МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
« » 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

РЕКОМЕНДАЦІЙНА СИСТЕМА ПЕРСОНАЛІЗАЦІЇ КОРИСТУВАЦЬКОГО ДОСВІДУ

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

Здобувач _____Олександр ЄВДОКІМОВ

«___»_____ 2024 р.

Керівник PhD, ст. викладач _____Ігор КАНДИБА

«___»_____ 2024 р.

м. Миколаїв – 2024 рік

Чорноморський національний університет імені Петра Могили

(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Другий (магістерський)
Освітній ступень	Магістр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри інженерії
програмного забезпечення
_____Євген ДАВИДЕНКО
«___»_____2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу здобувача

Євдокімов Олександр Ігорович

(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Рекомендаційна система персоналізації користувацького досвіду»

Затверджена наказом ЧНУ ім. Петра Могили від «04» вересень 2024 р. № 220

2. Строк представлення кваліфікаційної роботи «19» грудня 2024 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні:

Система рекомендацій персоналізації користувацького досвіду.

4. Перелік питань, що підлягають розробці:

– проведення аналізу існуючих рекомендаційних систем та алгоритмів машинного навчання, що використовуються для їх розробки;

– дослідження вимог до рекомендаційної системи для різних типів користувачів;

– вибір і тестування найбільш ефективних алгоритмів для створення персоналізованих рекомендацій;

– проєктування та реалізація рекомендаційної системи з використанням складних бінарних алгоритмів пошуку та створення дерев зв'язків;

– інтеграція рекомендаційної системи у сервіси та її тестування;

– оцінка ефективності системи та внесення покращень на основі зібраних даних.

5. Перелік графічних матеріалів: презентація.

6. Завдання до спеціальної частини

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи

Керівник роботи _____ Ігор КАНДИБА

Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ Олександр ЄВДОКІМОВ

Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Календарний план виконання кваліфікаційної роботи

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Рекомендаційна система персоналізації користувацького досвіду

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання навиконання КМР	13.08.2024	02.09.2024	виконано
2.	Огляд літератури за темою роботи	03.09.2024	06.09.2024	виконано
3.	Складання календарного плану КМР	06.09.2024	09.09.2024	виконано
4.	Аналіз предметної області	09.09.2024	11.09.2024	виконано
5.	Розробка проєктних рішень	11.09.2024	25.09.2024	виконано
6.	Моделювання та конструювання ПЗ	28.09.2024	04.10.2024	виконано
7.	Кодування, тестування та апробаціярозробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	04.10.2024	20.11.2024	виконано
8.	Відгук керівника КМР	20.11.2024	21.11.2024	виконано
9.	Оформлення КМР та презентації	21.11.2024	28.11.2024	виконано
10.	Попередній захист	28.11.2024	28.11.2024	виконано
11.	Рензування	29.11.2024	13.12.2024	виконано
12.	Завершення оформлення КМР та презентації	13.12.2024	19.12.2024	виконано
13.	Захист кваліфікаційної роботи	19.12.2024	20.12.2024	виконано

Розробив здобувач Євдокімов Олександр Ігорович _____

(прізвище, ім’я, по батькові) (підпис)

«_____» 2024р.

Керівник роботи PhD, ст. викладач Кандиба Ігор Олександрович _____

(посада, прізвище, ім’я, по батькові) (підпис)

«_____» 2024р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра

«Рекомендаційна система персоналізації користувацького досвіду»

Здобувач 608м гр.: Євдокімов Олександр Ігорович

Керівник: PhD, ст. викладач Кандиба Ігор Олександрович

Об’єктом кваліфікаційної роботи є процес обробки даних рекомендаційною системою.

Предметом кваліфікаційної роботи є інструментарії штучного інтелекту що використовуються для створення рекомендаційних систем.

Метою роботи є автоматизація процесу створення списку рекомендацій шляхом розробки програмного забезпечення на базі інструментарію штучного інтелекту.

Для досягнення визначеної мети необхідно вирішити такі завдання:

– проведення аналізу існуючих рекомендаційних систем та алгоритмів машинного навчання, що використовуються для їх розробки;

– дослідження вимог до рекомендаційної системи для різних типів користувачів;

– вибір і тестування найбільш ефективних алгоритмів для створення персоналізованих рекомендацій;

– проєктування та реалізація рекомендаційної системи з використанням складних бінарних алгоритмів пошуку та створення дерев зв'язків;

– інтеграція рекомендаційної системи у сервіси та її тестування;

– оцінка ефективності системи та внесення покращень на основі зібраних даних.

Основними проєктними рішеннями є:

– створення інтелектуальної системи моніторингу та аналізу даних рекомендаційною системою для автоматичного виявлення атрибутів рекомендацій і оптимізації процесів персоналізації;

– використання складних бінарних алгоритмів пошуку та створення дерев структур для генерації персоналізованих рекомендацій;

– забезпечення легкої інтеграції під час розробки вебсервісів;

– забезпечення можливості користувачам та розробникам надавати зворотний зв'язок для покращення якості рекомендацій;

– тестування та оцінка ефективності рекомендаційної системи з використанням реальних даних.

Кваліфікаційна магістерська робота складається з вступу, чотирьох розділів, висновків та переліку джерел посилань.

У першому розділі описується процес аналізу предметної області та специфікацій вимог.

У другому розділі моделюються програмні рішення та описуються математичні функції.

У третьому розділі описуються технології та архітектура розробки програмного забезпечення. Також розглядається діаграми UML.

У четвертому розділі проводиться демонстрація функцій, графіків даних, сервісів та компонентів програмного забезпечення.

Магістерська кваліфікаційна робота викладена на 96 сторінок, вона містить 4 розділи, 48 рисунка, 1 таблицю, 25 джерел в переліку посилань.

У висновках описується аналіз розробленої роботи та готових результатів.

Ключові слова: *система рекомендацій на основі спільної фільтрації, матрична факторизація, неявний зворотній зв’язок, матриця корисності, репозиторій даних*, *ембеддинг, оптимізатор градієнтного спуску*.

ABSTRACT

of the Master's Thesis

«Recommendation system for personalizing user experience»

Student of group 608m: Yevdokimov Oleksandr Ihorovich

Supervisor: Phd, s. lecture Kandyba Ihor Oleksandrovich

The object of the qualification work is the data processing process of the recommender system.

The subject of the qualification work is artificial intelligence tools used to create recommender systems.

The purpose of the work is to automate the process of creating a list of recommendations by developing software based on artificial intelligence tools.

To achieve the specified goal, the following tasks must be solved:

– analysis of existing recommendation systems and machine learning algorithms used for their development;

– research of requirements for the recommender system for different types of users;

– selection and testing of the most effective algorithms for creating personalized recommendations;

– design and implementation of a recommendation system using complex binary search algorithms and creation of connection trees;

– integration of the recommendation system into services and its testing;

– assessment of system performance and improvement based on collected data.

The main project solutions are:

– creation of an intelligent system for monitoring and analyzing data by a recommendation system for automatically detecting recommendation attributes and optimizing personalization processes;

– using complex binary search algorithms and creating structure trees to generate personalized recommendations;

– ensuring easy integration during web service development;

– enabling users and developers to provide feedback to improve the quality of recommendations;

– testing and evaluating the effectiveness of the recommender system using real data.

The qualifying master's thesis consists of an introduction, four chapters, conclusions and a list of reference sources.

The first section describes the process of domain analysis and requirements specifications.

In the second section, software solutions are modeled and mathematical functions are described.

The third chapter describes software development technologies and architecture. UML diagrams are also considered.

In the fourth section, the functions, services and components of the software are demonstrated.

The conclusions describe the analysis of the developed work and the finished results.

The master's thesis is 96 pages long, contains 4 sections, 48 figures, 1 table, and 25 sources in the list of references.

Keywords: *recommendation system based on collaborative filtering, matrix factorization, implicit feedback, utility matrix, data repository, gradient descent optimizer.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

ПК – персональний комп’ютер

ОС – операційна система

API – Application programming interface

CRM – customer relationship management

REST – representational state transfer

IT – Information Technology

IDEF – I-CAM DEFinition or Integrated DEFinition Methods

DFD – data flow diagrams

NMF – Non-negative Matrix Factorization

PCA – principal component analysis

SVD – singular value decomposition

SGD – stochastic gradient descent

RMSE – Root Mean Square Error

MAE – Mean Absolute Error

NCF – Neural Collaborative Filtering

WSGI – Web Server Gateway Interface

MF – Matrix factorization

MLP – Multilayer perceptron

UML – Unified Modeling Language

DCG – Discounted Cumulative Gain

NDCG – Normalized Discounted Cumulative Gain

ВСТУП

Інформаційні технології все більше проникають у повсякденне життя, і персоналізація користувацького досвіду стає однією з ключових задач для розробників програмного забезпечення. Рекомендаційні системи є важливою складовою багатьох онлайн-платформ, від музичних та відео-сервісів до інтернет-магазинів, соціальних мереж та стримінгових сервісів. Вони допомагають користувачам знайти релевантний контент, підвищуючи задоволення та збільшуючи взаємодію з платформою. Зважаючи на те, що кількість доступного контенту стрімко зростає, ефективні рекомендаційні системи стають все більш необхідними для покращення користувацького досвіду.

Об’єктом кваліфікаційної магістерської роботи є процес обробки даних рекомендаційною системою.

Предметом кваліфікаційної магістерської роботи є інструментарії штучного інтелекту що використовуються для створення рекомендаційних систем.

Метою роботи є автоматизація процесу створення списку рекомендацій шляхом розробки програмного забезпечення на базі інструментарію штучного інтелекту.

Для досягнення визначеної мети необхідно вирішити такі завдання:

– проведення аналізу існуючих рекомендаційних систем та алгоритмів машинного навчання, що використовуються для їх розробки;

– дослідження вимог до рекомендаційної системи для різних типів користувачів;

– вибір і тестування найбільш ефективних алгоритмів для створення персоналізованих рекомендацій;

– проєктування та реалізація рекомендаційної системи з використанням складних бінарних алгоритмів пошуку та створення дерев зв'язків;

– інтеграція рекомендаційної системи у сервіси та її тестування;

– оцінка ефективності системи та внесення покращень на основі зібраних даних.

На сьогоднішній день багато рекомендаційних систем мають обмежену точність і часто пропонують користувачам нерелевантний контент. Це може знизити задоволення користувачів та їхню взаємодію з платформою. Інтеграція рекомендаційної системи може бути складною, оскільки вона потребує ретельного налаштування для взаємодії з існуючими сервісами. Крім того, процес розробки таких систем є досить складним через необхідність використання складних алгоритмів та обробки великого обсягу даних. Створення ефективної рекомендаційної системи, що використовує сучасні алгоритми машинного навчання, допоможе вирішити ці проблеми, надаючи користувачам персоналізовані рекомендації, які більше зацікавлять їх.

Основними проєктними рішеннями є:

– створення інтелектуальної системи моніторингу та аналізу даних рекомендаційною системою для автоматичного виявлення атрибутів рекомендацій і оптимізації процесів персоналізації;

– використання складних бінарних алгоритмів пошуку та створення дерев структур для генерації персоналізованих рекомендацій;

– забезпечення легкої інтеграції під час розробки вебсервісів;

– забезпечення можливості користувачам та розробникам надавати зворотний зв'язок для покращення якості рекомендацій;

– тестування та оцінка ефективності рекомендаційної системи з використанням реальних даних.

У результаті роботи буде створено рекомендаційну систему, що здатна ефективно персоналізувати користувацький досвід та зручно інтегруватися у систему, підвищуючи ефективність розробки, зменшення об′єму ресурсів, збільшення задоволення та залученість користувачів та розробників.

1 АНАЛІЗ РЕКОМАНДАЦІЙНИХ СИСТЕМ

1.1 Система рекомендацій

Рекомендаційні системи є досить ефективними інструментами для фільтрації великої кількості онлайн-інформації, що стало особливо актуальним через зміну користувацьких звичок, зростання популярності персоналізації та ширший доступ до Інтернету. Попри високу точність сучасних рекомендаційних систем, вони стикаються з низкою викликів, таких як масштабованість, проблема холодного старту, розрідженість даних тощо. Велика кількість доступних методів ускладнює процес вибору підходів під час розробки додатків, орієнтованих на рекомендації. До того ж кожен метод має свої унікальні особливості, переваги та недоліки, що додає складнощів у прийнятті рішень.

Недавні досягнення в галузі технологій разом із поширеністю онлайн-сервісів запропонували більше можливостей для швидшого доступу до величезної кількості онлайн-інформації. Користувачі можуть публікувати відгуки, коментарі та рейтинги для різних типів послуг і продуктів, доступних в Інтернеті. Нещодавні досягнення у всеосяжних обчисленнях призвели до проблеми перевантаження онлайн-даними. Це перевантаження даними ускладнює процес пошуку відповідного та корисного вмісту в Інтернеті. Однак нещодавнє впровадження кількох процедур, що мають менші обчислювальні вимоги, може, однак, скеровувати користувачів до відповідного вмісту дуже простим і швидким способом. Через це розробці рекомендаційних систем приділяється значна увага та кількість часу. Загалом системи рекомендацій діють, як інструменти фільтрації інформації, пропонуючи користувачам відповідний персоналізований вміст або інформацію. Системи рекомендацій насамперед спрямовані на те, щоб зменшити зусилля та час користувача, необхідні для пошуку відповідної інформації в Інтернеті.

В даний час системи рекомендацій все частіше використовуються для великої кількості програм, таких як вебсервісів, книги, електронне навчання, туризм, фільми, музика, електронна комерція, новини, спеціалізовані дослідницькі ресурси,

телевізійні програми тощо. Тому важливо створювати високоякісні та ексклюзивні системи рекомендацій для надання персоналізованих рекомендацій користувачам у різних програмах. Незважаючи на різноманітні досягнення в рекомендаційних системах, поточне покоління рекомендаційних систем потребує подальших удосконалень, щоб надати більш ефективні рекомендації, застосовні до більш широкого діапазону застосувань. Потрібні додаткові дослідження існуючих останніх робіт щодо систем рекомендацій, які зосереджені на різноманітних застосуваннях.

Кілька існуючих оглядів літератури в цій галузі охоплюють лише частину статей або зосереджуються лише на окремих аспектах, таких як оцінка системи. Таким чином, вони не надають огляду сфери застосування, алгоритмічної категоризації або визначення найбільш перспективних підходів. Крім того, в оглядових статтях часто нехтують аналізом опису набору даних і використовуваних платформ моделювання.

По суті, системи рекомендацій мають справу з двома сутностями – користувачами та елементами, де кожен користувач дає оцінку (або значення переваги) елементу (або продукту). Оцінки користувачів зазвичай збираються за допомогою неявних або явних методів. Неявні оцінки збираються опосередковано від користувача через взаємодію користувача з елементами. Явні оцінки, з іншого боку, даються безпосередньо користувачем шляхом вибору значення на деякій кінцевій шкалі балів або позначених інтервальних значень. Наприклад, вебсайт може отримувати неявні оцінки для різних елементів на основі даних потоку клацання або часу, який користувач проводить на вебсторінці тощо. Більшість систем рекомендацій збирають оцінки користувачів як явними, так і неявними методами. Ці відгуки або оцінки, надані користувачем, упорядковуються в матрицю елементів користувача, яка називається матрицею корисності.

1.2 Типи систем рекомендацій

Системи рекомендацій загалом поділяються на три різні типи, а саме:

– системи рекомендацій на основі вмісту;

– системи рекомендацій спільної фільтрації;

– гібридні системи рекомендацій.

Схематичне зображення різних типів рекомендаційних систем наведено на рисунку 1.1.

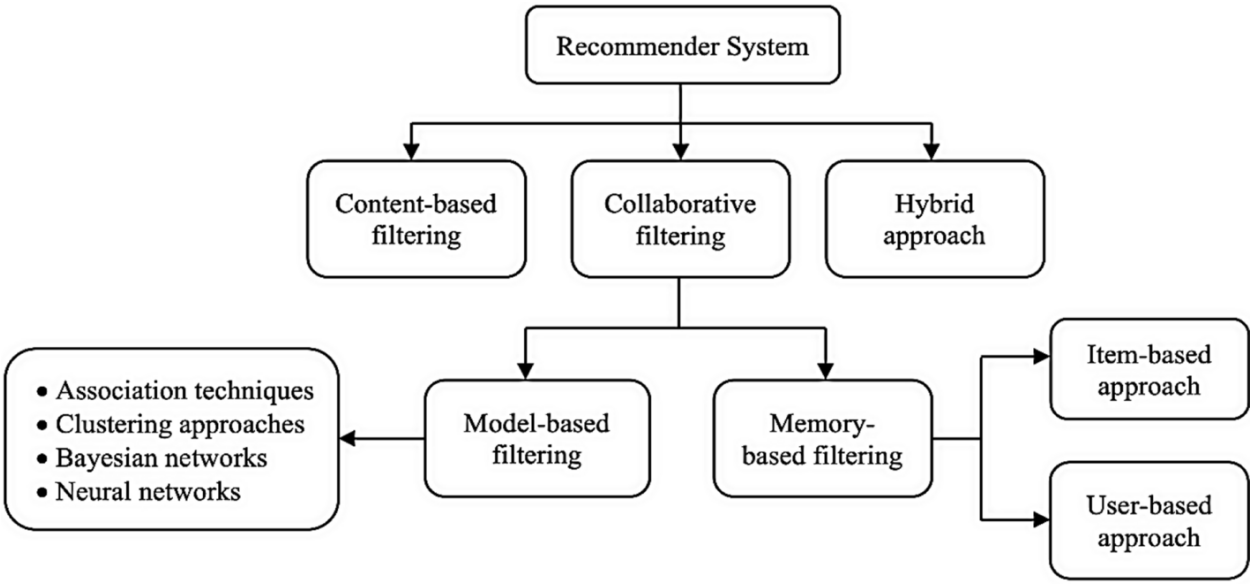

Рисунок 1.1 – Схема типів систем рекомендацій

Рекомендаційна система на основі вмісту

Рекомендаційні системи, засновані на вмісті, використовують інформацію про характеристики елементів та дані, зібрані з різних профілів цих елементів, на основі їх опису або властивостей. Наприклад, у випадку книг такими характеристиками можуть бути автор чи видавець, а у випадку фільмів – режисер чи актори. Коли користувач позитивно або негативно оцінює певний товар чи об'єкт, інші елементи з подібними характеристиками об'єднуються, формуючи профіль користувача. Усі елементи, які користувач оцінив позитивно, групуються в єдиний профіль. Після цього система рекомендує користувачеві елементи, що відповідають цьому профілю, як показано на рисунку 1.2.

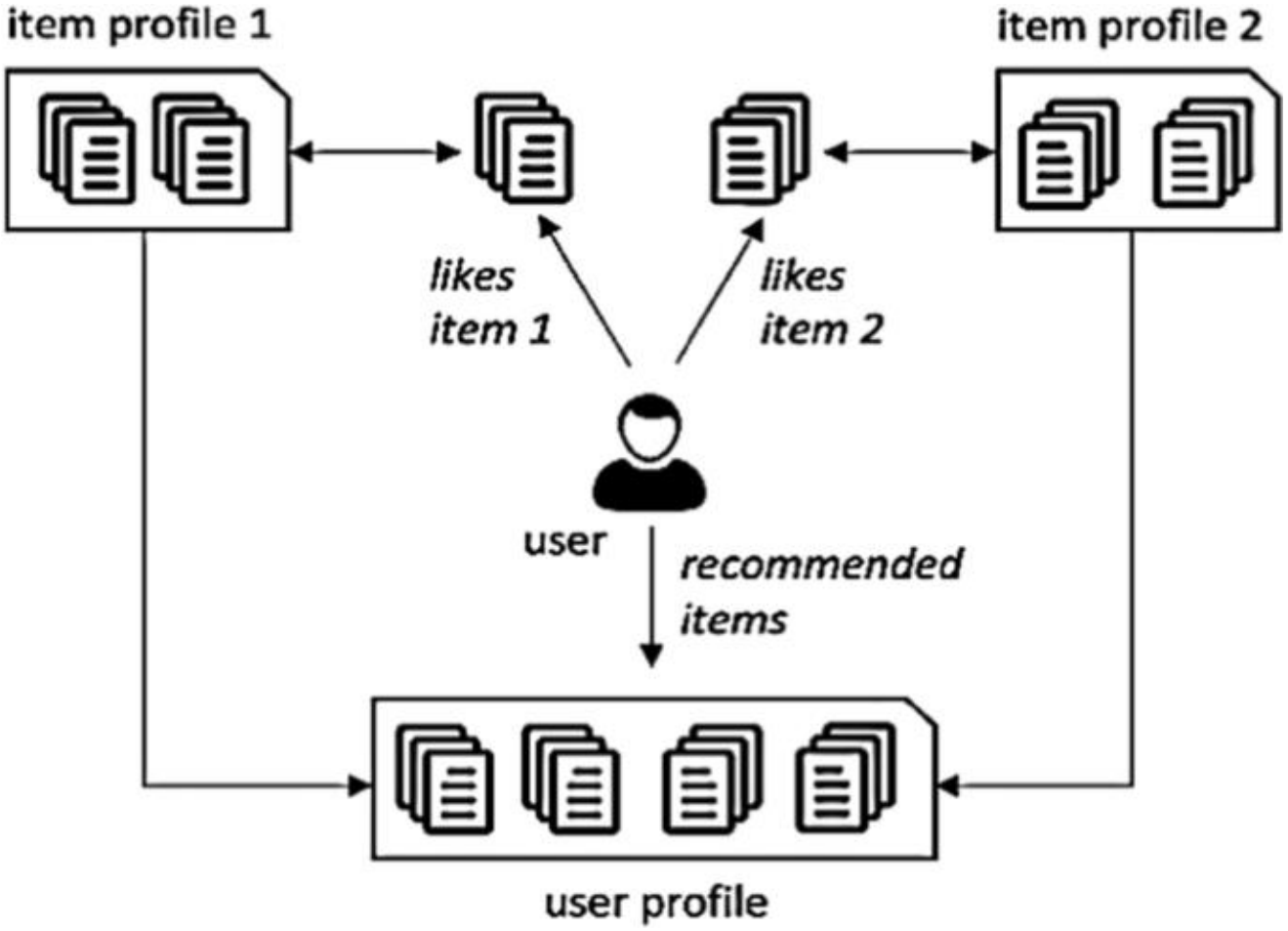

Рисунок 1.2 – Рекомендація на основі вмісту

Система рекомендацій засновані на основі спільної фільтрації

У системах спільної фільтрації використовується міра подібності між користувачами. Цей метод починається з визначення групи або колекції користувачів X, чиї вподобання, симпатії та антипатії схожі на вподобання користувача A. Група X називається «сусідством» A. Нові елементи, які користуються популярністю серед більшості користувачів із сусідства X, рекомендуються користувачеві A. Ефективність алгоритму спільної фільтрації залежить від здатності точно визначити сусідство для цільового користувача.

Традиційно такі системи мають проблеми холодного старту та конфіденційності через необхідність обміну даними користувача. Водночас підходи спільної фільтрації не вимагають знань про характеристики товару для формування рекомендацій. Більше того, вони дозволяють розширити існуючі інтереси користувача, відкриваючи нові елементи. Методи спільної фільтрації

поділяються на два основні типи: підходи, засновані на пам’яті, та підходи, засновані на моделі.

Методи спільної фільтрації, що базуються на пам’яті, пропонують нові елементи, враховуючи переваги користувачів зі свого «сусідства». Ці підходи безпосередньо використовують матрицю корисності для прогнозування. Першим кроком є створення моделі, яка функціонує як алгоритм, що приймає матрицю корисності як вхідні дані. Рекомендація здійснюється через функцію f (визначена модель, профіль користувача), де профіль користувача є частиною матриці корисності.

Методи спільної фільтрації на основі пам’яті поділяються на два типи. Перший тип це спільна фільтрація на основі користувачів. У цьому підході оцінка нового елемента для користувача розраховується шляхом пошуку інших користувачів із подібними вподобаннями, які вже оцінювали цей самий елемент. Якщо новий елемент отримує позитивні відгуки від цих користувачів, він рекомендується цільовому користувачеві. Цей метод ілюструється на рисунку 1.3.

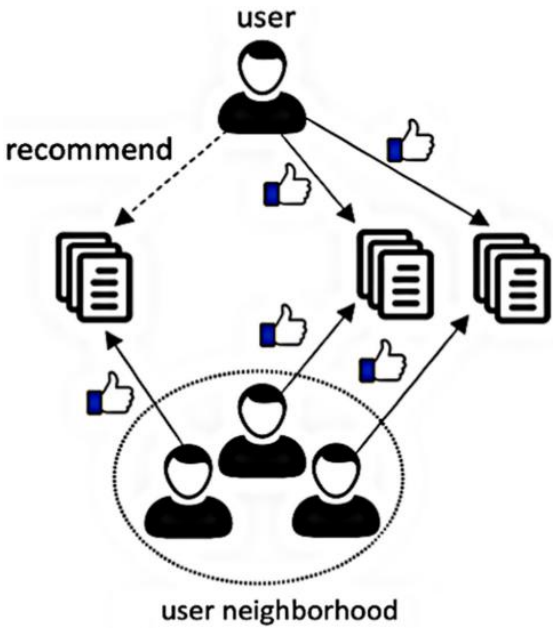

Рисунок 1.3 – Система рекомендацій на основі спільних даних

Та другий метод – спільна фільтрація на основі елементів. У цьому підході формується «сусідство» елементів, яке включає всі подібні елементи, що вже були оцінені користувачем. Як показано на рисунку 1.4, оцінка користувача для нового

елемента прогнозується шляхом обчислення середнього значення всіх оцінок, наявних у схожому «сусідстві» елементів.

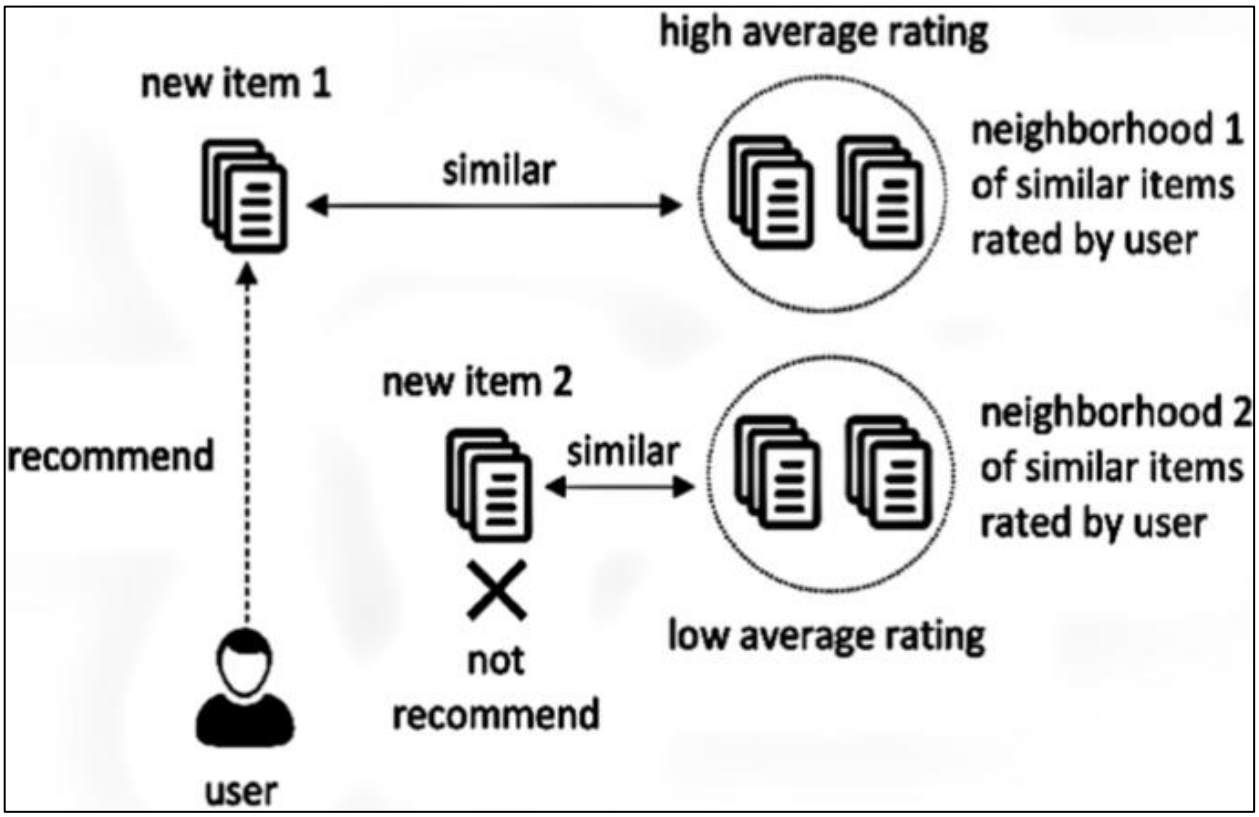

Рисунок 1.4 – Схема обчислення середнього значення всіх оцінок

Модельні системи рекомендацій використовують алгоритми інтелектуального аналізу даних та машинного навчання для створення моделей, які прогнозують оцінки користувачів для товарів, що ще не оцінювалися. На відміну від підходів, які потребують повного набору даних для обчислення рекомендацій, ці системи витягують важливі ознаки з даних, щоб побудувати модель. Цей метод відомий як модельний підхід.

Процес складається з двох основних етапів:

1. створення моделі, яка базується на навчальних даних;

2. прогнозування оцінок через функцію (f), яка використовує модель та профіль користувача як вхідні дані.

Модельні підходи дозволяють робити рекомендації без необхідності додавати профіль нового користувача до матриці корисності. Система здатна генерувати рекомендації навіть для користувачів, які не були враховані під час створення моделі. Такі системи особливо ефективні для групових рекомендацій,

оскільки можуть швидко пропонувати групу елементів завдяки попередньо навченій моделі. Точність таких систем залежить від якості алгоритму машинного навчання, використаного для побудови моделі. Модельні методи вирішують традиційні проблеми рекомендаційних систем, як-от розрідженість даних і масштабованість, через використання технік зменшення розмірності та ефективних методів навчання моделей.

Система рекомендацій на основі гібридної фільтрації

Гібридна методика об’єднує два або більше рекомендаційних методів, щоб подолати обмеження кожного з них. Поєднання різних технік може реалізовуватись різними способами. Гібридний алгоритм може інтегрувати результати, отримані окремими методами, або використовувати фільтрацію на основі вмісту в поєднанні зі спільним методом. Також можливо застосувати техніку спільної фільтрації всередині методів, орієнтованих на вміст. Таблиця 1.1 демонструє основні підходи до гібридних методів фільтрації.

Таблиця 1.1 – Гібридні методи фільтрації

Гібридні методи	Опис
Мета-рівень	Попередньо вивчена модель використовується як вхідні дані для іншої системи рекомендацій
Поєднання ознак	Характеристики однієї системи рекомендацій вводяться в іншу
Розширення функцій	Результат однієї моделі застосовується як вхідні дані для іншої
Змішана гібридизація	Результати різних систем рекомендацій змішуються, а об’єднаний результат надається як рекомендація

Кінець таблиці 1.1

Каскадна гібридизація	Одна система імпровізує результат іншої
Перемикання гібридизації	Вибирає одну рекомендовану модель на основі поточних вимог
Зважена гібридизація	Рейтинги різних методів узагальнюються для обчислення однієї рекомендації

Це гібридне включення різних методів, як правило, призводить до підвищення продуктивності та підвищення точності в багатьох додатках із рекомендаціями. Деякі з підходів гібридизації: мета-рівень, розширення ознак, комбінація ознак, змішана гібридизація, каскадна гібридизація, гібридизація з перемиканням і зважена гібридизація [1].

1.3 Аналіз аналогів

Характеристики Algolia Recommend

Назва: Algolia Recommend [2]

Розробник: Algolia

Архітектура: API-based architecture

Мова реалізації: JavaScript, Python

Перелік функцій, характеристик:

1. API для персоналізованих рекомендацій продуктів;
2. автоматизація «cross-sell» і «upsell» для збільшення продажів;
3. є можливість сортувати товари за категоріями;
4. інтеграція з пошуковими алгоритмами Algolia;
5. масштабування залежно від кількості запитів.

Аналіз переваг та недоліків

Переваги:

1. швидка інтеграція через API;

2. потужні та точні алгоритми пошуку;

3. підтримка «cross-sell» і «upsell»;

4. персоналізовані рекомендації;

5. можливість фільтрації за категоріями;

Недоліки:

1. висока вартість при великій кількості запитів;

2. обмежені можливості для великих масивів даних;

3. можливість затримок при великому трафіку;

4. потребує додаткової технічної підтримки;

5. вимагає складного налаштування для точності результатів.

Характеристики Dynamic Yield

Назва: Dynamic Yield [3]

Розробник: Dynamic Yield Ltd

Архітектура: Cloud-based architecture

Мова реалізації: Node.js, Java

Перелік функцій, характеристик:

1. автоматична персоналізація рекомендацій;

2. аналіз поведінки користувача для покращення результатів;

3. підтримка A/B тестування;

4. інтеграція з CRM та іншими маркетинговими платформами;

5. підтримка рекомендацій через різні платформи (веб, мобільні додатки);

6. збір та обробка великих даних для кращої персоналізації;

7. підтримка динамічного контенту;

8. постійне оновлення даних для реальних змін у рекомендаціях;

9. гнучке управління контентом;

10. автоматизація сегментації аудиторії.

Аналіз переваг та недоліків

Переваги:

1. підтримка багатоканальної персоналізації;

2. можливість аналізу поведінки користувачів;

3. широкий набір інструментів для рекомендацій;

4. інтеграція з багатьма платформами;

5. потужні алгоритми машинного навчання.

Недоліки:

1. складна інтеграція;

2. висока вартість для малого бізнесу;

3. обмежений контроль над алгоритмами;

4. вимагає багато часу для налаштування;

5. не завжди забезпечує точність рекомендацій.

Характеристики Nosto

Назва: Nosto [4]

Розробник: Nosto

Архітектура: Cloud-based

Мова реалізації: PHP, Python

Перелік функцій, характеристик:

1. створення персоналізованих рекомендацій для інтернет-магазинів;

2. автоматизація рекомендаційних алгоритмів;

3. оптимізація контенту через A/B тестування;

4. підтримка персоналізованих маркетингових кампаній;

5. надання динамічних рекомендацій на основі поведінки користувачів;

6. інтеграція з платформами електронної комерції;

7. автоматично оновлюйте рекомендації на основі доступного асортименту;

8. рекомендації щодо перехресних продажів і додаткових продажів;

9. аналіз ефективності персоналізованих рекомендацій;

10. реалізація рекомендацій в режимі реального часу.

Аналіз переваг та недоліків

Переваги:

1. легка інтеграція;

2. можливість A/B тестування;

3. простий інтерфейс користувача;

4. автоматичні оновлення продуктів;

5. гнучке налаштування рекомендацій.

Недоліки:

1. обмежена аналітика;

2. висока вартість;

3. потреба в додатковій технічній підтримці;

4. обмежені функції персоналізації для різних платформ;

5. менше можливостей для кастомізації рекомендацій.

Аналіз системи, що розробляється.

Основні задачі:

1. збір даних про поведінку користувачів;

2. персоналізація рекомендацій на основі вподобань користувача;

3. інтеграція з наявними системами та платформами;

4. обробка великих обсягів даних для навчання моделей;

5. автоматичне оновлення рекомендацій у реальному часі;

6. підтримка різних мов та регіональних налаштувань;

7. аналітика ефективності рекомендацій;

8. реєстрація та авторизація користувачів;

9. рейтинг та оцінка рекомендацій користувачем;

10. підтримка кросплатформенних рекомендацій (десктопні, мобільні додатки).

Користувачі системи:

– User (користувач);

– Data scientist;

– Web Server;

– Developer.

Special Requirements:

1. можливість адаптації системи для нових ринків;

2. підключатися за різними протоколами;

3. використовувати сервер як основний сервер сервісу для роботи з рекомендаціями.

Technology and Data Variations List:

– розробник повинен мати навички роботи з менеджером пакетів;

– обробка даних у хмарному середовищі для масштабування.

Frequency of Occurrence: система працює (24/7) якщо правильно налаштувати.

Miscellaneous (Open Issues):

– налаштування інтерфейсу за бажанням користувача;

– інтеграція з іншими додатками для поліпшення рекомендацій.

Основні таблиці БД:

– users (користувачі);

– items (товари або контент);

– models (моделі);

– templates (готові шаблони);

– recommendations (рекомендації);

– reports of data science (звіти аналітиків);

– processedData (предоброблені дані);

Засоби апаратної та програмної реалізації:

Мова програмування: Python;

Фреймворки та бібліотеки: TensorFlow, Flask, Keras, Matplotlib, Pandas, Numpy, Graphene-python;

Платформа розгортання: Heroku;

База даних: PostgreSQL;

Вихідні дані:

– оновленні шаблони моделей;

– персоналізовані рекомендації (таблиці з бази даних або завантажені дата сети);

– звіти аналітики про модель.

1.4 Специфікація вимог

Призначення та межі проєкту:

– призначення застосунку: створення рекомендацій для персоналізації користувацького досвіду та моделювання рекомендацій, створення звітів та графіків для аналізу;

– погодження: документація Python, Flask, TensorFlow, Kensar, PostgreSQL, GraphQL, Numpy, Pandas, Matplotlib, Graphene-Python;

– межі проєкту: розробка системи рекомендацій, створення стратегій рекомендацій та шаблонів, попередня обробка та нормалізація даних, створення звітів, аналіз взаємодії користувачів, можливість налаштування системи інструментами консолі та змінних оточення (.env).

Загальний опис:

– сфера застосування: e-commerce, IT, сфера програмування, медіа сфера, сфера продаж;

– характеристики користувачів: наявність доступу до інтернету, ПК або смартфон;

– характеристики розробника: наявність доступу до інтернету, ПК або смартфон, володіння навичками управлінням пакетами, аналізу даних, розробки алгоритмів, наявність доступу до віртуальної машини, рекомендовано з процесором Intel Xeon Scalable 3-го покоління та відеокартою Cisco NVidia Tesla M10 32Gb GDDR5 PCIe;

– структура системи: вебсервер WSGI, база даних.

Функції системи:

– завантаження товарів або контенту у форматі json, xml, html, csv;

– оптимізація набор даних за допомогою етапів попередньої обробки;

– генерація персональних рекомендацій на основі матриці суміжності та нейронного навчанню моделі;

– оцінка точності моделі;

– створення шаблонів рекомендацій для розробника сервісу;

– візуалізація статистики та графіків рекомендації;

– інтеграція з іншими сервісами за допомогою REST запитів, GraphQL запитів.

Вимоги до програмного забезпечення:

– архітектура: Microservices architecture;

– системне ПЗ: ОС Windows, Ubuntu;

– мови розробки та технології розробки: Python, Flask, TensorFlow, Kensar, PostgreSQL, GraphQL, Numpy, Pandas, Matplotlib, Graphene-Python;

– мережне програмне забезпечення: використання запитів GraphQL та REST;

– програмне забезпечення ведення інформаційної бази: PostgreSQL для збереження даних.

Вимоги до зовнішніх інтерфейсів:

– інтерфейс користувача: консоль розробника;

– апаратний інтерфейс: Intel Xeon Scalable 3-го покоління, Cisco NVidia Tesla M10 32Gb GDDR5 PCIe, SSD-диск розміром 100 – 200 ГБ;

– програмний інтерфейс: GraphQL API, REST API;

– комунікаційний протокол: TCP/IP.

Властивості програмного забезпечення:

– доступність: система повинна легко інтегруватися до вже існуючої системи за допомогою встановлення менеджеру пакетів, або завантаження на сервіс;

– супроводжуваність: легкість внесення змін і оновлень, з документованим кодом для підтримки;

– переносимість: система має бути здатною до розгортання на різних платформах і середовищах;

– продуктивність: висока швидкість обробки запитів і генерування рекомендацій;

– надійність: система повинна бути стійкою до збоїв та обробляти виключення;

– безпека: дані користувачів захищені асиметричним шифруванням RSA або ACC (за вибором розробника).

Висновок до розділу 1

У результаті виконання розділу 1 розглянуто базові поняття та основні принципи роботи систем рекомендацій. Встановлено, що рекомендаційні системи є ключовим елементом у багатьох галузях, сприяючи покращенню користувацького досвіду завдяки персоналізованому підходу до подання інформації. Вони допомагають ефективно обробляти великі обсяги даних, пропонуючи релевантний контент на основі поведінки користувача.

Проаналізовано типи рекомендаційних систем, зокрема систем на основі вмісту, спільної фільтрації та гібридних підходів. Було відзначено, що кожен із цих методів має свої переваги та недоліки, які визначають вибір підходу залежно від специфіки задачі. Також розглянуто роль матриці корисності, яка виступає основою для побудови моделей рекомендацій.

Проведено аналіз аналогів і визначено ключові вимоги до системи, що розробляється. Зокрема, система повинна забезпечувати ефективний збір даних, обробку великих обсягів інформації, персоналізацію рекомендацій на основі вподобань користувачів, а також зручну інтеграцію з іншими сервісами.

Визначено специфікації вимог які описують функції системи, оптимізацію наборів даних, створення персональних рекомендацій за допомогою матричної факторизації та нейронних мереж. Окрім цього, вимоги охоплюють описують інтеграцію з REST і GraphQL API для забезпечення кросплатформеність та доступності.

2 РОЗРОБКА ПРОЄКТНИХ РІШЕНЬ

2.1 Моделювання бізнес процесів рекомендаційної системи

Побудова IDF0 діаграми

На рисунку 2.1 зображено IDF0 рекомендаційної системи першого рівня, яка показує високорівневий процес роботи системи, тобто основні етапи і які дані надаються для обробки [5].

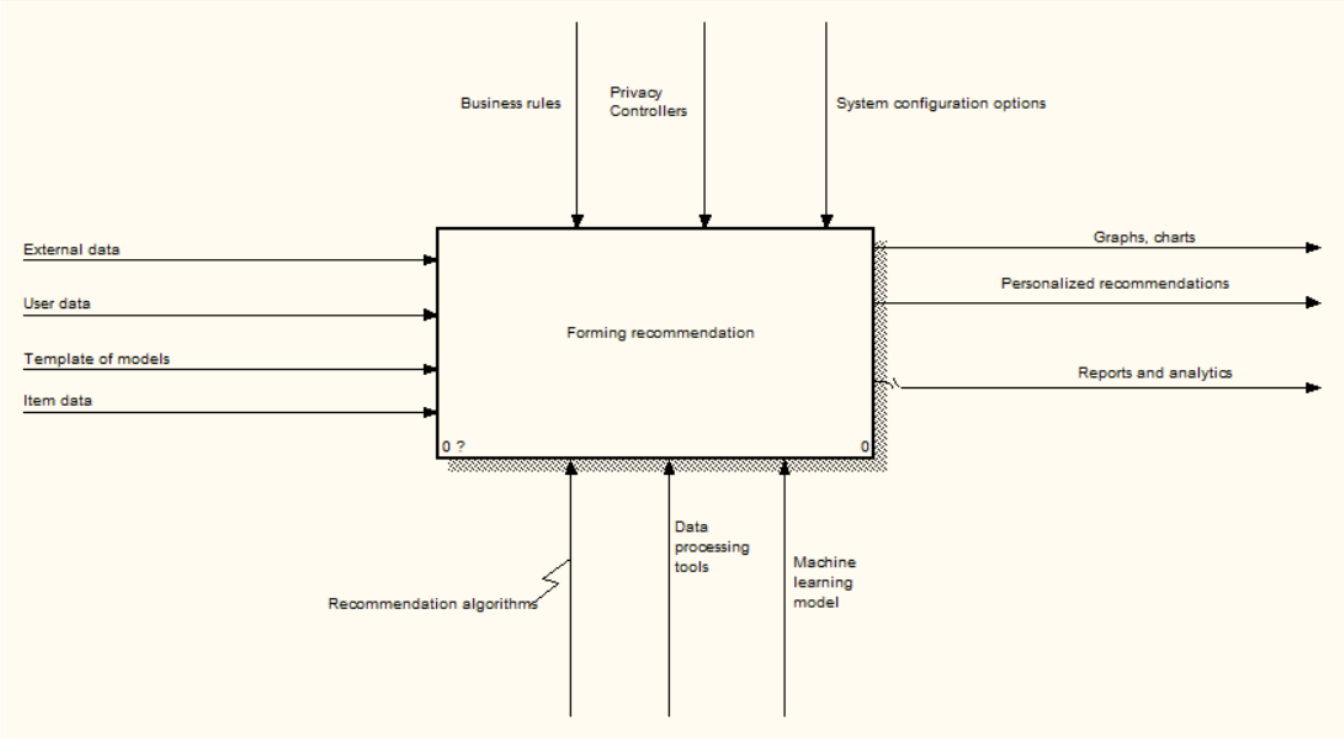

Рисунок 2.1 – Діаграма IDF0 рекомендаційної системи

На рисунку 2.2 продемонстровано декомпозицію рекомендаційної системи. Розбиття на підпроцеси передбачає детальний огляд, як функціонує система. Розділимо процес рекомендування продукту на більш детальні процеси такі як: збір даних, попередня обробка даних, побудова та тренування моделі, формування та надання рекомендацій, візуалізація даних та результатів.

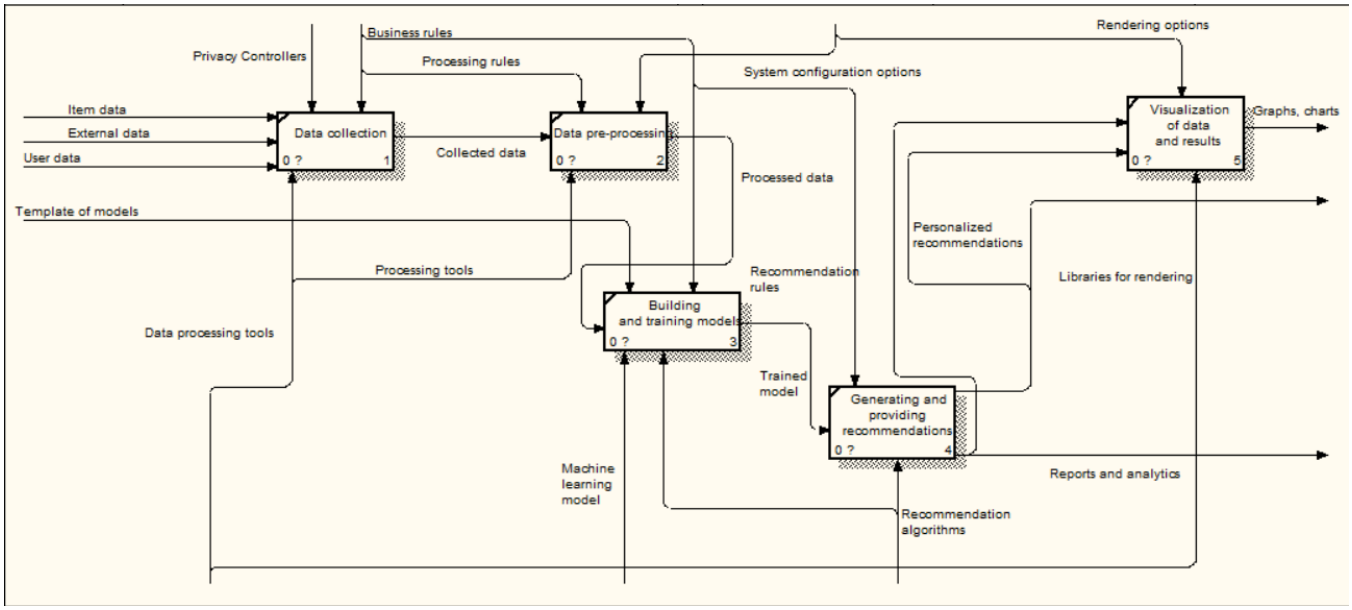

Рисунок 2.2 – Діаграма IDF0 другого рівня

Побудова DFD діаграми

На рисунку 2.3 зображено DFD діаграма. DFD на концептуальному рівні для рекомендаційної системи описує загальну структуру процесу та потоки даних без деталізації внутрішньої роботи кожного компонента. На цьому рівні треба зосередитися на тому, як основні частини системи взаємодіють між собою та які дані переміщуються між ними.

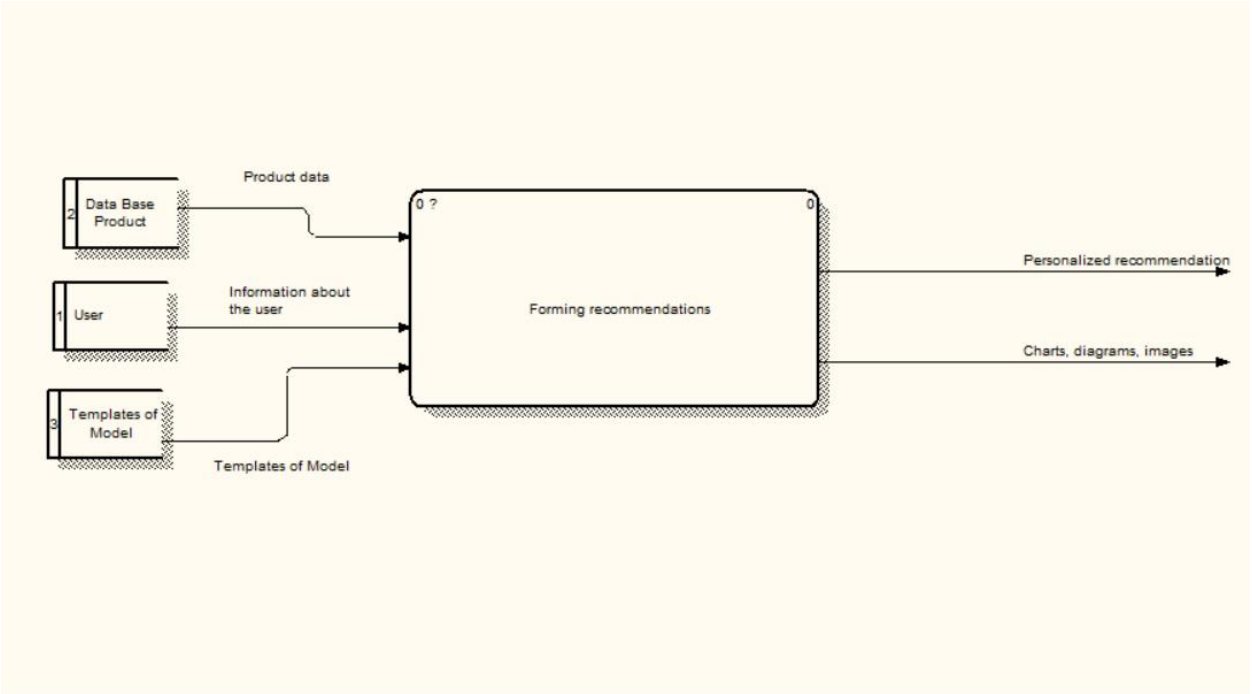

Рисунок 2.3 – Діаграма DFD на концептуальному рівні

Рисунок 2.4 демонструє декомпозицію яка відображає логіку перетворення даних у системі в кожному процесі, описує. Видні вхідні, проміжні, вихідні дані в кожному процесі, які надходять від зовнішніх сутностей.

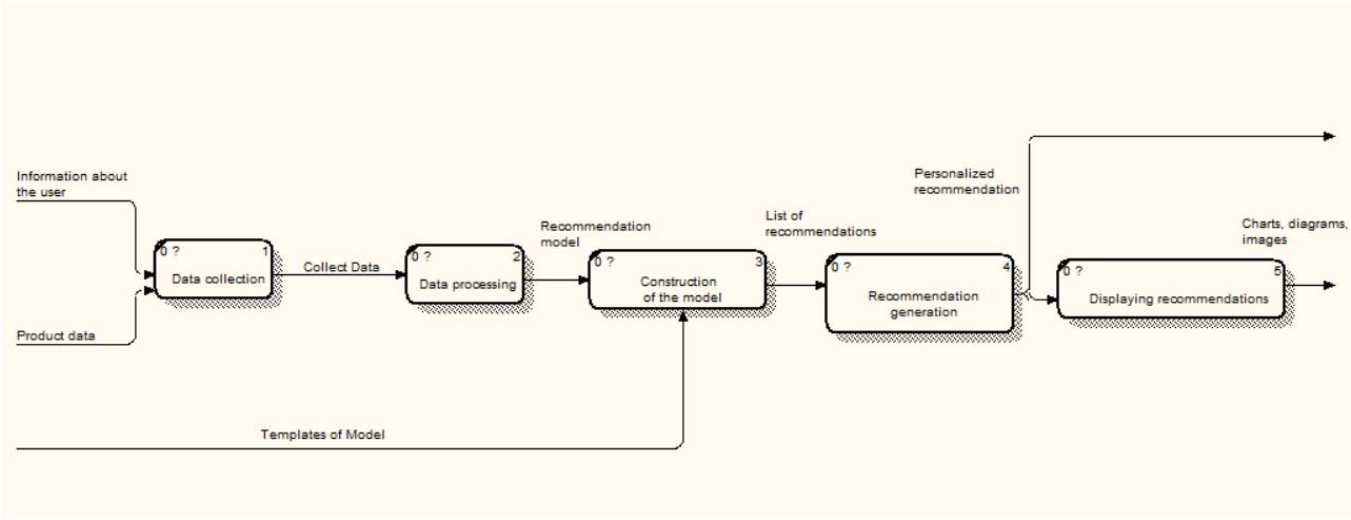

Рисунок 2.4 – Декомпозиція діаграми DFD

Цей процес дозволяє автоматизувати та персоналізувати рекомендації для користувача, забезпечуючи ефективну інтеграцію з різними сервісами.

2.2 Моделювання фільтрації на основі моделі

Фільтрація на основі моделей є один з основних підходів до рекомендацій в спільних фільтраціях. Цей метод використовує дані про оцінки користувачів для навчання моделі, що може передбачити невідомі рейтинги. За допомогою цього підходу можна зосередитися на побудові узагальнених моделей, які можуть використовуватися для рекомендацій. На відміну від алгоритмів, що відсилаються на базу даних що зберігається в пам’яті, які порівнюють поточного користувача або елемент з іншими для кожного запиту, модельні алгоритми обчислюють та зберігають модель попередньо.

Ймовірнісні моделі

Підхід в якому рекомендаційна система намагається оцінити ймовірність того, що первинний користувач оцінить конкретний елемент певним чином це підхід ймовірнісних моделей [6].

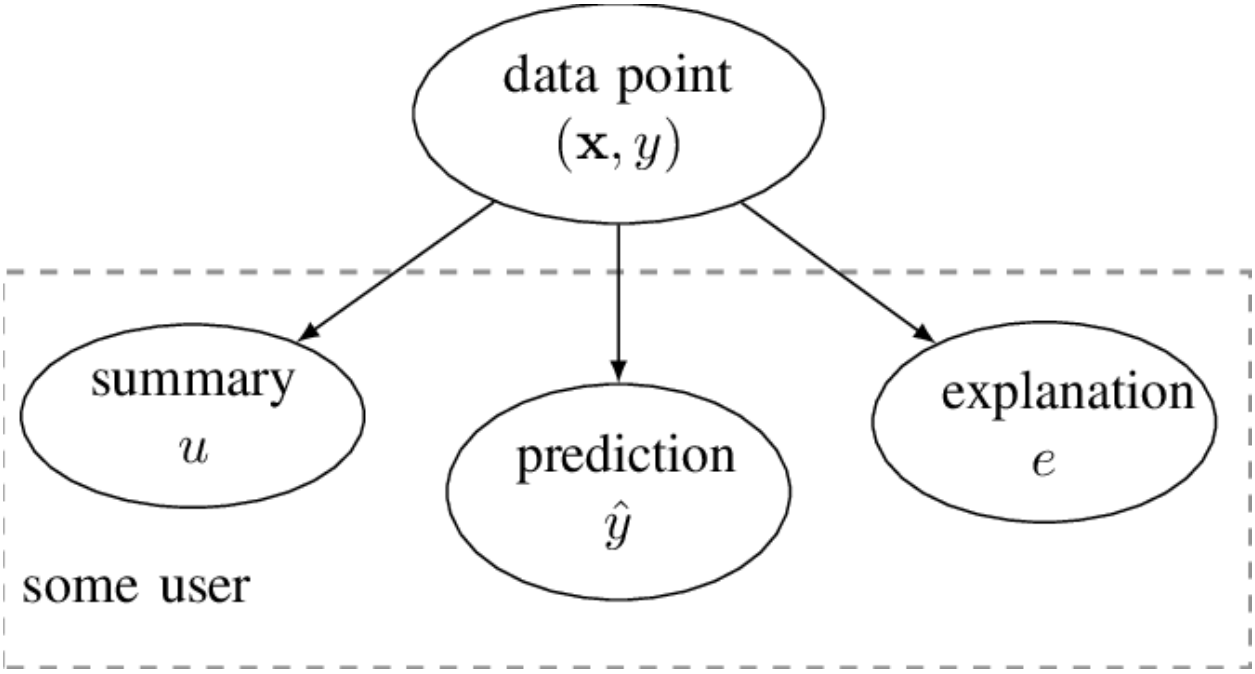

Рисунок 2.5 – Проста ймовірнісна модель

Якщо рейтинги користувача представлені у вигляді чисел 0 до m, тоді передбачуваний рейтинг «u» для елемента «i» можна розрахувати за формулою (1):

$$\widehat{r_{ui}} = \sum_{j=0}^{m} \Pr\left(r_{u,i} = j \middle| r_{u,k}, k \in R_u\right) \times j, \quad (1)$$

де R_u – це набір рейтингів користувача u;

$\Pr\left(r_{u,i} = j \middle| r_{u,k}, k \in R_u\right)$ – ймовірність того, що активний користувач «u» поставить рейтинг «j» елементу «i», з урахуванням його попередніх оцінок.

До популярних підходів для оцінки ймовірності відносять байєсівські мережі та методи кластеризації.

Матрична факторизація

Матрична факторизація – найпопулярніший з сучасних методів фільтрації на основі моделей. Цей метод використовує методи розкладу матриць для виявлення прихованих факторів, які пояснюють взаємозв’язки між користувачами та елементами. Матрична факторизація виготовує такі техніки як сингулярний розклад, щоб спростити дані і виявити схожість між користувачами та елементами.

Нехай матриця рейтингів R має розмір |U| × |I|, де U – кількість користувачів, а I – кількість елементів, f – кількість прихованих факторів, таких що їхній добуток, який можна порахувати за формулою (2) яка наближає матрицю [7]:

$$R \approx P \times Q^T = \hat{R}, \quad (2)$$

де P – перша матриця (розміру |U| × f);

Q – друга матриця (розміру |I| × f);

f – кількість прихованих факторів.

Далі залежно вимог та властивостей даних, обирається один із методів розкладання.

Поділ матриці на факторні матриці

PCA – техніка зниження розмірності, яка шукає лінійні комбінації вихідних ознак, що максимально варіюються в наборі даних. Метою PCA є перетворення початкових змінних у новий набір змінних, які несуть основну інформацію. У рекомендаційних систем PCA допомагає зменшити кількість ознак, що представляють користувачів і елементи, виділяючи ключові фактори, які найбільше впливають на оцінки.

SVD – один з методів для матричної факторизації. Як показано формулою (3), цей метод розкладає матрицю рейтингів R на три матриці [8]:

$$R = U\Sigma V^T, \quad (3)$$

де U – матриця, що представляє користувачів у просторі прихованих факторів;

∑ – діагональна матриця, що містить сингулярні значення;

V – матриця, що представляє елементи у просторі прихованих факторів.

За допомогою методу SVD можна зберігати найважливішу інформацію, видаляючи «шум». Це поліпшує точність рекомендацій особливо в ситуаціях із великими та розрідженими матрицями рейтингів.

NMF (Non-negative Matrix Factorization) – метод який розкладає матриці на дві невід’ємні матриці P та Q, де P представляє профілі користувачів, а Q – характеристики елементів. Всі значення в матрицях невід’ємні. Значення від нуля

до нескінченності сприяють інтерпретуванню факторів, оскільки у матриці чітко вказують на наявність чи відсутність характеристик.

Навчання моделі

Головним завданням у підході матричної факторизації є навчання цих матриць P та Q, тобто знаходження відповідних факторних векторів для кожного користувача та елемента. Для цього найчастіше використовується методи стохастичного градієнтного спуску (SGD) або Adam.

SGD – один з методів оптимізації матриць P та Q шляхом ітеративного оновлення значень з урахуванням функцій втрат. Стохастичний градієнтний спуск рахується за формулою (4):

$$\theta_{t+1} = \theta_t - \eta \times \nabla_\theta J(\theta_t; x_i; y_i), \quad (4)$$

де θ_t – параметри моделі (наприклад, ваги) на t-й ітерації;

η – швидкість навчання (learning rate), що визначає розмір кроку оновлення;

$\nabla_\theta J(\theta_t; x_i; y_i)$ – градієнт функції втрат J (θ) відносно параметрів θ, обчислений для одного прикладу (x_i, y_i);

t – номер ітерації.

Вектор $(\nabla_\theta J)$ який вказує напрямок найбільшого зростання функції втрат. У SGD градієнт обчислюється для одного випадкового прикладу (або міні-батчу), що є відмінністю від класичного градієтного спуску. Швидкість навчання (η) контролює, наскільки швидко параметри оновлюється. Зазвичай невелике значення η призводить до повільної збіжності, але більш точної, тоді як велике значення η може викликати нестабільність. Значення параметрів θ_t оновлюється в напрямку, протилежному до градієнта (щоб мінімізувати функцію втрат).

Для обчислення з використанням міні-батчу застосовується формула (5):

$$\theta_{t+1} = \theta_t - \eta \times \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta J(\theta_t; x_i; y_i), \quad (5)$$

де m – розмір міні-батчу (кількість прикладів у підмножині).

Adam (Adaptive Moment Estimation) – один із найпопулярніших алгоритмів оптимізації, що використовується для навчання нейронних мереж. Він поєднує переваги двох інших методів: AdaGrad і RMSProp. Алгоритм особливо ефективний для роботи з великими обсягами даних і великою кількістю параметрів моделей. Adam адаптує швидкість навчання для кожного параметра на основі першого моменту (градієнта) і другого моменту (квадрату градієнта). Це дозволяє оптимізатору ефективно справлятися як із низькорівневими, так і з високорівневими градієнтами.

Спершу треба обчислити градієнт за формулою (6):

$$g_t = \nabla_\theta J(\theta_t), \quad (6)$$

де $J(\theta)$ – функція втрат.

Далі рахуємо оновлення першого моменту (m_t) за формулою (7):

$$m_t = \beta_1 \times m_{t-1} + (1 - \beta_1) \times g_t, \quad (7)$$

де β_1 – коефіцієнт для першого моменту;

m_0 – початковий момент, який дорівнює 0;

g_t – вектор часткових похідних функції втрат $J(\theta)$ відносно параметрів моделі θ.

Обчислення оновлення другого моменту (v_t) за формулою (8):

$$v_t = \beta_2 \times v_{t-1} + (1 - \beta_2) \times {g_t}^2, \quad (8)$$

де β_2 – гіперпараметр, коефіцієнт для другого моменту;

v_0 – гіперпараметр, початковий момент, який дорівнює 0;

${g_t}^2$ – квадратичне значення поточного градієнту.

m_t та v_t на перших ітераціях мають зміщення до нуля, тому виконується корекція. Корекція зміщень виконується за формулами:

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t}, \quad (9)$$

де β_1 – коефіцієнт згладжування для першого моменту, піднесений до степеню t;

m_t – згладжене середнє градієнтів, яке «пам'ятає» напрямок градієнта з попередніх ітерацій.

$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t}, \quad (10)$$

де β_2 – коефіцієнт згладжування для другого моменту, піднесений до степеню t;

v_t – згладжене середнє квадратів градієнтів, яке визначає, наскільки великий градієнт у різних напрямках.

Отже отримаємо фінальну формулу обчислення оновлення параметрів у Adam (11):

$$\theta_{t+1} = \theta_t - \eta \times \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon}, \quad (11)$$

де β_2 – коефіцієнт згладжування для другого моменту, піднесений до степеню t;

v_t – згладжене середнє квадратів градієнтів, яке визначає, наскільки великий градієнт у різних напрямках.

2.3 Оцінка якості моделі

Оцінка прогнозованих рейтингів

Кожен рядок матриці P є векторним, що представляє приховані вподобання користувача «u», а кожен рядок Q – вектором, що представляє приховані характеристики елемента «i». Прогнозований рейтинг користувача «u» для елемента «i» можна обчислити за формулою (12) – скалярний добуток відповідних векторів:

$$\widehat{r_{u,i}} = p_u \times q_i^T, \quad (12)$$

де $\widehat{r_{u,i}}$ – вказує на ймовірний рейтинг користувача для певного елемента.

Вибір значення a f (кількість прихованих факторів) є важливим, оскільки занадто маленьке значення може призвести до втрати інформації, а занадто велике

до перенавчання. Коли намагаємся навчити методом SGD гіперпараметри впливають на швидкість та точність збіжності. Також використовують регуляризацію для уникнення перенавчання шляхом додавання штрафу на великі значення у матрицях P та Q.

Спочатку треба розділити дані на тренувальну і тестову вибірки. Далі порахуємо метрики оцінок.

MRE – метрика оцінки точності моделі, яка вимірює середню відносну похибку між передбаченими значеннями та фактичними значеннями. Вона часто використовується для завдань регресії, де важливо врахувати відносну різницю між передбаченим і фактичним значенням.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{|y_i|}, \quad (13)$$

де n – кількість передбачень;

$\hat{y}_i$ – передбачене значення для і-го елемента;

y_i – фактичне значення для і-го елемента;

$|y_i - \hat{y}_i|$ – абсолютна різниця між фактичним і передбаченим значенням;

$|y_i|$ – абсолютне фактичне значення для нормалізації помилки.

MRE вимірює похибку у відсотковому вираженні (відносно фактичних значень), що дозволяє порівнювати результати для даних із різними масштабами. MRE враховує відносну різницю, що дозволяє порівнювати моделі для наборів даних з різними масштабами. Результат у вигляді середньої відносної похибки зрозумілий і дозволяє легко оцінити продуктивність моделі.

Середньоквадратична похибка (RMSE) – обчислюється за формулою (14), як квадратний корінь із середнього значення квадратів між передбаченими і фактичними значеннями. RMSE акцентується на великих похибках, оскільки квадрати різниць значно збільшують вагу великих відхилень. RMSE рахується за формулою [9]:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}, \quad (14)$$

де n – кількість передбачень;

$\widehat{y_i}$ – передбачене значення для і-го елемента;

y_i – фактичне значення для і-го елемента.

RMSE є зручним, якщо важливіші великі похибки і можна штрафувати модель за значні відхилення від фактичних значень.

Середня абсолютна похибка – обчислюється за формулою (15), як середнє значення абсолютних різниць між передбаченими та фактичними значеннями. MAE менш чутлива до великих похибок, оскільки використовує абсолютні значення без зведення у квадрат:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\widehat{y_i} - y_i|, \quad (15)$$

де n – кількість передбачень;

$\widehat{y_i}$ – передбачене значення для і-го елемента;

y_i – фактичне значення для і-го елемента.

Ця метрику можна використовувати коли кожна помилка має однакову «вартість» і великі відхилення не настільки критичні.

Accuracy – метрика, яка вимірює частку правильних передбачень серед усіх правильних передбачень. Точність рахується за формулою (16):

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FN+FN}}, \quad (16)$$

де TP – кількість правильно класифіковані позитивні зразки;

TN – кількість правильно класифікованих негативних зразків;

FP – кількість неправильно класифікованих позитивних зразків;

FN – кількість неправильно класифікованих негативних зразків.

Accuracy може вимірювати, наскільки модель правильно передбачає релевантність елементів (фільмів, товарів, аніме тощо). У контексті бінарної класифікації, метрика показує який відсоток передбачень моделі збігається з реальними уподобаннями користувачів.

Precision (загальна точність) – оцінює частку за формулою (17) правильних передбачень серед усіх позитивних передбачень.

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}, \quad (17)$$

де TP – кількість правильно класифіковані позитивні зразки;

TN – кількість правильно класифікованих негативних зразків;

FP – кількість неправильно класифікованих позитивних зразків;

FN – кількість неправильно класифікованих негативних зразків.

Метрика оцінює скільки з рекомендованих елементів є релевантними для користувача.

Precision@K – метрика оцінює точність рекомендацій перших K передбачених елементів [10]. Ця метрика враховує лише топ-K рекомендацій, що особливо важливо в задачах рекомендаційних систем, де користувач зазвичай взаємодіє з обмеженою кількістю елементів. Метріка точності рекомендацій обчислюється за формулою (18):

$$\text{Precision@K} = \frac{1}{\text{N}}\sum_{\text{i}=1}^{\text{N}} \frac{|\text{R}_\text{i}(\text{K}) \cap \text{G}_\text{i}|}{\text{K}}, \quad (18)$$

де N – загальна кількість користувачів для яких оцінюється точність;

K – кількість топових елементів;

$\text{R}_\text{i}(\text{K})$ – множина топ-K рекомендацій, зроблених для користувача i;

G_i – множина релевантних елементів для користувача i;

$|\text{R}_\text{i}(\text{K}) \cap \text{G}_\text{i}|$ – кількість релевантних елементів серед топ-K рекомендацій.

Precision@K ігнорує релевантні елементи поза топ-K, що важливо для рекомендаційних систем, де користувач взаємодіє лише з кількома найпершими результатами. Якщо множина G_i пуста, Precision@K для цього користувача дорівнює 0 (або не враховується залежно від реалізації).

Recall – метрика яка вимірює частку правильно передбачених позитивних елементів серед усіх релевантних елементів. Загальна повнота розраховується за формулою (19):

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}, \quad (19)$$

де TP – кількість правильно класифіковані позитивні зразки;

FN – кількість неправильно класифікованих негативних зразків.

Метрика оцінює загальну здатність моделі знаходити релевантні елементи. Recall підходить, коли потрібно оцінити повноту моделі для всього набору даних. Наприклад, у задачах класифікації (знайти всі позитивні випадки).

Recall@K – метрика вимірює частку релевантних елементів серед усіх релевантних, які потрапили в топ-K рекомендацій. Повнота топу рекомендацій розраховується за формулою (20):

$$\text{Recall@K} = \frac{1}{N}\sum_{i=1}^{N}\frac{|R_i(K) \cap G_i|}{|G_i|}, \quad (20)$$

де N – загальна кількість користувачів для яких оцінюється точність;

K – кількість топових елементів;

$R_i(K)$ – множина топ-K рекомендацій, згенерованих для користувача i;

G_i – множина всіх релевантних елементів для користувача i;

$|R_i(K) \cap G_i|$ – кількість релевантних елементів серед топ-K рекомендацій;

$|G_i|$ – загальна кількість релевантних елементів.

Recall@K підходить для оцінки рекомендацій, коли користувач взаємодіє лише з обмеженим набором елементів. Наприклад, у стримінгових сервісах, де користувач бачить лише топ-5 або топ-10 рекомендацій.

F1 Score – метрика, яка поєднує Precision (точність) і Recall (повноту) в одне гармонійне середнє значення. Вона є особливо корисною, коли важливо знайти баланс між точністю і повнотою, наприклад, у задачах з незбалансованими класами. F1 score обчислюється за формулою (21):

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \quad (21)$$

де Precision – частка правильних позитивних передбачень серед усіх позитивних передбачень;

Recall – частка правильних позитивних передбачень серед усіх фактичних позитивних прикладів.

F1 Score@K – модифікована версія стандартної метрики F1 Score, яка використовується в задачах ранжування або рекомендаційних системах. Вона оцінює баланс між точністю (Precision@K) і повнотою (Recall@K) у топ-K рекомендаціях. F1 Score@K можливо розрахувати за формулою (22):

$$\text{F1 Score@K} = 2 \times \frac{\text{Precision@K} \times \text{Recall@K}}{\text{Precision@K} + \text{Recall@K}}, \quad (22)$$

F1 Score@K дає збалансовану оцінку точності та повноти в відсортованих рекомендаціях. Показує ефективність рекомендацій серед обмеженого числа елементів, що відповідає реальним сценаріям. Але не враховує порядок елементів (це враховується іншими метриками, як-от NDCG).

NDCG – метрика, яка оцінює якість ранжування рекомендацій, враховуючи релевантність і порядок елементів. Вона широко використовується в системах рекомендацій та інформаційного пошуку.

DCG (Discounted Cumulative Gain) – міра, яка враховує релевантність елементів у ранжованому списку з урахуванням позиції. Чим вище релевантний елемент у списку, тим більше його значимість. А NDCG (Normalized DCG) – нормалізована версія DCG, яка ділить значення DCG на ідеальне DCG (IDCG). Це дозволяє отримати значення в межах [0, 1], де 1 – ідеальне ранжування. DCG обчислюється за формулою (23):

$$\text{DCG} = \sum_{i=1}^{K} \frac{\text{rel}_i}{\log_2(i+1)}, \quad (23)$$

де K – кількість елементів у списку;

rel_i – релевантність елемента на позиції i;

i – позиція елемента у списку.

Чим далі релевантний елемент у списку, тим менше його внесок у загальний результат через логарифмічне зменшення значення. IDCG обчислюється за формулою (24):

$$\text{IDCG} = \sum_{i=1}^{|G|} \frac{\text{rel}_i}{\log_2(i+1)}, \quad (24)$$

де K – кількість елементів у списку;

rel_i – релевантність елемента на позиції i;

$|G|$ – кількість релевантних елементів у списку;

i – позиція елемента у списку.

IDCG обчислюється за формулою, для ідеального порядкування, коли всі релевантні елементи знаходяться на початку списку. Формулюємо формулу NDCG (25):

$$NDCG = \frac{DCG}{IDCG}, \quad (25)$$

де DCG – міра, яка враховує релевантність елементів у ранжованому списку з урахуванням позиції;

IDCG – максимально можливе значення DCG для заданого списку елементів.

2.4 Моделювання нейронної спільної фільтрації

Неявний зворотній зв’язок

Для вирішення ключової проблеми в рекомендаціях – спільної фільтрації на основі неявного зворотного зв'язку. Деякі системи в основному використовують глибинне навчання для моделювання допоміжної інформації, такої як текстові описи предметів. Коли справа доходить до моделювання ключового атрибута в спільній фільтрації – взаємодія між атрибутами користувачів та предметів, все ще використовують матричну факторизацію та внутрішній добуток до прихованих атрибутів користувачів і елементів. Шляхом заміни внутрішнього продукту нейронною архітектурою яка може дізнатися довільну інформацію з даних, представляється загальна структура під назвою NCF, скорочення від Neural network-based Collaborative Filtering. NCF є універсальною і може виражати та узагальнювати матричну факторизацію в своїй структурі. Щоб посилити моделювання NCF нелінійністю, пропонується використовувати багатошаровий перцептрон для вивчення функцій взаємодії користувач-елемент.

Визначимо матрицю взаємодії користувач-елемент $Y \in R^{M \times N}$ на основі неявного зворотного зв’язку користувачів наступним чином:

$$y_{ui} = \{^{1}_{0}, \quad (26)$$

де 1 – вказує на те що є взаємодія між користувачем u та об’єктом, однак це не означає, що «u» дійсно подобається «i»;

0 – «u» не подобається «i» можливо користувач не знає про об’єкт.

Це створює виклик для навчання на основі неявних даних, оскільки вони надають лише «зашумлені» сигнали про можливі переваги користувачів. У той час як спостережувані записи хоча б відображають зацікавленість користувачів у певних об’єктах, незаповнені записи можуть бути просто відсутніми даними, існує природний дефіцит негативного зворотного зв’язку.

Задачі рекомендацій з неявним зворотнім зв’язком формуються як задачі оцінювання балів для незаповнених записів у Y, які використовуються для ранжування об’єктів. Модельні підходи припускають, що дані можуть бути згенеровані підлеглою моделлю. Формально, їх можна абстрагувати як навчання і визначити за формулою (27):

$$\widehat{y_{ui}} = f(u, i|\theta), \quad (27)$$

де $\widehat{y_{ui}}$ – позначає передбачуваний бал для взаємодії y_{ui};

θ – позначає параметри моделі;

f – позначає функцію, яка відображає параметри моделі на передбачуваний бал (функція взаємодії).

Щоб оцінити параметри θ, існуючі підходи зазвичай слідують парадигмі машинного навчання, що оптимізує цільову функцію. У літературі найчастіше використовуються два типи цільових функцій – точкові втрати (pointwise loss) і парні втрати (pairwise loss). Як природне розширення численних робіт щодо явного зворотного зв’язку, методи точкового навчання зазвичай слідують регресійній структурі, мінімізуючи квадратичну похибку між $\widehat{y_{ui}}$ та його цільовим значенням y_{ui}.

Для вирішення відсутності негативних даних деякі методи вважали всі незаповнені записи негативним зворотним зв’язком або вибірково обирали

негативні приклади з незаповнених записів. Для парного навчання ідея полягає в тому, що спостережувані записи повинні мати вищий ранг, ніж незаповнені. Таким чином, замість мінімізації втрат між $\widehat{y_{ui}}$ і y_{ui}, парне навчання максимізує різницю між спостережуваним записом y_{ui}, і незаповненим записом $\widehat{y_{ui}}$.

Структура NCF параметризує функцію взаємодії f, використовуючи нейронні мережі для оцінки $\widehat{y_{ui}}$. Таким чином, вона природно підтримує як точкове, так і парне навчання.

Матрична факторизація NCF

Зображення демонструє, як можна використовувати зв'язки між різними рівнями даних (категорії й продукти) для створення більш точних рекомендацій. Цей підхід дозволяє не лише передбачити, з чим користувач може взаємодіяти, але й зрозуміти його інтереси на більш високому рівні (категорії). Це основа багатьох сучасних систем персоналізації.

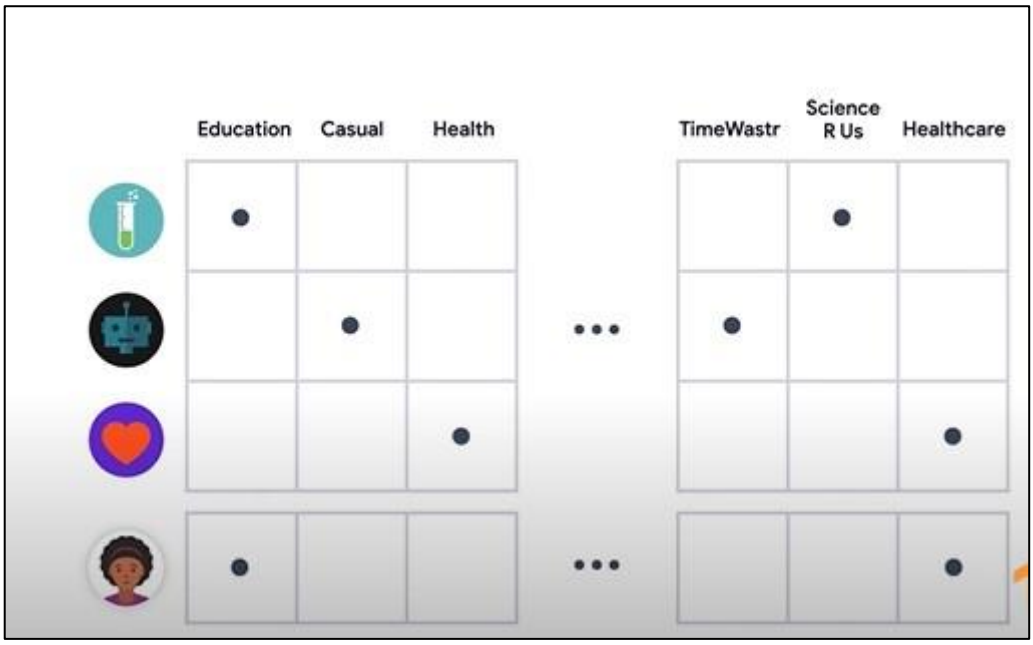

Рисунок 2.6 – Приклад розріджених матриць у контексті рекомендаційних систем

Матрична факторизація асоціює кожного користувача та об'єкт із вектором прихованих ознак, так можливо описати за формулою:

$$y_{ui} = p_u^T q_i = \sum_{k=1}^{K} p_{uk}\, q_{ik}, \quad (28)$$

де K – розмірність простору прихованих ознак.

Таким чином, MF моделює двосторонню взаємодію прихованих факторів користувача й об'єкта, припускаючи, що кожний вимір простору є незалежним і лінійно поєднується з однаковою вагою. Отже, MF можна вважати лінійною моделлю прихованих факторів.

На рисунку 2.7 проілюстровано, як функція скалярного добутку може обмежувати виразність MF. Є два основні налаштування для кращого розуміння прикладу. По-перше, оскільки MF відображає користувачів і об'єкти в один і той самий простір прихованих ознак, подібність між двома користувачами також можна вимірювати скалярним добутком або, що еквівалентно, косинусом кута між їхніми векторами.

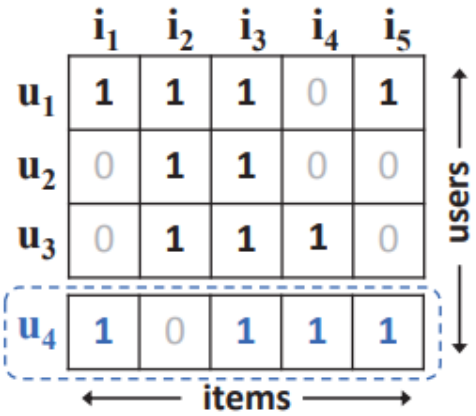

Рисунок 2.7 – Метод матричної факторизації

Для з'ясування справжньої подібності двох користувачів MF використовує коефіцієнт Жаккара. Спершу розглянемо перші три рядки (користувачів) на рисунку 2.7, де легко побачити, що s_{23} (0.66) > s_{12} (0.5) > s_{13} (0.4).

Отже, просторові співвідношення p_1, p_2, та p_3 в прихованому просторі можна зобразити, як показано на рисунку 2.8. Тепер розглянемо нового користувача u_4, дані для якого наведені пунктирною лінією на рисунку 2.7. Де значення s_{41} (0.6) > s_{43} (0.4) > s_{42} (0.2) тобто u_4, найбільше схожий на u_1, потім на u_3, і найменше – на u_2. Проте, якщо модель MF розташовує p_4, ближче до p_1 (два варіанти показані пунктирними лініями на рисунку 2.8) це призводить до того, що p_4 буде ближче до p_2, ніж до p_3, що спричинить значні помилки у ранжуванні.

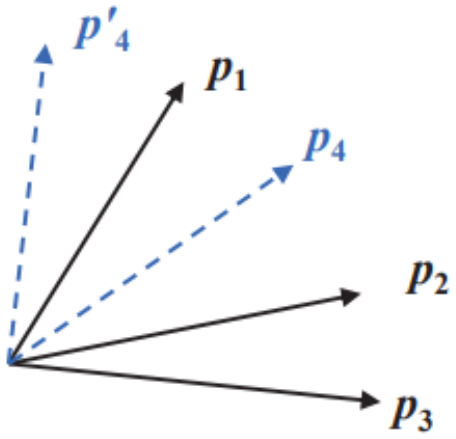

Рисунок 2.8 – Векторне представлення користувача

Цей приклад демонструє обмеження MF, спричинені використанням простого та фіксованого скалярного добутку для оцінювання складних взаємодій між користувачем і об'єктом у низьковимірному просторі. Одним зі способів вирішення цієї проблеми є збільшення кількості прихованих факторів K. Однак це може негативно вплинути на узагальнення моделі (наприклад, спричинити перенавчання), особливо за умови розріджених даних.

Загальна структура NCF описує процес навчання NCF за допомогою ймовірнісної моделі, що підкреслює бінарний характер імпліцитних даних. Метод матричної факторизації може бути виражений та узагальнений в рамках NCF. Для дослідження глибинних нейронних мереж у контексті колаборативної фільтрації запропоновано конкретний приклад NCF, який використовує багатошаровий перцепротон (MLP) для навчання функції взаємодії між користувачем та елементом. Нарешті представлено нову модель нейронної матричної факторизації, яка об’єднує MF та MLP в рамках NCF, поєднуючи лінійність MF та нелінійність MLP для моделювання прихованих структур взаємодії користувача з елементом.

Приймається багатошарове представлення для відображення взаємодії і елементу y_{ui} як показано на рисунку 2.9, де вихід одного шару слугує входом для наступного. Вхідний шар складається з двох векторів ознак v_u^U і v_i^I, u описують користувача, і описують елементи, таких як контекстно-орієнтовані, на основі контенту та на основі сусідніх елементів. Використовується лише ідентифікатор користувача та елемента як вхідні ознаки, яка перетворюється у бінаризований

розріджений вектор з однорозрядним кодуванням (ембеддинг). Такий універсальний підхід представлення ознак дозволяє легко налаштувати метод для вирішення проблеми «холодного старту» за допомогою контетних ознак для представлення користувачів та елементів.

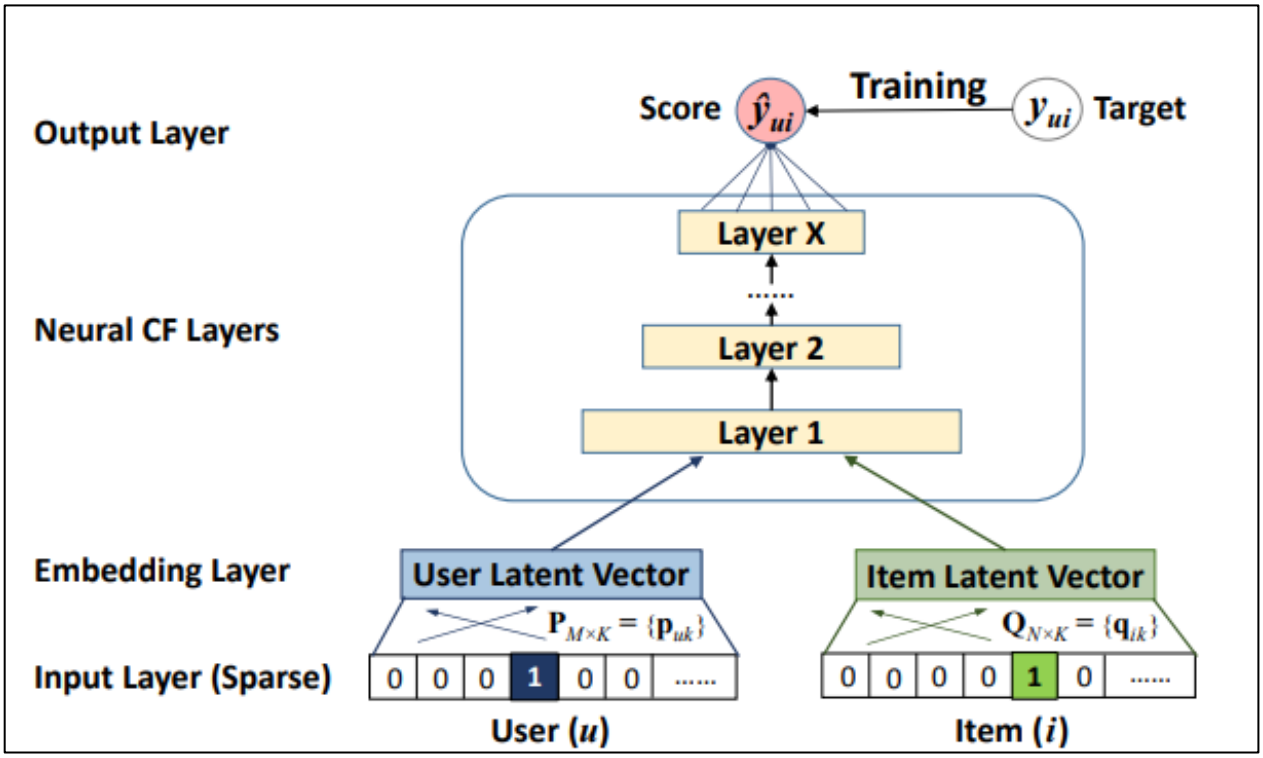

Рисунок 2.9 – Модельна стратегія нейронної спільної фільтрації

Над вхідним шаром розташовується шар вбудовування – повно зв’язний шар, який проєктує розріджене представлення у щільний вектор. Отримане вбудовування користувача можна розглядати як прихований вектор для користувача у контексті моделі прихованих факторів. Вбудовування користувача та елемента подаються у багатошарову нейронну архітектуру, яка називається шарами нейронної колаборативної фільтрації, для проєктування прихованих векторів у прогнозовані оцінки. Кожен шар нейронної колаборативної фільтрації можна налаштувати для виявлення певних прихованих структур взаємодії користувачів та елементів. Розмір останнього прихованого шару X визначає здатність моделі. Останній вихідний шар дає прогнозовану оцінку $\widehat{y_{ui}}$ а навчання виконується шляхом мінімізації точкових втрат між $\widehat{y_{ui}}$ і цільовим значенням y_{ui}.

Іншим підходом до навчання моделі є попарне навчання, таке як використання ранжування Bayesian Personalized Ranking і втрат на основі маржі.

Прогнозну модель NCF можна описати за формулою (29) як:

$$\widehat{y_{ui}} = f(P^T v_u^U, Q^T v_i^I | P, Q, \theta_f), \quad (29)$$

де $P \in R^{M \times K}$ і $Q \in R^{N \times K}$ – матриця прихованих факторів для користувачів та елементів відповідно;

θ_f – позначає параметри моделі функції взаємодії.

Всього є X шарів нейронної колаборативної фільтрації. Оскільки функція f визначається за формулою (30), як багатошарова нейронна мережа її можна записати як:

$$f\left(P^T v_u^U Q^T v_i^I\right) = \phi_{out}\left(\phi_x(\dots \phi_2\left(\phi_1(P^T v_u^U, Q^T v_i^I)\right)\dots)\right), \quad (30)$$

де ϕ_{out} та ϕ_x – познають функцію відображення для вихідного шару та x-го шару нейронної колаборативної фільтрації.

Далі навчимо модель NFC використовуючи існуючи точкові методи які в основному виконують регресію з квадратичною функцією втрат:

$$L_{sqr} = \sum_{(u,i) \in Y \cup Y^-} w_{ui}(y_{ui} - \widehat{y_{ui}})^2, \quad (31)$$

де Y – позначає набір спостережуваних взаємодій;

Y^- – позначає набір негативних прикладів, які можуть бути обрані з усіх з певної частини неспостережуваних взаємодій;

w_{ui} – гіперпараметр, що позначає вагу навчального випадку.

Хоча квадратичну функцію втрат можна пояснити припущенням, що спостереження генеруються з гауссового розподілу, це може не відповідати особливостям імпліцитних даних, де цільове значення y_{ui} є бінаризованим (1 або 0), позначаючи взаємодію або її відсутність. Розглядаючи одно-класну природу імпліцитного відгуку, значення y_{ui} можна розглядати як мітку: 1 означає, що елемент «i» є релевантним для користувача «u», і 0 – навпаки. Прогнозна оцінка $\widehat{y_{ui}}$

тоді представляє ймовірність релевантності «і» для «u». Для додання NCF ймовірнісного пояснення, вихід $\widehat{y_{ui}}$ обмежується в діапазоні від 0 до 1, за допомогою ймовірнісної функції, наприклад логістичної або пробіт-функції, як функції активації для вихідного шару ϕ_{out}. За цих умов функція правдоподібності визначається як:

$$p(Y, Y^{-}|P, Q, \theta_f) = \prod_{(u,i)\in Y} \widehat{y_{ui}} \prod_{(u,j)\in Y^{-}} (1 - \widehat{y_{ui}}), \quad (32)$$

де $(u, j) \in Y^{-}$ – набір пар, де користувач u не взаємодіє з елементом j;

Y^{-} – набір неспостережуваних або негативних прикладів взаємодій;

$\widehat{y_{ui}}$ – прогнозована ймовірність того, що користувач «u» взаємодіє з елементом «і»;

$1 - \widehat{y_{ui}}$ – ймовірність того що користувач u не взаємодіє з елементом і.

Взявши негативний логарифм правдоподібності, отримуємо формулу:

$$L = \sum_{(u,i)\in Y} \log \widehat{y_{ui}} - \sum_{(u,j)\in Y^{-}} \log(1 - \widehat{y_{ui}}) =$$

$$= -\sum_{(u,i)\in Y\cup Y^{-}} (y_{ui} \log \widehat{y_{ui}} + (1 - y_{ui}) \log(1 - \widehat{y_{ui}})), \quad (33)$$

де $\widehat{y_{ui}}$ – прогнозована модельна ймовірність взаємодії між користувачем u і елементом «і»;

$\log \widehat{y_{ui}}$ – логарифм цієї ймовірності.

Це функція втрат, яку мінімізують у методах NCF, і її оптимізацію можна виконати за допомогою стохастичного градієнтного спуску (SGD), або інших оптимізаторів таких як Adagard, Adam. Використання ймовірнісного підходу для NCF дозволяє розглядати рекомендації з імпліцитним зворотним зв’язком як задачу бінарної класифікації [11].

Висновок до розділу 2

Під час розробки розділу 2 розглянуто проектні рішення, що забезпечують основи для створення ефективної рекомендаційної системи. Детально проаналізовано бізнес-процеси рекомендацій, що дозволило побудувати IDF0 та

DFD діаграми для візуалізації основних етапів функціонування системи. Це надало можливість краще зрозуміти логіку взаємодії між компонентами системи та потоками даних.

Змодельовані алгоритми рекомендацій, такі як матрична факторизація та нейронна спільна фільтрація. Проілюстровано, що матрична факторизація є одним із найефективніших методів для роботи з розрідженими даними, оскільки вона дозволяє виявляти приховані закономірності в уподобаннях користувачів. Нейронна спільна фільтрація, своєю чергою, забезпечує гнучкість завдяки використанню нелінійних моделей, що дозволяє враховувати складні взаємодії між користувачами та елементами.

Проведений аналіз підтвердив доцільність використання стохастичного градієнтного спуску (SGD) та адаптивних алгоритмів оптимізації, таких як Adam, для навчання моделей. Це дозволяє швидко знаходити оптимальні параметри навіть при великому обсязі даних.

Також у розділі розглянуті методи що забезпечують можливість роботи як із явними, так і з неявними відгуками користувачів, що є ключовим аспектом для створення універсальної рекомендаційної системи.

3 ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Архітектура системи

Архітектура мікросервісів – архітектурний стиль, який структурує програму як набір невеликих автономних служб, кожна з яких відповідає за певні бізнес-можливості. Ці служби обмінюються даними через чітко визначені API і можуть розгортатися незалежно, дозволяючи командам ефективніше розробляти, розгортати та масштабувати програми. Кожен мікросервіс може бути написаний на різних мовах програмування або використовувати різні технології, залежно від його потреб. Мікросервіси повинні мати двома факторами:

– сильною зв'язаністю: кожен мікросервіс має бути спрямований на вирішення однієї чітко визначеної задачі, виконуючи її ефективно;

– слабкою залежністю: мікросервіси мають бути максимально незалежними один від одного, що дозволяє їм працювати незалежно один від одного, це полегшує масштабування, тестування та оновлення кожного сервісу без впливу на інші.

На рисунку 3.1 продемонстрована мікросервісна архітектура [12].

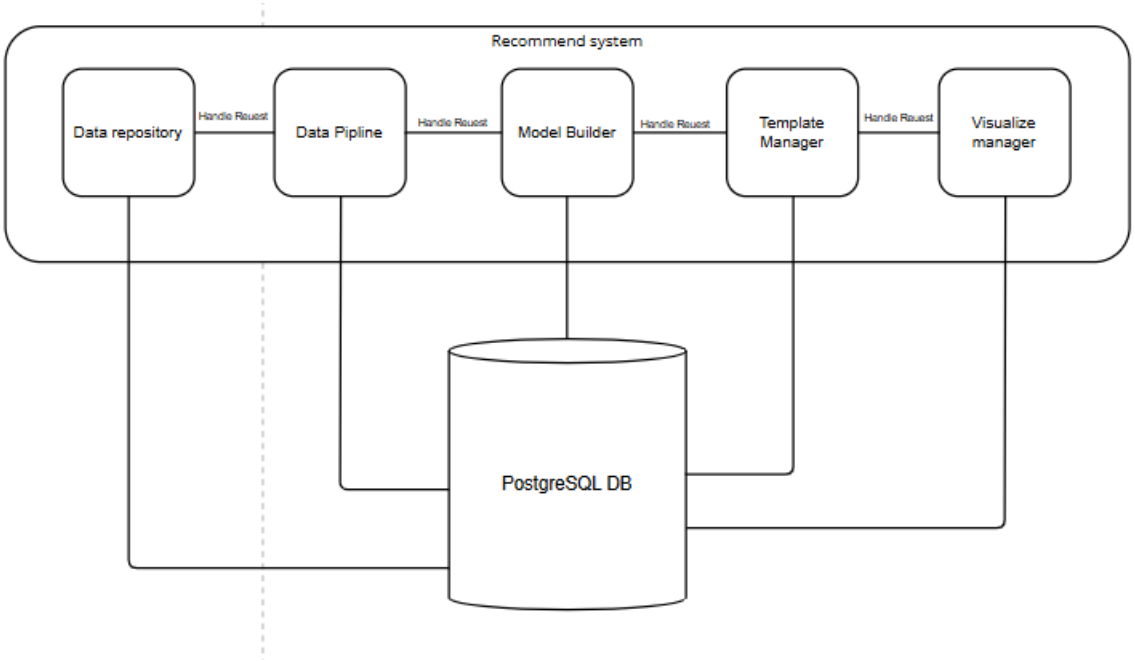

Рисунок 3.1 – Мікросервісна архітектура

Для системи рекомендацій, що використовує мікросервіси, найкращим вибором є мікросервісна архітектура. Цей шаблон дозволяє мікросервісам взаємодіяти незалежно один від одного, виконуючи свої задачі, що особливо корисно для обробки великих обсягів даних. Він добре підходить для алгоритмів рекомендацій, таких як TensorFlow, і дозволяє легко масштабувати систему. Використовуючи підхід, мікросервіс, кожна служба може обробляти запити за своїми правилами.

Архітектура рекомендацій у системі фактично обґрунтовується на кількох ключових компонентах:

– збір даних: дані про користувачів, їх уподобання, поведінку та взаємодію з продуктами зберігаються в базах даних;

– фільтрація та обробка даних: алгоритми фільтрації (Collaborative filtering, Content-based filtering або гібридні методи), які аналізують поведінку користувачів;

– модель машинного навчання: TensorFlow використовується для створення та навчання моделей, які прогнозують рекомендації;

– API: запити проходять через GraphQL та REST для інтеграції з інтерфейсом.

3.2 Структура системи

Опис дійових осіб

User – користувач який отримує рекомендації в застосунку.

Data Scientist – аналізує дані користувачів і їх взаємодію з системою для покращення алгоритмів рекомендацій.

Developer – створює та вибирає шаблони для рекомендаційної системи та працює над оптимізацією моделі, щоб підвищити її точність і швидкодію.

Server – надає необхідні дані з бази даних, забезпечуючи їх доступність для моделі та рекомендаційної системи.

Репозиторій даних

Перша частина системи яка має назву «Data Repository» розроблена за паттерном «Repository Pattern» [13]. Використання шаблону репозиторію дозволяє

абстрагувати роботу з джерелами даних (серверами або API). Це частина яка відповідає за отримання і збереження даних у різних форматах для використання в системі. Рисунок 3.2 зображує діаграму класів мікросервісу «Репозиторій даних».

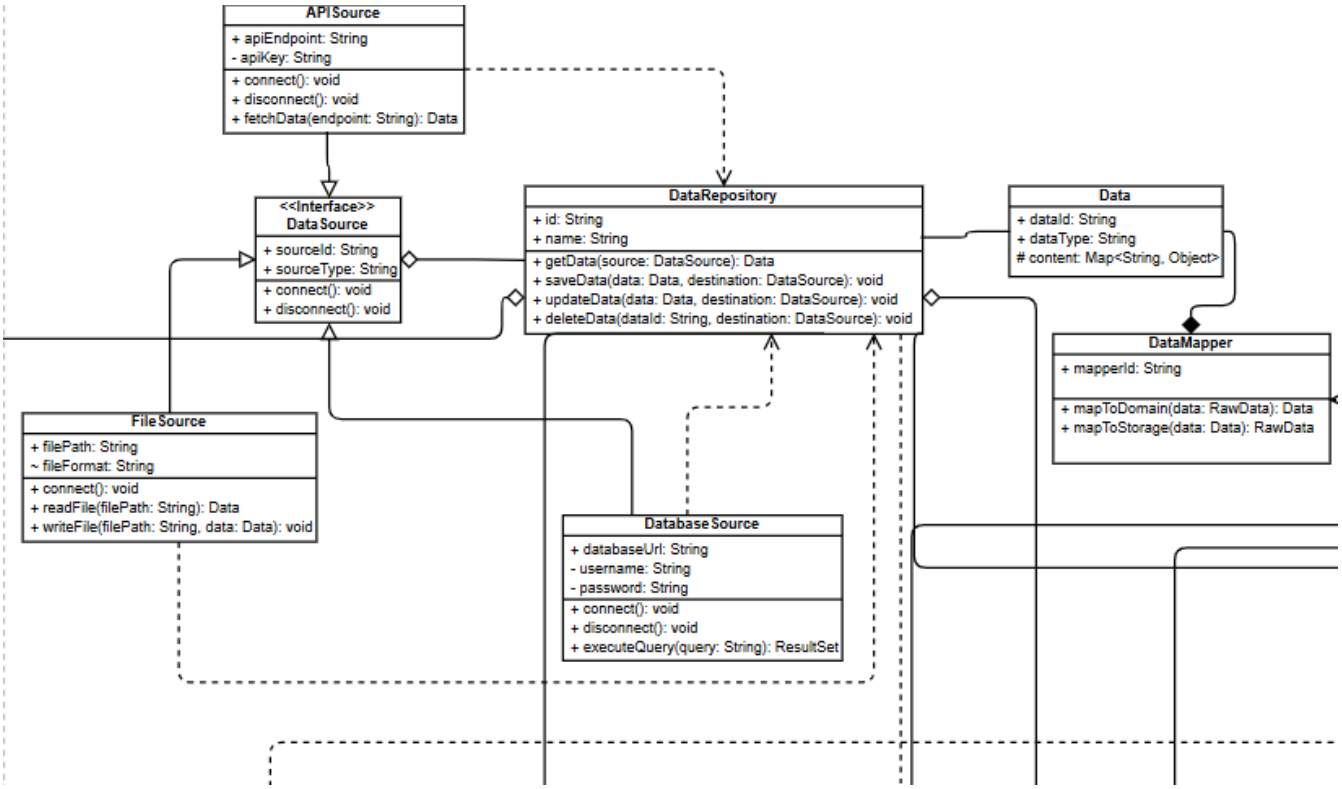

Рисунок 3.2 – Діаграма класів мікросервісу «Репозиторій даних»

Управління шаблонами

На рисунку 3.3 зображений мікросервіс що розробляє шаблони налаштувань моделей, абстрагуючись від процес їхнього налаштування і збереження за допомогою шаблону програмування «Factory Pattern». Цей шаблон підтримує зберігання шаблонів параметрів моделей та функцій попередньої обробки, як файлів у сховищі або у базі даних, забезпечуючи гнучкість у виборі збереження.

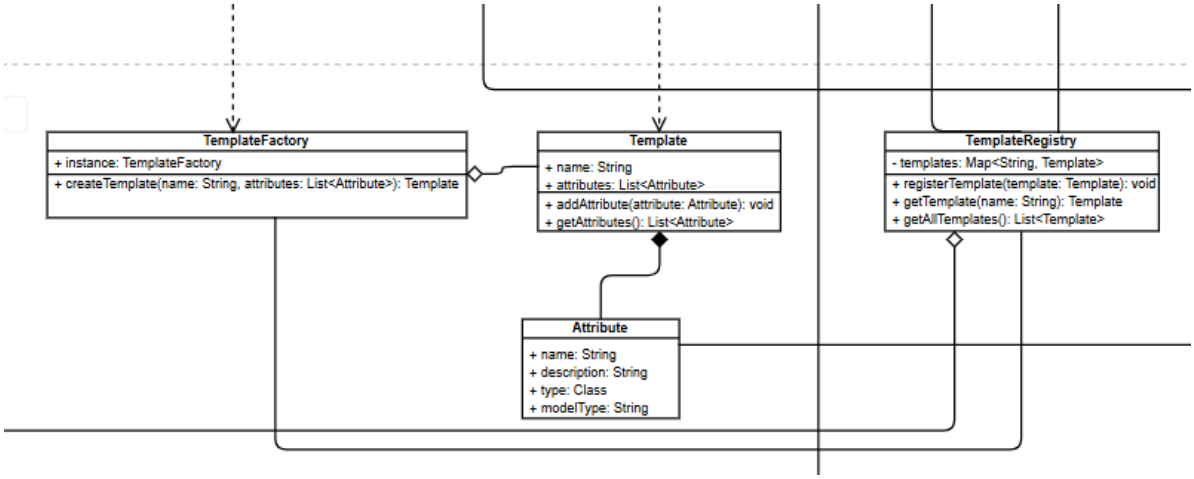

Рисунок 3.3 – Діаграма класів мікросервісу «Управління шаблонами»

Побудова моделей

Шаблон стратегії корисний для вибору алгоритму моделей, який найкраще підходить для поточних даних. Це дозволяє по різному змінювати моделі залежно від задачі без зміни основного коду. На рисунку 3.4 продемонстровано діаграму класів, що зображує мікросервіс який розробляє готові моделі, використовуючи шаблон програмування «Strategy».

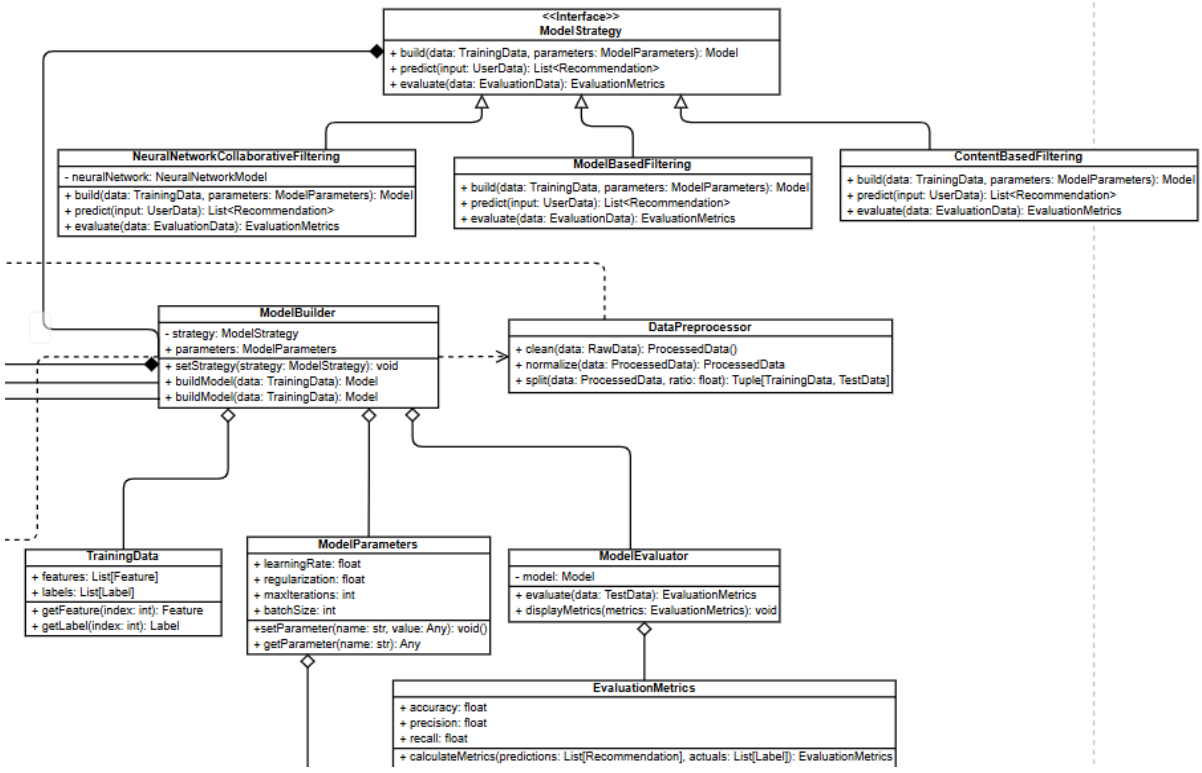

Рисунок 3.4 – Діаграма класів мікросервісу «Модельної стратегії»

Конвеєр даних

Далі на рисунку 3.5 показана діаграма класів четвертої частина, яка створена за допомогою шаблону «Pipeline Pattern» та відповідає за обробку даних, що може включати кроки кластеризації, нормалізації, очищення, створення матриць та інших процесів. Це допоможе зробити етапи трансформації даних простішими для підтримки та налагодження.

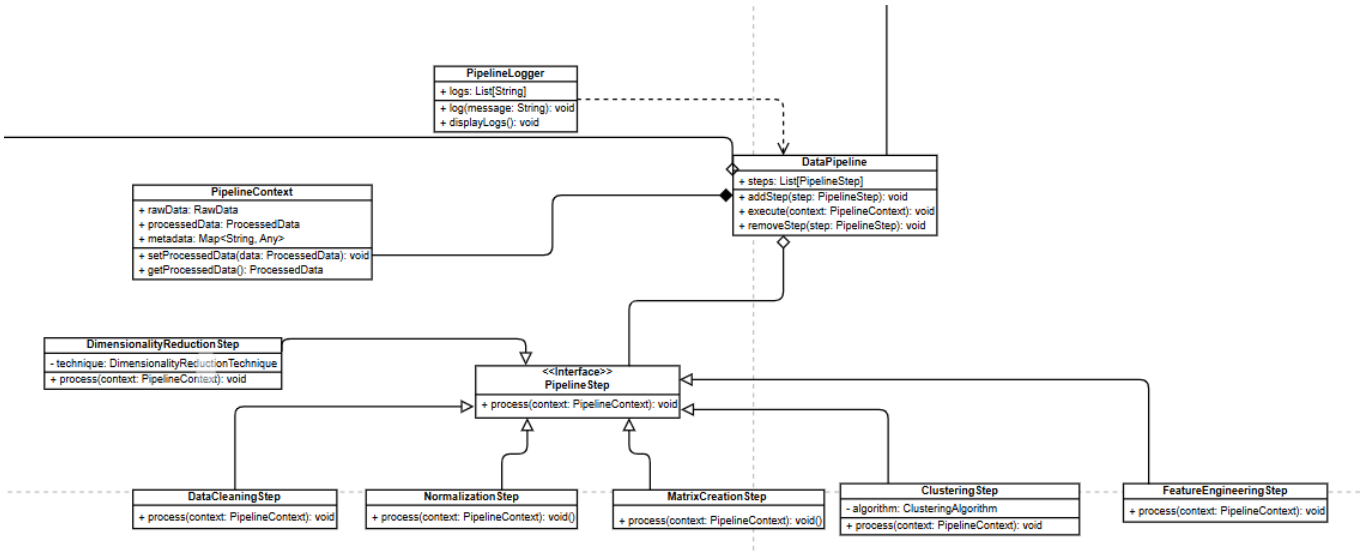

Рисунок 3.5 – Діаграма класів мікросервісу «Модельної стратегії»

Екранізація даних

Мікросервіс екранізації даних візуально продемонстровано за допомогою діаграми класів на рисунку 3.6.

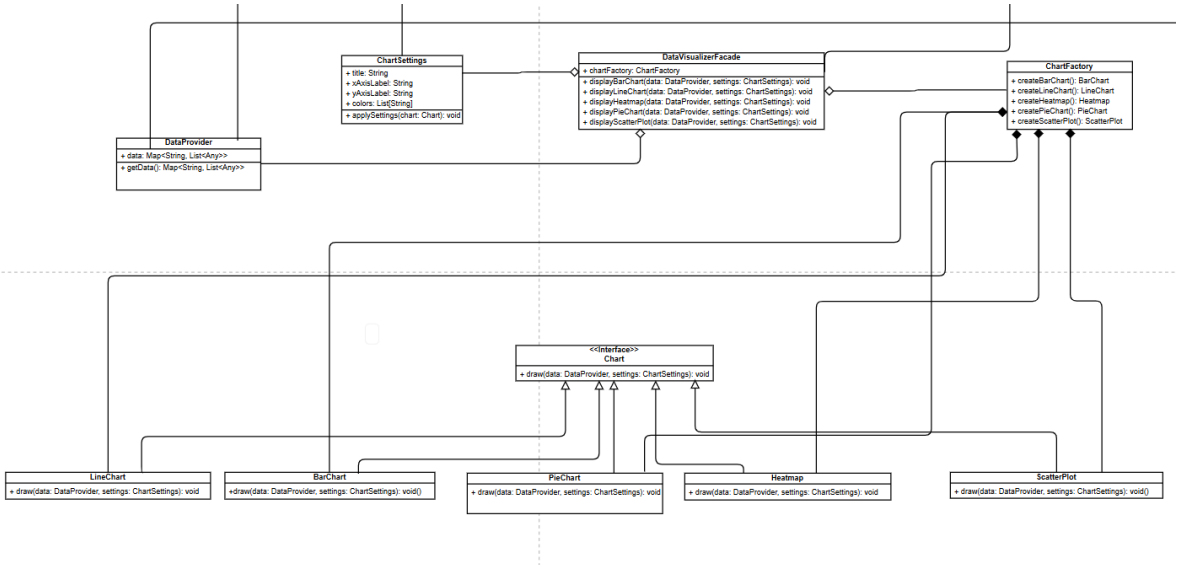

Рисунок 3.6 – Діаграма класів мікросервісу «Екранізація даних»

За допомогою «Façade Pattern» поліпшується доступ до різних візуалізацій, надаючи єдиний програмний інтерфейс для виклику різних типів графіків. Це робить візуалізацію більш узгодженою і легкою для налаштування.

3.3 Опис варіантів використання

Сценарії роботи системи:

– дата аналітику потрібно завантажити певний набір даних (короткий сценарій): для цього дата аналітик входить у систему, обирає набір даних для аналізу;

– сервер надає доступ до певного набору даних для побудови моделі для цього формує запит та віддає команду системі формувати модель (поверхневий сценарій);

– головний сценарій (успішний): користувачу завантажується на певний сервіс, де йому потрібно отримати рекомендаційний набір даних, для цього система отримає запит від серверу та починає конструювати модель або вибирати існуючу, за шаблоном який надається розробником, після побудови, оптимізації моделі, модель реалізується і оброблений набір даних відправляється на сервер, звідки користувач отримує рекомендації;

– альтернативні сценарії:

1. неправильний формат або набору даних від сервера;
2. відсутність шаблону моделі;
3. недостатньо даних для побудови моделі;
4. помилка оптимізації моделі;
5. помилка з відправленням рекомендацій серверу;
6. час обробки перевищено;
7. користувач оновить рекомендації;

– дата аналітик намагається отримати доступ аналітичних даних і рекомендацій у системі аналітики даних:

Scope: система аналітики даних

Level: мета користувача (user-goal)

Primary Actor: data science

Stakeholders and Interests:

1. **user:** отримати рекомендації в застосунку;

2. **data scientist:** аналізувати дані користувачів і їх взаємодію з системою та покращувати алгоритми рекомендацій;

3. **developer:** створювати та вибирати шаблони даних для рекомендаційної системи, а також інтегрувати у інші системи;

4. **server:** надавати та обробляти необхідні дані з бази даних, забезпечувати їх доступність для моделі та рекомендаційної системи.

Preconditions: необхідні дані доступні у системі

Main success scenario:

1. дата аналітик заходить до панелі управління даними;
2. дата аналітику потрібно отримати інформацію щодо даних, він обирає опцію для отримання аналітики з використанням бар-чарту, таблиці даних та візуалізацію графіку залежностей метрик моделі;
3. система отримує запит, обробляє дані та виводить запитувані візуалізації;
4. дата аналітик обирає кількість записів, які потрібно показати в таблиці, та запускає оновлення;
5. система оновлює таблицю з вибраною кількістю записів;
6. дата аналітик переходить до розділу рекомендацій, де система пропонує обрану модель для рекомендацій, наприклад модель «Collaborative Filtering», побудована на основі схожих користувачів;
7. дата аналітик отримує рекомендації на основі трендів;
8. система дає можливість експорту даних і візуалізацій у звіти.

Extensions

а) Дата аналітик обирає тип візуалізації:

1) система перевіряє наявність необхідних даних для візуалізації;

2) якщо даних недостатньо або вони відсутні: система виводить повідомлення про це, пропонує перевірити доступні фільтри або діапазон даних,

оновити запит до бази даних з іншими параметрами та звернутися до підтримки для вирішення проблеми;

3) дата аналітик коригує запит;

b) дата аналітик повторно надсилає запит, після чого система знову перевіряє доступність даних:

1) запит надсилається успішно;

2) система отримує доступ до даних;

c) дата аналітик обирає опцію для отримання рекомендацій:

1) система завантажує останні доступні дані для моделі рекомендацій;

2) якщо система виявляє застарілі або пошкоджені дані, помилка записується у log і система повідомляє аналітика, далі оновлює дані для рекомендаційної моделі;

3) система проводить оновлення даних.

d) дата аналітик запускає завантаження та обробку даних:

1) система виявляє проблему із завантаженням, тоді вона виконує автоматичну перевірку;

2) система повідомляє аналітику і пропонує повторне завантаження або звернення до підтримки;

3) помилку виправлено системою, система продовжує працювати в штатному режимі.

Кінець сценарію розширення

3.4 Побудова діаграм UML

Діаграма варіантів використання

На рисунку 3.7 показана діаграма використання що демонструє як саме використовується система декількома дійовими особами [14].

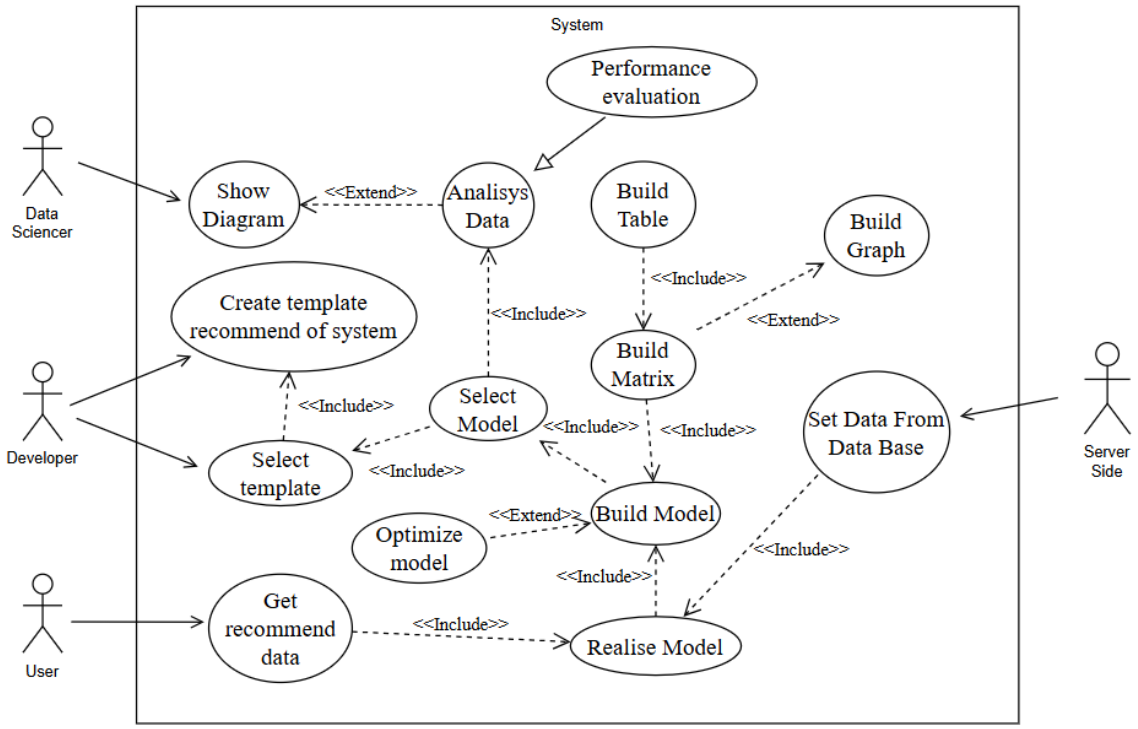

Рисунок 3.7 – Повна діаграма варіантів використання системи рекомендацій

Діаграма станів та переходів

На рисунку 3.8 станів та переходів для рекомендаційної системи, яка показує основні стани такі як завантаження даних, обробка, тренування моделі, генерація та відображення рекомендацій, а також обробка помилок. Станам відповідають переходи між діями, що виконуються на кожному етапі обробки і передачі даних.

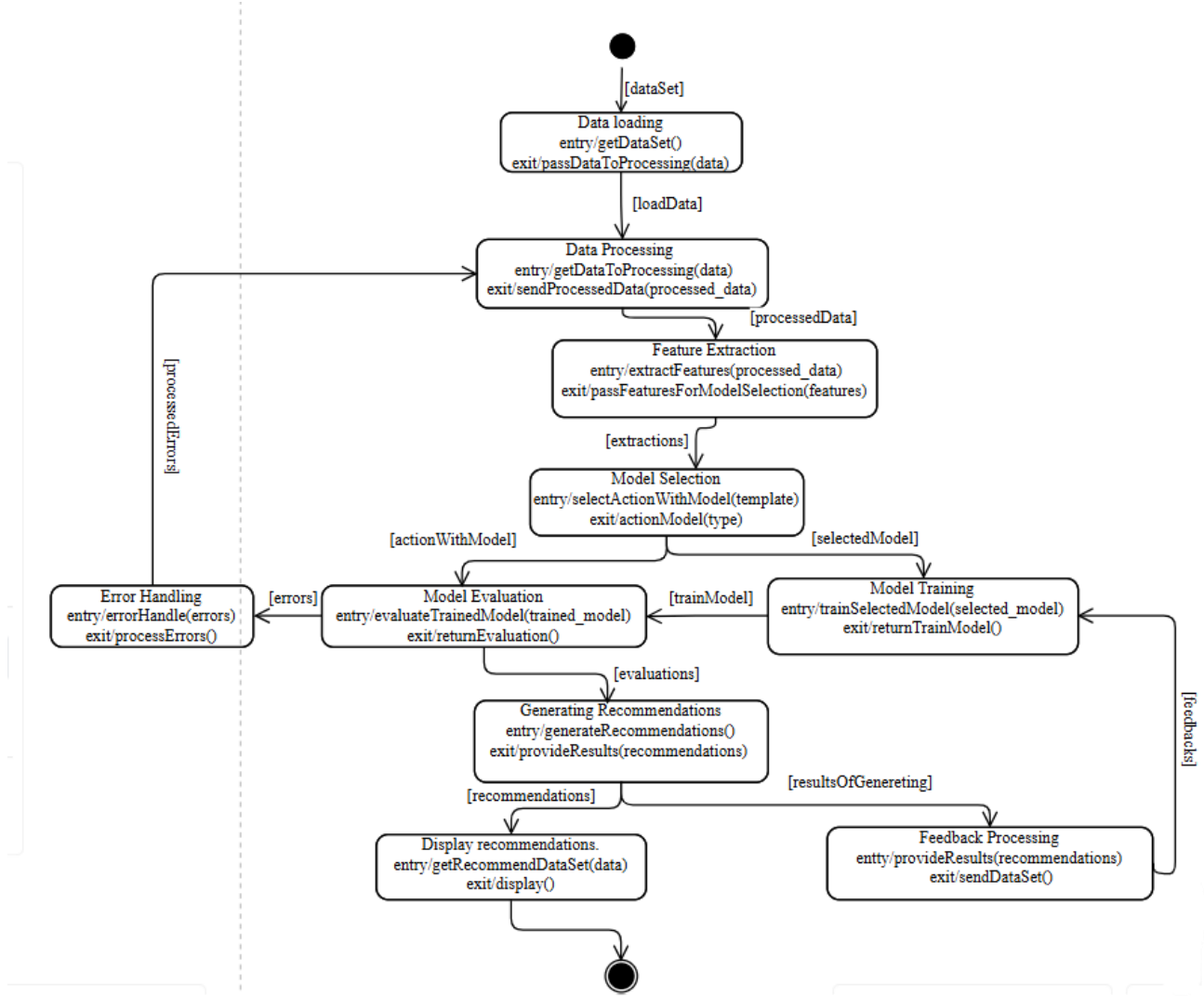

Рисунок 3.8 – Діаграма станів та переходів

Побудова діаграм діяльності

Діаграма показує процес підготовки даних до генерації рекомендацій. Сама генерація почнеться з попередньої обробки даних і виділення ознак, що потім розподіляються на тренувальну та тестову вибірки. Далі створюється модель або оновлюється, проходить перевірку, і на основі результатів створення моделі користувачу надаються персоналізовані рекомендації.

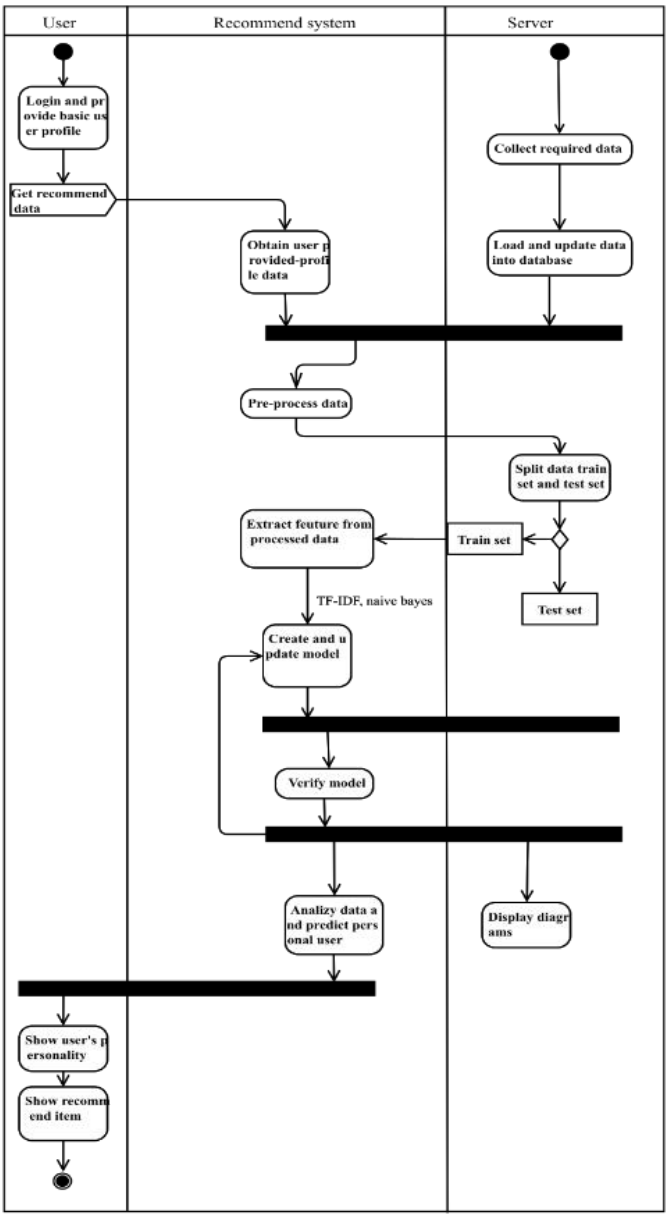

Рисунок 3.9 – Діаграма діяльності «Обробка даних і генерації рекомендацій»

На рисунку 3.10 продемонстровано процес роботи із шаблонами моделі, які можуть бути створені чи оновлені для подальшого використання в системі рекомендацій. Спочатку вибирається або встановлюється шаблон. Якщо потрібно зміни, додаються нові моделі до шаблону або створюється нова модель.

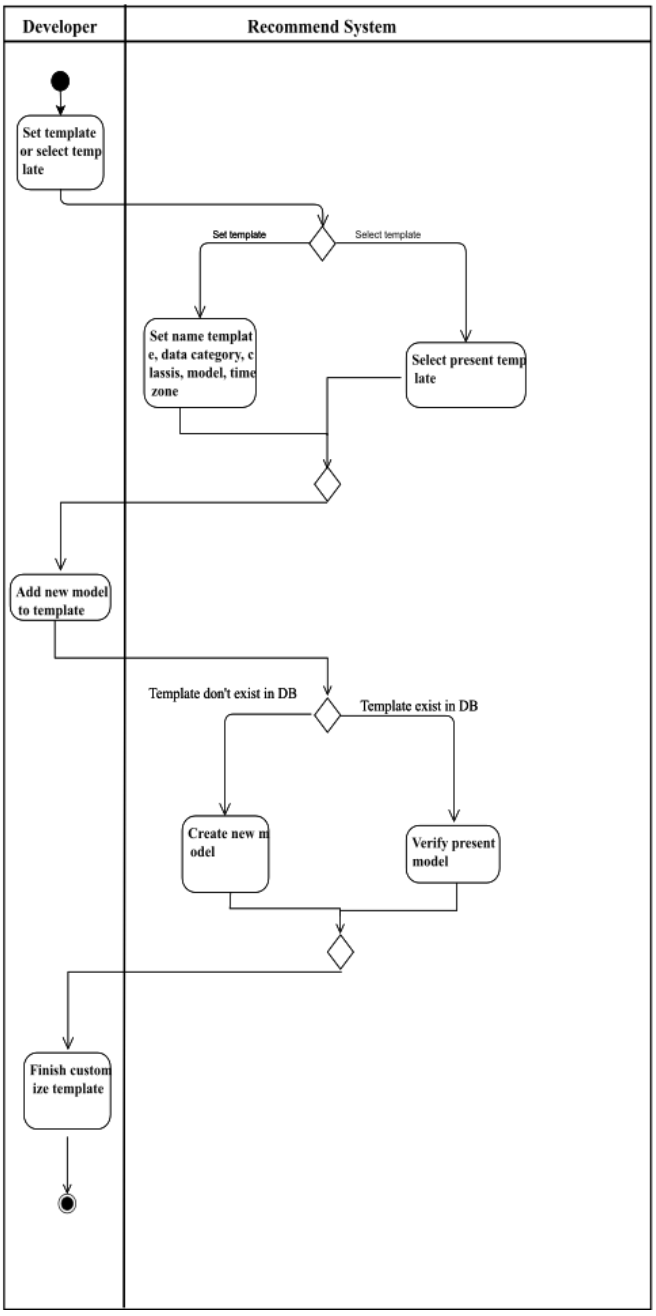

Рисунок 3.10 – Діаграма діяльності «Управління шаблонами моделі»

На рисунку 3.11 зображено процес генерації та оновлення рекомендацій для користувача. Ця діаграма відображає основний процес взаємодії користувача із системою рекомендацій. Спочатку користувач шукає конкретний об’єкт, сервер отримує дані з бази даних і знаходить елементи на основі опису або схожості. Система рекомендацій створює або використовує наявну модель для надання результатів користувачу. Далі користувач може залишати рейтинг та отримувати оновлений рекомендацій на основі своїх вподобань.

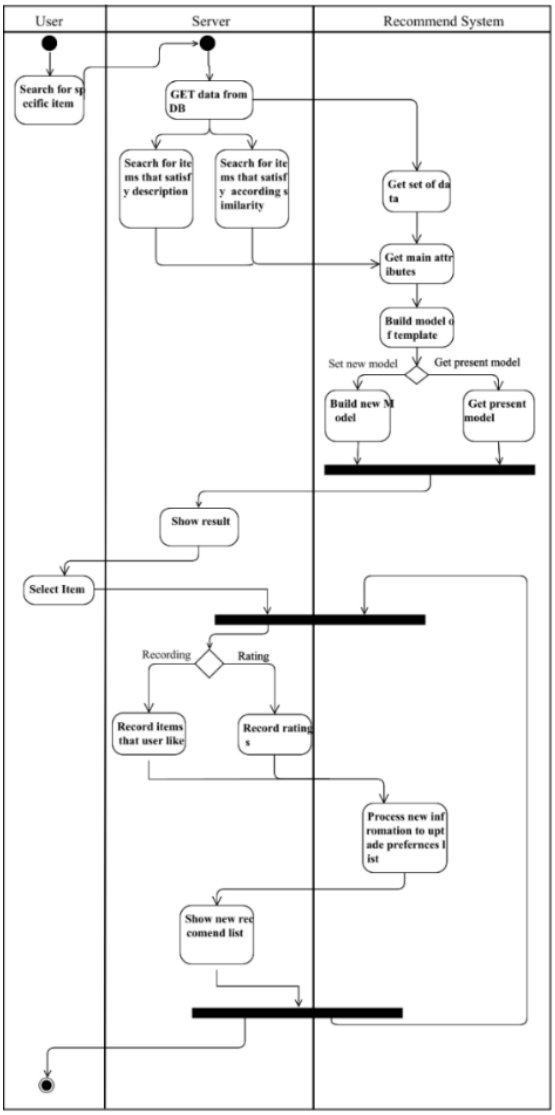

Рисунок 3.11 – Діаграма діяльності «Генерація та оновлення рекомендацій для користувача»

Побудова діаграми послідовності

На рисунку 3.12 зображено діаграму послідовності описує процес запиту рекомендацій користувачем у системи Filter Collaboration. Спочатку User надсилає запит до система фільтрації яка відповідає за процеси управління даними і дає вказівки куди і коли запускати той чи інший процес для правильної послідовності. Затим система фільтрації звертається до репозиторію даних який управляє процесом зберігання даних та отримання даних в правильному форматі. Система фільтрації передає дані до рекомендаційного двигуна для генерації рекомендації. Генератор рекомендацій запускає цикл обробки рекомендацій для уточнення результатів. Далі опціонально виконується фільтрація за релевантністю. В кінці

результати надсилаються системи яка відповідає за візуалізацію, а та в свою чергу надсилає на сервер для подальшої обробки.

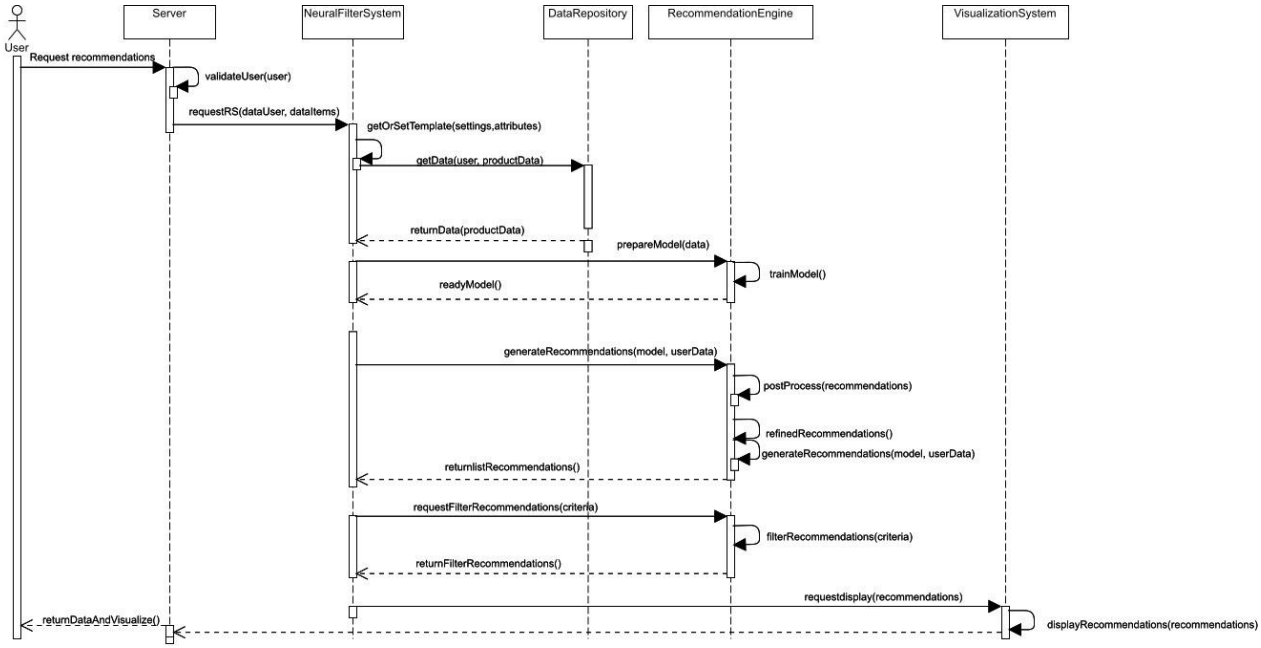

Рисунок 3.12 – Діаграма послідовності «Генерація рекомендацій для користувача»

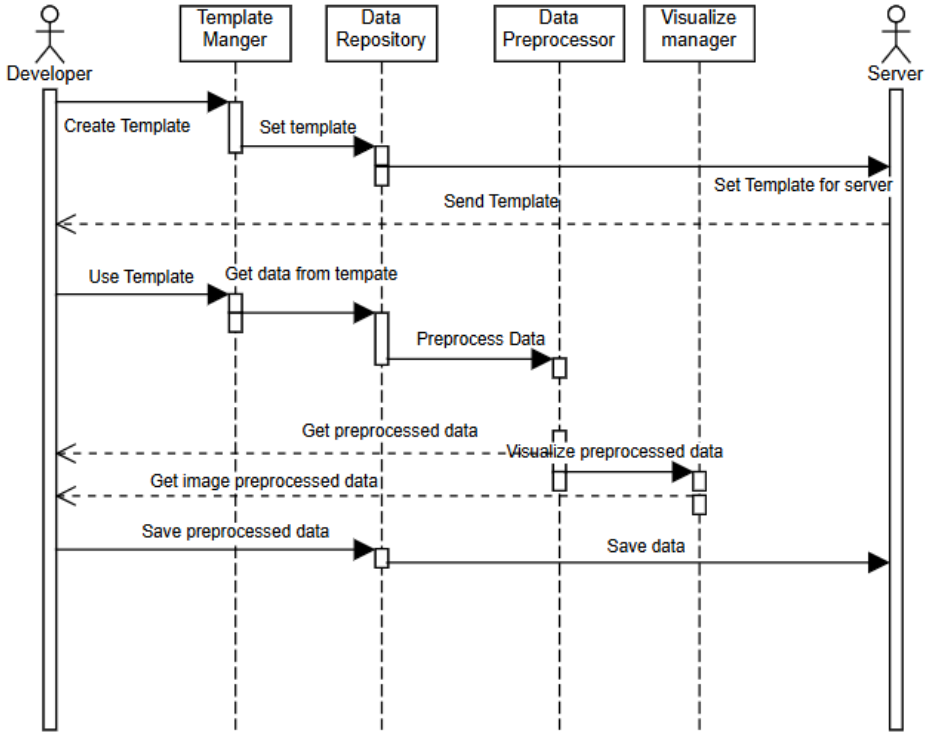

Рисунок 3.13 – Діаграма послідовності «Візуалізації етапів передобробки даних»

Діаграма на рисунку 3.13 відображає послідовність дій у системі, яка включає взаємодію між різними компонентами та ролями (розробник і сервер). Розробник ініціює дії, пов'язані зі створенням, використанням і збереженням шаблонів, а також із обробкою даних. Менеджер шаблонів відповідає за створення та управління шаблонами. Взаємодіє з іншими компонентами для передачі або отримання інформації про шаблони. Обробник даних виконує попередню обробку даних на основі отриманих шаблонів. Менеджер візуалізації відповідає за візуалізацію попередньо оброблених даних. Сервер отримує дані шаблонів для подальшого використання або обробки

Побудова діаграми пакетів

На наступному рисунку 3.14 відображається діаграма пакетів UML, яка спрощує діаграму класів системи рекомендацій її архітектуру та залежності. Діаграма включає пакети основних бібліотек такі Flask, Pandas, NumPy, Matplotlib, TensorFlow. Зв’язки імпорту показують, які пакети залежать один від одного для виконання своїх завдань. Зв’язки об’єднання вказують на внутрішні взаємодії компонентами в межах пакетів, які спільно обробляють дані. Загальна структура діаграми забезпечує модульність, яка полегшує інтеграцію, та розширення системи. Ця архітектура показує що система гнучка та легко налаштовується для різних завдань.

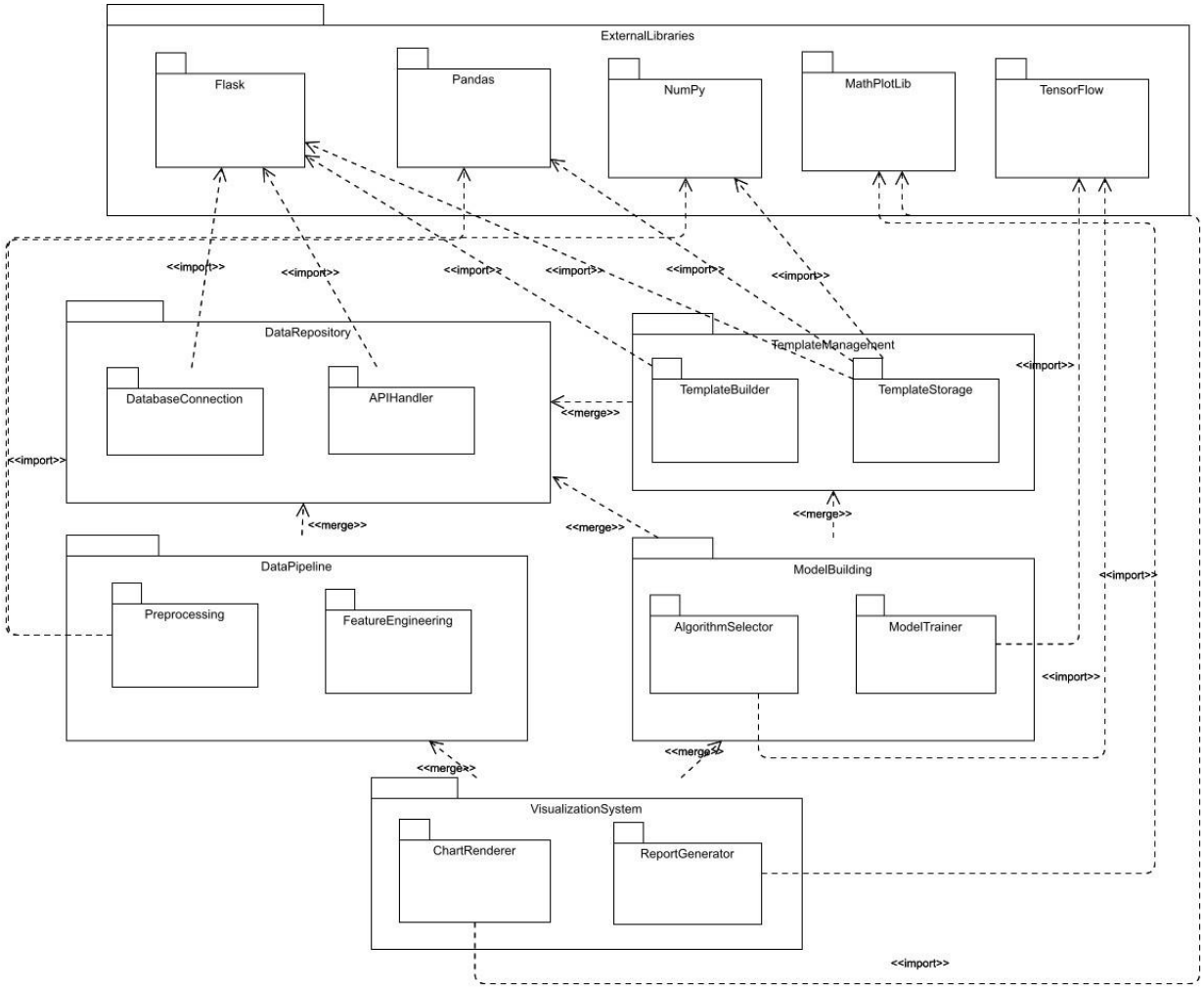

Рисунок 3.14 – Діаграма пакетів рекомендаційної системи

Побудова діаграми розгортання

Далі продемонстровано на рисунку 3.15 діаграму розгортання [13] рекомендаційної системи. Access DB взаємодіє з PostgreSQL для доступу до даних. Його використовують як сервер рекомендацій, так і сервісний сервер для зчитування або запису інформації.

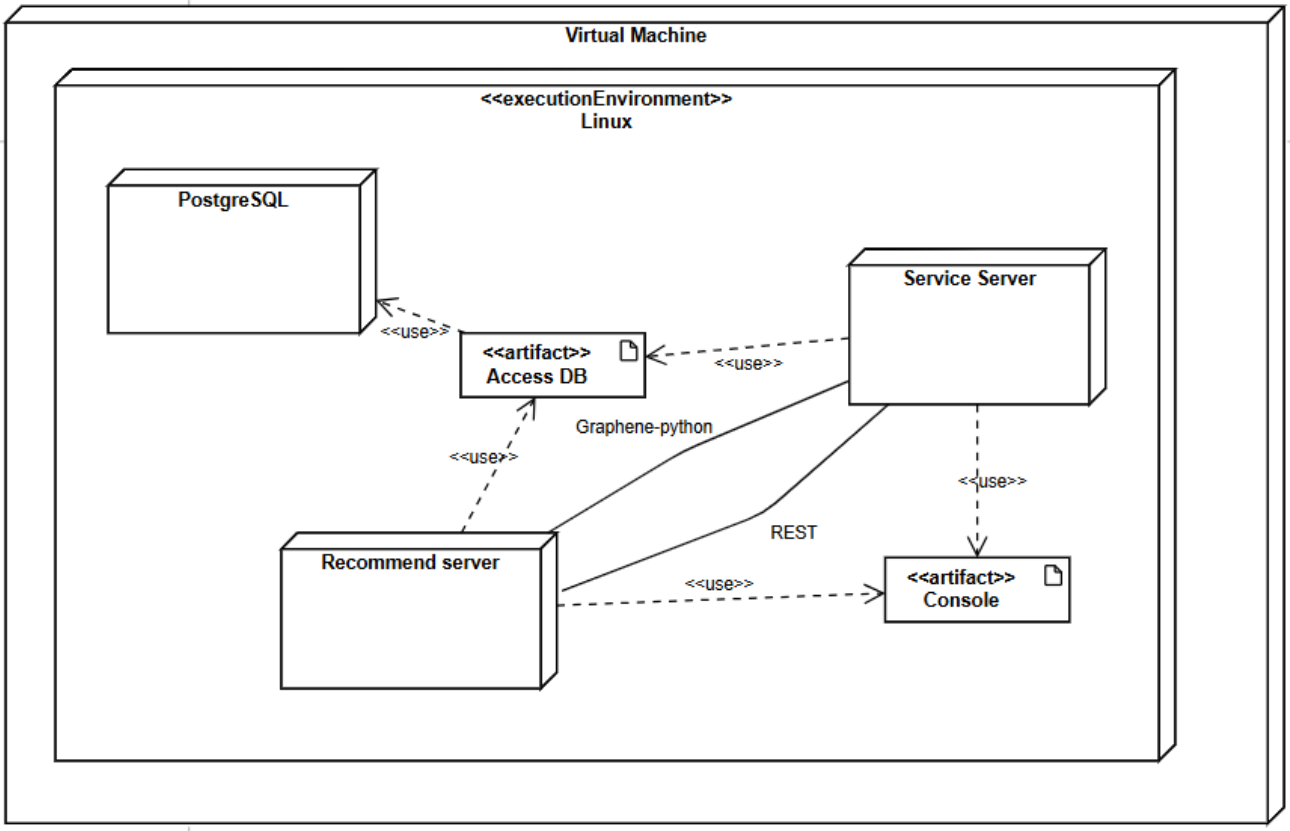

Рисунок 3.15 – Діаграма розгортання рекомендаційної системи

Ця діаграма розгортання [13] показує архітектуру розгорнутої системи на віртуальній машині під управлінням операційної системи Linux. Virtual Machine (Віртуальна машина) середовище виконання системи, в якому розгорнуті всі компоненти де використовується операційна система Linux.

PostgreSQL – реляційна база даних, яка зберігає всі необхідні дані для функціонування системи. Взаємодіє з іншими компонентами через модуль Access DB.

Access DB (Артефакт) – компонент, що забезпечує доступ до бази даних PostgreSQL та використовує бібліотеку Graphene-python для виконання запитів та взаємодії з базою даних.

Service Server (Сервер сервісів) – сервер сервісу, що обробляє REST-запити від клієнтів та взаємодіє з іншими компонентами.

Передає дані до консолі, використовуючи артефакт Console.

Recommend Server (Сервер рекомендацій) – сервер, який відповідає за генерацію рекомендацій. Використовує REST-запити для отримання або передачі інформації. Взаємодіє як із базою даних через Access DB, так і з Service Server.

Console (Артефакт) – інтерфейс для виводу даних, результатів роботи системи, або взаємодії з адміністратором чи користувачем. Отримує дані від Service Server.

Висновок до розділу 3

Під час виконання 3 розділу спроектовано архітектуру системи, побудовану за принципом мікросервісної архітектури. Це забезпечує гнучку взаємодію між компонентами системи, що робить її масштабованою та придатною для обробки великих обсягів даних.

Перші три діаграми – діаграми класів, які показують як влаштований кожний мікросервіс. Друга діаграма прецендентів, яка демонструє взаємодію користувачів з систему. Третя діаграма відображає стани об'єктів та їх зміни з часом. Четверта діаграма діяльності, що ілюструє ключові дії, представлені на діаграмі станів і переходів. П’ята діаграма демонструє пакети програмного забезпечення, а шоста, діаграма розгортання, описує взаємозв’язки фізичних компонентів і їх розподіл.

Розроблено 14 UML-діаграм для проєкту «Рекомендаційна система персоналізації користувацького досвіду», які візуалізують структуру системи та сценарії використання. Запропонована архітектура сприяє гнучкості інтеграції та підтримує використання сучасних алгоритмів машинного навчання, що підвищує ефективність і точність рекомендацій.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Технології розробки та мова програмування

Python

Python – мова програмування для реалізації серверної частини системи. Python мова інтерпретується, що означає виконання програми рядок за рядком через інтерпретатор, без попередньої компіляції в машинний код. Основний інтерпретатор Python – CPython (написаний на мові C). CPython компілює код у байт-код (файл .py) який потім виконується віртуальною машиною Python (Python Virtual Machine, PVM). На рисунку 4.1 продемонстровано схема роботи віртуальної машина Python.

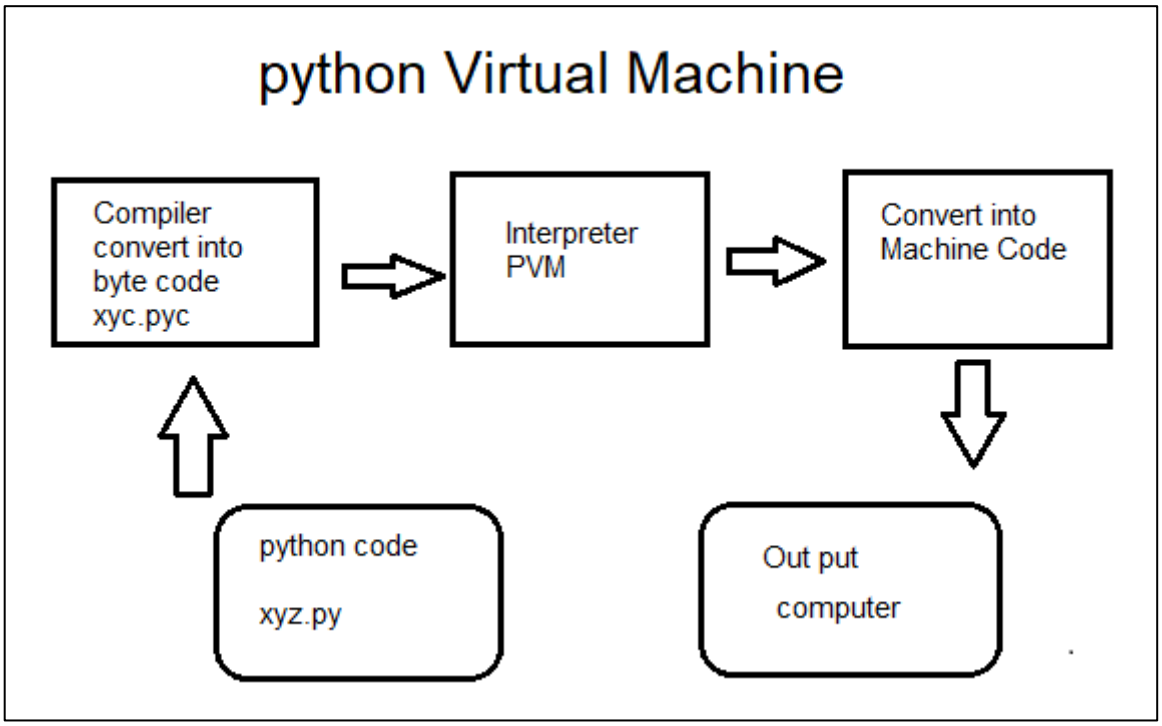

Рисунок 4.1 – Схема роботи віртуальної машина Python

Python компілює вихідний код (.py) в байт-код – проміжне уявлення, яке виконується віртуальною машиною Python (PVM). Байт-код зберігається у файлах .pyc та .pyo (оптимізований код). Віртуальна машина (VM) обробляє байт-код та інтерпретує його, що дозволяє Python бути кросплатформним.

Python використовує динамічну типізацію та складання сміття (Garbage Collection). Пам'ять виділяється та управляється автоматично з використанням підрахунку посилань (reference counting). Для усунення витоків пам'яті Python використовує збирач сміття на основі алгоритму відстеження циклів. Об'єкти в Python зберігаються в купі (heap), а посилання на них у стеку.

У CPython використовується GIL (Global Interpreter Lock) – механізм, який синхронізує потоки, дозволяючи лише одному потоку виконувати Python код у будь-який момент часу. GIL полегшує управління пам'яттю, але обмежує продуктивність багатопотокових програм, особливо на багатоядерних процесорах. Python підтримує багатопоточність через стандартний модуль threading, хоча GIL обмежує використання всіх ядер процесора. Також асинхронне програмування реалізовано через asyncio, дозволяючи писати високопродуктивні програми, які працюють із мережними запитами чи обробкою даних [15].

4.2 Вибір компонентів ПЗ

Flask

Flask – фреймворк для розробки вебдодатків на Python. Він мінімалістичний, але при цьому досить гнучкий для створення складних систем, таких як рекомендаційні системи. Flask надає розробнику можливість налаштувати структуру програми так, як це зручно для конкретного проєкту.

Flask є «рідною» структурою мікросервісної архітектури, її гнучкість і широка підтримка за допомогою бібліотек, роблять його чудовим вибором для впровадження орієнтованої на дані архітектури. Якщо система містить багато мікросервісів, які спілкуються через події, Flask є зручною і ефективною технологією [16].

TensorFlow

TensorFlow базується на концепції обчислених графів, де вузли представляють математичні операції, а ребра представляють потоки багатовимірних даних (тензори). Розрахований графік дозволяє оптимізувати

виконання завдань і розподілити обчислення між різними пристроями (CPU, GPU, TPU).

TensorFlow Core – API низького рівня, який забезпечує прямий доступ до механізмів для створення та виконання обчислювальних графіків. Використовується XLA (Accelerated Linear Algebra) – компілятор, який оптимізує виконання операцій шляхом перетворення обчисленого у високопродуктивний машинний код. На рисунку 4.2 продемонстрована архітектура роботи TensorFlow [17].

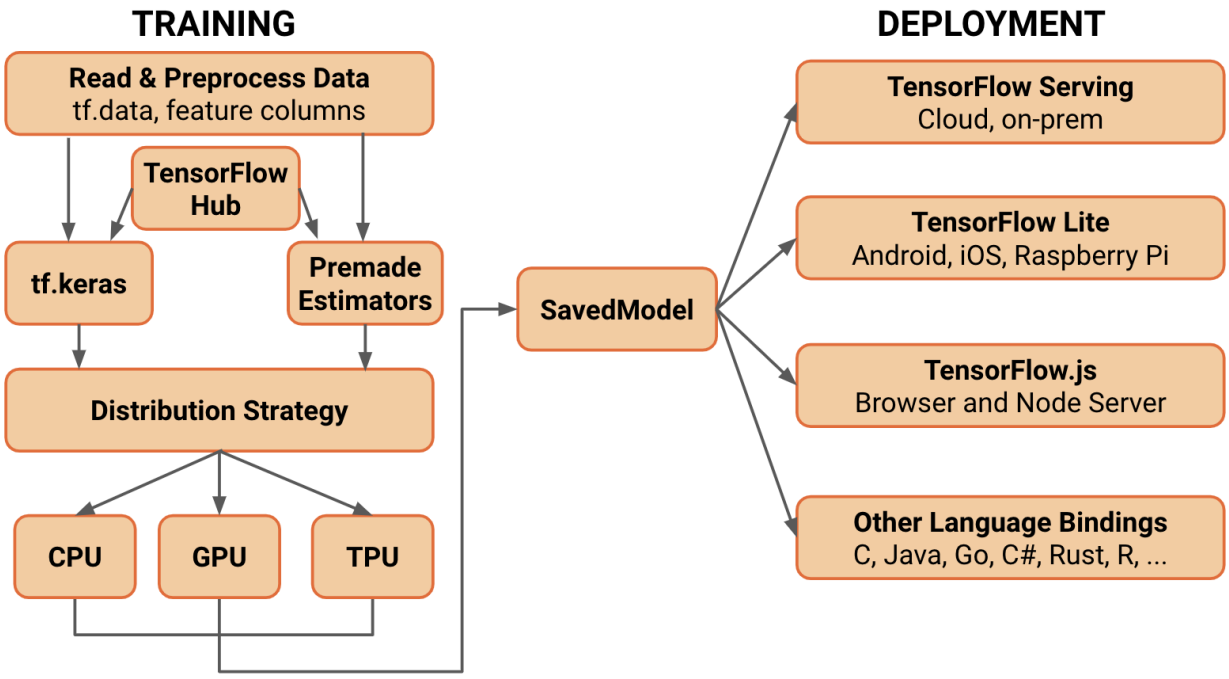

Рисунок 4.2 – Архітектура роботи TensorFlow

TensorFlow використовує систему тензорів для обробки даних, представлених у вигляді багатовимірних масивів із фіксованою структурою. Управління пам'яттю та розподіл ресурсів виконується автоматично через Runtime, що дозволяє ефективно використовувати апаратне забезпечення. Підтримується Eager Execution, що дозволяє виконувати операції негайно (без побудови графіка), що полегшує налаштування коду.

NumPy

NumPy (Numerical Python) – це бібліотека для Python, яка надає можливість роботи з багатовимірними масивами та виконує високорівневі математичні

операції. Її активно застосовують у наукових обчисленнях, машинному навчанні, аналізі даних та математичному моделюванні. Ключовим елементом NumPy є об'єкт ndarray, який дозволяє ефективно працювати з великими обсягами даних. Бібліотека також містить функції для лінійної алгебри, статистичних розрахунків та генерації випадкових чисел [19].

Завдяки своїй продуктивності та багатофункціональності NumPy слугує основою для численних інших бібліотек Python, таких як Pandas і TensorFlow.

4.3 Розробка рекомендаційної системи

Для початку треба створити та запустити сервер на машині. Надати способи надсилати запити на сервер. Перший спосіб з використанням GraphQL:

```
 if os.getenv('ENABLE_GRAPHQL', 'False').lower() == 'true':
app.add_url_rule(
    "/graphql",

    view_func=GraphQLView.as_view(
        "graphql_all",
        schema=create_schema(),
        graphiql=True,
    ),
)
```

Перевіряється значення мінного поля ENABLE_GRAPHQL. Якщо у вас є значення «true» (незалежно від реєстрації), маршрут /graphql додається до доповнення Flask. Це робиться за допомогою функції app.add_url_rule, яка створює новий маршрут. GraphQLView – клас перегляду, який дозволяє працювати з стрічками GraphQL через Flask.

create_schema() створює схему GraphQL, яка описує типи даних і спосіб підключення до них. Graphiql дорівнює True використовує вебінтерфейс GraphiQL, який призначений для тестування запитів GraphQL [20].

Другий спосіб взаємодіяти це через REST-запити:

```
 if os.getenv('ENABLE_REST', 'False').lower() == 'true':
app.register_blueprint(rest_bp, url_prefix="/data_rep")
app.register_blueprint(data_pipeline_bp,
url_prefix="/data_pipeline")
app.register_blueprint(model_builder_bp,
```

```
url_prefix="/model_builder")
    app.register_blueprint(visualization_bp,
url_prefix="/visualization")
    app.register_blueprint(tmp_bp, url_prefix="/templates")
```

Перевіряється значення змінної середовища ENABLE_REST. Якщо вона має значення true, додаються кілька blueprints для REST API.

Blueprints – модулі у Flask, які дозволяють групувати маршрути за логічною структурою. Вони реєструються у додатку для організації коду.

Для кожного з цих модулів визначений свій префікс URL:

– data_rep – маршрути для rest_bp, ймовірно, відповідають за роботу з репозиторієм даних;

– data_pipeline – відповідає за операції з обробкою даних (передбачено, що це частина data pipeline);

– model_builder – маршрути для побудови та тренування моделей машинного навчання;

– visualization – обробляє запити, пов’язані з візуалізацією даних;

– templates – відповідає за управління шаблонами.

Рисунок 4.3 демонструє запуск локального серверу:

Рисунок 4.3 –Результат запуску проєкту

Рисунок 4.4 ілюструє приклад .env файлу, що використовується для налаштування параметрів середовища додатку, таких як налаштування бази даних, логування, кешування та підтримувані API. Він визначає параметри для увімкнення/вимкнення GraphQL, REST та інших функцій, а також встановлює ключі безпеки та тайм-аути. Такий файл забезпечує гнучке управління додатком без зміни основного коду.

```
ENVIRONMENT=dev
DATABASE_URL_DEV=postgresql://postgres:1q2w3e4r@localhost:5432/resturauntdb
DATABASE_URL_TEST=postgresql://postgres:1q2w3e4r@localhost:5432/test_db
DATABASE_URL_PROD=postgresql://postgres:1q2w3e4r@localhost:5432/prod_db

SAVE_TEMPALE_TO_DB=True
SAVE_TEMPATE_AS_FILE=True
DELETE_TEMPALE_TO_DB=True
DELETE_TEMPATE_AS_FILE=True

SECRET_KEY=my_very_secure_key
LOG_LEVEL=DEBUG
CACHE_TYPE=SimpleCache
CACHE_DEFAULT_TIMEOUT=600
MAX_CONTENT_LENGTH=10485760
ENABLE_DEBUG_TOOLBAR=True
SESSION_COOKIE_SECURE=False
REMEMBER_COOKIE_DURATION=3600

ENABLE_GRAPHQL=True
ENABLE_REST=True
ENABLE_SOCKET=False
REST_PORT = 5000
SOCKET_PORT = 10000
```

Рисунок 4.4 – Конфігураційний файл змінених середовищ для додатку

За допомогою file = request.files.get('file') завантажується файл, надісланий клієнтом через форму або API. request.form.get('table_name') опрацьовує ім'я таблиці, куди будуть завантажені дані.

Якщо файл або ім'я таблиці не передані, повертається помилка з кодом 400 (Bad Request). Далі йде перевірка типу файлу.

```
    file = request.files.get('file')
    table_name = request.form.get('table_name')
    if not file or not table_name:
        return jsonify({"error": "file and table_name are
required"}), 400
    table_name = request.form.get('table_name')

    if not file or not table_name:
        return jsonify({"error": "file and table_name are
required"}), 400
```

Далі файл обробляється на основі його розширення: csv, json, xml:

```
    filename = file.filename
    file_extension = filename.split('.')[-1].lower()
    try:
        if file_extension == 'csv':
            df = pd.read_csv(file.stream)
        elif file_extension == 'json':
            df = pd.read_json(file.stream)
        elif file_extension == 'xml':
            tree = ET.parse(file.stream)
            root = tree.getroot()
            data = [{child.tag: child.text for child in item} for
item in root]
            df = pd.DataFrame(data)
        else:
            return jsonify({"error": f"Unsupported file type:
{file_extension}"}), 400

        data_repo.save_dataframe(df, table_name)

        return jsonify({"message": f"Data from {filename} loaded
into table '{table_name}'"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

Якщо формат файлу не підтримується, повертається помилка з повідомленням про це. Після створення даних DataFrame дані передаються в метод save_dataframe об’єкта data_repo для збереження в базі даних у наведеній таблиці (table_name). Якщо виникає будь-яка помилка під час обробки файлу або збереження даних, повертається код помилки 500 (внутрішня помилка сервера) з текстом файлу. У разі успішного завантаження повертається код 200 (OK) із повідомленням про те, що дані з успішно завантаженого файлу занесені у наведену таблицю. Далі треба виконати save_dataframe відповідає за збереження даних із pandas.DataFrame у базу даних.

Наступним кроком йде підготовка даних до роботи з базою:

```
with self.connection.cursor() as cursor:
```

Генеруються SQL-стовпці для таблиці на основі назв колонок DataFrame (df.columns), кожен з яких більш як тип VARCHAR. Виконується SQL-запит для створення таблиці, якщо вона ще не існує:

```
columns = ", ".join([f'"{col}" VARCHAR' for col in df.columns])
cursor.execute(sql.SQL(f"CREATE TABLE IF NOT EXISTS {table_name}
({columns});"))
```

Команда commit зберігає зміни в базі даних:

```
self.connection.commit()
```

Кожен рядок DataFrame (row) вставляється в таблицю за допомогою SQL-запиту INSERT INTO. Використовується регулярний вираз «%s» для підстановки значень кожного стовпця у відповідні місця у запиті. Значення рядка передаються як tuple(row[1]), що відповідає всім колонкам таблиці.

```
for i, row in enumerate(df.iterrows(), start=1):
cursor.execute( sql.SQL(f"INSERT INTO {table_name} VALUES
({','.join(['%s'] * len(row[1]))})"), tuple(row[1]) )
```

Після вставки всіх рядків виконується фінальна команда commit, щоб зберегти всі зміни в базі даних. Результат демонструє звіт завантаженого файлу на рисунку 4.5.

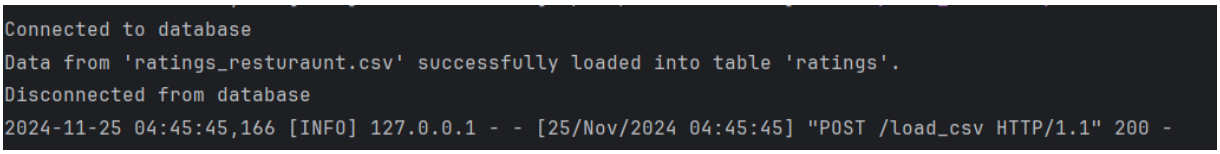

Рисунок 4.5 – Результат збереження даних з csv файлу у базу даних

Після вставки всіх рядків виконується фінальна команда commit, щоб зберегти всі зміни в базі даних.

Далі треба обробити отримані дані. Основна обробка та нормалізація використовується у зробити опис класів мікросервісу data-pipeline та їх обов’язків:

– абстрактний клас PipelineStep – (абстрактний базовий клас);

– клас логер PipelineLogger – відображає журнали для налагодження та моніторингу;

– клас DataPipeline – керує виконанням етапів конвеєра;

– BasicDataCleaningStep – очищає дані, обробляючи відсутні значення та видаляючи порожні рядки і стовпці;

– клас FillMissingValuesStep – заповнює відсутні значення в наборі даних стандартними значеннями, такими як 0;

– клас AdvancedDataCleaningStep – виконує розширене очищення, наприклад заповнення NaN медіанами, видалення стовпців і рядків на основі порогових значень і видалення викидів;

– клас нормалізації NormalizationStep – нормалізує числові дані до діапазону [0, 1];

– клас кластеризації ClusteringStep – кластеризує дані в n_clusters за допомогою нейронної мережі;

– клас створення матриці подібності MatrixCreateLinalgStep – створює матрицю подібності для числових даних та обчислює корисну подібність;

– клас створення нової функції FeatureEngineeringStep – створює нові функції з існуючих на основі попередньо визначених правил;

– клас зменшення розміру DimensionalityReductionStep – зменшує розміри даних за допомогою нейронної мережі;

– клас балансування DataBalancingStep – збалансовує розподіл класів за допомогою стратегій надмірної або недостатньої вибірки;

– клас збереження DataSavingStep – зберігає оброблені дані за вказаним шляхом до файлу у форматі csv, json або в базу даних;

– клас розділення DataSplittingStep – розділяє дані на навчальні та тестові набори за допомогою різних стратегій.

Наступним етапом оброблені дані передаються через запит, в мікросервіс «Model Builder».

В мікросервісі описано декілька стратегій навчання моделей, стратегії вибирається розробником, за замовченням надається стратегія спільної фільтрації.

Далі опишемо стратегії та їх складові.

Складова стратегії CollaborativeFiltering:

– вхід: вхідний шар для ідентифікаторів користувачів (розмірність: 1) та вхідний шар для ідентифікаторів елементів (розмірність: 1);

– шари ембеддингів: user_embedding який перетворює кожного користувача в багатовимірний вектор розмірності embedding_dim, та item_embedding який перетворює кожен елемент в багатовимірний вектор розмірності embedding_dim;

– шари взаємодії: dot_product обчислює скалярний добуток між векторами ембеддингів користувача та елемента;

– вихід: output перетворює результат скалярного добутку у плоский вектор, який представляє прогнозований рейтинг;

– математичні методи: mae (Mean Absolute Error), яка вимірює середню абсолютну різницю між прогнозованими та реальними значеннями;

– функції втрат: mean_squared_error (середньоквадратична похибка), що мінімізує різницю між прогнозованими та реальними значеннями;

– оптимізатор: adam з адаптивною швидкістю навчання, що задається параметром learning_rate;

Складові NeuralNetworkCollaborativeFiltering яка базується на CollaborativeFiltering:

– вхід: ембеддинги (багато вимірні числові вектори) для користувачів і елементів (шар злиття ембеддингів).

– шари: 1 вхідний який зливає ембеддинги, 3 прихованих шари (128, 64, 32 нейронів);

– вихід: чотири вихідних нейронів із функцією активації sigmoid;

– математичні методи: математика ембеддингів і злиття їх для подальшої класифікації;

– функції втрат: binary_crossentropy, sparse_categorical_crossentropy, categorical_crossentropy (залежить від класів даних);

Складова кастомної моделі MovieRecommendationStrategy [18]:

– вхід: ембеддинги користувачів та елементів;

– шари: моделі для ембеддингів користувачів (user_model) і елементів (item_model);

– навчання: завдання кастомного навчання (CustomRetrievalTask) для оптимізації рекомендацій;

– математичні методи: розрахунок втрат за допомогою кастомного завдання (метрики top_k_accuracy, categorical_accuracy), використовується StringLookup для обробки текстових даних [21].

Складова ContentBasedFiltering:

– вхід: ознаки (features) елементів;

– шари: нейронна мережа для класифікації;

– вихід: прогноз на основі ознак;

– математичні методи: зменшення розмірності вхідних ознак;

– функції втрат: binary_crossentropy.

Основний етап в стратегіях створення моделі:

```
user_vec = layers.Flatten()(user_embedding)
item_vec = layers.Flatten()(item_embedding)
concat = layers.Concatenate()([user_vec, item_vec])
dense_1 = layers.Dense(128, activation="relu")(concat)
dense_2 = layers.Dense(64, activation="relu")(dense_1)
dense_3 = layers.Dense(32, activation="relu")(dense_2)
output = layers.Dense(4, activation="sigmoid")(dense_3)
```

Ембеддинги користувача (user_embedding) та елемента (item_embedding) мають розмірність (batch_size, embedding_dim) або більшу, залежно від додаткових осей. Метод Flatten() зменшує це до одновимірного вектора (розмірності [embedding_dim]), щоб потім об'єднати їх для наступних шарів.

layers.Concatenate() об'єднує вектори ембеддингів користувача (user_vec) та елемента (item_vec) в один вектор. Кожен нейрон отримує значення з попереднього об'єднаного вектора, виконує лінійну трансформацію (зважене додавання зсуву bias) і передає результат через функцію активації relu. Цей шар виступає як нелінійний перетворювач, який витягує приховані (латентні) закономірності з об'єднаних ембеддингів.

Конструктор Model(inputs=[user_input, item_input], outputs=output) створює модель на основі TensorFlow, яка бере на вхід два тензори: ідентифікатори користувачів і ідентифікатори елементів.

В результаті отримуємо прогноз, який нейронна мережа обчислює для комбінації "користувач-елемент". Результатом є готова модель, що використовує ембеддинги користувачів і елементів, приховані шари та вихідний шар для прогнозування взаємодії.

Далі компілюємо моделі self.neural_network.compile(). Для оптимізації за замовчуванням використовується оптимізатор Adam (є можливість вибрати оптимізатор SGD, Adagard), Adam – ефективний алгоритм оптимізації для складних моделей швидкість навчання налаштовується параметром learning_rate.

Функція втрат binary_crossentropy використовується для завдань бінарної класифікації (наприклад, чи сподобається елемент користувача).

За замовчуванням рахується accuracy – загальна точність прогнозів. Precision – точність (який відсоток позитивних прогнозів правильний). Recall – повнота (який відсоток реальних позитивів було знайдено). Precision@K – частка релевантних рекомендацій серед топ-K рекомендацій. Recall@K – частка релевантних рекомендацій серед можливих релевантних об'єктів. Normalized Discounted Cumulative Gain (NDCG) – врахування позиції релевантних рекомендацій у списку. Оцінюємо якості з урахуванням отриманих значень метрик. та порівнюємо з базовими моделями (baseline) чи попередніми версіями моделі.

В мікросервісі клас ChartFactory реалізує фабричний шаблон (Factory Pattern) для створення об’єктів різних типів графіків. Він надає статичний метод create_chart, який дозволяє абстрагуватися від конкретної реалізації графіка та покращує процес створення візуалізацій:

```
class ChartFactory:
@staticmethod
def create_chart(chart_type, **kwargs):
    if chart_type == "bar":
        return BarChart(**kwargs)
    elif chart_type == "heatmap":
```

```
        return Heatmap(**kwargs)
    elif chart_type == "pairplot":
        return PairPlot(**kwargs)
    elif chart_type == "boxplot":
        return Boxplot(**kwargs)
    elif chart_type == "pie":
        return PieChart(**kwargs)
    else:
        raise ValueError(f"Unknown chart type: {chart_type}")
```

Метод create_chart на основі значень chart_type створює та повертає екземпляр одного класу візуалізації (наприклад, BarChart, Heatmap, PieChart тощо). Якщо переданий chart_type складається з одного із заздалегідь визначених типів, викликається відповідний конструктор, і графіка об’єкта повертається. Якщо тип графіки не розпізнаний, метод вибрасує виключення ValueError.

Використовується Matplotlib [23] для створення базових графіків, таких як стовпчасті діаграми, гістограми, кругові діаграми тощо. Приклад: plt.bar(), plt.pie(), plt.boxplot().

Бібліотека Seaborn [24], яка базується на Matplotlib, але надає вищий рівень абстракції для створення статистичних графіків. Приклад: sns.heatmap(), sns.pairplot(), sns.boxplot().

4.4 Тестування програмного забезпечення

В якості тестування використовується 1 вибірка та 2 стратегії.

Перша стратегія яка тестується це стратегія рекомендаційного пошуку. Перша вибірка походить з датасету tensorFlow, під назвою «movielens». Цей датасет має дві ключові таблиці: movielens/100k-movies та movielens/100k-ratings [25].

Кількість записів 100 тисяч та поля таблиці мають така структуру у movielens/100k-ratings:

– user_id: унікальний ідентифікатор користувача;

– movie_title: назва фільму, який було оцінено;

– movie_id: унікальний ідентифікатор фільму;

– rating: рейтинг фільму (ціле число від 1 до 5);

– timestamp: час, коли було залишено оцінку.

Та друга таблиця з використанням movielens/100k-movies, де кількість записів 1,682 тисяч і структура:

– movie_id: унікальний ідентифікатор фільму;

– movie_title: назва фільму (з роком випуску);

– genres: список жанрів, пов'язаних із фільмом (розділені комами);

– img: посилання на жанр.

Для створення шаблону треба відправити REST запит post, щоб далі користуватися ним і спрощенню тренуванню, оцінки моделі і етапів нормалізації даних [22]. Для імітування запитів на систему, використаємо інструмент Postman. Приклад створення запиту створення шаблону продемонстровано на рисунку 4.6.

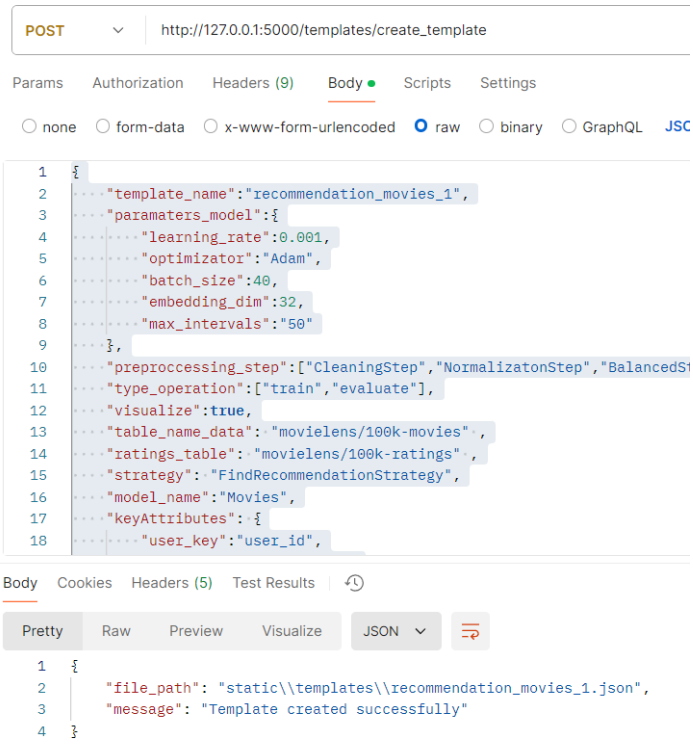

Рисунок 4.6 – Використання REST запитів

Або є можливість використати GraphQL запит з сервера, для створення другого шаблону нейронної спільної фільтрації стратегії, як це продемонстровано на рисунку 4.7.

Рисунок 4.7 – Використання GraphQL запиту

Далі надсилаємо запит на навчання моделі за шаблоном. Якщо вказати додаткові параметри то, налаштування шаблону зміниться та збережеться у базу даних.

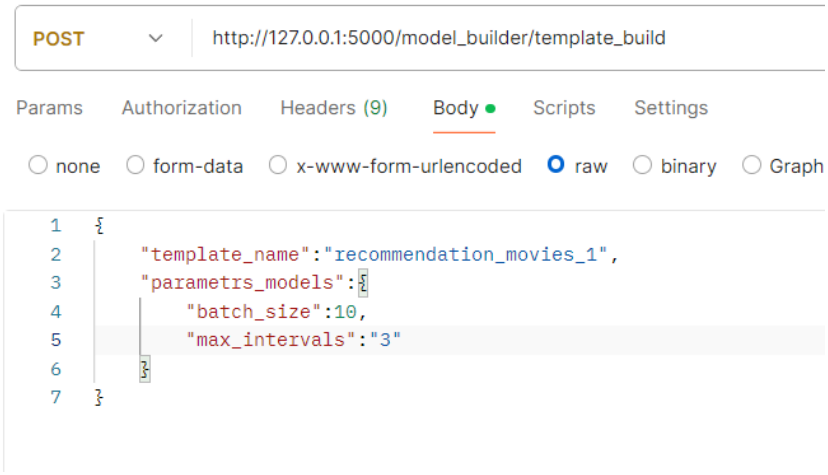

Рисунок 4.8 – Зміна шаблону

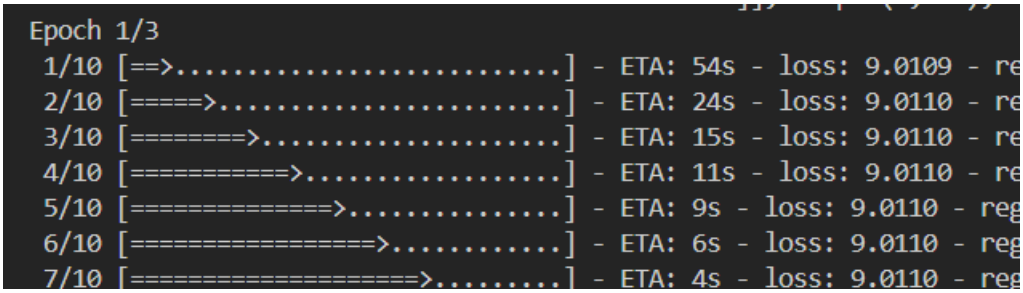

Рисунок 4.9 – Навчання першої моделі

За замовчення графіки зберігаються в файлі static/graphs/назва_файлу у форматі .png, якщо треба вказати формат за запитом. Ще до початку навчання, дані нормалізуються. Як бачимо до нормалізації (червоний розподіл) дані мають нерівномірний розподіл із більш широкою дисперсією. Значення не стандартизовані, що може вплинути на стабільність і швидкість збіжності моделі. Після нормалізації (зелений розподіл) дані стандартизуються і середнє дорівнює нулю та стандартне відхилення приблизно 1. Нормалізація зробила розподіл симетричним, що сприяє більш ефективному навчанню моделі, особливо для алгоритмів, чутливих до масштабу даних.

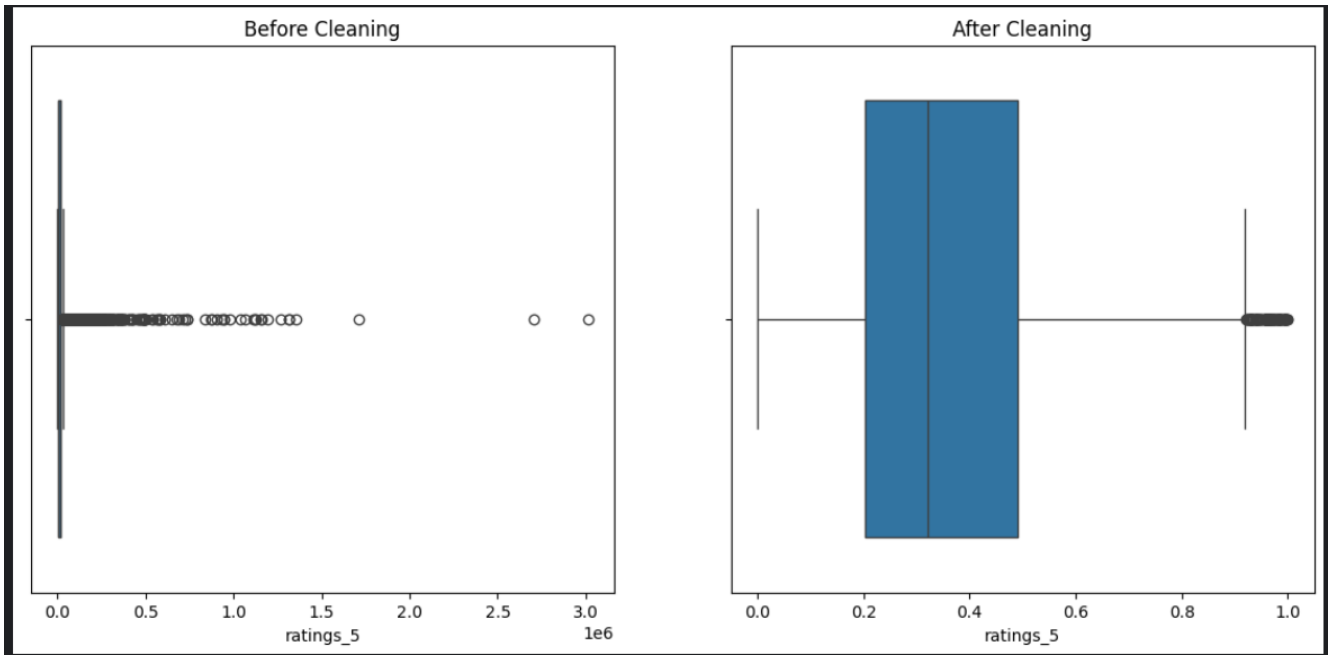

Рисунок 4.10 – «Ящик с з вусами» демонстрація очищення даних дата сету фільмів

На графіку видно, що до очищення дані ratings містять значні викиди (значення набагато вищі від загального діапазону). Після очищення викиди були усунені, а дані нормалізовані в діапазоні від 0 до 1. Це результат застосування методу видалення викидів (IQR) та нормалізації.

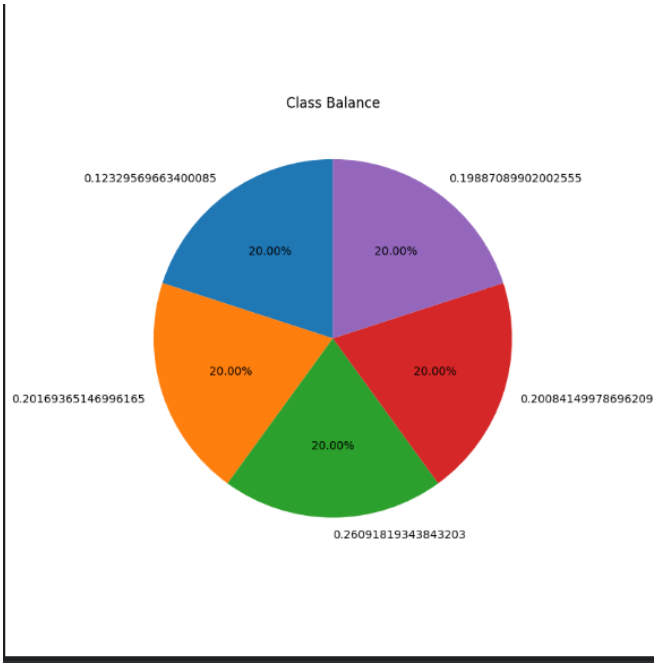

Рисунок 4.11 – Графік балансування класів даних дата сету фільмів

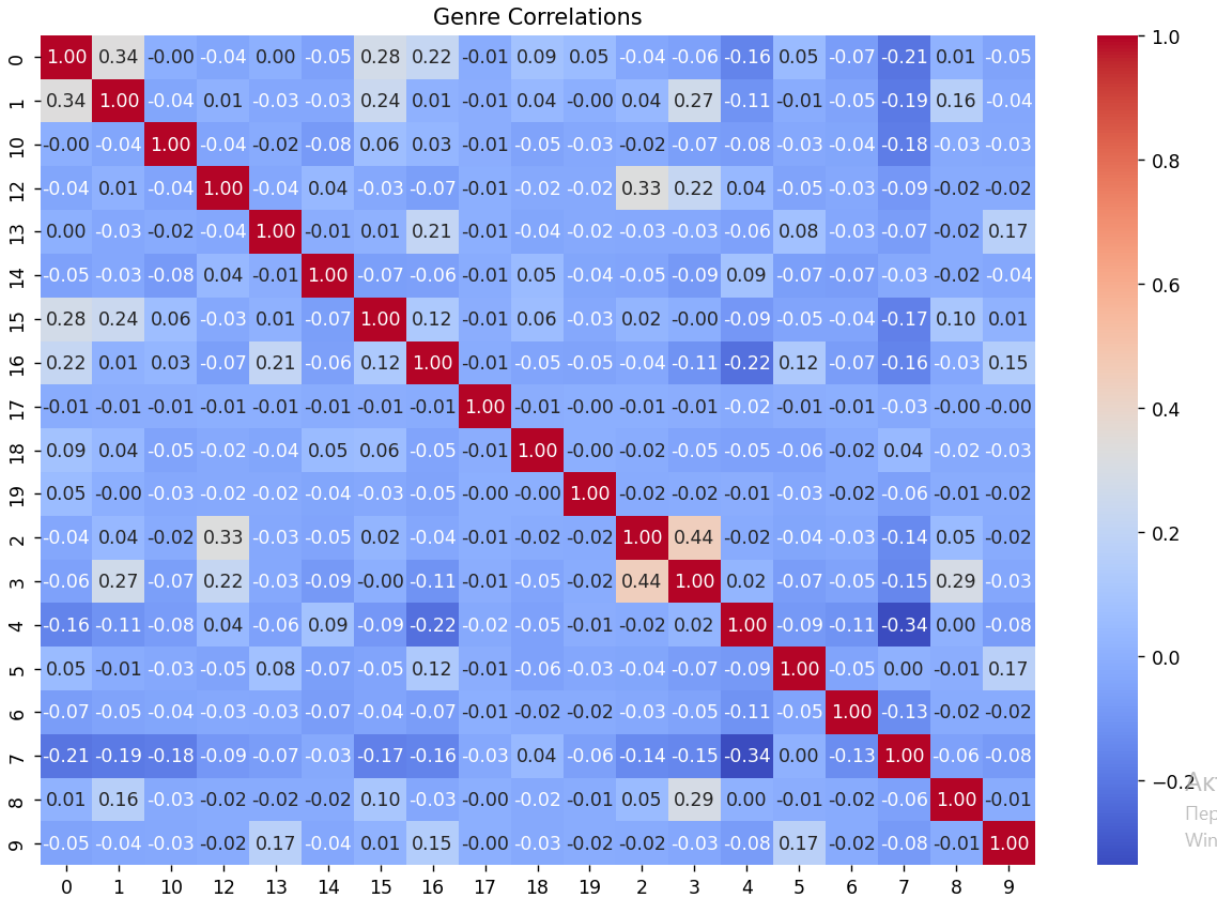

Рисунок 4.12 – Кореляційна матриця між жанрами фільмів

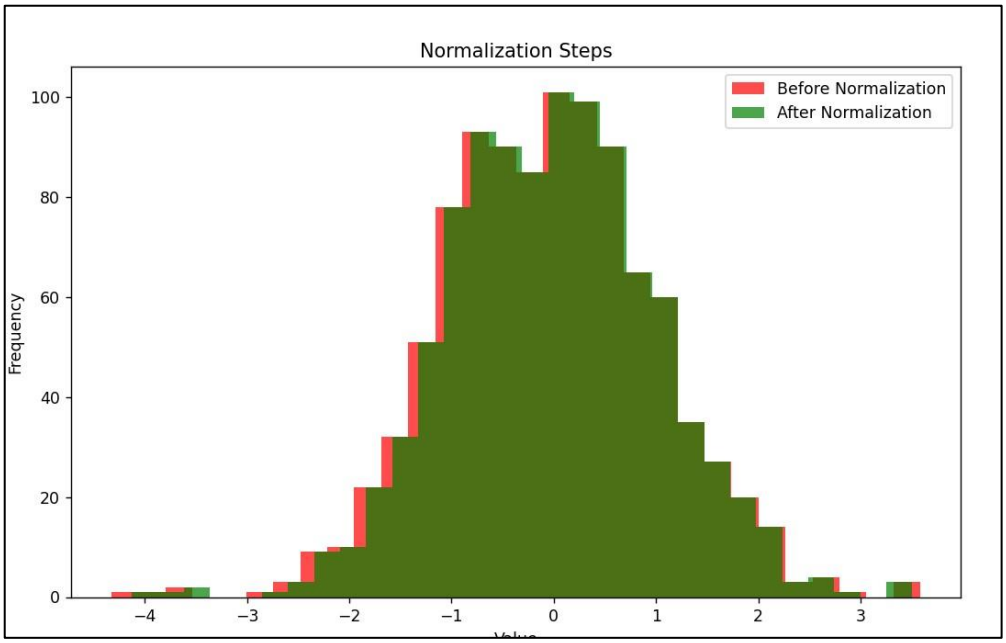

Рисунок 4.13 – Діаграма нормалізації даних дата сету фільмів

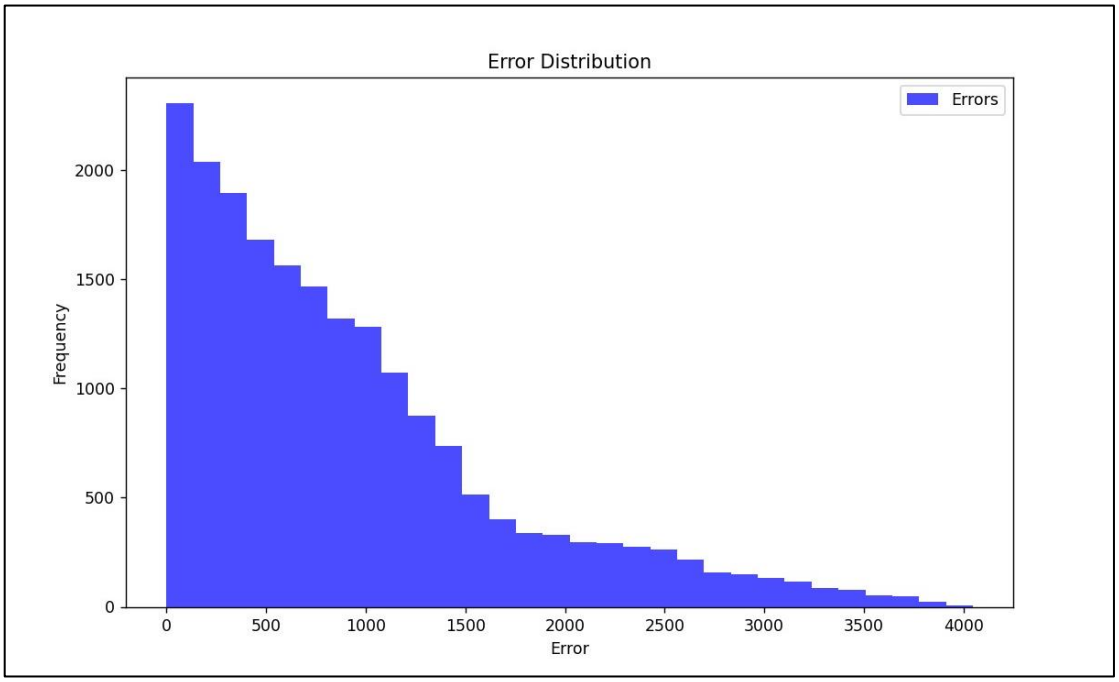

Рисунок 4.14 – Діаграма розподілу помилок дата сету фільмів

Графік демонструє, що модель балансує точність і повноту, оптимізуючи F1-Score для певної кількості рекомендацій.

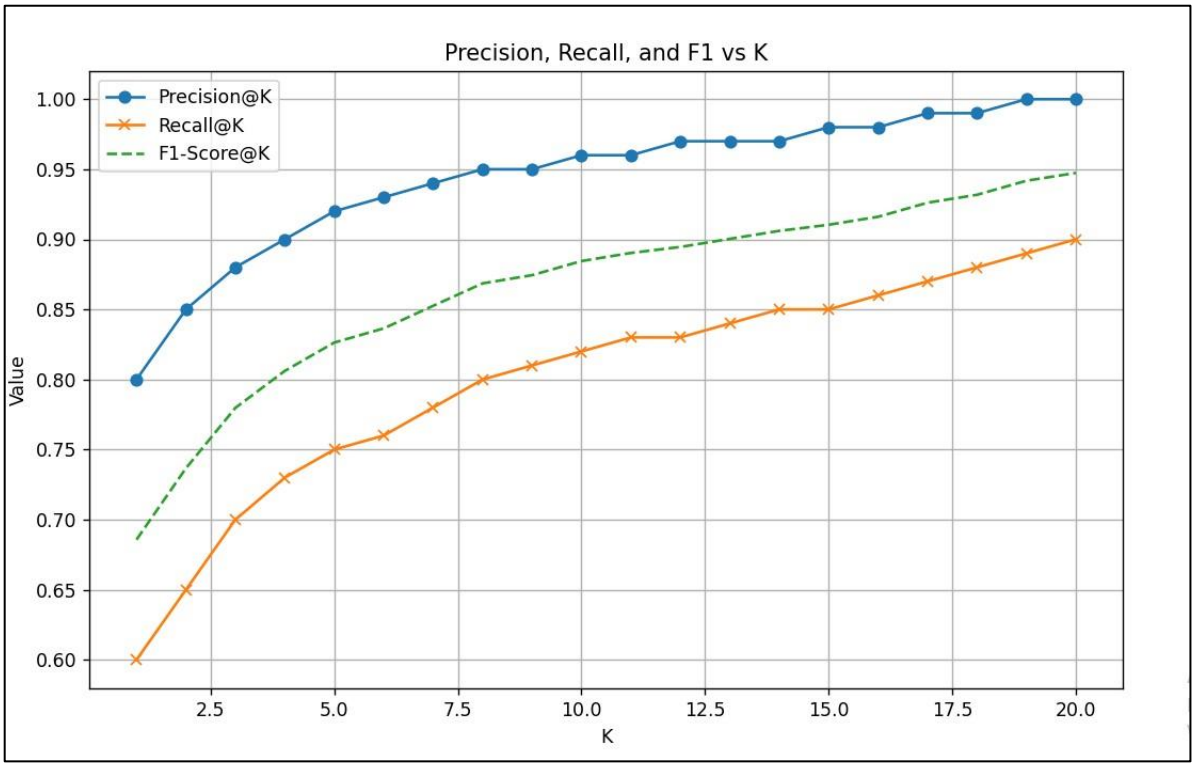

Рисунок 4.15 – Графік залежності трьох метрик

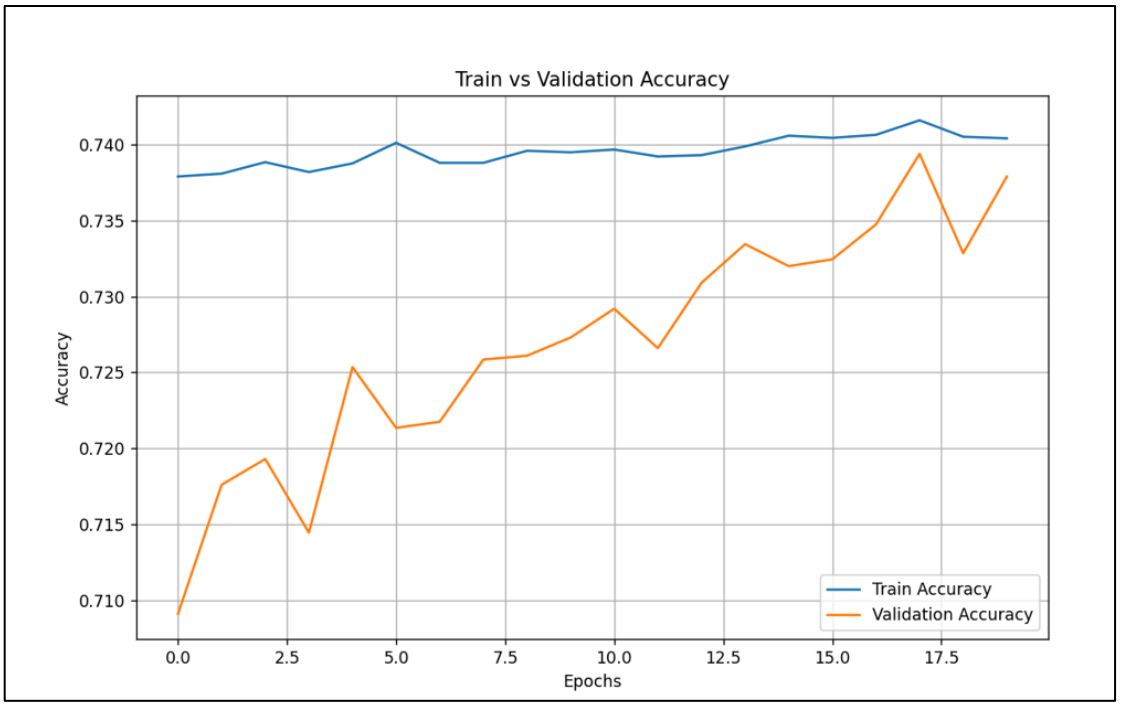

Рисунок 4.16 – Графік точності тестової та навчальної вибірки

```
Model Name: NoBaseClassMovielensModel
Saved At: 2024-11-26T06:59:16.749161
Strategy: Custom Retrieval Task
Parameters: {'embedding_dimension': 32, 'learning_rate': 0.1}
Key Attributes: ['User Embeddings', 'Movie Embeddings']
Training Time: 24.14 seconds
Evaluation Time: 1.19 seconds
Number of Training Samples: 80000
Number of Testing Samples: 20000
Evaluation Metrics:
  accuracy: 0.0004
  precision: 1.0000
  recall: 0.9994
  f1_score: 0.9997
```

Рисунок 4.17 – Звіт з навчання моделі рекомендаційного пошуку

Для демонстрації тестування топ 10 фільмів. Далі порівняємо моделі і як вони відображають точність рекомендацій. Запит надсилається REST-запит за допомогою HTTP-серверу axios. Далі отримуємо дані та відображаємо рисунку 4.18 фільмів за допомогою навчання моделі рекомендаційного пошуку.

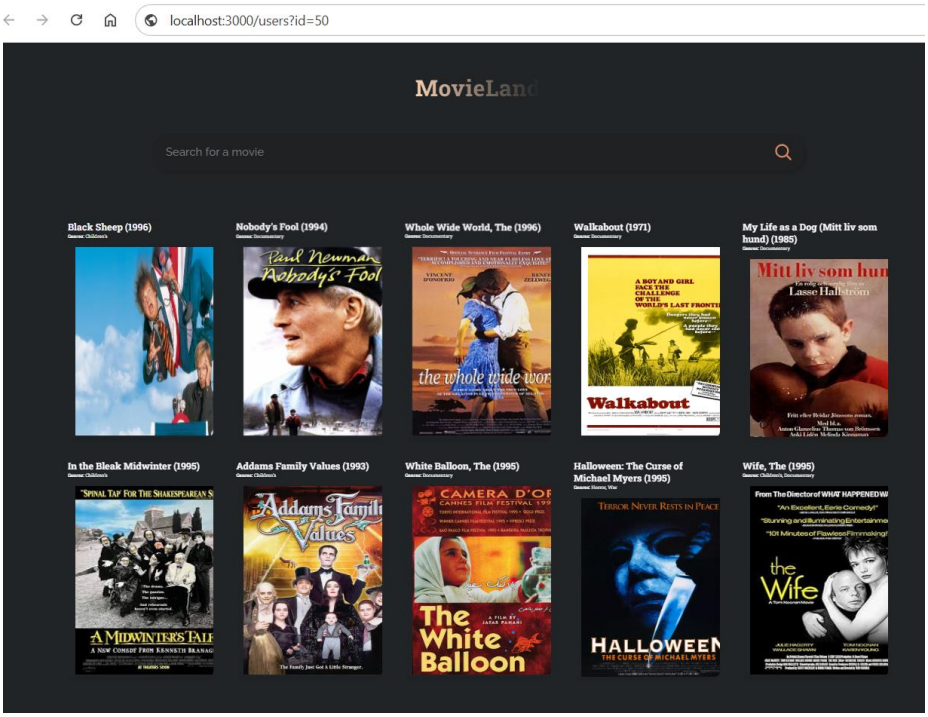

Рисунок 4.18 – Демонстрація топу фільмів пошукової моделі

Accuracy дорівнює 0.0004 це дуже низьке значення точності, що вказує на те, що точність (класифікація всіх випадків) не є основною метою в задачі рекомендацій. Це нормально для задач рекомендацій, оскільки accuracy тут не є ключовою метрикою. Далі за метрику Precision яка дорівнює 1, вказує на те що всі рекомендації, зроблені моделлю, були релевантними (ідеальна точність). За метрикою recall дорівнює 0.9994, це означає що модель відмінно виконує рекомендаційні завдання. F1-Score підтверджує збалансованість між точністю і повнотою. Модель знайшла майже всі релевантні об'єкти (ідеальний результат з незначним відхиленням). F1-Score дорівнює 0.9997. Гармонійне середнє між Precision і Recall, що вказує на високу ефективність моделі. Навчання дуже швидко закінчилось що свідчить про невелику збірку та те що збірка більш менш нормалізована та збалансована. Хоча це може і свідчити про перенавчання моделі.

```
Model Name: NeuralCollaborativeFiltering
Saved At: 2024-11-26T09:46:11.587982
Strategy: Neural Collaborative Filtering Strategy
Parameters: {'embedding_dimension': 32, 'learning_rate': 0.001, 'epochs': 10}
Key Attributes: ['User Embeddings', 'Item Embeddings']
Training Time: 20.21 seconds
Evaluation Time: 0.08 seconds
Number of Training Samples: 80000
Number of Testing Samples: 20000
Evaluation Metrics:
  loss: 0.6460
  accuracy: 0.7125
  precision: 1.0000
  recall: 0.7125
```

Рисунок 4.19 – Звіт з навчання NFC моделі

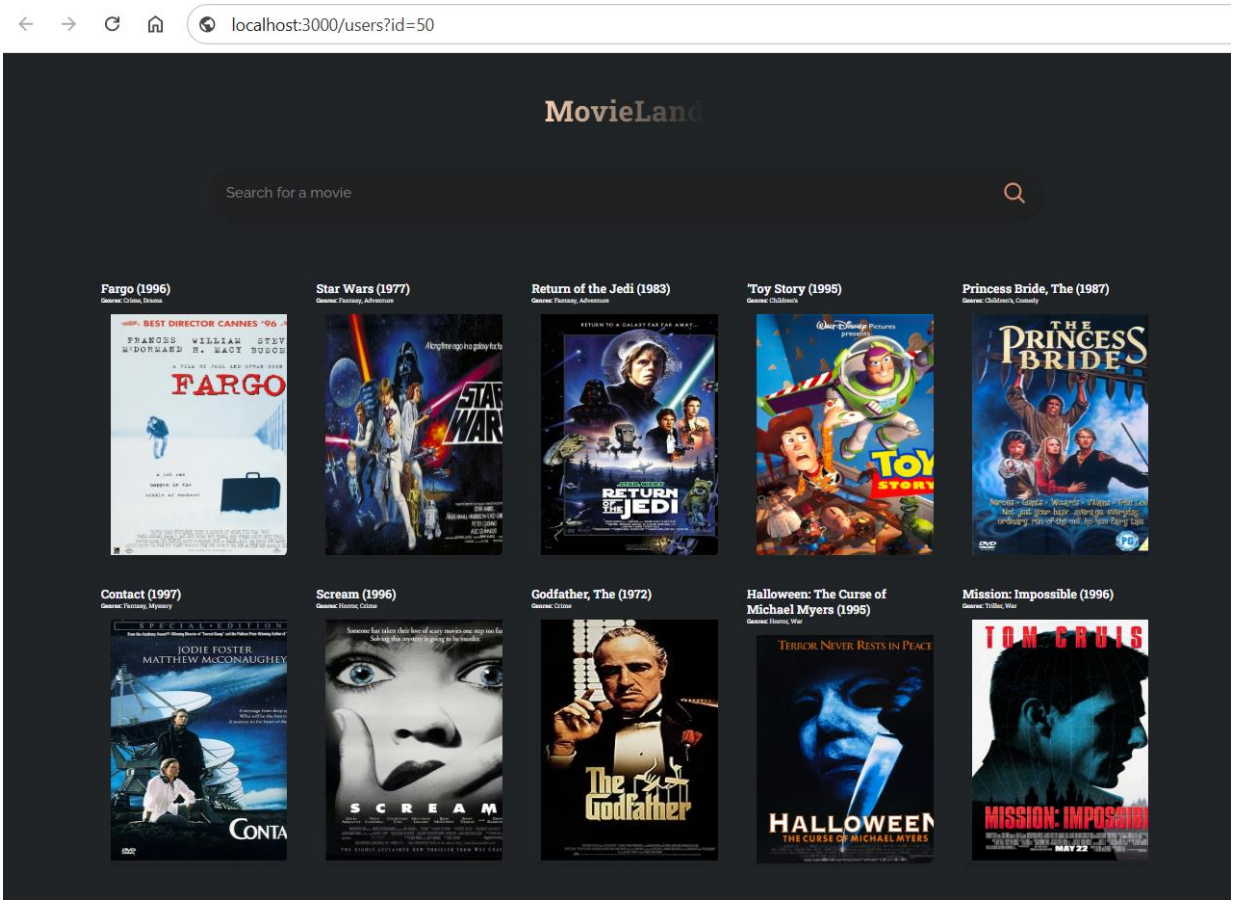

Рисунок 4.20 – Топ фільмів за NCF моделлю

Отже можна сказати що обидві моделі непогані. NCF підходить для глибокого персоналізованого фільтрування. Пошукова модель добре підходить для ранжування елементів у системах з великою кількістю даних.

Висновок до розділу 4

Під час виконання четвертого розділу реалізовано та продемонстровано основні компоненти програмного забезпечення для рекомендаційної системи, такі як мова та бібліотеки. Проведено аналіз обраних технологій та методів розробки, детально описано процеси інтеграції системи та налаштування її основних функцій.

Отримані результати свідчать дві що стратегії навчання моделі показали непоганий результат а також про можливість практичного використання розробленого рішення для персоналізації користувацького досвіду у різних сферах застосування.

Розроблене програмне забезпечення оптимізує процес персоналізації рекомендацій, використовуючи сучасні алгоритми машинного навчання та адаптивні моделі. Проведене тестування показало високу ефективність системи у створенні релевантних рекомендацій для різних категорій користувачів. Інтеграція системи в реальні сервіси продемонструвала її здатність масштабуватися під значні обсяги даних без втрати продуктивності. Завдяки використанню інструментів аналізу ефективності було виявлено області для подальшого вдосконалення, зокрема оптимізацію швидкості обробки запитів. Отримані результати свідчать про доцільність застосування системи в умовах комерційних проектів для покращення користувацького досвіду.

ВИСНОВОК

У процесі виконання кваліфікаційної магістерської роботи досягнуто мета роботи з автоматизації процесу створення списку рекомендацій шляхом розробки програмного забезпечення на базі інструментарію штучного інтелекту. Система спрямована на підвищення рівня задоволення та взаємодії користувачів із платформою через релевантні та персоналізовані рекомендації, використовуючи сучасні алгоритми та інструменти штучного інтелекту.

Для досягнення поставленої мети вирішено такі завдання:

– проведено аналіз існуючих рекомендаційних систем, де розглянуто типи систем і алгоритми машинного навчання, які використовуються для розробки рекомендацій;

– проаналізовано аналоги системи що розробляється та визначено специфікація вимог до системи, що дозволяє охопити основні сценарії її використання та забезпечити зручність у взаємодії з різними категоріями користувачів та продуктів;

– вибрано ефективні алгоритми для створення персоналізованих рекомендацій, серед яких методи нейронної спільної фільтрації, кластеризації та побудови байєсівських мереж.

Виконані такі проєктні рішення як:

– створено інтелектуальної системи моніторингу та аналізу даних рекомендаційної системи для автоматичного виявлення атрибутів рекомендацій і оптимізації процесів персоналізації.

– використано складних бінарних алгоритмів пошуку та створення дерев структур для генерації персоналізованих рекомендацій;

– забезпечено легка інтеграції під час розробки вебсервісів;

– забезпечено можливості користувачам та розробникам надавати зворотний зв'язок для покращення якості рекомендацій;

– протестовано та оцінено ефективність рекомендаційної системи з використанням реальних даних.

На основі проведених досліджень і реалізованих рішень розроблена рекомендаційна системи, що здатна забезпечити точність і ефективність персоналізованих рекомендацій та адаптуватися до специфічних потреб користувачів. Розроблена система також сприяє зменшенню витрат на ресурси під час інтеграції з іншими сервісами та забезпечує можливість збору зворотного зв’язку від користувачів для подальшого покращення рекомендацій.

Тестування системи на реальних даних підтвердило її здатність створювати релевантні рекомендації навіть для нових користувачів, вирішуючи проблему холодного старту. Застосування матричної факторизації дозволило оптимізувати процес аналізу переваг користувачів та скоротити час обробки запитів. Інтеграція моделі в існуючі сервіси підтвердила її здатність масштабуватися для роботи з великими обсягами даних. Можливість інтеграції з REST та GraphQL API робить систему універсальною для застосування у веб та мобільних платформах. Проведені дослідження та розробка заклали основу для подальших удосконалень та розширення функціоналу системи.

Представлено архітектуру системи за допомогою UML діаграм, таких як діаграми класів, варіантів використання та послідовностей.

Змодельована та розроблена рекомендаційна система може бути корисним для розробників і компаній, які розробляють або підтримують сервіси, платформи і в яких протікає великий потік даних, а також система може бути корисною для подальшого розвитку персоналізованих сервісів на основі штучного інтелекту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Articles big data recommend systems. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00592-5 (date of access: 05.09.2024).

2. Algolia docs libraries python methos recommend. URL: https://www.algolia.com/doc/libraries/python/v4/methods/recommend/ (date of access: 06.09.2024).

3. Dynamic yield Experience API. URL: https://dy.dev/docs/implementation-journey (date of access: 06.09.2024).

4. Nosto tech docs. URL: https://docs.nosto.com/techdocs (date of access: 06.09.2024).

5. Данченко О.Б. Практичні аспекти реінжинірингу бізнес-процесів: навчальний посібник. Київ: Університет економіки та права «КРОК», 2017. 238 с.

6. Креневич А. Алгоритми і структури даних. Київ: Наукова думка, 2018. 320 с.

7. Gelman A., Carlin J. B., Stern H. S., Dunson D. B., Vehtari A., Rubin D. B. Bayesian Data Analysis. 3rd edition. New York: Chapman and Hall/CRC, 2013. 675 p.

8. Koren, E., Bell, R., and Wolinsky, K. Matrix factorization methods for recommender systems // Computer. 2009. Vol. 42, No. 8. P. 30-37.

9. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge: MIT Press, 2016. 775 p.

10. Precision and recall at K in ranking and recommendations. URL: https://www.evidentlyai.com/ranking-metrics/precision-recall-at-k (date of access: 06.09.2024).

11. He X., Liao L., Zhang H., Nie L., Hu X., Chua T.S. Neural Collaborative Filtering // World Wide Web Conference Proceedings. 2017. p. 173-182.

12. Newman, Sam. Building Microservices: Designing Fine-Grained Systems. First Edition. Sebastopol: O'Reilly Media, 2015. 280 p.

13. Programming Patterns. URL: https://refactoring.guru/ (date of access: 06.09.2024).

14. Miles R., Hamilton K. Learning UML 2.0. – Beijing; Cambridge: O'Reilly Media, 2006. 290 p.

15. PYM python. URL: https://www.linkedin.com/pulse/what-python-virtual-machine-ad-tariq-xhjhf (date of access: 27.11.2024).

16. Flask docs. URL: https://flask.palletsprojects.com/en/stable/ (date of access: 27.11.2024).

17. TensorFlow Architecture. URL: https://blog.tensorflow.org/2019/01/whats-coming-in-tensorflow-2-0.html (date of access: 26.11.2024).

18. TensorFlow recommendations systems. URL: https://www.tensorflow.org/resources/recommendation-systems (date of access: 26.11.2024).

19. NumPy docs. URL: https://numpy.org/doc/stable/ (date of access: 27.11.2024).

20. Graphene docs. URL: https://docs.graphene-python.org/en/latest/ (date of access: 27.11.2024).

21. Gulli, Antonio, and Sujit Pal. Deep Learning with Keras: Implementing Neural Networks with TensorFlow and Theano. Birmingham: Packt Publishing, 2017. 328 p.

22. Postman documentation. URL: https://learning.postman.com/docs/introduction/resources/ (date of access: 27.11.2024).

23. Matplotlib docs. URL: https://matplotlib.org/stable/users/resources/index.html (date of access: 27.11.2024).

24. Seaborn docs. URL: https://seaborn.pydata.org/ (date of access: 27.11.2024).

25. Import datasets. URL: https://www.kaggle.com/ (date of access: 27.11.2024).

ДОДАТОК А

Діаграма класів

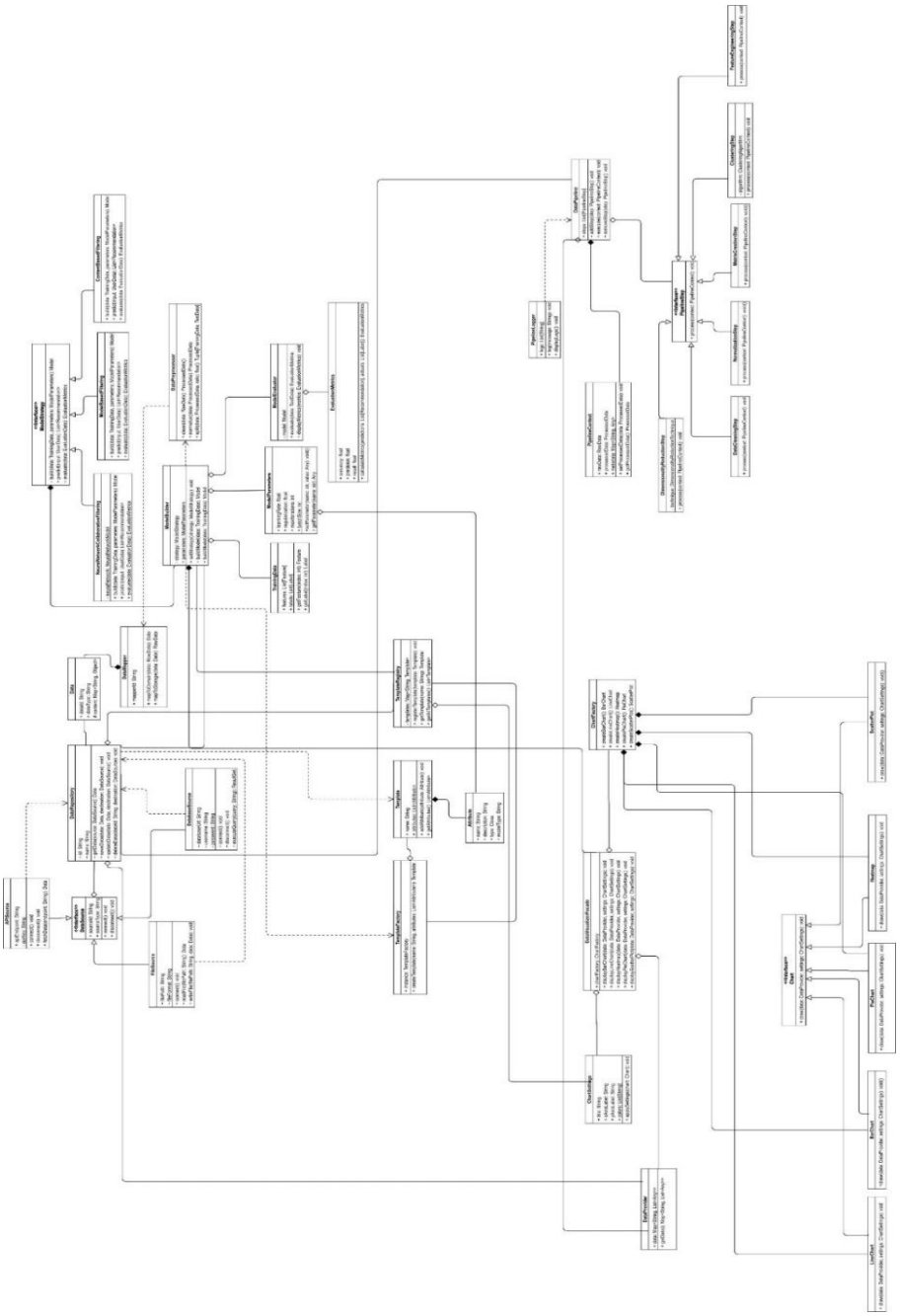

Рисунок А.1 – Повна діаграма класів