Аналогічно, заміна скорочень або надмірне спрощення синтаксичної структури може порушити логіку висловлювання, що вплине на якість аналізу. Отже,

попередня обробка повинна бути адаптована до конкретного типу моделі, з урахуванням її архітектурних особливостей та спроможності працювати з “сирим” текстом без шкоди для точності.

1.2.2 Векторизація та вкладання слів

Векторизація тексту - це процес перетворення текстових даних у числові вектори, що дозволяє використовувати їх для аналізу та обробки за допомогою алгоритмів машинного навчання. Методи векторизації, такі як Bag-of-Words і TF-IDF, перетворюють текст у вектори на основі частоти слів або їхньої важливості в корпусі, що є ефективним для простих задач. Однак ці методи не враховують глибші семантичні зв'язки між словами.

У порівнянні з ними, вкладання слів (англ. word embedding) надають контекстно-залежне представлення слів, що дозволяє моделювати складніші зв'язки і підвищує точність при вирішенні більш складних задач, таких як аналіз емоційного забарвлення чи класифікація текстів. Популярні алгоритми, як Word2Vec, GloVe та FastText, використовують багатовимірні вектори для кожного слова, враховуючи його контекст.

Вибір методу векторизації або вкладання слів залежить від конкретної задачі, обсягу даних і вимог до точності моделі. Векторизація є підходом для простіших завдань, тоді як вкладання слів дозволяє працювати з більш складними, контекстно-залежними зв'язками між словами.

1.2.3 Класичні методи машинного навчання

1. Наївний баєсів класифікатор

Наївний баєсівський класифікатор (англ. Naive Bayes Classifier) - це один із найпростіших і водночас ефективних методів машинного навчання для задач класифікації, особливо в обробці текстів, зокрема для аналізу емоційного забарвлення. Цей метод базується на теоремі Байєса, яка обчислює ймовірність належності певного прикладу до певного класу на основі апріорної ймовірності класу та ймовірності ознак у межах цього класу. Основне припущення моделі - умовна незалежність ознак одна від одної при заданому класі, звідси й термін -

наївний [6].

Переваги:

- простота реалізації та висока швидкість навчання;
- добра продуктивність на високовимірних даних;
- ефективність на невеликих вибірках;
- інтерпретованість результатів.

Недоліки:

- припущення про незалежність ознак часто не відповідає дійсності, особливо в NLP;
- може бути менш точним у порівнянні з методами, що враховують залежності між ознаками;
- чутливість до нечастих слів, якщо не застосовуються згладжування (наприклад, Лапласове згладжування).

2. Випадковий ліс

Випадковий ліс (англ. Random Forest) - це ансамблевий метод машинного навчання, який створює ансамбль дерев рішень, кожне з яких навчається на випадковій підмножині навчальних даних та з випадковим підмноженням ознак при розбитті вузлів. Остаточне рішення приймається більшістю голосів дерев (у класифікації) або середнім значенням (у регресії). Цей метод застосовують як для класифікації, так і для регресії [7].

Переваги:

- висока точність завдяки поєднанню багатьох дерев;
- стійкість до перенавчання: на відміну від окремих дерев, метод не схильний до перенавчання, навіть при великій кількості дерев;
- може працювати з числовими та категоріальними ознаками;
- підтримує оцінку важливості ознак, що дозволяє зрозуміти, які фактори впливають на рішення;
- стійкість до шуму та пропущених значень в даних.

Недоліки:

- велике споживання пам'яті та обчислювальних ресурсів, особливо

на великих наборах даних;

- мала інтерпретованість: важко візуалізувати чи пояснити рішення ансамблю дерев;
- повільне прогнозування при великій кількості дерев.

3. Логістична регресія

Логістична регресія є базовим, але ефективним методом машинного навчання, який широко застосовується для розв’язання задач бінарної класифікації. Метод оцінює ймовірність належності прикладу до певного класу, використовуючи логістичну функцію для перетворення лінійної комбінації ознак у значення в інтервалі [0, 1] [8].

Переваги:

- забезпечує високу ефективність на лінійно віддільних даних, зокрема у випадку векторизованого тексту;
- швидке навчання та передбачення, що дозволяє застосовувати модель у великих обчислювальних середовищах;
- простота реалізації та інтерпретації: коефіцієнти моделі мають чітке статистичне значення;
- добре працює з розрідженими та високорозмірними вхідними даними, типовими для задач обробки природної мови;
- підтримує регуляризацію (L1, L2) для контролю складності моделі та зменшення ризику перенавчання.

Недоліки:

- обмежена здатність до моделювання нелінійних зв’язків між ознаками та класами без додаткових перетворень;
- чутливість до мультиколінеарності між ознаками, що може впливати на стабільність оцінок;
- нижча продуктивність у порівнянні зі складнішими моделями (наприклад, ансамблевими або глибокими нейронними мережами) на даних із вираженою нелінійною структурою.

4. k-найближчих сусідів

Метод k-найближчих сусідів (k-NN) - непараметричний метод машинного навчання, використовуваний для класифікації та регресії. В обох випадках результат базується на найближчих k сусідах в навчальному наборі, але має різний принцип визначення результату, в залежності від задачі [8]:

- для класифікації: об'єкт відноситься до класу, який найбільш поширений серед його k-найближчих сусідів. Якщо k=1, об'єкт відноситься до класу одного сусіда;
- для регресії: результатом є середнє значення з k-найближчих сусідів.

Метод чутливий до локальної структури даних та вимірювань ознак, тому нормалізація даних може покращити точність. Для поліпшення результатів використовують зважування сусідів, де більше значення надається ближчим сусідам.

Переваги:

- не потребує етапу навчання, швидко адаптується до нових даних;
- підходить як для класифікації, так і для регресії;
- може ефективно класифікувати складні структури без явного моделювання;
- простий для розуміння і аналізу.

Недоліки:

- для кожного прогнозу потрібно обчислювати відстані до всіх навчальних зразків, що робить алгоритм повільним на великих наборах даних;
- різні метрики можуть значно вплинути на результат;
- погано працює з високорозмірними даними.

Для розуміння ефективності роботи розглянутих методів машинного навчання було проаналізовано роботу “Sentiment analysis based on machine learning models” [9], в якій проведено порівняння чотирьох класичних методів машинного навчання для задачі аналізу емоцій у контексті бінарної

класифікації. Для оцінки моделей використовувалися два набори даних: SST-2, на якому проводилося навчання, і IMDB, на якому проводилось тільки тестування моделей. Результати показали, що всі чотири моделі демонструють хороші результати на наборі даних SST-2, оскільки це дані, на яких моделі були навчені, проте при тестуванні на наборі даних IMDB, де дані є невідомими для моделей, спостерігається значна різниця в їхній продуктивності.

Модель k-найближчих сусідів (k-NN) показала найгірші результати на наборі даних IMDB, з точністю 0.50 та F1-мірою - 0.35. Це може бути пов'язано з перенавчанням моделі на наборі даних SST-2, що призводить до низької точності на нових даних. З детальним порівнянням характеристих моделей можна ознайомитись у таблицях 1 та 2.

Таблиця 1. Порівняння моделей на наборі даних SST-2

Модель	**Точність**	**Влучність**	**Повнота**	**F1-міра**
k-NN	0.88	0.88	0.88	0.87
Random Forest	0.89	0.89	0.89	0.89
Naive Bayes	0.87	0.87	0.87	0.87
Logistic Regression	0.88	0.88	0.88	0.88

Таблиця 2. Порівняння моделей на наборі даних IMDB

Модель	**Точність**	**Влучність**	**Повнота**	**F1-міра**
k-NN	0.50	0.54	0.50	0.35
Random Forest	0.67	0.68	0.67	0.67
Naive Bayes	0.80	0.80	0.80	0.80
Logistic Regression	0.81	0.81	0.81	0.81

Отже, результати дослідження демонструють, що хоча всі чотири моделі

можуть бути застосовані до задачі аналізу емоцій, k-NN та випадковий ліс можуть бути більш вразливими до перенавчання і менш ефективними при роботі з невідомими даними. Наївний баєс і Логістична регресія виявилися більш стабільними та надійними, особливо для бінарної класифікації емоцій на невідомих даних.

1.2.4 Методи глибокого навчання

У сфері обробки природної мови (NLP) для аналізу текстових даних активно використовуються три основні архітектури глибокого навчання: згорткові нейронні мережі (CNN), рекурентні нейронні мережі (RNN) та трансформери. Причина цього полягає у здатності цих архітектур ефективно моделювати складні залежності, властиві природній мові, яка, за своєю суттю, є послідовністю символів, слів, фраз і контекстів, організованих у структуровану послідовність.

Рекурентні нейронні мережі (RNN) були спеціально розроблені для роботи з послідовностями, що досягається за рахунок їх можливості зберігати внутрішній стан, який дозволяє враховувати інформацію про попередні елементи. Це робить їх придатними для аналізу текстів, де значення окремого слова залежить від контексту. Однак класичні RNN мають обмеження, зокрема проблему затухаючого градієнта, що ускладнює засвоєння довгострокових залежностей. Для її вирішення запропоновано вдосконалені моделі, такі як LSTM (Long Short-Term Memory) і GRU (Gated Recurrent Unit). Ефективність RNN значною мірою залежить від якості обраного способу векторного представлення слів (вкладення), адже саме ці вектори є вхідними ознаками, з яких модель формує уявлення про контекст [10].

Переваги:

- придатні для обробки послідовних даних з часовими або структурними залежностями;
- моделюють контекст шляхом урахування послідовності слів у тексті відповідно до їхнього порядку появи;

- LSTM і GRU здатні зберігати довготривалі залежності.

Недоліки:

- неможливість ефективного паралельного обчислення;
- складність навчання на довгих послідовностях;
- високі обчислювальні витрати при масштабуванні.

Згорткові нейронні мережі (CNN), попри початкову орієнтацію на обробку зображень, показали ефективність у задачах NLP, зокрема класифікації текстів. Вони добре виявляють локальні шаблони (наприклад, сталі фрази чи граматичні конструкції) та узагальнюють їх через шари згортки та субдискретизації. Як і у випадку з RNN, успішність застосування CNN до текстових задач значною мірою визначається типом використаних векторних представлень, які повинні ефективно відображати семантичні й синтаксичні властивості слів [11][12].

Переваги:

- висока швидкість обробки завдяки паралелізації;
- ефективне виявлення локальних залежностей;
- простота архітектури і налаштування.

Недоліки:

- обмеженість у моделюванні довгих залежностей;
- відсутність глобального контексту у представленні тексту.

Трансформери - новітня архітектура, що була запропонована у статті “Attention Is All You Need” [13]. В них повністю відмовилися від рекурентних структур і замість цього використовують механізм самоуваги (англ. self-attention), який дозволяє одночасно враховувати зв’язки між усіма токенами в послідовності. У 2018 році з'явилася модель BERT (Bidirectional Encoder Representations from Transformers), що реалізує двонаправлене контекстуальне представлення та стала проривом у сфері семантичного аналізу [14]. Подальші вдосконалення, зокрема RoBERTa [15] та DeBERTa [16], поліпшили точність у багатьох задачах, включно з класифікацією, узагальненням, виявленням сутностей тощо.

Особливої уваги заслуговує модель TinyBERT - компактна варіація оригінального BERT, розроблена для застосунків з обмеженими обчислювальними ресурсами. Незважаючи на незначне зниження точності порівняно з повнорозмірними моделями, TinyBERT забезпечує суттєве підвищення швидкості інференції та зменшення обсягу необхідної пам’яті, що робить її придатною для розгортання на мобільних пристроях та в режимі реального часу [17].

Вершиною розвитку трансформерів на даний момент є модель T-ULRv6, представлена Microsoft у 2022 році [18]. Вона стала однією з найефективніших у сфері обробки природної мови, очоливши GLUE-рейтинг [19], що є стандартом для оцінки мовних моделей за сукупністю завдань. Це свідчить про її високу здатність до узагальнення та точність у вирішенні різноманітних NLP-задач.

Переваги:

- моделювання довготривалих залежностей незалежно від їхнього розташування;
- паралельна обробка токенів - висока продуктивність;
- висока точність у широкому спектрі NLP-задач.

Недоліки:

- навчання на великих наборах даних триває години, а іноді дні, в залежності від потужності системи;
- потреба в значних обчислювальних ресурсах - сучасні GPU або TPU, як для навчання так і для прогнозуванн;
- складність тонкого настроювання та інтерпретації результатів.

Для кращого розуміння ефективності розглянутих моделей глибокого навчання було проаналізовано ряд робіт, де для представників описаних архітектур проводиться тестування на популярних бенчмарк наборах даних.

Аналіз показників точності, наведених у таблиці 3, на датасеті SST-2 (Stanford Sentiment Treebank v2), який є популярним для перевірки моделей у задачах бінарної класифікації тексту за емоційним забарвленням, демонструє

чітку еволюцію ефективності архітектур глибокого навчання в обробці природної мови. Найпростіші з розглянутих моделей - CNN (91.2%) та RNN, а саме її модифікація LSTM (92%) - показують пристойну точність, що свідчить про їхню здатність моделювати контекст та виявляти релевантні текстові патерни. Проте їхня ефективність поступається сучасним трансформерним моделям.

Застосування архітектури Transformer, навіть у компактній версії TinyBERT4, вже забезпечує точність 92.6%, незначно випереджаючи LSTM. Повноцінна модель BERT у базовій ($BERT_{BASE}$, 93.5%) та великій конфігураціях ($BERT_{LARGE}$, 94.9%) демонструє помітне зростання продуктивності. Подальші вдосконалення, зокрема RoBERTa (96.4%) та DeBERTa (97.5%), відображають прогрес у покращенні механізмів уваги та попереднього навчання, що сприяє глибшому розумінню семантики тексту. Найвищий результат - 97.5% - показує модель T-ULRv6, яка наразі є однією з найбільш потужних у контексті універсального представлення природної мови.

Таблиця 3. Порівняння основних архітектур моделей глибокого навчання та їх підвидів на наборі даних SST-2

Модель	**Точність**
CNN	91.2
RNN (LSTM)	92.0
Transformer ($TinyBERT_4$)	92.6
Transformer ($BERT_{BASE}$)	93.5
Transformer ($BERT_{LARGE}$)	94.9
Transformer (RoBERTa)	96.4
Transformer (DeBERTa)	97.5
Transformer (T-ULRv6)	97.5

Варто підкреслити, що хоча модифікації BERT та T-ULRv6

демонструють значне зростання точності, це супроводжується істотним збільшенням обчислювальних витрат. Їх навчання, інференція та зберігання потребують значних апаратних ресурсів, що може бути критичним чинником при їх використанні в реальних або обмежених середовищах.

Загалом результати свідчать про те, що сучасні трансформерні моделі суттєво переважають класичні методи машинного навчання та навіть RNN/CNN, насамперед завдяки здатності одночасно враховувати складні залежності між словами та моделювати глобальний контекст у тексті. Їх складна структура, багаторівневі механізми уваги та глибоке попереднє навчання забезпечують якісно новий рівень семантичного аналізу, що особливо важливо в задачах сентимент аналізу.

1.3 Вибір методології для проєкта аналізу даних

1.3.1 CRISP-DM

CRISP-DM (Cross-Industry Standard Process for Data Mining) - це загальноприйнята методологія аналізу даних. Вона забезпечує структурований і гнучкий підхід до побудови моделей та ухвалення рішень на основі даних, широко використовується в бізнесі, науці й індустрії, незалежно від галузі [20].

CRISP-DM базується на 6 етапах (рисунок 1):

1) розуміння бізнесу - формулювання цілей проєкту з урахуванням бізнес-контексту та потреб, постановка задач аналізу даних відповідно до бізнес-проблеми;
2) розуміння даних - збір, знайомство з даними, первинний аналіз, виявлення проблем якості та вивчення основних характеристик;
3) підготовка даних - очищення, перетворення, вибір релевантних ознак і формування остаточного датасету для побудови моделі;
4) моделювання - вибір алгоритмів, навчання моделей, налаштування параметрів і оцінка початкових результатів;
5) оцінка - інтерпретація моделей, перевірка їх відповідності бізнес-цілям, прийняття рішення щодо подальших дій;

6) впровадження - інтеграція результатів моделювання в практичні бізнес-процеси, створення звітів або автоматизованих рішень.

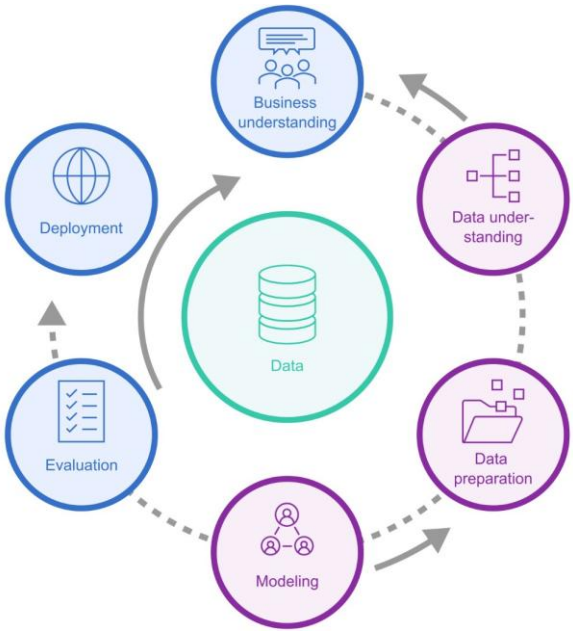


Рисунок 1. Діаграма життєвого циклу CRISP-DM

Переваги:

- надає чітку покрокову структуру процесу аналізу даних;
- добре підходить для ітеративного вдосконалення моделей;
- сумісний з різними інструментами та технологіями;
- сприяє ефективній комунікації між аналітиками та бізнесом;
- широко визнаний і підтримується у спільноті.

Недоліки:

- не враховує специфіку сучасних гнучких методологій;
- не включає автоматизацію або управління потоками даних;
- недостатньо детально описує етапи після впровадження моделі.

1.3.2 KDD

KDD (Knowledge Discovery in Databases) - це процес виявлення знань у великих базах даних, що включає серію етапів для перетворення сирих даних у корисну інформацію [21].

Кроки KDD:

1) вибір даних - визначення відповідних даних для аналізу з бази даних;
2) попередня обробка даних - очищення даних від шуму та неповних записів;
3) трансформація даних - перетворення даних у форму, зручну для аналізу;
4) моделювання - використання алгоритмів для пошуку патернів або створення моделей;
5) оцінка - аналіз отриманих результатів і перевірка їх на корисність;
6) представлення знань - візуалізація та інтерпретація результатів для кінцевих користувачів.

Переваги:

- підходить для великих обсягів даних і складних проблем;
- дозволяє інтегрувати різні джерела даних;
- орієнтований на виявлення знань і патернів у даних;
- добре підходить для багатьох галузей, таких як бізнес та наука;
- має чітку послідовність етапів для обробки даних.

Недоліки:

- процес може бути дуже довгим;
- потребує значних ресурсів для обробки великих даних;
- складність у виборі правильних алгоритмів для конкретних задач;
- інтерпретація результатів може бути складною без спеціалізованих знань.

1.3.3 SEMMA

SEMMA (Sample, Explore, Modify, Model, Assess) - це методологія, розроблена компанією SAS для процесу аналізу даних, особливо в контексті використання SAS Enterprise Miner. Вона фокусується на етапах обробки даних і моделювання [22].

Основні кроки SEMMA:

1) вибірка - вибір підмножини даних для аналізу, щоб зменшити обсяг обробки та покращити ефективність;
2) дослідження - аналіз даних для виявлення патернів, аномалій і взаємозв’язків між змінними;
3) модифікація - очищення і трансформація даних, щоб підготувати їх до моделювання (наприклад, нормалізація, заповнення пропусків);
4) моделювання - створення моделей машинного навчання для прогнозування або класифікації даних;
5) оцінка - оцінка ефективності моделей за допомогою різних метрик та вибір найкращої моделі для подальшого використання.

Переваги:

- чітка і структурована методологія, що полегшує процес аналізу даних;
- дозволяє ефективно обробляти великі обсяги даних;
- добре підходить для вирішення завдань прогнозування і класифікації;
- включає детальні етапи для підготовки та трансформації даних;
- підтримується багатьма популярними інструментами для аналізу даних.

Недоліки:

- може бути недостатньо гнучким для більш складних або нестандартних завдань;
- акцентує увагу на технічних аспектах, але не завжди враховує специфіку бізнес-процесів;
- потребує значних обчислювальних ресурсів, особливо при роботі з великими даними;
- автоматизація процесів може бути обмеженою.

1.3.4 OSEMN

Osemn (obtain, scrub, explore, model, interpret) - це методологія для аналізу даних, яка зосереджена на практичних етапах роботи з даними та їх інтерпретації [23].

Основні кроки osemn:

1) отримання - збір даних з різних джерел (бази даних, api, файли тощо).
2) очищення - обробка та очищення даних (видалення пропусків, виправлення помилок, нормалізація).
3) дослідження - аналіз даних для виявлення патернів, аномалій, кореляцій та інших важливих характеристик.
4) моделювання - побудова моделей машинного навчання для прогнозування або класифікації.
5) інтерпретація - представлення результатів моделювання, візуалізація та надання рекомендацій для прийняття рішень.

Переваги:

- забезпечує чітку і логічну послідовність етапів для обробки даних;
- орієнтований на отримання максимального знання з даних за допомогою аналізу та візуалізації;
- дозволяє інтегрувати різні джерела даних і використовувати їх для глибокого аналізу;
- забезпечує ефективну комунікацію між аналітиками та бізнес-підрозділами.

Недоліки:

- може бути складним для початківців через широкий спектр етапів і технік;
- вимагає висококваліфікованих аналітиків для коректної реалізації етапів;
- потребує значних ресурсів для обробки великих обсягів даних;
- процес може бути часозатратним, особливо на етапах аналізу і

моделювання.

Проаналізувавши переваги та недоліки всіх моделей, можна стверджувати, що CRISP-DM є найбільш підходящою методологією для задачі проєкту аналізу даних. Ця методологія перевірена часом і широко використовується у різних галузях, що свідчить про її ефективність та універсальність. CRISP-DM забезпечує чітку структурованість процесу і дозволяє інтегрувати бізнес-вимоги на кожному етапі аналізу, що робить її більш адаптованою до реальних умов. У порівнянні з іншими методологіями, CRISP-DM більш гнучка та орієнтована на результат, що робить її ідеальною для розробки моделі аналізу емоційного забарвлення.

1.4 Постановка задачі

Задача аналізу даних полягає у розробці системи для автоматичної класифікації полярності текстових коментарів, що зібрані на різноманітних онлайн-платформах. Це важлива задача в контексті електронної комерції та інших сфер, де відгуки споживачів відіграють ключову роль у формуванні іміджу бренду, а також у поліпшенні клієнтського сервісу. Завдання проєкта має на меті точне визначення, чи є коментар позитивним, або негативним, що дозволить у випадку інтеграції даної моделі оперативно реагувати на критичні ситуації, поліпшувати продукцію та адаптувати маркетингові стратегії.

Як інструмент вирішення поставленої задачі було обрано TinyBERT, яка є компактною та швидкою версією BERT, розробленою для ефективної роботи з обмеженими ресурсами при збереженні високої точності. TinyBERT є оптимальним вибором для завдань, де важлива швидкість обробки даних та масштабованість, зокрема для аналізу великих обсягів текстових даних на реальних онлайн-платформах. Завдяки використанню механізму самоуваги та глибокому попередньому навчанні на великих наборах даних, TinyBERT дозволяє досягти високої точності класифікації, що критично для задачі сентимент-аналізу.

Застосування методу CRISP-DM для розв'язання цієї задачі дозволить

ефективно структурувати процес розробки моделі на всіх етапах: від розуміння бізнес-проблеми та підготовки даних до побудови та впровадження моделі. Цей підхід забезпечує чітку та послідовну організацію роботи, що дозволяє адаптувати модель до конкретних потреб бізнесу та досягати високих результатів у практичному застосуванні. CRISP-DM також дає змогу гнучко коригувати модель під час тестування та виведення, що критично для розв'язання проблем у реальному часі.

Висновки

У цьому розділі було здійснено глибокий аналіз проблеми автоматичної класифікації полярності відгуків на e-commerce платформах, що є важливим аспектом для поліпшення маркетингових стратегій та забезпечення оперативного реагування на думки користувачів. З огляду на специфіку даної задачі, було розглянуто основні етапи обробки текстових даних, включаючи попередню обробку тексту, методи векторизації та вкладання слів, а також порівняння класичних методів машинного навчання та методів глибокого навчання, таких як рекурентні нейронні мережі, згорткові нейронні мережі та трансформери.

Аналіз існуючих методологій показав, що, хоча класичні методи машинного навчання дають можливість ефективно вирішувати базові задачі класифікації тексту, сучасні методи глибокого навчання, зокрема трансформери, демонструють значно вищу точність завдяки здатності моделювати складні контекстуальні залежності та враховувати багатозначність слів. Трансформери, як-от BERT і його модифікації, зарекомендували себе як найефективніші для задач аналізу тексту, хоча вони потребують значних обчислювальних ресурсів.

Вибір методології для цього проєкту ґрунтується на використанні моделі TinyBERT, яка є оптимальним компромісом між точністю та обчислювальними вимогами. Зважаючи на здатність TinyBERT зберігати переваги BERT при меншій кількості параметрів, ця модель є чудовим варіантом для автоматизації

класифікації відгуків у реальних умовах, де швидкість та ефективність є важливими факторами.

Отже, на основі проведеного аналізу було сформульовано задачу автоматичної класифікації полярності відгуків з використанням глибоких нейронних мереж, що дозволяє ефективно вирішити проблему автоматизації аналізу емоційного забарвлення текстів, зокрема на e-commerce платформах.

РОЗДІЛ 2. МЕТОДИ ТА МОДЕЛІ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ ВИЗНАЧЕННЯ ЕМОЦІЙНОГО ЗАБАРВЛЕННЯ ТЕКСТОВОГО КОНТЕНТУ

2.1 Штучна нейронна мережа

Штучна нейронна мережа (ШНМ) - це математична модель, натхненна структурою та функціонуванням біологічних нейронних мереж, що складаються з нейронів у мозку. Вона є основою для багатьох алгоритмів глибокого навчання і використовується для вирішення складних задач, таких як класифікація, регресія, розпізнавання образів, аналіз тексту тощо [24].

Штучна нейронна мережа складається з кількох основних компонентів: нейронів (або вузлів), зв'язків між ними (які мають ваги), а також функцій активації, що дозволяють моделювати нелінійні залежності між вхідними та вихідними даними. Найпростішою формою є одношарова нейронна мережа, яка складається з одного шару нейронів, але для складніших задач використовуються багатошарові нейронні мережі (так звані глибокі нейронні мережі).

Основні елементи нейронної мережі, які також зображені на рисунку 2:

1) вхідний шар - цей шар приймає дані, які надходять у вигляді числових значень (наприклад, векторів слів або пікселів зображення). Кожен елемент вхідного шару представляє окрему характеристику вхідних даних;
2) приховані шари - це шари, які знаходяться між вхідним та вихідним шарами. Вони обробляють інформацію, виконуючи математичні операції (наприклад, лінійні перетворення через ваги та зсуви, а також нелінійні перетворення через функції активації). Чим більше прихованих шарів, тим складніше може бути модель і її здатність виявляти складні патерни у даних;
3) вихідний шар - цей шар генерує кінцеві результати на основі інформації, обробленої через приховані шари. У задачах

класифікації, наприклад, вихідний шар може виводити ймовірність належності кожного об'єкта до певної категорії (класу).

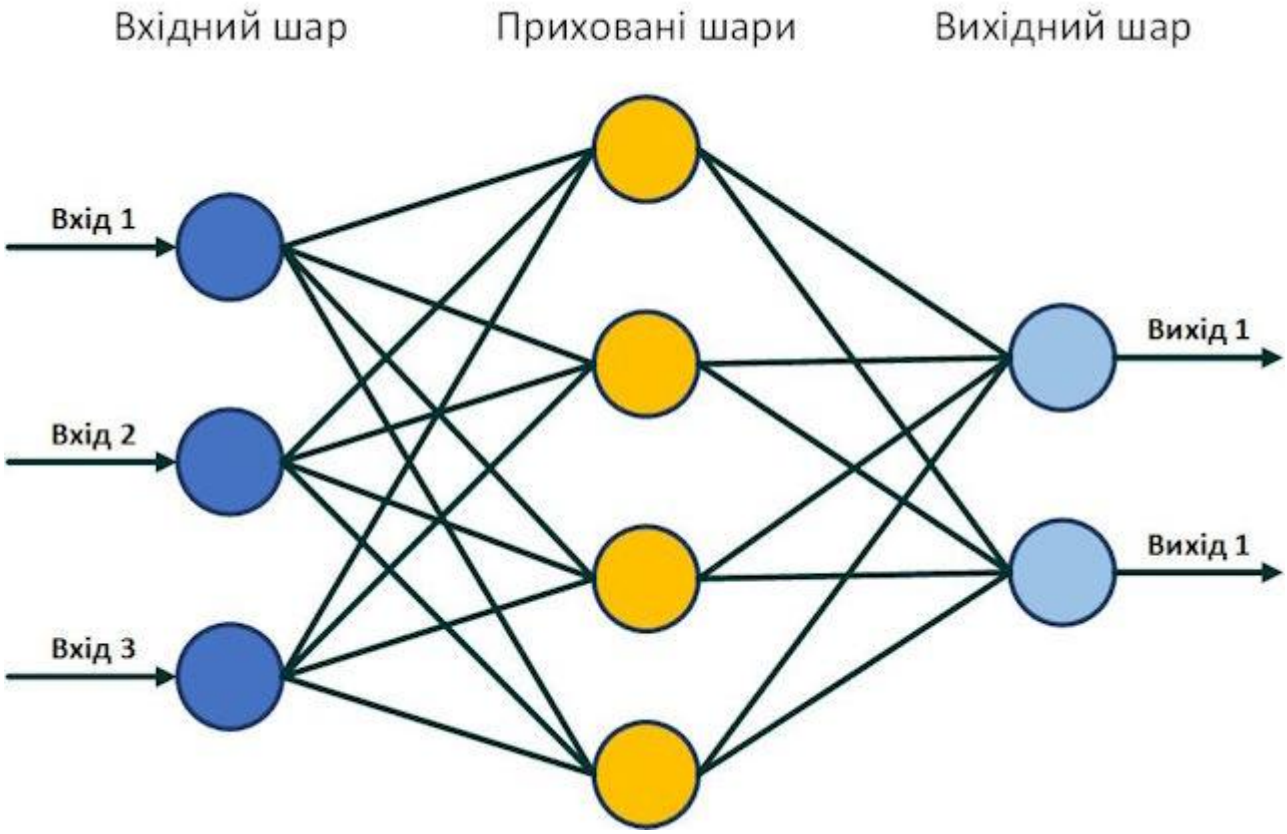


Рисунок 2. Схема штучної нейронної мережі

Ваги та зсуви є основними параметрами штучної нейронної мережі, які визначають, як інформація передається та обробляється на кожному етапі в обчислювальному процесі.

Ваги - це коефіцієнти, що визначають важливість кожного входу (вхідних значень) для конкретного нейрона. У мережах із багатьма шарами ці ваги регулюють, як сильно вхідні дані повинні впливати на результат, обчислений нейроном. Кожен зв'язок між нейронами має свою вагу, яка коригує значення, що проходить через цей зв'язок.

Математично це можна уявити так: якщо x - це вхід нейрона, а w - це вага, то вхід на нейрон можна представити як добуток $x \cdot w$. Тобто кожен вхід, помножений на свою вагу, буде мати певний вплив на результат нейрона.

Процес навчання нейронної мережі включає коригування цих ваг з метою

мінімізації помилки між передбаченими та реальними результатами.

Зсув - це додатковий параметр, який додається до вхідної інформації після її множення на ваги. Зсув дозволяє моделі робити корекції і допомагає в побудові більш гнучкої моделі. Завдяки зсуву нейронна мережа може навчитися не тільки приймати рішення на основі вхідних значень, але й додавати константний вплив, незалежний від вхідних даних.

Математично зсув можна представити як додавання константного значення b до лінійної комбінації входів і ваг:

$$y = (x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots + x_n \cdot w_n) + b, \quad (2.1)$$

де $x_1, x_2, \ldots, x_n$- вхідні значення;

$w_1, w_2, \ldots, w_n$ - ваги;

b - зсув.

Зсув дозволяє моделі здійснювати обчислення з різними порогами для різних входів, що дає змогу вирішувати задачі з більшою гнучкістю, наприклад, приймати рішення, навіть коли всі входи мають значення, близькі до нуля.

У процесі навчання нейронної мережі ваги та зсуви коригуються таким чином, щоб мінімізувати похибку моделі (відмінність між прогнозом і фактичним результатом). Це зазвичай досягається за допомогою алгоритмів оптимізації, таких як метод зворотного поширення помилки (backpropagation), де помилка використовується для оновлення ваг і зсувів шляхом градієнтного спуску.

Функція активації - це математична функція, яка визначає вихід нейрона штучної нейронної мережі на основі його вхідних даних. Вона є важливою частиною нейронної мережі, оскільки дозволяє мережі моделювати складні нелінійні залежності між вхідними даними і виходом, що є необхідним для вирішення складних задач, таких як розпізнавання образів, обробка тексту або передбачення. Без нелінійних функцій активації мережа складалася б лише з лінійних перетворень, що значно обмежило б її здатність моделювати складні патерни та структури.

Функції активації застосовуються після лінійного комбінаційного процесу, коли на нейрон подають суму зважених входів (вхідні дані помножені на ваги), до яких додається зсув. Функція активації визначає, як нейрон реагуватиме на ці вхідні значення [25].

Основні типи функцій активації:

1) Сигмоїда:

$$\sigma(x) = \frac{1}{1+e^{-x}}, \tag{2.2}$$

де x - значення вхідного параметра;

e - число Ейлера.

Сигмоїдальна функція приймає значення в межах від 0 до 1, що робить її корисною для задач бінарна класифікації. Однак її недолік - проблема затухаючого градієнта для великих або малих значень входу, що ускладнює навчання.

2) Гіперболічний тангенс:

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \tag{2.3}$$

де x - значення вхідного параметра;

e - число Ейлера.

Функція тангенса є симетричною щодо нуля і приймає значення від -1 до 1. Вона є поліпшеною версією сигмоїди, оскільки має кращі властивості для обробки від'ємних значень. Однак також має проблему затухаючого градієнта.

3) ReLU (Rectified Linear Unit):

$$ReLU(x) = max(0, x), \tag{2.4}$$

де x - значення вхідного параметра.

ReLU є однією з найбільш популярних функцій активації завдяки своїй простоті та ефективності. Вона повертає нуль для від'ємних значень

і саму величину для позитивних, що дозволяє швидше навчатися та зменшує проблему затухаючого градієнта. Однак для значень, що постійно залишаються від'ємними, може виникнути проблема "мертвих нейронів" (коли нейрон більше не оновлюється).

4) Leaky ReLU:

$$Leaky\ ReLU(x) = max(\alpha x, x) \quad (2.5)$$

де x - значення вхідного параметра;

α - дуже маленька константа, зі значенням близьким до нуля.

Leaky ReLU - це варіант ReLU, який дозволяє не нульовий вихід для від'ємних значень. Leaky ReLU допомагає зменшити проблему мертвих нейронів, дозволяючи малий, але постійний потік градієнта через від'ємні значення

5) Softmax:

$$Softmax(x_i) = \frac{e^{x_i}}{\Sigma_j e^{x_j}}, \quad (2.6)$$

де x_i - і-й аргумент вхідного вектора x, тобто сирий вихід моделі для і-го класу;

e - число Ейлера;

i - індекс поточного класу, для якого обчислюється softmax-ймовірність;

j - індекс, що використовується в знаменнику при підсумовуванні по всіх класах, щоб нормалізувати значення в межах [0, 1].

Функція softmax використовується в багатокласовій класифікації, де її виходи представляють ймовірності для кожного класу. Вона перетворює лінійні виходи нейрона в значення між 0 і 1, так що їх сума дорівнює 1, що дозволяє інтерпретувати виходи як ймовірності.

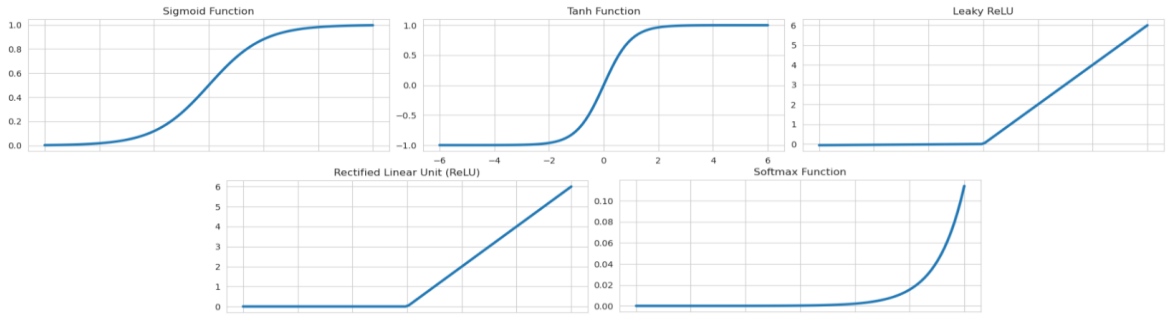


Рисунок 3. Графіки функцій активації

2.2 Трансформер

Трансформер (Transformer) - це революційна архітектура нейронної мережі, яка була представлена у статті “Attention Is All You Need” у 2017 році [13]. Вона стала основою для багатьох сучасних моделей обробки природної мови (NLP) і значно покращила результати у порівнянні з попередніми архітектурами, такими як рекурентні нейронні мережі (RNN) та згорткові нейронні мережі (CNN). Відмінною рисою трансформера є повна відмова від рекурентних структур, що дало змогу значно підвищити ефективність і швидкість обробки даних, особливо на великих обсягах текстових даних.

Основною інновацією трансформера є використання механізму самоуваги, який дозволяє моделі враховувати всі слова в послідовності незалежно від їхнього порядку. На відміну від RNN, де кожен елемент послідовності залежить від попередніх, трансформер може паралельно обробляти весь текст, що значно знижує час обчислень. Механізм самоуваги дозволяє кожному слову в тексті звертати увагу на всі інші слова, зокрема на ті, що є важливими для розуміння контексту. Наприклад, у фразах типу "the bank of the river" і "the bank of the city", механізм самоуваги допомагає правильно інтерпретувати різницю в значенні слова "bank" залежно від контексту.

Трансформер складається з двох основних компонентів: енкодера (encoder) і декодера (decoder), хоча для деяких застосувань (наприклад, в BERT або GPT) використовується лише один з них.

Структурно трансформер складається з двох симетричних частин:

енкодера та декодера, кожна з яких побудована з багатьох ідентичних шарів, що включають низку фундаментальних компонентів. На вході модель приймає токенізований текст, який проходить через шар векторного представлення слів (англ. embedding layer). Оскільки трансформер не має вбудованого механізму порядку слів, до векторів додаються позиційні кодування (англ. positional encoding), які фіксують порядок елементів у послідовності. Схематично архітектура зображена на рисунку 4.

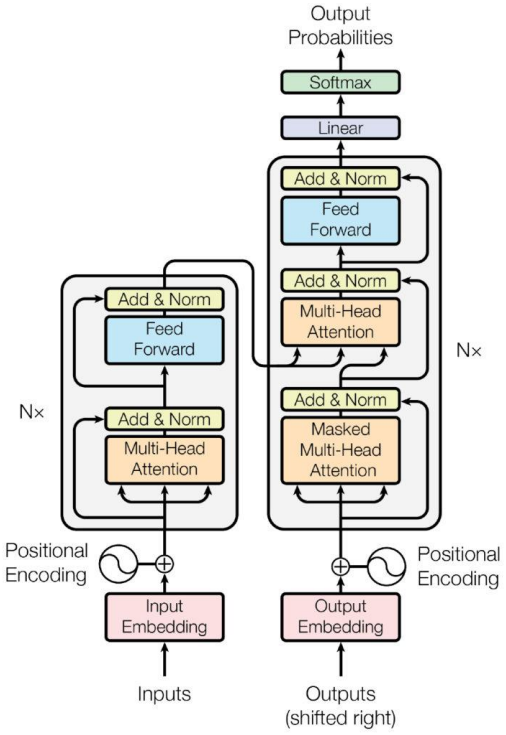


Рисунок 4. Схема архітектури трансформер

2.2.1 Embedding Layer

У трансформері кожен вхідний токен перетворюється на вектор фіксованої розмірності. Кожен токен з послідовності перетворюється на вектор за допомогою матриці векторів словника.

$$E(x) = W_{e^x}, \tag{2.7}$$

де $E(x)$ векторне представлення для токену x;

W_{e^x} - матриця векторів, яка відповідає за перетворення токену в його ембединг;

x - індекс токену в словнику.

2.2.2 Positional Encoding

Трансформер не має здатності обробляти порядок елементів у послідовності, як це роблять RNN або CNN. Тому для врахування порядку слів додається позиційне кодування. Позиційне кодування додається до векторів токенів, щоб моделі не було потрібно окремо вивчати позиції.

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}), \quad (2.8)$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}}), \quad (2.9)$$

де $PE_{(pos,i)}$ - позиційне кодування для позиції pos та індексу i;

pos - позиція токену у послідовності;

d - розмірність простору ембеддінгу (наприклад, 512).

2.2.3 Scaled Dot-Product Attention

Механізм уваги (англ. attention) дозволяє кожному токену в послідовності враховувати інші токени, зважаючи на їх контекст. У трансформері використовується механізм багатоголової уваги, який дозволяє ефективніше обробляти інформацію в кількох підпросторах.

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V, \quad (2.10)$$

де Q - матриця запитів, сформована з вхідної послідовності (наприклад, токенів);

K - матриця ключів, яка також створюється з тієї ж або іншої послідовності залежно від типу уваги (self-attention або encoder-decoder attention);

V - матриця значень, пов’язаних з ключами;

d_k - розмірність простору запитів та ключів.

Схема механізму уваги зображена на рисунку 5.

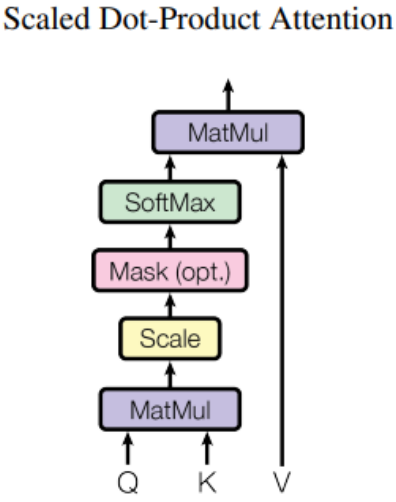


Рисунок 5. Схема механізму уваги

2.2.4 Multi-Head Self-Attention

Механізм багатоголової уваги дозволяє виконувати декілька паралельних операцій уваги, зберігаючи різні "перспективи" на вхідну послідовність.

$$MultiHead(Q,K,V) = Concat(head_1, \dots, head_h)W^O, \quad (2.11)$$

де $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$;

h - кількість голів уваги;

W^O - матриця для лінійної комбінації результатів усіх голів.

Схема механізму багатоголової уваги зображено на рисунку 6.

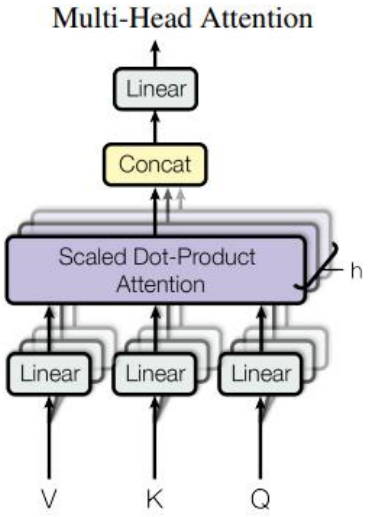


Рисунок 6. Схема механізму багатоголової уваги

2.2.5 Add & Layer Normalization

Після кожного підшару застосовується залишковий зв'язок (residual connection), що дозволяє уникнути проблеми згаснення градієнта. Крім того, проводиться нормалізація шару.

$$LayerNorm(x + Sublayer(x)) = \frac{x + Sublayer(x) - \mu}{\sigma} \cdot \gamma + \beta, \qquad (2.12)$$

де x - вхідний вектор,

$Sublayer(x)$ - результат підшару (наприклад, механізм уваги або FFN);

μ, σ - середнє та стандартне відхилення для нормалізації;

γ, β - параметри для масштабування та зміщення.

2.2.6 Feed-Forward Neural Network

У кожному шарі трансформера застосовується мережа FFN, яка є двошаровою нейронною мережею з нелінійною активацією. Це базовий тип штучної нейронної мережі, в якій дані проходять лише в одному напрямку: від вхідного шару через один або кілька прихованих шарів до вихідного, без зворотних зв’язків. FFN використовується для класифікації, регресії та інших задач, де зв’язки між шарами не потребують пам’яті про попередні стани (на відміну від рекурентних мереж).

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2, \qquad (2.13)$$

де W_1, W_2 - ваги двох лінійних шарів;

b_1, b_2 - зміщення;

$max(0, x)$ - функція активації ReLU.

2.2.7 Masked Multi-Head Self-Attention

В декодері також використовується механізм баготоголової уваги, але з маскуванням, щоб кожен токен не бачив майбутні токени. Це дозволяє генерацію послідовностей в умовах автогресивного моделювання.

$$Attention(Q, K, V) = softmax(\frac{QK^T + M}{\sqrt{d_k}})V, \qquad (2.14)$$

де Q - матриця запитів (queries);

K - матриця ключів (keys);

M - маска, яка дозволяє токенам бачити лише попередні слова;

V - матриця значень (values);

d_k - розмірність векторів запитів та ключів.

2.2.8 Encoder-Decoder Attention

Цей етап дозволяє декодеру зважати на вихід енкодера при генерації нових токенів.

$$Attention(Q, K_{enc}, V_{enc}) = softmax(\frac{QK_{enc}{}^T}{\sqrt{d_k}})V_{enc}, \quad (2.15)$$

де Q - матриця запитів (queries);

K_{enc}, V_{enc} - ключі та значення, отримані з енкодера;

V - матриця значень (values);

d_k - розмірність векторів запитів та ключів.

2.2.9 Output Layer

На виході декодера застосовується лінійна трансформація для отримання ймовірностей для кожного токену в словнику.

$$y = softmax(W_o h_{dec} + b_o), \quad (2.16)$$

де h_{dec} - вихід декодера;

W_o - матриця ваг для проекції;

b_o - зміщення.

Популярні моделі, що базуються на архітектурі трансформера, включають BERT, GPT, RoBERTa, T5 та інші. Всі ці моделі продемонстрували значні покращення результатів на різних NLP задачах, таких як аналіз емоційного забарвлення, текстовий аналіз, машинний переклад, а також розуміння контексту.

Однак, незважаючи на свої переваги, трансформери мають і деякі недоліки. Вони вимагають значних обчислювальних ресурсів для навчання, особливо у разі великих моделей, таких як BERT або T-ULRv6. Потрібно багато

пам'яті та обчислювальних потужностей для обробки великих обсягів даних, що робить трансформери менш доступними для невеликих проєктів або організацій з обмеженими ресурсами.

2.3 BERT

BERT (Bidirectional Encoder Representations from Transformers) - це глибока попередньо натренована мовна модель, створена дослідниками Google у 2018 році [14]. Її основна мета - забезпечити універсальне контекстуальне подання слів для широкого кола задач обробки природної мови (NLP), таких як класифікація тексту, розпізнавання іменованих сутностей, відповіді на запитання та інші.

На відміну від традиційного трансформера, BERT:

- BERT не має декодера і не використовується для генерації тексту, тобто використовує лише енкодерну частину;
- реалізує механізм двобічної уваги (bidirectional attention) - модель має доступ одночасно до лівого та правого контексту кожного токена, що принципово відрізняє її від авторегресійних моделей, які навчаються прогнозувати наступні слова лише зліва направо;
- навчається за допомогою спеціально сконструйованих завдань, що забезпечують глибше мовне розуміння.

Основні особливості BERT:

1) Масковане мовне моделювання (Masked Language Modeling, MLM)

Під час попереднього навчання модель отримує речення, де деякі токени замінено на спеціальний символ [MASK]. Завдання моделі - передбачити масковані токени, використовуючи контекст як зліва так і справа.

$$L_{MLM} = -\sum_{i \in M} logP(x_i|x_{\setminus i}), \qquad (2.17)$$

де M - множина позицій (індексів) у вхідній послідовності, які були замасковані (наприклад, замінені на [MASK]) під час навчання;

i - індекс у множині M, конкретна позиція (токен), для якої модель

намагається передбачити правильне слово на основі контексту;

$P(\ldots)$ - модельна оцінка ймовірності правильного слова;

x_i - правильний токен, який намагаються передбачити;

$x_{\setminus i}$ - вхідна послідовність без токена на позиції i (маскований контекст.

2) Передбачення наступного речення (Next Sentence Prediction, NSP)

Моделі подається пара речень A і B, і вона має визначити, чи йде речення B безпосередньо після речення A в оригінальному тексті. Це завдання дозволяє моделі опанувати елементарні дискурсивні зв'язки.

$$L_{NSP} = -ylog(p) - (1-y)log(1-p), \qquad (2.18)$$

де y - істинна мітка, що дорівнює 1, якщо друге речення дійсно йде після першого у вихідному тексті (тобто вони зв'язані), або 0, якщо друге речення випадкове (вибране з іншої частини корпусу);

p - ймовірність, що модель вважає речення зв'язаними, тобто що друге речення логічно слідує за першим. Ця ймовірність отримується застосуванням сигмоїдальної функції до лінійної трансформації вектора [CLS], який представляє всю послідовність.

$$p = \sigma(w^T h_{[CLS]} + b), \qquad (2.19)$$

де $h_{[CLS]}$ - вектор прихованого стану токена [CLS] після проходження всіх шарів енкодера;

w, b - вага і зсув класифікатора;

σ(...) - сигмоїдальна функція.

3) [CLS]-токен як агреговане подання

На початку кожної послідовності додається спеціальний токен [CLS], вектор якого після проходження через всі шари використовується як узагальнене представлення вхідного тексту.

Для задач класифікації саме цей вектор подається до вихідного класифікатора.

4) Попередньо тренована на великому обсягу даних

Модель була натренована на масштабних корпусах:

- BooksCorpus (~800 млн слів) - художній контекст;
- English Wikipedia (~2.5 млрд слів) - фактологічна енциклопедична мова.

Така комбінація надала моделі багатий лінгвістичний та енциклопедичний досвід.

Схематично архітектуру навчання та тонкого настроювання BERT зображено на рисунку 7.

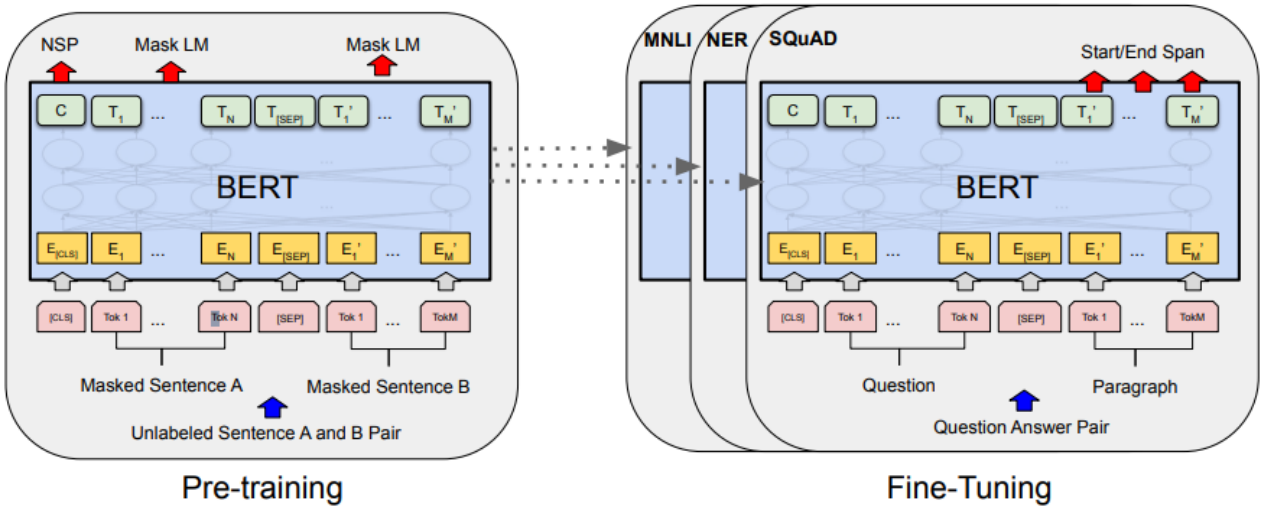


Рисунок 7. Архітектура навчання та тонкого настроювання BERT

BERT став проривом у сфері NLP, встановивши нові рекорди на численних бенчмарках (GLUE, SQuAD, MNLI тощо), і заклав основу для подальших моделей, таких як RoBERTa, ALBERT, DistilBERT та TinyBERT.

2.4 TinyBERT

TinyBERT - це компактна та ефективна мовна модель, розроблена для зменшення обчислювальних витрат та прискорення інференції без суттєвого

зниження точності. У роботі "TinyBERT: Distilling BERT for Natural Language Understanding" (Jiao et al., 2019) [17] запропоновано новий підхід до дистиляції знань, спеціально адаптований для трансформерних моделей, що дозволяє ефективно передавати знання від великої моделі-вчителя (BERT) до компактної моделі-учня (TinyBERT).

TinyBERT значно відрізняється від $BERT_{BASE}$ меншою кількістю шарів (4 або 6 замість 12), меншою розмірністю прихованого простору (наприклад, 312 замість 768), та відповідно меншою кількістю параметрів (до 28% від початкової моделі). Таке скорочення архітектури дозволяє досягти понад 9-кратного прискорення інференції при збереженні до 96.8% продуктивності $BERT_{BASE}$ на бенчмарку GLUE [19].

Основна інновація полягає у двоетапному процесі дистиляції:

- загальна дистиляція (General Distillation): модель-учень навчається імітувати поведінку моделі-вчителя на великому корпусі загального призначення, засвоюючи загальні мовні закономірності;
- дистиляція, орієнтована на конкретне завдання (Task-specific Distillation): модель додатково навчається на даних конкретного завдання, аугментованих за допомогою моделі-вчителя, що забезпечує адаптацію до прикладного контексту.

У процесі дистиляції використовуються кілька функцій втрат для узгодження поведінки моделей на різних рівнях:

- втрати на рівні вбудовувань: мінімізується середньоквадратична помилка між вхідними представленнями моделей;
- втрати на рівні трансформерного шару: узгоджуються матриці уваги та приховані стани;
- втрати на рівні передбачень: зменшується розбіжність між логітами моделей, що забезпечує подібність вихідних прогнозів.

Експериментальні результати демонструють ефективність такого підходу: 4-шарова версія TinyBERT показує майже ідентичну точність у порівнянні з $BERT_{BASE}$, але значно економніша в обчисленнях [17]. Це робить TinyBERT

особливо придатною для використання в умовах обмежених ресурсів, наприклад на мобільних пристроях або в режимі реального часу.

2.5 Метрики оцінки ефективності алгоритмів машинного навчання

У задачах машинного навчання, зокрема класифікації, метрики відіграють ключову роль в об’єктивному оцінюванні якості моделей. Вони дозволяють не лише порівнювати різні алгоритми між собою, але й контролювати процес навчання, виявляти перенавчання, а також адаптувати модель до специфіки конкретного завдання. Найпоширенішими метриками для оцінки ефективності класифікаційних алгоритмів є точність, влучність, повнота, F1-міра, а також площа під кривою ROC (AUC-ROC).

2.5.1 Точність

Точність (англ. accuracy) [26] є однією з найпростіших та найпоширеніших метрик для оцінювання ефективності алгоритмів класифікації. Вона відображає частку правильно класифікованих прикладів серед усіх прикладів у тестовій вибірці. Accuracy дає загальну картину того, як часто модель приймає правильні рішення. Якщо, наприклад, accuracy дорівнює 0.93 (або 93 %), це означає, що у 93 % випадків модель зробила правильне передбачення. Простота цієї метрики робить її особливо зручною на початковому етапі аналізу ефективності моделі.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \tag{2.20}$$

де TP - кількість істинно позитивних передбачень;

TN - кількість істинно негативних передбачень;

FP - кількість хибно позитивних передбачень;

FN - кількість хибно негативних передбачень.

Переваги:

- легка у розрахунку та інтерпретації;
- підходить для задач із збалансованими класами, де кількість

об'єктів кожного класу приблизно однакова.

Недоліки:

- у задачах з незбалансованими класами accuracy може давати *оману про якість моделі*. Наприклад, якщо 95 % прикладів належать до одного класу, модель, яка завжди передбачає цей клас, матиме accuracy 95 %, але не буде корисною для виявлення іншого класу;
- не дозволяє оцінити помилки в контексті важливості - тобто не відрізняє наслідки FP і FN, що критично в деяких прикладних задачах (медицина, безпека тощо).

2.5.2 Влучність

Влучність (англ. precision) [27] є однією з ключових метрик для оцінки ефективності алгоритмів класифікації, особливо у задачах, де важливо мінімізувати хибнопозитивні передбачення. Ця метрика показує, яка частка об'єктів, що були класифіковані як позитивні, насправді належать до позитивного класу.

$$Precision = \frac{TP}{TP+FP}, \tag{2.21}$$

де TP - кількість істинно позитивних передбачень;

FP - кількість хибно позитивних передбачень.

Влучність фокусується лише на тих прикладах, які модель позначила як позитивні. Якщо вона дорівнює 0.85, це означає, що 85 % усіх передбачень позитивного класу є правильними, а решта 15 % - помилкові. Метрика особливо важлива в умовах, коли хибнопозитивні передбачення мають серйозні наслідки, як-от у випадках автоматичної модерації контенту, щоб не блокувати безпечний вміст, або наприклад, пошукових систем, щоб зменшити кількість нерелевантних результатів.

Переваги:

- є чутливою до кількості хибнопозитивних класифікацій;
- дозволяє оцінити, наскільки надійні позитивні передбачення моделі.

Недоліки:

- влучність не враховує кількість хибнонегативних, тобто не говорить про те, скільки позитивних об'єктів модель не виявила;
- відокремлено від інших метрик може створювати викривлену картину ефективності, особливо при сильному дисбалансі між істинно позитивними та хибно негативними.

Для всебічної оцінки ефективності моделі влучність часто аналізується у поєднанні з повнотою, а також у складі комбінованої метрики - F1-міри. Підвищення влучності части веде до зниження повноти, і навпаки, тому доцільно враховувати обидві метрики для балансу.

2.5.3 Повнота

Повнота (англ. recall) - це метрика оцінки якості класифікації, яка характеризує здатність моделі виявляти всі об’єкти, що належать до позитивного класу. Вона особливо важлива в тих задачах, де критично не пропустити позитивні випадки, навіть якщо при цьому виникають хибнопозитивні передбачення. Прикладами можуть бути медична діагностика, де висока повнота важлива, щоб не пропустити жодного хворого пацієнта [27].

$$Recall = \frac{TP}{TP+FN}, \tag{2.22}$$

де TP - кількість істинно позитивних передбачень;

FN - кількість хибно негативних передбачень.

Повнота показує, яку частку з усіх фактичних позитивних прикладів модель змогла правильно класифікувати. Наприклад, якщо повнота дорівнює 0.9, це означає, що модель виявила 90 % усіх позитивних випадків, пропустивши 10 %.

Переваги:

- дає уявлення про здатність моделі виявляти усі релевантні (позитивні) випадки;
- є критично важливою у задачах, де пропуск позитивного прикладу

неприпустимий.

Недоліки:

- не враховує кількість хибно позитивних, тобто може бути високою навіть при їх великій кількості;
- при ізольованому аналізі може вводити в оману, оскільки модель із високим показником повноти, але низьким показником влучності, просто класифікуватиме майже все як позитивне.

Повнота часто аналізується разом із влучністю, адже ці метрики відображають різні аспекти класифікаційної точності. У багатьох випадках зростання однієї призводить до зниження іншої, і навпаки.

2.5.4 F1-міра

F1-міра - це інтегральна метрика ефективності класифікації, яка враховує як влучність, так і повноту, поєднуючи їх у єдине значення. Вона широко використовується у задачах машинного навчання, зокрема в обробці природної мови, де важливим є баланс між здатністю моделі виявляти всі релевантні об’єкти та її точністю у передбаченнях [27].

F1-міра визначається як гармонічне середнє між влучністю і повнотою:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}, \quad (2.23)$$

де $Precision$ - влучність;

$Recall$ - повнота.

F1-міра досягає найвищого значення - 1, коли і влучність, і повнота максимальні. Якщо хоча б одна з цих метрик дорівнює нулю, то і F1-міра також дорівнює нулю. Це забезпечує збалансовану оцінку, уникаючи завищення результату в разі високого значення однієї метрики за рахунок низького значення іншої.

F1-міра є особливо важливою у задачах з незбалансованими класами, де звичайна точність може бути оманливо високою. Наприклад, у випадку, коли позитивний клас становить лише 5 % усіх прикладів, модель, що завжди

передбачає негативний клас, досягне 95 % точності, але F1 буде дорівнювати нулю.

У задачах виявлення спаму, медичної діагностики, класифікації тональності тощо - F1-міра дозволяє об’єктивно оцінити, наскільки добре модель балансує між виявленням релевантних об’єктів і точністю у своїх передбаченнях.

2.5.5 ROC-крива

ROC-крива відображає залежність між повнотою (TPR) та хибнопозитивним відношенням (FPR) [28].

Хибнопозитивне відношення (англ. false positive rate, FPR) - це частка негативних прикладів, які модель помилково класифікувала як позитивні. Обчислюється як:

$$FPR = \frac{FP}{FP + TN}, \tag{2.24}$$

де FP - кількість хибно позитивних передбачень;

TN - кількість істинно негативних передбачень.

Модель змінює поріг класифікації, і на кожному пороговому значенні розраховуються TPR та FPR. Крива ROC будується шляхом нанесення TPR на вісь Y, а FPR - на вісь X, для кожного порогу.

Площа під кривою, або AUC (Area Under Curve) - числове значення від 0 до 1, яке визначає загальну якість моделі:

- AUC = 1.0 - ідеальна класифікація;
- AUC = 0.5 - випадкове передбачення (немає дискримінаційної здатності);
- AUC < 0.5 - модель працює гірше за випадкову (або переплутала класи).

AUC показує ймовірність того, що модель розташує випадковий позитивний приклад вище за випадковий негативний. Наприклад, AUC = 0.97 означає, що у 97 % випадків модель правильно розрізняє позитивний і

негативний приклади.

Переваги:

- не залежить від вибраного порогу класифікації;
- добре працює з незбалансованими наборами даних;
- дає загальне уявлення про дискримінаційну здатність моделі.

Недоліки:

- у багатокласових задачах потребує узагальнення (наприклад, мікро- або макроусереднення);
- не враховує реальну вартість помилок (може бути недостатньо інформативним у прикладних задачах, де важливі конкретні типи помилок).

Метрика AUC-ROC, що графічно зображена на рисунку 8, дозволяє кількісно оцінити здатність моделі розрізняти класи незалежно від вибраного порогу, що робить її незамінною в задачах з незбалансованими класами та в системах, де важливо аналізувати поведінку класифікатора на різних рівнях чутливості. Її доцільно використовувати разом з іншими метриками, зокрема точністю, повнотою та F1, для формування повної картини ефективності моделі.

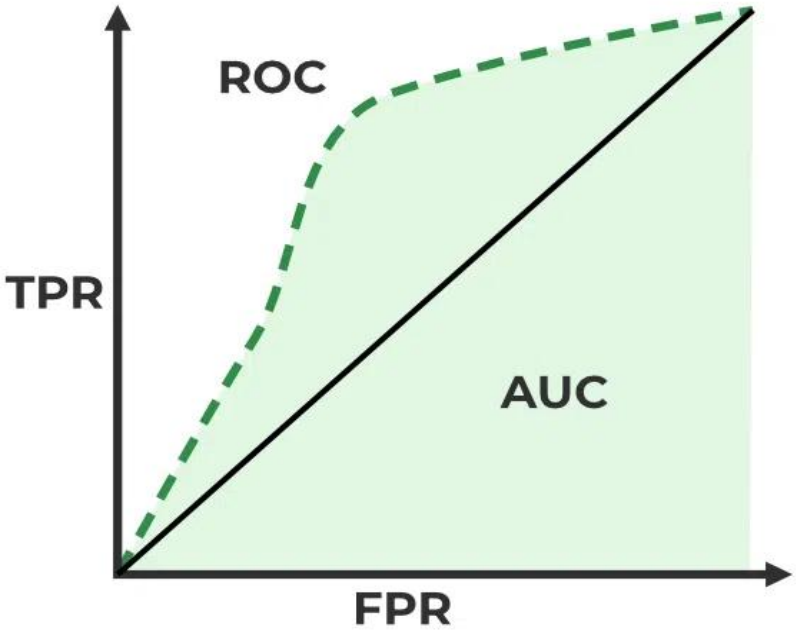


Рисунок 8. Графік ROC-кривої та AUC

Висновки

У цьому розділі було розглянуто ключові теоретичні основи, які лежать в основі створення моделі бінарної класифікації текстів за емоційним забарвленням. Перш за все, було охарактеризовано штучну нейронну мережу як математичну модель, натхненну принципами функціонування біологічного мозку, яка здатна автоматично виявляти складні залежності в даних. Далі було розглянуто архітектуру трансформера, що стала основою сучасних моделей обробки природної мови, зокрема завдяки механізму самоуваги, який дозволяє моделі ефективно враховувати контекст у тексті незалежно від його довжини.

Особливу увагу було приділено моделі BERT - двонапрямленому трансформеру, попередньо навчена версія якого використовується для широкого спектра NLP-завдань. Зокрема, розглянуто її дистильовану версію TinyBERT, яка за рахунок меншої кількості параметрів забезпечує швидше навчання та ефективне використання ресурсів при збереженні високого рівня точності.

Крім архітектурних особливостей моделей, було детально охарактеризовано метрики, які використовуються для оцінки їхньої ефективності, зокрема, точність, влучність, повноту, F1-міру, а також ROC-криву та площа під нею, які дають змогу всебічно оцінити наскільки результативно працює модель.

Загалом, поданий теоретичний матеріал слугує фундаментом для реалізації мети дослідження - створення ефективної та оптимізованої моделі для бінарної класифікації англомовних текстів за їх емоційною полярністю.

РОЗДІЛ 3. РОЗРОБКА ТА ОЦІНКА МОДЕЛІ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ЕМОЦІЙНОГО ЗАБАРВЛЕННЯ ТЕКСТІВ

3.1 Аналіз вхідних даних

Для навчання моделі було обрано набір даних із відгуками з Amazon [29], що є підмножиною набору від Stanford Network Analysis Project, який охоплює період у 18 років та складається з близько 35 мільйонів відгуків. Цільовий набір даних включає рівну кількість у 1,800,000 тренувальних та 200,000 тестових зразків для кожної полярності (негативні та позитивні відгуки).

У наборі даних Amazon Reviews Polarity, що використовується для задачі класифікації емоційної полярності текстів, кожен запис представлений у вигляді трьох основних колонок: polarity, title та text (рисунок 9). Дані зберігаються у форматі CSV.

	polarity	title	text
0	2	Stuning even for the non-gamer	This sound track was beautiful! It paints the ...
1	2	The best soundtrack ever to anything.	I'm reading a lot of reviews saying that this ...
2	2	Amazing!	This soundtrack is my favorite music of all ti...
3	2	Excellent Soundtrack	I truly like this soundtrack and I enjoy video...
4	2	Remember, Pull Your Jaw Off The Floor After He...	If you've played the game, you know how divine...
...	...	...	...
3599995	1	Don't do it!!	The high chair looks great when it first comes...
3599996	1	Looks nice, low functionality	I have used this highchair for 2 kids now and ...
3599997	1	compact, but hard to clean	We have a small house, and really wanted two o...
3599998	1	what is it saying?	not sure what this book is supposed to be. It ...
3599999	2	Makes My Blood Run Red-White-And-Blue	I agree that every American should read this b...

3600000 rows × 3 columns

Рисунок 9. Приклад вмісту набору даних Amazon Reviews Polarity

Колонка polarity виконує роль цільової змінної у задачі машинного навчання. Вона містить цілочисельні значення, що відповідають емоційній полярності відгуку: значення 1 позначає негативний відгук, а значення 2 - позитивний. Ця класифікація сформована на основі рейтингу користувача: відгуки з оцінками 1 та 2 були віднесені до негативних, а з оцінками 4 та 5 - до позитивних. Оцінки з нейтральним значенням 3 були виключені з аналізу, що

дозволяє зосередитися виключно на бінарній класифікації.

Колонка title містить заголовок відгуку у вигляді короткого текстового фрагмента. Заголовки можуть містити основні враження або ключові емоційні характеристики, тому їх включення до моделі може покращити якість класифікації.

Колонка text є основним корпусом відгуку. Це розгорнутий текст, у якому користувачі висловлюють своє ставлення до продукту. Тексти можуть мати довільну довжину та включати розповідні елементи, опис досвіду використання товару, а також емоційно забарвлену лексику.

Технічно, усі текстові поля (тобто title та text) представлені у вигляді рядків з подвійним екрануванням лапок та символів нового рядка. Це дозволяє коректно обробляти дані навіть при наявності багаторядкових описів або спеціальних символів у тексті.

Оскільки в наборі даних рівна кількість негативних і позитивних відгуків для кожного зразка тренувальних (рисунок 10) та тестових даних (рисунок 11), можна зробити висновок про збалансованість датасету з точки зору класів. Співвідношення вибірок по класам є важливим фактором для ефективного навчання моделі, оскільки відсутність сильного дисбалансу між класами дозволяє уникнути проблеми з домінуванням одного класу над іншим.

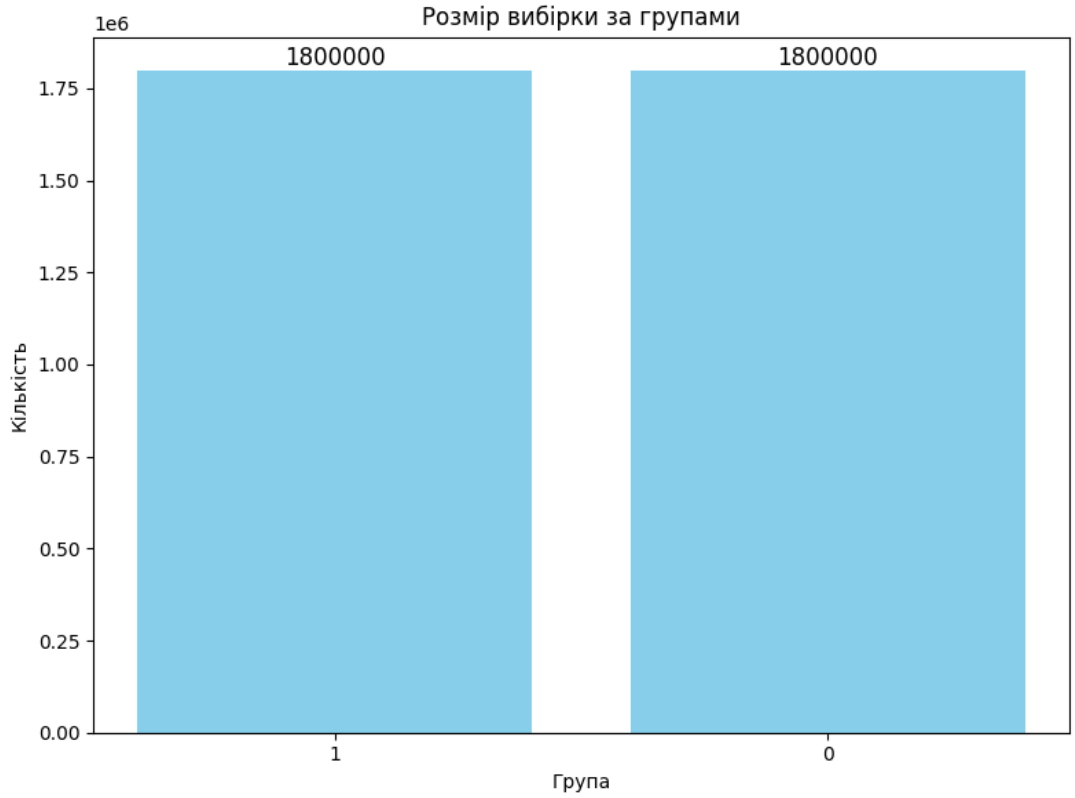


Рисунок 10. Розмір вибірки за групами навчального набору даних

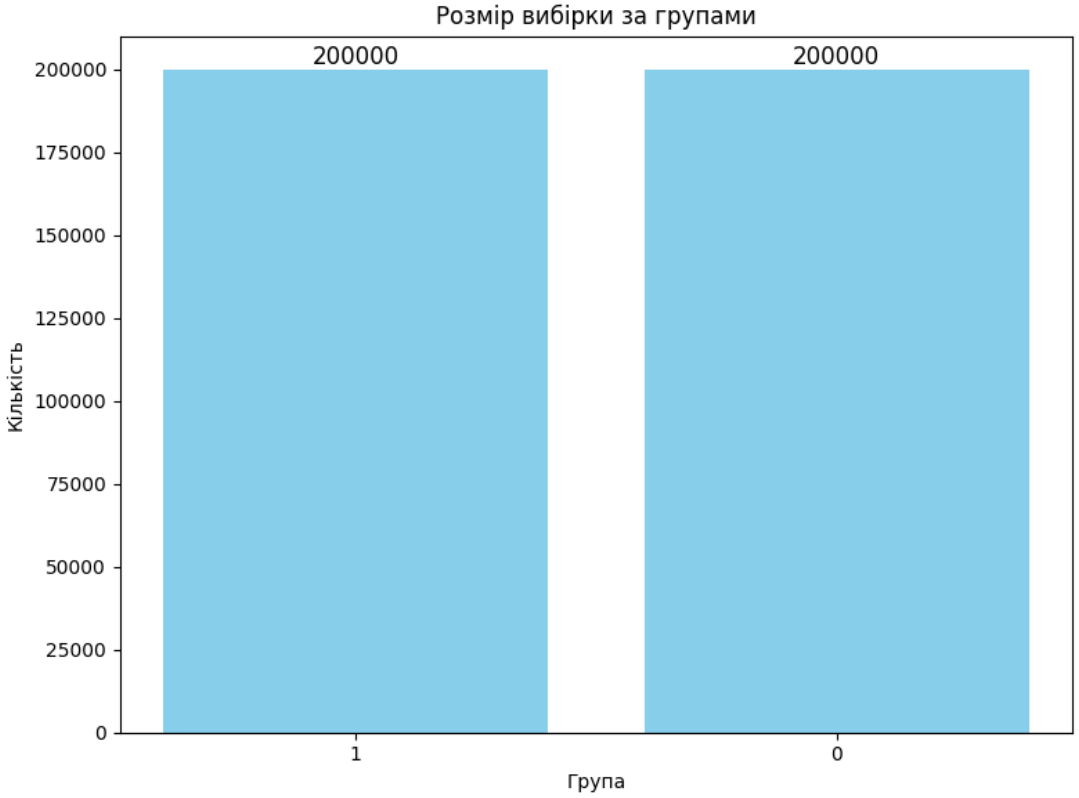


Рисунок 11. Розмір вибірки за групами тестового набору даних

3.2 Вибір засобів для реалізації моделі

3.2.1 Мова програмуванная Python

Мова програмування Python стала основною платформою для реалізації моделі. Цей вибір є зумовленим її широким розповсюдженням у науковій та прикладній спільноті в сфері штучного інтелекту. Python поєднує в собі простоту синтаксису та велику кількість бібліотек, що дозволяє швидко реалізовувати та тестувати складні алгоритми. Особливо важливим є її інтеграція з фреймворками глибокого навчання, що підтримують обчислення на GPU, а також потужні засоби роботи з текстовими даними та візуалізацією.

3.2.2 Jupyter Notebook

Jupyter Notebook - це веб-інтерактивне середовище для створення та виконання документів, які поєднують у собі програмний код, текстові пояснення (у форматі Markdown), математичні формули (на основі LaTeX), візуалізації та мультимедійний контент. Це середовище дозволяє працювати з обчисленнями в інтерактивному режимі, що особливо зручно для дослідницької роботи, аналізу даних, створення прототипів, викладання та візуалізації результатів [30].

Особливості Jupyter Notebook:

- Основою є браузерний інтерфейс, що забезпечує просту та зручну роботу без потреби в складних налаштуваннях локального середовища;
- Документ Jupyter зберігається у форматі JSON-файлу з розширенням .ipynb, який містить впорядковані комірки коду та тексту;
- Підтримує інтерактивне виконання коду з можливістю переглядати результат безпосередньо під кодом;
- Комірки можуть містити не лише код, а й текстові пояснення, графіки, таблиці, тощо.

Jupyter Notebook є особливо зручним для Python через низку причин. По-

перше, він історично виник з IPython, тому має тісну інтеграцію з Python-екосистемою. По-друге, більшість інструментів для наукових обчислень і аналізу даних у Python мають повну сумісність із середовищем Jupyter. По-третє, підтримка графіки, інтерактивних візуалізацій і можливість поступового виконання коду дозволяє зручно налагоджувати алгоритми та експериментувати з даними.

3.2.3 Бібліотека pandas

Бібліотека pandas застосовується для обробки структурованих даних у вигляді таблиць (DataFrame), забезпечуючи ефективні операції з читання, фільтрації, агрегації, трансформації та збереження даних. У контексті задачі класифікації відгуків pandas дозволяє зручно завантажити датасет, об’єднати текстові поля, видалити порожні значення, а також перетворити категоріальні змінні у числові формати, необхідні для машинного навчання.

3.2.4 Бібліотека transformers від Hugging Face

Бібліотека transformers надала доступ до попередньо навченої моделі TinyBERT, а також її токенізатора. Ця бібліотека абстрагує складну архітектуру трансформерів у простий у використанні інтерфейс, дозволяючи швидко ініціалізувати, налаштовувати та донавчати моделі на власних даних. Hugging Face також забезпечує велику колекцію моделей, підтримку багатьох мов і активне оновлення репозиторію.

3.2.5 Бібліотека datasets від Hugging Face

Бібліотека datasets дозволяє зручно завантажувати, обробляти та форматувати текстові набори даних. У рамках проєкту вона може використовуватися як для імпорту стандартних корпусів, так і для організації наборів даних у форматі, сумісному з моделями transformers. Підтримка автоматичної токенізації, батчінгу та трансформацій даних робить цю бібліотеку особливо корисною при масштабних обчисленнях.

3.2.6 Бібліотека scikit-learn

Нарешті, scikit-learn (sklearn) - це класична бібліотека машинного навчання в Python, яка була використана для розділення даних на навчальну та тестову вибірки, а також для розрахунку метрик ефективності, таких як accuracy, precision, recall і F1-міра. Sklearn забезпечує стабільність, зручний API та численні утиліти для оцінки результатів моделей, що робить її незамінною частиною конвеєру машинного навчання.

3.2.7 Бібліотека optuna

Optuna - це сучасна бібліотека для автоматизованої оптимізації гіперпараметрів, яка використовує підхід на основі байєсівської оптимізації з елементами евристик. Вона дозволяє ефективно шукати найкращі параметри для машинного навчання завдяки гнучкій архітектурі, підтримці early stopping, розподіленим обчисленням та можливості фіксувати результати експериментів. Optuna активно використовується для тонкого настроювання моделей, зокрема у випадках, коли простір гіперпараметрів великий або дорого обходиться повний перебір варіантів.

У раніше розглянутій бібліотеці transformers передбачено вбудовану підтримку optuna через метод Trainer.hyperparameter_search. Цей механізм дозволяє здійснювати автоматичну оптимізацію гіперпараметрів, під час тонкого настроювання трансформерних моделей. Така інтеграція значно спрощує процес налаштування моделей і підвищує якість результатів, не вимагаючи ручного добору параметрів.

3.2.8 Фреймворк PyTorch

PyTorch - це потужний відкритий фреймворк глибокого навчання, розроблений компанією Meta AI, який став одним із найпопулярніших інструментів для створення, тренування та розгортання нейронних мереж. Він забезпечує гнучкість та зручний синтаксис, що робить його зручним як для дослідників, так і для інженерів.

PyTorch підтримує автоматичне диференціювання, оптимізатори, модулі для побудови моделей, обробку даних та інтеграцію з GPU, що дозволяє ефективно працювати з великими обчисленнями. Завдяки широкій екосистемі (TorchVision, TorchText, TorchAudio) та сумісності з іншими бібліотеками (наприклад, transformers, optuna, scikit-learn) PyTorch активно використовується у наукових дослідженнях, промислових застосуваннях, NLP, CV та багатьох інших галузях.

3.3 Підготовка даних

З метою оптимізації часу, необхідного для проведення експериментів на початкових етапах розробки моделі, а також для пришвидшення процесу тестування архітектурних рішень і налаштування гіперпараметрів, було прийнято рішення використовувати не повний набір даних, а лише його зменшену репрезентативну підмножину. Повний обсяг наявного корпусу складає приблизно 3 600 000 текстових прикладів, рівномірно розподілених між двома класами емоційної полярності. Обробка такого масиву даних потребувала б значних обчислювальних ресурсів і часу, що на етапі попереднього моделювання є недоцільним.

Тому для пришвидшення експериментів було здійснено випадкову вибірку 50 000 прикладів із загальної сукупності. Такий обсяг було визначено як достатній для формування попередніх висновків щодо поведінки моделі, при цьому він дозволяє істотно скоротити час навчання й оцінки. Вибірка проводилася методом випадкового семплювання з урахуванням балансу класів, що дало змогу зберегти пропорційне співвідношення позитивних і негативних відгуків, забезпечивши тим самим репрезентативність підмножини. Такий підхід дозволяє ефективно протестувати ключові параметри моделі та структуру навчання, не витрачаючи надлишкових ресурсів до моменту остаточної стабілізації архітектури. Код для цього кроку зображений на рисунку 12.

```
df_train = pd.read_csv("train.csv").sample(50000)
```

Рисунок 12. Код завантаження набору даних та взяття його підвибірки

Для перевірки репрезентативності отриманої вибірки було побудовано стовпчикову діаграму з розподілом зразків по класах полярності (рисунок 13). На цьому графіку видно, що збалансованість по класах збереглася і тому є сенс продовжувати працювати з отриманим набором даних.

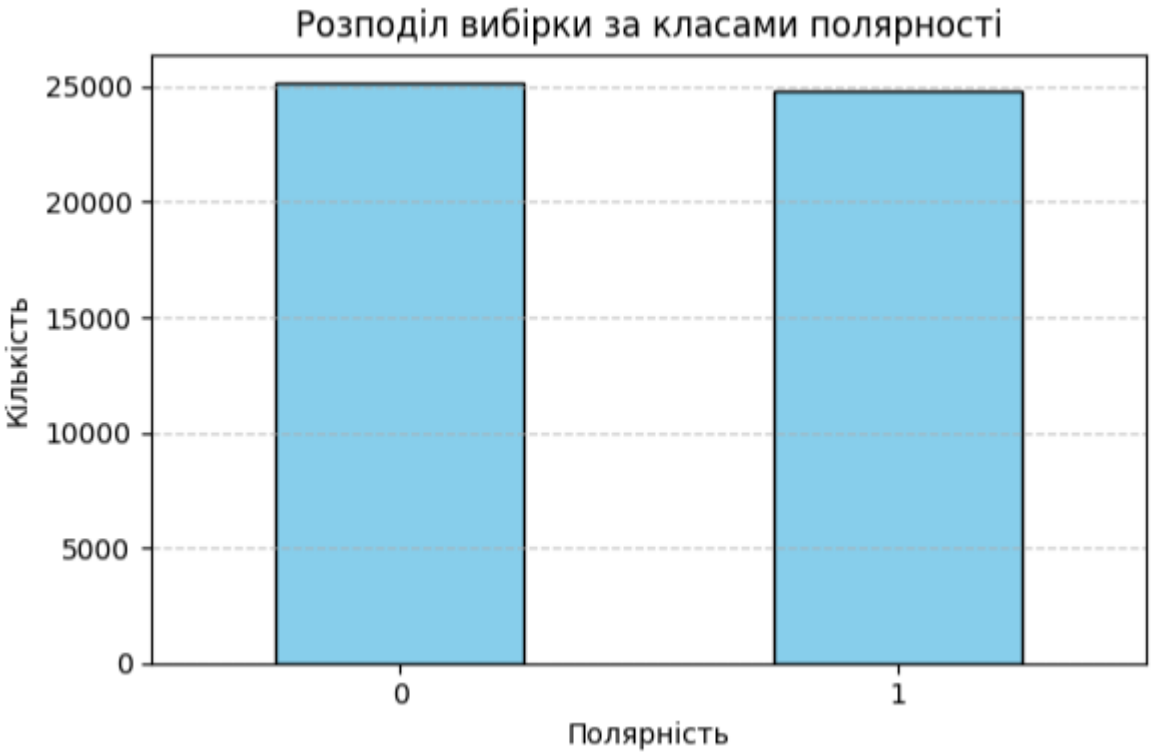


Рисунок 13. Розподіл підвибірки за класами полярності

Наступним етапом виконується попередня обробка текстових даних, що передбачає створення нового датафрейму, у якому зберігаються результати очищення вхідної інформації. Основним завданням цієї функції є підвищення якості текстових представлень для подальшого аналізу. На початку виконання об’єднуються два текстові стовпці: title (заголовок відгуку) та text (основна частина відгуку). Для уникнення технічних помилок, які можуть виникнути в разі наявності пропущених значень (NaN), обидва поля попередньо заповнюються порожніми рядками. Потім вміст цих колонок зливається у єдине текстове поле, оскільки заголовок часто містить стисле узагальнення думки

користувача, тоді як основна частина деталізує його емоційну оцінку.

Зконкатенований текст передається до функції *clean_text*, код якої зображено на рисунку 14, яка здійснює очищення від зайвих символів, включно з пунктуацією, HTML-тегами иа спеціальними символами, таким чином підвищується однорідність та якість текстових вхідних даних. Крім цього, мітки класів, що зберігаються в колонці polarity, перетворюються з початкового представлення (де 1 - негативний відгук, а 2 - позитивний) у формат, необхідний для моделі: 0 для негативного класу та 1 для позитивного. Таким чином, функція забезпечує базову, але важливу підготовку даних перед навчанням моделі. Код функції *df_preprocess* зображено на рисунку 15.

```
def clean_text(text):
    text = re.sub(r"<.*?>", "", text)  # Видалення HTML
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)  # Видалення всіх символів, окрім латинських літер, цифр та пробілів
    return text
```

Рисунок 14. Код функції *clean_text*

```
def df_preprocess(df: pd.DataFrame):
    processed_df = pd.DataFrame()
    processed_text = df['title'].fillna('') + ' ' + df['text'].fillna('')
    processed_text = processed_text.apply(clean_text)
    processed_df['text'] = processed_text
    processed_df['label'] = df['polarity'].map({1: 0, 2: 1})
    return processed_df
```

Рисунок 15. Код функції *df_preprocess*

Останнім етапом відбувається виконання раніше описаної функцІЇ *df_preprocess* для попередньо отриманої вибірки з 50 000 рядків, після чого результат зберігається у новий CSV-файл під назвою amazon_comments_train.csv. Код цього кроку зображено на рисунку 16.

```
df_preprocess(df_train).to_csv('amazon_comments_train.csv', index=False)
```

Рисунок 16. Код виконання попередньої обробки для тренувального набору даних та збереження отриманого результату

Загалом наведений фрагмент коду ілюструє типову процедуру підготовки текстових даних для задач машинного навчання, яка охоплює зменшення

обсягу, агрегацію релевантної інформації, очищення текстів, уніфікацію форматів міток та збереження результату.

В рамках процесу попередньої обробки тексту було свідомо пропущено такі традиційні етапи, як лематизація або стемінг, приведення до нижнього регістру, а також видалення стоп-слів. Це рішення зумовлене характером обраної моделі - TinyBERT, яка базується на архітектурі трансформера та використовує попередньо навчену токенізаторну систему, що вже передбачає низку внутрішніх механізмів попередньої обробки.

Перш за все, лематизація або стемінг, які зазвичай застосовуються для приведення слів до їх основної форми, не були використані з огляду на особливості роботи обраної моделі. TinyBERT, як і інші моделі BERT-подібного типу, застосовує токенізацію за принципом WordPiece, що дозволяє моделі автоматично опрацьовувати лексеми незалежно від їх морфологічних варіантів. Іншими словами, модель розбиває рідкісні або похідні слова на підслова, які часто повторюються в корпусі, що дозволяє зберігати семантичну інформацію навіть без лематизації. Втручання у вихідну словоформу, особливо у вигляді агресивного стемінгу, може навіть зашкодити точності, оскільки моделі були навчені на контекстах, які включають повноцінні словоформи.

Щодо приведення до нижнього регістру, ця операція є стандартною для традиційних моделей NLP, наприклад, у випадках із TF-IDF чи word2vec, однак у випадку TinyBERT вона вже вбудована у токенізатор. Усі моделі, що мають маркування uncased, а huawei-noah/TinyBERT_General_4L_312D є саме такою, автоматично трансформують текст до нижнього регістру на етапі токенізації. Таким чином, виконання цього кроку вручну є зайвим і не впливає на подальшу якість або узгодженість вхідних даних.

Також було пропущено етап видалення стоп-слів, оскільки трансформерні моделі, на відміну від традиційних методів, не потребують жорсткого відсікання часто вживаних функціональних слів. У класичних моделях на кшталт наївного баєсівського класифікатора чи логістичної регресії стоп-слова могли розмивати статистичні зв’язки між термінами, однак у BERT-

подібних моделях контекстуальна увага дозволяє моделі самостійно визначати важливість кожного токена залежно від його ролі в конкретному контексті. Видалення стоп-слів вручну могло би навіть зашкодити, оскільки функціональні слова, такі як заперечення "not" або "never", нерідко мають ключове значення для визначення емоційної полярності.

Підсумовуючи етап обробки тексту, потрібно звернути увагу на те, що всі вищезазначені етапи були опущені не через неувагу чи спрощення, а навпаки - внаслідок усвідомленого врахування архітектурних особливостей і навчальної парадигми моделі TinyBERT. Цей підхід дозволяє уникнути зайвих модифікацій вхідного тексту та зберегти його максимально наближеним до природного вигляду, на подібному якому модель уже була попередньо навчена.

3.4 Розробка моделі

3.4.1 Підбір гіперпараметрів моделі

Для досягнення оптимальної продуктивності моделей машинного навчання, особливо таких складних архітектур, як трансформер, важливим етапом є підбір гіперпараметрів. Ефективність моделі значною мірою залежить від коректного вибору таких параметрів, як темп навчання, кількість епох навчання, коефіцієнт регуляризації, тощо.

Для систематичного та ефективного пошуку найкращих комбінацій цих параметрів було застосовано автоматизований підхід. З цією метою було використано функціонал *hyperparameter_search*, що доступний у бібліотеці transformers Цей інструмент є потужним засобом для автоматизованого пошуку оптимальних гіперпараметрів і використовує бібліотеку *Optuna* як бекенд, що забезпечує ефективні алгоритми оптимізації для ітеративного дослідження простору гіперпараметрів на основі результатів попередніх спроб.

Підбір гіперпараметрів, відомий також як оптимізація гіперпараметрів (Hyperparameter Optimization, HPO), є процесом пошуку такого набору значень гіперпараметрів моделі та процесу навчання, який дозволяє досягти найкращої продуктивності на валідаційному наборі даних. Цей процес принципово

відрізняється від навчання самої моделі, де оптимізуються ваги та зміщення (weights and biases) на основі тренувальних даних.

Основний принцип полягає у систематичному дослідженні багатовимірного простору гіперпараметрів. Кожна точка в цьому просторі відповідає певній комбінації значень гіперпараметрів. Мета - знайти точку, яка мінімізує або максимізує певну цільову функцію, яка, як правило, базується на метриці якості моделі (наприклад, точність, F1-score, втрати) на валідаційному наборі даних після завершення (або проміжного етапу) навчання моделі з цими параметрами.

На відміну від простих методів, таких як повний перебір (Grid Search) або випадковий пошук (Random Search), Optuna використовує більш інтелектуальні та ефективні стратегії. Вона належить до класу Баєсової оптимізації (Bayesian Optimization) та використовує методи, такі як TPE (Tree-structured Parzen Estimator), для адаптивного вибору наступних значень гіперпараметрів для тестування. Це означає, що Optuna аналізує результати попередніх спроб (trials) і на основі цієї інформації приймає більш обґрунтовані рішення щодо того, яку частину простору гіперпараметрів досліджувати далі, швидше рухаючись до потенційного оптимуму. Крім того, Optuna підтримує “обрізку” (pruning) - можливість достроково зупиняти неефективні спроби, якщо стає очевидним, що дана комбінація гіперпараметрів не приведе до хорошого результату, заощаджуючи таким чином обчислювальні ресурси та час.

Процес за допомогою hyperparameter_search та Optuna виглядає наступним чином:

- визначається простір пошуку - діапазони або дискретні значення для кожного гіперпараметра, який підлягає оптимізації;
- визначається цільова функція - це функція, яка приймає набір гіперпараметрів, навчає модель з цими параметрами на тренувальних даних та повертає значення метрики якості на валідаційному наборі даних;
- optuna (через hyperparameter_search) запускає спроби (trials): на

кожній спробі обирається новий набір гіперпараметрів з простору пошуку, модель навчається та оцінюється.

Після заданої кількості спроб або досягнення критерію зупинки, Optuna надає результати, включаючи найкращу знайдену комбінацію гіперпараметрів та відповідне значення цільової функції.

Хоча існує багато гіперпараметрів, які можна було б оптимізувати, деякі з них мають значно більший вплив на продуктивність Трансформерних моделей.

При роботі з обраною моделлю, особливо важливими для підбору є:

- темп навчання (learning_rate) - гіперпараметр, що визначає розмір кроку, з яким оптимізатор оновлює ваги моделі на основі градієнтів. Він є важливим, оскільки занадто високий темп навчання може призвести до "перестрибування" через оптимум, нестабільності навчання і навіть розбіжності. Занадто низький темп робить процес оптимізації дуже повільним і може застрягти в локальних мінімумах;
- кількість епох навчання (num_train_epochs) - визначає, скільки разів модель пройде через весь тренувальний набір даних. Недостатня кількість епох призведе до недонавчання (underfitting), тоді як надмірна кількість - до перенавчання (overfitting) на тренувальних даних та погіршення продуктивності на валідаційних/тестових даних. Оптимальна кількість епох дозволяє моделі навчитися достатньо, щоб мати високу спроможність до узагальнення;
- коефіцієнт регуляризації (weight_decay) - додає штраф до функції втрат пропорційно квадрату величини ваг (L2 регуляризація). Даний параметр допомагає запобігти перенавчанню, обмежуючи величину ваг. Це змушує модель бути менш залежною від окремих ознак і сприяє більш гладкій функції рішень, покращуючи узагальнення на невідомих даних;
- співвідношення "розігріву" швидкості навчання (warmup_ratio) - визначає, яку частку (або скільки кроків) від загального процесу

навчання швидкість навчання поступово збільшуватиметься від нуля до початкового значення learning_rate. Ця техніка є стандартною для Трансформерів і допомагає стабілізувати процес навчання на ранніх етапах, коли градієнти можуть бути особливо нестабільними через випадкову ініціалізацію ваг. Правильна тривалість "розігріву" може значно вплинути на сходимість та фінальну продуктивність.

Для вирішення задачі пошуку гіперпараметрів було використано бібліотеку transformers, яка містить необхідний токенізатор, переднавчену модель TinyBERT і власне функціонал для пошуку оптимальних гіперпараметрів, який базується на бібліотеці optuna.

Як і для будь якої задачі роботи з даними, початково здійснюється завантаження набору даних. Далі здійснюється розділення корпусу на навчальну та валідаційну підвибірки у співвідношенні 90/10. Обидві частини перетворюються у формат Dataset із подальшим об’єднанням у структуру DatasetDict. Відповідний код зображено на рисунку 17.

```
df = pd.read_csv("amazon_comments_train.csv")
train_texts, val_texts = train_test_split(df, test_size=0.2, stratify=df['label'], random_state=42)
dataset = DatasetDict({
    "train": Dataset.from_pandas(train_texts.reset_index(drop=True)),
    "validation": Dataset.from_pandas(val_texts.reset_index(drop=True))
})
```

Рисунок 17. Код завантаження та підготовки даних для моделі

Для обраної моделі ініціалізується відповідний токенізатор, який застосовується до текстів з урахуванням усічення занадто довгих послідовностей. Результатом є токенізований корпус, придатний для подачі на вхід моделі. Код для описної логіки зображено на рисунку 18.

```
model_name = "huawei-noah/TinyBERT_General_4L_312D"
tokenizer = AutoTokenizer.from_pretrained(model_name)

def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True)

tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

Рисунок 18. Код токенізації набору даних

На наступному етапі реалізується конфігурація навчального процесу, а також ініціалізується інтерфейс Trainer - високорівнева обгортка бібліотеки Transformers, яка забезпечує спрощену взаємодію з моделлю, автоматичну обробку даних, логування та оцінювання якості.

Функція *compute_metrics* відповідає за обчислення стандартних метрик класифікації на основі результатів моделі, в результаті виконання якої повертається словник із чотирма основними метриками: точність, precision влучність, повнота та F1-міра. Ці метрики є базовими для оцінки ефективності у задачах бінарної класифікації.

Наступним кроком задаються параметри навчання у вигляді об’єкта *TrainingArguments*. Зокрема, вказується директорія для збереження результатів, частота виконання оцінювання, частота логування, розміри міні-пакетів для навчання та валідації, а також відключається збереження найкращої моделі , що відповідає потребам оптимізації гіперпараметрів. Також зазначається, що цільовою метрикою для підбору оптимальних параметрів виступає F1-міра.

Оскільки Trainer підтримує як пряме передавання вже готової моделі, так і ініціалізацію моделі “на вимогу”, для сумісності з механізмами Optuna використовується функція *model_init*. Вона створює новий екземпляр *AutoModelForSequenceClassification* на основі попередньо навченої трансформерної моделі TinyBERT, адаптованої до задачі бінарної класифікації.

Також ініціалізується об'єкт *data_collator*, який відповідає за вирівнювання довжин вхідних послідовностей у межах одного міні-пакету під час навчання. Це дозволяє зменшити витрати пам’яті та покращити ефективність обробки.

У підсумку створюється об'єкт trainer, який поєднує всі раніше визначені компоненти: модель, параметри навчання, навчальну та валідаційну вибірки, токенізатор, об'єкт вирівнювання та функцію обчислення метрик. Trainer виступає ключовим об’єктом для запуску навчання, оцінювання продуктивності моделі та, у подальшому, інтеграції з процесом автоматичного підбору

гіперпараметрів. Код описаного фрагмента зображено на рисунку 19.

```
def compute_metrics(p: EvalPrediction):
    preds = p.predictions[0] if isinstance(p.predictions, tuple) else p.predictions
    preds = preds.argmax(axis=1)
    labels = p.label_ids
    return {
        "accuracy": accuracy_score(labels, preds),
        "precision": precision_score(labels, preds),
        "recall": recall_score(labels, preds),
        "f1": f1_score(labels, preds),
    }

training_args = TrainingArguments(
    output_dir=OUTPUT_DIR,
    eval_strategy="epoch",
    save_strategy="no",
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    load_best_model_at_end=False,
    metric_for_best_model="f1",
    report_to="none",
    logging_steps=50,
)

def model_init():
    return AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
trainer = Trainer(
    model_init=model_init,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

Рисунок 19. Код конфігурації навчального процесу

Для завершального етапу пошуку гіперпараметрів було створено Функція *optuna_hp_space* описує простір пошуку для чотирьох гіперпараметрів:

1) learning_rate: для цього параметра вибрано простір у межах від 1×10^{-5} до 5×10^{-4}. Логарифмічний масштаб є типовим для параметрів, які сильно впливають на процес навчання, оскільки змінюється порядок величини значення, що дозволяє краще досліджувати широкий діапазон можливих значень, від дуже малих до відносно більших;
2) num_train_epochs: діапазон від 3 до 10 епох забезпечує достатню кількість навчальних ітерацій для моделі, зберігаючи при цьому

ефективність навчання, оскільки більші значення можуть призвести до перенавчання;

3) weight_decay: діапазон від 0.0 до 0.1 дозволяє варіювати рівень регуляризації без надмірного її збільшення, що могло б призвести до недонавчання;
4) warmup_ratio - значення від 0.0 до 0.2 дозволяє моделі адаптуватися до більш складних задач без надмірних стрибків у швидкості навчання на початкових етапах.

Параметри пошуку вибрані таким чином, щоб покрити широкий спектр можливих варіацій, що можуть покращити навчання та результативність моделі на задачі класифікації текстів. Проте важливо, що значення було обрано таким чином, щоб забезпечити ефективне використання часу та ресурсів під час пошуку оптимальних параметрів.

Запуск пошуку гіперпараметрів здійснюється через функцію *trainer.hyperparameter_search*, де вказано:

1) direction - "maximize" означає, що шукаємо максимальне значення метрики (в даному випадку F1-міри);
2) backend - "optuna" вказує на використання саме бібліотеки *Optuna* для оптимізації;
3) n_trials - 10 спроб для оцінки різних комбінацій гіперпараметрів;
4) hp_space - використання раніше визначеного простору пошуку;
5) compute_objective - вказує на те, що для кожної спроби оптимізації ми максимізуємо значення F1-міри на валідаційній вибірці.

Код завершального етапу зображено на рисунку 20.

```
def optuna_hp_space(trial):
    return {
        "learning_rate": trial.suggest_float("learning_rate", 1e-5, 5e-4, log=True),
        "num_train_epochs": trial.suggest_int("num_train_epochs", 3, 10),
        "weight_decay": trial.suggest_float("weight_decay", 0.0, 0.1),
        "warmup_ratio": trial.suggest_float("warmup_ratio", 0.0, 0.2),
    }

best_run = trainer.hyperparameter_search(
    direction="maximize",
    backend="optuna",
    n_trials=10,
    hp_space=optuna_hp_space,
    compute_objective=lambda metrics: metrics["eval_f1"]
)
```

Рисунок 20. Код пошуку гіперпараметрів для визначеної моделі

У результаті оптимізації гіперпараметрів було визначено найкращу комбінацію параметрів, яка забезпечила максимальне значення F1-міри на валідаційній вибірці. Отримані значення дозволяють зробити такі висновки:

- learning_rate = 5.2×10^{-5}- помірно низький темп навчання, що забезпечує стабільну та поступову адаптацію ваг моделі;
- num_train_epochs = 3 - модель досягає найкращих результатів після трьох повних проходів по навчальній вибірці, що свідчить про достатню кількість ітерацій для збіжності без перенавчання;
- weight_decay = 0.015 - помірний рівень регуляризації, який запобігає перенавчанню, водночас не надмірно обмежуючи гнучкість моделі;
- warmup_ratio = 0.11 - близько 11% початкових кроків відведено на поступове збільшення темпу навчання, що сприяє стабільному старту оптимізації.

Загалом, знайдене поєднання параметрів балансує між точністю, стійкістю навчання та здатністю до узагальнення, однак, варто зазначити, що в процесі оптимізації гіперпараметрів можна було б вдосконалити й інші параметри, такі як аспекти підбору стратегії навчання, наприклад, параметри для оптимізації поетапного навчання. Однак подальша оптимізація таких

параметрів вимагала б значно більшого часу на тренування та валідацію, що не було б доцільним у рамках цієї роботи. Тому було вирішено зосередитись лише на вибраних гіперпараметрах, які продемонстрували найкращі результати для даної задачі.

3.4.2 Навчання моделі бінарної класифікації текстів за полярністю

На цьому етапі виконується навчання моделі бінарної класифікації, метою якої є автоматичне визначення емоційної полярності текстового контенту - позитивної або негативної. Для цього використовується повноцінний корпус з 3.6 мільйона текстових записів, який пройшов попередню обробку з урахуванням вимог до вхідних даних у трансформерних моделях, зокрема було приведено значення цільової змінної до числового формату, який відповідає умовам задачі класифікації, де 0 - негативне емоційне забарвлення, 1 - позитивне. Значення, що містили пропуски (NaN), були замінені на порожні рядки з метою забезпечення коректної токенізації. Крім того, для зниження рівня шуму в даних було вилучено всі символи, окрім латинських літер, арабських цифр та пробілів, що дозволило стандартизувати вхідний текст і зменшити кількість нерелевантної інформації. Така підготовка забезпечує стабільну роботу моделі та сприяє досягненню вищої якості класифікації.

Наступним кроком є підготовка попередньо обробленого набору даних до навчання на ньому. Для цього датасет *amazon_comments_train.csv*, завантажений у вигляді *DataFrame* за допомогою бібліотеки pandas, розділяється на навчальну та валідаційну частини у пропорції 90% до 10% відповідно. Важливо, що розподіл виконується зі збереженням балансу між класами, за допомогою параметра stratify=df['label'], що гарантує репрезентативність кожної з вибірок для обох класів. Параметр *random_state=42* забезпечує відтворюваність результатів розподілу. Після цього створюється об’єкт *DatasetDict*, який містить два піднабори: *“train”* і *“validation”*, що формуються з відповідних *DataFrame* за допомогою *Dataset.from_pandas*. Обидва піднабори мають скинутий індекс, щоб уникнути

конфліктів або дублювання при подальшій обробці. Така структура дозволяє зручно передавати дані в пайплайн навчання моделі, зокрема до об’єктів *Trainer* або токенізатора. Код для описаної логіки зображено на рисунку 21.

```
df = pd.read_csv("amazon_comments_train.csv")
train_texts, val_texts = train_test_split(df, test_size=0.1, stratify=df['label'], random_state=42)
dataset = DatasetDict({
    "train": Dataset.from_pandas(train_texts.reset_index(drop=True)),
    "validation": Dataset.from_pandas(val_texts.reset_index(drop=True))
})
```

Рисунок 21. Фрагмент коду для завантаження та розбиття набору даних на тренувальний та затверджувальний

Слід зазначити, що методика перехресного затверджування (k-fold cross-validation) не застосовувалася, оскільки обсяг вхідного набору даних у 3.6 мільйона записів є достатнім для навчання і валідації моделі без потреби в багаторазовому повторному розбитті, а також з метою зменшення обчислювальних витрат.

Після розбиття набору даних на тренувальний та затверджувальний, спочатку завантажується токенізатор за допомогою функції *AutoTokenizer.from_pretrained*, де як *model_name* вказано *"huawei-noah/TinyBERT_General_4L_312D"*. Цей токенізатор відповідає архітектурі моделі та знає, як саме розбивати текст на токени, яким чином їх кодувати та які спеціальні токени використовувати.

Далі визначається функція *tokenize_function*, яка застосовується до кожного прикладу в наборі даних. Вона викликає токенізатор з параметром *truncation=True*, що дозволяє автоматично обрізати надто довгі тексти до максимально допустимої довжини моделі.

Останнім кроком цього етапу є застосування методу *map* до об’єкта *dataset* (який містить навчальну та валідаційну вибірки), що дозволяє токенізувати всі приклади в пакетах *(batched=True)*. Результатом є новий об’єкт *tokenized_datasets*, що містить токенізовані дані, готові до подачі на вхід моделі для навчання. Код цього етапу зображено на рисунку 22.

```
model_name = "huawei-noah/TinyBERT_General_4L_312D"
tokenizer = AutoTokenizer.from_pretrained(model_name)

def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True)

tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

Рисунок 22. Фрагмент коду для завантаження токенізатора попередньо натренованої моделі TinyBERT

Після токенізації текстових даних наступним кроком є безпосереднє навчання моделі бінарної класифікації. Для цього спочатку завантажується попередньо навчена модель, попередньо вказана у змінній *model_name*, а саме *"huawei-noah/TinyBERT_General_4L_312D"*, адаптована для задачі класифікації послідовностей за допомогою класу *AutoModelForSequenceClassification*. Параметр *num_labels=2* вказує на те, що задача має два класи - позитивний і негативний.

Далі задаються параметри навчання за допомогою класу *TrainingArguments*. Серед основних налаштувань:

- виведення результатів і збереження моделі відбувається після кожної епохи (*eval_strategy* та *save_strategy*);
- розміри батчів для навчання та валідації (*per_device_train_batch_size*, *per_device_eval_batch_size*) встановлено відповідно 16 і 64. Ці дані було отримано емпіричним шляхом, під час визначення можливостей аппаратного забезпечення;
- параметр *fp16=True* активує тренування з використанням 16-бітної точності, що прискорює обчислення і зменшує споживання пам’яті. Необхідність використання цього параметру також визначена емпірично;
- кількість епох (*num_train_epochs*) задана відповідно до даних отриманих на етапі підбору гіперпараметрів і дорівнює 3;
- темп навчання (*learning_rate*) так само визначено за допомогою

оптимізації гіперпараметрів і дорівнює 5.2e-5;

- регуляризація через ваговий коефіцієнт (*weight_decay*) встановлено відповідно до підібраних гіперпараметрів і дорівнює 0.015;
- співвідношення "розігріву" швидкості навчання (*warmup_ratio*) встановлено відповідно до підібраних гіперпараметрів і дорівнює 0.11.

Функція *compute_metrics* визначає метрики, які обчислюються після кожної валідації: точність (*accuracy*), влучність (*precision*), повнота (*recall*) та F1-міра - основна метрика для відбору найкращої моделі. Для забезпечення правильного оброблення вхідних даних при формуванні батчів використовується об’єкт *DataCollatorWithPadding*, який динамічно вирівнює довжину токенізованих послідовностей усередині кожного батчу.

Усі описані на цьому етапі компоненти об’єднуються в об’єкт *Trainer*, який керує процесом навчання. У нього передається модель, параметри навчання, навчальна й валідаційна вибірки, токенізатор для обробки, колатор для пакетування даних та функція для обчислення метрик. Навчання моделі запускається викликом *trainer.train()*, після чого натренована модель зберігається локально у вказаній директорії *"./tinybert-emotion-model"*.

Відповідний фрагмент коду зображено на рисунку 23.

```python
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
training_args = TrainingArguments(
    output_dir="./results_tinybert",
    eval_strategy="epoch",
    save_strategy="epoch",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    learning_rate=5.2e-5,
    num_train_epochs=3,
    weight_decay=0.015,
    warmup_ratio=0.11,
    fp16=True,
    logging_dir='./logs',
    logging_steps=50,
    save_total_limit=1,
    load_best_model_at_end=True,
    metric_for_best_model="f1",
)

def compute_metrics(p: EvalPrediction):
    preds = p.predictions.argmax(axis=-1)
    labels = p.label_ids
    return {
        "accuracy": accuracy_score(labels, preds),
        "precision": precision_score(labels, preds),
        "recall": recall_score(labels, preds),
        "f1": f1_score(labels, preds),
    }

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    processing_class=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
trainer.train()
trainer.save_model("./tinybert-emotion-model")
```

Рисунок 23. Фрагмент коду для навчання та збереження моделі класифікації на базі TinyBERT

Аналіз результатів навчання за метрикою F1-міри та втратами, зображеними на рисунку 24, показує, що:

1) F1-міра:
    a) протягом трьох епох спостерігається поступове покращення F1-міри, що свідчить про стабільне навчання моделі;
    b) у першій епосі F1-міра дорівнює 0.9419, в другій - 0.9495, а в третій - 0.9510. Це вказує на незначне, але стабільне покращення результатів, що свідчить про ефективність обраних параметрів навчання;

c) найвищий показник F1-міри досягнуто на третій епосі (0.9510), що є найкращим результатом.

2) Втрати:

a) тренувальна втрата (Training Loss) зменшується від 0.2034 до 0.1482, що вказує на поліпшення якості моделі в процесі навчання;

b) валідаційна втрата (Validation Loss) на першій епосі становить 0.1781, на другій - 0.1634, а на третій підвищується до 0.1923. Це зростання валідаційних втрат на останній епосі може свідчити про початок перенавчання моделі, але воно є незначним;

На третій епосі модель досягла найкращих результатів за усіма метриками:

- точність - 0.9511;
- влучність - 0.9532;
- повнота - 0.9488;
- F1-міра - 0.9510.

Однак в цей же час спостерігається невелике зростання валідаційних втрат, що може бути ознакою перенавчання. Це може свідчити про те, що модель досягла певного рівня адаптації до тренувальних даних, але вже почала погіршувати свої результати на невідомих даних.

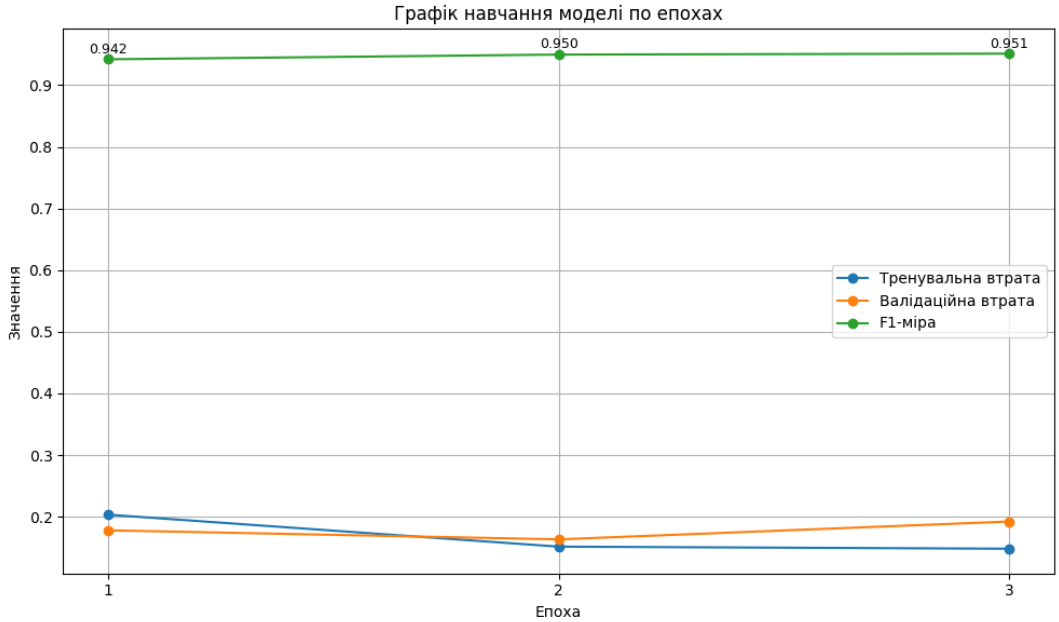


Рисунок 24. Графік навчання моделі по епохах

3.4.3 Верифікація моделі бінарної класифікації текстів за полярністю

Верифікація моделі бінарної класифікації текстів за полярністю буде проводитись на тестовому наборі даних, який не використовувався під час навчання моделі. Це дозволить об'єктивно оцінити ефективність моделі на нових, раніше невідомих даних, що є важливим кроком для перевірки її здатності до узагальнення. Тестовий набір даних складається з коментарів Amazon, що мають помічені категорії полярності (позитивні чи негативні), і на його основі буде проводитись обчислення основних метрик оцінки моделі, таких як точність, влучність, повнота та F1-міра. Тестовий набір даних був попередньо оброблений, так само як і будь який інший набір, що був використаний для моделі.

Перевірка моделі на тестовому наборі даних, відокремленого від тренувального і затверджувального наборів, є стандартною практикою, яка дозволяє виявити перенавчання моделі і забезпечує реалістичну оцінку показників моделі на практичних, а не тільки теоретичних даних.

Першим етапом верифікації моделі є завантаження тестового набору даних, який містить раніше невикористані під час навчання записи, та перетворення його у формат Dataset. Це дозволяє зручно використовувати його

в подальшому для токенізації, оцінювання та прогнозування. Відповідний код зображено на рисунку 25.

```
df_test = pd.read_csv("amazon_comments_test.csv")

dataset_test = Dataset.from_pandas(df_test)
```

Рисунок 25. Фрагмент коду для завантаження та підготовки тестового набору даних

Наступним кроком після завантаження тестового датасету є підготовка даних до передбачення за допомогою токенізатора, збереженого разом із тренованою моделлю, тобто використовується той самий токенізатор, що й під час навчання, який завантажується із директорії *./tinybert-emotion-model*.

Оскільки модель не має заздалегідь визначеної максимальної довжини, для уникнення помилок і забезпечення однакового розміру входів використовується параметр *max_length=512*, що є типовим обмеженням для BERT-подібних моделей. Крім того, застосовується параметр *padding=True* для вирівнювання довжини усіх прикладів у батчі. Процедура токенізації виконується за допомогою методу map із параметром *batched=True*, що значно пришвидшує обробку великого набору даних. Фрагмент коду з описаною логікою зображено на рисунку 26.

```
model_path = "./tinybert-emotion-model"
tokenizer = AutoTokenizer.from_pretrained(model_path)

def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True, max_length=512, padding=True)

tokenized_test = dataset_test.map(tokenize_function, batched=True)
```

Рисунок 26. Фрагмент коду для токенізації тестового набору даних за допомогою попередньо збереженого токенізатора

Після завершення токенізації тестового набору даних виконується завантаження попередньо тренованої моделі бінарної класифікації текстів. Для цього використовується клас *AutoModelForSequenceClassification* із бібліотеки transformers. Для здійснення передбачення створюється об’єкт *Trainer*, який

пов’язується з моделлю, та викликається метод *predict*, що застосовується до токенізованого тестового набору. В результаті отримується об'єкт *predictions*, що містить необроблені вихідні значення моделі, а також відповідні мітки, якщо вони є в наборі даних. Відповідний код зображено на рисунку 27.

```
model = AutoModelForSequenceClassification.from_pretrained(model_path)
trainer = Trainer(model=model)
predictions = trainer.predict(tokenized_test)
```

Рисунок 27. Фрагмент коду для отримання моделі та виконання передбачення на тестовому наборі

Після обробки даних, було отримано такі результати передбачення на тестовому наборі даних:

- точність - 0.9517;
- влучність - 0.9539;
- повнота - 0.9494;
- F1-міра - 0.9516.

Показники ключових характеристик засвідчили високу ефективність моделі. Точність у 0.9517 свідчить про правильність більшості передбачень, тоді як влучність з показником 0.9539 підтверджує низький рівень хибнопозитивних результатів, а повнота 0.9494 демонструє здатність моделі виявляти майже всі позитивні приклади. F1-міра 0.9516 вказує на збалансованість між влучністю та повнотою, що загалом характеризує модель як надійну та стабільну у класифікації емоційної полярності текстів. Для наглядного розуміння характеристик, було побудовано графічні представлення, а саме матрицю невідповідностей (рисунок 28), криву влучності-повноти (рисунок 29) та ROC-криву та зону під нею (рисунок 30).

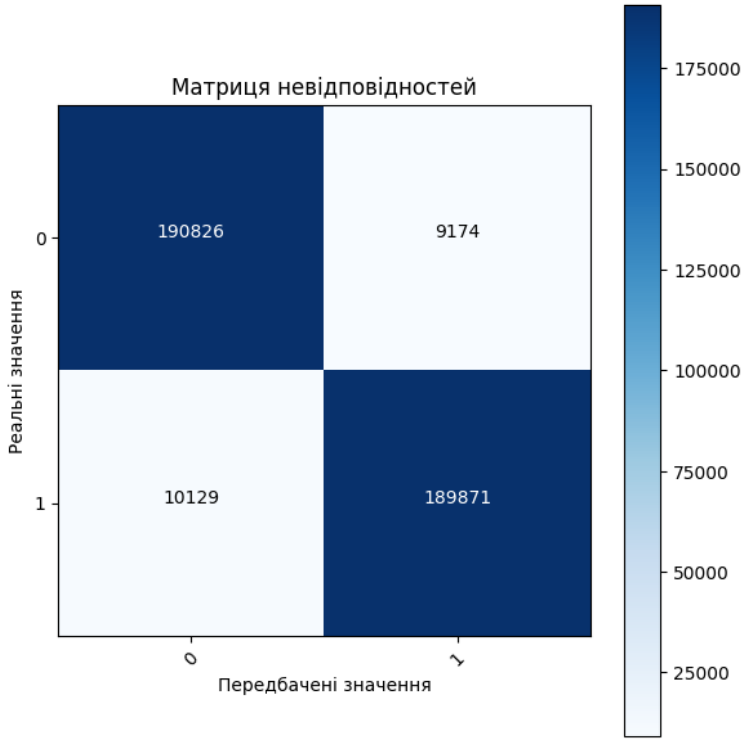


Рисунок 28. Матриця невідповідностей тестованої моделі

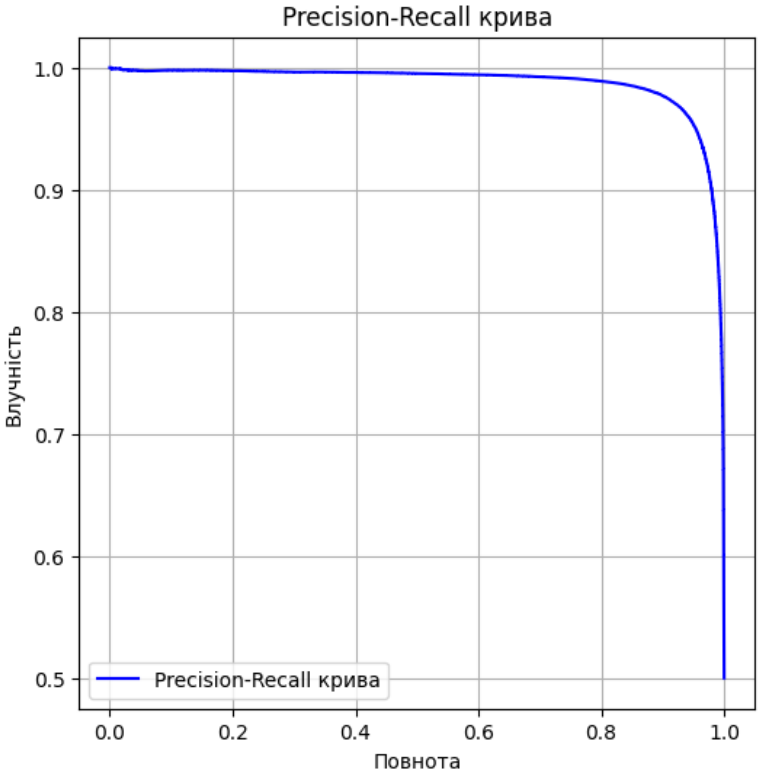


Рисунок 29. Precision-Recall крива

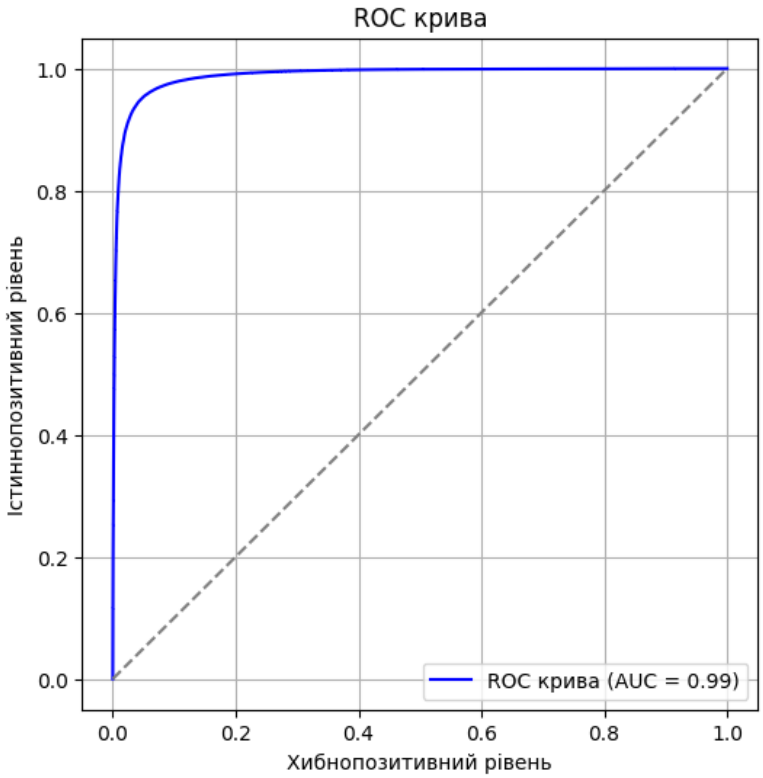


Рисунок 30. Крива ROC та AUC

Висновки

У цьому розділі було здійснено всебічний аналіз даних, використаних для побудови системи класифікації емоційної полярності текстів, а саме масштабного набору відгуків користувачів на платформі Amazon, що дозволило забезпечити змістовну основу для навчання моделі. Ретельне попереднє опрацювання текстової інформації сприяло зменшенню шумів та підвищенню якості вхідного представлення. Ці етапи відповідають першому та другому крокам методології CRISP-DM, зокрема розумінню бізнес-проблеми та підготовці даних.

Окрему увагу приділено огляду програмних інструментів, зокрема мові програмування python та відповідним бібліотекам для задач аналізу даних, а також середовищу для створення та виконання документів - Jupyter Notebook. Описані інструменти стали ключовими засобами реалізації проєкту. На початковому етапі побудови моделі проводилась гіперпараметрична оптимізація на репрезентативній підвибірці з 50 000 зразків, що було зроблено з метою зниження обчислювальних витрат. Отримані оптимальні параметри були

використані для повноцінного навчання моделі на повному наборі даних, що налічує 3.6 мільйонів зразків. Ці кроки відображають етап моделювання в рамках CRISP-DM.

На завершальному етапі модель було протестовано на незалежному наборі тестових даних, також сформованому з відгуків Amazon. За результатами оцінювання модель продемонструвала високі показники ефективності: точність, влучність, повнота та F1-міра перебувають на рівні понад 0.95, що дозволяє зробити висновок про високу ефективність побудованої системи класифікації, її здатність до узагальнення та практичну придатність. Цей етап відповідає останньому етапу CRISP-DM - оцінці моделі, де було здійснено фінальну перевірку її якості та працездатності на реальних даних.

Таким чином, всі етапи розробки моделі чітко співвідносяться з основними кроками методології CRISP-DM, що забезпечило системний підхід до вирішення поставленої задачі.

РОЗДІЛ 4. ЗАСТОСУВАННЯ РОЗРОБЛЕНОЇ МОДЕЛІ ДЛЯ ОБРОБКИ ВІДГУКІВ В СФЕРІ E-COMMERCE

4.1 Створення та розгортання API для роботи з моделлю

4.1.1 Створення API

Одним із можливих варіантів використання моделі є створення веб-додатку для доступу до її функціонало та можливості інтеграції його в існуючі продукти, зокрема електронної комерції.

Для розробки додатку, які і для моделювання, було використано мову програмування python та фреймворк Flask, який забезпечує можливість швидкого та зручного написання серверної логіки та обробки HTTP запитів.

Веб-додаток представляє собою простий сервер з одним ендпоінтом та логікою роботи з моделлю машинного навчання, отриманої в результаті цієї роботи. Але, не зважаючи на його простоту, доцільно розглянути логіку обробки запитів. Єдиний ендпоінт який має додаток, обробляє запити з методом *POST* та приймає дані у форматі *application/json*. Тіло запиту складається з об'єкта, який в свою чергу має одне поле *texts*, що має містити списку текстів, строкового типу даних. Приклад тіла запиту зображено на рисунку 31.

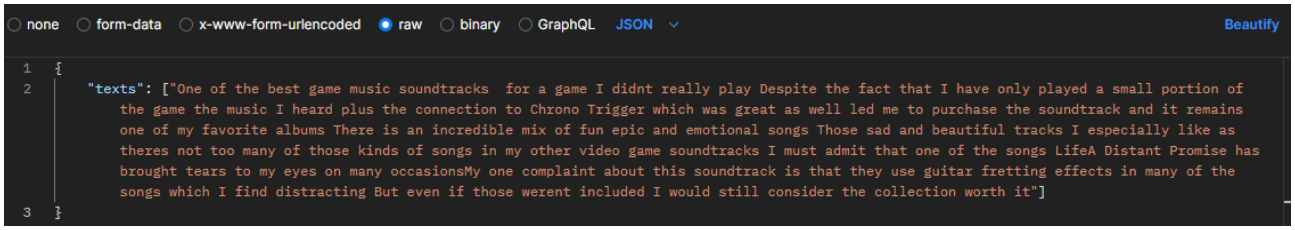


Рисунок 31. Тіло запиту до веб-додатку

При старті додатку модель, збережена локально, завантажується у пам'ять, оскільки не доцільно робити це при кожному запиті на прогнозування через обчислювальні витрати на це.

Після того як модель доступна для використання, сервер готовий отримувати запити. При отриманні запиту відбувається обробка даних, які

обов'язково мають бути у форматі JSON, інакше клієнт отримає відповідь з кодом клієнтської помилки. Наступним кроком тіло запита трансформується у внутрішній об'єкт python та відправляється у функцію *extract_texts_from_request_data*, яка в свою чергу валідує отримані дані та повертає список текстів, попередньо очистивши його від пунктуації та HTML тегів. У випадку якщо дані не валідні, функція повертає помилку з відповідною причиною. Код описаної функції зображено на рисунку 32.

```python
def extract_texts_from_request_data(data):
    texts_to_predict = []

    if 'texts' in data:
        if isinstance(data['texts'], list):
            texts_to_predict = [t for t in data['texts'] if (isinstance(t, str) and bool(t.strip()))]

            if data['texts'] and not texts_to_predict:
                raise ValueError("'texts' list contains elements that are not non-empty strings.")
        else:
            raise ValueError("'texts' should be a list.")
    else:
        raise ValueError("Missing field 'texts'. Expected a list of non-empty strings.")

    return list(map(clean_text, texts_to_predict))
```

Рисунок 32. Фрагмент коду із функцією для отримання списку текстів із тіла запита

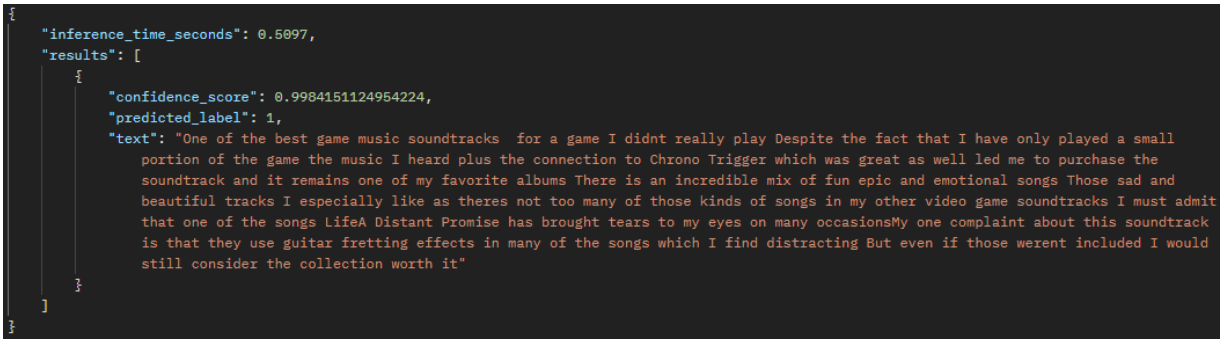
Після того як дані запиту було успішно підготовано для прогнозування, відбувається прогнозування за допомогою функції *perform_prediction*, яка звертається до моделі машинного навчання та повертає сирі результати. Код функції передбачення зображено на рисунку 33.

```python
def perform_prediction(texts_list):
    if pipeline is None:
        error_msg = "Model and pipeline are not loaded. Server initialized incorrectly."
        raise RuntimeError(error_msg)

    if not texts_list:
        return []

    try:
        results = pipeline(texts_list)
        return results

    except Exception as e:
        raise RuntimeError(f"Prediction failed: {e}")
```

Рисунок 33. Фрагмент коду із функцією для виконання передбачення за допомогою моделі машинного навчання

Наступним кроком сирі дані передбачення форматуються у JSON за

допомогою функції *format_prediction_results*. Отримані дані повертаються у відповіді на запит у форматі зображеному на рисунку 34.



```
{
    "inference_time_seconds": 0.5097,
    "results": [
        {
            "confidence_score": 0.9984151124954224,
            "predicted_label": 1,
            "text": "One of the best game music soundtracks  for a game I didnt really play Despite the fact that I have only played a small
                portion of the game the music I heard plus the connection to Chrono Trigger which was great as well led me to purchase the
                soundtrack and it remains one of my favorite albums There is an incredible mix of fun epic and emotional songs Those sad and
                beautiful tracks I especially like as theres not too many of those kinds of songs in my other video game soundtracks I must admit
                that one of the songs LifeA Distant Promise has brought tears to my eyes on many occasionsMy one complaint about this soundtrack
                is that they use guitar fretting effects in many of the songs which I find distracting But even if those werent included I would
                still consider the collection worth it"
        }
    ]
}
```


Рисунок 34. Формат відповіді на запит на прогнозування

Слід зазначити, що ендпоінт підтримує обробку як одного так і багатьох текстів у запиті, що може бути зручним у випадках коли необхідно обробляти декілька текстів за раз. Код всього ендпоінту зображено на рисунку 35.

```python
@app.route('/predict', methods=['POST'])
def predict():
    if not request.is_json:
        print("Received non-JSON request.", file=sys.stderr)
        return jsonify({"error": "Unsupported data type. Use JSON (Content-Type: application/json)"}), 415

    data = request.get_json()

    try:
        texts_to_predict = extract_texts_from_request_data(data)
    except ValueError as e:
        return jsonify({"error": f"Data validation error: {e}"}), 400

    if not texts_to_predict:
        return jsonify({"inference_time_seconds": 0.0, "results": []}), 200

    try:
        start_time = time.time()

        raw_prediction_results = perform_prediction(texts_to_predict)

        end_time = time.time()
        inference_duration = end_time - start_time

    except RuntimeError as e:
        return jsonify({"error": "Prediction internal error", "details": str(e)}), 500
    except Exception as e:
        return jsonify({"error": "Unexpected internal error during prediction", "details": str(e)}), 500

    try:
        formatted_results_list = format_prediction_results(raw_prediction_results, texts_to_predict)
    except Exception as e:
        return jsonify({"error": "Internal server error during result formatting", "details": str(e)}), 500

    final_response = {
        "inference_time_seconds": round(inference_duration, 4),
        "results": formatted_results_list
    }

    return jsonify(final_response), 200
```

Рисунок 35. Фрагмент коду із ендпоінтом для обробки запиту на прогнозування

4.1.2 Контейнеризація та розгортання локально

Для запуску додатку було вирішено використовувати Docker, оскільки це є доцільним насамперед з огляду на потребу в ізольованому, відтворюваному та контрольованому середовищі виконання. Розробка системи на базі глибокого навчання, яка працює з попередньо натренованими моделями, передбачає залежність від конкретних версій бібліотек, середовищ виконання Python та системних бібліотек, а також доступ до обчислювальних ресурсів. Docker дозволяє «запакувати» весь додаток разом з усіма його залежностями в образ, що гарантує однакову поведінку незалежно від конфігурації хост-системи. Це особливо важливо, коли модель розгортатиметься у продакшн-середовищі або на хмарній платформі, де можуть бути обмеження щодо встановлення стороннього програмного забезпечення або неможливість точного відтворення середовища локальної розробки.

Крім того, Docker забезпечує високу портативність і спрощує масштабування. Наприклад, створення кількох інстансів API-сервісу для обслуговування запитів з балансуванням навантаження відбувається набагато легше завдяки стандартизованому контейнеру. Це особливо актуально у випадках, коли сервіс має обробляти велику кількість запитів або інтегрується в мікросервісну архітектуру. У поєднанні з такими засобами, як Docker Compose або Kubernetes, це дозволяє легко розгортати систему в кластерному середовищі, автоматизувати оновлення образів, керувати масштабуванням і моніторингом.

Ще однією важливою перевагою є безпека. Контейнери працюють у відокремленому від основної системи середовищі, що знижує ризик конфлікту залежностей, сторонніх втручань або пошкодження системних компонентів.

Структурно система для розгортання додатку складається з кількох взаємопов’язаних компонентів, кожен з яких виконує окрему функцію в процесі побудови, запуску та експлуатації API-сервісу. Основним елементом є сам додаток, реалізований як Flask-сервер, який обробляє HTTP-запити та взаємодіє з попередньо натренованою моделлю на основі TinyBERT. Друга складова - це

збережена модель, що містить ваги, конфігураційні файли та токенізатор, необхідні для функціонування класифікаційного пайплайна. Усі ці компоненти використовуються в межах єдиного Docker-образу, який створюється за допомогою відповідного Dockerfile. Цей файл описує інструкції щодо побудови контейнера, включаючи завантаження залежностей, копіювання коду, клонування або копіювання моделі, а також визначення робочого середовища. Схематично описану сткруктуру зображено на рисунку 36.

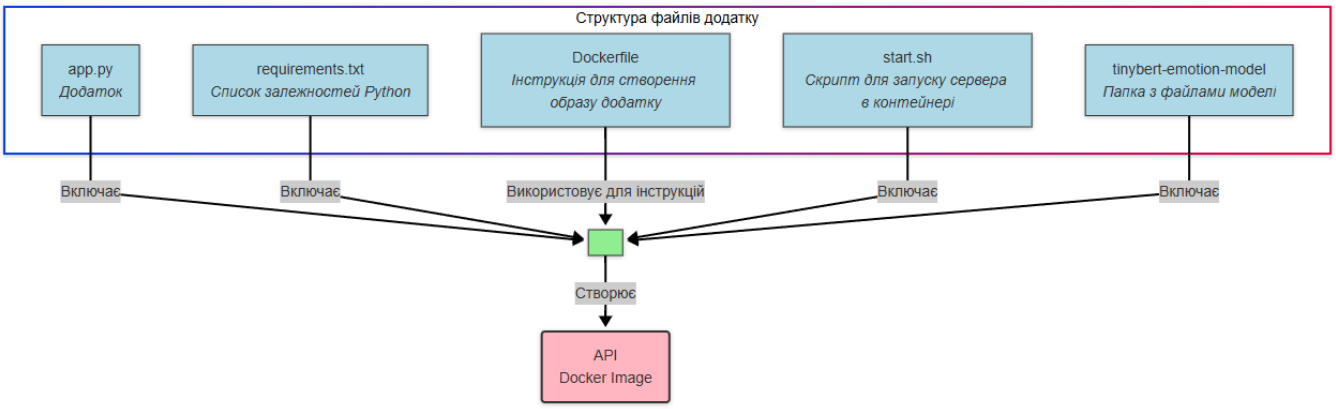


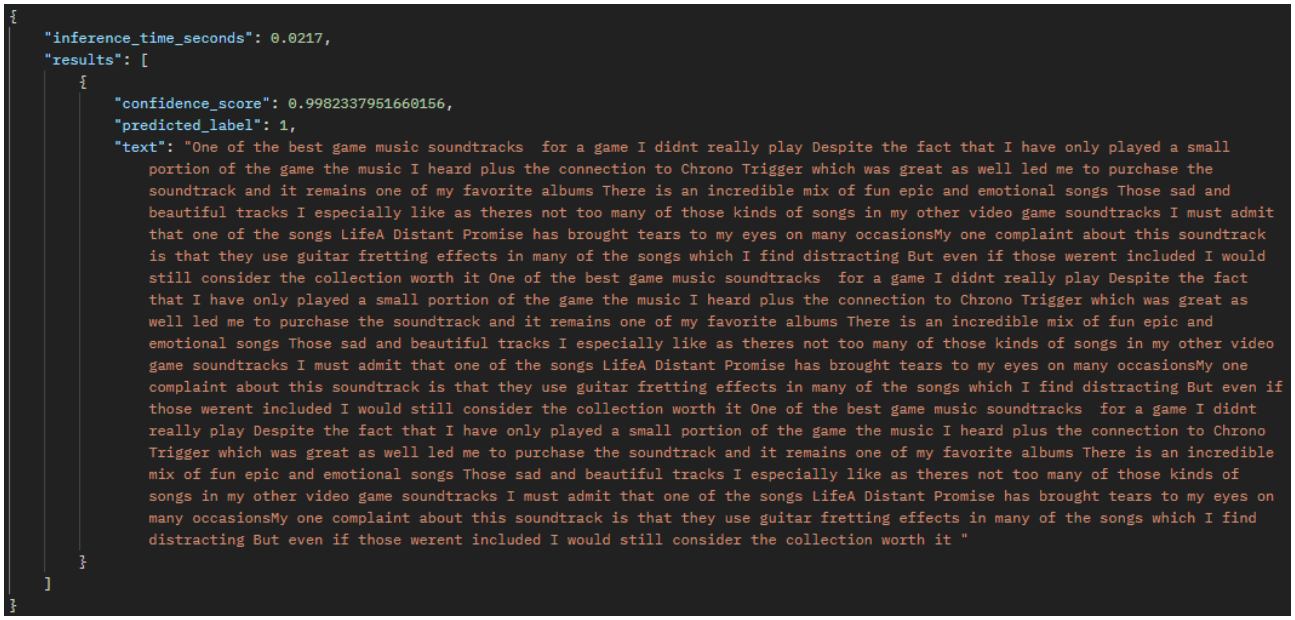
Рисунок 36. Структура системи для побудови образу додатка

Окремий скрипт відповідає за побудову Docker-образу на основі зазначеного Dockerfile. Він може бути інтегрований у систему безперервної інтеграції або виконуватись вручну для генерації ізольованого середовища запуску. Після цього другий скрипт, скрипт запуску, використовується для ініціалізації контейнера. Нарешті, всередині самого контейнера виконується спеціальний стартовий скрипт, який автоматично спрацьовує при його запуску. Саме цей скрипт ініціалізує модель, проводить попередню перевірку залежностей і, зрештою, запускає сервер через Gunicorn - WSGI-сумісний HTTP-сервер, що забезпечує ефективну обробку паралельних запитів і стабільну роботу додатку у продакшн-середовищі. Таким чином, архітектура системи охоплює повний цикл - від побудови середовища до автоматичного розгортання та запуску сервісу.

Оскільки TinyBERT було обрано як модель з підвищеною продуктивністю, для симуляції роботи у середовищі з обмеженими ресурсами,

локально було розгорнуто контейнер із додатком та встановлено штучні обмеження, а саме відсутність графічного процесора та виділення лише 1-го процесорного ядра.

Для перевірки роботи додатку було виконано запит, результат якого зображено на рисунку 37.



```
{
    "inference_time_seconds": 0.0217,
    "results": [
        {
            "confidence_score": 0.9982337951660156,
            "predicted_label": 1,
            "text": "One of the best game music soundtracks  for a game I didnt really play Despite the fact that I have only played a small
                portion of the game the music I heard plus the connection to Chrono Trigger which was great as well led me to purchase the
                soundtrack and it remains one of my favorite albums There is an incredible mix of fun epic and emotional songs Those sad and
                beautiful tracks I especially like as theres not too many of those kinds of songs in my other video game soundtracks I must admit
                that one of the songs LifeA Distant Promise has brought tears to my eyes on many occasionsMy one complaint about this soundtrack
                is that they use guitar fretting effects in many of the songs which I find distracting But even if those werent included I would
                still consider the collection worth it One of the best game music soundtracks  for a game I didnt really play Despite the fact
                that I have only played a small portion of the game the music I heard plus the connection to Chrono Trigger which was great as
                well led me to purchase the soundtrack and it remains one of my favorite albums There is an incredible mix of fun epic and
                emotional songs Those sad and beautiful tracks I especially like as theres not too many of those kinds of songs in my other video
                game soundtracks I must admit that one of the songs LifeA Distant Promise has brought tears to my eyes on many occasionsMy one
                complaint about this soundtrack is that they use guitar fretting effects in many of the songs which I find distracting But even if
                those werent included I would still consider the collection worth it One of the best game music soundtracks  for a game I didnt
                really play Despite the fact that I have only played a small portion of the game the music I heard plus the connection to Chrono
                Trigger which was great as well led me to purchase the soundtrack and it remains one of my favorite albums There is an incredible
                mix of fun epic and emotional songs Those sad and beautiful tracks I especially like as theres not too many of those kinds of
                songs in my other video game soundtracks I must admit that one of the songs LifeA Distant Promise has brought tears to my eyes on
                many occasionsMy one complaint about this soundtrack is that they use guitar fretting effects in many of the songs which I find
                distracting But even if those werent included I would still consider the collection worth it "
        }
    ]
}
```


Рисунок 37. Відповідь від сервера

Як можна бачити на рисунку, час на виконання передбачення склав всього 0.0217 секунди, що дозволить без будь яких проблем використовувати модель навіть на системах з незначними ресурсами та без GPU або TPU, проте, за необхідності, наприклад для гігантів електронної комерції, можна розширити обчислювальні можливості.

4.1.3 Розгортання моделі на GCP Endpoints

Розгортання сучасних сервісів у хмарних платформах стало надзвичайно зручним та ефективним рішенням для розробників і компаній. Завдяки масштабованості, високій надійності, а також широкому набору інструментів для автоматизації та моніторингу, хмарні сервіси значно спрощують підтримку і розгортання застосунків.

Для демонстрації процесу впровадження розробленого додатку за

допомогою хмарної платформи було обрано GCP (Google Cloud Platform). Зокрема, сервіс Vertex AI, який надає інтегровану інфраструктуру для розгортання моделей машинного навчання через Endpoints, що дозволяє швидко запускати, масштабувати і керувати API з мінімальними зусиллями.

Процес розгортання моделі складається з кількох основних етапів. Спершу потрібно підготовувати Docker образ створенного додатку, що вже було зроблено при демонстрації локального розгортання. Цей образ буде потрібен на наступних етапах.

Для роботи з GCP зручно використовувати gcloud CLI, що дозволяє працювати з хмарною платформою без необхідності використовувати графічний інтерфейс. Після встановлення цього застосунку, потрібно провести аутентифікацію. Команду для цього зображено на рисунку 38.

```
$ gcloud auth login
```

Рисунок 38. Команда для виконання аутентифікації

Після цього необхідно встановити активний проєкт, що робиться за допомогою команди зображеної на рисунку 39.

```
$ gcloud config set project storied-toolbox-459912-b6
```

Рисунок 39. Команда для встановлення активного проєкта

Наступним кроком є активація необхідних сервісів в рамках GCP, як зображено на рисунку 40.

```
$ gcloud services enable \
    aiplatform.googleapis.com \
    artifactregistry.googleapis.com \
    cloudbuild.googleapis.com
```

Рисунок 40. Команда для активації необхідних сервісів GCP

Додатково, потрібно надати сервісному акаунту Vertex AI права на доступ до Artifact Registry, інакше з’явиться помилка. Для цього потрібно виконати команду зображену на рисунку 41.

```
$ export PROJECT_ID=storied-toolbox-459912-b6
export PROJECT_NUMBER=$(gcloud projects describe $PROJECT_ID --format="value(projectNumber)")
export SERVICE_ACCOUNT="service-${PROJECT_NUMBER}@gcp-sa-aiplatform.iam.gserviceaccount.com"

gcloud projects add-iam-policy-binding $PROJECT_ID \
    --member="serviceAccount:$SERVICE_ACCOUNT" \
    --role="roles/artifactregistry.reader"
```

Рисунок 41. Команда для надання необхідних прав доступу

Для завантаження образу Docker, створеного раніше, потрібно спочатку на Artifact Registry створити нове сховище за допомогою команди зображеної на рисунку 42.

```
$ gcloud artifacts repositories create sentiment-analysis-api-repo \
  --repository-format=docker \
  --location=us-central1 \
  --description="Docker repository for sentiment analysis API"
```

Рисунок 42. Команда для створення сховища на Artifact Registry

На створений репозиторій необхідно завантажити Docker образ, попередньо надавши йому спеціальний тег, за яким система зрозуміє куди його потрібно завантажити. Щоб це зробити, потрібно виконати команди зображені на рисунку 43.

```
$ docker tag sentiment-analysis-api us-central1-docker.pkg.dev/storied-toolbox-459912-b6/sentiment-analysis-repo/sentiment-analysis-api
docker push us-central1-docker.pkg.dev/storied-toolbox-459912-b6/sentiment-analysis-api-repo/sentiment-analysis-api
```

Рисунок 43. Кодманда для завантаження Docker образа на Artifact Registry

Після завантаження образа, на його базі необхідно створити модель за допомогою команди на рисунку 44. Для розгортання моделі, в код API необхідно було додати новий ендпоінт - */health*, який використовується платформою для перевірки готовності API опрацьовувати запити. Також було змінено логику обробки запитів на прогнозування, а саме додано коректну обробку поля *instances*, яке Vertex AI має за стандарт для тіла запиту.

```
$ gcloud beta ai models upload \
  --region=us-central1 \
  --display-name=sentiment-analysis-api-model \
  --container-image-uri=us-central1-docker.pkg.dev/storied-toolbox-459912-b6/sentiment-analysis-api-repo/sentiment-analysis-api \
  --container-ports=8080 \
  --container-predict-route=/predict \
  --container-health-route=/health
```

Рисунок 44. Команда для створення моделі на базі образа

Перед тим як розгортати модель на ендпоінті, його спочатку потрібно створити. Для цього потрібно виконати команду зображену на рисунку 45.

```
$ gcloud beta ai endpoints create \
  --region=us-central1 \
  --display-name=sentiment-api-endpoint
```

Рисунок 45. Команда для створення ендпоінта

Як тільки ендпоінт було створено на нього розгортається модель. Команда для цього зображена на рисунку 46. Як можна побачити на цьому рисунку, було обрано найслабшу із доступних машин, для демонстанції того, що розроблена модель ефективно працює на обмежених ресурсах. Час розгортання моделі може варіюватися від декількох хвилин до декількох десятків хвилин, в залежності від розміру образа.

```
$ gcloud beta ai endpoints deploy-model 4445744425028550656 \
    --region=us-central1 \
    --model=6167401513242066944 \
    --display-name=sentiment-analysis-deployment \
    --traffic-split=0=100 \
    --machine-type=n1-standard-2
```

Рисунок 46. Команда для розгортання моделі на ендпоінт

Після того як процес розгортання успішно завершився, сервер готовий обробляти запити. Для перевірки було відправлено запит за допомогою HTTP клієнта Postman. Результат можна побачити на рисунку 47.

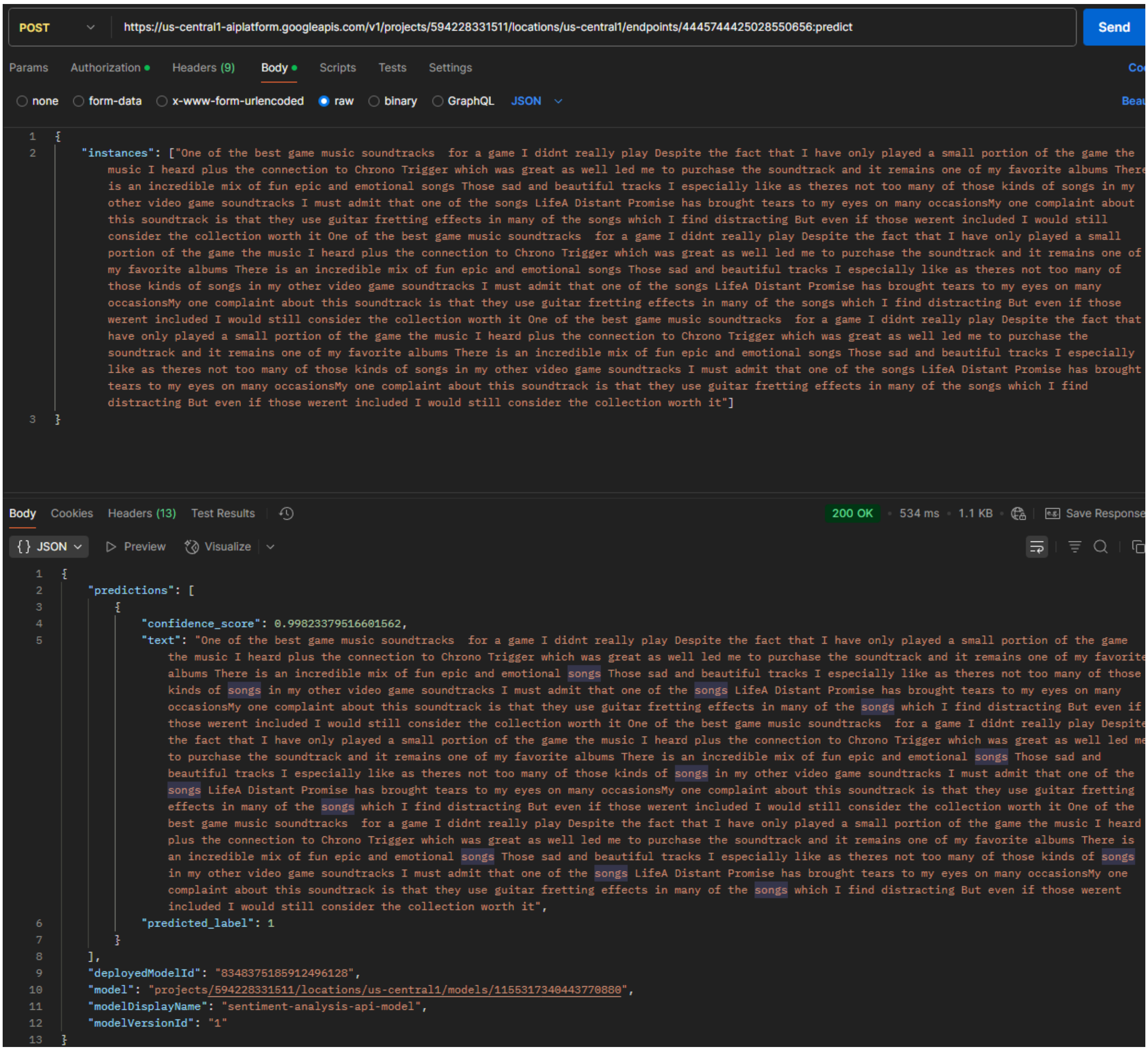


Рисунок 47. Запит та відповідь від розгорнутого API

Розгортання моделі на платформі Vertex AI забезпечує зручний і масштабований спосіб надання доступу до машинного навчання через хмарні сервіси Google. Використання готових інструментів значно спрощує процес розгортання, автоматизуючи керування інфраструктурою, балансування навантаження та моніторинг. Проте для успішного розгортання важливо чітко дотримуватися вимог платформи, таких як коректне налаштування форматів запитів та маршрутів здоров’я контейнера. Врахування цих нюансів дозволяє мінімізувати помилки на етапі деплою та забезпечити стабільну роботу моделі у продакшені. Таким чином, Vertex AI є ефективним вибором для організації

продуктивних ML-сервісів із гнучкістю та масштабованістю, властивими сучасним хмарним рішенням.

4.2 Інтеграція розробленої моделі в існуючі системи e-commerce

Інтеграція розробленої моделі класифікації емоційної полярності у наявні e-commerce рішення є критично важливим етапом для досягнення її прикладної цінності. На практиці система аналізу відгуків повинна безперешкодно взаємодіяти з наявною інфраструктурою, яка зазвичай охоплює бази даних відгуків, API-шлюзи, модулі зворотного зв’язку та інтерфейси адміністративного доступу.

Для забезпечення гнучкості та масштабованості запропоновано реалізувати модель у вигляді окремого веб-сервісу, що працює за архітектурою REST API. Таке рішення дозволяє відокремити логіку обробки тексту від основного додатку, тим самим зменшуючи ризики для продуктивного середовища при оновленнях або експериментах з моделлю. Основними компонентами інтеграції є (рисунок 48):

- сервіс моделі, розгорнутий у хмарному середовищі (наприклад, Google Cloud Vertex AI, Cloud Run або альтернативні рішення);
- внутрішні API платформи e-commerce, які відповідають за отримання й обробку відгуків користувачів;
- логіка маршрутизації запитів до сервісу класифікації;
- база даних або аналітичний модуль, у якому зберігаються результати класифікації та використовуються для подальшої візуалізації чи автоматичних рішень (наприклад, фільтрації негативних відгуків, модерації тощо).

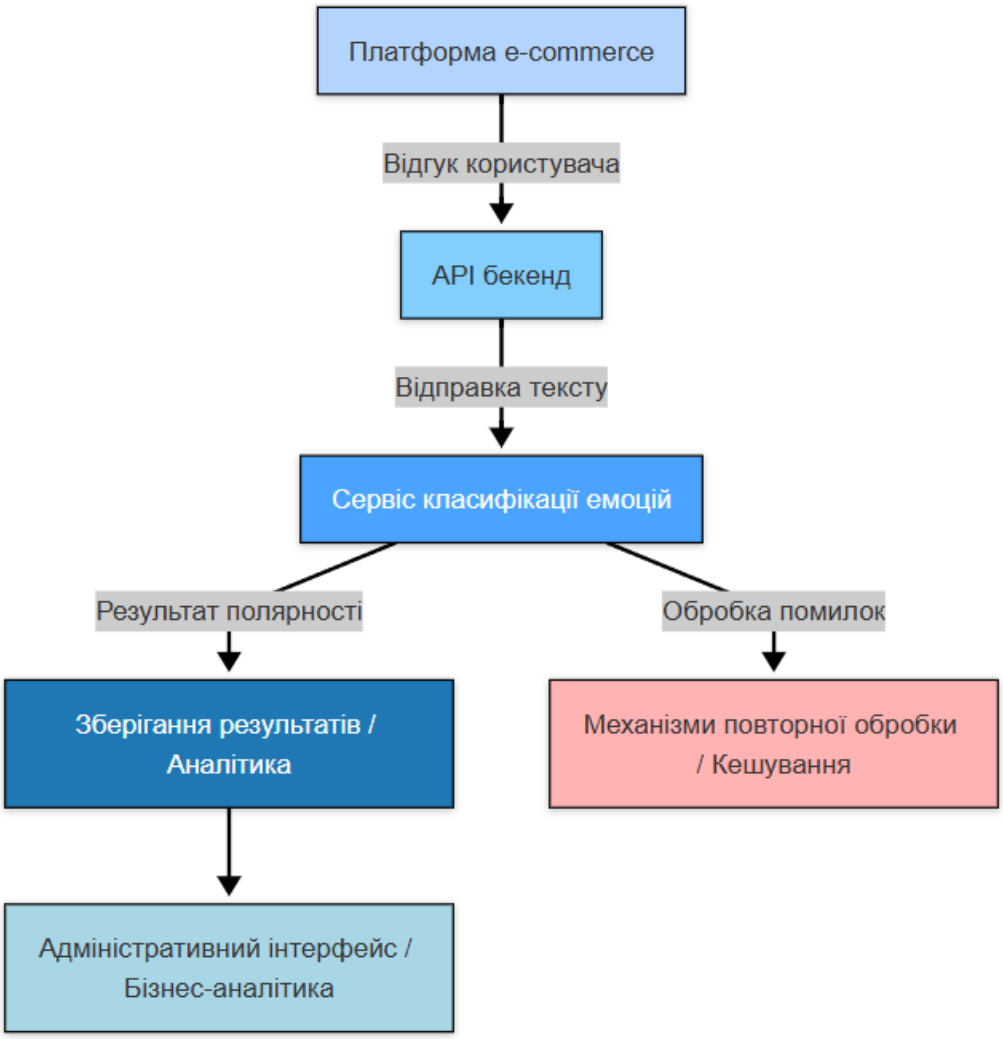


Рисунок 48. Схема взаємодії компонентів моделі емоційного аналізу з платформою e-commerce

Реалізація інтеграції не потребує значної модифікації існуючої системи. У більшості випадків достатньо доповнити бекенд новим ендпоінтом, який передає текст відгуку на обробку й отримує у відповідь категорію полярності. Такий підхід також дозволяє поступово вбудовувати модель у бізнес-процеси компанії: від внутрішньої аналітики до безпосередньої взаємодії з клієнтом через інтерфейс.

Особливу увагу слід приділити аспектам продуктивності, надійності та обробки помилок. Зокрема, у випадку тимчасової недоступності сервісу класифікації, необхідно реалізувати механізми кешування або повторної обробки, щоб уникнути втрати даних або зниження якості обслуговування

користувачів.

Таким чином, інтеграція моделі емоційного аналізу в існуючі системи e-commerce відкриває широкі можливості для покращення клієнтського досвіду, автоматизації обробки великої кількості текстових даних та підвищення ефективності прийняття управлінських рішень на основі якісного аналізу користувацького контенту.

4.3 Економічне обгрунтування використання розробленої моделі

Описана модель класифікації текстів за емоційним забарвленням із самого початку проєктувалася як ефективне рішення з мінімальними вимогами до обчислювальних ресурсів. Основою для її реалізації стала дистильована версія BERT - TinyBERT, яка поєднує достатню точність із компактною архітектурою. Завдяки зменшеній кількості параметрів та низькому навантаженню на систему модель може ефективно функціонувати на пристроях із обмеженими ресурсами - зокрема, на стандартних серверах без графічних прискорювачів або навіть на системах із базовими процесорами. Це знижує поріг входу для впровадження та відкриває можливість її використання в малих і середніх організаціях.

У межах економічного аналізу було розглянуто два основні сценарії розгортання моделі: на власному фізичному сервері компанії та у хмарному середовищі. Локальне розгортання передбачає одноразові витрати на апаратне забезпечення з подальшим повним контролем над середовищем виконання. Для такої моделі достатньо бюджетного сервера з багатоядерним CPU, що значно знижує вартість початкової інтеграції. Водночас експлуатація власної інфраструктури передбачає витрати на електроенергію, технічну підтримку та обслуговування, які формують щомісячні витрати.

Альтернативно, хмарне розгортання передбачає оплату за фактичне використання ресурсів без початкових капіталовкладень. Модель TinyBERT може бути легко інтегрована в сервіси, як-от AWS Lambda, Google Cloud Run або Vertex AI, завдяки своїй сумісності з контейнерними рішеннями й

підтримці безсерверних обчислень. Такий підхід забезпечує гнучкість масштабування, швидкий запуск і мінімізацію витрат на обслуговування. Вартість хмарних рішень варіюється залежно від обсягу викликів та використаних обчислювальних ресурсів, проте для проєктів із нерівномірним або помірним навантаженням цей варіант часто є економічно обґрунтованішим.

Для об’єктивного порівняння варіантів розгортання було здійснено розрахунок витрат на основі актуальних цін станом на травень 2025 року. Усі суми наведено в доларах США з урахуванням середньої вартості електроенергії в Україні ($0,14/кВт·год) і типових тарифів на популярних хмарних платформах. У таблиці 4 представлено порівняльний аналіз початкових та щомісячних витрат для кожного з варіантів.

Таблиця 4. Порівняльна таблиця витрат на різні серверні рішення

Варіант хостингу	Початкові витрати (USD)	Щомісячні витрати (USD)	Деталі
Локальний сервер (CPU)	$264,46	$20,30	AMD Ryzen 5 3400G (4 ядра, 8 потоків), 8 ГБ RAM
Локальний сервер (з GPU)	$428,22	$24,09	AMD Ryzen 5 3400G (4 ядра, 8 потоків), 8 ГБ RAM, GTX 1650 (4 ГБ VRAM)
GCP (Cloud Run + API)	$0	$58,50	e2-standard-2 (2 vCPU, 8 ГБ RAM)
AWS (EC2 +	$0	$57,00	t3.large (2 vCPU, 8

Lambda API GW)			ГБ RAM) + Lambda
Google Vertex AI API	$0	від $38,53	Дозволяє вибрати тип машини (CPU, RAM), але інфраструктуру і масштабування сервіс автоматично керує без участі користувача

Розглянуті варіанти розгортання моделі демонструють компроміс між початковими інвестиціями та щомісячними витратами. Локальний сервер вимагає одноразового капіталовкладення, але має низькі поточні витрати, тоді як хмарні сервіси забезпечують гнучкість і масштабованість без початкових витрат, але з вищими регулярними платежами. Вибір оптимального варіанту залежить від специфіки використання, технічних можливостей і фінансових пріоритетів компанії.

Висновки

В рамках розділу про впровадження моделі в існуючі проєкти в сфері e-commerce було успішно розроблено API на основі Flask, що забезпечує зручний та ефективний інтерфейс для взаємодії з моделлю класифікації емоційного забарвлення текстів. Створені скрипти для запуску додатку в Docker контейнері дозволили забезпечити портативність і простоту розгортання в різних середовищах. Для демонстрації практичної реалізації було розгорнуто сервер як локально, так і в хмарному сервісі Google Vertex AI Endpoint, що підтвердило гнучкість і масштабованість рішення. В процесі роботи також було розглянуто

можливості інтеграції розробленої системи в існуючі e-commerce платформи, що підкреслює її практичну цінність для автоматизованого аналізу клієнтських відгуків. Крім того, детально проаналізовано варіанти розгортання моделі з економічної точки зору, що дає змогу обрати найбільш оптимальний баланс між початковими інвестиціями та експлуатаційними витратами відповідно до потреб конкретної організації. Таким чином, виконана робота створює міцну базу для подальшого впровадження і масштабування системи в реальних бізнес-сценаріях.

ЗАГАЛЬНІ ВИСНОВКИ

У межах виконаної роботи було послідовно та всебічно розглянуто проблематику автоматичної класифікації емоційного забарвлення текстових відгуків у сфері e-commerce, що має важливе значення для підвищення якості обслуговування клієнтів і оптимізації маркетингових стратегій.

Перший розділ присвячено аналізу існуючих підходів до обробки текстових даних, де було визначено переваги глибоких нейронних мереж, зокрема трансформерів, над класичними методами. Обґрунтовано вибір моделі TinyBERT як збалансованого рішення, що поєднує високу точність із помірними вимогами до обчислювальних ресурсів.

Другий розділ поглиблено висвітлює теоретичні засади створення моделі, включаючи принципи роботи нейронних мереж, особливості архітектури трансформерів і ключові метрики оцінки якості моделі. Це забезпечує міцний фундамент для розуміння принципів функціонування і критеріїв ефективності розробленої системи.

У третьому розділі проведено детальний аналіз даних, які були використані для навчання і тестування моделі, а також описано застосовані програмні інструменти і методологію розробки. Отримані результати підтверджують високу ефективність моделі на великому масиві реальних даних, що свідчить про її здатність до узагальнення та практичну придатність для задач автоматичного аналізу текстів.

Четвертий розділ демонструє практичну реалізацію та впровадження моделі в реальні бізнес-процеси. Розроблене API на Flask та контейнеризація додатку через Docker забезпечують зручність інтеграції і гнучкість розгортання як локально, так і в хмарному середовищі Google Vertex AI. Проведений аналіз варіантів розгортання з урахуванням економічних аспектів дає змогу оптимально підібрати модель експлуатації відповідно до ресурсів і потреб організації.

Отже, виконана робота поєднує теоретичні дослідження, практичну розробку та економічний аналіз, створюючи комплексне рішення для

автоматизованої класифікації полярності текстів, яке може ефективно застосовуватися в реальних e-commerce проєктах, підвищуючи їх конкурентоспроможність та якість взаємодії з користувачами.

СПИСОК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. GeeksforGeeks. Natural Language Processing (NLP) - Overview: веб-сайт. URL: https://www.geeksforgeeks.org/natural-language-processing-overview
2. Josh Howarth. 79+ Brand New Ecommerce Statistics for 2025: веб-сайт. URL: https://explodingtopics.com/blog/ecommerce-stats
3. 51 eCommerce Statistics In 2025 (Global and U.S. Data): веб-сайт. URL: https://www.sellerscommerce.com/blog/ecommerce-statistics
4. Karlee Reistroffer. Online Review Statistics for 2025 to Know: веб-сайт. URL: https://www.textedly.com/blog/online-review-statistics-for-2025-to-know
5. Sheikh Muhammad Abdullah. Kaggle. Text Preprocessing | NLP | Steps to Process Text: веб-сайт. URL: https://www.kaggle.com/code/abdmental01/text-preprocessing-nlp-steps-to-process-text
6. GeeksforGeeks. Naive Bayes Classifiers: веб-сайт. URL: https://www.geeksforgeeks.org/naive-bayes-classifiers
7. Biau, Gérard & Scornet, Erwan. (2015). A Random Forest Guided Tour. TEST. 25. DOI: 10.1007/s11749-016-0481-7
8. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Learning, 2021.
9. Xinya Liao. Sentiment analysis based on machine learning models. Applied and Computational Engineering, 54, 129-134, 2024.
10. Heaton, J. Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning. Genet Program Evolvable Mach 19, 305-307 (2018). DOI: 10.1007/s10710-017-9314-z
11. Jose Camacho-Collados, Mohammad Taher Pilehvar. On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis, 2018.
12. Zhang, Ye & Wallace, Byron. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.
13. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones,

Aidan N Gomez, Łukasz Kaiser, Illia Polosukhin. Attention Is All You Need. NeurIPS, 2023.

14. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2019.
15. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach, 2019.
16. Pengcheng He, Xiaodong Liu, Jianfeng Gao, Weizhu Chen. DeBERTa: Decoding-enhanced BERT with Disentangled Attention, 2021
17. Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, Qun Liu. TinyBERT: Distilling BERT for Natural Language Understanding, 2020
18. Microsoft Turing Universal Language Representation model, T-ULRv6, tops both XTREME and GLUE leaderboards with a single model. URL: https://blogs.bing.com/search-quality-insights/october-2022/Microsoft-Turing-Universal-Language-Representation-model,-T-ULRv6,-tops-both-XTREME-and-GLUE-leaderb
19. GLUE benchmark leaderboard: веб-сайт. URL: https://gluebenchmark.com/leaderboard
20. Colin Shearer. The CRISP-DM Model: The New Blueprint for Data Mining. Journal of Data Warehousing, 5, 13-22, 2000.
21. Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. 1996. From Data Mining to Knowledge Discovery in Databases. AI Mag. 17, 3 (September 1996), 37-54. DOI: 10.1609/aimag.v17i3.1230
22. Exploring SAS Enterprise Miner: Special Collection. Cary, NC: SAS Institute Inc. 2018 URL: https://support.sas.com/content/dam/SAS/support/en/books/free-books/exploring-sas-enterprise-miner-special-collection.pdf
23. Lantz, B. Machine Learning with R : textbook. Berhad : Packt Publishing, 2013. 400 p.
24. Haykin S. Neural Networks: A Comprehensive Foundation. Upper Saddle River,

NJ: Prentice Hall, 1994. 842 p.

25. Towards Data Science. Niklas Lang. Activation Functions in Neural Networks: How to Choose the Right One: веб-сайт. URL: https://towardsdatascience.com/activation-functions-in-neural-networks-how-to-choose-the-right-one-cb20414c04e5
26. Evelyn Fix, J.L. Hodges Jr. Discriminatory analysis: nonparametric discrimination, consistency properties. Randolph Field, Tex. : USAF School of Aviation Medicine, 1951. 21 p. URL: https://catalog.hathitrust.org/Record/100923824
27. C. J. van Rijsbergen. Information Retrieval, 2nd ed. London: Butterworths, 1979. 208 p. URL: https://www.dcs.gla.ac.uk/Keith/Preface.html
28. Swets J. A. Signal Detection Theory and ROC Analysis in Psychology and Diagnostics: Collected Papers. Mahwah, NJ : Lawrence Erlbaum Associates, 1996. 518 p. URL: https://www.worldcat.org/title/34953188
29. Kaggle. Amazon reviews: веб-сайт. URL: https://www.kaggle.com/datasets/kritanjalijain/amazon-reviews
30. Daily.dev. Nimrod Kramer. Jupyter for Beginners: веб-сайт. URL: https://daily.dev/blog/jupyter-for-beginners#:~:text=Jupyter%20Notebook%20is%20a%20versatile,text%20in%20a%20single%20documen

ДОДАТКИ

ДОДАТОК А

Фрагмент коду для очищення даних

```
import pandas as pd
import re

df_train = pd.read_csv("train.csv")

def df_preprocess(df: pd.DataFrame):
    processed_df = pd.DataFrame()
    processed_text = df['title'].fillna('') + ' ' + df['text'].fillna('')
    processed_text = processed_text.apply(clean_text)
    processed_df['text'] = processed_text
    processed_df['label'] = df['polarity'].map({1: 0, 2: 1})
    return processed_df

def clean_text(text):
    text = re.sub(r"<.*?>", "", text)
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)
    return text

df_preprocess(df_train).to_csv('amazon_comments_train_full.csv', index=False)
```

ДОДАТОК Б

Фрагмент коду для пошуку гіперпараметрів

```
import pandas as pd
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    Trainer,
    TrainingArguments,
    DataCollatorWithPadding,
    EvalPrediction
)
from datasets import Dataset, DatasetDict
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split

OUTPUT_DIR = "./results_hp_search"

df = pd.read_csv("amazon_comments_train.csv")

train_texts, val_texts = train_test_split(df, test_size=0.1, stratify=df['label'],
random_state=42)

dataset = DatasetDict({
    "train": Dataset.from_pandas(train_texts.reset_index(drop=True)),
    "validation": Dataset.from_pandas(val_texts.reset_index(drop=True))
})

model_name = "huawei-noah/TinyBERT_General_4L_312D"
tokenizer = AutoTokenizer.from_pretrained(model_name)

def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True)

tokenized_datasets = dataset.map(tokenize_function, batched=True)

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

def compute_metrics(p: EvalPrediction):
    preds = p.predictions[0] if isinstance(p.predictions, tuple) else p.predictions
    preds = preds.argmax(axis=1)
    labels = p.label_ids
    return {
        "accuracy": accuracy_score(labels, preds),
        "precision": precision_score(labels, preds),
        "recall": recall_score(labels, preds),
```

```
        "f1": f1_score(labels, preds),
    }

def model_init():
    return AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)

training_args = TrainingArguments(
    output_dir=OUTPUT_DIR,
    eval_strategy="epoch",
    save_strategy="no",
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    load_best_model_at_end=False,
    metric_for_best_model="f1",
    report_to="none",
    logging_steps=50,
)

trainer = Trainer(
    model_init=model_init,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    processing_class=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

def optuna_hp_space(trial):
    return {
        "learning_rate": trial.suggest_float("learning_rate", 1e-5, 5e-4, log=True),
        "num_train_epochs": trial.suggest_int("num_train_epochs", 3, 10),
        "weight_decay": trial.suggest_float("weight_decay", 0.0, 0.1),
        "warmup_ratio": trial.suggest_float("warmup_ratio", 0.0, 0.2),
        "max_grad_norm": trial.suggest_float("max_grad_norm", 0.5, 2.0),
    }

best_run = trainer.hyperparameter_search(
    direction="maximize",
    backend="optuna",
    n_trials=10,
    hp_space=optuna_hp_space,
    compute_objective=lambda metrics: metrics["eval_f1"]
)
```

ДОДАТОК В

Фрагмент коду для навчання моделі

```
import pandas as pd
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    Trainer,
    TrainingArguments,
    DataCollatorWithPadding,
    EvalPrediction
)
from datasets import Dataset, DatasetDict
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split

df = pd.read_csv("amazon_comments_train_full.csv")

train_texts, val_texts = train_test_split(df, test_size=0.1, stratify=df['label'],
random_state=42)

dataset = DatasetDict({
    "train": Dataset.from_pandas(train_texts.reset_index(drop=True)),
    "validation": Dataset.from_pandas(val_texts.reset_index(drop=True))
})

model_name = "huawei-noah/TinyBERT_General_4L_312D"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)

def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True)

tokenized_datasets = dataset.map(tokenize_function, batched=True)

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

def compute_metrics(p: EvalPrediction):
    preds = p.predictions.argmax(axis=-1)
    labels = p.label_ids
    return {
        "accuracy": accuracy_score(labels, preds),
        "precision": precision_score(labels, preds),
        "recall": recall_score(labels, preds),
        "f1": f1_score(labels, preds),
    }
```

```
training_args = TrainingArguments(
    output_dir="./results_tinybert",
    eval_strategy="epoch",
    save_strategy="epoch",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=5.5e-05,
    num_train_epochs=5,
    weight_decay=0.03,
    warmup_ratio=0.148,
    logging_dir='./logs',
    logging_steps=50,
    save_total_limit=1,
    load_best_model_at_end=True,
    metric_for_best_model="f1",
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    processing_class=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

trainer.train()

trainer.save_model("./tinybert-emotion-model")

eval_results = trainer.evaluate()
```

ДОДАТОК Г

Фрагмент коду API для роботи з моделлю

```
import torch
import os
import sys
import re
from transformers import AutoTokenizer, AutoModelForSequenceClassification,
TextClassificationPipeline
from flask import Flask, request, jsonify

SAVED_MODEL_PATH = "./tinybert-emotion-model"

tokenizer = None
model = None
pipeline = None
device = None

def load_model():
    global tokenizer, model, pipeline, device

    print("Starting model and tokenizer loading...", file=sys.stderr)

    if not os.path.isdir(SAVED_MODEL_PATH):
        error_msg = f"Error: Saved model path '{SAVED_MODEL_PATH}' not found. Ensure
the '{SAVED_MODEL_PATH}' directory with model files is present."
        print(error_msg, file=sys.stderr)
        sys.exit(f"FATAL: {error_msg}")


    try:
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        print(f"Using device: {device}", file=sys.stderr)

        tokenizer = AutoTokenizer.from_pretrained(SAVED_MODEL_PATH)
        print("Tokenizer successfully loaded.", file=sys.stderr)

        if not os.path.exists(os.path.join(SAVED_MODEL_PATH, 'config.json')):
             error_msg = f"Error: '{SAVED_MODEL_PATH}' does not appear to be a valid
Hugging Face model directory (config.json missing)."
             print(error_msg, file=sys.stderr)
             sys.exit(f"FATAL: {error_msg}")


        model = AutoModelForSequenceClassification.from_pretrained(SAVED_MODEL_PATH)
        model.to(device)
```

```
        model.eval()
        print("Model successfully loaded and set to evaluation mode.", file=sys.stderr)

        pipeline_device = 0 if torch.cuda.is_available() else -1
        pipeline = TextClassificationPipeline(model=model, tokenizer=tokenizer,
device=pipeline_device)
        print("Text classification pipeline successfully created.", file=sys.stderr)

    except Exception as e:
        error_msg = f"Error loading model or creating pipeline: {e}"
        print(error_msg, file=sys.stderr)
        sys.exit(f"FATAL: {error_msg}")

load_model()
print("Model loading process completed.", file=sys.stderr)

app = Flask(__name__)

def extract_texts_from_request_data(data):
    texts_to_predict = []

    if 'instances' in data:
        if isinstance(data['instances'], list):
            texts_to_predict = [t for t in data['instances'] if (isinstance(t, str) and
bool(t.strip()))]

            if data['instances'] and not texts_to_predict:
                 raise ValueError("'instances' list contains elements that are not non-
empty strings.")
        else:
            raise ValueError("'instances' should be a list.")
    else:
         raise ValueError("Missing field 'instances'. Expected a list of non-empty
strings.")

    return list(map(clean_text, texts_to_predict))


def perform_prediction(texts_list):
    if pipeline is None:
        error_msg = "Model and pipeline are not loaded. Server initialized
incorrectly."
        raise RuntimeError(error_msg)

    if not texts_list:
        return []
```

```
    try:
        results = pipeline(texts_list)
        return results

    except Exception as e:
         raise RuntimeError(f"Prediction failed: {e}")


def format_prediction_results(raw_results, original_texts):
    formatted_results = []
    for i, result in enumerate(raw_results):
        predicted_label_raw = result.get('label', 'N/A')
        score = result.get('score', 0.0)

        predicted_label = predicted_label_raw
        try:
            if isinstance(predicted_label_raw, str) and
predicted_label_raw.startswith('LABEL_'):
                label_index = int(predicted_label_raw.split('_')[-1])
                predicted_label = label_index
        except (ValueError, IndexError, AttributeError):
             pass

        original_text = original_texts[i] if i < len(original_texts) else "N/A (текст
не знайдено)"


        formatted_results.append({
            "text": original_text,
            "predicted_label": predicted_label,
            "confidence_score": float(score)
        })

    return formatted_results

def clean_text(text):
    text = re.sub(r"<.*?>", "", text)
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)
    return text

@app.route('/predict', methods=['POST'])
def predict():
    if not request.is_json:
        print("Received non-JSON request.", file=sys.stderr)
        return jsonify({"error": "Unsupported data type. Use JSON (Content-Type:
```

```
application/json)"}), 415

    data = request.get_json()

    try:
        texts_to_predict = extract_texts_from_request_data(data)
    except ValueError as e:
        return jsonify({"error": f"Data validation error: {e}"}), 400

    if not texts_to_predict:
         return jsonify({"inference_time_seconds": 0.0, "results": []}), 200

    try:
        raw_prediction_results = perform_prediction(texts_to_predict)

    except RuntimeError as e:
        return jsonify({"error": "Prediction internal error", "details": str(e)}), 500
    except Exception as e:
        return jsonify({"error": "Unexpected internal error during prediction",
"details": str(e)}), 500

    try:
        formatted_results_list = format_prediction_results(raw_prediction_results,
texts_to_predict)
    except Exception as e:
        return jsonify({"error": "Internal server error during result formatting",
"details": str(e)}), 500

    final_response = {
        "predictions": formatted_results_list
    }

    return jsonify(final_response), 200

@app.route("/health", methods=["GET"])
def health():
    return jsonify({"status": "ok"}), 200

if __name__ == '__main__':
    print("Running Flask development server (for local testing)...", file=sys.stderr)
    app.run(debug=True, host='0.0.0.0', port=8080)
    print("Flask development server stopped.", file=sys.stderr)
```