Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розробка рекомендаційної системи
«BookPicker»
(тема)

Виконав:
студент 2 курсу, групи СШІм-20-3
Тетерук Д. В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник проф., каф. ШІ, Кулішова Н. Є.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Тетерука Дмитра Вікторовича
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка рекомендаційної системи «BookPicker»

затверджена наказом університету від 24 03 2022 р. № 414

2. Термін подання студентом роботи до екзаменаційної комісії _____ _____ 20 ___ р.

3. Вихідні дані до роботи Науково-технічні публікації дані Інтернет та відомих проектів, електронні документації, тестові набори даних

4. Перелік питань, що потрібно опрацювати в роботі аналіз предметної області та постановка задачі, аналіз існуючих систем, аналіз існуючих алгоритмів рекомендацій, реалізація обраного алгоритму рекомендацій, розробка програмного веб-додатку з використанням сучасних фреймворків

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Рисунок 1 – Добуток матриць на прикладі книг та користувачів, Рисунок 2 – Схема бази даних, Рисунок 3 – Приклад реалізації моделі, Рисунок 4 – Реалізація основного методу з алгоритмом рекомендацій, Рисунок 5 – Приклад реалізації роутера та HTTP-запиту, Рисунок 6 – Структура проекту, Рисунок 7 – Інтерфейс початкової сторінки, Рисунок 8 – Інтерфейс сторінки реєстрації, Рисунок 9 – Інтерфейс сторінки авторизації, Рисунок 10 – Інтерфейс домашньої сторінки, Рисунок 11 – Інтерфейс спливаючого вікна зміни пошти, Рисунок 12 – Інтерфейс сторінки оцінювання книг

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Отримання завдання на кваліфікаційне проектування	28.03.2022	виконано
2.	Аналіз завдання та пошук літератури за темою	29.03-05.04	виконано
3.	Опрацювання літератури та аналіз об'єкту	06.04-12.04	виконано
4.	Вибір програмних засобів для розробки системи	13.04-19.04	виконано
5.	Розробка програмного засобу	20.04-03.05	виконано
6.	Аналіз отриманих результатів	04.04-06.05	виконано
7.	Оформлювання пояснювальної записки	06.05-11.05	виконано
8.	Оформлення презентаційних матеріалів	12.05.2022	виконано
9.	Представлення на рецензування	13.05.2022	виконано
10.	Представлення кваліфікаційної роботи		

Дата видачі завдання 28 03 2022 р.

Студент _____
(підпис)

Керівник роботи _____ _____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Записка пояснювальна: 66 с., 12 рис., 1 табл., 2 дод., 20 джерел.

АЛГОРИТМ, РЕКОМЕНДАЦІЙНА СИСТЕМА, ФРЕЙМВОРК, FLASK, PYTHON, SQLITE, WEB

Об’єкт дослідження – процес вибору користувачем книги для прочитання.

Предмет дослідження – простіші алгоритми такі як таблиця оцінок, user-based, item-based та алгоритм SVD.

Мета роботи – розробка WEB-додатку для рекомендаційної системи книжок.

Методи дослідження – аналіз існуючих найпростіших алгоритмів та алгоритму SVD, які використовуються у рекомендаційних системах.

Розроблено програмний WEB-додаток та інтегровано алгоритм рекомендаційної системи, реалізовано користувацький інтерфейс та проведено тестування працездатності програми.

РЕФЕРАТ

Пояснительная записка: 66 с., 12 рис., 1 табл., 2 прил., 20 источников.

АЛГОРИТМ, РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА, ФРЕЙМВОРК, FLASK, PYTHON, SQLITE, WEB

Объект исследования – процесс выбора пользователем книги для прочтения.

Предмет исследования – простые алгоритмы такие как таблица оценок, user-based, item-based и алгоритм SVD.

Цель работы – разработка WEB-приложения для рекомендательной системы книг.

Методы исследования – анализ существующих простых алгоритмов и алгоритма SVD, которые используются в рекомендательных системах.

Разработано программное WEB-приложение и интегрирован алгоритм рекомендательной системы, реализован интерфейс и проведено тестирование работоспособности программы.

ABSTRACT

Explanatory note: 66 p., 12 fig., 1 tabl., 2 ann., 20 sources.

ALGORITHM, FRAMEWORK, FLASK, PYTHON, RECOMMENDATION SYSTEM, SQLITE, WEB

The object of research is the process of selecting a book by the user for reading.

The subjects of research are simplest algorithms such as rating table, user-based, item-based, and SVD algorithm.

The purpose of the work is to develop a WEB-application for the book recommendation system.

Research methods are analysis of the existing simplest and SVD algorithms used in recommendation systems.

A software WEB-application was developed and recommendation system algorithm was integrated, a user interface was implemented, and the program was tested.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ І ТЕРМІНІВ

API – Application Programming Interface – програмний інтерфейс додатку;

DVD – Digital Versatile Disk – цифровий багатоцільовий диск;

IDE – Integrated Development Environment – інтегрована середа розробки;

REST – Representational State Transfer – передача репрезентативного стану;

SVD – Singular Value Decomposition – сингулярне розкладання величини;

VHS – Video Home System – домашня відео система;

WWW – World Wide Web – всесвітнє павутиння.

ВСТУП

Сучасний світ досить важко собі уявити без такої технології, як інтернет. З його допомогою компанії можуть продавати що завгодно, починаючи з продуктів і ліків й закінчуючи виробами мистецтва. Проте не існує досконалих технологій і відповідно у всього є і недоліки, а саме, надлишок товарів і складність вибору для користувача необхідного товару саме для нього, так як виробники намагаються зібрати навколо себе якомога більшу цільову групу.

Різноманітні рекомендаційні системи з'явилися в інтернеті досить давно, близько 20 років тому. Однак справжній прорив в цій області трапився приблизно 10-15 років тому, коли відбулося змагання Netflix Prize. Тоді компанія Netflix надавала в прокат VHS-касети і DVD, а не цифрові копії. Вже в той час для них було дуже важливо підвищити якість рекомендацій. Чим краще компанія рекомендує користувачам свої фільми, тим більше касет або дисків вони беруть в прокат. Відповідно, підвищується і прибуток. У 2006 році Netflix запустили змагання Netflix Prize. Вони поширили у відкритому доступі зібрані дані: близько 100 мільйонів оцінок за п'ятибальною шкалою з ідентифікаторами користувачів, які поставили оцінку. Учасникам цього змагання необхідно було якнайкраще передбачати оцінку користувача, яку він поставить тому чи іншому фільму. Якість передбачення вимірювалося за допомогою метрики RMSE (середньо-квадратичне відхилення). Netflix вже мав алгоритм, який передбачав оцінку з середньоквадратичною похибкою 0.9514 за метрикою RMSE. Завдання було поліпшити прогноз хоча б на 10% – до 0.8563. Переможцю змагання був обіцяний приз в $ 1 000 000. Змагання тривало близько трьох років. За перший рік похибку зменшили на 7%, далі все почало сповільнюватися. Однак, в кінці дві команди з різницею в 20 хвилин відправили свої рішення, кожне з яких проходило поріг в 10%, похибка у них була однакова з точністю до четвертого знаку. У задачі, над якою безліч команд билося три

роки, все вирішили якихось двадцять хвилин. Команда, яка запізнилася (як і багато інших, які брали участь в конкурсі) залишилися без призу, однак сам конкурс дуже сильно прискорив розвиток в галузі рекомендаційних систем [1].

Завдяки Netflix Prize розвиток рекомендаційних систем отримав величезний поштовх у розвитку в абсолютно різних предметних областях. Адже користувач може завжди знаходити для себе щось нове і це дозволяє утримувати його на довший проміжок часу.

В результаті наразі існує велика кількість різноманітних алгоритмів, від дуже простих і тривіальних до дуже складних, що враховують величезну кількість факторів. Методи, які використовують, найчастіше можливо поділити на три групи – колаборативна фільтрація, контентна фільтрація та експертні системи. Звичайно, у кожної групи є свої недоліки, тому пізніше з'явилися ще четверта група – гібридні, які поєднують в собі декілька підходів з усіх трьох груп.

Окрім цього, рекомендаційні системи часто використовуються в контексті штучного інтелекту. Можливість видачі рекомендацій, прогнозування подій і підкреслення кореляцій – все це результати застосування ШІ. З іншого боку, при імплементації рекомендаційних систем часто використовуються прийоми машинного навчання. Нейронні мережі дозволяють швидко вирішувати складні завдання і з легкістю маніпулювати великими даними. Взявши в якості вхідних даних список книжок, і зіставивши висновок з одними оцінками, мережа може засвоїти правила і потім з їх допомогою прогнозувати подальші рейтинги, які може проставити конкретний користувач [2].

Таким чином, рекомендаційні системи зараз є досить актуальною темою, так як їх можна застосувати в безлічі галузей в інтернеті і багато компаній намагаються їх реалізувати в своїх системах, якщо ще не зробили цього, і вдосконалити вже існуючі алгоритми.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Поняття рекомендаційної системи

Кожного дня люди використовують інтернет для того, щоб знайти речі, які їх цікавлять, будь то фільм, книга, пісня або одяг. І якщо раніше була проблема дізнатися чи з’явилось щось нове, то з появою того самого інтернету цей процес став набагато простішим. Сьогодні можна знайти що завгодно і коли завгодно, але тільки у тому випадку, коли користувач точно знає, що хоче побачити.

Швидке зростання кількості доступної інформації в цифровому просторі та кількості відвідувачів Інтернету створили потенційну проблему перевантаження інформації, яка заважає своєчасному доступу до цікавих предметів в Інтернеті. Системи пошуку, такі як Google або Yahoo, частково вирішили цю проблему, проте пріоритетність та персоналізація (де система відображає наявний вміст на вподобання та інтереси користувача) інформації відсутні. Це збільшило попит на системи рекомендацій сильніше, ніж будь-коли раніше. Рекомендаційні системи – це системи фільтрації інформації, які вирішують проблему перевантаження інформації, фільтруючи фрагмент життєвої інформації з великої кількості динамічно сформованих даних відповідно до уподобань, інтересів або спостережуваної поведінки щодо предмета. Система рекомендацій має можливість передбачати, чи конкретний користувач надасть перевагу елементу чи ні на основі профілю відповідного користувача. Система рекомендацій визначається як стратегія прийняття рішень для користувачів у складних інформаційних середовищах. Також система рекомендацій була визначена з точки зору електронної комерції як інструмент, який допомагає користувачам шукати записи знань, що стосуються інтересів та уподобань користувачів. Система рекомендацій була визначена як засіб сприяння та розширення соціального процесу, використовуючи рекомендації інших,

щоб зробити вибір, коли недостатньо особистих знань чи досвіду альтернатив. Системи рекомендацій надають користувачам персоналізовані, ексклюзивні рекомендації щодо вмісту та обслуговування.

Системи рекомендацій вигідні як постачальникам послуг, так і користувачам. Вони знижують час на пошук та вибір предметів в торгівельному онлайн середовищі. Необхідність використання ефективних та точних методів рекомендацій у системі, що дасть відповідні та надійні рекомендації для користувачів, не треба перекреслювати.

1.2 Підходи у рекомендаційних системах

Системи рекомендацій зазвичай використовують або колаборативну фільтрацію, або колаборативну фільтрацію разом з фільтруванням на основі змісту (також відомий як підхід на основі особистості), а також інші системи як, наприклад, системи на основі знань. Зараз рекомендаційні системи, як правило, поєднують один або кілька підходів до гібридної системи.

Різницю між колаборативною та контентною фільтрацією можна продемонструвати, порівнюючи дві рекомендаційні системи музики - Last.fm та Pandora Radio.

Last.fm динамічно створює «станцію» рекомендованих пісень, спостерігаючи за тим, які музикальні групи та окремі пісні користувач регулярно прослуховує, і порівнюючи їх із поведінкою слухачів інших користувачів. Last.fm додасть до «станції» композиції, які не відображаються в бібліотеці користувача, але які часто відтворюють інші користувачі зі схожими смаками. Оскільки такий підхід використовує поведінку користувачів, він є прикладом колаборативної методики фільтрації.

Пандора використовує властивості виконавця або пісні (підмножина 400 атрибутів, передбачених проектом «Музичний геном» в системі), щоб створити «станцію», яка додає музику з подібними властивостями. Відгуки

користувачів використовуються для уточнення результатів станції, демефазування певних атрибутів, коли користувачу не подобається певна композиція, і підкреслюючи інші атрибути, коли користувачеві подобається пісня. Це приклад контентної фільтрації [3].

Кожен тип системи має свої сильні та слабкі сторони. У наведеному вище прикладі Last.fm вимагає відразу великої кількості інформації про користувача системи, щоб дати якомога точні рекомендації. Це приклад проблеми холодного старту, що поширений у колаборативних системах фільтрації. З іншого боку, системі Pandora потрібно дуже мало інформації для початку і вона значно обмежена за обсягом (наприклад, вона може давати лише такі рекомендації, які схожі на початкові дані).

Один із підходів до проектування рекомендаційних систем, який широко застосовується – це спільна фільтрація. Спільна фільтрація базується на припущенні, що люди, які мали схожі оцінки в минулому, будуть мати їх і у майбутньому, і що їм сподобаються подібні предмети, як вони сподобалися в минулому. Система генерує рекомендації, використовуючи лише інформацію про рейтингові профілі для різних користувачів або елементів. Розташовуючи однорангових користувачів/елементи, що мають історію рейтингу, аналогічну поточному користувачеві або елементу, вони генерують рекомендації, використовуючи цих сусідів. Методи колаборативної фільтрації поділяються на ті, що мають в основі пам’ять та модель. Відомим прикладом підходів, заснованих на пам'яті, є алгоритм, заснований на користувачеві, тоді як модельний підхід – це Kernel-Mapping Recommender.

Основною перевагою підходу спільної фільтрації є те, що він не покладається на зміст, що аналізується машиною, і саме тому він здатний рекомендувати складні елементі з досить великою точністю, такі як книги, але при цьому не вимагає «розуміння» самого елемента. Велика кількість алгоритмів використовувались для вимірювання подібності користувачів

або подібності елементів у системах рекомендацій, наприклад, k-найближчий сусід (k-NN) і Pearson Correlation.

При побудові моделі з поведінки користувача часто розрізняють явні та неявні форми збору даних. Приклади явного збору даних включають наступне:

1) запитати користувача оцінити предмет за певною шкалою;

2) запитати користувача здійснити пошук;

3) запитати користувача класифікувати колекцію елементів від улюблених до найменш улюблених;

4) представити користувачу два елементи та попросити вибрати той, що більше подобається;

5) запитати користувача створити список предметів, які йому подобаються.

Приклади неявного збору даних включають наступне:

1) спостереження за елементами, які користувач переглядає в системі;

2) аналіз часу перегляду елемента/користувача;

3) ведення записів про елементи, які користувач переглядає в Інтернеті;

4) аналіз лайків та дизлайків користувача в соціальних мережах.

Підходи до спільної фільтрації часто страждають від трьох проблем: холодний старт, рідкість та масштабованість.

Холодний старт – проблема нового користувача або елемента з недостатньою кількістю даних для точних рекомендацій.

Масштабованість – мільйони користувачів та продуктів в системах, що надають рекомендації досить сильно навантажують її. Таким чином, для надання рекомендацій часто потрібна велика кількість обчислювальної потужності.

Розрідженість – кількість предметів, що продаються на основних сайтах електронної комерції, надзвичайно велика. Найактивніші

користувачі оцінюють лише невелику підмножину загальної бази даних. Таким чином, навіть найпопулярніші предмети мають дуже мало оцінок.

Одним з найвідоміших прикладів спільної фільтрації є спільне фільтрування по предметах (люди, які купують x, також купують y), алгоритм, популяризований системою рекомендацій Amazon.

Багато соціальних мереж спочатку використовували спільну фільтрацію, щоб рекомендувати нових друзів, груп та інших соціальних зв'язків, вивчаючи подібності між користувачем та його друзями. Спільна фільтрація все ще використовується як частина гібридних систем.

Ще один поширений підхід при розробці систем рекомендацій – це фільтрація на основі змісту. Методи фільтрування на основі змісту засновані на описі елемента та профілі користувача. Методи цієї групи найкраще підходять у тих ситуаціях, коли є деякий список характеристик об'єкта (назва, місцезнаходження, опис тощо), але не користувача. Рекомендаційні системи на основі змісту розглядають рекомендації як специфічну для користувача проблему класифікації, та вивчають класифікатор вподобань користувача на основі особливостей товару.

У подібних системах ключові слова використовуються для опису елементів, а профіль користувача будується для позначення типу елемента, який йому подобається. Іншими словами, ці алгоритми намагаються рекомендувати предмети, схожі на ті, які сподобалися користувачеві в минулому або ті, що він переглядає в теперішній час. Підхід не покладається на механізм авторизації користувача для створення цього зазвичай тимчасового профілю. Зокрема, різні позиції кандидатів порівнюються з предметами, які раніше оцінив користувач, і рекомендується ті, що найкраще відповідають. Цей підхід має своє коріння в дослідженнях пошуку та фільтрації інформації.

Для створення профілю користувача система в основному зосереджується на двох типах інформації:

1) модель уподобань користувача;

2) історія взаємодії користувача з системою рекомендацій.

В основному ці методи використовують профіль елемента (тобто набір дискретних атрибутів і особливостей), що характеризують елемент у системі. Для абстрагування особливостей елементів у системі застосовується алгоритм подання елементів. Широко використовуваний алгоритм – це векторне представлення простору. Система створює контентний профіль користувачів на основі зваженого вектору функцій предмета. Ваги позначають важливість кожної функції для користувача і можуть бути обчислені з індивідуально оцінених векторів змісту, використовуючи різні методи. Прості підходи використовують середні значення векторного рейтингу предмета, тоді як інші складні методи використовують методи машинного навчання, такі як класифікатори Байєса, аналіз кластерів, дерева рішень та штучні нейронні мережі, щоб оцінити ймовірність того, що користувачеві цей предмет сподобається.

Основне питання фільтрування на основі змісту полягає в тому, чи система здатна дізнатися шо подобається користувачу з його дій щодо одного джерела змісту та використовувати їх для інших типів змісту. Якщо система обмежується рекомендацією контенту того ж типу, який користувач вже використовує, значення системи рекомендацій значно менше, ніж коли можна рекомендувати інші типи зміст. Наприклад, рекомендувати статті новин на основі перегляду новин корисно, але було б набагато корисніше, коли музика, відео, продукти, дискусії тощо від різних служб можуть бути рекомендовані на основі перегляду новин. Для подолання цього більшість систем, що базуються на контентних рекомендаціях, зараз використовують певну форму гібридної системи.

Системи рекомендацій на основі змісту можуть також включати системи рекомендацій на основі думки. У деяких випадках користувачі можуть залишати огляд тексту або відгуки про елементи. Ці генеровані користувачем тексти – це неявні дані для системи рекомендацій, оскільки вони є потенційно багатим ресурсом як функцій товару, так і

оцінок/настроїв користувачів до елемента. Особливості, витягнуті з відгуків, можна розглядати як оцінки користувачів за відповідні функції. Популярні підходи до системи рекомендацій на основі думки використовують різні методи, включаючи пошук тексту, пошук інформації, аналіз настроїв та глибинне навчання.

Більшість систем рекомендацій зараз використовують гібридний підхід, поєднуючи спільну фільтрацію, фільтрацію на основі змісту та інші підходи. Немає жодної причини, чому кілька різних технік одного типу не можна було би гібридизувати. Гібридні підходи можуть бути реалізовані декількома способами: шляхом складання прогнозів на основі змісту та спільної роботи окремо, а потім їх поєднання; додаючи можливості підходу на основі змісту до підходу, що базується на спільній праці (і навпаки); або об'єднавши обидва підходи в одну гібридну модель. Кілька досліджень, які емпірично порівнюють ефективність роботи гібриду з чистими методами спільної роботи та змісту показали, що гібридні методи можуть дати більш точні рекомендації. Ці методи також можуть бути використані для подолання деяких найпоширеніших проблем в системах рекомендацій, таких як холодний запуск та проблеми з обмеженим ступенем дії, а також вузьким місцем в галузі інженерних знань у підходах на основі знань.

Netflix – хороший приклад використання гібридних систем рекомендацій. Веб-сайт дає рекомендації, порівнюючи звички перегляду та пошуку схожих користувачів (тобто спільну фільтрацію), а також пропонуючи фільми, які мають спільні характеристики з фільмами, які користувач високо оцінив (фільтрування на основі змісту) [4].

Деякі методи гібридизації включають:

1) зважений – комбінування кількості різних рекомендаційних компонентів чисельно;

2) переключення – вибір між рекомендаційними компонентами та застосування обраного;

3) змішані – рекомендації різних рекомендаційних систем подаються разом;

4) поєднання особливостей – особливості, отримані з різних джерел знань, поєднуються разом і надаються єдиному алгоритму рекомендацій;

5) розширення функції – обчислення функції або набору функцій, яка потім є частиною вводу до наступної методики;

6) каскад – кожній рекомендації відповідає деякий суворий пріоритет, причому нижчі пріоритети впливають на вищі і порушують їх зв'язки в оцінці;

7) метарівень – застосовується одна методика рекомендацій і виробляється якась модель, яка потім є входом, використовуваним наступною методикою.

Системи рекомендацій є корисною альтернативою алгоритмам пошуку, оскільки вони допомагають користувачам виявляти предмети, які вони могли б не знайти. Слід зазначити, що системи рекомендацій часто реалізуються за допомогою пошукових систем, що індексують нетрадиційні дані.

1.3 Проблема холодного старту

Так як майже усі вищеперераховані методи мають проблему холодного старту, то слід розглянути її, чому вона з’являється та як її уникати.

Холодний старт – це потенційна проблема в комп'ютерних інформаційних системах, які передбачають ступінь автоматизованого моделювання даних. Зокрема, це стосується того, що система не може робити жодних висновків для користувачів або елементів, про які вона ще не зібрала достатньої інформації.

Проблема холодного старту – добре відома і добре вивчена проблема для систем рекомендування. Рекомендаційні системи формують

специфічний тип фільтрації інформації, який намагається представити інформаційні елементи (електронна комерція, фільми, музика, книги, новини, зображення, веб-сторінки), які можуть зацікавити користувача. Зазвичай система рекомендацій порівнює профіль користувача з деякими еталонними характеристиками. Ці характеристики можуть бути пов'язані з характеристиками предметів (фільтрування на основі змісту) або соціальним середовищем користувача та минулою поведінкою (спільна фільтрація). Залежно від системи, користувач може бути пов'язаний з різними видами взаємодій: рейтинги, закладки, покупки, лайки, кількість відвідувань сторінки тощо.

Є три випадки через що відбувається холодний старт:

1) нова спільнота – стосується запуску рекомендаційної системи, хоча каталог елементів може існувати, майже немає користувачів, і через відсутність взаємодії з користувачем дуже важко надавати надійні рекомендації;

2) новий елемент – у систему додано новий елемент, він може містити інформацію про зміст, але взаємодії відсутні;

3) новий користувач – новий користувач зареєструвався та ще не завершив жодної взаємодії, тому не можна надавати персоналізовані рекомендації.

Проблема холодного старту товару стосується того, коли елементи, додані до каталогу, не мають жодної або дуже мало взаємодії. Це створює проблему, головним чином для алгоритмів спільної фільтрації через те, що вони роблять посилання на взаємодію елемента для надання рекомендацій. Якщо немає взаємодії, то чистий алгоритм спільної роботи не може рекомендувати цей елемент. Якщо є лише кілька взаємодій, хоча алгоритм спільної роботи зможе його рекомендувати, якість цих рекомендацій буде поганою. Тому виникає ще одне питання, яке вже не пов’язане з новинками, а скоріше з непопулярними предметами. У деяких випадках (наприклад, рекомендації щодо фільмів) може трапитися, що декілька предметів

отримують надзвичайно велику кількість взаємодій, тоді як більшість предметів отримує лише їх частину. Це називається упередженістю популярності [5].

У контексті предметів, які мають холодний старт, важливе значення має зміщення популярності, оскільки може статися, що багато предметів, навіть якщо вони були в каталозі місяцями, отримали лише кілька взаємодій. Це створює негативний цикл, в якому непопулярні предмети будуть погано рекомендовані, тому отримають набагато меншу видимість, ніж популярні, і будуть намагатися отримувати взаємодію. Хоча очікується, що деякі пункти будуть менш популярними, ніж інші, в цьому питанні конкретно йдеться про те, що рекомендаційна система не має достатньої кількості спільної інформації, щоб рекомендувати їх змістовно та надійно.

Алгоритми фільтрації на основі змісту, з іншого боку, теоретично набагато менше схильні до проблеми нового елементу. Оскільки рекомендаційні системи на основі змісту вибирають, які саме елементи рекомендувати, виходячи з функції, якою вони володіють, навіть якщо взаємодії для нового елемента не існує, все-таки його функції дозволять робити рекомендації. Звичайно, це передбачає, що новий елемент вже буде описаний його атрибутами, що не завжди так. Розглянемо випадок так званих редакційних особливостей (наприклад, режисер, роль, назва, рік), які завжди відомі, коли товар, у цьому випадку фільм, додається до каталогу. Однак інших видів атрибутів може не бути, наприклад, функцій, витягнутих із відгуків користувачів та тегів. Алгоритми, що базуються на змісті, спираючись на функції, що надаються користувачем, також страждають від проблеми елемента «холодний старт», оскільки для нових елементів, якщо немає (або дуже мало) взаємодій, також не будуть доступні (або дуже мало) огляди користувачів та теги [6].

Новий випадок користувача стосується того, коли новий користувач зареєструвався в системі і протягом певного періоду часу рекомендаційна система повинна надавати рекомендації, не покладаючись на минулі

взаємодії користувача, оскільки жодної ще не відбулося. Ця проблема має особливе значення, коли рекомендаційна система є частиною послуги, що пропонується користувачам, оскільки користувач, який стикається з рекомендаціями низької якості, може незабаром вирішити припинити використання системи, перш ніж забезпечити достатню взаємодію, щоб дозволити рекомендаційній системі зрозуміти його інтереси. Основна стратегія роботи з новими користувачами – попросити їх надати певні переваги для створення початкового профілю користувача. Потрібно знайти поріг між тривалістю процесу реєстрації користувача, який, якщо занадто довгий, може призвести до того, що користувач не пройде її, та кількістю вихідних даних, необхідних для належного функціонування рекомендаційної системи.

Як і у випадку з новими елементами, не всі алгоритми рекомендацій впливають однаково. Рекомендаційні системи для предметів будуть зачеплені, оскільки вони покладаються на профіль користувача, щоб оцінити, наскільки важливі вподобання інших користувачів. Найбільше впливають алгоритми спільної фільтрації, оскільки без взаємодій не можна робити висновок щодо налаштувань користувача. Алгоритми рекомендацій користувач-користувач поводяться дещо інакше. Алгоритм, орієнтований на зміст користувача, покладається на функції користувача (наприклад, вік, стать, країна), щоб знайти подібних користувачів та рекомендувати позитивні елементи, з якими взаємодіяв подібний користувач і через це є надійним для нового користувача. Вся ця інформація набувається в процесі реєстрації, або шляхом прохання користувача ввести дані самостійно, або шляхом використання даних, уже доступних, наприклад, у облікових записах у соціальних мережах.

Завдяки великій кількості доступних алгоритмів рекомендацій, а також типу та характеристикам системи було розроблено багато стратегій пом’якшення проблеми холодного старту. Основний підхід – покладатися

на гібридні рекомендаційні системи, щоб пом'якшити недоліки однієї категорії чи моделі, поєднуючи її з іншою.

Усі три категорії холодного старту (нова спільнота, новий елемент та новий користувач) мають загальну рису – відсутність взаємодії з користувачами, та представляють деякі спільності у стратегіях, доступних для їх вирішення.

Загальна стратегія роботи з новими елементами полягає в поєднанні рекомендаційної системи на основі спільної фільтрації, для теплих предметів, із рекомендаційною системою на основі змісту, для холодних предметів. Хоча два алгоритми можуть поєднуватися по-різному, головний недолік цього методу пов’язаний з низькою якістю рекомендацій, яку часто демонструють рекомендації на основі змісту в сценаріях, коли важко забезпечити вичерпний опис характеристик елемента. Якщо для нових користувачів немає демографічних особливостей, або їх якість занадто низька, загальною стратегією є надання їм неперсоніфікованих рекомендацій.

Один із доступних варіантів роботи з холодними користувачами або предметами – це швидке отримання деяких даних про перевагу. Існують різні способи зробити це залежно від кількості необхідної інформації. Ці методи називаються стратегіями виявлення переваг. Це може бути зроблено явно (шляхом запиту користувача) або неявно (шляхом спостереження за поведінкою користувача). В обох випадках проблема «холодного старту» означитиме, що користувач повинен докласти багато зусиль, використовуючи систему в її «тупому» стані – сприяючи побудові свого профілю користувача – перш ніж система може почати надавати будь-які розумні рекомендації.

Наприклад, MovieLens, веб-система рекомендацій для фільмів, просить користувача оцінити деякі фільми як частину реєстрації. Хоча стратегія отримання переваг є простим та ефективним способом спілкування з новими користувачами, додаткові вимоги під час реєстрації

зроблять процес більш трудомістким для користувача. Більше того, якість отриманих налаштувань може бути не ідеальною, оскільки користувач може оцінювати товари, які він бачив місяці чи роки тому, або надані рейтинги можуть бути майже випадковими, якби користувач надав їх, не звертаючи уваги, щоб швидко завершити реєстрацію [7].

Побудова профілю користувача також може бути автоматизована шляхом інтеграції інформації з інших дій користувачів, таких як історії перегляду чи платформи соціальних медіа. Якщо, наприклад, користувач читав інформацію про певного виконавця музики з медіа-порталу, то пов'язана система рекомендацій автоматично запропонувала б релізи виконавця, коли користувач відвідує музичний магазин.

Різновидом попереднього підходу є автоматичне присвоєння рейтингів новим предметам, виходячи з рейтингів, присвоєних громадою іншим аналогічним предметам. Подібність предметів визначається відповідно до змістових характеристик предметів.

Ще одна з можливих методик – застосовувати активне навчання (машинне навчання). Основна мета активного навчання полягає в тому, щоб керувати користувачем у процесі вибору переваг, щоб попросити його оцінити лише ті пункти, які з точки зору рекомендацій будуть найбільш інформативними. Це робиться шляхом аналізу доступних даних та оцінки корисності точок даних (наприклад, рейтингів, взаємодій). Як приклад, скажімо, що ми хочемо побудувати два кластери з певної хмари точок. Як тільки ми визначили дві точки, кожна з яких належить до іншого кластеру, що є наступним найбільш інформативним моментом? Якщо ми візьмемо точку, близьку до однієї, ми вже знаємо, що вона, ймовірно, буде належати до того ж кластеру. Якщо ми виберемо точку, що знаходиться між двома кластерами, то знання, до якого кластеру належить, допоможе нам знайти місце, де знаходиться межа, що дозволить класифікувати безліч інших точок лише за допомогою кількох спостережень [8].

В останні роки були запропоновані більш прогресивні стратегії, всі вони покладаються на машинне навчання та намагаються об’єднати зміст та спільну інформацію в єдину модель. Одним із прикладів такого підходу називається атрибут картографічного означення, який налаштований на алгоритми матричної факторизації. Основна ідея полягає в наступному. Матрична модель факторизації представляє взаємодію елементів користувача як добуток двох прямокутних матриць, зміст яких засвоюється за допомогою відомих взаємодій за допомогою машинного навчання. Кожному користувачу буде відповідати рядок з першої матриці, а кожному елементу буде відповідати стовпець другої матриці. Рядок або стовпець, який зв’язаний з певним користувачем або елементом, має назву прихований фактор. У разі додавання нового елементу, у нього немає пов'язаних прихованих факторів, а відсутність взаємодій не дозволяє вивчити їх, як це було зроблено з іншими предметами. Якщо кожен елемент асоціюється з деякими функціями (наприклад, автором, роком, видавцем, акторами), можна визначити функцію вбудовування, яка за ознаками елемента оцінює відповідні елементи прихованих факторів. Функція вбудовування може бути розроблена багатьма способами, і вона навчається з уже доступними даними з теплих предметів. Це ж стосується і нового користувача, як якщо б для них доступна якась інформація (наприклад, вік, національність, стать), то його латентні фактори можна оцінити за допомогою функції вбудовування.

Ще один недавній підхід, який має схожість із відображенням функцій, – це побудова гібридного рекомендаційного фільтрування на основі змісту, в якому функції, будь-яких елементів або користувачів, зважуються відповідно до сприйняття користувачем важливості. Для того, щоб визначити фільм, який може сподобатися користувачеві, різні атрибути (наприклад, актори, режисер, країна, назва) матимуть різне значення. Як приклад розглянемо франшизу про Джеймса Бонда, головний актор багато раз змінювався протягом багатьох років, а дехто цього не робив, як Лоїс

Максвелл. Тому її присутність, мабуть, буде кращим ідентифікатором такого фільму, ніж присутність одного з різних головних акторів. Хоча існують різні методи зважування функцій для функцій користувачів або елементів у системах рекомендування, більшість з них є з домену пошуку інформації, наприклад tf – idf, Okapi BM25, лише деякі були розроблені спеціально для рекомендаційних систем.

Зокрема, методи зважування гібридних характеристик розроблені для домену системи рекомендацій. Деякі з них засвоюють вагому особливість, безпосередньо використовуючи взаємодію користувача з предметами, наприклад, FBSM. Інші покладаються на проміжну модель спільної роботи, що навчається на теплих предметах, і намагаються навчитися вмісту, який визначає ваги, які краще наближають модель спільної роботи [9].

Системи рекомендацій відкривають нові можливості отримання персоналізованої інформації в Інтернеті. Це також допомагає полегшити проблему перевантаження інформації, що є дуже поширеним явищем із системами пошуку інформації та дає можливість користувачам мати доступ до продуктів та послуг, які не є доступними для користувачів у системі.

1.4 Вимірювання ефективності

Оцінка важлива при оцінці ефективності алгоритмів рекомендацій. Для вимірювання ефективності систем рекомендування та порівняння різних підходів доступні три типи оцінок: дослідження користувачів, онлайнові оцінки (тести A/B) та офлайн-оцінки.

Найпоширенішими показниками є середня помилка у квадраті та середня коренева помилка у квадраті, остання використана в премії Netflix. Показники пошуку інформації, такі як точність і відкликання або DCG, корисні для оцінки якості рекомендованого методу. Різноманітність, новизна та охоплення також розглядаються як важливі аспекти оцінювання. Однак багато класичних оціночних заходів піддаються високій критиці.

Дослідження користувачів досить малі. Кілька десятків або сотень користувачів представляють рекомендації, створені різними рекомендаційними підходами, а потім користувачі оцінюють, які рекомендації найкращі. У тестах A/B рекомендації демонструються, як правило, тисячам користувачів реального продукту, і система рекомендацій випадковим чином вибирає принаймні два різні рекомендаційні підходи для створення рекомендацій. Ефективність вимірюється за допомогою неявних заходів ефективності, таких як швидкість конверсії або швидкість кліку. Офлайн-оцінки базуються на історичних даних, наприклад набір даних, що містить інформацію про те, як користувачі раніше оцінювали фільми.

Потім ефективність рекомендаційних підходів оцінюється на основі того, наскільки вдало рекомендаційний підхід може передбачити рейтинги користувачів у наборі даних. Хоча оцінка є явним вираженням того, чи сподобався користувачеві фільм, така інформація доступна не у всіх областях. Наприклад, у сфері систем рекомендування цитат користувачі, як правило, не оцінюють цитування чи рекомендовані статті. У таких випадках офлайн-оцінки можуть використовувати неявні заходи ефективності. Наприклад, можна припустити, що ефективна система рекомендацій, яка здатна рекомендувати якомога більше статей, що містяться в довідковому списку дослідницької статті. Однак такі дослідження в режимі офлайн багато дослідників вважають критичним. Наприклад, було показано, що результати офлайн-оцінок мають низьку кореляцію з результатами досліджень користувачів або тестів A/B. Показано, що набір даних, популярний для офлайн-оцінки, містить дублікати даних і, таким чином, призводить до неправильних висновків при оцінці алгоритмів. Часто результати так званих офлайн-оцінок не співвідносяться з фактично оціненою задоволеністю користувачів. Це, мабуть, тому, що навчання в режимі офлайн є дуже упередженим по відношенню до високодоступних предметів, а на дані тестування в режимі офлайн дуже сильно впливають результати роботи онлайнового модуля рекомендацій [10].

1.5 Важливі фактори

Як правило, дослідження рекомендаційних систем стосуються пошуку найбільш точних алгоритмів рекомендацій. Однак є ряд факторів, які також важливі.

Різноманітність – користувачі, як правило, більше задовольняються рекомендаціями, коли існує більша різноманітність у внутрішньому списку, наприклад предмети різних художників.

Наполегливість рекомендацій – у деяких ситуаціях ефективніше повторно показувати рекомендації або дозволяти користувачам повторно оцінювати елемент, ніж показувати нові елементи. Для цього є кілька причин. Користувачі можуть ігнорувати елементи, коли вони відображаються вперше, наприклад, оскільки вони не мали часу уважно перевірити рекомендації.

Конфіденційність – рекомендаційні системи зазвичай мають вирішувати проблеми конфіденційності, оскільки користувачі повинні розкривати конфіденційну інформацію. Створення профілів користувачів за допомогою спільної фільтрації може бути проблематичним з точки зору конфіденційності. Багато європейських країн мають сильну культуру конфіденційності даних, і кожна спроба запровадити будь-який рівень профілювання користувачів може призвести до негативного відгуку клієнтів. Проведено багато досліджень щодо поточних питань конфіденційності в цьому просторі. Приз Netflix особливо примітний детальною особистою інформацією, опублікованою в його наборі даних.

Демографічні дані користувачів – Beel et al. з'ясували, що демографічна інформація користувачів може впливати на те, наскільки задоволені користувачі рекомендаціями. У своїй роботі вони показують, що літні користувачі, як правило, більше цікавляться рекомендаціями, ніж молодші користувачі.

Надійність – коли користувачі можуть брати участь у системі рекомендацій, необхідно вирішити питання шахрайства.

Інтуїтивна прозорливість – це міра «наскільки дивовижні рекомендації». Наприклад, система рекомендацій, яка рекомендує молоко покупцеві в продуктовому магазині, може бути цілком точною, але це не є хорошою рекомендацією, оскільки це очевидний предмет, який клієнт може придбати.

Довіра – рекомендаційна система має мало значення для користувача, якщо користувач не довіряє системі. Довіру можна побудувати за допомогою системи рекомендацій, пояснивши, як вона генерує рекомендації та чому рекомендує товар [11].

1.6 Постановка задачі

Задачею є проаналізувати існуючи алгоритми рекомендаційних систем, обрати або вдосконалити один з них та створити WEB-додаток в основі якого буде працювати метод рекомендацій. В ході роботи будуть спроектовані та реалізовані база даних, користувальницький інтерфейс, серверна частина за допомогою сучасних технологій з алгоритмами рекомендацій. Ґрунтуючись на даній цілі були сформульовані основні завдання:

1) обрати алгоритм фільтрації, дослідити та вдосконалити його властивості;

2) виконати проектування бази даних;

3) реалізувати серверну частину та алгоритм рекомендацій;

4) побудувати користувацький інтерфейс;

5) об’єднати всі частини додатку;

6) провести тестування.

2 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

2.1 Найпростіші алгоритми колаборативної фільтрації

Колаборативна фільтрація – це найбільш поширена методика, яка застосовується, коли мова йде про створення інтелектуальних систем рекомендацій, які мають можливість навчитися надавати кращі рекомендації, так як збирається більше інформації про користувачів. Мотивація спільної фільтрації походить від ідеї, що люди часто отримують найкращі рекомендації від когось із смаком, схожим на свій. Колаборативна фільтрація використовує методи відповідності користувачів з подібними інтересами та надання рекомендацій на цій основі.

Алгоритми спільної фільтрації вимагають:

1) активної участі користувачів;

2) простого методу представлення інтересів користувачів;

3) алгоритмів, здатних відповідати людям з подібними інтересами.

Користувач висловлює свої переваги рейтинговими елементами (наприклад, книгами, фільмами чи компакт-дисками) системи. Ці рейтинги можна розглядати як приблизне відображення інтересу користувача до відповідної категорії елементів.

Система аналізує наданий рейтинг відповідного користувача, а далі порівнює його з іншими користувачами та знаходить людей з найбільш «схожими» інтересами.

Зі схожими користувачами система рекомендує елементи, які подібні користувачі оцінили високо, але ще не оцінювались цим користувачем (відсутність оцінки часто розглядають як незнайомість товару).

Основна проблема спільної фільтрації – як поєднувати та зважувати інтереси сусідів користувачів. Інколи користувачі можуть відразу оцінити рекомендовані предмети. В результаті система отримує все більш точне представлення інтересів користувачів із часом [12].

Для початку формалізуємо задачу. Нехай є множина користувачів $u \in U$, множина об'єктів $i \in I$ і множина подій $(r_{ui}, u, i, ...) \in \mathcal{D}$ (дії, які користувачі здійснюють з об'єктами). Кожна подія задається користувачем u, об'єктом i, своїм результатом r_{ui} й, іноді, ще деякими характеристиками. Рекомендаційна система має виконати кілька дій.

Передбачити інтерес користувача:

$$\hat{r_{ui}} = \text{Predict}(u, i, ...) \approx r_{ui}, \quad (2.1)$$

де $\hat{r}_{ui}$ – оцінка користувача u об’єкту i;

r_{ui} – результат події;

u – користувач;

i – об’єкт.

Створити персональні рекомендації:

$$u \longmapsto (i_1, ..., i_K) = \text{Recommend}_K(u, ...), \quad (2.2)$$

де u – користувач;

i_K – об’єкт з номером K.

Знайти схожі об'єкти:

$$u \longmapsto (i_1, ..., i_M) = \text{Similar}_M(i), \quad (2.3)$$

де u – користувач;

i_M – об’єкт з номером M.

Також, є наявною таблиця з оцінками користувачів деяких об’єктів та невідомі оцінки, позначені знаками питань, які треба спрогнозувати (таблиця 2.1).

Таблиця 2.1 – Оцінки користувачів

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
User 1	5	4	5			
User 2	4	?	5			
User 3		4	5	?	4	
User 4			?	3	4	
User 5	?		4	2	4	?

2.1.1 Кластеризація користувачів

Основною мотивацією колаборативної фільтрації є ідея, що схожі користувачі частіше за все обирають одні й ті ж самі елементи. Почнемо з найпростішого методу.

Виберемо деяку умовну міру схожості користувачів за їх історією оцінок – $sim(u, v)$.

Розіб’ємо всіх користувачів в різні групи (кластери), керуючись принципом їх подібності між собою – $u \mapsto F(u)$.

Оцінку користувача деякому об'єкту будемо розраховувати як середню оцінку серед оцінок представників кластера користувачів даному об'єкту:

$$\hat{r}_{ui} = \frac{1}{|F(u)|} \sum_{v \in F(u)} r_{vi}, \tag{2.4}$$

де $\hat{r}_{ui}$ – оцінка користувача u об’єкту i;

$F(u)$ – кластер користувачів;

u – користувач;

v – користувач зі схожими оцінками;

r_{vi} – оцінка користувача з кластеру конкретному об’єкту.

У цього алгоритму є кілька недоліків:

а) система не може нічого рекомендувати новим або нетиповим користувачам, через те, що їх неможливо віднести до жодного кластеру та немає характеристик, за якими можливо розрахувати подібність;

б) не враховується специфіка кожного користувача. В якомусь сенсі ми ділимо всіх користувачів на якісь класи;

в) прогноз об’єкту зробити не вийде, якщо в кластері немає жодної оцінки цього об'єкту.

2.1.2 User-based

Спробуємо дещо покращити попередній метод і замінити жорстку кластеризацію наступним чином:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\Sigma_{v \in \bigcup_i \text{sim}\,(u,v)} (r_{vi} - \bar{r}_v)}{\Sigma_{v \in \bigcup_i \text{sim}\,(u,v)}}, \tag{2.5}$$

де $\hat{r}_{ui}$ – оцінка користувача u об’єкту i;

$\bar{r}_u$ – середня оцінка користувача u;

U_i – множина користувачів, які оцінили об’єкт i;

v – подібний користувач;

$\text{sim}\,(u, v)$ – подібність користувачів u та v;

r_{vi} – оцінка користувача v об’єкту i;

$\bar{r}_v$ – середня оцінка користувача v;

Однак, даний алгоритм також має свої проблеми:

а) система не може нічого рекомендувати новим або нетиповим користувачам. Для таких користувачів все ще не має можливості знайти подібних;

б) холодний старт – нові об'єкти не можуть бути рекомендованими.

2.1.3 Item-based

Метод item-based є симетричним методом до user-based:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\Sigma_{j \in I \text{sim}\,(i,j)}(r_{uj} - \bar{r}_j)}{\Sigma_{j \in I \text{sim}\,(i,j)}}, \tag{2.6}$$

де $\hat{r}_{ui}$ – оцінка користувача u об’єкту i;

$\bar{r}_i$ – середня оцінка об’єкту i;

j – об’єкт;

I – множина об’єктів;

$\text{sim}\,(i,j)$ – подібність об’єктів i та j;

r_{uj} – оцінка користувача u об’єкту j;

$\bar{r}_j$ – середня оцінка об’єкта j.

У попередньому методі ми відштовхувалися від ідеї, що, наприклад, книга сподобається користувачеві, якщо він сподобався його друзям. В цьому методі ми вважаємо, що книга сподобається користувачеві, якщо йому сподобалися схожі елементи.

Проблеми:

а) холодний старт – нові об'єкти нікому не рекомендуються;

б) рекомендації часто досить тривіальні.

Всі зазначені методи мають певні недоліки. Розрідженість даних – на практиці багато комерційних систем рекомендацій мають великі набори даних. В результаті, матриця елементів користувача, яка використовується для спільної фільтрації, може бути надзвичайно великою і розрідженою, що спричиняє проблеми при розрахунку рекомендації.

Типова проблема, спричинена рідкістю даних, є проблема холодного старту, яка була розглянута раніше. Оскільки методи спільної фільтрації рекомендують предмети, що основані на попередніх уподобаннях користувачів, новим користувачам потрібно оцінити достатню кількість

елементів, щоб система змогла точно запам’ятати їх переваги, і тим самим, надавати якісні рекомендації.

Аналогічно, нові предмети також стикаються з цією проблемою. Коли нові елементи додаються в систему, їх необхідно оцінювати значною кількістю користувачів, перш ніж вони можуть бути рекомендовані користувачам, які мають схожі інтереси з тими, хто їх оцінив. Проблема з новим елементом не впливає на рекомендації, що основані на змісті предмету, оскільки рекомендація елемента базується на її дискретному наборі метаданих, а не на рейтингу.

Коли кількість користувачів та об’єктів зростає, традиційні алгоритми спільної фільтрації зазнаватимуть серйозних проблем з масштабованістю. Наприклад, із мільйонами клієнтів O(M) та мільйонами предметів O(N), алгоритм спільної фільтрації, що має складність f(n), вже вимагає занадто великих обчислювальних ресурсів. Крім того, великій кількості систем потрібно відразу реагувати на вимоги в Інтернеті, та надавати рекомендації для всіх користувачів незалежно від їх історії покупок та рейтингів, що вимагає ще більшої масштабованості системи. Великі компанії, такі як Twitter, використовують кластери машин для масштабування рекомендацій для своїх сотень мільйонів користувачів, при цьому більшість обчислень відбувається в дуже великих машинах пам’яті [13].

Під синонімами розуміється тенденція, що ряд однакових або дуже подібних предметів має різні назви чи записи. Більшість рекомендаційних систем не мають можливості виявити цю приховану асоціацію і через це ставитися до цих елементів по-різному.

Сірі вівці відносяться до користувачів, думки яких не узгоджуються з жодною групою людей і, відповідно, не отримують користі від спільної фільтрації. Чорні вівці – це група користувачів, чиї нетипові інтереси роблять надавання рекомендації майже неможливим. Хоча це є недоліком системи рекомендацій, звичайні неелектронні рекомендації також мають

великі проблеми в таких випадках, тому мати чорних овець є прийнятним збоєм.

В рекомендаційних системах, де кожен може надавати оцінки, люди можуть ставити безліч гарних оцінок за власні позиції та негативних для своїх конкурентів. Досить часто системам спільної фільтрації необхідно вводити запобіжні заходи для елімінації подібних маніпуляцій.

Очікується, що алгоритми спільної фільтрації збільшать різноманітність, оскільки вони допомагають нам відкривати нові продукти. Проте, деякі алгоритми можуть робити навпаки. Оскільки алгоритми спільної фільтрації рекомендують елементи на основі минулих показів або рейтингів, вони зазвичай не можуть рекомендувати елементи з обмеженими історичними даними. Це може створити ефект для популярних елементів, схожий на позитивні відгуки. Наступний розглянутий алгоритм більш складний, проте може вирішити зазначені вище проблеми [14].

2.2 Алгоритм SVD

SVD (Singular Value Decomposition), що перекладається як сингулярне розкладання матриці. У теоремі сингулярного розкладання стверджується, що будь-яка матриця **A** розміру $n \times m$ має розкладання в добуток трьох матриць: U, Σ і V^T:

$$A = U \cdot \Sigma \cdot V^T, \tag{2.7}$$

де A – деяка матриця розміру $n \times m$;

U – ортогональна матриця розміру $n \times n$;

Σ – діагональна не квадратна матриця розміру $n \times m$;

V^T – ортогональна матриця розміру $m \times m$.

Матриці U і V ортогональні, з цього маємо наступне:

$$UU^T = I_n, VV^T = I_m, \tag{2.8}$$

де U – ортогональна матриця розміру n;

U^T – транспонована матриця U;

I_n – одинична матриця розміру n;

V – ортогональна матриця розміру m;

V^T – транспонована матриця V;

I_m – одинична матриця розміру m.

Σ – діагональна (хоча і не квадратна):

$$\Sigma = \text{diag}\left(\lambda_{1,\dots}, \lambda_{min(n,m)}\right), \lambda_1 \geq \cdots \geq \lambda_{min(n,m)} \geq 0, \tag{2.9}$$

де $\text{diag}\left(\lambda_{1,\dots}, \lambda_{min(n,m)}\right)$ – елементи, які знаходяться на діагоналі матриці;

$\lambda_{min(n,m)}$ – елемент, який знаходиться у рядку n та стовпці m.

Причому лямбда в матриці Σ будуть відсортовані по спаданню. Крім звичайного розкладу може бути ще усічене, коли з лямбда залишаються тільки перші d чисел, а решту ми вважаємо рівними нулю.

$$\lambda_{d+1}, \dots, \lambda_{min(n,m)} \coloneqq 0, \tag{2.10}$$

де λ_{d+1} – елемент на діагоналі матриці.

Це рівнозначно тому, що у матриць U і V ми залишаємо тільки перші d стовпців, а матрицю Σ зменшуємо до квадратної розміром $d \times d$.

$$A' = U' \cdot \Sigma' \cdot V'^T, \tag{2.11}$$

де A' – наближена матриця **A** розміру $n \times m$;

U' – матриця U зменшена до розміру $n \times$ d;

Σ' – матриця Σ зменшена до розміру d × d;

V'^T – матриця V^T зменшена до розміру d × m.

Отримана матриця **A**' добре наближає вихідну матрицю **A** і, крім цього, є найкращим низькоранговим наближенням з позиції середньо-квадратичного відхилення.

2.2.1 SVD в рекомендаційних системах

Даний алгоритм широко використовується в розрахунку рекомендацій. Наприклад, є матриця, яку розклали на її добуток трьох матриць. Важливо зазначити, що розклали не точно, а приблизно. Спростимо всі множники, позначивши добуток перших двох матриць як одну матрицю (рисунок 2.1):

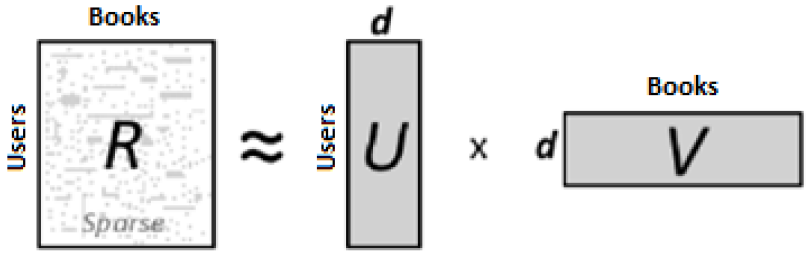

Рисунок 2.1 – Добуток матриць на прикладі книг та користувачів

Щоб передбачити оцінку користувача U для книги I, ми беремо деякий вектор p_u (набір параметрів) для відповідного користувача і вектор для відповідної книги q_i. Їх скалярний добуток і буде потрібним нам прогнозом оцінки: $r_{ui} = \langle p_u, q_i \rangle$.

Алгоритм досить простий, але дає досить гарні результати. Крім передбачень оцінок, з його допомогою є можливість з історії користувачів виявляти приховані характеристики об'єктів і інтереси користувачів. Наприклад, може трапитись ситуація, коли на першій координаті вектору у кожного користувача буде стояти число, яке показує, чи схожий користувач більше на чоловіка або жінку, а на другій координаті - число, що відображає приблизний вік користувача. У книзі же перша координата буде показувати, чи цікавий він більше чоловікам або жінкам, а друга – для якої вікової групи користувачів він більш цікавий [15].

Як висновок, в ході реалізації програмного WEB-додатку за основу краще брати алгоритм SVD, так як він позбавлений проблеми холодного старту. До того ж очікується, що користувач, який буде користуватися даним додатком, буде новим і оцінить лише кілька книг. Іншими словами, початкова матриця буде складатися з оцінок користувачів, які будуть взяті із тестового набору даних, а потім доповнюватися ще одним рядком з оцінками цього користувача, який хоче отримати рекомендації в даному WEB-додатку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Вибір мови програмування

Першим кроком при розробці є вибір мов програмування, за допомогою яких буде реалізований програмний додаток. Якщо для імплементації користувальницького інтерфейсу вибір не великий – HTML, CSS, JS та його фреймворки, то для back-end частини необхідно більш детальне порівняння. Для більш швидкої реалізації front-end інтерфейсу буде використовуватись бібліотека з готовими стилями Bootstrap для надання красивого вигляду. Це дозволить створити достатньо сучасний на вигляд сайт.

Вибір мови для back-end складніший. Наразі популярна дуже велика кількість різних мов програмування таких як Java, Python, Ruby, C++, C#. Порівняємо декілька мов із мовою Python.

Python проти C#:

1) Python об'єктно-орієнтований, процедурний та функціональний, а C# тільки об'єктно-орієнтований;

2) Python має велику кількість компіляторів під різні операційні системи. C# залежить від платформи;

3) Python використовується для розробки веб-додатків та ігор, та машинного навчання;

4) C# використовується для ігрових і мобільних розробок.

Отже, у цьому порівнянні можемо зробити висновок, що Python більш спрямований на веб-додатки та машинне навчання, тому можна відкинути C#, так як він менше підходить для вирішення поставленої задачі.

Python проти C++:

1) Python має велику кількість компіляторів під різні операційні системи. C++ компілюється відразу до машинного коду та має менший перелік платформ, що підтримуються;

2) Python об'єктно-орієнтований, процедурний та функціональний, C++ є і об'єктно-орієнтований і процедурний;

3) Python не підтримує перевантаження оператора, C++ підтримує;

4) Python не підтримує структури та об'єднання, C++ підтримує.

5) Python виконується повільніше, ніж C++;

6) на Python простіше писати код, ніж на C++.

Python це більш сучасний варіант за C++ та Java, хоча працює він не так швидко, але переваги у вигляді великої кількості бібліотек для машинного навчання та підтримки різних парадигм програмування роблять цю мову програмування зручнішою.

Python проти Ruby:

1) Python та Ruby – підтримують об'єктно-орієнтовану парадигму;

2) Python та Ruby – мови програмування загального призначення;

3) Python та Ruby – динамічні мови програмування;

4) Ruby не залежить від платформи, але і Python підтримує багато платформ;

5) Python та Ruby підтримують спадкування;

6) для еквівалентної програми Python вимагає менше рядків коду, ніж Ruby;

7) Python швидша за Ruby.

Майже в усьому ці дві мові дуже схожі, але Python швидший, більш розповсюджений та більше підходить для вирішення задач з області штучного інтелекту.

Отже, після порівняння можна зробити висновок, що усі мови програмування мають свої переваги та недоліки, тому необхідно відштовхуватися від мети використання та особливостей мови, які найбільше підходять під конкретну задачу. Так як планується розробка WEB-додатку, в якому ключовою частиною є швидкість виконання алгоритму рекомендацій та невелика складність написання цього алгоритму, то найбільш доречна мова програмування буде Python. Крім

того, ще однією важливою перевагою цієї мови є бібліотеки та фреймворки для WEB-додатків та машинного навчання, такі як Flask та scipy, які слід розглянути.

Flask — це мікро веб-фреймворк, написаний на Python. Його класифікують як мікро-фреймворк, оскільки він не вимагає особливих інструментів або бібліотек. У ньому немає рівня абстракції бази даних, перевірки форм чи будь-яких інших компонентів, де вже існуючі сторонні бібліотеки надають загальні функції. Однак, Flask підтримує розширення, які можуть додавати функції програми, як якщо б вони були реалізовані в самому Flask. Існують розширення для об’єктно-реляційних маперів, перевірки форм, обробки завантаження, різних технологій відкритої аутентифікації та кількох загальних інструментів, пов’язаних із фреймворком [16].

Мікро-фреймворк Flask є частиною Pallets Projects (раніше Pocoo) і базується на декількох з них.

Werkzeug — це допоміжна бібліотека для мови програмування Python, іншими словами, набір інструментів для додатків інтерфейсу шлюзу веб-сервера. Werkzeug може реалізувати програмні об'єкти для запитів, відповідей та корисних функцій. Його можна використовувати для створення користувацького програмного забезпечення поверх нього і підтримує версії Python 2.7, 3.5 і новіші [17].

Jinja також створений автором Flask та являється движком шаблонів для мови програмування Python. Подібно до веб-фреймворку Django, він обробляє шаблони в пісочниці [18].

MarkupSafe — це бібліотека обробки строк для мови програмування Python. Одноіменний тип MarkupSafe розширює строковий тип Python і позначає його вміст як «безпечний». Поєднання MarkupSafe зі звичайними строками автоматично екранує немарковані рядки, уникаючи подвійного екранування вже позначених рядків.

ItsDangerous — це безпечна бібліотека серіалізації даних для мови програмування Python. Вона використовується для збереження сеансу програми Flask у файлах cookie, не дозволяючи користувачам змінювати вміст сеансу.

Крім цього, фреймворк Flask має наступні особливості:

1) сервер розробки та налагоджувач;

2) інтегрована підтримка модульного тестування;

3) розподілення RESTful запитів;

4) підтримка безпечних файлів cookie (сеанси на стороні клієнта);

5) 100% сумісність з WSGI 1.0;

6) сумісність з Google App Engine;

7) можливість розширення функціональності за допомогою інших бібліотек чи фреймворків.

Так як Flask був обраний як основний фреймворк, то для зв'язування front-end та back-end частин програми доречно використовувати вбудований движок шаблонів Jinja.

Фреймворк Flask з безліччю можливостей для інтеграцій з іншими інструментами та вже вбудованими компонентами в результаті ідеально підходить для сучасної веб-розробки в Інтернеті, надаючи можливість швидко реалізувати поставлену задачу.

3.2 Інструменти для роботи з базою даних

Так як програмний додаток буде використовувати базу даних для збереження профілів користувачів з їх оцінками, то потрібно вибрати таке рішення, яке найкраще підходить.

В сучасних системах наразі є два популярних напрями – реляційні та нереляційні бази даних. Основними перевагами нереляційних баз даних є легка масштабованість та реплікація, а також відсутність потреби створення структури даних. Так як для даної задачі необхідно зберігати невелику

кількість даних, то ці переваги суттєво не вплинуть на швидкість, тому краще вибрати реляційну базу даних.

Одними з популярних сьогодні реляційних баз даних є MySQL, PostgreSQL та SQLite. Порівняємо ці рішення.

Раніше MySQL мав репутацію дуже швидкої бази даних для великих робочих навантажень, деколи ціною паралелізму при змішуванні з операціями запису.

Розробники PostgreSQL, що також відомий як Postgres, позиціонують свій продукт як «найсучаснішу у світі реляційну базу даних з відкритим кодом». Вона була розроблена для того, щоб бути багатофункціональною, мала можливість розширення та відповідала необхідним стандартам. У минулому продуктивність Postgres була більш усередненою – читання, зазвичай, було повільніше, ніж у MySQL, але вона могла записувати великі обсяги даних більш ефективно і краще працювати з паралелізмом.

Різниця в продуктивності між MySQL та Postgres значною мірою стерта в останніх версіях. MySQL по новим стандартам все ще надзвичайно швидко читає дані, але лише за умови використання старого механізму MyISAM. Якщо використовується InnoDB, то відмінності в швидкості незначні. Функції InnoDB є дуже важливими для корпоративних та споживчих програм, тому від використання старого двигуна вони відмовляються. Проте, MySQL був також оптимізований для зменшення різниці у швидкості, коли йде мова про записи даних у великих обсягах.

Вибираючи між MySQL та PostgreSQL, продуктивність не повинна бути фактором для більшості звичайних програм – її вистачає у будь-якому випадку, тому і для даної задачі це не є ключовим фактором.

На відміну від розглянутих баз даних SQLite не є окремою програмою, а скоріше бібліотекою, яку розробники програмного забезпечення вбудовують у свої програми. Це позитивно відображається на кількості пам’яті, необхідної для підтримки роботи бази даних. SQLite має інструменти для інтеграції до багатьох мов програмування. Зазвичай вона

дотримується синтаксису PostgreSQL, але не вимагає перевірки типів. Це означає, що можна, наприклад, вставити рядок у стовпець, визначений як ціле число. SQLite використовує автоматичне регресійне тестування перед кожним релізом нової версії. Понад 2 мільйони тестів виконуються в рамках перевірки. Починаючи з версії 3.6.17, релізи SQLite мають 100% покриття тестами коду, що є важливим фактором, який впливає на цілісність даних та безпеку у використанні даної бази даних. SQLite зберігає данні в єдиному файлі, який дуже легко можна перенести, а сама бібліотека з програмним кодом бази даних займає лише 600 кілобайт. Однак, серед недоліків SQLite можна виділити відсутність поняття користувачів бази даних, так як вона підтримаю тільки одне з'єднання одночасно [19]. Це може стати ключовим фактором, через який компанії можуть відмовитися від даного рішення. Проте, для вирішення поставленої задачі більшого не треба через те, що весь код знаходиться в рамках одного програмного додатку і немає необхідності створювати розподілену систему для реалізації системи рекомендацій даного типу.

Після наведеного порівняння баз даних, можна дійти висновку, що MySQL та Postgres це сучасні та самодостатні рішення, які доречно використовувати для великих або розподілених систем, в яких виконується велика кількість запитів до бази даних одночасно. Для такого типу програмних додатків не буде великою витратою виділення ресурсів для забезпечення роботи таких баз даних. Для того, щоб зберігати невелику кількість інформації для однієї невеликої системи більш доречно використовувати більш легке рішення, а саме SQLite.

Для роботи з базою даних використовується розширення для Flask – Flask-SQLAlchemy, яке додає підтримку бібліотеки SQLAlchemy до програми. Воно спрямоване на спрощення використання SQLAlchemy з Flask, надаючи корисні параметри за замовчуванням і додаткові помічники, які полегшують виконання звичайних завдань.

У більшості випадків SQLAlchemy використовується як інструмент ORM, який перекладає класи Python в таблиці реляційних баз даних і автоматично перетворює виклики функцій у оператори SQL. SQLAlchemy надає стандартний інтерфейс, який дозволяє розробникам створювати незалежний від бази даних код для зв’язку з різноманітними механізмами баз даних. Також SQLAlchemy дозволяє описувати структури баз даних та способи взаємодії з ними мовою Python без використання SQL.

Основна задача технології ORM – забезпечити роботу з даними в термінах класів, а не таблиць даних і навпаки, перетворити терміни і дані класів в дані, що будуть придатні для зберігання в СУБД. Необхідно також забезпечити інтерфейс для CRUD-операцій над даними. Ідея технології полягає в тому, що необхідно позбутися від необхідності писати SQL-код для взаємодії в СУБД [20].

3.3 Проектування схеми бази даних

Першим кроком треба виділити сутності, які будуть зберігатися у базі даних додатку. Для роботи рекомендаційної системи потрібні такі сутності, як книга, користувач та оцінки користувачів.

Для того, щоб отримати максимальну ефективність зберігання даних необхідно використовувати нормалізацією і привести схему в 3 нормальну форму. Розглянемо поняття нормалізації.

Нормальна форма – властивість відношення у реляційній моделі даних, що характеризує його з точки зору надмірності, яка потенційно призводить до логічних помилкових результатів вибору або зміни даних. Нормальна форма визначається як сукупність вимог, яким повинно дотримуватися відношення.

Процес перетворення відношень таблиць баз даних до виду, відповідних нормальним формам, називається нормалізацією. Нормалізація використовується для того, щоб привести структуру бази даних до виду, що

забезпечує мінімальну логічну збитковість, і ціль не є зменшення чи збільшення швидкості роботи або зменшення чи збільшення фізичного об’єму бази даних. Ціль нормалізації включає в себе зменшення потенціального протиріччя в інформації, що зберігається в базі даних. Загальний процес нормалізації включає наступне:

1) виключення деяких типів збитковості;

2) виключення деяких аномалій оновлення;

3) полегшення процедури застосування необхідних обмежень цілісності даних.

Для того, щоб таблиця була в першій нормальній формі, вона повинна дотримуватися наступних правил:

1) таблиця повинна мати лише окремі цінні атрибути/стовпці;

2) значення, які зберігаються у стовпці, повинні бути в одному домені;

3) всі стовпці в таблиці повинні мати унікальні найменування;

4) порядок збереження даних не має значення.

Щоб таблиця була у другій нормальній формі:

1) таблиця повинна бути у першій нормальній формі;

2) таблиця не повинна мати часткової залежності.

Таблиця відповідає третій нормальній формі, коли:

1) таблиця повинна бути в другій нормальній формі;

2) таблиця не має транзитивних залежностей.

Отже, розглянувши необхідні правила, можна створювати схему бази даних. Вона має наступний вигляд (рис. 3.1):

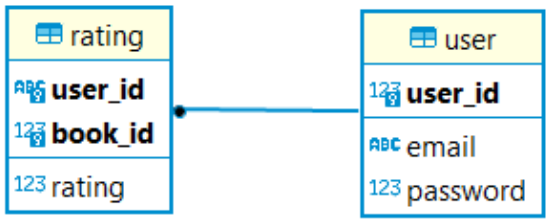

Рисунок 3.1 – Схема бази даних

3.4 Реалізація функціоналу

Для реалізації були використані такі IDE як PyCharm, Brackets та DBeaver через те, що вони були найбільш зручні.

Перш за все у requirements.txt файлі збірки проекту були додані необхідні для роботи програми залежності.

Після додавання залежностей до проекту, була створена база даних. Далі, був експортований файл бази даних, який був поміщений до репозиторію проекту для подальшої роботи з нею за допомогою програмного додатку.

Наступним кроком були створені моделі, які мають такий самий набір полів, як і таблиці у базі даних. Для роботи з цими об'єктами були розроблені класи, кожен з яких наслідує специфічний клас SQLAlchemy для зв'язування класів та таблиць бази даних. Приклад реалізації моделі має такий вигляд (рис. 3.2):

```
9
10  class User(db.Model, UserMixin):
11      user_id = db.Column(db.Integer, primary_key=True)
12      email = db.Column(db.String(30), unique=True, nullable=False)
13      password = db.Column(db.Integer, nullable=False)
14
15      def get_id(self):
16          return (self.user_id)
17
18      def __repr__(self):
19          return f"User('{self.email}','{self.password}')"
20
```

Рисунок 3.2 – Приклад реалізації моделі

На прикладі модель наслідує клас db.Model, що надає розуміння бібліотеці SQLAlchemy, що він повинен створити зв'язок між класом та

відповідною таблицею, або створити таблицю, якщо її немає в базі даних. За допомогою методу db.Column виконується зв'язок між полями та колонками таблиці.

Після реалізації моделей можна переходити до імплементації алгоритму рекомендації, який буде розмішений в окремому файлі. Даний файл містить у собі два методи, один з яких є допоміжним, а інший використовується в інших файлах. Основний метод приймає шлях до файлу з книгами, шлях до файлу з рейтингами та ідентифікатор авторизованого користувача. Цей метод завантажує до пам'яті дані книг та рейтингів з файлів, та знаходить оцінки користувача в базі даних, якими потім наповнює матрицю. Для того, щоб отримати рекомендації, додаємо до матриці ще один рядок з оцінками користувача, який оцінив хоча б одну книгу. Розкладаємо матрицю згідно алгоритму SVD та достаємо окремо матриці U та V. Обчислюємо приблизний прогноз оцінки для кожного фільму та повертаємо список лише з десяти книг, прогнози яких мають найбільші значення (рис. 3.3):

```
33  def runRecommender(file1, file2, user_id):
34      pd.set_option('display.float_format', lambda x: '%.3f' % x)
35
36      book_list = pd.read_json(file1)
37      ratings_list = pd.read_json(file2)
38
39      # book_data = pd.merge(ratings_list, book_list, on='book_id')
40      book_list['book_id'] = book_list['book_id'].apply(pd.to_numeric)
41      book_data = ratings_list.pivot(index='user_id', columns='book_id', values='rating').fillna(0)
42      R = book_data.to_numpy()
43      user_ratings_mean = np.mean(R, axis=1)
44      R_demeaned = R - user_ratings_mean.reshape(-1, 1)
45      U, sigma, Vt = svds(R_demeaned, k=50)
46      sigma = np.diag(sigma)
47
48      all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-1, 1)
49      preds_df = pd.DataFrame(all_user_predicted_ratings, columns=book_data.columns)
50      already_rated, predictions, rated = recommend_books(preds_df, user_id, book_list, ratings_list, 10)
51      predict = predictions.to_json(orient="records")
52      rated = rated.to_json(orient="records")
53      return predict, rated
54
```

Рисунок 3.3 – Реалізація основного методу з алгоритмом рекомендацій

Також необхідний функціонал, який буде виконувати примітивну роботу – видавати список книг для оцінки, проводити реєстрацію та авторизацію.

Після реалізації даного функціоналу необхідно створити роутер, який буде безпосередньо приймати запити з front-end та видавати результати взаємодії користувача з сервером. Для реалізації краще використовувати сучасну REST-архітектуру.

REST – архітектурний стиль взаємодії компонентів розподіленого додатка в мережі. REST являє собою узгоджений набір правил, що враховуються при проектуванні клієнт-серверної системи. У деяких випадках це призводить до спрощення архітектури і підвищення продуктивності. Загалом, компоненти в REST взаємодіють на зразок взаємодії клієнтів і серверів у всесвітньому павутині.

В мережі Інтернет виклик віддаленої процедури може являти собою звичайний HTTP-запит (GET, POST, PUT та інші), що зазвичай називаються «REST-запитом». Необхідні дані для серверу передаються в якості параметрів запиту від клієнта.

Для WEB-додатків, які побудовані з урахуванням всіх правил REST, застосовують термін «RESTful».

На відміну від WEB-сервісів на основі SOAP, не існує «офіційного» стандарту для RESTful API. REST є архітектурним стилем, в той час як SOAP є протоколом, таким чином REST не описує в якому форматі клієнт та сервер повинні надавати дані. Незважаючи на те, що REST не є стандартом сам по собі, більшість систем з даним архітектурним стилем використовують такі стандарти, як HTTP, URL, JSON і XML.

Для роботи з системою рекомендацій необхідно створити методи, які будуть відповідати наступним HTTP-запитам:

1) видача початкової сторінки з посиланнями на сторінки реєстрації та авторизації;

2) видача та обробка даних з сторінки реєстрації;

3) видача та обробка даних з сторінки авторизації;

4) обробка запиту виходу з акаунту користувача;

5) видача списку книг для оцінки та обробка оцінок користувача;

6) видача списку оцінених книг та рекомендацій на основі цих оцінок.

Розглянемо код одного з методів HTTP-запиту цього роутера (рис. 3.4):

```
@app.route('/home', methods=['GET', 'POST'])
@login_required
def home():
    print("User:::::", current_user.user_id)
    form = updateEmail()
    if request.method == 'POST':
        if form.validate_on_submit():
            current_user.email = form.email.data
            db.session.commit()
            flash('Your account has been updated!', 'success')
            return redirect(url_for('home'))
        elif request.method == 'GET':
            form.email.data = current_user.email
        return render_template('home.html', title='Home', form=form)
    else:
        available = False
        fileRating = os.path.join(app.static_folder, 'ratings.json')
        fileBook = os.path.join(app.static_folder, 'books.json')

        with open(fileRating) as f:
            dataRating = json.load(f)
        for i in range(len(dataRating)):
            if dataRating[i]["user_id"] == current_user.user_id:
                print(dataRating[i]["user_id"])
                available = True
        if available == False:
            suggested = [{"book_id": 0, "original_title": "No rating, data unavailable", "genre": "-"}]
            rated = [{"book_id": 0, "original_title": "No rating, data unavailable", "genre": "-"}]
        if available == True:
            predict, rated = runRecommender(fileBook, fileRating, current_user.user_id)
            suggested = json.loads(predict)
            rated = json.loads(rated)
        return render_template('home.html', title='Home', form=form, suggested=suggested, rated=rated)

    return render_template('home.html', title='Home', form=form)
```

Рисунок 3.4 – Приклад реалізації роутера та HTTP-запиту

Анотація @app.route описує шлях та HTTP-методи, які підтримуються даним методом, а @login_required повідомляє, що користувач повинен бути авторизованим в системі, щоб мати доступ до методу. Також, важливо зазначити, що використаний метод render_template, з бібліотеки Flask,

використовується для того, щоб відобразити відповідну сторінку користувачу з зазначеними параметрами, які в ній використовуються.

Кінцевий вигляд структури проекту має такий вигляд (рис. 3.5):

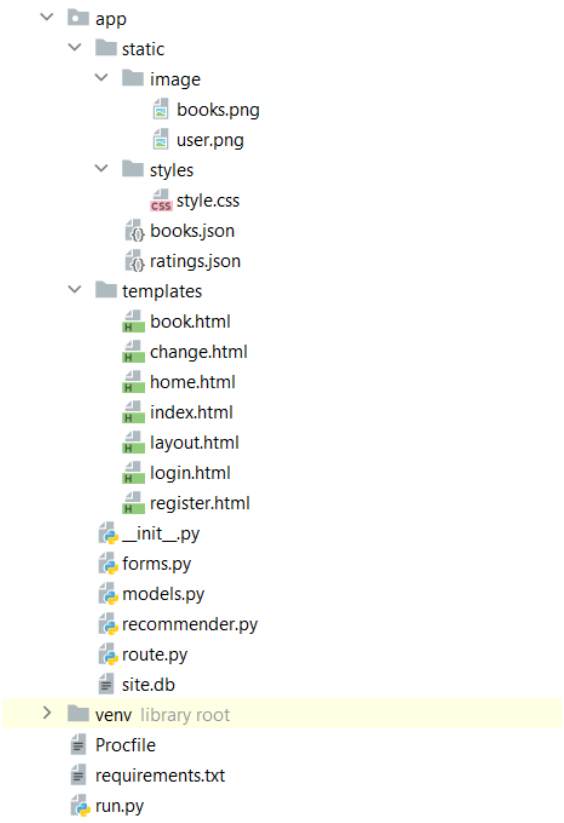

Рисунок 3.5 – Структура проекту

Після завершення роботи над базою даних та back-end частиною можна приступати до шаблонів HTML сторінок, які будуть відображатися користувачу.

На початковій сторінці відображаються лише дві кнопки-посилання на сторінку авторизації та реєстрації, так як використання додатку необхідно, щоб користувач існував в системі. Для відображення усіх значень, які приходять на front-end з back-end використовувалися Jinja оператори. Кінцевий вигляд сторінки має наступний вигляд (рис. 3.6):

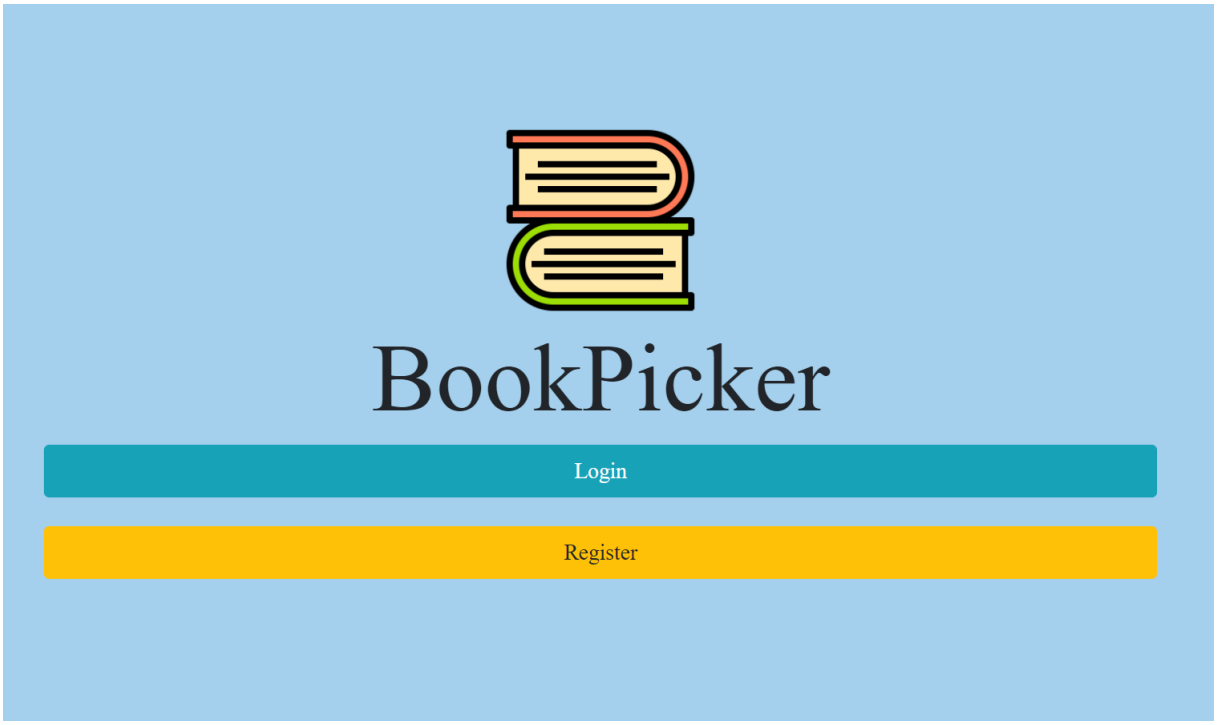

Рисунок 3.6 – Інтерфейс початкової сторінки

Сторінка реєстрації має досить універсальний вигляд та вимагає від користувача заповнити типові поля (рис. 3.7):

Рисунок 3.7 – Інтерфейс сторінки реєстрації

Сторінка авторизації виглядає наступним чином (рис. 3.8):

Рисунок 3.8 – Інтерфейс сторінки авторизації

Після авторизації в системі, користувач потрапляє на домашню сторінку, де відображається список оцінених книг та рекомендації до них.

Крім цього, користувач також має можливість перейти до інших сторінок, вийти з акаунту та змінити пошту, використовуючи відповідні кнопки. Домашня сторінка користувача має наступний вигляд (рис. 3.9):

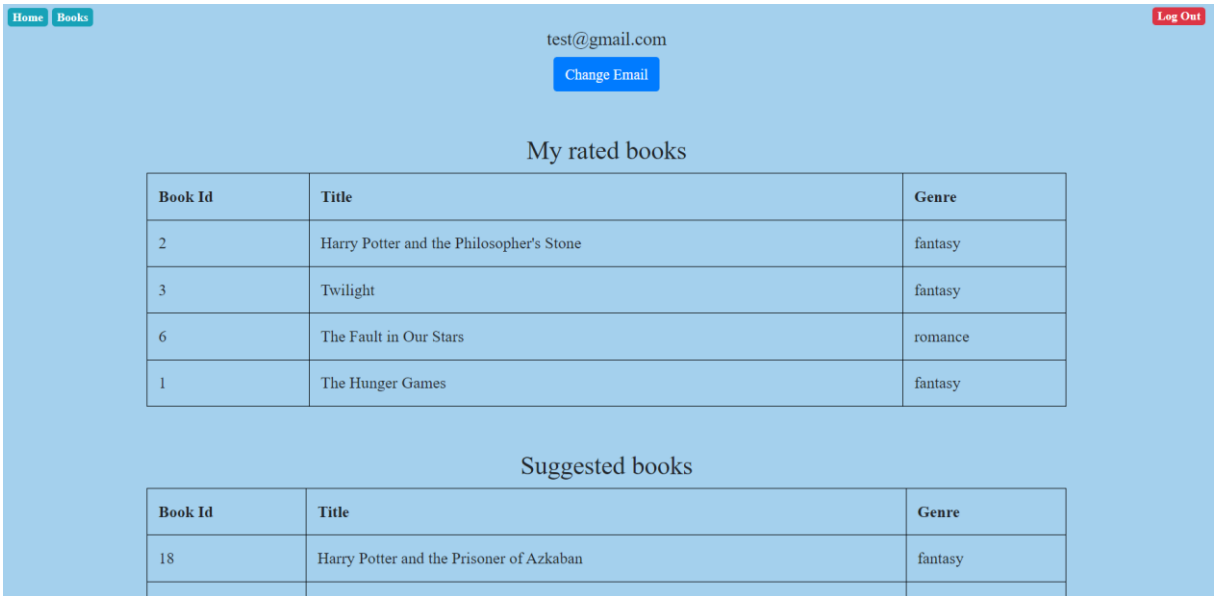

Рисунок 3.9 – Інтерфейс домашньої сторінки

Якщо користувачу необхідно буде змінити пошту в акаунті, він може натиснути на відповідну кнопку на домашній сторінці та ввести нову в спливаючому вікні (рис. 3.10):

Рисунок 3.10 – Інтерфейс спливаючого вікна зміни пошти

Для того, щоб отримати рекомендації, користувачу необхідно перейти до відповідної сторінки та виставити оцінки книгам. Ця сторінка має інтерфейс, в якому відображається перелік книг та їх жанр, а також форма для додавання оцінки та пошук (рис. 3.11):

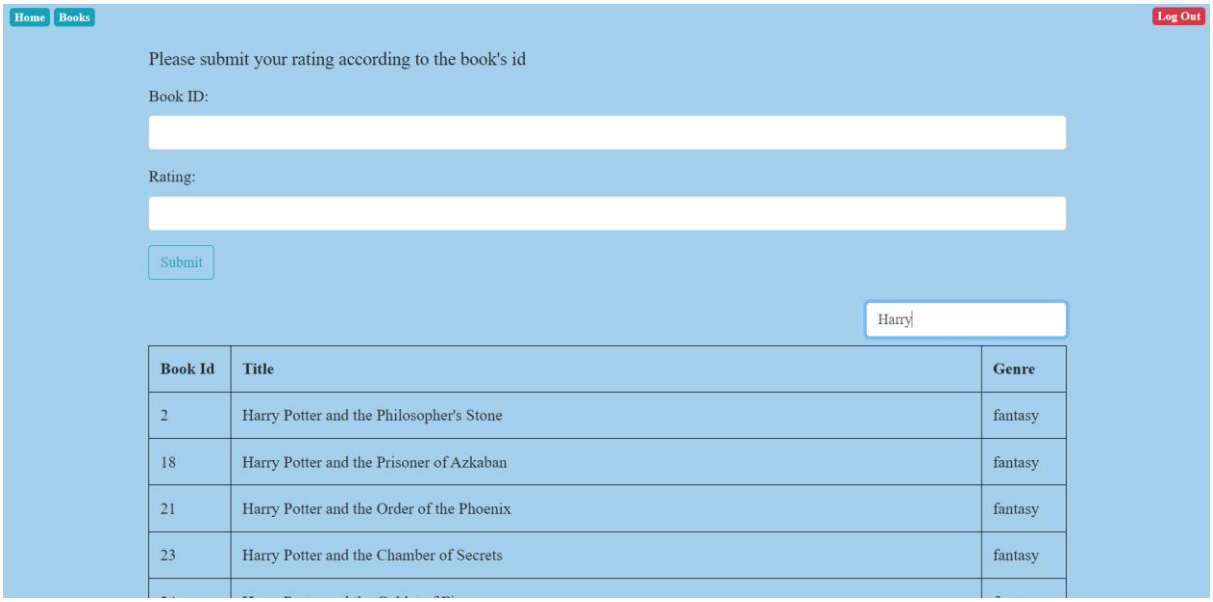

Рисунок 3.11 – Інтерфейс сторінки оцінювання книг

Після завершення роботи над front-end частиною, було проведене тестування, в результаті якого можна зробити висновок, що програмний додаток виконує своє завдання, помилок не виникає, рекомендації видаються достатньо логічні.

3.5 Варіанти подальшого розвитку

Одним із варіантів подальшого розвитку є удосконалення алгоритму рекомендацій. В ході аналізу алгоритмів, було з’ясовано, що гібридні алгоритми частіше надають більш кращий результат, тому в майбутньому можна додати ще декілька алгоритмів до вже реалізованого алгоритму SVD та створити гібридний підхід для пошуку рекомендацій.

У разі, якщо продовжувати розвивати WEB-додаток, то можливе використання сучасних cloud технологій, за допомогою яких можливо запустити програмний додаток на виділених ресурсах сторонньої компанії, наприклад AWS, не купляти окремі сервера та домен. Це дозволить з легкістю керувати навантаженням на сервер та базу даних, масштабувати кількість серверів у разі необхідності, а також дає гарантію, що проблеми зі струмом чи природні катастрофи не вплинуть на працездатність додатку.

ВИСНОВКИ

Людство завжди мало потребу у товарах чи послугах. З плином часу інтернет став одним із основних торгівельних майданчиків з неймовірною кількістю товарів чи послуг, і зараз обирати між схожими товарами все складніше, а коли людина не впевнена, що саме їх необхідно, то стає ще складніше. Однією з причин цієї проблеми стало зростання ринку пропозицій та реклама. Для вирішення даної проблеми були створені системи рекомендацій. Завдяки поштовху від компанії Netflix було удосконалено старі алгоритми та розроблено нові.

У ході роботи була проаналізована предметна область, розглянуті існуючі приклади різноманітних систем рекомендацій, їх переваги та недоліки. Виділені основні групи алгоритмів, які використовуються у сучасних рекомендаційних системах та порівняні на прикладах реалізованих систем. В результаті було виявлено, що прості методи, такі як кластерізація, user-based, item-based мають проблеми масштабованістю, холодним стартом та примітивністю рекомендацій. У той же час, методи, що засновані на розкладанні матриці майже позбавлені цих недоліків, тому для подальшого розгляду та реалізації у програмному WEB-додатку був обраний саме цей алгоритм.

Основним принципом алгоритму SVD є розклад прямокутної матриці на три різні матриці, які мають менше значень, ніж початкова. У ході аналізу даного алгоритму було виявлено, що для своєї не самої великої складності він має результати на достатньо високому рівні. Для роботи потрібна лише матриця з оцінками інших користувачів та оцінки користувача, для якого буде проведений прогноз. Крім цього, за допомогою даного алгоритму можна виявляти деякі скриті характеристики користувача чи об’єкта.

Для програмної реалізації було вирішено розробити WEB-додаток рекомендаційної системи книжок з використанням бази даних. Проведено аналіз списку популярних мов програмування та обраний Python, через свої

переваги у вигляді великої кількості зручних інструментів для роботи в сфері машинного навчання та розробки WEB-додатків, а також достатньої швидкості роботи. Після аналізу вимог до баз даних, було вирішено використовувати легке рішення у вигляді SQLite. З метою пришвидшення розробки, допоміжними інструментами були обрані Flask фреймворк та SQLAlchemy. Для розробки користувальницького інтерфейсу використовувалися такі технології як HTML, CSS, JS, бібліотека стилів Bootstrap та движок шаблонів Jinja.

Після імплементації програмного додатку, було проведено тестування працездатності системи, в результаті якого проблем не виявлено. Виділені шляхи подальшого розвитку програми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Frey M., Developing Netflix's Recommendation Algorithms. *Netflix Recommends*. California, University of California Press, 2021. P. 63–95.

2. Aggarwal C. C. An Introduction to Recommender Systems. *Recommender Systems*. Cham, Springer International Publishing, 2016. P. 1–28.

3. Мелешко Е. В., Семенов С. Г., Хох В. Д. ДОСЛІДЖЕННЯ МЕТОДІВ ПОБУДОВИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ В МЕРЕЖІ ІНТЕРНЕТ. *Системи управління, навігації та зв'язку. Збірник наукових праць*. 2018. Т. 1, № 47. С. 131–136.

4. Gomez-Uribe C. A., Hunt N. The Netflix Recommender System. *ACM Transactions on Management Information Systems*. 2016. Vol. 6, No. 4. P. 1–19.

5. Ricci F., Rokach L., Shapira B. Introduction to Recommender Systems Handbook. *Recommender Systems Handbook*. Boston, MA: Springer, 2010. P. 1–35.

6. Sharma P., Yadav L. MOVIE RECOMMENDATION SYSTEM USING ITEM BASED COLLABORATIVE FILTERING. *International Journal of Innovative Research in Computer Science & Technology*. 2020. Vol. 8, No. 4. P. 1–10.

7. Raghuwanshi S. K., Pateriya R. K. Movie Recommendation System Content-Based and Collaborative Filtering. *International Journal of Computer Sciences and Engineering*. 2018. Vol. 6, No. 4. P. 476–481.

8. Saini V. K., Singh J. R. Recommendation System. *International Journal for Modern Trends in Science and Technology*. 2021. Vol. 6, No. 12. P. 484–492.

9. Tout K., Evans D. J., Yakan A. Collaborative filtering: special case in predictive analysis. *International Journal of Computer Mathematics*. 2005. Vol. 82, No. 1. P. 1–11.

10. Мелешко Ю. МЕТОДИ ОЦІНКИ ЯКОСТІ РОБОТИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ. *Системи управління, навігації та зв'язку. Збірник наукових праць*. 2018. Т. 5, № 51. С. 92–97.

11. Devika P. V. Book Recommendation System. *International Conference on Computing Communication and Networking Technologies (ICCCNT)*: Proceedings of 12th conference, Kharagpur, India, 6–8 July, 2021. P. 3–9.

12. Cacheda F. Comparison of collaborative filtering algorithms. *ACM Transactions on the Web*. 2011. Vol. 5, No. 1. P. 1–33.

13. Dahiya P., Duhan N. Comparative Analysis of Various Collaborative Filtering Algorithms. *International Journal of Computer Sciences and Engineering*. 2019. Vol. 7, No. 8. P. 347–351.

14. Herzog D., Wörndl W. Extending Content-Boosted Collaborative Filtering for Context-aware, Mobile Event Recommendations. *International Conference on Web Information Systems and Technologies*: Proceedings of 12th conference, Rome, Italy, 23–25 April, 2016. P. 293–303.

15. Nath A., Ghosh A., Mitra A. Building a Movie Recommendation System using SVD algorithm. *International Journal of Computer Sciences and Engineering*. 2018. Vol. 6, No. 11. P. 727–729.

16. Mainland G., Morrisett G., Welsh M. Flask: staged functional programming for sensor networks. *ACM SIGPLAN international conference*: Proceeding of the 13th conference, Victoria, BC, 20–28 September, 2008. P. 335–346.

17. Stouffer J., Dwyer G., Aggarwal S. Flask: Building Python Web Services. Birmingham: Packt Publishing, 2017. 770 p.

18. Ashley D. Using Flask and Jinja. *Foundation Dynamic Web Pages with Python*. Berkeley, CA: Apress, 2020. P. 159–181.

19. Newman C. SQLite. Upper Saddle River: Pearson Education, 2004. 313 p.

20. Chauhan N. Implementation of database using python flask framework. *International Journal of Engineering and Computer Science*. 2019. Vol. 8, No. 12. P. 24894–24899.