Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук

(повна назва)

Кафедра програмної інженерії

(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження моделей обробки природної мови для рекомендаційних систем на основі тексту

(тема)

Виконала:

Студентка 2 курсу, групи ІПЗм-22-2

Середа Д.А.

(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення

(код і повна назва спеціальності)

Тип програми освітньо-наукова

Освітня програма Інженерія програмного забезпечення

(повна назва освітньої програми)

Керівник доц. Голян Н.В.

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ З.В. Дудар

(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми освітньо-наукова

Освітня програма Інженерія програмного забезпечення

(повна назва)

ЗАТВЕРДЖУЮ:
Зав.кафедри _____
(підпис)

«__»_____2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентці Середі Дар'ї Антонівні

(прізвище, ім'я, по батькові)

1. Тема роботи: «Дослідження моделей обробки природної мови для рекомендаційних систем на основі тексту»

Затверджена наказом по університету від: 29 березня 2024р. №250 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18 червня 2024 р.

2. Вихідні дані до проекту: *відкриті дані та інформація про рекомендаційні системи та моделі NLP, технології для проведення теоретичних та експериментальних досліджень, мова програмування Python, бібліотеки моделей NLP: transformers, fasttext та genism, фреймворк Django та React.js.*

3. Перелік питань, що потрібно опрацювати в роботі: *вступ, аналіз предметної області, обгрунтування проблеми та актуалізація рішень, постановка задачі, теоретичні та експериментальні дослідження існуючих методів NLP, проведення оцінки моделей NLP, порівняння моделей та аналіз результатів.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Відмітки про виконання
1	Аналіз предметної галузі	01.04.2024 – 02.04.2024	Виконано
2	Аналіз літературних джерел	03.04.2024 – 08.04.2024	Виконано
3	Аналіз існуючих моделей обробки природної мови	09.04.2024 – 12.04.2024	Виконано
4	Проведення теоретичного дослідження моделей	12.04.2024 – 13.04.2024	Виконано
5	Планування експериментальних досліджень	13.04.2024 – 14.04.2024	Виконано
6	Програмна реалізація та навчання NLP методів для проведення досліджень	14.04.2024 – 20.04.2024	Виконано
7	Проведення експериментів	20.04.2024 – 26.05.2024	Виконано
8	Оцінка моделей та аналіз результатів	26.05.2024 – 27.05.2024	Виконано
9	Написання статті за темою магістерської роботи	27.02.2024 – 29.04.2024	Виконано
10	Опис можливості використання отриманих результатів та розробка рекомендаційної системи	30.04.2024 – 15.05.2024	Виконано
11	Підготовка пояснювальної записки	16.05.2024 – 29.05.2024	Виконано
12	Перевірка пояснювальної записки на унікальність тексту	05.06.2024	Виконано
13	Перевірка на нормоконтроль	06.06.2024	Виконано
14	Рецензування	11.06.2024	Виконано
15	Електронна архівація	15.06.2024	Виконано
16	Попередній захист	15.06.2024	Виконано
17	Допуск до захисту	16.06.2024	Виконано

Дата видачі завдання «29» березня 2024 р.

Студентка _____ Середа. Д. А.
(підпис)

Керівник роботи _____ доц. каф. ПІ. Голян Н.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 105 с., 15 табл., 25 рис., 6 додатків, 21 джерел.

РЕКОМЕНДАЦІЇ, СИСТЕМИ, NLP, ОБРОБКА, ПРИРОДНА, МОВА, ТОЧНІСТЬ, ТЕКСТ, КОНТЕКСТ.

Об'єктом дослідження є моделі NLP для обробки природної мови.

Предметом дослідження є процес застосування моделей обробки природної мови у рекомендаційних системах на основі текстових даних.

Метою роботи є оцінка ефективності моделей NLP для рекомендаційних систем на основі тексту, проведення експериментів для аналізу, порівняння та оцінки ефективності моделей природної мови за певними метриками, навчання та оцінка моделей NLP, адаптованих до вимог рекомендаційних систем з урахуванням настрою та особливостей текстових даних на основі проведених експериментів та розробка рекомендаційної системи на основі отриманих результатів.

Методами рішення є проведення теоретичних та експериментальних досліджень та визначення корисності моделей на основі шкал та критеріїв.

У результаті роботи повинен бути зроблений висновок щодо корисності розглянутих моделей NLP, доцільності їх використання та точності при певних задачах. На основі цього висновку повинна бути розроблена рекомендаційна система з використанням отриманих результатів досліджень, а саме отриманих найефективніших методів NLP, для надання рекомендацій з урахуванням емоційної складової та контексту текстової інформації.

RECOMMENDATIONS, SYSTEMS, NLP, PROCESSING, NATURAL, LANGUAGE, ACCURACY, TEXT, CONTEXT.

The object of research is NLP models for natural language processing.

The subject of research is the process of applying natural language processing models in recommender systems based on text data.

The purpose of the work is to evaluate the effectiveness of NLP models for text-based recommender systems, conduct experiments to analyze, compare and evaluate the effectiveness of natural language models according to certain metrics, train and evaluate NLP models adapted to the requirements of recommender systems taking into account the mood and features of text data based on conducted experiments and development of a recommendation system based on the obtained results.

The decision methods are conducting theoretical and experimental studies and identifying usefulness based on scales and criteria.

As a result of the work, a conclusion should be drawn regarding the usefulness of the considered NLP models, the expediency of their use and accuracy in certain tasks. Based on this conclusion, a recommender system should be developed using the obtained research results, namely the obtained most effective NLP methods, to provide recommendations taking into account the emotional component and the context of textual information.

Я, Середа Д.А., студентка гр.ІПЗм-22-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: представлена моя робота, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений (а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 10
1 Аналіз предметної області та постановка задачі 11
1.1 Аналіз предметної області 11
1.2 Аналіз наукових джерел 14
1.3 Визначення проблем та актуалізація рішень 15
1.4 Постановка задачі 17
2 Аналіз існуючих моделей обробки природної мови 19
2.1 Моделі векторного представлення 19
2.2 Рекурентні нейронні мережі 27
2.3 Трансформери 33
3 Проведення теоретичного дослідження 41
3.1 Виділення альтернатив та критеріїв порівняння моделей 41
3.2 Опис шкал оцінок за критеріями 42
3.3 Нормування критеріїв 45
3.4 Розрахунок коефіцієнтів корисності та аналіз результатів 47
4 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ 49
4.1 Метрики, методи та матеріали дослідження 49
4.2 Налаштування програмного середовища 54
4.3 Проведення експериментів 62
5 Оцінка та аналіз результатів 66
6 Використання результатів дослідження 69
6.1 Вимоги до програмної системи 70
6.2 Опис технологій для серверної частини 71
6.3 Розробка клієнтської частини 74
Висновки 80
Перелік джерел посилання 80
Додаток А Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії ..... **Ошибка! Закладка не определена.**

Додаток Б Звіт з результатами перевірки на унікальність тексту в базі ХНУРЕ ..... **Ошибка! Закладка не определена.**
Додаток В Слайди презентації ..... **Ошибка! Закладка не определена.**
Додаток Г Апробація результатів роботи..... **Ошибка! Закладка не определена.**
Додаток Д Повідомлення про прийняття статті до публікації у конференції CoLiNs 2024 в якості CEUR-WS Style (NON-SCOPUS)..... 100
Додаток Е Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015..... 105

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

NLP (Natural Language Processing), НЛП – обробка природної мови.

РС – рекомендаційні системи.

TF-IDF – Term Frequency-Inverse Document Frequency, технологія, що використовується в рекомендаційних системах на основі тексту.

Токен – найменша одиниця структури тексту, можуть бути представлені словами, числами, пунктуацією, тощо.

Токенізація – процес розбиття тексту на токени.

Векторне представлення слова – представлення слова у вигляді вектора числових значень.

Векторизація – процес перетворення текстових даних в числові вектори.

Ембединг – спосіб представлення слів або інших об'єктів у векторному просторі чисел.

w – слово.

v_w – вектор слова w.

C_w – контекст слова w.

u_t – вектор слова t, яке є членом контексту C_w.

P(t|w) – ймовірність того, що слово t з'явиться в контексті слова w.

P(w|t) – ймовірність того, що слово w з'явиться в контексті слова t.

V – множина слів, словник.

W – матриці вагових коефіцієнтів, що зв'язують вхідний вектор з вхідними шлюзами, шлюзом забування і вихідним шлюзом відповідно,

b_i, b_f, b_o – вектори зсувів, що додаються до виразів для розрахунку шлюзів,

X_t - це вхідний вектор, що містить дані для поточного кроку обробки,

h_{t-1} – прихований стан, вектор, що містить стан мережі після попереднього кроку обробки,

I_t – вхідний шлюз, вектор, що визначає, скільки інформації з вхідного вектору буде передано до прихованого стану,

F_t – шлюз забування, вектор, що визначає, скільки інформації з попереднього стану буде забуто,

O_t - вихідний шлюз, вектор, що визначає, скільки інформації з прихованого стану буде передано на вихід.

Q, K, V – матриці запитів, ключів та значень після лінійного перетворення,

A – матриця скалярних добутків та зважених значень,

d_k – розмірність ключа вихідних матрць Q, K.

RNN – рекурентні нейронні мережі.

LSTM (Long Short-Term Memory) – вид RNN, який спроектований для роботи з послідовностями даних і вирішення проблеми зниклого градієнту, що виникає у звичайних RNN.

GRU (Gated Recurrent Unit) – інший вид RNN, яка, подібно до LSTM, призначена для роботи з послідовностями даних і вирішення проблеми зниклого градієнту.

FNN (Feedforward Neural Network) – пряма нейронна мережа.

Precision – точність.

Recall – повнота.

F-score, F-міра – гармонічне середнє між точністю і повнотою.

MSE (Mean Squared Error) – середньоквадратична помилка.

MAE (Mean Absolute Error) – середньоабсолютна помилка.

ВСТУП

В наші часи, рекомендаційні системи є однією з важливих технології у мережі Інтернет. Саме рекомендаційні систем дозволяють веб-платформам та сервісам полегшити та персоналізувати досвід використання користувачів, надаючи рекомендації за їх індивідуальними вподобаннями. Рекомендаційні системи знайшли своє застосування у багатьох сферах, таких як: електронна комерція та онлайн магазини, освітні платформи, медицина, подорожі, медіа та розваги, соціальні мережі і т.д. Одним з найпоширеніших видів рекомендацій є рекомендації на основі текстової інформації. Це можуть бути певні описи товарів, відгуки, коментарі, пости соціальних мереж та інше.

Для аналізу та обробки цієї текстової інформації та надання подальших рекомендацій на її основі застосовуються моделі обробки природної мови. Саме моделі природної мови дозволяють рекомендаційним системам на основі тексту надавати влучні рекомендації.

Актуальність даної теми полягає в тому, що за останні роки в галузі комп’ютерних наук відбувся стрімкий розвиток обробки природної мови, що відкриває нові можливості для надання персоналізованих рекомендацій. Нові моделі NLP можуть краще розуміти складний текст та контекст, в якому він застосовується, що дозволяє знаходити більш точні та схожі за темою об'єкти.

Метою роботи є оцінка ефективності моделей NLP для рекомендаційних систем на основі тексту, проведення експериментів для аналізу, порівняння та оцінки ефективності моделей природної мови за певними метриками, навчання та оцінка моделей NLP, адаптованих до вимог рекомендаційних систем з урахуванням настрою та особливостей текстових даних на основі проведених експериментів та розробка рекомендаційної системи на основі отриманих результатів.

.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Рекомендаційні системи – це системи програмного забезпечення, що призначені для прогнозування та надання певних продуктів/об’єктів користувачам, в залежності від їх попередніх уподобань, взаємодій з продуктами, тощо.

Механізми рекомендацій забезпечують персоналізовану взаємодію системи з користувачем, допомагаючи кожному окремому споживачеві знайти улюблені фільми, телешоу, цифрові продукти, книги, статті, послуги на основі повередньо оброблених даних.

Рекомендаційні системи широко застосовуються в різних сферах діяльності, таких як:

- електронна коммерція. РС домопагають користувачам знайти товари, які могли б їм сподобатись на основі попередніх переглядів та здійснених покупок;
- соціальні мережі, медіа та розваги. Усі найпопулярніши стримінгові платформи та соціальні мережі, такі як YouTube, Netflix, Spotify, Tiktok, Instagram, Twitter використовують рекомендаційні системи для пропозицій контенту, який би міг зацікавити користувача на основі його історії переглядів, коментарів, вподобань та взаємодії з контентом. Окрім цього, РС використовуються для надання рекомендацій нових друзів за спільними інтересами;
- освітні платформи. У сфері освіти рекомендаційні системи також допомагають користувачам отримувати навчальні матеріали, курси з урахуванням рівня знань, інтересів та вже пройдених раніше матеріалів;
- медицина. У медичній галузі рекомендаційні системи можуть надавати персоналізовані поради щодо схожих ліків, надавати медичні рекомендації та поради, дієти та спосіб харчування, спортивні вправи;

– подорожі та готельний бізнес: рекомендаційні системи можуть допомагати користувачам знаходити оптимальні варіанти подорожей, готелів та ресторанів на основі їхніх вподобань та відгуків;
– сфера фінансів. У даній області РС можуть пропонувати фінансові послуги, платіжні опції, поради щодо інвестицій, тощо.

Існує два основних типи рекомендаційних систем – це фільтраційна за вмістом та колаборативна фільтрація. У фільтраційних за вмістом системах рекомендацій об’єкти визначаються на основі їх ознак. На основі наявних атрибутів в об’єктах, які користувач оцінив вивчається профіль інтересів нового користувача, тобто можна сказати, що це система рекомендацій на основі ключових слів, що описують ці об’єкти. Таким чином, у системі рекомендацій на основі вмісту використовуються алгоритми, які пропонують користувачам схожі елементи, які йому сподобалися в минулому або які вони вивчають зараз. Колаборативні системи об’єднують оцінки або рекомендації об’єктів, розпізнають спільні риси між користувачами на основі їхніх оцінок і генерують нові рекомендації на основі порівнянь між користувачами [1].

Рекомендаційні системи на основі тексту для надання рекомендацій використовують текстову інформацію, а саме це можуть бути описи та назви об’єктів, відгуки та коментарі, пости соц. мереж, тощо.

Обробка природної мови (NLP) – це технологія машинного навчання, яка надає комп'ютерам можливість інтерпретувати, маніпулювати та розуміти людську мову. NLP має широкий спектр застосувань у таких задачах, як машинний переклад, розпізнавання мови, створення контенту та рекомендаційні системи.

У контексті рекомендаційних систем на основі тексту, NLP може використовуватися для вирішення наступних завдань:

– для створення опису об'єктів, що рекомендуються. Ці описи можуть бути створені на основі заголовків, назв, коротких змістів, резюме, відгуків інших користувачів та інших текстових даних. Наприклад, у рекомендаційній системі для книг NLP може використовуватися для

створення описів книг на основі їхніх заголовків, резюме, відгуків інших користувачів та інших факторів;

– для пошуку подібності між об'єктами, що дозволяє знаходити об'єкти, які можуть бути цікаві користувачам на основі їхніх вподобань. Наприклад, у рекомендаційній системі для книг NLP може використовуватися для пошуку подібності між книгами на основі їхніх заголовків, резюме, жанрів та інших факторів;
– розуміння вподобань користувачів, що дозволяє надавати більш релевантні рекомендації. Наприклад, у рекомендаційній системі для книг NLP може використовуватися для розуміння інтересів користувачів на основі їхніх оцінок книг, відгуків та інших даних.

Моделі NLP для рекомендаційних систем можна класифікувати за типом задачі:

– класифікація, коли дані класифікуються до певних категорій.
– сегментація, коли текстові дані розбиваються на менші сегменти для подальшого аналізу;
– подібність тексту;
– емоційний аналіз, коли виявляється емоційна складова тексту.

Окрім колабаративної фільтрації та фільтрацй за змстом існує ще декілька традиційних підходів в рекомендаційних системах на основі тексту: TF-IDF, що вимірює важливість слова в документі відносно до колекції документів. TF-IDF перетворює тексти у числові вектори, які використовуються для обчислення схожості між документами або запитами, надаючи релевантні рекомендації. Матричний факторинг, що розкладає матрицю користувачів і елементів на дві менші, які використовуються для передбачення уподобань користувачів і надання рекомендацій.

Класичні підходи також включають наївний Байєсів класифікатор, логістичну регресію і машина опорних векторів (SVM).

1.2 Аналіз наукових джерел

Використання моделей обробки природної мови для текстових систем рекомендацій є відносно новою областю досліджень. Проте в останні роки зростає інтерес до цієї сфери через збільшення доступності текстових даних і розробки нових технік NLP. Рекомендаційні системи стали об’єктом дослідження багатьох наукових публікацій, проводились дослідження окремих типів рекомендаційних систем, їх структур, переваг і недоліків, проводились дослідження та розробка алгоритмів рекомендаційних систем на основі глибокого навчання. Глибоке навчання дозволяє моделям NLP ефективно вивчати мову на великих обсягах даних, покращуючи їх здатність розуміти текст та вилучати інформацію, однією з таких архітектур є рекурентні нейронні мережі, що враховують послідовний контекст тексту. Однією з таких архітектур є рекурентні нейронні мережі, які враховують послідовний контекст тексту.

Рат і Саху досліджують застосування рекурентних нейронних мереж (RNN) у системах рекомендацій. Їх дослідження заглиблюється в ефективність RNN у захопленні послідовних моделей у даних про поведінку користувачів для надання персоналізованих рекомендацій [2]. Вони обговорюють архітектуру та процес навчання RNN і дають зрозуміти, як ці моделі можуть підвищити ефективність рекомендацій.

Ще одним видом архітектур NLP в рекомендаційних системах є моделі трансформери, що використовують механізм уваги для визначення глобальних залежностей між інформацією на вході та на виході та уникає повторень. Ворнарс, Совіто та Гаул провели систематичний огляд щодо систем рекомендацій з використанням трансформерних моделей [3]. Вони аналізують існуючі дослідження щодо застосування архітектур Transformer, таких як BERT і GPT у завданнях рекомендацій. Огляд обговорює різні аспекти, включаючи архітектуру моделі, методи навчання та оцінку ефективності.

Було виявлено, що NLP також можуть бути класифіковані за методами представлення тексту:

– Bag-of-Words – моделі, які представляють текст у вигляді невпорядкованих колекцій слів;
– Word Embeddings – використання векторного представлення слів для зберігання контексту та семантики.

Ель-Кафраві та Рагаб представили огляд і порівняння рекомендаційних систем на основі векторних представлень[4]. Автори дослідили різні методи вбудовування, такі як Word2Vec і GloVe, і їх застосування в рекомендаційних завданнях. Дослідження оцінює ефективність цих методів вбудовування в охопленні семантики предметів і уподобань користувачів, пропонуючи зрозуміти їх сильні сторони та обмеження в сценаріях рекомендацій.

Огляд наукової та патентної літератури з даної теми дозволяє виділити ряд перспективних напрямків досліджень, до яких належать:

– використання моделей NLP для вилучення та представлення інформації з текстових даних, акцентуючи увагу на використанні моделей NLP не лише для розуміння контексту тексту, а й для визначення емоційної складової [5];
– розробка нових методів оцінки ефективності рекомендаційних систем на основі NLP. Це важливо, оскільки для цих систем не існує стандартного протоколу оцінювання.

1.3 Визначення проблем та актуалізація рішень

Багато з традиційних рекомендаційних методів не враховують контекстуальну інформацію текста так добре, як це могли б робити сучасні моделі NLP. Також вони можуть поступатися в точності і якості рекомендацій і мають обмежену здатність до масштабування. Таким чином, більш сучасні моделі NLP можуть пропонувати значні покращення в якості і релевантності рекомендаційних систем, що і робить це дослідження актуальним.

Виходячи з огляду наукової та патентної літератури, було виявлено низку проблем, що виникають в цій області. Однією з таких проблем є неоднорідність

текстових даних. Через те, що тексти є неоднорідними за своєю структурою та контекстом, існує необхідність використання та розробка методів, що б ефективно враховували цю неоднорідність. Наступною проблемою є складність тексту, який обробляється та його стилістика, особливо, якщо текст містить сленгові та жаргонні слова і є неформальним. Іншою важливою проблемою є те, що рекомендаційні системи можуть не зрозуміти контекст та надати хибні рекомендації, тому важливо, щоб моделі NLP враховували стилістичні особливості тексту. Пошук контекстуальних зв'язків між словами та розумінням контексту є також однією з проблем при наданні рекомендацій.

Також було виявлено низку протиріч у відомих теоретичних та експериментальних результатах. Ці протиріччя є наслідками використання різних наборів даних, різних систем оцінювання ефективності, точності та якості, а також використання різних версій моделей НЛП. Існують певні розбіжності щодо ефективності моделей НЛП для рекомендаційних завдань. Важливо вирішити ці протиріччя, щоб краще зрозуміти сильні та слабкі сторони моделей НЛП для використання у рекомендаційних системах.

Огляд наукової та патентної літератури даної теми дозволяє виявити низку перспективних напрямків досліджень, до яких належать:

- впровадження нових моделей NLP для вилучення та представлення інформації з текстових даних, що зосереджують використання моделей НЛП не лише для розуміння контексту тексту, а ще й для визначення настрою користувачів;
- розробка нових методів оцінки ефективності рекомендаційних систем на основі НЛП. Це важливо, оскільки для цих систем не існує стандартного протоколу оцінювання;
- використання методів машинного навчання для навчання моделей NLP на наборах даних, які включають текст та метадані можуть поліпшити розуміння контексту та зв'язків, так само як і розробка спеціальних методів обробки природної мови, адаптованих до потреб

рекомендаційних систем, таких як обробка великих обсягів даних та надання релевантних рекомендацій.

Узагальнюючи, дослідження в цьому напрямку є важливим для подальшого розвитку рекомендаційних систем, оскільки текстова інформація є однією з ключових елементів в сучасному вирішенні завдань рекомендацій та персоналізації, а дослідження в галузі NLP для рекомендаційних систем може вирішити значну частину проблем в цій області. Окрім того, дане дослідження є доцільним, оскільки з’являються нові рішення та моделі для обробки природної мови. Моделі трансформаторів, такі як BERT і GPT, продемонстрували чудову ефективність у задачах розуміння природної мови, що робить їх використання для таких завдань, як аналіз настроїв і семантичне розуміння в системах рекомендацій доцільним. Подібним чином моделі вбудовування слів, такі як fastText, пропонують ефективне представлення слів у векторному просторі, полегшуючи значущі порівняння та асоціації між словами. Використовуючи ці методи, системи рекомендацій можуть краще розуміти вподобання користувачів і надавати точні та персоналізовані рекомендації.

1.4 Постановка задачі

Метою роботи є оцінка ефективності моделей NLP для рекомендаційних систем на основі тексту, проведення експериментів для аналізу, порівняння та оцінки ефективності моделей природної мови за певними метриками, навчання та оцінка моделей NLP, адаптованих до вимог рекомендаційних систем з урахуванням настрою та особливостей текстових даних на основі проведених експериментів та розробка рекомендаційної системи на основі отриманих результатів.

Постановкою задачі для виконання кваліфікаційної роботи та досягнення поставленої мети є:

– виявлення проблеми для дослідження;

- здійснення аналізу предметної області, аналіз наукової та патентної літератури за даною тематикою та постановка задачі;
- аналіз та опис існуючих моделей NLP для рекомендаційних систем на основі тексту;
- проведення теоретичних досліджень: порівняння моделей на основі відкритих даних з наукових та інших джерел для формулювання вибірки моделей для проведення експериментальних досліджень;
- проведення експериментальних досліджень за допомогою мови програмування Python, що складається з чотирьох етапів: виявлення емоційної складової текстів, багатокласова класифікація описів фільмів за жанрами, задача знаходження подібністі текстів та порівняння результатів та оцінка моделей NLP;
- висновок та оцінка застосування результатів проведених досліджень;
- застосування отриманих результатів для реалізації рекомендаційної системи з використанням NLP моделей на основі тексту, опису думок, настрою чи подій користувача для надання рекомендацій, що буде складатися з бази даних, серверної та клієнтської частин.

Об'єктом дослідження є моделі NLP для обробки природної мови.

Предметом дослідження є процес застосування моделей обробки природної мови у рекомендаційних системах для аналізу текстової інформації, що включає виявлення емоційної складової текстів, багатокласову класифікацію, знаходження подібності текстів та надання персоналізованих рекомендацій.

2 АНАЛІЗ ІСНУЮЧИХ МОДЕЛЕЙ ОБРОБКИ ПРИРОДНОЇ МОВИ

Проаналізувавши предметну область, найвідомішими моделями NLP, що можуть використовуватись в рекомендаційних системах є сучасні моделі-трансформери, моделі, основані на векторному представлені та рекурентні нейронні мережі.

2.1 Моделі векторного представлення

У моделях векторного представлення слова або деякі фрази зі словника зіставляються до відповідних векторів дійсних чисел, які далі використовуються для розуміння значення контексту слів, пошуку подібностей та семантики слів.

Процес приведення слів до числових значень називається векторизацією.

В даних моделях важливу роль відіграє розмірність вектора – кількість вимірів, у яких визначено векторне представлення слова. Зазвичай це фіксоване значення, яке визначається під час створення вектора. Розмірність вектора представляє загальну кількість ознак, які закодовані у векторному представленні. Векторні представлення створюються таким чином, щоб слова зі схожим контекстом або використанням були близькими одне до одного в просторі. Тобто, семантично схожі слова мають схожі векторні представлення.

Існують різні методи створення векторного представлення.

Word2Vec – модель, що використовує нейронну мережу для навчання, враховуючи синтаксичні та семантичні зв'язки слова з іншими словами у тексті. Word2Vec використовує контекстне навчання, що означає, що векторні представлення слів визначаються на основі їх контексту у тексті і слова, які зустрічаються поряд часто отримують схожі векторні представлення [6].

Word2Vec використовує два основних підходи до навчання: Skip-gram і Continuous Bag of Words (CBOW).

Skip-gram намагається передбачити оточуючі слова на основі центрального слова, тоді як CBOW передбачає центральне слово на основі оточуючих.

Навчання моделі Word2Vec складається з етапів, що наведені нижче.

Перший етап – ініціалізація векторів слів. Кожне слово в словнику отримує свій унікальний вектор.

Другий етап – вибір центрального слова та оточуючих слів для отримання контексту. На кожному кроці обирається центральне слово та його контекст. Центральне слово – це слово, для якого модель намагається передбачити контекст (у випадку моделі Skip-gram, це слово, яке модель намагається передбачити на основі контексту).

Векторне представлення слова w за моделлю Skip-gram рахується за формулою (1).

$$v_w = \Sigma_{t \in C_w} P(t|w) \cdot u_t, \quad (1)$$

де v_w – вектор слова w,

C_w – контекст слова w,

u_t – вектор слова t, яке є членом контексту C_w,

P(t|w) – ймовірність того, що слово t з'явиться в контексті слова w.

Векторне представлення слова w у моделі CBOW визначається за формулою (2).

$$v_w = \Sigma_{t \in C_w} P(w|t) \cdot u_t, \quad (2)$$

де v_w – вектор слова w,

C_w – контекст слова w,

u_t – вектор слова t, яке є членом контексту C_w,

P(w|t) – ймовірність того, що слово w з'явиться в контексті слова t.

Наступний етап – розрахунок ймовірностей.

Для моделі Skip-gram відбувається обчислення ймовірності входження кожного слова в контексті центрального слова.

Обчислення ймовірності з'явлення слова t в контексті слова w у моделі Skip-gram відбувається за формулою (3).

$$P(t|w) = \frac{e^{v_w \cdot u_t}}{\sum_{t' \in V} e^{v_w \cdot u_{t'}}}, \quad (3)$$

де P(t|w) – ймовірність того, що слово t з’явиться в контексті слова w,

v_w – вектор слова w,

t’ – індекс слова в множині слів,

u_t – вектор слова t,

V – множина слів, словник.

Для моделі CBOW обчислення ймовірності входження центрального слова в контексті його оточуючих слів відбувається за формулою (4).

$$P(w|C_w) = \frac{e^{v_w \cdot C_w}}{\sum_{w' \in V} e^{v_{w'} \cdot u_{C_w}}}, \quad (4)$$

де $P(w|C_w)$ – ймовірність того, що слово w з'явиться в контексті його оточення C_w,

v_w – вектор слова w,

t’ – індекс слова в множині слів,

u_{Cw} – вектор, що представляє контекст C_w,

w’ – індекс (представлення) слова в словнику,

V – множина слів, словник.

Наступний етап – це розрахунок оцінки втрат на основі того, наскільки великою є різниця між розрахованою ймовірністю та фактичною появою слова в контексті та оптимізація. Задачею оптимізації є мінімізація втрат.

В моделі Skip-gram ми максимізуємо ймовірність з'явлення контекстних слів t′ при відомому центральному слові w. Розрахунок логарифмічної ймовірності (5).

$$\log P(C_w|w) = \sum_{t' \in C_\omega} \log P(t'|w), \quad (5)$$

де C_w – контекст слова w,

P(t'|w) – ймовірність приналежності слова t' до контексту слова w.

У (6) наведена мінімізація витрат.

$$Loss_s = -\log P(C_w|w). \quad (6)$$

Для моделі CBOW цей процес відбувається навпаки, ми максимізуємо ймовірність входження центрального слова w при контексті C_w (7).

$$\log P(w|C_w) = \sum_{t' \in C_\omega} \log P(w|t'). \quad (7)$$

Мінімізація витрат відбувається за формулою (8).

$$Loss_C = -\log P(w|C_w). \quad (8)$$

Наступним кроком на основі градієнта втрат вектори слів оновлюються за допомогою методу градієнтного спуску.

Увесь процес повторюється для центральних слів та їх контекстів або, в залежності від моделі, для багатьох контекстів та їх центральних слів.

Для визначення семантичної схожості слів використовується косинусна подібність – це косинус кута між двома векторами, який дає нам кутову відстань між векторами (9).

$$cos(\theta) = \frac{v_1 \cdot v_2}{\|v_1\| \cdot \|v_2\|}, \quad (9)$$

де v_1,v_2 – два вектори між якими шукаємо подібність.

На рисунку 2.1 наведено графічне представлення двох векторів у двовимірному просторі.

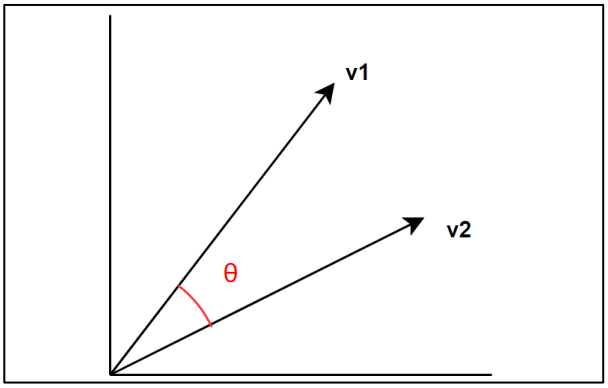

Рисунок 2.1 – Представлення двох векторів у 2d просторі (створено самостійно)

Не дивлячись на те, що Word2Vec є дуже популярною моделлю векторного представлення, в ній існує ряд недоліків: модель Word2Vec не враховує порядок слів в реченні під час навчання, через що може втратитися частина змісту, взаємозв'язку та контексту цих слів. Модель надає усім однаковим словам один і той самий вектор, не дивлячись на можливі інші значення в різних контекстах використання. Контекст слів може бути дуже обмежений, оскільки враховується лише контекст у певному вікні, а не в цілому реченні чи тексті.

Ще одною моделлю векторного представлення є модель GloVe. Вона враховує глобальні статистичні залежності між словами у великих текстових корпусах. Навчання Glove відбувається на основі технік матричної факторизації та складається з наступних етапів [7].

Першим етапом є створення матриці спільного входження, що визначає статистичні зв'язки між словами в корпусі текстів, де кожний елемент P_{ij} відображає ймовірність того, що слово j зустрінеться зі словом і в заданому контексті.

Другим етапом є ініціалізація векторних представлень слів w_i та $\tilde{w}_j$, а також зсувів b_i, $\tilde{b}_j$ випадковим чином.

Наступною йде оптимізація параметрів за допомогою градієнтного спуску, з використанням матриці спільного входження та векторних представлень слів, обчислюється функція втрат і проводиться оптимізація параметрів моделі.

Цей повторюється доти, доки не досягається задовільний рівень оптимізації.

Після завершення оптимізації отримуються оновлені векторні представлення слів, які враховують семантичні відносини та частоту спільного вживання слів в контексті.

Приклад матриці ко-входжень слів.

Нехай, маємо корпус з 4 речень:

```
corpus =
    ["Кіт має цей капелюх",
    "Цей кіт чорний",
    "Цей капелюх чорний",
    "Чорний кіт"]
```

Створюємо словник індексів слів (див. рис. 2.2).

```
In [3]: #Створення словника індексів слів
        word_index = {}
        for sentence in corpus:
            tokens = sentence.lower().split()
            for token in tokens:
                if token not in word_index:
                    word_index[token] = len(word_index)

In [11]: print("Словник індексів:")
         print(word_index)

         Словник індексів:
         {'кіт': 0, 'має': 1, 'цей': 2, 'капелюх': 3, 'чорний': 4}
```

Рисунок 2.2 – Словник індексів слів (створено самостійно)

Отримали наступний словник: {'кіт': 0, 'має': 1, 'цей': 2, 'капелюх': 3, 'чорний': 4}

Наступним кроком є підрахунок входжень слів до контексту та створення матриці ко-входжень (див. рис. 2.3).

```
In [17]: #Ініціалізація матриці ко-входжень
         co_occurrence_matrix = np.zeros((len(word_index), len(word_index)))

         #Підрахунок кількості спільних входжень між словами
         for sentence in corpus:
             tokens = sentence.lower().split()
             for i, target_word in enumerate(tokens):
                 target_index = word_index[target_word]
                 context_words = tokens[max(0, i-2):i] + tokens[i+1:i+3]
                 for context_word in context_words:
                     context_index = word_index[context_word]
                     co_occurrence_matrix[target_index, context_index] += 1

         print("Матриця ко-входжень:")
         print(co_occurrence_matrix)

         Матриця ко-входжень:
         [[0. 1. 2. 0. 2.]
          [1. 0. 1. 1. 0.]
          [2. 1. 0. 2. 2.]
          [0. 1. 2. 0. 1.]
          [2. 0. 2. 1. 0.]]
```

Рисунок 2.3 – Матриця ко-входжень (створено самостійно)

Далі треба факторизувати цю матрицю, отримавши дві векторні матриці: слів та контексту, які оптимізуються під час навчання.

Факторизація матриці логарифмів частот спільного входження X наведена у формулі (10).

$$X = U \cdot V^T, \tag{10}$$

де U – матриця векторних представлень слів,

V^T – транспонована матриця векторних представлень оточуючих слів (контексту),

X – матриця логарифмів частот спільного входження.

Мінімізація функції витрат J (оптимізація) обчислюються як відхилення взаємодії між словами та контекстом від логарифму їхнього спільного входження за формулою (11).

$$J = \sum_{i,j=1}^{v} f(X_{ij}) \cdot \left(w_i^T \cdot \widetilde{w}_j + b_i + \widetilde{b_j} - \log(X_{ij})\right)^2, \quad (11)$$

де X_{ij} – матриця кількісті входжень слова w_i в контексті слова $\widetilde{w}_j$,

$f(X_{ij})$ – вагова функція,

w_i ,$\widetilde{w}_j$ – векторні представлення слів та контексту слів,

b_i, $\tilde{b}_j$ – зсуви для векторного представлення слів та контексту.

Недоліками даної моделі є фіксований розмір вектора для кожного слова, що може негативно сказатися на словах, які мають декілька значень. Модель не завжди точно враховує синтаксичні та семантичні відносини, оскільки вона основана на статистичних властивостях корпусу текстів. Слова з низькою частотою входження можуть мати менш точні векторні представлення порівняно зі словами, що мають вищу.

Наступною моделлю є FastText – розширення моделі Word2Vec, модель векторного представлення слів, що працює з n-грамами (підсловами).

Основні відмінності з Word2Vec полягають в тому, що модель FastText здатна працювати з підсловами та обробляти слова поза словником, тобто знаходити вбудовані слова, яких немає на момент навчання.

Слова поза словником (OOV) – це слова, які не зустрічаються під час навчання даних і відсутні в словнику моделі. Моделі векторного представлення слів, такі як Word2Vec або GloVe не можуть цього.

Здатність моделі FastText працювати з n-грамами (підсловами) дозволяє більше уваги приділяти словам, що рідко зустрічаються у текстовому корпусі та словам, які можуть бути утворені зі схожих частин слів. Модель враховує структуру слова за допомогою підслів, що може поліпшити розуміння семантики та контексту Підслова визначаються як уніграми та біграми символів, які зустрічаються в слові [8]. Наприклад, слово “кіт” у FastText буде представлено у вигляді:

– уніграми: <к, і, т>;

– біграми: <кі, іт, т>.

Word2Vec має недоліки з полісемією, коли одне слово має багато значень, оскільки кожне слово має лише один вектор. Завдяки ж використанню n-грамів, FastText може створювати вектори для кожного значення слова окремо. Моделі Word2Vec та FastText навчаються на статичних корпусах тексту, тому їхні вбудовані словники складаються зі слів, що зустрічаються в цьому корпусі і можуть сягати мільйонів.

2.2 Рекурентні нейронні мережі

LSTM (Long Short-Term Memory) – рекурентна нейронна мережа, що здатна зберігати та використовувати довгострокові залежності в послідовностях. Проблема рекурентних нейронних мереж полягає в тому, що вони просто зберігають попередні дані у своїй «короткочасній пам’яті». Коли пам’ять вичерпується, він просто видаляє інформацію, та замінює її новими даними. Модель LSTM намагається уникнути цієї проблеми, зберігаючи вибрану інформацію в довготривалій пам’яті. Основна частина LSTM, що відрізняє її від класичних RNN – це комірка пам'яті, яка здатна зберігати інформацію тривалий час. Ця комірка пам'яті може зберігати значення від -1 до 1.

Архітектура LSTM складається з таких основних компонентів:

Комірка пам'яті (Memory Cell) – це комірка пам'яті, яка здатна зберігати інформацію тривалий час та може зберігати значення від -1 до 1.

Шлюзи. LSTM має три типи воріт – шлюз забування (Forget Gate), Шлюз входу (Input Gate) та Шлюз виходу (Output Gate). Ці елементи архітектури регулюють потік інформації в комірку пам'яті та виходу з неї.

Шлюз забування визначає, про яку частину інформації у комірці пам'яті може бути забуто.

Шлюз входу визначає, яка нова інформація буде додана до комірки.

Шлюз виходу визначає, яка інформація з комірки пам'яті буде використана для прогнозування вихідних даних.

Функції активації. LSTM використовує функції активації, такі як гіперболічний тангенс і сигмоїдна функція для регулювання значень, що проходять через шлюзи та комірку пам'яті [9]. Архітектуру LSTM наведено на рисунку 2.4.

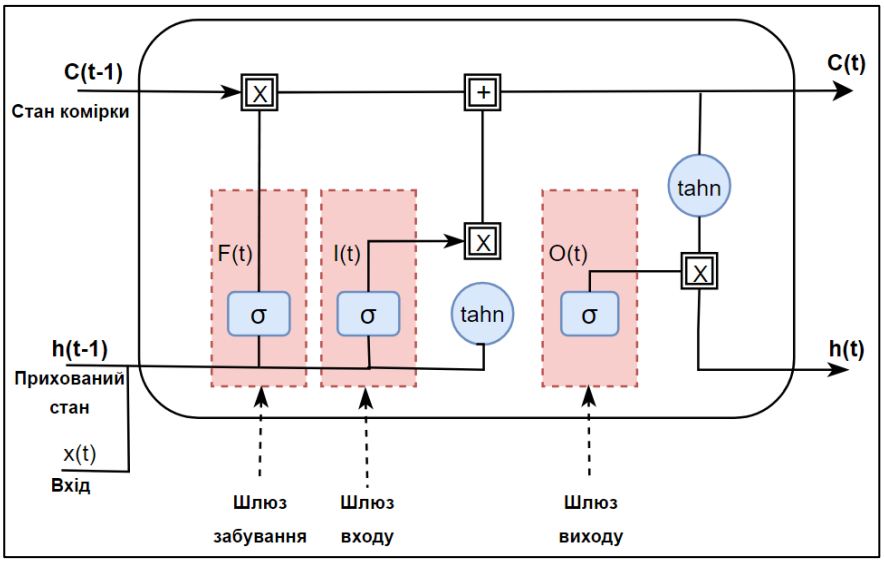

Рисунок 2.4 – Архітектура LSTM (створено самостійно)

Як вже було зазначено вище, у Шлюзі забування F(t) вирішується, яка поточна та попередня інформація зберігається, а якою можна знехтувати та видалити. Це включає прихований стан попереднього проходу та поточний вхід. Ці значення передаються в сигмоїдну функцію σ, яка може виводити лише значення від 0 до 1, де 0 означає, що попередню інформацію можна забути, оскільки, можливо, є нова, важливіша інформація, 1 означає, відповідно, що попередня інформація може бути збережена. Ці результати множаться на поточний стан комірки, таким чином, що знання, які більше не потрібні зникають.

У Шлюзі входу I(t) вирішується, наскільки цінним є поточний вхід для вирішення завдання. Для цього поточний вхід множиться на прихований стан і вагову матрицю останнього запуску. Уся інформація, яка здається важливою у вхідних воротах потім додається до стану комірки c(t-1) та формує новий стан комірки c(t). Цей новий стан комірки тепер є поточним станом довгострокової пам’яті та буде використовуватися під час наступного запуску.

У Шлюзі виходу O(t) вихід моделі LSTM обчислюється в прихованому стані. Залежно від застосування це може бути, наприклад, слово, яке доповнює зміст речення. Для цього сигмоїдна функція вирішує, яка інформація може надходити через вихідний вентиль, а потім стан комірки множиться після його активації за допомогою функції tanh.

Шлюзи обчислюються за нижче наведеними формулами (12).

$$\begin{aligned} I_t &= \sigma(X_t W_{xi} + h_{t-1} W_{hi} + b_i) \\ F_t &= \sigma(X_t W_{xf} + h_{t-1} W_{hf} + b_f) \\ O_t &= \sigma(X_t W_{xo} + h_{t-1} W_{ho} + b_o), \end{aligned} \tag{12}$$

де W_{xi}, W_{xf}, W_{xo} – матриці вагових коефіцієнтів, що зв'язують вхідний вектор з вхідними шлюзами, шлюзом забування і вихідним шлюзом відповідно,

W_{hi}, W_{hf}, W_{ho} – матриці вагових коефіцієнтів, що зв'язують прихований стан з вхідними шлюзами, забутим шлюзом і вихідним шлюзом відповідно,

b_i, b_f, b_o – вектори зсувів, що додаються до виразів для розрахунку шлюзів,

X_t - це вхідний вектор, що містить дані для поточного кроку обробки,

h_{t-1} – прихований стан, вектор, що містить стан мережі після попереднього кроку обробки,

I_t – вхідний шлюз, вектор, що визначає, скільки інформації з вхідного вектору буде передано до прихованого стану,

F_t – шлюз забування, вектор, що визначає, скільки інформації з попереднього стану буде забуто,

O_t - вихідний шлюз, вектор, що визначає, скільки інформації з прихованого стану буде передано на вихід.

Вхідний вузол $\tilde{C}_t$ використовує функцію активації tanh, яка повертає значення в діапазоні від -1 до 1. Це дозволяє вхідному вузлу генерувати нові значення стану пам'яті, які можуть бути як негативними, так і позитивними. Він дозволяє мережі обробляти поточні вхідні дані та попередні вихідні стани, щоб генерувати нові значення стану пам'яті. Це дозволяє мережі зберігати

інформацію в часі і навчатися на послідовності даних. Формула для обчислення вхідного вузла (13):

$$\tilde{C}_t = \tanh(XW_{xc} + h_{t-1}W_{hc} + b_c), \quad (13)$$

де W_{xc} і W_{hc} – вагові параметри,

b_c – дельта.

Внутрішній стан комірки пам'яті C_t – це вектор, який зберігає інформацію про попередні вхідні дані та стан мережі.

Для оновлення внутрішнього стану комірки пам'яті застосовується формула (14).

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t, \quad (14)$$

де C_t – внутрішній стан комірки пам'яті,

$\tilde{C}_t$ – оновлений внутрішній стан комірки пам'яті,

$\odot$ – покомпонентний добуток Адамара.

Якщо шлюз F_t завжди дорівнює 1, а вхідний шлюз I_t завжди дорівнює 0, то внутрішній стан комірки пам'яті буде постійним і не зміниться з часом. Однак вхідні та шлюз забування дають мережі можливість навчатися, коли зберігати старі дані, а коли заміняти їх новими, що дозволяє мережі навчатися на великих послідовностях даних, уникаючи проблеми згасання градієнта.

Останнім кроком обчислення вихідного стану комірки пам'яті є розрахунок прихованого стану h_t. Вихідний стан комірки пам'яті h_t є вектором розміру $n \times h$, (де n – кількість одиниць прихованого стану, а h – розмірність кожної одиниці), який передається на наступні шари мережі. Вихідний стан обчислюється за формулою (15).

$$h_t = O_t \odot \tanh(C_t). \quad (15)$$

Недоліками LSTM є те, що мережа вимагає значних обчислювальних ресурсів, особливо коли мова йде про навчання на великих обсягах даних. Як і у багатьох інших рекурентних нейронних мережах, у LSTM є проблема зникаючого градієнту, яка може вплинути на здатність моделі враховувати довгострокові залежності. Дана модель може вимагати збалансованих даних для ефективного тренування. Якщо деякі класи або категорії текстової інформації представлені в обмеженому вигляді, якість та точність наданих рекомендацій може бути на низькому рівні. Для завдання рекомендацій, коли ми маємо велику кількість певних невеликих текстів, описів, тощо LSTM є занадто складним інструментом.

Gated Recurrent Unit (GRU) – ще одна архітектура рекурентної нейронної мережі, яка як і LSTM розроблена для вирішення проблеми зникаючого градієнту та здатна працювати з довгостроковими залежностями в послідовностях. GRU є спрощеною версією LSTM, що має меншу кількістю параметрів, за рахунок чого має менші обчислювальні витрати.

Основними компонентами архітектури GRU є два шлюза: Шлюз оновлення (Update gate) та Шлюз скидання (Reset gate), які регулюють потік інформації до мережі (див. рис. 2.5) [10].

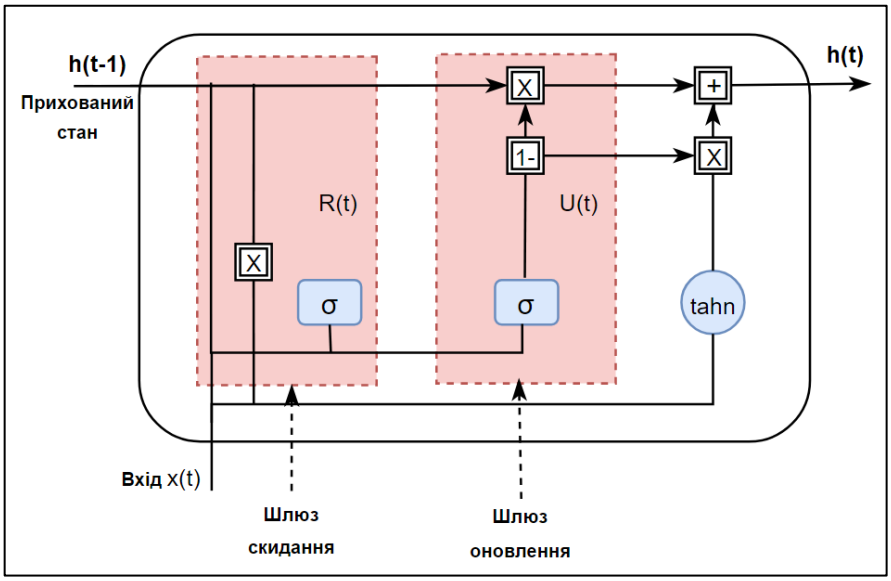

Рисунок 2.5 – Архітектура GRU (створено самостійно)

Шлюз скидання R_t відповідає за вирішення проблеми зникаючого градієнту, дозволяючи мережі вирішувати, яку частину інформації слід прибрати з попереднього кроку. Шлюз оновлення U_t визначає, яку частину інформації ми хочемо оновити в поточному стані комірки. Формули для шлюзів наведені у (16).

$$\begin{aligned} R_1 &= \sigma(X_t W_{xr} + h_{t-1} W_{hr} + b_r) \\ U_t &= \sigma(X_t W_{xu} + h_{t-1} W_{hu} + b_u), \end{aligned} \tag{16}$$

де X_t – вхідний вектор,

h_{t-1} – прихований стан мережі на попередньому кроці,

W_{xr}, W_{hr} – вагові коефіцієнти шлюзу скидання,

W_{xu}, W_{hu} – вагові коефіцієнти для шлюзу оновлення,

b_r, b_u – дельти зсуву.

Поточний стан комірки h_t – це новий стан комірки, який оновлюється на кожному кроці. Кандидат на поточний стан (новий поточний стан) $\tilde{h}_t$ обчислюється за формулою (17).

$$\tilde{h}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h), \tag{17}$$

де X_t - вхід на поточному кроці часу,

W_{xh} – матриця ваг, яка зв'язує вхідні дані з кандидатом на новий стан,

R_t – шлюз скидання, матриця ваг, що зв'язує попередній стан h_{t-1} з кандидатом на новий стан,

W_{hh} – матриця ваг, яка зв'язує кандидат на новий стан з попереднім станом,

$\odot$ – операція поелементного множення,

b_h – вектор зсуву для кандидата на новий стан,

$\tanh(\cdot)$ – гіперболічний тангенс, функція активації, яка обмежує значення в діапазоні (-1, 1).

Фінальна формула оновлення прихованого стану наведена у (18).

$$h_t = U_t \odot h_{t-1} + (1 - U_t) \odot \tilde{h}_t, \quad (18)$$

де U_t – шлюз оновлення,

h_{t-1} – попередній прихований стан,

$\tilde{h}_t$ – новий поточний стан.

Коли шлюз оновлення U_t близький до 1, ми залишаємо старий стан. У цьому випадку інформація з кандидата на новий стан $\tilde{h}_t$ ігнорується, пропускаючи крок в послідовності залежностей. Якщо ж U_t близький до 0, новий прихований стан h_t наближається до $\tilde{h}_t$.

Як і звичайні рекурентні мережі, GRU може стикатися з проблемами зникання та вибування градієнтів, що може впливати на здатність моделі навчатися на довгих послідовностях. GRU має менше параметрів порівняно з більш складними архітектурами, як LSTM. Це може бути обмежуючим фактором у вивченні складних структур мови та контекстів. Моделі важко вивчити відносини між словами, що знаходяться на великій відстані в тексті. Як і у всіх рекурентних мережах, GRU чутлива до порядку введення. Перестановка слів у реченні може вплинути на результат моделі.

2.3 Трансформери

З появою архітектури транформерів, сфера обробки природної мови (NLP) зазнала значних змін та використання рекурентних нейронних мереж для рекомендаційних систем відійшло на другий план. Трансформери революціонізували різноманітні завдання NLP, такі як машинний переклад, генерація тексту та аналіз настрою, дозволивши моделям охоплювати глобальні залежності ефективним і розпаралельованим способом. Використовуючи механізми самоконтролю, вони можуть фіксувати довгострокові залежності в

послідовності, усуваючи потребу в повторних підключеннях і забезпечуючи паралельні обчислення.

Основною відмінністю трансформерів є використання механізму уваги для моделювання залежностей між елементами послідовностей.

У порівнянні з рекурентними нейронними мережами, трансформери можуть паралельно обробляти вхідні дані, що пришвидшує їхнє навчання.

Основними компонентами трансформерів є енкодер та декодер. Енкодер (кодувальник) складається з декількох шарів. Кожен шар трансформера складається з двох основних операцій: механізм уваги – це операція, яка визначає, наскільки важливий кожний елемент вхідної послідовності для кожного елемента вихідної послідовності та пряма нейронна мережа (FNN) – це операція, яка обробляє вихід механізму уваги. Декодер (декодувальник) також складається з декількох шарів, однак, крім двох основних операцій, що використовуються в енкодері, він ще містить додаткову операцію прогнозування, яка генерує наступний елемент вихідної послідовності [11].

Загальна схема архітектури компонентів трансформерів наведена на рисунку 2.6.

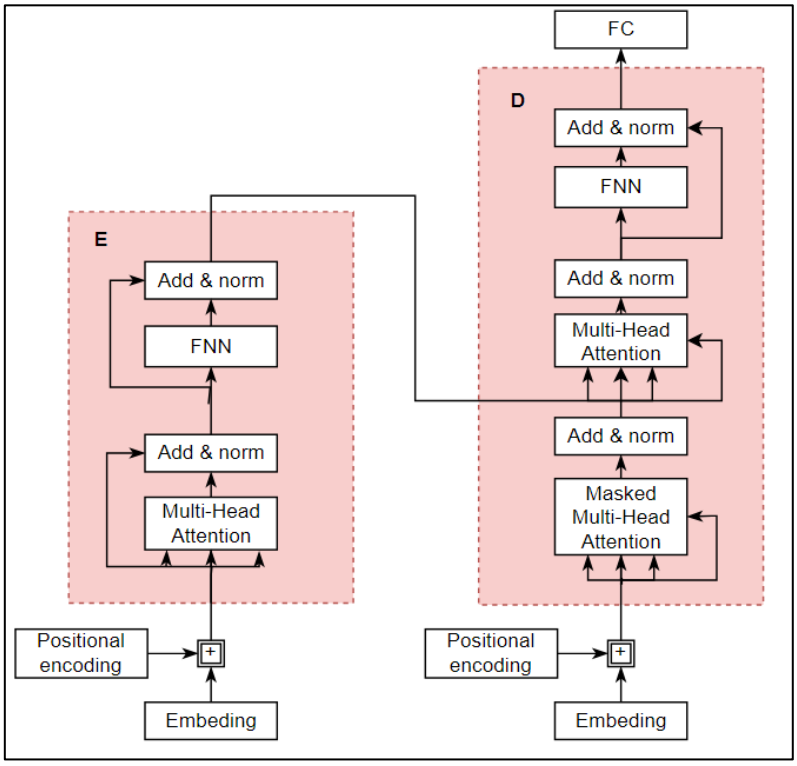

Рисунок 2.6 – Схема архітектури трансформерів (створено самостійно)

Енкодер представлено на схемі зліва. До вихідної послідовності, що складається з N блоків застосовуються по черзі наступні дії: кожен блок видає послідовність такої самої довжини. У цьому блоці є два важливі шари, multi-head attention (механізм уваги) і FNN (пряма нейронна мережа). Після кожного з них до виходу додається вхід і потім проходимо через шар нормалізації «Add & norm».

Механізм уваги для розрахунку ваг уваги та маскування позицій може бути описано формулою (19).

$$A = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{19}$$

де Q, K, V – матриці запитів, ключів та значень після лінійного перетворення,

A – матриця скалярних добутків та зважених значень,

d_k – розмірність ключа вихідних матрць Q, K.

Матриці запитів, ключів і значень Q, K, V обчислюються за формулами, що наведені у (20).

$$\begin{aligned} Q &= Y'W_Q \\ K &= XW_K, \\ V &= XW_V \end{aligned} \tag{20}$$

де X, Y – вхідні послідовності енкодера та декодера відповідно,

W_Q, W_K, W_V – це вагові матриці для запитів, ключів та значень відповідно.

Шар FNN – це блок, що містить два лінійних перетворення з функцією активації ReLU. Спочатку виконується лінійне перетворення з функцією активації ReLU, потім виконується друге лінійне перетворення (21).

$$\begin{aligned} X' &= ReLU(XW_1 + b_1) \\ Y' &= X'W_2 + b_2, \end{aligned} \tag{21}$$

де W_1, W_2 – матриці ваг, що визначають, як кожен вхідний елемент взаємодіє з іншими,

b_1,b_2 – матриці зсуву.

GPT (Generative Pre-trained Transformer) – це серія моделей для обробки природної мови, що використовують архітектуру трансформерів та основані на ідеї передбачення наступного слова в контексті тексту. GPT розроблений OpenAI і представлений у 4 ітераціях GPT-1-4.

Модель піддається пре-тренуванню на великих обсягах текстових даних. Під час цього етапу вона намагається передбачити наступні слова в контексті речення. Це дозволяє їй навчатися широким представленням мови та усвідомлювати контекстуальні залежності між словами.

Однією з ключових особливостей GPT є здатність створювати контекстуалізовані векторні представлення для слів. Це означає, що представлення слова залежить від його контексту в конкретному реченні чи тексті.

Після пре-тренування GPT може використовуватися для генерації нового тексту. Модель може продовжувати текст на основі введеного контексту, створюючи логічно продовжені речення.

Модель може бути доналаштована для конкретних завдань.

Спочатку вхідний текст розбивається на послідовність числових токенів. Далі цей список токенів проходить через Embedding Layer, перетворюючись в вектори (процес схожий на word2vec). До кожного ембеддингу додається позиційний шар. Список векторних перетворень проходить через кілька ідентичних блоків Декодера з архітектури трансформерів (див. рис. 2.7) [12].

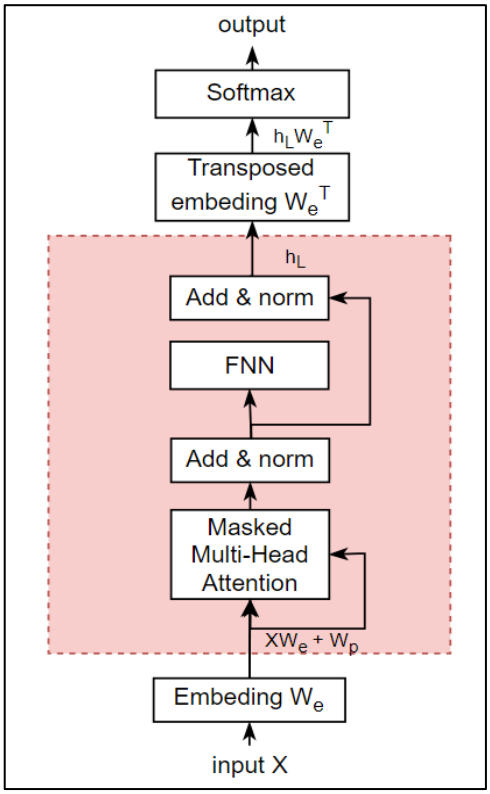

Рисунок 2.7 – Архітектура Декодера GPT (створено самостійно)

Після того, як список ембеддингів пройшов останній блок, ембеддинг, який відповідає останньому токену матрично множиться на вхідний, але вже транспонований. Після застосування SoftMax отримується розподіл ймовірностей наступного токена. З цього розподілу вибирається наступний токен (наприклад, за допомогою функції argmax). Далі цей токен додається до вхідного тексту і кроки повторюються.

GPT -1 – це перша модель серії GPT, яка була навчена приблизно на 40000 токенах ГБ текстових даних. З максимальною довжиною контексту 512 токенів модель може зберігати інформацію для відносно коротких речень або документів на запит.

Похідна від моделі GPT-1 модель GPT-2 зберігає ті самі архітектурні особливості. Однак, навчання проходить ще на більшому масиві текстових даних порівняно з GPT-1. Маючи майже 1,5 мільярда параметрів, GPT-2 демонструє значне збільшення можливостей і потенціалу для моделювання мови. Вбудований словник – 50,257 слів [13].

Модель GPT-3 є еволюцією моделі GPT-2, перевершуючи її в кількох аспектах. Вона був навчений на значно більшому масиві текстових даних і містив максимум 175 мільярдів параметрів [14]. Та разом зі збільшеним розміром GPT-3 представив кілька важливих покращень, а саме GShard (паралелізм моделі Giant-Sharded), що дозволяє розділити модель на кілька прискорювачів. Це полегшує паралельне навчання та логічний висновок, особливо для великих мовних моделей із мільярдами параметрів.

BERT(Bidirectional Encoder Representations from Transformers) – це модель представлення тексту, яка використовує архітектуру трансформера та була розроблена для досягнення високого рівня розуміння природної мови.

Ключовою технічною інновацією BERT є застосування двонапрямленого навчання трансформера до мовного моделювання. Це відрізняється від попередніх спроб, які розглядали послідовність тексту зліва направо або поєднували навчання зліва направо та справа наліво.

Окрім архітектури трансфомера, BERT має ще інші особливості, що наведені нижче.

BERT піддається попередньому навчанню (pre-training) на великому корпусі текстових даних. Під час цього етапу модель навчається передбачати наступні слова в реченні (Next Sentence Prediction) та заповнювати пропуски в тексті (Masked Language Model) [15].

Next Sentence Prediction (NSP) полягає в тому, щоб навчити модель передбачати, чи є друге речення продовженням першого в даній текстовій парі. Це важливо для розуміння зв'язків та контексту між реченнями та структури тексту.

Процес NSP включає наступні кроки: на першому кроці обирається пара речень з корпусу тексту. Далі вводяться позитивні та негативні приклади: якщо пара речень є продовженням одне одного, це вважається позитивним прикладом, якщо вони не пов'язані, то це негативний приклад. Наступним кроком вводяться мітки класу [CLS] і [SEP], які вказують на початок та кінець

речення в послідовності. Також додається мітка [IsNext] або [NotNext], яка вказує, чи є наступне речення продовженням поточного.

Останній етап – це навчання класифікатора: модель навчається класифікувати пари речень за наявністю зв'язку між ними. Це допомагає моделі розуміти синтаксичні та семантичні взаємозв'язки між реченнями.

Masked Language Model (MLM) включає в себе маскування деяких токенів у тексті та навчання моделі передбачати їхні значення на основі контексту.

Основні кроки MLM: маскування токенів: відбувається маскування деяких токенів (15%), вони замінюються токеном [MASK]. Додаються мітки класу [CLS] та [SEP] для позначення початку та кінця послідовності. Також вводяться мітки маскованих токенів для ідентифікації маскованих позицій.

Останній крок – це навчання передбачення, коли модель навчається передбачати оригінальні значення маскованих токенів. Це змушує модель розробляти контекстуальні вектори, які уможливлюють передбачення токенів на основі контексту всього тексту.

Наступним важливим аспектом BERT є отримання контекстуальних векторів: модель генерує контекстуальні вектори для кожного токену, враховуючи його контекст з обох боків, що дозволяє отримати векторні представлення, які враховують семантику та зв'язки між словами в тексті.

Отримані контекстуальні вектори можна використовувати як основу для покращення моделей рекомендацій, наприклад, для векторизації текстових описів товарів або для моделювання інтересів користувачів на основі їх взаємодій з контентом.

Після попереднього навчання BERT можна доналаштувати для конкретних завдань рекомендацій.

Модель має розмір вбудованого словника 30522 та 110 млн. параметрів [16]. Для ефективного навчання та використання BERT необхідні великі обсяги обчислювальних ресурсів та обсяги даних. Через те, що модель працює з фіксованим розміром послідовності, є обмеження щодо довжини вхідного тексту. Сама по собі модель може бути складною для інтерпретації через свою

глибоку структуру та велику кількість параметрів. Модель може не завжди ефективно враховувати довготривалі залежності та взаємодії в дуже великих текстових послідовностях.

XLNet – це модель, розроблена Google, яка також використовує трансформер як основну архітектуру. Вона поєднує два ключові концепти: авторегресійний підхід (аналогічний тому, що використовується в моделі GPT) та авторегресійно-обернутий підхід, що дозволяє краще розуміти контекст та підвищує точність в завданнях обробки природної мови NLP [17].

XLNet використовує трансформер, який включає в себе механізм уваги для ефективної обробки послідовностей та поєднує комбінацію авторегресійного та авторегресійно-обернутого підходів. Модель комбінує два підходи для контекстуалізації. Авторегресійний підхід полягає в тому, що модель передбачає наступні токени в послідовності, використовуючи попередні токени. Авторегресійно-обернутий підхід використовує інформацію з обох напрямків (ліворуч та праворуч) для кращого розуміння контексту.

Так само, як BERT, XLNet використовує підхід Masked Language Model для навчання. Тобто, частини вхідного тексту випадково маскуються, і модель повинна передбачити масковані токени на основі навколишнього контексту. Окрім цього, модель використовує додатковийметоду навчання Permutation Language Modeling, де порядок послідовності токенів переставляється перед введенням в модель, і модель повинна передбачити цей порядок та має Segment-Level Recurrent Mechanism – механізм, який дозволяє моделі взаємодіяти з сегментами тексту, щоб краще розуміти взаємозв'язки між різними частинами тексту.

Розмір вбудованого словника моделі становить 32000, а кількість параметрів моделі становить 340 млн. [18].

XLNet виявляється ефективною для різних завдань NLP завдяки своїй здатності краще моделювати контекст та використовувати двонапрямленість для отримання широкого розуміння контексту, проте потребує багато обчислювальних ресурсів.

3 ПРОВЕДЕННЯ ТЕОРЕТИЧНОГО ДОСЛІДЖЕННЯ

3.1 Виділення альтернатив та критеріїв порівняння моделей

Проаналізувавши існуючі популярні моделі для обробки природної мови, було виділено наступні сучасні моделі, для яких буде проводитися порівняння: BERT, GPT-2, GPT-3, XLNet, Word2Vec, FastText.

Для порівняння моделей для надання рекомендацій на основі тексту було виділено наступні важливі критерії, що можуть впливати на надані рекомендації:

- розмір вбудованого словника (S): вказує на кількість унікальних токенів або слів, які модель може враховувати. Більший словник може дозволяти моделі працювати з більш різноманітним контентом;
- обсяг пам’яті у гігабайтах (D): обсяг пам'яті, який потребують моделі для зберігання і обробки на основі пам'яті для зберігання параметрів моделі;
- open source (O): визначає, чи є модель доступною для публічного використання з відкритим вихідним кодом;
- простота навчання (U): можливість легко навчати чи донавчати модель є важливою для надання рекомендацій, а саме задач класифікацій.
- ефективність контекстуалізації (E): визначає, наскільки ефективно модель враховує та використовує контекст в обробці природної мови для надання рекомендацій на основі тексту.

Знайдену інформацію за кожною моделлю на основі попередньо проведеного аналізу було наведено у таблиці 3.1.

Таблиця 3.1 – Значення критерій вибору для кожної моделі (таблиця виконана самостійно)

Модель	S	D	O	U	E
BERT	30522	0,5	Так	Складно	Дуже висока

Продовження таблиці 3.1

Модель	S	D	O	U	E
GPT-2	50257	6	Так	Середня	Висока
GPT-3	50257	70	Ні	Дуже складно	Висока
Word2Vec	∞	0.1	Так	Дуже легко	Нижче середнього
XLNet	32000	1.4	Так	Середня	Дуже висока
FastText	∞	0.1	Так	Дуже легко	Середня

3.2 Опис шкал оцінок за критеріями

Наступним кроком є описання шкал оцінок за критеріями.

Критерій розмір вбудованого словника – абсолютна шкала, чим більший словник, тим різноманітніший контент модель буде здатна обробляти для отримання контексту та надання рекомендацій. Оскільки для двух моделей значення необмежене, встановимо найбільше значення серед інших моделей – 50300.

Критерій Open Source – якісна шкала, щоб перевести категоріальний критерій до кількісної використаємо наступний підхід: якщо модель є Open Source, присвоюємо значення 1, якщо ні, то 0.

Простота навчання: приводимо якісну шкалу до порядкової кількісної від 1 до 5:

- дуже легко: модель не потребує великих обчислювальних ресурсів і легко адаптується до нових даних – 5;
- середня: потребує більших ресурсів і часу – 3;
- складно: модель вимагає значну кількість ресурсів та часу– 2;
- дуже слладно: модель вимагає дуже велику кількість ресурсів та часу через значну кількість параметрів, донавчання також є складним – 1.

Чим більша здатність до оновлення – тим краще.

Переведення ефективності контекстуалізації в порядкову кількісну шкалу від 1 до 5:

– дуже висока – 5;

– висока – 4;

– середня – 3;

– нижче середнього – 2;

– низька – 1;

Чим більше значення – тим краще, оскільки розуміння контексту дуже важливе для рекомендаційних систем. Переведені шкали за критеріями наведені у таблиці 3.2.

Таблиця 3.2 –Переведення критеріїв до шкал (виконана самостійно)

Модель	Розмір вбудованого словника	Обсяг пам’яті (гб)	Open Source	Простота навчання	Ефективність контекстуалізації
BERT	30522	0.5	1	2	5
GPT-2	50257	6	1	3	4
GPT-3	50257	70	0	1	4
Word2Vec	50300	0.1	1	5	2
XLNet	32000	1.4	1	3	5
FastText	50300	6	1	5	3

За шкалою обсягу пам’яті виходить, що чим значення менше, тим краще, тому приводимо шкалу до оптимальності за принципом максимуму і перетворюємо її на нову шкалу: економія за обсягом пам’яті, що наведено у таблиці 3.3.

Таблиця 3.3 – Приведення шкал до оптимальності за принципом максимуму (таблиця виконана самостійно)

Модель	Розмір вбудованого словника	Економія за обсягом пам’яті	Open Source	Простота навчання	Ефективність контекстуалізації
BERT	30522	69.5	1	2	5
GPT-2	50257	64	1	3	4
GPT-3	50257	0	0	1	4
Word2Vec	50300	69.9	1	5	2
XLNet	32000	68.6	1	3	5
FastText	50300	69.9	1	5	3

За принципом Парето дивимось на множину альтернатив, та порівнюємо між собою. Бачимо, що GPT-3 на фоні GPT-2 програє за усіма параметрами, тому цю модель прийнято вважати неефективною (таблиця 3.4). Так само, бачимо, що модель FastText переважає Word2Vec за усіма параметрами, тому її теж викреслюємо.

Таблиця 3.4 – Результат після використання принципу Парето (таблиця виконана самостійно)

Модель	Розмір вбудованого словника	Економія за об’ємом пам’яті	Open Source	Простота навчання	Ефективність контекстуалізації
BERT	30522	69.5	1	2	5
GPT-2	50257	64	1	3	4
FastText	50300	69.9	1	5	3
XLNet	32000	68.6	1	3	5

3.3 Нормування критеріїв

Наступним кроком йде нормування критеріїв за принципом урахування мінімального та максимального значень. Формула для нормування наведена у (22).

$$\frac{f\text{крит}-fmin}{fmax-fmin}, \quad (22)$$

де $f_{крит}$ – значення критерія,

f_{min} – мінімальне значення за шкалою серед критеріїв,

f_{max} – максимальне значення серед значень критеріїв.

Встановимо мінімальні та максимальні значення для кожного критерія:

- розмір вбудованого словника: min=25000, max=50300;
- економія за о’ємом пам’яті: min=64, max = 69.9;
- open source: min=0, max=1;
- простота навчання: min=2, max=5;
- ефективність контекстуалізації: min=3, max=5.

Результати нормування критеріїв представлені у таблиці 3.5.

Таблиця 3.5 – Результати нормування критеріїв (виконана самостійно)

	Розмір вбудованого словника	Економія за кількістю даних	Open Source	Простота навчання	Ефективність контекстуалізації
BERT	0	0.932	1	0	1
GPT-2	0.998	0	1	0.333	0.5
FastText	1	1	1	1	0
XLNet	0.075	0.780	1	0.333	1

Для того, щоб знайти модель, яка підходить найбільше, будемо використовувати лінійну адитивну згортку з ваговими коефіцієнтами, оскільки певні критерії мають більший вплив на ефективність та вибір моделі. Формула згортки наведена у (23).

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j \alpha_{i_j} \tag{23}$$

де $\alpha_j -$ нормуючі множники,

$\beta_j -$ вагові коефіцієнти.

Як вже було сказано, деякі критерії мають більший внесок та є важливішими за інші, тому необхідно ввести вагові коефіцієнти.

За методом ранжування, оцінимо важливість критеріїв за наступним порядком: найважливішим критерієм виступає ефективність контекстуалізації, оскільки для рекомендаційних систем на основі тексту важливо, щоб модель ефективно розуміла та враховувала контекст. Далі йде розмір вбудованого словника. Він вказує на те, скільки різноманітних слів може враховувати модель. Більший словник може дозволити роботу з різноманітнішим контентом. Далі йде простота навчання, цей критерій значимий та свідчить про можливість моделі адаптуватися до різних умов. Економія за об’ємом пам’яті вказує на обсяг пам’яті, який необхідний для коректної роботи моделі. Оскільки після застосування Парето, моделі, що не мають відкритого доступу було виключено, open source не є важливим критерієм.

Отже, маємо такі ранги:

– ефективність контекстуалізації: 5;
– розмір вбудованого словника: 4;
– простота навчання: 3;
– економія за об’ємом даних: 2;
– open source : 1;

Сума рангів: 5 + 4 + 3 +2 + 1 = 15.

Вагові коефіцієнти:

– ефективність контекстуалізації = 5/15;
– розмір вбудованого словника= 4/15;
– здатність до оновлення = 3/15;
– економія за об’ємом даних = 2/15;
– open source = 1/15.

3.4 Розрахунок коефіцієнтів корисності та аналіз результатів

Для розрахунку корисності моделей за лінійною адитивною згорткою використаємо програмний застосунок Excel. Отримані результати наведені на рисунку 3.1.

	Розмір вбудованого словника	Економія за об'ємом пам'яті	Open Source	Простота навчання	Ефективність контекстуалізації	**K**
BERT	0	0,932	1	0	1	**0,21**
GPT-2	0,998	0	1	0,333	0,5	**0,22263**
FastText	1	1	1	1	0	**0,26836**
XLNet	0,7	0,5833	1	0,67	0,75	**0,29478**
βj	0,27	0,133333333	0,07	0,2	0,33	
αj	0,370644922	0,397566891	0,25	0,499251123	0,444444444	

Рисунок 3.1 – Розрахунок корисності моделей (створено самостійно)

В результаті, отримали розрахунки коефіцієнту корисності за заданими критеріями для досліджуваних моделей на основі відомих даних. Відобразимо графічно ці коефіцієнти для наглядності (див. рис. 3.2).

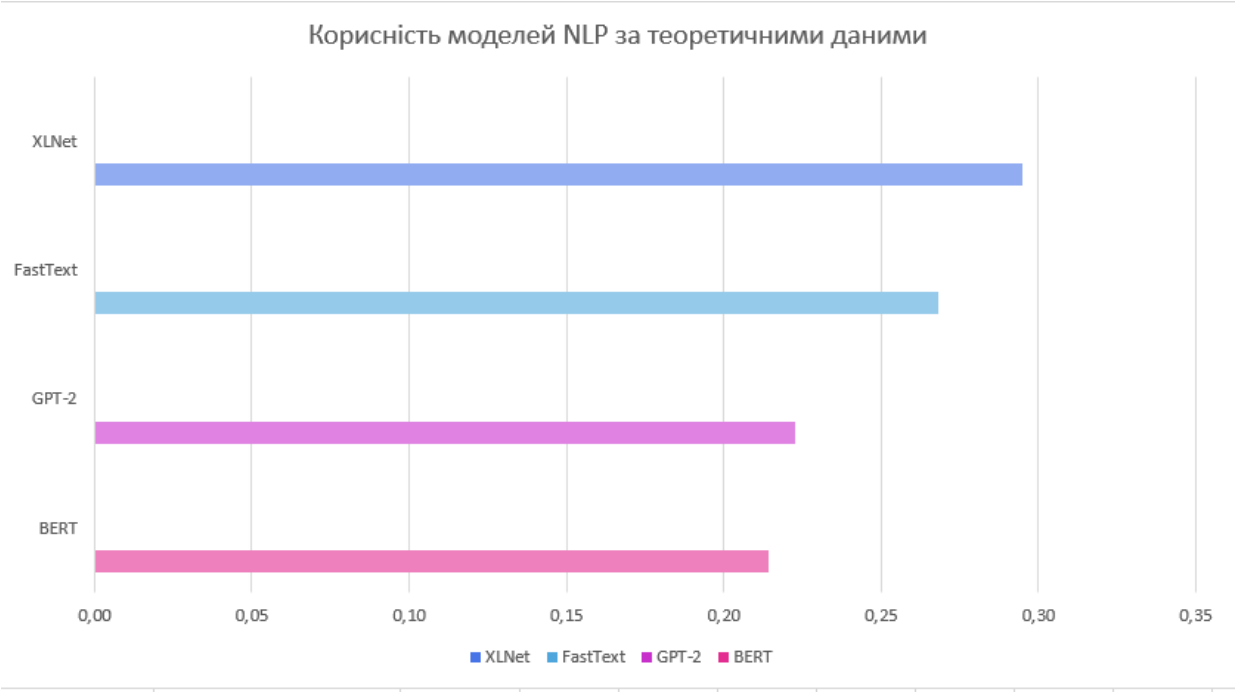

Рисунок 3.2 – Коефіцієнти корисностей моделей (створено самостійно)

Подивившись на результати корисності, можна побачити, що коефіцієнти досить близькі та жодна модель не є беззаперечним лідером. Схожість результатів свідчить про те, що всі моделі мають свої сильні та слабкі сторони, і вибір моделі залежить від того, які критерії є найбільш важливими. Наприклад, якщо важлива простота навчання і економія пам'яті, FastText може бути кращим вибором. Якщо потрібна ефективна контекстуалізація, то BERT, XLNet чи GPT можуть бути більш пріоритетним варіантом.

Як бачимо, найбільший коефіцієнт корисності, враховуючи усі критерії та вагові коефіцієнти, отримала модель XLNet, він становить 0,294. Модель комбінує авторегресійний та авторегресійно-перевернутий підходи, дозволяючи високу ефективність контекстуалізації. Має великий розмір вбудованого словника, але вимагає значної кількості ресурсів для навчання. BERT та GPT-3, є моделями з великими вбудованими словниками. Вони ефективні для задач, де контекст є важливим, проте також є ресурсомісткими. Модель FastText може бути менш ефективною в контекстуалізації порівняно із попередніми моделями, але не вимагає великих обчислень та процес навчання проходить досить легко та швидко.

4 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

На основі проведених теоретичних досліджень та на основі отриманих результатів, можна перейти до проведення експериментальних досліджень.

Експериментальні дослідження будуть складатися з наступних етапів:

– формування датасетів, для трьох задач: сентиментальний аналіз, багатокласова класифікація фільмів за жанрами для надання рекомендацій та передбачення подібності текстів;
– очищення та обробка датасетів;
– програмування та навчання обраних для дослідження моделей;
– оцінка моделей для трьох задач в результаті досліджень;
– аналіз результатів та моделей за отриманими метриками.

На основі проведених експериментів буде зроблено висновок, щодо ефективності моделей. Далі буде розроблено програмну систему з використанням отриманих результатів для надання рекомендацій користувачу на основі тексту. На основі текстового опису того, як пройшов день користувача чи його настрою, будуть надаватися рекомендації фільмів, що можуть бути цікаві цьому користувачу. Рекомендаційна система буде складатися з 2 частин: серверна та клієнтська частина.

4.1 Метрики, методи та матеріали дослідження

Точність рекомендацій – міра того, наскільки часто модель правильно класифікує позитивні прогнози. Вона визначається як відношення кількості правильних позитивних прогнозів до загальної кількості позитивних прогнозів (24).

$$P = \frac{Tp}{Tp + Fp}, \tag{24}$$

де P – точність,

Tp – кількість правильних позитивних прогнозів,

Fp – кількість помилкових позитивних прогнозів.

Повнота – міра того, як часто модель надає вірні рекомендації (25).

$$R = \frac{Tp}{Tp + Fn}, \tag{25}$$

де R – повнота,

Tp – кількість правильних позитивних прогнозів,

Fn – кількість помилкових негативних прогнозів.

F-score – це комбінована міра точності P та повноти моделі R. Вона розраховується за допомогою середнього гармонічного між P та R (26).

$$Fscore = \frac{2 \cdot P \cdot R}{P + R}, \tag{26}$$

де P – міра точності,

R – міра повноти.

MAE – середня абсолютна помилка, вимірює середню величину абсолютних помилок між передбаченими та фактичними значеннями, показує, наскільки в середньому передбачення моделі відхиляються від справжніх значень (27). Чим нижче значення MAE, тим краще модель справляється із завданням.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{27}$$

де n – загальна кількість спостережень,

y_i – фактичне значення для і-го спостереження,

$\hat{y}_i$ – передбачене значення для і-го спостереження.

MSE – середньоквадратична помилка, вимірює середню величину квадратів помилок між передбаченими та фактичними значеннями, показує, наскільки передбачення моделі відхиляються від справжніх значень, але з урахуванням того, що більші помилки отримують більшу вагу за рахунок піднесення до квадрата (28). Тобто, MSE сильніше штрафує великі помилки. Чим нижче значення MSE, тим краще модель справляється із завданням.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{28}$$

де n – загальна кількість спостережень,

y_i – справжнє значення для і-го спостереження,

$\hat{y}_i$ – передбачене значення для і-го спостереження.

Також буде здійснюватись порвівняння шввидкісті моделей, а саме: час навчання моделі, час прогнозування для нового прикладу.

Щоб оцінити використання моделей для задач, що можуть бути використані в рекомендаційних системах, було визначено три основні цілі:

– аналіз настроїв. Аналіз настроїв дозволяє глибше зрозуміти емоційний контекст, вбудований у створений користувачем текст, полегшуючи створення рекомендацій, які резонують на більш особистому рівні. Розпізнаючи емоційний стан користувача, рекомендації можна налаштувати відповідно до його поточних потреб, настрою та вподобань;

– формулювання рекомендацій на основі вподобань користувачів підвищує ефективність систем рекомендацій у різних областях. Аналізуючи створений користувачами контент або взаємодії та надаючи пропозиції, адаптовані до індивідуальних смаків, ці системи пропонують точніші та задовольняючі рекомендації, тим самим підвищуючи загальну задоволеність користувачів і взаємодію з платформою чи сервісом;

– знаходження подібності текстів. Знаходження подібності текстів може бути доцільним для рекомендаційних систем, оскільки дозволяє надавати точні та релевантні рекомендації, персоналізувати контент та глибше розуміти семантичний зміст текстів, що робить рекомендації більш змістовними та точними.

У рамках даного дослідження та поставленої задачі вищезазначені завдання формулюються наступним чином:

– аналіз настроїв у створеному користувачем вільному тексті (на кшталт постів соціальних мереж, коментарів);

– генерація рекомендацій щодо жанру фільму на основі атрибутів фільму, яким надає перевагу користувач;

– розрахунок подібності між парою текстів.

Для першого завдання використовувався набір даних, що містить твіти користувачів і мітки для емоційних станів: сум (0), радість (1), любов (2), гнів (3), страх (4), здивування (5) [19]. Навчальний набір містить 15969 унікальних значень (див. рис. 4.1).

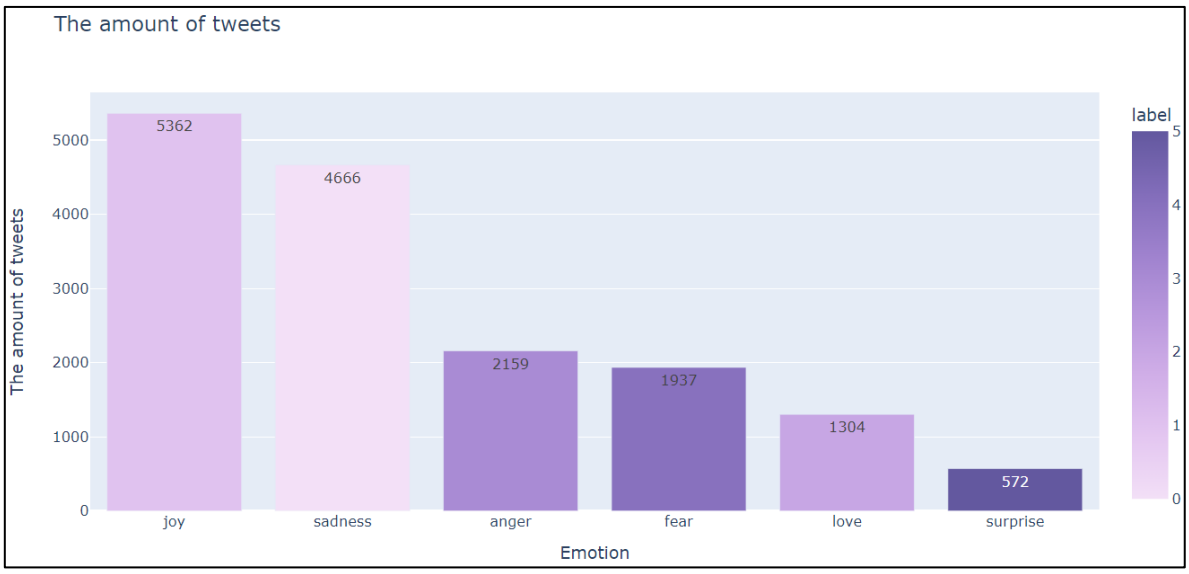

Рисунок 4.1 – Розподіл твітів між мітками (створено самостійно)

Гістограма на рисунку 4.1 візуалізує розподіл твітів за емоціями в наборі даних. Кожна смужка представляє певну категорію емоцій, а її висота відповідає кількості твітів, пов’язаних із цією емоцією.

Для другого завдання набір даних було створено за допомогою API The Movie Database (TMDb) [20]. Цей набір даних містить інформацію про фільми, отримані з кінцевих точок. Функція fetch_movies отримує дані про фільми на основі популярності з кількох сторінок, тоді як функція get_movie_genres отримує назви жанрів для певних ідентифікаторів жанрів. Отриманий набір даних складається з 20 000 рядків і містить такі стовпці:

– title – назва фільму;

– description – короткий огляд або синопсис фільму;

– genres – список жанрів, пов’язаних із фільмом: [жанр №1, жанр №2, … жанр №n].

Загалом у списку 19 жанрів, із різною кількістю жанрів для кожного фільму. Серед жанрів: бойовик, драма, комедія, трилер, пригоди, наукова фантастика, анімація, фентезі, жахи, сімейний, кримінальний, мелодрама, містика, історія, війна, музика, телефільм, вестерн і документальний фільм (див. рис. 4.2).

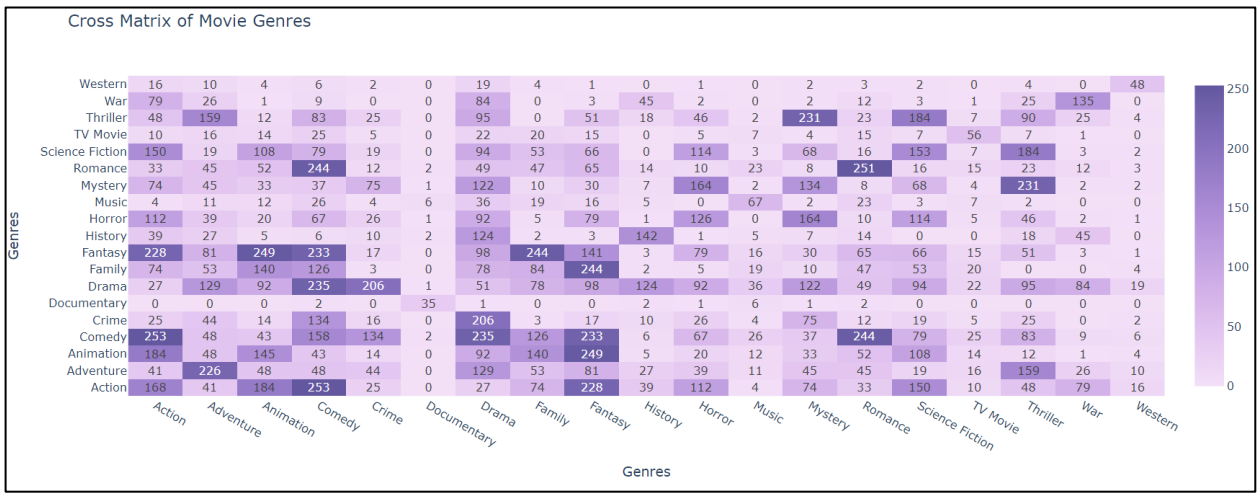

Рисунок 4.2 – Перехресна матриця жанрів фільмів (створено самостійно)

Перехресна матриця на рисунку 4.2 представляє частоту появи жанрів фільмів разом. Кожна клітинка в матриці вказує, скільки фільмів належать до відповідного жанру рядка та стовпця. Жанри перераховані по осях x і y, а значення в кожній клітинці вказує на кількість фільмів, які належать до обох жанрів.

Для останньої задачі використовуємо датасет, що знаходиться у відкритому доступі у huggingface [21]. Вигляд датасету наведено на рисунку 4.3.

```
df = pq.read_table("train-00000-of-00001.parquet").to_pandas()
df.head(10)
```

	sentence1	sentence2	similarity_score
0	A plane is taking off.	An air plane is taking off.	5.00
1	A man is playing a large flute.	A man is playing a flute.	3.80
2	A man is spreading shreded cheese on a pizza.	A man is spreading shredded cheese on an uncoo...	3.80
3	Three men are playing chess.	Two men are playing chess.	2.60
4	A man is playing the cello.	A man seated is playing the cello.	4.25
5	Some men are fighting.	Two men are fighting.	4.25
6	A man is smoking.	A man is skating.	0.50
7	The man is playing the piano.	The man is playing the guitar.	1.60
8	A man is playing on a guitar and singing.	A woman is playing an acoustic guitar and sing...	2.20
9	A person is throwing a cat on to the ceiling.	A person throws a cat on the ceiling.	5.00

Рисунок 4.3 – Датасет для подібності текстів (створено самостійно)

Датасет містить пару текстів англійською мовою та їх подібність в діапазоні від 0 до 5. Усього набір даних містить 5,749 тис. рядків.

4.2 Налаштування програмного середовища

Процес включає попередню обробку даних, навчання моделі, тестування та оцінку моделі. Для проведення експериментів та оцінки моделей було використано мову програмування Python.

Для моделей BERT, GPT та XLNet було використано бібліотеки transformers та torch. Для моделі FastText – бібліотеку fasttext. Для розрахунку метрик для оцінки було використано бібілотеку scikit-learn.

Задача класифікації твітів за настроєм. Процес для моделей GPT-2, BERT та XLNet дуже схожий.

Першим кроком підключаємо необхідні бібліотеки. Використовуємо бібліотеки pandas для роботи з CSV файлами, torch для роботи з тензорами та моделями, transformers для завантаження та використання моделей і

токенізаторів BERT, GPT-2 та XLNet, а також бібліотеки для попередньої обробки тексту (nltk для стоп-слів та лематизації).

Наступним кроком завантажуємо тренувальний та тестовий набори даних за допомогою pandas та перетворюємо тексти та мітки в списки для подальшої обробки.

Для попередньої обробки тексту було створено клас для розбиття слів та розділових знаків на токени та функцію preprocess_text, що перетворює текст на нижній регістр, видаляє непотрібні символи, токенізує текст, видаляє стоп-слова та виконує лематизацію токенів. Код методу наведено нижче.

```
class CustomTokenizer:
    def __init__(self):
        self.pattern = re.compile(r"\w+|[^\w\s]")

    def tokenize(self, text):
        tokens = self.pattern.findall(text)
        return tokens

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r"[^a-zA-Z0-9]", " ", text)
    tokenizer = CustomTokenizer()
    tokens = tokenizer.tokenize(text)
    tokens = [token for token in tokens if token
                not in stop_words]
    tokens = [lemmatizer.lemmatize(token) for token
                in tokens]
    return " ".join(tokens)
```

Лематизація токенів – це процес зведення словоформ до їхніх базових або початкових форм (лем), враховуючи його частину мови.

Наступним кроком звантажуємо попередньо натреновані токенізатори та моделі для класифікації послідовностей з бібліотеки transformers:

– для моделі BERT: BertTokenizer, BertForSequenceClassification, модель ‘bert-base-uncased’;
– для XLNet: XLNetTokenizer, XLNetForSequenceClassification, модель ‘xlnet-base-cased’;

– для моделі GPT-2: GPT2Tokenizer, GPT2ForSequenceClassification, GPT2Config, модель ‘gpt2’.

Приклад завантаження токенізатора та моделі на основі BERT наведено на рисунку 4.4.

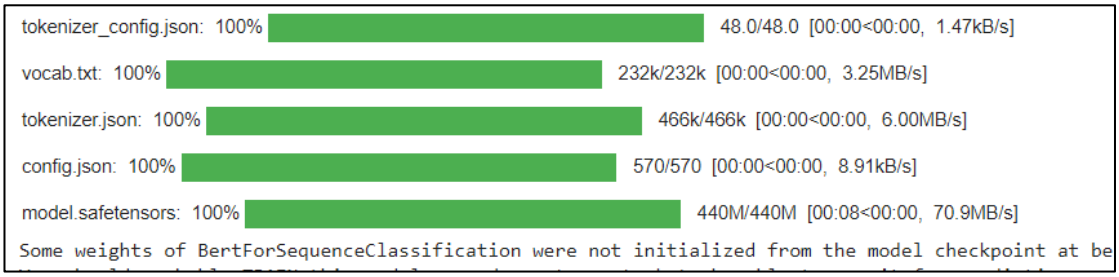

Рисунок 4.4 – Завантаження моделі та токенізатора (створено самостійно)

Наступним кроком використовуємо токенізатор для перетворення текстових даних у числові послідовності (input_ids) та маски уваги (attention_mask), які використовуються моделями, також обмежуємо довжину послідовностей до 128 токенів. Код на прикладі моделі BERT наведено нижче.

```
train_encodings = tokenizer(train_texts,
                truncation=True, padding=True, max_length=128)
test_encodings = tokenizer(test_texts,
                truncation=True, padding=True, max_length=128)
```

Наступним кроком необхідно підготувати дані для подачі їх до моделі. Перетворюємо мітки (лейбли) в тензори torch – багатовимірні масиви. Створюємо датасети TensorDataset для тренувальних та тестових даних, які включають input_ids, attention_mask та мітки. Створюємо DataLoader для тренувальних та тестових даних для ітерації по міні-батчах під час тренування та тестування.

Далі відбувається навчання моделей на основі тренувальних даних. В якості оптимізатора використовуємо AdamW оптимізатор з бібліотеки

transformers. Встановлюємо модель у режим тренування model.train(). Код донавчання моделі наведено рисунку 4.5.

```
optimizer = AdamW(model.parameters(), lr=2e-5)


model.train()
for epoch in range(7):
    for batch in train_loader:
        optimizer.zero_grad()
        input_ids, attention_mask, labels = batch
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
```

Рисунок 4.5 – Код для навчання моделі на прикладі BERT (створено самостійно)

Тут для кожної епохи ітеративно обробляємо міні-батчі тренувальних даних. Обнуляємо градієнти, передаємо вхідні дані через модель, обчислюємо втрати, виконуємо зворотне розповсюдження для обчислення градієнтів та оновлюємо параметри моделі за допомогою оптимізатора.

Останнім кроком виконуємо оцінку моделі на тестових даних та розрахунок метрик. Для цього встановлюємо модель у режим оцінки model.eval(). Відключаємо обчислення градієнтів для тестових даних за допомогою torch.no_grad(). Ітеративно обробляємо міні-батчі тестових даних, передаємо їх через модель та збираємо передбачення. Для розрахунку метрик точності, повноти та F1-міри використовуємо метод classification_report з бібліотеки sklearn. Код для оцінки моделі наведено на рисунку 4.6.

```
model.eval()
predictions = []
true_labels = []
with torch.no_grad():
    for batch in test_loader:
        input_ids, attention_mask, labels = batch
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        predictions.extend(torch.argmax(logits, axis=1).cpu().numpy())
        true_labels.extend(labels.cpu().numpy())

report = classification_report(true_labels, predictions)
print(report)
```

Рисунок 4.6 – Оцінка моделі та розрахунок метрик (створено самостійно)

Такий самий процес відбувається для XLNet та GPT-2 з використанням відповідних токенізаторів та моделей, але оскільки модель GPT-2 використовує лише токени з лівої сторони від поточного, необхідно додати деякі налаштування за допомогою GPT2Config, а саме додаємо лівий падінг, що використовується для вирівнювання послідовностей токенів у батчі так, щоб всі послідовності мали однакову довжину, а також визначаємо PAD Token як EOS Token (токен кінця послідовності). Код наведено нижче.

```
model_config = GPT2Config.from_pretrained('gpt2', num_labels=6)
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
tokenizer.padding_side = "left"
tokenizer.pad_token = tokenizer.eos_token
model = GPT2ForSequenceClassification.from_pretrained('gpt2',
     config=model_config)
model.resize_token_embeddings(len(tokenizer))
model.config.pad_token_id = model.config.eos_token_id
```

Для моделі FastText алгоритм навчання та класифікації відрізняється і є значно легшим. На вхід до моделі подаються .txt файли, що повинні містити мітку, що починається з позначки __label__, яка сигналізує моделі про приналежність тексту до певного класу, тому, до кожної мітки додається префікс __label__. Код для створення файлів відповідного формату наведено нижче.

```
def process_data(df, label_prefix='__label__'):
    df['text'] = df['text'].str.replace('\n', ' ',
           regex=True).str.replace('\t', ' ', regex=True)
    df['label'] = label_prefix + df['label'].astype(str)
    return df
train_data[['label', 'text']].to_csv('train.txt',
     index=False, header=False, sep=' ')
test_data[['label', 'text']].to_csv('test.txt',
     index=False, header=False, sep=' ')
valid_data[['label', 'text']].to_csv('valid.txt',
     index=False, header=False, sep=' ')
```

Кожен рядок файлу містить мітку, яка починається з позначки __label__ і текст, які розділені пробілом.

Для тренування моделі використовується метод train_supervised з автоматичним налаштуванням на валідаційних даних. Код наведено нижче.

```
model = train_supervised(input="train.txt",
      autotuneValidationFile="valid.txt")
model.save_model("trained_model.bin")
```

Для оцінки моделі завантажуємо тестові дані з файлу та розділяємо їх на тексти і мітки та використовуємо натреновану модель для прогнозування міток для тестових даних.

```
predicted_labels = []
for text in test_texts:
    labels, _ = model.predict(text, k=1)  # k=1 to return top label
    predicted_labels.append([label.replace('__label__', '') \
           for label in labels])
```

Для обчислення метрик точності, повноти та F1-міри також використовуємо функцію classification_report.

Процес навчання моделей для задачі класифікації фільмів за жанрами майже той самий, але є декілька додаткових моментів.

Спочатку необхідно зібрати всі унікальні жанри з датасету про фільми та конвертувати жанри у формат one-hot encoding. Для цього створено функцію, яка перетворює жанр на бітове значення 1 або 0. Код для обробки жанрів наведено нижче.

```
    all_genres = set(genre for genres in movies_df['genres'] for genre
in genres)
    genre_list = sorted(list(all_genres))
    def encode_genres(genres, genre_list):
        return [1 if genre in genres else 0 for genre in genre_list]
```

```
    train_labels_encoded = np.array([encode_genres(genres, genre_list) for genres in train_df['genres']])
    test_labels_encoded = np.array([encode_genres(genres, genre_list) for genres in test_df['genres']])
```

Також для навчання моделей XLNet, BERT та GPT-2 в якості функції втрат встановлюємо loss_fn = torch.nn.BCEWithLogitsLoss(), яка підходить для багатокласової класифікації з декількома мітками та працює на основі застосування сигмоїдної активації та обчислення бінарної крос-ентропії.

Для задачі знаходження подібностей текстів використовуються лише попередньо навчені моделі, а саме їх ембедінги, а для розрахунку подібностей використано косинусну подібність (29).

$$\frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{29}$$

де A_i,B_i – координати вектору A і B відповідно,

$\sum_{i=1}^{n} A_i B_i$ – скалярний добуток двох векторів, який вимірює суму попарних добутків їх компонент,

$\sqrt{\sum_{i=1}^{n} A_i^2}$, $\sqrt{\sum_{i=1}^{n} B_i^2}$ – добуток норм векторів, що нормалізує скалярний добуток на довжини вектора в діапазоні від -1 до 1.

Оскільки в наборі даних діпазон схожості в інтервалі від 0 до 5, необхідно їх перетворити в діапазон від -1 до 1 для уніфікації з косинусною подібністю, код для цього наведено нижче.

```
    old_min, old_max = 0, 5
    new_min, new_max = -1, 1
    transformed_scores = [((score - old_min) * (new_max - new_min) / (old_max - old_min)) + new_min for score in true_scores]
```

Далі токенізуємо тексти, перетворюємо їх в тензори, отримуємо вихідні ембеддинги з моделей та обчислюємо косинусну схожість між ембеддингами двох текстів.

```
for idx, row in df.iterrows():
    inputs = tokenizer(row['sentence1'],
           return_tensors="pt", padding=True, truncation=True)
    inputs2 = tokenizer(row['sentence2'],
           return_tensors="pt", padding=True, truncation=True)

    with torch.no_grad():
        outputs = model(**inputs)
        outputs2 = model(**inputs2)

    similarity =
           cosine_similarity(outputs.last_hidden_state.mean(dim=1),
           outputs2.last_hidden_state.mean(dim=1))
    similarities.append(similarity[0][0])
```

Далі за допомогою бібліотеки sklearn обчислюємо метрики помилок MSE і MAE між передбачуваними і фактичними значеннями.

Для моделі Fasttext процес отримання векторних представлень відрізняється від попередніх моделей-трансформерів. Для кожного слова в реченні окремо отримується векторне представлення і потім обчислюється середнє значення векторів для всіх слів у реченні для отримання представлення усього речення. Для задачі знаходження подібностей було використано Fasttext з бібліотеки Gensim та пренавчену модель fasttext-wiki-news-subwords-300.

```
for idx, row in df.iterrows():
    embedding1 = np.mean([fasttext_model[word] for word in
     row['sentence1'].split() if word in fasttext_model], axis=0)
    embedding2 = np.mean([fasttext_model[word] for word in
     row['sentence2'].split() if word in fasttext_model], axis=0)

    similarity = cosine_similarity([embedding1], [embedding2])
    similarities.append(similarity[0][0])
```

4.3 Проведення експериментів

На основі проведеного теоретичного дослідження для порівняльного аналізу були проведені експерименти на моделях GPT-2, BERT, XLNet і FastText.

BERT середній час виконання навчання: ≈ 12 годин. Результати, отримані під час тестування аналізу настрою твітів BERT, наведено в таблиці 4.1.

Таблиця 4.1 – Результати аналізу настрою BERT (виконана самостійно)

Experiment	Точність	Повнота	F-score
1	0.77	0.83	0,80
2	0.82	0.86	0,83
3	0.84	0.86	0,85
4	0.79	0.84	0,80
5	0.81	0.85	0,82
Average	0,806	0,848	0,82

Результати вказують на продуктивність моделі аналізу настроїв. Середня точність становить 0,80, що свідчить про те, що з усіх випадків, прогнозованих як позитивні, 80% були правильно класифіковані. Із середнім показником повноти 0,848 модель правильно визначила 84,8% усіх фактичних позитивних випадків. Із середнім показником F1 0,82 модель досягла балансу між точністю та запам’ятовуванням, що свідчить про загальну ефективність аналізу настрою.

Результати задачі класифікації жанрів BERT наведені в таблиці 4.2.

Таблиця 4.2 – Результат рекомендацій жанрів BERT (виконана самостійно)

Експеримент	Точність	Повнота	F-score
1	0.93	0.81	0.86
2	0,89	0,87	0,88

Продовження таблиці 4.2

Експеримент	Точність	Повнота	F-score
3	0.85	0.88	0,86
4	0.82	0.86	0,83
5	0.84	0.88	0,86
Середнє	0.866	0,86	0,858

Результати вказують на ефективність моделі рекомендації жанрів. Із середнім показником F1 0,858 модель досягла хорошого балансу між точністю та повнотою, що вказує на загальну високу ефективність виконання цього завдання.

Для GPT-2 середній час виконання навчання: ≈ 17 годин. Результати задачі класифікації настрою GPT наведені в таблиці 4.3.

Таблиця 4.3 – Результати аналізу настрою GPT (виконана самостійно)

Експеримент	Точність	Повнота	F-score
1	0.70	0.75	0.72
2	0.73	0.77	0.75
3	0.75	0.77	0.76
4	0.71	0.76	0.72
5	0.73	0.77	0.74
Середнє	0,724	0,764	0,738

Підводячи підсумок, ми бачимо, що результати класифікації настроїв демонструють задовільну продуктивність моделі GPT щодо точної класифікації емоцій. У таблиці 4.4 ми можемо спостерігати результати того, як модель виконала завдання рекомендації жанру.

Таблиця 4.4 – Результати рекомендацій жанрів GPT (виконана самостійно)

Експеримент	Точність	Повнота	F-score
1	0.85	0.82	0.83
2	0.81	0.83	0.82
3	0.78	0.82	0.79
4	0.87	0.75	0.80
5	0.80	0.82	0.81
Середнє	0,82	0,80	0,81

Середній час виконання навчання XLNet ≈ 13,5 годин. Результати аналізу настроїв XLNet (S) і рекомендації жанрів (R) наведені в таблиці 4.5.

Таблиця 4.5 – Результати XLNet (таблиця виконана самостійно)

Задача	Точність	Повнота	F-score
S	0.84	0.80	0.82
R	0.83	0.87	0.849

XLNet досяг точності 0,84 і повноти 0,80 з F-оцінкою 0,82 для класифікації настроїв, тоді як для рекомендації жанрів модель досягла точності 0,83, 0,87 повноти і 0,849 F-score.

Середній час виконання навчання FastText: ≈ 4,5 години. Результати аналізу настрою (S) і рекомендації жанрів (R) FastText наведено в таблиці 4.6.

Таблиця 4.6 – Результати FastText (таблиця виконана самостійно)

Задача	Точність	Повнота	F-score
S	0.84	0.80	0.85
R	0.66	0.68	0.66

FastText отримав високу F-міру 0,85 для класифікації настроїв, тоді як для рекомендації жанру він результат становить лише 0,66.

Результати моделей, а саме MSE та MAE помилок для знаходження подібностей текстів наведено в таблиці 4.7.

Таблиця 4.7 – Результати знаходження подібностей (таблиця виконана самостійно)

Метрика	BERT	GPT-2	XLNet	Fasttext
MSE	0.81	1.488	1.447	1.25
MAE	0.71	1.037	1.0193	0.93

Бачимо, що моделі BERT та Fasttext мають найнижчі помилки – 0.81 і 0.71 та 1.25 та 0.93 відповідно, в той час, як GPT-2 і BERT показують гірший результат.

5 ОЦІНКА ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Щоб зробити висновки щодо використання моделей для текстових завдань у системах рекомендацій, було створено три таблиці порівняння результатів. Як метрики для порівняння були обрані F-міра і час навчання для задач класифікацій та MSE і MAE для задачі знаходження подібності. Таблиця 5.1 представляє результати для проблеми аналізу настроїв, а таблиця 5.2 – жанрові рекомендації.

Таблиця 5.1 – Порівняння для задачі аналізу настроїв (таблиця виконана самостійно)

Модель	F-Score показник	Час виконання (год.)
BERT	0.858	12
GPT-2	0.738	17
XLNet	0.82	13,5
FastText	0,85	4,5

Хоча BERT демонструє конкурентоздатну продуктивність у аналізі настроїв, FastText перевершує моделі як у точності, так і з боку ефективності. Його здатність досягати високого значення F-міри при мінімальних обчислювальних ресурсах робить FastText привабливим вибором для завдань аналізу настроїв, особливо в сценаріях, де є часові та ресурсні обмеження.

Таблиця 5.2 – Порівняння для рекомендації жанрів (виконана самостійно)

Модель	F-Score показник	Час виконання (год.)
BERT	0.82	11,5
GPT-2	0.81	18
XLNet	0.849	16
FastText	0,66	6

Щоб знайти модель, яка найкраще підходить, ми будемо використовувати лінійну адитивну згортку з ваговими коефіцієнтами за формулою (23) для розрахунку коефіцієнта корисності, оскільки певні критерії мають більший вплив на ефективність і вибір моделі.

Деякі критерії мають більший внесок і важливіші за інші, тому необхідно ввести вагові коефіцієнти. Використовуючи метод ранжирування, ми оцінюємо важливість критеріїв у такому порядку: найважливішим критерієм є F-Score, його ранг ми встановили як 5. З іншого боку, час виконання також важливий, тому його ранг дорівнює 3. Сума рангу = 8. Обчислення корисності моделей з використанням лінійної адитивної згортки показано на рисунку 5.1.

	F-score	Time saving (h)	K
BERT	0,82	6,5	**0,275337522**
GPT	0,81	0	0,161277477
XLNet	0,849	2	0,203525447
FastText	0,66	6	0,234859553
βj	0,625	0,25	
αj	0,318572794	0,068965517	

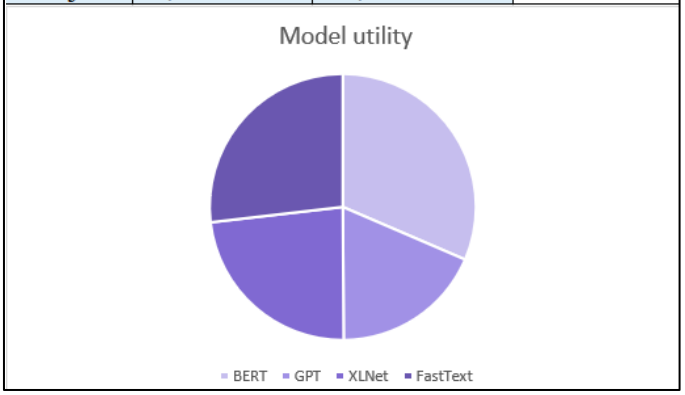

Рисунок 5.1 – Корисність моделей для задачі жанрів (створено самостійно)

На підставі коефіцієнтів корисності, розрахованих за допомогою адитивної лінійної згортки, модель BERT виявляється найбільш пасуючим вибором для рекомендаційних рішень для багатокласової класифікації серед розглянутих моделей.

У таблиці 5.3 наведені порівняння метрик MAE та MSE для розрахунку подібностей текстів.

Таблиця 5.3 – Порівняння для подібності текстів (виконана самостійно)

Модель	MSE	MAE
BERT	0.81	0.71
GPT-2	1.48	1.04
XLNet	1.45	1.02
FastText	1.25	0.93

BERT показує найкращі результати як за MSE, так і за MAE. Це означає, що ця модель є найточнішою і найменш схильною до великих помилок при знаходженні подібностей текстів. Fasttext займає друге місце за обома метриками, що вказує на її досить високу точність. XLNet і GPT-2 мають схожі результати, але вони менш точні порівняно з BERT і Fasttext. Таким чином, для завдання знаходження подібностей текстів найкращим вибором є BERT.

У висновку, інтеграція комбінації моделей BERT і FastText пропонує комплексний підхід до систем рекомендацій, що дозволяє їм надавати персоналізовані пропозиції користувачам, враховуючи емоційну складову, тим самим підвищуючи загальний досвід та задоволення користувачів.

6 ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

У науковій діяльності отримані моделі дозволяють досліджувати та порівнювати різні підходи до обробки природної мови в рекомендаційних системах, оптимізувати моделі для покращення їх точності і швидкості, а також автоматично класифікувати текстові дані, виявляти тематичні групи і аналізувати їх, що може бути корисно:

- в соціальних науках для аналізу текстів соціальних мереж і новин;
- біомедичних дослідженнях для аналізу медичних записів і наукових статей;
- у гуманітарних науках для аналізу літературних текстів для виявлення стилістичних ознак;
- в освітніх дослідженнях для автоматичного оцінювання студентських робіт і надання персоналізованих рекомендацій для покращення навчання.

У практичній діяльності моделі NLP можуть значно покращити персоналізовані рекомендаційні системи.

В електронній комерції вони можуть використовуватися для покращення рекомендацій продуктів на основі поведінки користувачів і аналізу відгуків.

На контентних платформах, таких як Netflix або Spotify, моделі можуть допомогти у покращенні рекомендацій фільмів, книг, музики та іншого контенту на основі текстових описів і уподобань користувачів.

Крім того, ці моделі можуть автоматизувати обробку текстів, маршрутизуючи документи, аналізуючи текстові відгуки клієнтів для покращення обслуговування, і вдосконалюючи пошукові системи для підвищення релевантності результатів.

На основі проведених досліджень, аналізу результатів та висновків було обрано найкращі моделі для застосування їх в задачах надання рекомендацій та було прийнято рішення розробити програмну систему для рекомендацій фільмів

в залежності від настрою та змісту поста користувача з використанням цих моделей.

Основний функціонал програмної системи “inTheMood” наведено нижче:

– надання можливості отримувати рекомендації за настроєм та змістом тексту, що був введений користувачем (як пройшов день, думки користувача і тд.). Для кожного поста аналізується його настрій та надаються рекомендації щодо фільмів для перегляду (6 фільмів);
– можливість перегляду історії своїх публікацій, їх настрою та отриманих рекомендацій.
– надання можливості перегляду та взаємодії з постами та рекомендаціями інших користувачів.

6.1 Вимоги до програмної системи

Програмна система складається з 2 частин: серверна частина та клієнтська Вимоги до кожної частини сформовано нижче.

Вимоги до серверної частини:

– підключення до СУБД;
– клієнт-серверна взаємодія за використанням DRF (REST технологія для Djnago);
– отримання GET, POST запитів з клієнтської частини, їх обробка та відправка результату;
– забезпечення реєстрації та авторизації;
– захист даних;
– алгоритм надання рекомендацій, що складається з двох етапів: аналіз настрою поста користувача та надання рекомендацій на основі схожого контексту тексту;
– операції CRUD;
– обробка отримання та публікації постів;
– ендпоїнти для надсилання запитів з клієнтської частини.

Вимоги до клієнтської частини:

– форма для введення тексту для отримання рекомендацій фільмів для всіх користувачів;
– вікно надання рекомендацій;
– форма для реєстрації та авторизації;
– сторінка з постами та рекомендаціями інших користувачів;
– сторінка для перегляду історії власних публікацій та рекомендацій;
– форма для публікації власного поста та отримання рекомендацій для авторизованих користувачів;
– відображення дати постів;
– можливість поставити вподобання для авторизованих користувачів;
– перегляд і редагування профілю.

6.2 Опис технологій для серверної частини

Для розробки серверної частини використано мову програмування Python та фреймворку Django. Для забезпечення клієнт-серверної взаємодії використано Django REST Framework. В якості структури зберігання даних було обрано СУБД PostgreSQL.

База даних складається з 5 сутностей: “Movie”, “User”, “Post”, “Like”, “Recommendations”. Сутність User представляє користувача та містить основну інформацію про нього, сутність Movie представляє фільм для надання рекомендацій та має атрибути такі, як назва, жанр, опис. Сутність Post – це пост, опублікований користувачем для отримання рекомендацій, основними атрибутами є текст та настрій. Сутність Like таблиця, що представляє собою вподобання (лайк), зроблене користувачем на певну публікацію. Сутність Recommendations – проміжна сутність для збереження рекомендацій для певної публікації. Структура бази даних, сутності та взаємодії між ними представлена на ER діаграмі, що зображена на рисунку 6.1.

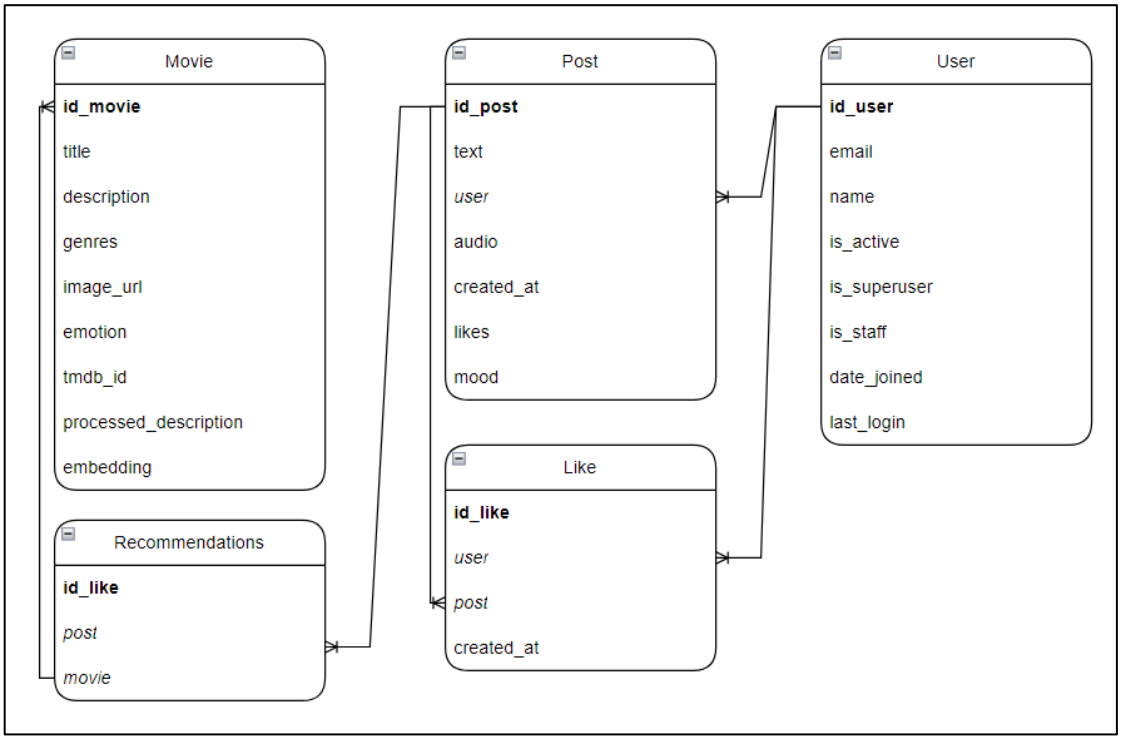

Рисунок 6.1 – ER діаграма програмної системи (створено самостійно)

Отже, було встановлено наступні відношення між сутностями:

– User має відношення один-до-багатьох з Post, бо один користувач може мати багато публікацій;
– CustomUser має відношення один-до-багатьох з Like, один користувач може мати багато вподобань;
– Post має відношення один-до-багатьох з Like, одна публікація може мати багато вподобань;
– Post має відношення багато-до-багатьох з Movie через проміжну таблицю Recommendations.

Оскільки найкращі результати для задач надання рекомендацій на основі текстів показали BERT і Fasttext, використаємо їх для програмної системи надання рекомендацій фільмів за настроєм та змістом публікації. Алгоритм надання рекомендацій наступний: спочатку за допомогою моделі Fasttext аналізуємо настрій публікації користувача на основі введеного тексту, далі серед фільмів зі схожим настроєм шукаємо подібні за змістом фільми, за допомогою моделі BERT отримуємо ембединг публікації користувача, та

розраховуємо подібність між ембедингами описів фільмів та публікації за допомогою косинусної подібності (29). На виході людина отримує рекомендації фільмів в розмірі 6 штук. Код методу для класифікації постів за настроями наведено нижче.

```
def classify_text(text):
    global model
    if model is None:
        try:
            model = load_model("trained_model.bin")
        except FileNotFoundError:
            return "Model is not trained yet!"

    labels, probabilities = model.predict(text)
    labels = [label.replace('__label__', '') for label in labels]
    return int(labels[0]), probabilities[0]
```

Оскільки BERT показав найкращі результати для знаходження подібностей тексту та існує ще краще натренована бібліотека BERT для цієї задачі, ніж transformers, а саме sentence_transformers, будемо використовувати її. Основна відмінність – це використання спеціально підготовлених для задач семантичного пошуку і порівняння текстів та замість використання тільки ембеддингу [CLS]-токена, SentenceTransformers використовує mean_pooling – усереднення всіх токенів для отримання ембеддингів на рівні речень. Код методу рекомендації фільмів наведено нижче.

```
def recommend_movies_for_post(user_post, mood, top_n=6):
    movies = Movie.objects.filter(emotion=mood)
    movie_embeddings = [movie.embedding for movie in movies]
    model = SentenceTransformer('bert-base-nli-mean-tokens-model')
    user_post_embedding = model.encode(preprocess_text(user_post))

    cosine_similarities = cosine_similarity([user_post_embedding],
                          movie_embeddings)
    movie_similarities = list(zip(movies, cosine_similarities[0]))
    movie_similarities.sort(key=lambda x: x[1], reverse=True)
    print(movie_similarities)
    top_movies = movie_similarities[:top_n]
    recommended_movies = [movie for movie, _ in top_movies]

    return recommended_movies
```

Метод повертає об’єкти фільмів, які мають найбільшу подібність з текстом публікації користувача.

Для забезпечення клієнт-серверної взаємодії розроблені представлення, що наслідують базовий клас APIView (DRF). Приклад коду представлення для обробки надання рекомендацій для усіх користувачів, незалежно від автентифікації, наведено нижче.

```
class GetMoodNoPost(APIView):
     permission_classes = (permissions.AllowAny, )

     def post(self, request):
          text = request.data["text"]
          predict_label, accuracy = classify_text(text)

          recommended_movies = recommend_movies_for_post(text,
                               predict_label)
           recommended_movies_data =
                               MovieSerializer(recommended_movies,
                          many=True).data
           response_data = {
                "label": predict_label,
                 "accuracy": accuracy,
                 "recommended_movies": recommended_movies_data
          }

          return Response(response_data,
                          status=status.HTTP_200_OK)
```

В методі отримуємо текст публікації користувача, виконуємо класифікацію за емоцією, передаємо емоцію до методу знаходження подібності для рекомендації фільмів, отримуємо результат та надсилаємо відповідь до клієнта.

6.3 Розробка клієнтської частини

Для розробки клієнтської частини було обрано мову програмування JavaScript та бібліотеку React.js, для швидкої збірки та запуску застосунку на реакті використано інструмент для побудови веб-застосунків Vite.

Для взаємодії з сервером використовуємо бібліотеку axios, що дозволяє відправляти асинхронні запити до серверів. Код налаштування axios для зручного виконання HTTP-запитів до сервера наведено нижче.

```
import axios from 'axios';
const token = localStorage.getItem('token');
console.log(token)

if (token !== null) {
    axios.defaults.headers.common['Authorization'] =
                                    `Token ${token}`;
}else{
    console.log("no token")
}

axios.defaults.xsrfCookieName = 'csrftoken';
axios.defaults.xsrfHeaderName = 'X-CSRFToken';
axios.defaults.withCredentials = true;

const server = axios.create({
    baseURL: "http://127.0.0.1:8000",
});

export default server;
```

Він автоматично додає токен авторизації до всіх запитів, якщо такий токен збережено в localStorage, та налаштовує CSRF-захист.

Було створено наступні основні компоненти: для авторизації та реєстрації, компонент для головної сторінки надання рекомендації для всіх користувачів, незалежно від їх автентифікації, компонент для перегляду публікацій інших користувачів, компонент для отримання історії своїх публікацій та рекомендацій та компонент для публікації своїх постів та отримання рекомендацій для авторизованих користувачів.

На рисунку 6.2 представлено головну сторінку веб-застосунку.

Рисунок 6.2 – Головна сторінка рекомендаційної системи (створено самостійно)

Тут бачимо блок для введення своїх думок, чи будь-якого тексту, щоб отримати на основі цього рекомендації фільмів. Вигляд форми з введеним текстом наведено на рисунку 6.3.

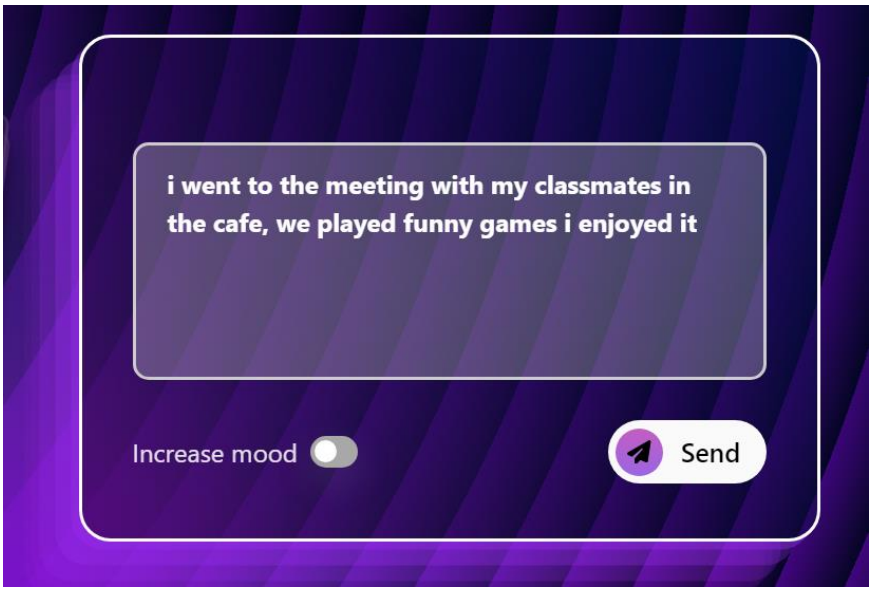

Рисунок 6.3 – Головна сторінка рекомендаційної системи (створено самостійно)

Після того, як користувач натискає кнопку Send, він отримує рекомендації фільмів, що показано на рисунку 6.4.

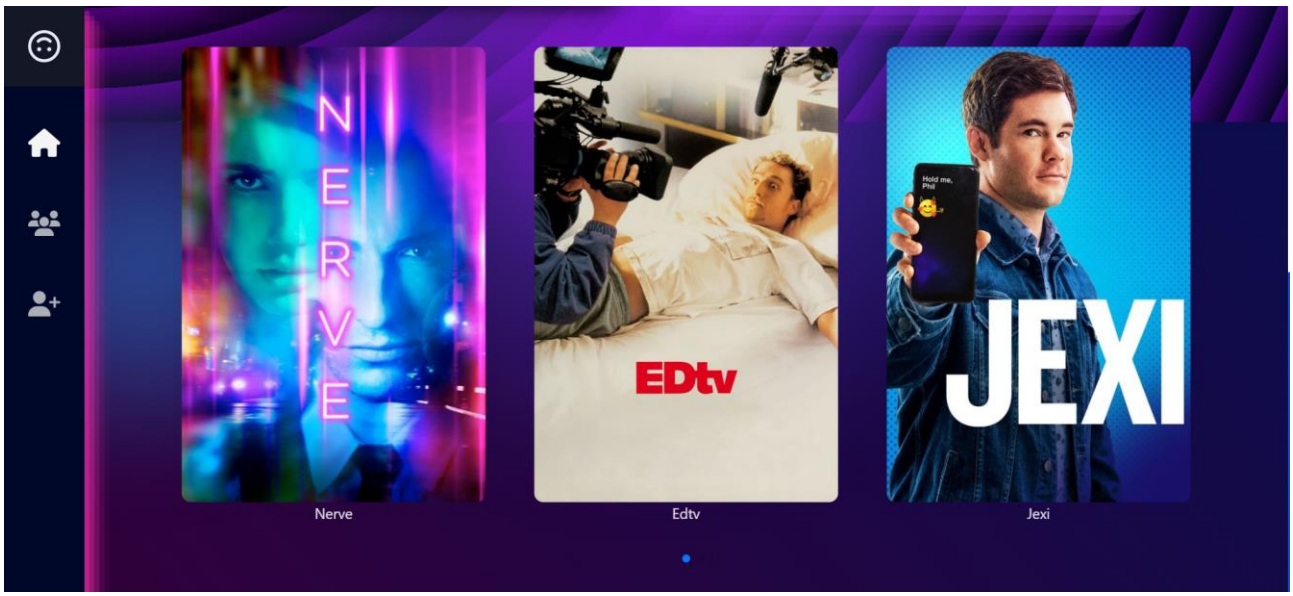

Рисунок 6.4 – Отримання рекомендацій фільмів (створено самостійно)

Для реєстрації та авторизації було створено дві форми, що наведені на рисунку 6.5.

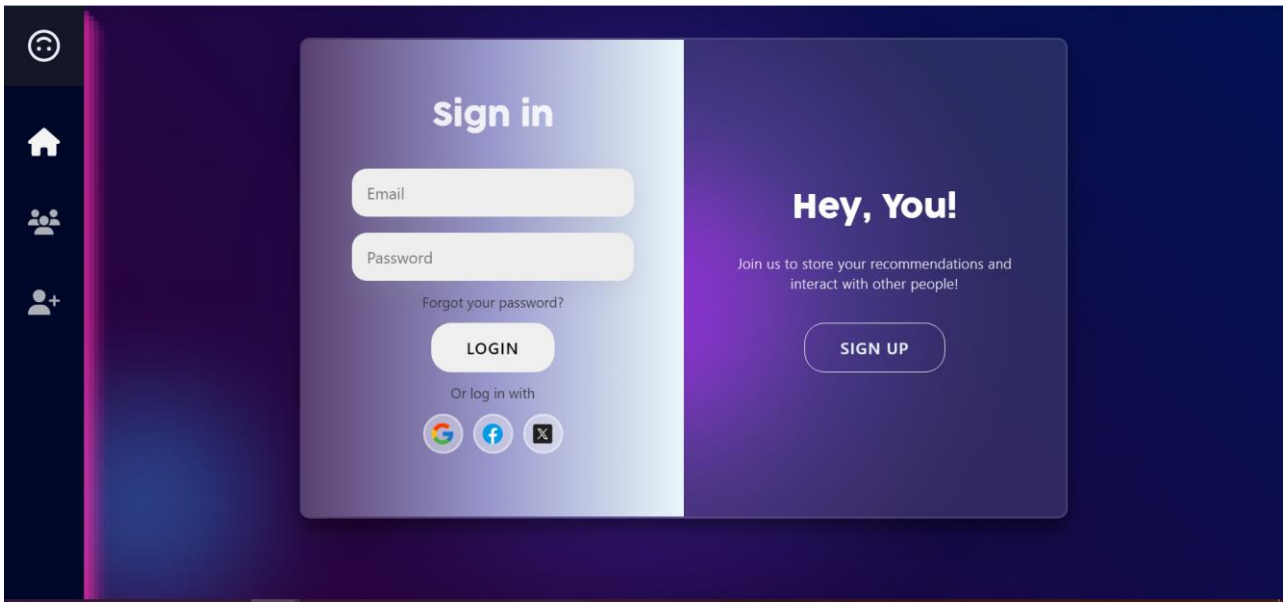

Рисунок 6.5 – Форма авторизації (створено самостійно)

Після авторизації користувач має доступ до створення та збереження публікацій та рекомендацій фільмів до них, що представлено на рисунку 6.6.

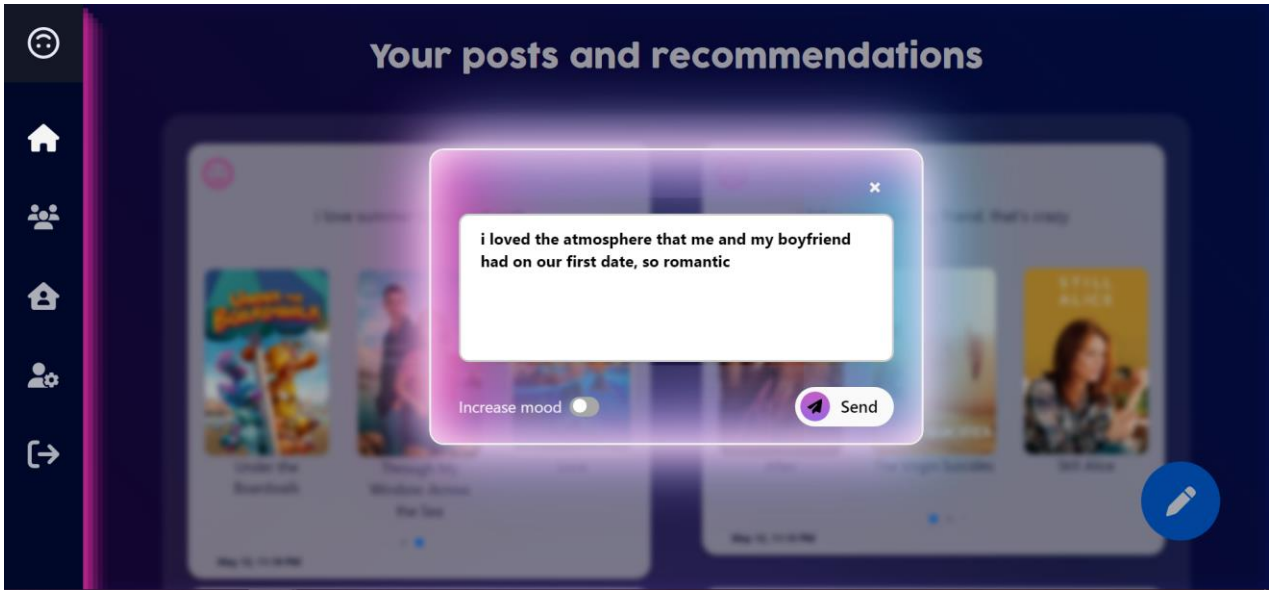

Рисунок 6.6 – Форма створення публікації (створено самостійно)

Усі користувачі мають доступ до перегляду публікацій інших користувачів, проте можливість поставити вподобання є лише у авторизованих. Сторінку постів наведено на рисунку 6.7.

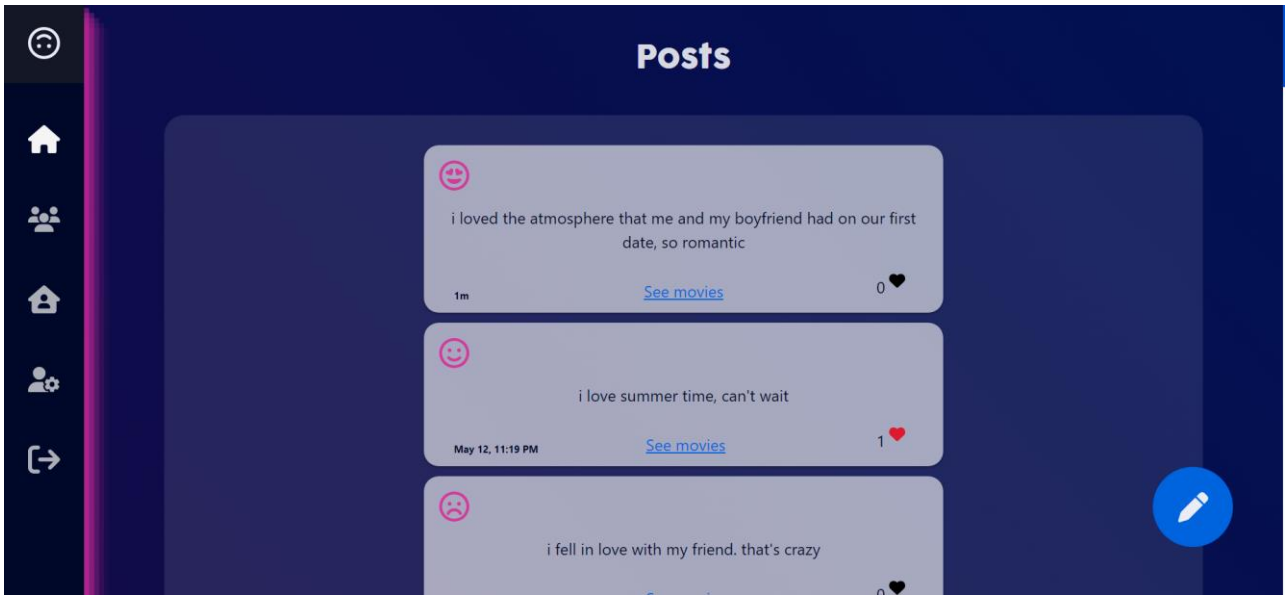

Рисунок 6.7 – Сторінка публікацій користувачів (створено самостійно)

Тут можна переглянути рекомендації фільмів до публікацій інших користувачів та знайти щось за спільними інтересами (див. рис. 6.8.).

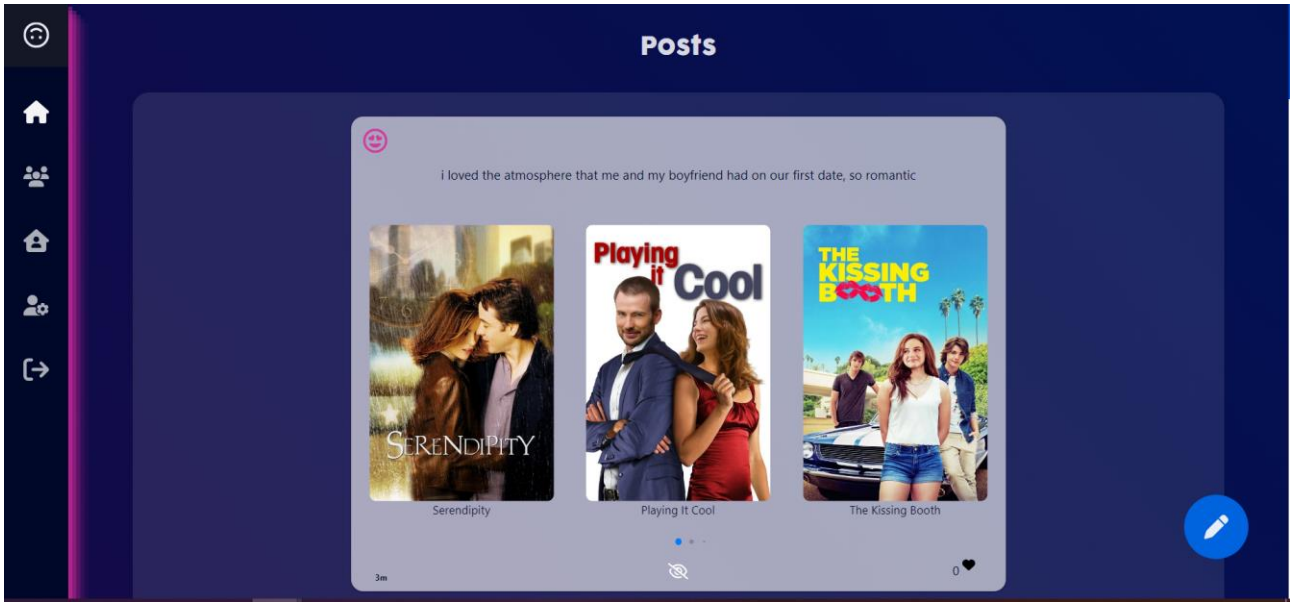

Рисунок 6.8 – Перегляд рекомендацій користувачів (створено самостійно)

Також у авторизованих користувачів є доступ до переглядів історії власних публікацій та рекомендацій до них, що наведено на рисунку 6.9.

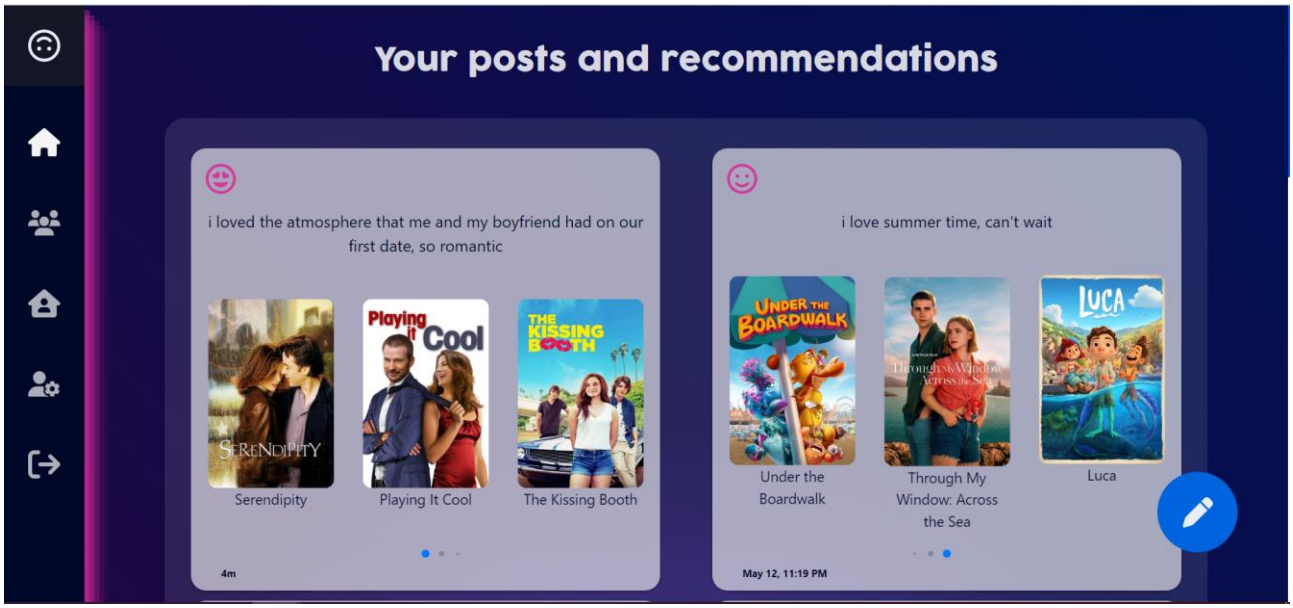

Рисунок 6.9 – Сторінка історії публікацій (створено самостійно)

Також зроблено можливість виходу з власного акаунту та перегляд профілю.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи магістра було проведено дослідження методів обробки NLP для рекомендаційних систем на основі тексту, було виявлено моделі, що мають найбільший коефіцієнт корисності за важливими для надання рекомендацій критеріями та виконують поставлені задачі найефективніше.

В ході дослідження було проведено аналіз предметної області та літератури, виявлено існуючі проблеми та описано актуалізацію рішень, описано постановку задач для проведення дослідження, проведено аналіз існуючих методів NLP, їх особливостей архітектури, їх переваги та недоліки. В результаті аналізу моделей, було виділено шість, для яких проводилось порівняння та розрахунок критерію корисності за допомогою лінійної адитивної згортки для вибору моделей для експериментальних досліджень. Проведене теоретичне дослідження дозволяє покращити та виявити кращі існуючі моделі NLP для використання у рекомендаційних на основі отриманих результатів. Було описано метрики та оцінки моделей для проведення експериментальних досліджень та порівнянь моделей за критеріями швидкості, повноти та точності рекомендацій та проведено експериментальні дослідження для трьох задач: емоційний аналіз, багатокласова класифікація фільмів за жанрами та знаходження подібності між текстами. В результаті було знайдено найефективнішні моделі для виконання задач у рекомендаційних системах. Отримані результати були проаналізовані, що надалі дало змогу розробити власну рекомендаційну систему з використанням найефективніших моделей NLP.

Проведена робота відповідає усім поставленим вимогам та цілям та робить можливим розробку систем для надання рекомендацій на основі текстових даних з використанням сучасних NLP методів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Aggarwal C. C. Recommender Systems: The Textbook. Springer, 2018. 519c.
2. Rath A., Sahu S. R. Recurrent Neural Networks for Recommender Systems. Computational Intelligence and Machine Learning. 2020. Vol. 1, no. 1. P. 31–36. DOI: 10.36647/ciml/01.01.a004 (дата звернення: 03.04.2024).
3. Recommender System Using Transformer Model: A Systematic Literature Review / H. I. Pohan et al. 2022 1st International Conference on Information System & Information Technology (ICISIT), Yogyakarta, Indonesia, 27–28 July 2022. 2022.URL: https://doi.org/10.1109/icisit54091.2022.9873070 (дата звернення: 04.04.2024).
4. Ragab A., El-Kafrawy P. Embedding Based Recommender systems, a review and comparison. The Egyptian Journal of Language Engineering. 2022. P. 0. URL: https://doi.org/10.21608/ejle.2022.91884.1025 (дата звернення: 05.04.2024).
5. Golian N., Afanasieva I., Nazarenko D. Investigation of the Deep Learning Approaches to Classify Emotions in Texts. COLINS, 2021, pp. 206-224.
6. Kyrychenko I., Tereshchenko G., Gruzdo I., Cherednichenko O. Application of paragraphs vectors model for semantic text analysis // Proceedings of the 4th International Conference on Computational Linguistics and Intelligent Systems (COLINS-2020), 2020, Lviv, Ukraine. Vol. I. P. 283-293. 2020.
7. Pennington J., Socher R., Manning C. Glove: Global Vectors for Word Representation // Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Stroudsburg, PA, USA, 2014, Doha, Qatar. URL: http://dx.doi.org/10.3115/v1/d14-1162 (дата звернення: 07.04.2024).
8. FastText. URL: https://fasttext.cc/ (дата звернення: 09.04.2024).
9. Dive Into Deep Learning. URL: https://d2l.ai/chapter_recurrent-modern/lstm.html (дата звернення: 10.04.2024).

10. Dive Into Deep Learning. URL: https://d2l.ai/chapter_recurrent-modern/gru.html (дата звернення: 12.04.2024).
11. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A., Kaiser L., Polosukhin I. Attention is all you need // Advances in neural information processing systems, 2017. P. 5998–6008. DOI: 10.48550/arXiv.1706.03762. (дата звернення: 13.04.2024).
12. PAPERSWITHCODE. URL: https://paperswithcode.com/method/gpt (дата звернення: 09.04.2024).
13. Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I. et al. Language models are unsupervised multitask learners // OpenAI blog. – 2019. – Vol. 1, No. 8. – P. 9.
14. Brown T., Mann B., Ryder N., Subbiah M., Kaplan J., Dhariwal P., Neelakantan A., Shyam P., Sastry G., Askell A. та ін. Language models are few-shot learners, 2020. DOI: 10.48550/arXiv.2005.14165.
15. Medium. URL: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270 (дата звернення: 24.02.2024).
16. Hugging Face. URL: https://huggingface.co/transformers/v3.2.0/model_doc/bert.html (дата звернення: 13.04.2024).
17. Borealis AI. URL: https://www.borealisai.com/research-blogs/understanding-xlnet/ (дата звернення: 26.02.2024).
18. Hugging Face. URL: https://huggingface.co/docs/transformers/en/model_doc/xlnet (дата звернення: 14.04.2024).
19. Saravia E., Liu H.-C. T., Huang Y.-H., Wu J., Chen Y.-S. CARER: Contextualized Affect Representations for Emotion Recognition // Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics., 2018.
20. TMDb API. URL: https://developer.themoviedb.org/docs/getting-started.
21. Hugging Face. URL: https://huggingface.co/datasets/PhilipMay/stsb_multi_mt/viewer/en.