НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

В.о.завідувача кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Сервіс рекомендаційної системи з використанням методів колаборативної фільтрації»

Виконав:
студент IV курсу, групи КА-64
Овчар Антон Сергійович _____

Керівник:
в.о. завідувача кафедри, доцент, к.т.н
Тимощук Оксана Леонідівна _____

Консультант з економічного розділу:
доцент, к.е.н. Шевчук О.А. _____

Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А.Є. _____

Рецензент:
доцент, к.т.н. Булах Б.В. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
В.О.Завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Овчару Антону Сергійовичу

1. Тема роботи «Сервіс рекомендаційної системи з використанням методів колаборативної фільтрації», керівник роботи Тимощук Оксана Леонідівна, к.т.н., доцент, затверджені наказом по університету від «25» травня 2020 р. №1143-с

2. Термін подання студентом роботи 08 травня 2020 року

3. Вихідні дані до роботи _датасет, що складається з 5953 користувачів, 44200 питанньь та 1888865 відповідей та коментарів

4. Зміст роботи Розділ1. Рекомендаційні системи Розділ2. Математичне обргунтування і аналіз моделі. Розділ3. Розробка програмного продукту Розділ4. Функціонально-вартісний аналіз програмного продукту

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) Слайди № 2,3,4,6,7,8,9

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	доцент, к.е.н. Шевчук О.А.	21.04.20	28.05.20

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	14.04.20	
2	Аналіз існуючих підходів	21.04.20	
3	Проектування сервісу	29.04.20	
4	Підбір тестових даних	05.05.20	
5	Розробка програмного продукту	15.05.20	
6	Тестування програмного продукту	18.05.20	
7	Оформлення дипломної роботи	20.05.20	

Студент А.С. Овчар

Керівник роботи О.Л. Тимощук

* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

РЕФЕРАТ

Дипломна робота містить 116 с., 8 табл., 22 рис., 2 дод. та 16 джерел.

РЕКОМЕНДАЦІЙНІ СИСТЕМИ, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, ГІБРИДНА КОНТЕНТНО-КОЛАБОРАТИВНА МОДЕЛЬ, LIGHTFM, RESTFUL API, PYTHON.

Об’єкт дослідження – підходи до рекомендації контенту, а саме методи колаборативної фільтрації та їх гібридних модифікацій для розв’язання проблеми рекомендацій для ресурсів з великими об’ємами інформації.

Предмет дослідження – моделі рекомендаційних систем, а саме гібридна контентно-колаборативна модель.

Мета роботи – аналіз існуючих підходів до рекомендації контенту та побудова сервісу рекомендаційної системи, що не залежить від предметної області, взаємодія з яким відбуватиметься за допомогою протоколу HTTP.

У ході виконання роботи було проведено аналіз існуючих підходів до рекомендації контенту та порівняння гібридної контентно-колаборативно моделі порівняно с класичними підходами; описано процес проектування сервісу рекомендаційної системи та аналіз створеного програмного продукту.

В рамках подальшої роботи доцільно покращувати результати існуючої моделі шляхом додавання нових характеристик для користувачів та об’єктів, проводити експерименти з гібридизації системи з іншими підходами та розширювати функціонал системи для більш точного її налаштування.

ABSTRACT

Diploma thesis includes: 116 p., 8 tables, 22 fig., 2 add. And 16 references.

RECOMMENDER SYSTEMS, COLABORATIVE FILTERING, HYBRID CONTENT-COLABORATIVE MODEL, LIGHTFM, RESTFUL API, PYTHON.

The research objects are different approaches to content recommendation, especially collaborative filtering methods and their hybrid modifications in order to solve the recommendation problem of the big data resources.

The research subjects are recommender systems models, especially hybrid content-colaborative model.

The research purpose is an analysis of the existing approaches to content recommendation and building a subject independent recommender system service that provides RESTful HTTP user interface.

During the research there were existing approaches to content recommendations analyzed, hybrid content-colaborative model compared with classic approaches, recommender service designed and created.

As for further research, it is recommended to enhance existing model with user and item features, experiment with recommender system hybridization and to expand system functionality in order to tune the system more precisely.

ЗМІСТ

ВСТУП

Починаючи з 90-х років XX-го століття ми можемо спостерігати вибухове зростання об’ємів інформації, що доступна суспільству. Об’єм цієї інформації є значно більшим за той, що здатна самостійно обробити людина, з метою знайти необхідну їй інформацію.

Люди намагаються впоратись з цією проблемою власними силами та силами інших людей, але такий підхід все одно не вирішує дану проблему. По-перше, значна кількість інформації втрачається через те, що вона не знаходиться поза їх зоною сприйняття. По-друге, уже без участі людини велика частина інформації відфільтровується. Так, наприклад, редактори газет та журналів обираються лише ті статті, які люди хочуть читати, а книжкові магазини вирішують книги випускати у продаж.

Таким чином, для згаданої вище проблеми, виникає необхідність залучення додаткових технологій фільтрації. Головним інструментом персоніфікованої фільтрації інформації є рекомендаційна система.

Метою цієї роботи є розробка програмного продукту, що виконує роль рекомендаційної системи не залежно від предметної області.

В першому розділі розглянуто поняття рекомендаційної системи, основні підходи до рекомендацій та основні проблеми таких систем.

В другому розділі вказані методи і алгоритми, що використовуються у сучасних системах. Також детально розглянуто один із найефективніших існуючих підходів до гібридної контентно-колаборативної фільтрації, а саме моделі LightFM.

У третьому розділі описано процес створення сервісу, його архітектура та приведені приклади взаємодії з ним.

РОЗДІЛ 1 РЕКОМЕНДАЦІЙНІ СИСТЕМИ

1.1 Поняття рекомендаційної системи

Рекомендаційні системи – це комплекс алгоритмів, програм та сервісів, головна мета яких заздалегідь передбачити, які об’єкти (товари, послуги, контент) будуть цікаві користувачеві, беручи за основу інформацію про користувача, що відома рекомендаційній системі (особисті уподобання, історія відвідувань і т.д.)[1].

В деякому сенсі, рекомендаційні алгоритмі є підкласом систем фільтрації інформації.

Рекомендаційні системи використовуються у багатьох сферах, більшість з яких тісно пов’язана з генеруванням неперервного потоку відео/музичного ряду (Netflix, YouTube, Spotify), забезпеченням товарами та послугами (Amazon) або донесенням контенту для платформу соціальних медіа (Facebook, Twitter). Також рекомендаційні системи знайшли своє застосування у сферах досліджень та фінансових послуг.

1.2 Основні типи рекомендаційних систем

1.2.1 Колаборативна фільтрація

Підхід колаборативної фільтрацій заснований на припущенні, що користувачі, які мали спільні інтереси та смаки у минулому, матимуть спільні інтереси та смаки у майбутньому. Тобто їм будуть подобатися одні і ті самі одиниці контенту. Система генерує рекомендації використовуючи лише інформацію про рейтингові профілі користувачів та об’єктів. Знаходячи схожих користувачів/об’єкти до поточного користувача/об’єкта, система генерує рекомендації базуючись на їх схожості[3].

Приведемо приклад. Розглянемо систему в якій вже існують записи про користувацькі вподобання, що зображені наступною таблицею (рисунок 1.1)

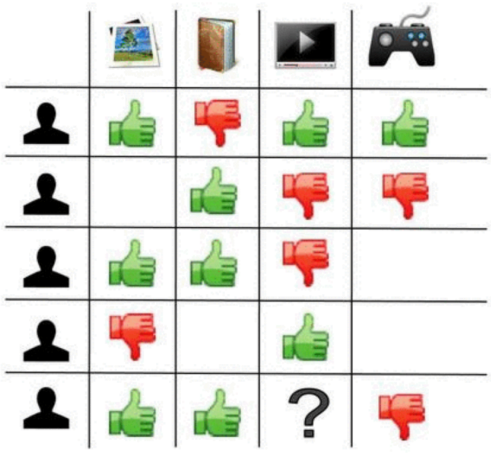

Рисунок 1.1 – Матриця вподобань користувачів

Система визначає схожих користувачів за їх історіями вподобань. Схожі користувачі відмічені у таблиці зеленим кольором (рисунок 1.2).

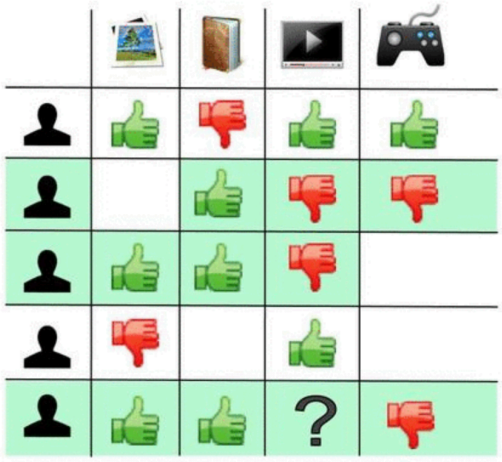

Рисунок 1.2 – Користувачі зі схожими інтересами

Постає питання, чи рекомендувати третій об’єкт п’ятому користувачу. Виходячи з вподобань користувачів зі схожими історіями ми доходимо висновку, що швидше за все піддослідному користувачу сподобається цей об’єкт. Отже, знак питання у таблиці (рисунок 1.3) змінюється на знак вподобання.

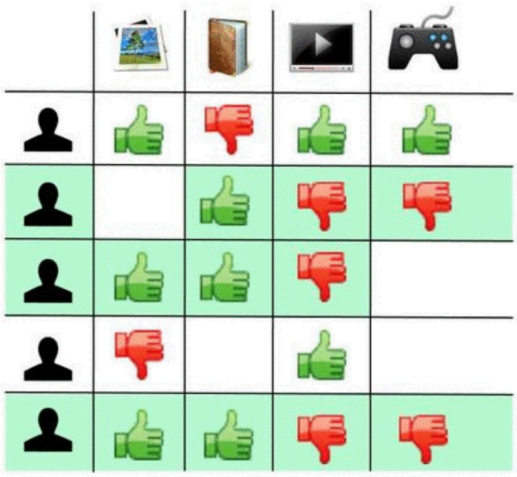

Рисунок 1.3 – Передбачення на основі досвіду схожих користувачів

Метод колаборативної фільтрації класифікують як той, що залежить від пам’яті системи та від моделі.

Однією з ключових переваг методу колаборативної фільтрації є факт того, що він не залежить від змісту об’єкту, що рекомендується. Тому, вона здатна до більш точних рекомендацій складних об’єктів таких, як, наприклад, фільми, не вимагаючи глибокого аналізу і «розуміння» системою його суті.

Існує багато алгоритмів, що використовуються для вимірювання схожості користувачів або об’єктів. Наприклад, підхід k-NN (k найближчих сусідів), або кореляція Пірсона.

Одним з найпопулярніших алгоритмів колаборативної фільтрації є item-to-item колаборативна фільтрація, розроблений компанією Amazon. Більшість соціальних мереж досі використовують колаборативну фільтрацію як частину гібридних систем по рекомендації друзів, груп або інших соціальних контактів.

1.2.1.1 Основні типи методів колаборативної фільтрації

На основі сусідства. Даний підхід є історично найпершим у колаборативній фільтрації та використовується у багатьох рекомендаційних системах. При даному підході для кожного активного користувача обирається

підгрупа користувачів, що мають схожі інтереси. У даного підходу можна виокремити наступні основні кроки:

Асоціювати ваговий коефіцієнт для кожного користувача з урахуванням схожості його оцінок та активного користувача.

Обрати декілька користувачів, що мають найбільший ваговий коефіцієнт. Тобто вони є максимально схожими на активного користувача. Дана група користувачів називається сусідами.

Розрахувати передбачення оцінок користувача для нових неоціненних предметів з урахуванням вагових коефіцієнтів та оцінок його сусідів.

На основі моделі. Даний підхід пропонує рекомендації, на основі вимірювання параметрів статистичних моделей для оцінок користувачі, що побудовані за допомогою таких методів, як байесовскі мережі, кластеризація, лантентна семантична модель, сингулярне розкладення, ймовірнісний латентний семантичний аналіз, прихоховане розподілення Діріхле та макрківскій процес прийняття рішень на основі моделей. Моделі розробляються з використанням інтелектуального аналізу даних, алгоритмів машинного навчання з ціллю знайти закономірності на основі даних для навчання. Кількість параметрів моделі може бути зменшено в залежності від типу за допомогою методу головних компонент.

Цей підхід є найбільш комплексним та дає найбільш точні прогнози, оскільки допомагає виявити приховані фактори, що пояснюють закономірності у розподіленні оцінок.

Даний підхід має декілька суттєвих переваг. Він оброблює розрідженні матриці значно краще, ніж підхід, що базується на сусідстві, що в свою чергу має великий вплив на властивість масштабування на великі набори даних.

Одним з найважливіших недоліків такого підходу є обчислювально складна процедура створення моделі. Необхідно підбирати компроміс між точністю моделі та її об’ємом, оскільки можна втратити корисну інформацію у зв’язку зі зменшенням кількості аспектів моделі.

Гібридний. Даний підхід об’єднує у собі підхід, що базується на сусідстві та підхід, що базується на моделі. Гібридний підхід є найбільш розповсюдженим при розробці рекомендаційних систем для комерційних сайті, так як він ефективно себе виявляє у питанні розв’язання проблеми оригінального підходу (того, що базується на сусідстві) та покращити якість передбачень. Такий підхід також дозволяє вирішити проблему розрідженості даних та втрати інформації. Але частіше за все, такий підхід складний та дорогий в реалізації.

1.2.1.2 Основні проблеми методу колаборативної фільтрації

Розрідженість даних. Як правило, більшість комерційних рекомендаційних систем базуються на занадто великій кількості об’єктів, у той час як більшість користувачів не виставляє оцінки товарам. У результаті цього матриця «об’єкт-користувач» отримується занадто великою та розрідженою, що може стати проблемою під час розрахунку рекомендацій. Ця проблема є особливо гострою для нових систем, що щойно з’явилися. Також розрідженість даних посилює проблему холодного старту

Масштабування. Зі збільшенням кількості користувачів в системі, з’являється проблема масштабування. Наприклад, маючи 10 мільйонів користувачів O(M) та мільйон предметів O(N), алгоритм колаборативної фільтрації зі складністю рівною O(MN) вже є занадто складним для розрахунків. Також більшість систем повинні моментально реагувати на онлайн запити від усіх користувачів, незалежно від історії їх покупок та оцінок, що потребує більшого масштабування.

Проблема холодного старту. Нові предмети або користувачі являють собою велику проблему для рекомендаційних систем. Частково уникнути цієї

проблеми дозволяє підхід, що базується на аналізі змісту об’єкту, так як він не опирається не на оцінки, а на атрибути, що допомагає додавати нові предмети до рекомендації для користувачів. Однак, цю проблему з додаванням нового користувача розв’язати значно складніше.

Синонімія. Синонімією називають тенденцію схожих або однакових предметів мати різні імена. Більшість рекомендаційних систем не здатні виявити ці приховані зв’язки і тому відносяться до цих предметів як до різних. Наприклад, «фільми для дітей» та «дитячий фільм» відносяться до одного жанру, але система сприймає їх як різні об’єкти.

Шахрайство. В рекомендаційних системах, де кожен користувач може залишати свої оцінки, люди можуть створювати позитивні оцінки своїм об’єктам або негативні своїм конкурентам. Також, рекомендаційні системи стали сильно впливати на продажі та прибуток, з тих пір як отримали широкий застосунок у комерційних сайтах. Це призводить до того, що недоброчесні продавці намагаються шахрайськими методами піднімати рейтинг своїх продуктів та знижувати рейтинг своїх конкурентів.

Різноманітність. Колаборативна фільтрація у першу чергу має на меті збільшити різноманітність та відкривати для користувачів нові продукти з їх незліченної кількості. Однак деякі алгоритми, наприклад ті які базуються на продажах та рейтингах, створюють дуже складні умови для просування нових та маловідомих продуктів, так як їх заміщують нові популярні продукти, які вже давно знаходяться на ринку. Це в свою чергу лише збільшує ефект «багатіші багатіють» та призводить до меншої різноманітності

Білі ворони. До білих ворон відносяться користувачі, чия думка постійно не співпадає з більшістю інших. Через їх унікальні смаки, їм неможливо що-небудь рекомендувати. Однак, такі люди мають проблеми з отриманням

рекомендацій у реальному житті, тому пошуку розв’язку цієї проблему у наш час не ведуться.

1.2.2 Фільтрація за змістом

Іншим популярним підходом в проектуванні рекомендаційної системи є фільтрація за змістом. Фільтрація за змістом базується на описі об’єкту та персональних вподобаннях користувача[5].

Такий підхід є найбільш ефективним у випадках, коли відома інформація про об’єкт (ім’я, опис, місцезнаходження), але невідома інформація про користувача. Задачею рекомендаційної системи за змістом є вивчити та класифікувати користувача по його вподобанням. Іншими словами, така система бере за основу твердження, що майбутні вподобання користувача будуть схожими до тих, що були у минулому.

Рекомендаційна система за змістом опирається на профіль об’єкту, що характеризує об’єкт у межах системи. Для того, щоб абстрагуватися від змісту об’єкту, використовуються алгоритми репрезентації. Одним із найпоширеніших алгоритмів є TF-IDF (term frequency-inverse document frequency) репрезентація. Її також називають репрезентацією у векторному просторі. Система створює профіль користувача базуючись на зважених векторах. Вагові коефіцієнти відповідають за важливість кожної з характеристик для користувача та можуть бути розраховані з індивідуально визначених контентних векторів за допомогою різних технік. Одним з найпоширеніших методів є усереднення значеннь векторів характеристик. Іншими більш складними є методи машинного навчання. Наприклад, Байесовскі класифікатори, кластерний аналіз, дерева рішень або штучні нейронні мережі. Вони використовуються для прогнозування вподобань користувача відносно об’єкту.

Одним із ключових недоліків такої системи є її здатність до аналізу об’єктів різних типів. Якщо система обмежена за типом рекомендації, тобто

рекомендує лише той тип об’єктів, які вже використовуються користувачем, то різноманіття рекомендацій суттєво зменшується. Наприклад, рекомендація новин була б значно кориснішою, якщо б окрім текстових статей, у рекомендаціях також були б присутні відео- так аудіоматеріали.

Одним із підвидів рекомендаційних систем за змістом, є система, що базується на думках користувачів. Якщо ресурс дозволяє користувачу залишати свою думку про об’єкт (наприклад, коментарі), то цю інформацію також можна використовувати як метаінформацію про об’єкт. Такий підхід широко використовується у таких техніках, як текстовий, інформаційний або сентиментний аналізи.

1.2.3 Багатокритеріальні рекомендаційні системи

Багатокритеріальні рекомендаційні системи можуть бути визначені як системи, що базуються на користувацьких побажаннях за декількома критеріями. Замість розробки методу рекомендації, що базується на єдиному значенні критерію, система намагається передбачити оцінку для невідомих користувачеві об’єктів за іншими критеріями, що впливають на загальне значення переваги. Деякі дослідники розглядають багатокритеріальні рекомендаційні системи як багатокритеріальні системи прийняття рішенн.

1.2.4 Рекомендація з урахуванням ризику

Більшість існуючих підходів до рекомендаційних систем у першу чергу зосереджуються на рекомендації найбільш актуального контенту не беручи до уваги той факт, що існує ризик потурбувати користувача небажаною інформацією або сповіщенням. Важливим є прийняття до уваги ризику засмутити користувача, пропонуючи йому об’єкт рекомендації у несвоєчасний момент. Наприклад, у нічний час. Тому, діяльність такої рекомендаційної системи має залежати від ступеню ризику рекомендаційного процесу. Одним

з підходів до вирішення цієї проблеми є використання DRARS (Dynamic Risk-Aware Recommendation System), що будує модель рекомендації в залежності від контексту. Така система комбінує рекомендації за змістом та алгоритм контекстуального бандиту (задача однорукого бандиту).

1.2.5 Рекомендація на основі місцезнаходження

Рекомендаційні системи, що базуються на містознаходженні, використовують смартфони з доступом до мережі Інтернет для рекомендацій, що сильно залежать від контексту. Ця галузь значно складніша для досліджень, адже мобільні дані значно складніші за ті, якими зазвичай оперують рекомендаційні системи. Вони є одночасно неоднорідними, зашумленими, потребують постійної просторової та часової автокореляції. Також вони часто мають проблеми з валідністю та узагальненням.

Існує три фактори, що можуть впливати на мобільні рекомендаційні системи та точність їх передбачень. Це контекст, рекомендації та приватність. В додаток до цього, мобільні рекомендаційні системи схильні до проблем з переносимістю. Наприклад, не всі рекомендації будуть влучними в усіх географічних регіонах. Немає сенсу рекомендувати страву, інгридієнти для якої неможливо знайти у даному регіоні.

Одним з прикладів застосунку рекомендаційних систем, що базуються на місцезнаходженні, є підходи, що розробили компанії Uber та Lyft для генерації маршруту для водіїв таксі у містах. Система використовує дані GPS про маршрути, на яких працюють водії таксі. Ці дані включають у себе геолокацію (довготу та широту), часову мітку та операційний стан (з пасажиром або без). Система агрегує ці дані для формування списку точок висадки та посадки для того, щоб оптимізувати прибутки та навантаження на маршрути.

1.2.6 Гібридні рекомендаційні системи

Більшість рекомендаційних систем сьогодні використовують гібридний підхід. Тобто комбінують підходи колаборативної фільтрації, фільтрації на основі змісту та інших[6]. Не існує однозначно виграшної комбінації для гібридизації. Користуються популярністю наступні підходи до гібридизації:

- Зважений: Комбінування рейтингів з різних рекомендаційних систем арифметично
- Перемикання: Обирання серед рекомендаційних компонентів та застосування найращого
- Змішаний: Рекомендації з різних систем комбінуються та пропонуються разом
- Характеристичне комбінування: Різні характеристики, що надходять з різних джерел інформації, використовуються різними алгоритмами та комбінуються в одну рекомендаційну систему
- Характеристична композиція: Рейтинг, що підрахований однією рекомендаційною системою, є характеристикою для наступної
- Каскадний: Рекомендації строго пріорітизуються. Тобто рекомендація з меншим рейтингом може мати перевагу над рекомендацією з більшим.

1.3 Оцінка ефективності рекомендаційних систем

Вибір правильної техніки оцінки рекомендаційного алгоритму грає чи не найважливішу роль у побудові моделі. Для вимірювання ефективності рекомендаційних систем використовуються три різних підходи: вивчення користувачів, online тест (A/B тестування) та offline тест.

Найбільш поширеними оцінками є сердньоквадратична похибка та нормалізоване середньоквадратичне відхилення. Також використовуються оцінки точності рекомендаційного підходу, такі як точність, повнота та DCG (discounted cumulative gain). Різноманітність, оригінальність та узагальненість

є також важливими показниками. Більш того, більшість класичних методів оцінки моделей активно критикуються у розробницькій спільноті.

Підхід вивчення користувачів використовуються лише на невеликих масштабах. Декілька десятків або сотень користувачів отримують рекомендації від різних систем, а потім дають свою оцінку відносно того, що їм сподобалось більше. При A/B тестуванні, рекомендації застосовуються до тисяч реальних користувачів продукту. Алгоритм випадковим чином обирається з щонайменше двох варіантів та генерує рекомендації. Ефективність такого підходу вимірюється за допомогою показників конверсії та CTR (click-trough-rate). Офлайн тестування в основному базується на історичних даних, тобто на попередніх оцінках користувачів.

Ефективність рекомендаційного підходу вимірюються за допомогою показників того, як добре система передбачає реальну поведінку користувача. Оскільки оцінка – це явна оцінка того, чи сподобалась рекомендація користувачу, вона не завжди може бути доступною. Наприклад, у випадку с новинними порталами, користувачі майже ніколи не оцінюють рекомендовану їм статтю. У таких випадках, офлайн оцінка ефективності може використовувати неявні показники ефективності. Наприклад, рекомендаційна система вважається ефективною, якщо вона здатна підібрати найбільшу можливу кількість статей, на які посилається поточна стаття. Хоча такий підхід часто критикується у спільноті дослідників. Наприклад, було показано, що офлайн оцінкі часто розходяться з оцінками, зробленими на основі реальних дослідженнь поведінки користувачів за допомогою A/B тестування.

Якщо йти далі, за поняття точності, то можна дійти висновку, що також важливими для рекомендаційної системи є наступні характеристики:

- Різноманітність – користувачі здебільшого задоволені рекомендаціями, в яких є досить різноманітна тематика
- Рекомендаційна впертість – в деяких ситуаціях може виявитись більш ефективним знову показати користувачу старий для нього об’єкт та надати йому можливість змінити свою оцінку. Для цього

є декілька вагомих причин. Наприклад, користувач міг проігнорувати об’єкт з першого разу, оскільки не мав часу для його детального вивчення.

- Приватність – більшість рекомендаційних систем мають справу з досить приватною інформацією про користувача. Побудова моделі колаборативної фільтрації може бути проблематичної з точки зору приватності користувацьких даних.
- Користувацька демографія – досліди показали, що користувацька демографія може виявляти значний вплив на оцінку роботи рекомендаційної системи. Проведені досліди виявили закономірність, що люди похилого віку більш схильні до слідування рекомендаціям, аніж молодь.
- Міцність – оскільки рекомендаційна система базується на зовнішніх даних, на неї можуть виявляти значний вплив зловмисники
- Несподіваність – це міра того, наскільки для користувача була несподіваною рекомендація. Наприклад, рекомендація купити молока у продуктовому магазині може бути дуже влучною, але у той самий час, вона є досить очевидною.
- Довіреність – користувач може не довіряти рекомендаційній системі, оскільки він може мати підозру відносно того, як генеруються рекомендації. Тому система повинна мати можливість пояснити саму себе
- Упередженість – користувачі схильні упереджено відноситись до більшості рекомендацій, що відмічені у системі як «Реклама». У той час як з іншими рекомендаціями у користувача проблем може і не бути.

1.4 Основні підходи до збору необхідних даних

У процесі роботи рекомендаційні системи збирають інформацію про користувачів. Більшість систем використовують комбінації явних та неявних методів.

1.4.1 Явний сбір даних

Явний сбір даних бере за основу взаємодію користувача з об’єктом та його оцінку якості рекомендації. Наприклад:

- Користувач оцінює запропонований об’єкт за диференційованою шкалою
- Користувач ранжує групу об’єктів від найкращого до найгіршого
- Користувач обирає найкращий з двох запропонованих об’єктів
- Користувачу пропонують створити список його улюблених об’єктів

1.4.2 Неявний сбір даних

Неявний збір даних використовує інформацію взаємодії користувача з ресурсом. Але ця інформація не є спеціалізованою і від користувача не вимагається додаткових особливих дій. Наприклад:

- Спостереження за ти, що користувач оглядає у інтернет-магазинах або базах даних іншого типу
- Ведення записі про поведінку користувача онлайн
- Відстеження вмісту комп’ютера користувача

1.5 Netflix Prize

Однією з подій, що підштовхнула дослідження у сфері рекомендаційних систем був конкурс «Netflix Prize»[7]. З 2006 по 2009 роки, компанія Netflix була організатором та спонсором змагань, головною винагородою у якому був

1 мільйон доларів для команди переможця. Завданням команди було проаналізувати набір даних, що складається зі 100 мільйонів оцінок кінофільмів, та розробити алгоритм рекомендації, що працював би хоча б на 10% точніше за той, що вже використовувала компанія. Це змагання підігріло інтерес спільноти дослідників та підштовхнула їх до створення більш ефективних та точних алгоритмів. 21 вересня 2009 року, головним призом у 1 мільйон доларів, була нагороджена команда BellKor's Pragmatic Chaos.

Найбільш точний алгоритм було розроблено у 2007 році. Він використовував ансамбль з 107 різних алгоритмічних підходів, що змішувались в єдине передбачення.

Інтернет в цілому отримав дуже багато від цього змагання. Більшість команд застосували свої розробки на інших ринках. Декілька учасників команди, що посіла друге місце, заснували Gravity R&D – рекомендаційне ядро, що і до сьогодні активно використовуються спільнотою розробників. 4-Tell, Inc. Розробила аналогічне проекту Netflix'у рішення для використання у сфері e-commerce.

Також виникла серія правових проблем навколо датасету, що пропонувався компанією Netflix для цих змагань. Хоча датасет було анонімізовано для приховання особистих даних користувачів, у 2007 році двоє дослідників з університету Техасу змогли встановити особистості окремих користувачів співставляючи оцінки з датасету з реальними оцінками на сайті Netflix. Як результат, у грудні 2009 року, анонімний користувач Netflix судився з компанією Netflix, на підставі того, що вони порушили закон Сполучених Штатів Америки про чесні угоди та Акт про захист відео кофіденційності. Цей судовий позов Федеральна Торгівельна Комісія використала як причину заборонити друге аналогічне змагання.

Висновки до розділу 1

У першому розділі було розглянуто поняття рекомендаційної системи та основні причини виникнення подібних систем. Також було розглянуто основні підходи, що використовуються у рекомендаційних системах.

Колаборативна фільтрація – метод в основі якого лежить принцип, що якщо користувачі мали схожі інтереси у минулому, то вони матимуть схожі інтереси у майбутньому.

Контентні рекомендації – метод в основі якого лежить алгоритм представлення об’єкту або його ознак у вигляді об’єкту n-вимірного простору та пошуку найближчих до нього об’єктів.

Гібридні рекомендації – метод в основі якого лежить комбінування різних підходів та алгоритмів за допомогою різних способів.

Також було розглянуто основні проблеми та переваги основних підходів до генерації рекомендацій та методи їх оцінки.

РОЗДІЛ 2 МАТЕМАТИЧНЕ ОБГРУНТУВАННЯ І АНАЛІЗ МОДЕЛІ

2.1 Розріджені матриці

Разріджена матриця – це матриця з переважно нульовими елементами. В іншому випадку, якщо більша частина елементів матриці є ненульовою, матриця вважається щільною[8]. Серед спеціалістів не існує єдиної думки на рахунок того, яка саме кількість нульових елементів робить матрицю розрідженою. Різні автори пропонують різні варіанти. Наприклад, для матриці порядку n число ненульових елементів:

- $O(n)$. Таке визначення можливо використовувати лише для теоретичного аналізу асимптотичних властивостей матричних алгоритмів.
- Для кожного рядка не перевищує 10 у типовому випадку
- Обмежене $n^{1+\gamma}$, де $\gamma < 1$
- Таке, що для даного алгоритму та обчислювальної системи має сенс отримувати вигоду з наявності в ній нулів.

2.1.1 Зберігання розрідженої матриці

Формат RR(C)O

Скорочена назва даного формату походить від англійського «Row-wise Representation Complete and Ordered» (рядкове представлення, повне та упорядковане). В даному форматі замість одного двовимірного масиву використовуються три одновимірних. Значення ненульових елементів матриці та відповідні їм стовпчикові індекси зберігаються в цьому форматі по рядкам в двох масивах AN та JA. Масив покажчиків IA, що використовується для посилання на компоненти масивів AN та JA, з яких починається опис нового рядка. Остання компонента масиву IA містить покажчик першої вільної

компоненти в масивах AN та JA, тобто рівна числу ненульових елементів матриці, збільшеному на одиницю. Наприклад, розглянемо матрицю:

$$A = \begin{pmatrix} 0 & 0 & 0 & 2 & 0 \\ 1 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \end{pmatrix}$$

Тоді її представлення в форматі RR(C)O набуватиме вигляду

$$IA = [1\ 2\ 4\ 4\ 5\ 6]$$
$$JA = [4\ 1\ 3\ 2\ 2]$$
$$AN = [2\ 1\ 3\ 6\ 4]$$

Тобто, масив AN містить всі ненульові елементи вихідної матриці A, масив JA номер стовбця в якому знаходиться відповідний елемент з AN та масив IA містить номер, з якого починається останній опис елементів в масивах JA та AN. Таким чином, інформація про елементи 2-го рядка матриці зберігається в елементах з $IA[2] = 2$ по $IA[3] - 1 = 3$ включно масиві JA та AN. Також, можна звернути увагу на те, що $IA[3] = IA[4] = 4$, а це означає, що 3-ій рядок матриці A нульовий.

В загальному випадку опис r-го рядка матриці A зберігається в компонентах з $IA[r]$ по $IA[r+1] - 1$ включно масивів AN та JA. Якщо $IA[r+1] = IA[r]$, то це означає, що r-ий рядок нульовий. Кількість елементів масиву IA на один більша за кількість рядків вихідної матриці, а кількість елементів в масивах JA та AN рівне числу ненульових елементів вихідної матриці.

Даний спосіб представлення називають повним, оскільки вся матриця A, впорядкованим, оскільки елементи кожної строки матриці A зберігаються у

відповідності до зростання стовпчикових індексів та рядковим, оскільки інформація про матрицю A вказується по рядкам.

Масиви IA та JA представляють собою портрет (структуру матриці A), що задається як множина списків суміжності асоційованої з A графу. Якщо алгоритм, що реалізує будь-яку операція над розрідженими матрицями, розбитий на етапи символічної обробки, на якому визначається портрет результуючої матриці, та чисельної обробки, на якому визначаються значення елементів результуючої матриці, то масиви IA та JA заповнюються на першому етапі, а масив AN – на другому.

Формат RR(C)U

Скорочена назва даного формату походить від англійського «Row-wise Representation Complete and Unordered» (рядкова представлення, повне, але невпорядковане). Формат RR(C)U відрізняться від RR(C)O тим, що в даному випадку зберігається впорядкованість рядків, але кожен елемент рядка вихідної матриці може зберігатися у довільному порядку. Для матриці A з попереднього прикладу досить було б використовувати й рядкове представлення, повне, але невпорядковане таке:

$$IA = [1\ 2\ 4\ 4\ 5\ 6]$$
$$JA = [4\ 3\ 1\ 2\ 3]$$
$$AN = [2\ 1\ 3\ 6\ 4]$$

Таке невпорядковане представлення може бути дуже зручним в практичних розрахунках. Результати більшості матричних операцій отримуються невпорядкованими (наприклад, операції вставки та видалення нових коефіцієнтів), а їх впорядкування коштувало б значних витрат машинного часу. В той самий час, за нечастими виключеннями, алгоритми для

розріджених матриць не потребують, щоб їх представлення були впорядкованими.

Зауваження

Декілька зауважень з приводу розглянутих форматів представлення:

- Очевидно, що представлення матриці в форматі RR(C)O також є і представленням в форматі RR(C)U, але не навпаки
- Із представлення матриці в форматі RR(C) неможливо отримати інформацію про точну кількість стовпчиків вихідної матриці
- Доцільно (з метою економії пам’яті) використовувати представлення RR(c) у випадку, коли матриця містить значну кількість нульових елементів.

2.1.2 Алгоритм множення розрідженої матриці на вектор

Важливим додатком цих алгоритмів є обчислення векторів Ланцоша, необхідне при ітераційному розв’язанні лінійних рівнянь методом спряжених градієнтів, а також при обчисленні власних значень та власних векторів матриці. Перевагою цих процедур, з обчислювальної точки зору, є те, що єдина необхідна матрична операція – це поворотний добуток матриці на послідовність заповнених векторів. Сама матриця не змінюється.

Ми розглянемо множення матриці загального виду, що зберігається в формі RR(C)U за допомогою масивів $IA,\ JA, AN$ на заповнений вектор-стовпець.

Математичний опис алгоритму

Вихідні дані: розріджена матриця загального виду A з елементами $a_{ij}(i = 1 \dots n, j = 1 \dots m)$. Заповнений вектор-стовпець b з елементами $b_j(j = 1 \dots m)$.

Дані, що обчислюються: заповнений вектор-стовпець c з елементами $c_i(i = 1 \dots n)$

Формули методу:

$$c_i = \sum_{k=1}^{l_i} a_{i,j=j(k)} b_{j=j(k)},$$

де l_i – кількість ненульових елементів рядку i матриці A, $j(k)$ – індекс k-го ненульового елементу матриці A.

Обчислювальне ядро алгоритму

Обчислювальне ядро послідовної версії добутку розрідженої матриці на вектор можна створити з множинних (всього n) обчислень скалярних добутків рядків матриці

$$c_i = \sum_{k=1}^{l_i} a_{i,j=j(k)} b_{j=j(k)}$$

Схема реалізації алгоритму

Вважатимемо, що розріджена матриця загального вигляду A зберігається у формі RR(C)U за допомогою масивів $IA,\ JA, AN$. Послідовність виконання методу наступна:

Виконувати для i від 1 до n

$$c_i = 0$$

$$IAA = IA[i]$$

$$IAB = IA[i+1] - 1$$

$$c_i = \sum_{k=IAA}^{IAB} AN[k] b_{JA[k]}$$

Після цього (якщо $i \leq n$) відбувається перехід до першого кроку зі збільшеним i.

Складність алгоритму

Для множення розрідженої матриці загального вигляду, що зберігається у формі RR(C)U, розміром $n * m$ на заповнений вектор $m * 1$ в послідовному (найшвидшому) варіанті необхідно: $O(l)$ додаваннь та $O(l)$ множеннь. Множення та додавання складають основну складність алгоритму.

По класифікації по послідовній складності, таким чином, алгоритм множення розрідженої матриці на вектор відноситься до алгоритмів $O(l)$.

Для множення розрідженої матриці загального вигляду, що зберігається у формі RR(C)U, розміром $n * m$ на заповнений вектор $m * 1$ в паралельному варіанті необхідно виконати наступні яруси: не більше ніж m додаваннь та множеннь (n обчислень на кожному з ярусів)

При класифікації по висоті ЯПФ, таким чином, алгоритм множення розрідженої матриці на вектор відноситься до алгоритмів зі складністю $O(m)$. При класифікації по ширині ЯПФ його складність буде $O(n)$

2.2 Сингулярний розклад матриці

2.2.1 Означення

Нехай матриця M порядку $m * n$ складається з елементів поля K, де K-будь-яке поле дійсних чисел, або будь-яке поле комплексних чисел.

Невід'ємне дійсне число σ називається сингулярним числом матриці M тоді і тільки тоді, коли існують два вектори одиничної довжини $u \in K^m$ та $v \in K^n$ такі, що

$$Mv = \sigma u$$
$$M^* u = \sigma v$$

Такі вектори u та v називаються, відповідно, лівим сингулярним вектором та правим сингулярним вектором, що відповідають сингулярному числу σ.

Сингулярним розкладом матриці M порядку $m * n$ є розклад наступного вигляду

$$M = U\Sigma V^*,$$

де Σ – матриця розмірності $m * n$ з невід'ємними елементами, в якої елементи, що розміщені на головній діагоналі – це сингулярні числа (а всі елементи, що не розміщені на головній діагоналі, є нульовими), а матриці U (порядку m) та V (порядку n) – це дві унітарні матриці, що складаються з лівих та правих сингулярних векторів відповідно (а V^* - це спряжено-транспонована матриця до V)

Унітарна матриця – квадратна матриця з комплексними елементами, результат множення якої на ермітово спряжену дорівнює одиничній матриці

$$U^\dagger U = UU^\dagger = I.$$

2.2.2 Приклад

Нехай дана матриця

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{bmatrix}$$

Одним із сингулярних розкладів цієї матриці є розклад

$$M = U\Sigma V^*,$$

де матриці U, Σ та V^* наступні:

$$U = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$V^* = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{bmatrix}$$

так як матриці U та V унітарні ($UU^* = I$ та $VV^* = I$, де I – одинична матриця), а Σ – прямокутна діагональна матриця, тобто $\Sigma_{ij} = 0$, якщо $i \neq j$.

2.2.3 Геометричний зміст

Нехай матриці A поставлений у відповідність лінійний оператор. Сингулярний розклад можна переформулювати в геометричних термінах. Лінійний оператор, що відображує елементи простору R^n в себе представимо у вигляді лінійних операторів обертання і розтягнення, що виконуються послідовно. Тому компоненти сингулярного розкладу наочно демонструють геометричні зміни при відображенні лінійним оператором A множини векторів з векторного простору в себе або в векторний простір іншої розмірності.

Для більш візуального представлення розглянемо сферу S одиничного радіусу в просторі R^n. Лінійне відображення T відображує цю сферу в еліпсоїд простору R^m. Тоді ненульові сингулярні значення діагоналі матриці Σ є довжина півосей цього еліпсоїда. У випадку коли $n = m$ і всі сингулярні величині не рівні нулю, сингулярний розклад лінійного відображення T може бути легко проаналізовано як послідовність трьох операцій:

1. Розглянемо еліпсоїд $T(s)$ та його осі. Розглянемо напрями в R^n, які відображення T переводить в ці осі. Ці напрями є ортогональними. Спочатку застосуємо ізометрію v^*, відобразивши ці напрями на координатні осі R^n.
2. Застосуємо ендоморфізм d, що діагоналізовано вздовж координатних осей та який розширює/стискає ці напрями, використовуючи довжини півосей $T(S)$ як коефіцієнти розтягнення. Тоді добуток $d \otimes v^*$ відображує одиничну сферу на ізометричний еліпсоїд $T(S)$.
3. Застосуємо ізометрію до цього еліпсоїду так, що перенести його в $T(S)$. Легко перевірити, що добуток $u \otimes d \otimes v^*$ співпадає з T.

Графічне зображення перелічених вище операцій можна знайти на рисунку 2.1.

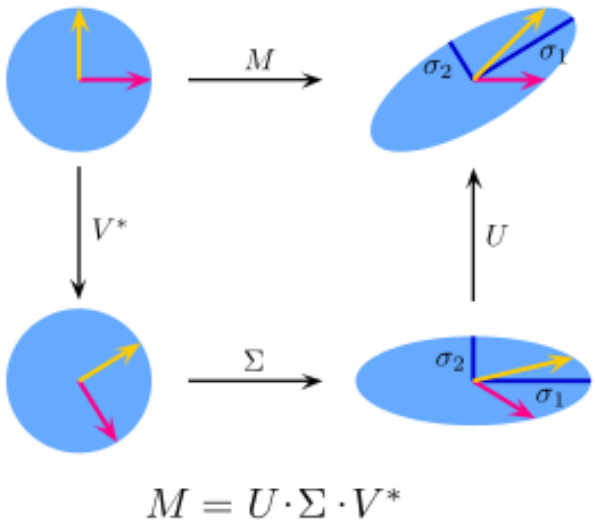

Рисунок 2.1 – Геометричний зміст сингулярного розкладу у двовимірному випадку

2.2.4 Приклади застосування

Псевдообернена матриця

Сингулярний розклад може бути застосовано для знаходження псевдообернених матриць, що застосовуються, наприклад, у методі найменших квадратів.

Якщо

$$M = U\Sigma V^*,$$

то псевдообернена до неї матриця знаходиться по формулі

$$M^+ = V\Sigma^{-1}U^*,$$

де Σ^{-1} – обернена до матриці Σ, що отримується шляхом заміни кожного діагонального елемента σ на обернений до нього: σ^{-1} та транспонуванням.

Наближення матрицею нижчого рангу

В деяких практичних задачах необхідно наближувати задану матрицю M іншою матрицею M_k с попередньо заданим рангом k. Відома наступна теорема, яку іноді називають теоремою Еккарта-Янга.

Якщо вимагається, щоб таке наближення було найкращими в тому сенсі, що норма Фробініуса різниці матриць M та M_k мінімальна, при обмеженні $rank(M_k) = k$, то виявляється, що найкраща така матриця M_k отримується з сингулярного розкладу матриці M по формулі

$$M_k = U\Sigma_k V^*,$$

де Σ_k – матриця Σ, в якій замінили нулями всі діагональні елементи, окрім k найбільших елементів.

Якщо елементи матриці Σ впорядковані не у порядку зростання, то вираз для матриці M_k можна переписати в такій формі:

$$M_k = U_k\Sigma_k V_k^*,$$

де матриці U_k, Σ_k та V_k отримуються з відповідних матриць в сингулярному розкладі матриці M відрізанням до рівно k перших стовпчиків.

Таким чином видно ,що наближуючи матрицю M матрицею меншого рангу, ми виконуємо свого роду стиснення інформації, що міститься в M: матриця M розміру $m * n$ замінюється меншими матрицями розмірів $m * k$ та $k * n$ та діагональною матрицею з k елементами. При цьому стиснення відбувається з втратам – в наближенні зберігається лише найсуттевіша частина матриці М.

В більшості завдяки цій властивості сингулярний розклад має широке застосування: в стисненні даних, обробленні сигналів, чисельних ітераційних методах для роботи з матрицями, методі головних компонент, летентно-семантичному аналізі та інших сферах.

2.2.5 Скорочене представлення

Для матриці M порядку $m * n$ при необхідності наближення матриці рангу r меншого ніж n часто застосовують компактне представлення розкладу

$$M = U_r \Sigma_r V_r^*$$

Обчислюються лише r стовпчиків U та r рядків V^*. Інші стовпці U та рядки V^*не обчислюються. Це дозволяє зекономити велику кількість пам'яті при $r \ll n$.

Наприклад, нехай m – це кількість користувачів, кожен з яких залишив частину оцінок кінофільмам, загальну кількість яких будемо позначати n, тоді матриця (сильно розріджена, оскільки кожен користувач оцінив лише невелику частину фільмів) буде позначатися M та мати досить велику розмірність $[m * n]$.

При бажанні працювати з матрицею меншої розмірності ми повинні обчислити сингулярний розклад

$$M = U\Sigma V^*$$

При цьому матриця Σ як було вказано раніше є діагональною. Після цього, якщо ми хочемо зберігати лише 90% інформації, то ми повинні взяти r таким чином, щоб сума квадратів перших елементів Σ була 90% від загальної суми всіх квадратів діагональних елементів Σ.

Таким чином ми отримуємо U розмірністю $[m * r]$ (взявши r стовпчиків), Σ з розмірністю $[r * r]$ та V^* з $[r * n]$. Після, замість матриці M ми можемо маніпулювати матрицею $M' = U\Sigma$ з меншою розмірністю $[m * r]$, яку часто інтерпретують, як матрицю оцінок користувачів по категоріям фільмів.

2.3 Матрична факторизація

Матрична факторизація – це клас алгоритмів колаборативної фільтрації, що використовується в рекомендаційних системах. Алгоритми матричної факторизації працюють за допомогої декомпозиції матриці взаємодії користувача з об’єктами на дві прямокутні матриці нижчого рангу[9]. Родина таких методів стала широко відомою після змагання Netflix Prize, оскільки довела свою ефективність. Першим про досягнення у цій поділився Саймон Фанк у 2006 році в своєму блозі. Передбачення результатів може бути покращено за допомогою призначення різних вагових коефіцієнтів регуляризації для прихованих факторів популярності об’єктів та активності користувачів.

Декілька найбільш відомих алгоритмів матричної факторизації:

2.3.1 Funk MF

Funk MF – це оригінальний алгоритм запропонований Саймоном Фанком в його записі у блозі. Алогритм факторизував матрицю взаємодії користувачів і об’єктів на добуток двох прямокутних матриць нижчого порядку. Перша мала рядок для кожного користувача, а драгу мала стовчик для кожного об’єкту. Рядок або колонка призначені для окремих користувача або об’єкту посилаються на приховані фактори. Важливо відмітити, що хоча у Funk MF не використовуються методи сингулярного розкладу, він є моделлю машинного навчання з групи SVD-like. Перебачені оцінки можуть бути розраховані як

$$\tilde{R} = HW,$$

де $\tilde{R} \in R^{\text{користувачі} \times \text{об'єкти}}$ це матриця оцінок об’єктів користувачами,

$H \in R^{\text{користувачі} \times \text{приховані фактори}}$ це матриця прихованих користувацьких факторів,

$W \in R^{\text{приховані фактори} \times \text{об'єкти}}$ це матриця прихованих факторів об'єктів.

Передбачення оцінок користувача u для об'єкту i розраховується за формулою

$$\tilde{r}_{ui} = \sum_{f=0}^{nfactors} H_{u,f} W_{f,i}$$

Також можливо налаштовувати потужність моделі змінюючи кількість прихованих факторів[10]. Було показано ,що матрична факторизація з одним прихованим фактором еквівалентна рекомендуванню найпопулярнішого елемента. Збільшення кількості прихованих факторів збільшує персоналізацію, а разом з тим і якість рекомендації, до тих поки факторів не стає занадто багато. Тоді модель стає перенавчаною і якість рекомендацій знижується. Найбільш розповсюдженим способом запобігання перенавчанню є регуляризація об'єктів в о'єктну функцію. Funk MF було розроблено як рішення для проблеми передбачення оцінок, а тому вона використовує зовнішні чисельні значення оцінок взаємодії користувача з об'єктом.

Алгоритм Funk MF мінімізує наступну функцію:

$$\underset{H,W}{\operatorname{argmin}} \left|\left|R - \tilde{R}\right|\right|_F + \alpha\left|\left|H\right|\right| + \beta\left|\left|W\right|\right|,$$

де $||.||_F$ – це норма Фробініуса. Але її можна замініти на будь-яку іншу норму в залежності ід конкретної рекомендаційної задачі.

2.3.2 SVD++

В той час, як Funk MF могла запропонувати дуже високу якість рекомендації, її здатність використовувати лише зовнішні оцінки взаємодії користувачів з об'єктами вважається обмеженням. Сучасні рекомендаційні системи мають використовувати усі доступні оцінки, як зовнішні (наприклад, чисельні оцінки), так і внутрішні (вподобання, покупки, пропускання, занесення в збережені). З цією метою була розроблена модель SVD++[11]. Порівняно з Funk MF, SVD++ приймає до уваги розподілення користувачів та об’єктів.

Прогнозоване значення оцінки користувача u об’єкта i розраховується за формулою:

$$\tilde{r}_{ui} = \mu + b_i + b_u + \sum_{f=0}^{nfactors} H_{u,f} W_{f,i}$$

У той самий час SVD++ має декілька недоліків, головний з яких полягає у том, що цей метод не оснований на моделі. Це означає, що якщо буде додано нового користувача, алгоритм не зможе запропонувати оцінку без обов’язкового перенавчання всієї моделі. Навіть якщо система знайде якісь взаємодії для цього нового користувача, приховані фактори не будуть доступні, а отже ніякі рекомендації не можуть бути розраховані. Це є прикладом проблеми холодного старту, коли рекомендаційна система не може ефективно впоратись с додаванням нового користувача.

Можливим розв’язком цієї проблеми холодного старту може бути модифікація SVD++ з метою створити алгоритм, що базується на моделі, який в свою чергу ефективно впорається з новими об’єктами та користувачами.

Як вже було зазначено, в SVD++ не має прихованих факторів для користувачів, тому виникає необхідність їх відтворити у деякий інший спосіб. Користувацькі приховані фактори представляють собою надання переваги користувачем певним об’єктам, а тому приховані користувацькі фактори можуть бути розраховані як минулі користувацькі взаємодії. Якщо система

здатна віднайти взаємодії для нового користувача, то для нього можливо розрахувати приховані фактори. Важливо також відмітити, що таке рішення не повністю розв'язує проблему холодного старту, оскільки рекомендаційна система досі вимагає декілька взаємодій для нового користувача. Але хоча б тепер немає необхідності перераховувати всю модель кожного разу. Було показано, що таке формулювання майже повністю еквівалентне SLIM моделі, що базується на об'єкт-об'єкт взаємодії.

$$\tilde{r}_{ui} = \mu + b_i + b_u + \sum_{f=0}^{nfactors} \left(\sum_{j=0}^{nitems} r_{uj} W_{j,f} \right) W_{f,i}$$

З таким формулюванням, еквівалентна об'єкт-об'єкт рекомендаційна система буде

$$\tilde{R} = RS = RW^T W$$

2.3.3 Asymmetric SVD

Асиметрична SVD модель була створена з метою об'єднати переваги SVD++ та алгоритмів, що базуються на моделі, а тому вона здатна оброблювати ситуацію нового користувача без необхідності перенавчання всієї моделі. На відміну від SVD алгоритму, що базується на моделі, вона використовує матрицю прихованих факторів Q замість матриці H, що являє собою користувацькі вподобання як функцію від їх оцінок[12].

Передбачення для користувача u відносно об'єкту i може бути розраховане як

$$\tilde{r}_{ui} = \mu + b_i + b_u + \sum_{f=0}^{nfactors} \sum_{j=0}^{nitems} r_{uj} Q_{j,f} W_{f,i}$$

З таким формулюванням, еквівалентна об’єкт-об’єкт рекомендаційна система набуває вигляду

$$\tilde{R} = RS = RQ^T W$$

Оскільки матриці Q та W – це різні матриці, то матриця схожості стає асиметричною. Звідси і назва моделі.

2.3.4 Hybrid MF

Останніми роками було створено багато інших моделей матричної факторизації для більш точного і широкого визначення можливих взаємодій у даних. Гібридні алгоритми матричної факторизації здатні до обробки внутрішніх та зовнішніх взаємодій або колаборативної та контентної інформації[13].

2.3.5 Deep-Learning MF

Також за останні роки було розроблено багато нейронних мереж та моделей глибокого навчання. Деякі з них узагальнювали традиційні алгоритми матричної факторизації через нелінійну нейронну архітектуру[14]. В той самий час як глибинне навчання застосовується у багатьох інших сценаріях: взаємодія з контекстом, взаємодія з послідовностями, соціальне розмічення та багато інших. Тому їх реальна ефективність при використанні для простої колаборативної фільтрації часто ставиться під сумнів. Систематичний аналіз публікацій застосування моделей глибокого навчання та нейронних мереж для проблеми вибору найкращих k об’єктів, що було опубліковано на найбільших конференціях (SIGIR, KDD, WWW, RecSys), показав, що в середньому менше 40% статей можливо відтворити. Загальне вивчення визначає 18 статей, серед

яких лише 7 можуть бути відтворені та 6 з яких можуть бути перевершені значно старішими і простішими методами, якщо їх правильно підлаштувати. Ця стаття також висвітлює ряд потенційних проблем в сьогоднішньому дослідницькому навчанні та закликає до виправлення наукових практик у цій області. Схожі проблеми були знайдені у підході до рекомендаційних систем, що базуються на послідовності дій.

2.4 Гібридна контентно-колаборативна модель

Проектування рекомендаційної системи, що добре показує себе у сценаріях холодного старту досі залишається викликом для спільноти дослідників. Стандартні методи матричної факторизації погано підходять для цієї задачі. Досить важко ефективно передбачити користувацькі та об’єктні приховані фактори, якщо дані є розрідженими.

Підходи, що базуються на змісті, вирішують цю проблему представляючи об’єкти через їхню метаінформацію[5]. Як вже добре відомо, рекомендації можуть бути розраховані навіть для нових об’єктів, що не мають колаборативних даних про них. Нажаль, в моделях, що базуються на змісті, неможливий обмін досвіду між користувачами. Тому, більшість таких моделей мають нижчу ефективність порівняно з моделями матричної факторизації, де колаборативна інформація доступна, та які вимагають великі об’єми даних для опрацювання на одного користувача, що не підходить для сценарію холодного старту[16].

Для компанії Lyst розв’язання цієї проблеми було невідкладним. Lyst – це компанія, що спеціалізується на онлайн продажах модного одягу. Для цього вони підтримують дуже великий онлайн каталог продуктів. На 2015 рік їх сайт агрегував інформацію про 8 мільйонів предметів одягу з усієї мережі інтернет та додавав десятки тисяч предметів кожного дня.

До розробки свого рекомендаційного алгоритму їх спонукали три основні фактори. По-перше, їх система містила дуже великий набір окремих

об’єктів, що робило дані дуже розрідженими. По-друге, специфіка роботи. Найчастіше, найбільш релевантними одиницями товару були ті, що були випущені найпізніше. Це призводило до того, що для генерування ефективних рекомендацій про окремий об’єкт, було дуже мало часу. По-трете, більша частина користувачів сайту була новою. Тобто вони не мали історії для аналізу. Комбінація всіх цих трьох факторів призводила до подвійної проблеми холодного старту і жодна з чистих колаборацій них або змістовних моделей не були здатні показати себе ефективно в таких умовах.

Для розв’язання цієї проблеми, було розроблено контентно-колаборативну модель, яку назвали LightFM. Назва пішла від її схожості з іншими алгоритмами матричної факторизації. В LighFM, як і в моделях колаборативної фільтрації, користувачі та об’єкти представлені як вектори (embedding, ембединг). Більш того, так само як і в моделях, що базуються на змісті, вони однозначно визначаються функціями (у даному конкретному випадку лінійною комбінацією) від векторних представлень змістовних ознак, що описують окремий об’єкт або користувача. Наприклад, якщо фільм «Чарівник з країни Оз» описати наступними ознаками: «музикальне фентезі», «Джуді Гарланд» та «Чарівник країни оз», то векторна репрезентація такого об’єкту буде представлено як векторна сума таких прихованих ознак.

Таким чином, LightFM об’єднує в собі переваги колаборативної фільтрації та моделі, що базується на змісті. Такий підхід має декілька переваг для рекомендаційних систем, що використовуються в «реальному світі». Через те, що LightFM добре працює на щільних та розріджених даних, зникає необхідність розробки та підтримки окремих спеціалізованих моделей для кожного з налаштувань. Також, модель може використовувати метаінформацію про користувача та об’єкт, що має позитивний ефект у сценаріях пов’язаних з холодним стартом.

Структура моделі була обумовлена наступними двома факторами:

1. Модель повинна мати можливість визначити репрезентації користувача та об’єкта з даних взаємодії. Якщо об’єкти описані як

«біла сорочка» та «блакитна сорочка» були вподобані усіма користувачами одночасно, то модель має зрозуміти, що білі сорочки схожі на блакитні сорочки.

2. Модель повинна вміти розрахувати рекомендації для нових об’єктів та користувачів.

Перша умова виконується за допомогою використання підходу прихованих зв’язків. Якщо білі сорочки та сині сорочки обидві подобаються одним і тим самим користувачам, то їх векторні представлення будуть знаходитися поруч. Якщо білі та сині сорочки ніколи не подобаються одним і тим самим користувачам, то їх векторні представлення будуть знаходитися далеко одне від одного.

Такі представлення дозволяють з’явитися передачі даних. Якщо представлення білих та синіх сорочок схожі, то ми можемо із впевненістю рекомендувати сині сорочки новому користувачеві, який встиг лише подивитися на білі сорочки.

Описаний вище підхід відповідає чистому контентному підходу, що використовує техніки пониження розмірності простору (такі як latent semantic indexing, LSI). Схожого результату чистий контентний підхід буде досягнуто шляхом аналізу входжень схожих характеристик, а не користувацьку активність. Наприклад, нехай всі користувачі, що переглядають об’єкти описані як «авіатори», а також переглядають об’єкти описані як «чоботи», але ці дві характеристики ніколи не будуть описувати один і той же об’єкт. В такому випадку LSI вектор для чобіт не буде схожим на вектор для авіаторів, хоча колаборативні дані пропонують інакше.

Друга умова виконується за допомогою репрезентації об’єктів та користувачів за допомогою лінійної комбінації їх змістових векторів. Через те, що змістові характеристики відомі у момент, коли об’єкт або користувач потрапляють у систему, це дозволяє генерувати рекомендації від самого початку. Отриману структуру також легко зрозуміти. Репрезентація білої сорочки – це сума репрезентацій білого та сорочок. Репрезентація користувача

жінки з США буде векторна сума репрезентацій жінки-користувача та користувача з США.

2.4.1. Формальний опис моделі

Для формального опису моделі, нехай U – це множина користувачів, а I – це множина об'єктів. F^U – це множина характеристик користувачів, а F^I- це множина характеристик об'єктів. Кожен користувач взаємодіє з декількома об'єктами, або вподобавши (позитивна взаємодія), або не вподобавши (негативна взаємодія). Множина всіх пар взаємодії користувач-об'єкт $(u, i) \in U \times I$ є об'єднанням позитивної S^+ та негативної S^- взаємодій.

Користувачі та об'єкти повністю описуються їх характеристиками. Кожен користувач u описаний набором характеристик $f_u \subset F^U$. Аналогічно, кожен об'єкт i описаний набором характеристик $f_i \subset F^I$. Характеристики відомі заздалегідь та представляють собою метаінформацію про користувача та об'єкти.

Модель параметризована у термінах d-вимірних користувацьких та об'єктних характеристичних векторів e_f^U та e_f^I для кожної характеристики f. Кожна характеристика також описана скалярним зміщення (b_f^U для користувацької та b_f^I для об'єктної характеристики)

Прихована репрезентація користувача u виражається сумою векторів його характеристик

$$q_u = \sum_{j \in f_u} e_j^U$$

Аналогічно і для об'єкту i:

$$p_i = \sum_{j \in f_i} e_j^I$$

Зміщення для користувача u виражається сумою зміщень його характеристик

$$b_u = \sum_{j \in f_u} b_j^U$$

Аналогічно для об’єкту i

$$b_i = \sum_{j \in f_i} b_j^I$$

Передбачення моделі для користувача u та об’єкту i розраховується як векторний добуток репрезентацій користувача та об’єкта з додаванням користувацького та об’єктного зміщень:

$$\hat{r}_{ui} = f(q_u * p_i + b_u + b_i)$$

В якості функції f можуть підійти найрізноманітніші функції. Для оцінки передбачень добре підійшла би функція тотожного відображення[17]. Але оскільки автори моделі були зацікавлені у передбачені бінарних даних, була обрана функція сігмоіда

$$f(x) = \frac{1}{1 + \exp(-x)}$$

Ціль оптимізації моделі полягає у максимізації ймовірності отримати даних, що обумовлені параметрами. Ймовірність розраховується за формулою:

$$L(e^U, e^I, b^U, b^I) = \prod_{(u,i)\in S^+} \hat{r}_{ui} \times \prod_{(u,i)\in S^-} (1 - \hat{r}_{ui})$$

Навчання відбувається за допомогою стохастичного градієнтного спуску.

2.4.2. Зв'язок з іншими моделями

Схожість LightFM та моделей колаборитвної фільтрації обумовлена наборів характеристик користувача та об’єкту. Якщо набір характеристик складається лише з змінних індикаторів для кожного користувача та об’єкту, то LightFM спрощується до звичайної моделі матричної факторизації. Якщо множини характеристик містять метаінформацію, що використовується більш ніж одним користувачем або об’єктом, то LightFM розширює модель матричної факторизації дозволяючи прихованим факторам впливати на рекомендації. Цей факт є важливим з трьох причин:

У більшості додатків метаінформації буде значно менше, ніж користувачів або об’єктів, наприклад через використання онтології з фіксованою структурою типів/категорій, або через фіксований словників слів, що часто зустрічаються, які використовуються при обробці тексту. Це означає, що буде обраховуватися менша кількість наближених параметрів для обмежених тренувальних даних, що зменшує ризик перенавчання та покращує загальну продуктивність.

Приховані вектори для індикаторних змінних не можуть бути розраховані для нових користувачів або об’єктів (проблема холодного старту). Репрезентація їх як комбінації характеристик метаінформації, що може бути

розрахована с тренувальних даних, дозволяє робити передбачення навіть в умовах холодного старту.

Якщо присутні лише індикаторні характеристики, LightFM повинна показувати себе не гірше звичайних моделей матричної факторизації.

Якщо доступна лише метаінформація, а індикаторних змінних немає, то модель не спрощується до структури моделі, що базується на змісті. LightFM розраховує характеристичні вектори факторизуючи матрицю колаборативної взаємодії. Цей підхід є відмінним від того, що використовуються чистими моделями, що базуються на змісті.

Єдиним особливим випадком, коли LightFM спрощується до чистої контентої моделі, є ситуація, коли користувач описаний індикаторною змінною мав взаємодію лише з одним об’єтом. В такому випадку, користувацький вектор є еквівалентним до вектору у формалізації LSI моделі, і тільки характеристики, що присутні у описах продукту будуть мати схожі характеристичні вектори.

Факт того, що LightFM містить підходи контентної моделі для розріджених даних та матричну факторизацію для щільних, свідчить про те, що модель здатна адаптуватись до будь-яких наборів даних. До речі, емпіричні результати показали, що LightFM працює не гірше ніж чисті моделі у кожному із сценаріїв.

2.4.3. Альтернативні підходи

Існує декілька схожих гібридних моделей, що намагаються розв’язати аналогічну проблему холодного старту поєднуючи контентний та колаборативний підходи.

- Модель Soboroff представляє користувачів у вигляді лінійної комбінації характеристичних векторів об’єктів з якими вони взаємодіяли. Потім вони використовують LSI підхід на отриманій матриці для отримання користувацьких профілів. Репрезентації

нових об’єктів отримуються з проектування їх на простір прихованих характеристик. Перевага такого підходу, відносно контентного полягає у тому, що модель використовує колаборативну інформацію закодовану в матрицю користувач-характеристика. Хоча, він моделює користувацькі вподобання як похідну від характеристик об’єкту, аніж від самого об’єкту. Навідміну від LightFM, де ефект характеристики на передбачення взаємодії завжди розглядається в контексті всіх характеристик, що описують дану пару користувач-об’єкт.

- Модель Saveski використовує факторизацію матриць користувач-об’єкт та об’єкт-характеристика, використовуючи одну і ту ж матрицю прихованих характеристик у обох декомпозиціях. Параметри оптимізуються за допомогою мінімізації зваженої суми функцій втрат для обох матриць. Ваговий гіперпараметр визначає відносну важливість точності в декомпозиції колаборитивної і контентної матриць. Схожий підхід використовувала модель McAuley для моделювання оцінок та характеристик продуктів. LightFM у цьому випадку є простішою оскільки вимагає простішої оптимізації для факторизації єдиної матриці користувач-об’єкт.
- Модель Shmueli предавляє об’єкт як лінійну комбінацію прихованих факторів для рекомендації новинних статей. Аналогічно до LightFM, вона використовує одноцільовий підхід і мінімізує функцію втрат для матриці об’єкт-користувач. Такий підхід виявився ефективним із модифікованими налаштуваннями холодного старту, де метаінформація і дані про користувача, який залишив хоча б один коментар до статті, відомі з самого початку. Більш того, такий підхід не розширюється до моделювання користувацьких характеристик і не довів своєї ефективності у випадку не холодного старту.

- LightFM підходить під традиційне поняття гібридної моделі використовуючи факторизацію матриць користувач-об’єкт, об’єкт-характеристика та користувач-характеристика. З теоретичної точки зору, її можна визначити як частковий випадок факторизуючих машин (Factorisation Machines, FM). FM – це ефективний метод наближення змінної взаємодії в лінійних моделях в умовах розрідженості. Кожна змінна представлена k-вимірним характеристичним вектором. Взаємодії між змінними і та j розраховується як матричний добуток їх прихованих факторів. Такий підхід має перевагу завдяки меншій кількості параметрів, що необхідно наближувати. LighFM звужує структуру взаємодії лише до взаємодії характеристик користувача та об’єкту.

2.4.4. Датасети

Для оцінки ефективності моделі LightFM було використано 2 датасети. Ці датасети представляють собою два крайніх випадки для рекомендаційної системи. Перший датасет (MovieLens) є дуже розрідженим. На ньому мають добре себе проявити моделі матричної факторизації. Другий датасет (CrossValidated) є дуже щільним і на ньому мають добре працювати контентні моделі. Обидва датасети є публічно відкритими.

- Перший експеримент використовує широко відомий датасет MovieLens 10M в поєднанні з Tag Genome набором тегів. Датасет складається з приблизно 10 мільйонів оцінок фільмі, що були виставлені 71567 користувачами для 19681 фільму. Всі фільми мають в описі свій жанр та список тегів з Tag Genome. Кожен тег для фільму супроводжується оцінкою його релевантності (від 0 до 1), що показує наскільки точно тег описує фільм. Для приведення задачі до бінарного вигляду, всі оцінки, що нижче 4 (по шкалі від 1 до 5) будуть оцінюватись як негативні. Ті, що рівні або вищі 4

будуть оцінені як позитивні. Також було відкинуто всі теги, що не проходили мінімальний допустимий поріг релевантності встановлений на значенні 0.8. Фінальний датасет складався з 69878 користувачів, 10681 об’єктів, 9996948 взаємодій та 1030 унікальних тегів.

- Другий датасет складається з питань та відповідей що було запропоновано у CrossValidated – частині великої мережі Stack-Exchange колаборації Q&A сайтів, що фокусуються на статистиці та машинному навчанні. Датасет складається з 5953 користувачів, 44200 питаннь та 1888865 відповідей та коментарів. Кожне питання супроводжується одним або більше з 1032 унікальних тегів (наприклад, «регресія» або «перевірка гіпотези»). Також, інформація про користувачів була доступна у якості секції «Про мене» профілю користувача. Ціль рекомендації співставити користувача з питаннями, на які він зможе відповісти. Якщо користувач відповідає на питання, це розцінюється як позитивна оцінка. Всі питання, на які користувач не відповів, розцінюються як неявна негативна оцінка. Для тесту було побудова 3 негативні тренувальні пари для кожної позитивної пари користувач-питання за допомогою випадкового обирання з усіх питань, що не отримали відповіді[18].

2.4.5. Порівняння з іншими підходами

Для кожного датасету було проведено два експерименти. Перший симулював сценарій теплого старту: 20% всі взаємодій були випадкових чином додані до тестового набору. Другий симулював сценарій холодного старту. Всі взаємодії для 20% об’єктів були видалені з тренувального набору та додані до тестового. Такий підхід наближує умови до тих, де рекомендаційна система має продукувати рекомендації з набору об’єктів, що

мають мало колаборативної інформації. Тобто на основі даних про зміст (у цьому випадку тегів).

Точність моделі вимірюється за допомогою метрики «mean receiver operating characteristics area under the curve» (ROC AUC). Для окремого користувача AUC відповідає ймовірності того, що випадково обраний позитивний об’єкт буде оцінено вище, ніж випадково обраний негативний. Високий результат метрики AUC є еквівалентним низькій «rank-inversion» ймовірності, де рекомендаційна система помилково оцінює негативний об’єкт краще, ніж позитивний. Для оцінки ця метрика розраховується для всіх користувачів і за її значення приймається середнє значення всіх отриманих оцінок.

Метрика AUC розраховується як середнє арифметичне 10 повторюваних експериментів. Кожен експеримент проводиться на тренувальних даних, що складають 80% датасету, та тестових, що складають 20% датасету.

В тесті приймають участь наступні моделі:

1. MF: модель матричної факторизації
2. LSI-LR: контентна модель. Для оцінки спочатку формуються приховані тематик з матриці характеристик об’єкта через приховане семантичне індексування та представлення об’єктів як лінійної комбінації прихованих тематик. Потім тренується окремі моделі логістичної регресії (logistic regression, LR) для кожного окремого користувача у просторі тематик. Навідміну від моделі LightFM, що використовує колаборативні дані для генерування прихованих репрезентацій, LSI-LR базується тільки на факторизації контеної матриці.
3. LSI-UP: гібридна модель, що представляє профілі користувачів (user profiles, UP) як лінійну комбінацію котентних векторів об’єкті, а потім застосовує LSI до результуючої матриці для визначення прихованих користувацьких та об’єктних репрезентацій. Наближення моделі відбувається за допомогою конструювання

матриці користувач-характеристика: кожен рядок відповідає користувачу і являє собою суму контентних характеристик, що репрезентують об’єкти з якими користувач позитивно взаємодіяв. Потім застосовується SVD до нормалізованої матриці для визначення прихованих користувацьких факторів. Об’єктні приховані фактори отримуються через проектування їх на простір прихованих характеристик. Оцінка рекомендації для пари користувач-об’єкт являє собою скалярний добуток їх прихованих репрезентацій.

4. LightFM (tags): модель LightFM, що використовує лише теги
5. LightFM (tags + ids): модель LightFM, що використовує теги та індикаторні характеристики
6. LightFM (tags + about): модель LightFM, що використовує і користувацькі, і об’єктні характеристики.

В обох LightFM (tags) та LightFM (tags + ids) користувачі описуються лише індикаторними характеристиками.

Моделі LightFM тренуються за допомогою градієнтного спуску з початковою точністю рівною 0.05. Прихована розмірність моделей встановлена на 64 для всіх моделей та експериментів. Це налаштування було обрано як компроміс між точністю та обчислювальною вартістю тренування. Регуляризація моделі відбувається через критерій ранньої зупинки: тренування зупиняється тоді, коли ефективність моделі перестає покращуватися.

Як видно з таблиці (рисунок 2.2), LightFM показує себе дуже ефективною. Навіть ефективнішою за спеціалізовані моделі для кожного із сценаріїв.

	CrossValidated		MovieLens	
	Warm	Cold	Warm	Cold
LSI-LR	0.662	0.660	0.686	0.690
LSI-UP	0.636	0.637	0.687	0.681
MF	0.541	0.508	0.762	0.500
LightFM (tags)	0.675	0.675	0.744	0.707
LightFM (tags + ids)	0.682	0.674	**0.763**	**0.716**
LightFM (tags + about)	**0.695**	**0.696**		

Рисунок 2.2 – Результати експерименту

В умовах теплого старту та щільних даних (теплий старт, MovieLens), LightFM показує себе ефективнішою за MF, якщо використовувати теги та індикаторні характеристики. Такий результат заохочує використовувати метаінформацію, оскільки вона може мати досить сильний позитивний вплив на рекомендацію.

Також, LightFM (tags) майже така ж ефективна як MF, не зважаючи на те, що вона використовує лише метаінформацію. LSI-LR та LSI-UP попри те, що вони використовують ту ж саму інформацію, показали себе значно гірше. Цей факт демонструє, що дуже важливо враховувати колаборативну інформацію при розрахунку характеристичних векторів. Також він демонструє, що LightFM здатна аналізувати цю інформацію значно краще, ніж інші гібридні моделі, такі як LSI-UP.

В умовах теплого старту та високої розрідженості даних (теплий старт, CrossValidated), матрична факторизація показала себе дуже погано. Через те, що дані дуже розріджені (матриця користувач-об’єкт для цього датасету розріджена на 99.95% в порівнянні з 99% для MovieLens), MF не здатна визначити якісні приховані репрезентації. Контентні моделі показали себе значно краще.

Різні варіанти LightFM показали себе найкраще. LightFM (tags + about) виявилась найкращою моделлю, що демонструє переваги можливостей LightFM інтегрувати користувацьку метаінформацію в рекомендаційну

модель. Очевидно, через це і покращилась якість передбачень для користувачів з невеликим набором даних у тренувальному датасеті.

Результати для сценарію холодного старту виявились дуже схожими. На датасеті CrossValidated, всі варіанти LightFM проявили себе кращі, ніж інші моделі. LightFM (tags + about) знову показала себе найкращою. Також цікаво було б відмітити, що LightFM (tags. + indicators) показала себе краще ніж LightFM (tags) на датасеті MovieLens не зважаючи на факт того, що у тестовому наборі не можуть бути розраховані характеристичні вектори. Це може бути наслідком того, що використання метаінформації та об'єктних характеристик дозволяє моделі краще розраховувати характеристичні вектори для користувачів та об'єктів. Недивно, що MF модель воказала себе не краще, ніж випадковий вибір для сценарію холодного старту.

В усіх сценаріях LSI-UP модель проявляє себе не краще, ніж LSI-LR модель, що не зважаючи на факт того, що вона також використовує колаборативні дані. На датасеті CrossValidated вона показує себе значно гірше. Це може бути наслідком того, що приховані репрезентації розраховуються на меншій кількості даних, ніж в LSI-LR, оскільки користувачів у датасеті менше, ніж об'єктів, а тому менше рядків у матриці користувач-характеристика, ніж у матриці об'єкт-характеристика.

Результати підтверджують факт того, що LightFM виявилась ефективнішою за спеціалізовані моделі MF та LSI-LR у спеціальних сценаріях, показуючи себе ефективнішою за LSI-LR у випадку розріджених даних та ефективнішою за MF у випадку щільних даних. Це означає, що LightFM може використовуватись не тільки у якомусь одному сценарії. Вона також може використовуватись у випадках, коли щільність даних змінюється з часом.

Висновки до розділу 2

В рамках даного розділу було детальніше розглянуто колаборативний підхід до генерації рекомендацій, а також способи розв’язання основних технічних проблем, що з ним пов’язані.

Було прийнято рішення в рамках дипломної роботи використовувати покращену версію колаборативного підходу, а саме гібридну-контентно колаборативну модель LightFM. Гібридизація даної моделі дозволяє розв’язати проблему холодного старту та покращити якість рекомендацій на нещільних даних. Ефективність даного підходу було підтверджено розробниками програмного модулю для LightFM на датасетах MovieLens та CrossValidation.

Оскільки дані датасети є невеликими (20 мільйонів та 500 тисяч записів), то розробка сервісу та відповідні допоміжні розрахунки можуть відбуватися на персональному комп’ютері.

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1. Аналіз моделі даних

Відповідно до поставленої задачі, було знайдено датасет, предметна область якого дозволяє розробку і створення рекомендаційної системи. Відповідний датасет було знайдено на інтернет ресурсі Internet Archive. Даний датасет включає в себе ананонімізовану вигрузку питань та віповідей із категорії CrossValidated на великому ресурсі з питань та відповідей StackExchange. Конкретно ця категорія спеціалізується на питаннях зі статистики та машинного навчання.

Даний датасет включає в себе:

- 5953 користувача
- 44200 питань
- 188865 відповідей та коментарів

Кожне питання супроводжується хоча б одним з 1032 унікальних ідентифікаторів, що характеризують тематику питання. Наприклад, «Регресія» або «Тестування гіпотез».

Більшість користувачів у датасеті також мають додаткову інформацію, що вказується у категорії профілю «Про себе».

Метою рекомендаційної системи для цього датасету буде співставити користувача і питання, що можуть його зацікавити.

Вхідні дані в датасеті представлені наступними структурами:

- Матриця користувач-питання. Дана матриця репрезентує взаємодію користувачів та питань. Якщо користувач із унікальним номером 1 відповів на питання з унікальним номером 3, то відповідний елемент матриці у рядку 1 та стовпчику 3 буде мати значення 1. У іншому випадку цей елемент буде рівним 0.
- Матриця питання-характеристика. Кожен рядок матриці представляє собою відповідний характеристичний вектор для

кожного питання у датасеті. Характеристичний вектор представляє собою масив у якому кожному з 1032 унікальних ідентифікаторів питань ставиться у відповідність чисельне значення. Тобто якщо питання має категорію «Регресія», що має унікальний ідентифікатор 1, то відповідне значення вектору із індексом 1 буде встановлено як 1.

- Матриця користувач-характеристика. Кожен рядок матриці представляє собою векторне представлення категорії профілю користувача «Про себе». Векторне представлення формується за допомогою технології word2vec. В даній роботи дана матриця не буде використовуватись, оскільки технологія word2vec вимагає окремих досліджень та не всі користувацькі профілі мають заповнені категорії «Про себе».

3.2. Проектування сервісу

Кінцевий програмний продукт складатиметься з наступних структурних елементів:

- Програма, що реалізовує API інтерфейс додатку
- Програма, що розраховує рекомендації для користувачів та тимчасово зберігає їх у базу даних
- База даних для зберігання інформації про користувачів, питання та їх взаємодії
- База даних для тимчасового зберігання згенерованих рекомендацій

Основна програма з API інтерфейсом впроваджує методи взаємодії користувача з системою за допомогою протоколу HTTP. Користувачу будуть представлені наступні ресурси (таблиця 3.1):

Таблиця 3.1 – Опис HTTP ресурсів та методів взаємодії з ними

Метод	Ресурс	Опис
GET	/items/{item_id}	Отримати інформацію про об’єкт за його унікальним ідентифікатором
PUT	/items/{item_id}	Зберегти інформацію про об’єкт з унікальним ідентифікатором
DELETE	/items/{item_id}	Видалити інформацію про об’єкт за його унікальним ідентифікатором
GET	/users/{user_id}	Отримати інформацію про користувача за його унікальним ідентифікатором
PUT	/users/{user_id}	Зберегти інформацію про користувача з унікальним ідентифікатором
DELETE	/users/{user_id}	Видалити інформацію про користувача за його унікальним ідентифікатором
GET	/users/{user_id}/history	Отримати історію взаємодій користувача з об’єктами за його унікальним ідентифікатором
POST	/users/{user_id}/interact/{item_id}	Зберегти інформацію про взаємодію користувача з

		об’єктом за їх унікальними ідентифікаторами
GET	/users/{user_id}/recommendations	Отримати рекомендацію для користувача за його унікальним ідентифікатором

У разі виникнення помилок системи, до яких призвели дії користувача або несправності у самій системі, користувачу будуть відправлені відповідні статуси HTTP відповідей з гідно до загальноприйнятих правил побудови RESTful API.

Для постійного зберігання даних обрано реляційну модель з наступною структурою даних, що приведена до третьої нормальної форми (таблиця 3.2)

Таблиця 3.2 – Опис таблиць реляційної БД

Назва таблиці	Опис
users	Зберігає дані користувачів (унікальний ідентифікатор та іншу додаткову інформацію)
items	Зберігає дані об’єктів (унікальний ідентифікатор та іншу додаткову інформацію)
interactions	Зберігає дані взаємодії користувачів та об’єктів
user_features	Зберігає ознаки користувачів та їх векторне представлення
item_features	Зберігає ознаки об’єктів та їх векторне представлення
user_description	Зберігає опис користувача за допомогою ознак
item_description	Зберігає опис об’єкту за допомогою ознак

Для тимчасового зберігання даних обрано модель ключ-значення, де в якості ключа виступатиме унікальний ідентифікатор користувача, а у якості значення – масив об’єктів, що йому рекомендовано.

Програма, що розраховує попередні рекомендації для користувачів має впроваджувати наступний алгоритм роботи:

1. Завантаження з БД інформації про актуальні об’єкти рекомендацій та їх ознаки
2. Завантаження з БД інформації про активних користувачів та їх ознаки
3. Завантаження з БД інформації про взаємодії користувачів та об’єктів
4. Розбиття інформації взаємодії на тренувальну та тестову
5. Формування матриці взаємодії об’єкт-користувач та матриці характеристик об’єкт-ознака
6. Ініціалізація та тренування моделі на тренувальних даних
7. Оцінка параметрів моделі за допомогою метрики ROC AUC на тренувальних та тестових даних
8. Генерування рекомендацій з актуальних об’єктів для кожного активного користувача.
9. Збереження отриманих рекомендацій у БД тимчасового збереження.

3.3. Розробка сервісу

В якості БД для постійного зберігання даних додатку було обрано реляційну БД PostgreSQL. На вибір впливали наступні переваги даної БД:

- Потужна та продуктивна БД з ефективними інструментами масштабування на випадок росту об’ємів інформації, що зберігається
- Підтримує багато різних вбудованих типів даних, що значно полегшує розширення моделі даних на випадок, якщо знадобиться зберігати будь-яку метаінформацію про користувача, наприклад.

- Активна російськомовна спільнота, що значно спрощує процес дослідження принципів роботи БД та розв’язання будь-якого роду питань

В якості БД для тимчасового зберігання рекомендацій для користувачів було обрано БД ключ-значення Redis. Вибір був керований наступними факторами:

- Потужна і ефективна БД з можливістю кешування даних в оперативній пам’яті серверу, що пришвидшує пошук необхідних даних.
- Широкий вибір вбудованих типів, що дозволяє зберігати дані ширшого спектру складності на випадок розвитку сервісу в сторону необхідності кешування додаткової інформації

Програмна реалізація API інтерфейсу створена за допомогою мови програмування Python з використанням принципів асинхронного програмування, що значно збільшує продуктивність додатку порівняно з синхронними аналогами.

Програмна реалізація для генерування рекомендація також створена за допомогою мови програмування Python. В якості моделі для генерування рекомендацій використовується гібридна контентно-колаборативна модель LightFM.

Приклади HTTP запитів для взаємодії з сервісом (рисунки 3.1-3.9)

```
PUT items/1
{
    "feature_ids": [3, 4]
}
```

Рисунок 3.1 – Збереження об’єкту

```
DELETE items/1
```

Рисунок 3.2 – Видалення об’єкту

```
GET items/1

{
    "data": {
        "id": 1,
        "embedding": [...],
        "features": [
            {
                "id": 3,
                "description": "distributions",
                "embedding": [...]
            },
            {
                "id": 4,
                "description": "normality",
                "embedding": [...]
            }
        ]
    }
}
```

Рисунок 3.3 – Отримання даних про об’єкт

```
PUT users/1
{
    "feature_ids": [3, 4]
}
```

Рисунок 3.4 – Збереження користувача

```
DELETE users/1
```

Рисунок 3.5 – Видалення користувача

```
GET users/1

{
    "data": {
        "id": 1,
        "embedding": null,
        "features": []
    }
}
```

Рисунок 3.6 – Отримання даних про користувача

```
GET /users/1/history?limit=10

{
    "data": [
        {
            "id": 62688
        },
        {
            "id": 63529
        },
        ...
    ]
}
```

Рисунок 3.7 – Отримання історії взаємодій користувача

```
POST /users/1/interact/1
```

Рисунок 3.8 – Збереження події взаємодії користувача з об’єктом

```
GET /users/1/recommendations?limit=10

{
    "data": [
        {
            "id": 61825
        },
        {
            "id": 39719
        },
        ...
    ]
}
```

Рисунок 3.9 – Отримання рекомендації для користувача

Для взаємодії з сервісом можна використовувати будь-який модуль HTTP запитів. Ресурси з методом GET можна викликати навіть з веб браузера.

Базовий алгоритм взаємодії з сервісом описано схемою (рисунки 3.10 – 3.11)

Рисунок 3.10 – Кроки 1-3 алгоритму взаємодії з сервісом

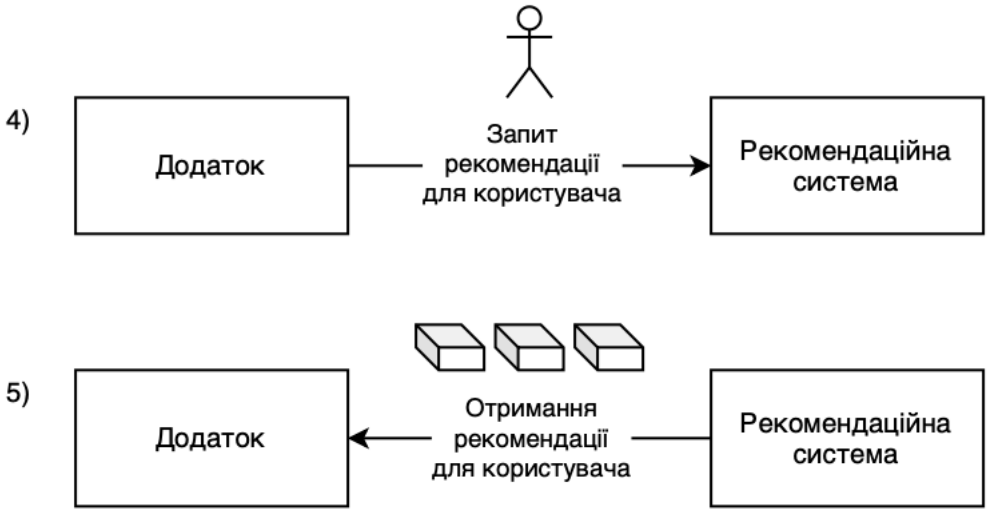

Рисунок 3.11 – Кроки 4-5 алгоритму взаємодії з сервісом

Висновки до розділу 3

Під час виконання дипломної роботи був розроблений сервіс рекомендаційної системи з використанням гібридної контентно-колаборативної моделі.

Отриманий сервіс задовольняє необхідним вимогам, а саме:

- Тренування моделі за прийнятний час (тренування моделі відбувається менше 1 хвилини)
- Генерування точних рекомендацій (ROC AUC складає 0.96)
- Максимальна можлива кількість паралельних запитів не менше 30 в секунду
- Час відповіді сервісу складає менше 300 мс

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1. Постановка задачі проектування

Розробити програмний продукт рекомендаційної системи, що задовольнятиме наступним технічним вимогам:

- Взаємодія з сервісом має відбуватися за допомогою протоколу HTTP
- Інтерфейс взаємодії із сервісом має бути реалізований за стандартами RESTful API
- Сервіс має забезпечити прийнятну швидкість обробки запиту
- Сервіс має забезпечити стабільну роботу при високих навантаженнях
- Сервіс має забезпечити інтуїтивний інтерфейс взаємодії для розробника
- Сервіс має потребувати мінімальні витрати на впровадження в робоче середовище

4.2. Обґрунтування функцій програмного продукту

В якості головної фукнції F_0 обрано розробку програмного продукту, що здатен приймати дані про нових користувачів та об’єкти з їх характеристиками та здатен рекомендувати об’єкти для користувача в залежності від його історії взаємодії з іншими об’єктами. Виходячи з цієї мети, можна визначити наступні головні фукнції програмного продукту:

- F_1 – платформа для якої відбуватиметься розробка
  a) UNIX
  b) Windows
- F_2 – допоміжна бази даних, що зберігатиме інформацію

a) PostgreSQL
b) MySQL
c) SQLite
d) MongoDB

- F_3 – мова програмування для розробки
  a) Python
  b) JavaScript
- F_4 – модуль для побудови HTTP інтерфейсу
  a) Built-in
  b) Flask
  c) Aiohttp

Виходячи із запропонованих варіантів побудуємо наступну морфологічну карту (рис 4.1)

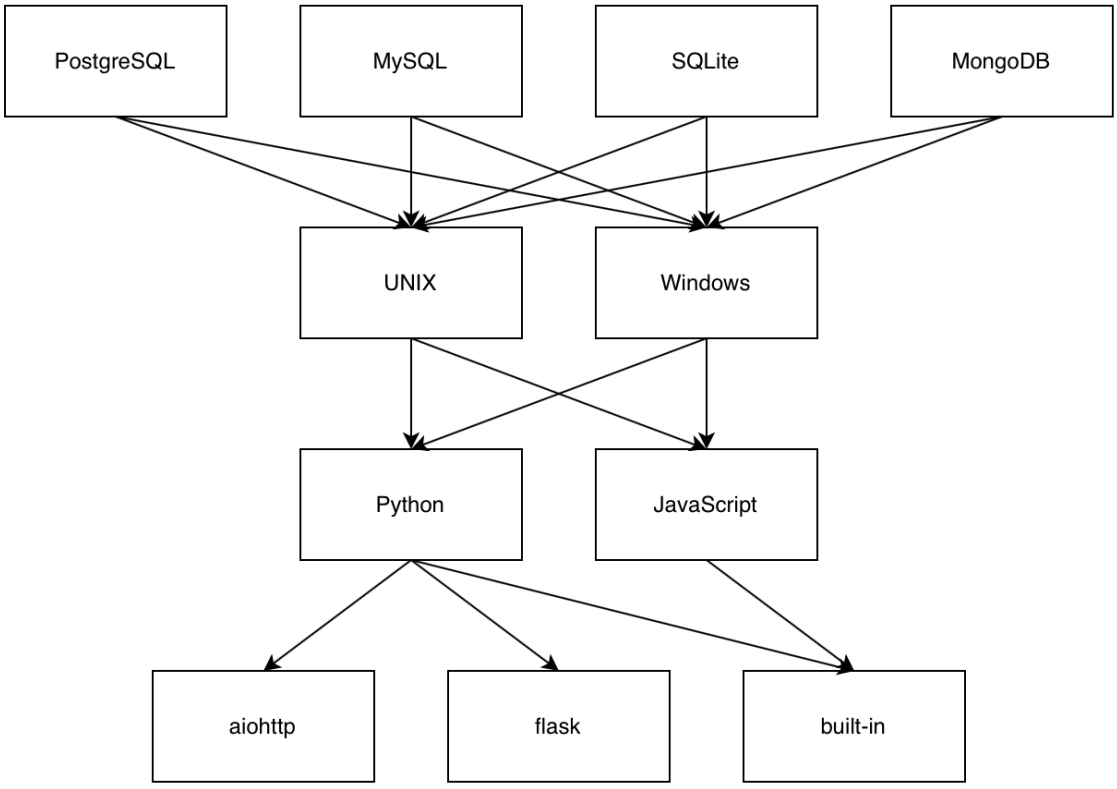

Рис 4.1 – Морфологічна карта

Спираючись на побудовану морфологічну карту можна побудувати наступну позитивно-негативну таблицю (таблиця 4.1)

Таблиця 4.1 – Позитивно-негативна матриця

Основна функція	Варіант реалізації	Переваги	Недоліки
F_1	А	Простота взаємодії і розробки	Невисока популярність серед побутових користувачів
	Б	Популярна ОС	Складність розробки та несумісність деяких програмних модулів
F_2	А	Популярна потужна реляційна база з великою російськомовною спільнотою	Реляційна модель накладає обмеження на динамічність моделі даних
	Б	Найпопулярніша реляційна база з простим механізмом взаємодії	Реляційна модель та обмеженість вбудованих типів даних
	В	Проста та «легка» база	Реляційна модель, обмеженість функціоналу та неможливість ефективного масштабування
	Г	Потужна документо-орієнтована база з ефективними механізмами масштабування	Нижча ефективність на великих масштабах та обмеженість у складності структурної взаємодії даних

F_3	А	Проста і популярна мова програмування	Невисока ефективність на великих навантаженнях
	Б	Проста і популярна мова програмування	Нестача експертизи у розробника
F_4	А	Вбудовані методи, що не вимагають додаткових програмних модулів	Складність розробки та невисока ефективність кінцевого продукту
	Б	Популярний і простий у розробці модуль з відкритою кодовою базою	Невисока швидкодія та складність масштабування
	В	Популярний і ефективний програмний модуль із відкритою кодовою базою	Складність розробки та відсутність підтримки деяких інструментів через відносну новизну модуля

Виходячи з даних про основні функції, що повинен реалізувати програмний продукт, та вимог до нього, можемо визначити основні параметри виробу, що будуть використовуватись для розрахунку коефіцієнта технічного рівня (таблиця 4.2)

Таблиця 4.2. – Основні параметри програмного продукту

Найменування параметру	Позначення параметру	Одиниці виміру	Значення параметру		
			Гірше	Середнє	Краще
Швидкодія мови програмування	X_1	нс/оп	200	50	1

Час обрахунку моделі	X_2	с	600	120	20
Максимальна кількість паралельних запитів	X_3	од.	1	10	50
Час обробки запиту	X_4	мс	1000	500	300
Потенційний об'єм програмного коду	X_5	рядок коду	5000	3000	1000

На основі таблиці 4.2 побудуємо графічні характеристики параметрів (рисунки 4.2 – 4.6)

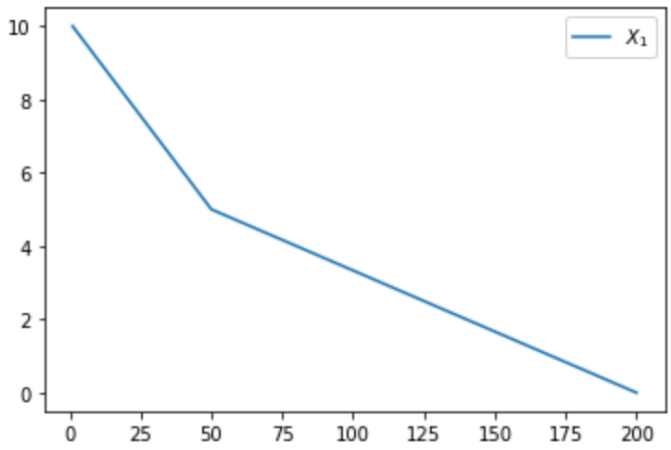

Рисунок 4.2 - X_1, швидкодія мови програмування

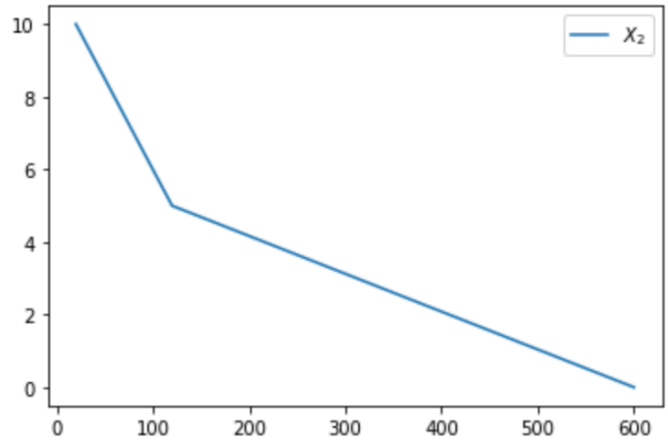

Рисунок 4.3 - X_2, час обрахунку моделі

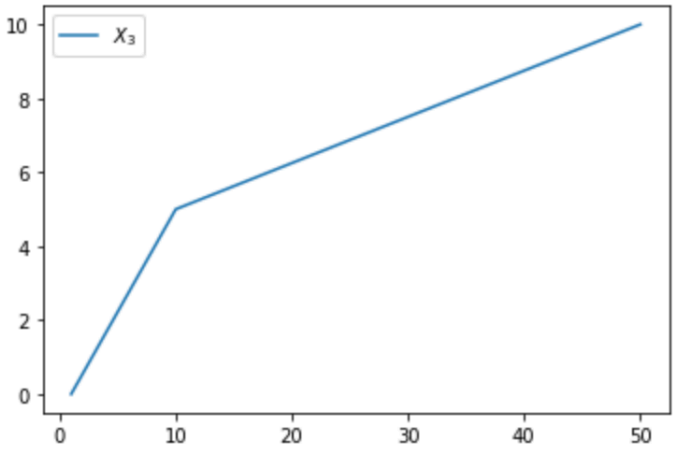

Рисунок 4.4 - X_3, максимальна кількість паралельних запитів

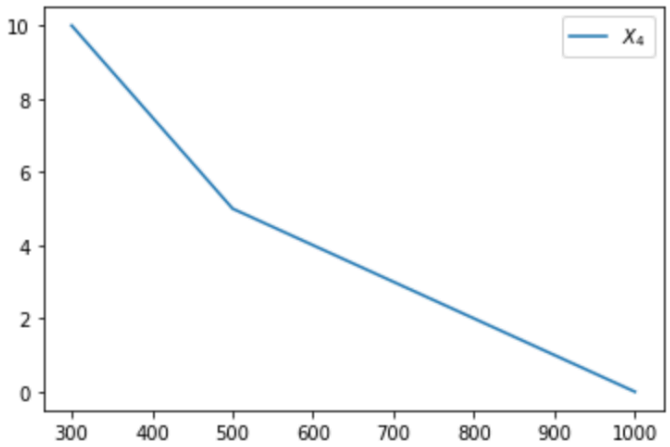

Рисунок 4.5 - X_4, Час обробки запиту

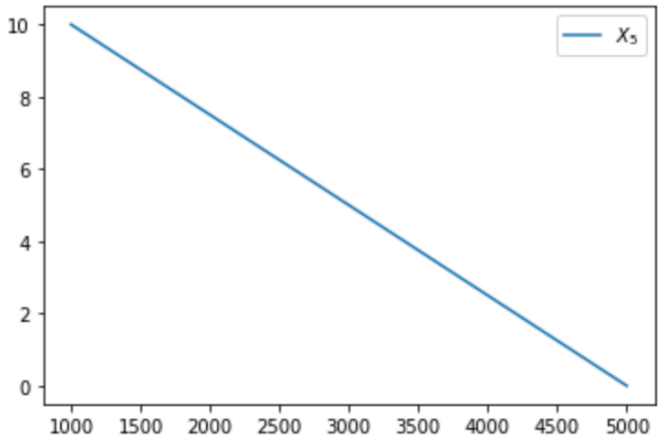

Рисунок 4.6 - X_5, Потенційний об’єм програмного коду

Вагомість параметрів оцінюється за допомогою методу попарного порівняння. Результати для рангів у проміжку від 1 до 5 можна знайти у таблицях 4.3 та 4.4. За найбільший ранг приймається 5.

Таблиця 4.3 – Результати ранжування параметрів

Параметр	Ранг параметру по оцінці експерта							Сума рангів, R_i	Відхилення, Δ_i	Квадрат відхилення, Δ_i^2
	1	2	3	4	5	6	7			
X_1	5	5	4	5	5	5	4	33	12	144
X_2	3	4	5	4	4	4	5	29	8	64
X_3	4	1	3	2	3	3	3	19	-2	4
X_4	2	2	2	3	2	1	1	13	-8	64
X_5	1	3	1	1	1	2	2	11	-10	100
Разом	15	15	15	15	15	15	15	105	0	376

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X_1 та X_2	>	>	<	>	>	>	<	>	1.5
X_1 та X_2	>	>	>	>	>	>	>	>	1.5

X_1 та X_4	>	>	>	>	>	>	>	>	1.5
X_1 та X_5	>	>	>	>	>	>	>	>	1.5
X_2 та X_3	<	>	>	>	>	>	>	>	1.5
X_2 та X_4	>	>	>	>	>	>	>	>	1.5
X_2 та X_5	>	>	>	>	>	>	>	>	1.5
X_3 та X_4	>	<	>	<	>	>	>	>	1.5
X_3 та X_5	>	<	>	>	>	>	>	>	1.5
X_4 та X_5	>	<	>	>	>	<	<	>	1.5

Визначимо коефіцієнт узгодженості

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 376}{7^2(5^3 - 5)} = 0.77$$

Результати можна вважати достовірними, оскільки коефіцієнт узгодженості перевищує нормативний, що дорівнює 0.67

Розрахунок вагомості параметрів наведено в таблиці 4.5.

Таблиця 4.5 – Розрахунок вагомості параметрів

ПараметриX_i	Параметри X_j					Перший крок		Другий крок		Третій крок	
	X_1	X_2	X_3	X_4	X_5	b_i	$K_{вi}$	b_i^1	$K_{вi}^1$	b_i^2	$K_{вi}^2$
X_1	1	1.5	1.5	1.5	1.5	7	0.28	34	0.30	155.5	0.30
X_2	0.5	1	1.5	1.5	1.5	6	0.24	27.5	0.24	124.75	0.24
X_3	0.5	0.5	1	1.5	1.5	5	0.2	22	0.19	100	0.19
X_4	0.5	0.5	0.5	1	1.5	4	0.16	17.5	0.15	80.25	0.15
X_5	0.5	0.5	0.5	0.5	1	3	0.12	14	0.12	64.5	0.12
Загалом						25	1	115	1	525	1

Визначимо рівень якості кожного варіанту виконання основних функцій.

Абсолютні значення параметрів X_2 (потужність процесора та об'єм оперативної пам'яті), X_3 (кількість паралельних запитів) та X_4 (час обробки запиту) відповідають технічним вимогам умов функціонування даного програмного продукту.

Абсолютне значення параметру X_1 (швидкодія мови програмування) обрано не найгіршим (не максимальним). Таким чином це значення має відповідати варіантам а) 50 нс/оп або б) 1 нс/оп.

Розрахуємо показник рівня якості варіантів реалізації основних функцій програмного продукту (таблиця 4.6)

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних пунктів програмного продукту

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
$F_1(X_1)$	А	80	3.8	0.3	1.14
	Б	100	3.4		1.02
$F_2(X_2)$	Г	200	4	0.24	0.96
$F_4(X_3)$	А	30	7	0.19	1.33
$F_4(X_4)$	В	300	10	0.15	1.5
$F_4(X_5)$	В	2500	6.25	0.12	0.75

Обрахуємо коефіцієнти якості кожного з варіантів розробки

$$K_{я1} = 1.14 + 0.96 + 1.33 + 1.5 + 0.75 = 5.68$$
$$K_{я2} = 1.02 + 0.96 + 1.33 + 1.5 + 0.75 = 5.56$$

Оскільки другий варіант має найвищий коефіцієнт якості, то він є найкращим.

4.3. Економічний аналіз варіантів розробки

Для визначення вартості розробки програмного продукту спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Обробка HTTP запитів
2. Перенавчання моделі

Кожен з варіантів має одне додаткове завдання, яке є реалізацією розгалужених варіантів розробки незалежного модуля

3.1 Кешування розрахунків у незалежній БД

3.2 Кешування розрахунків у пам’яті додатку

Варіант 1 складається з основних та додаткового завдання 3.1 Варіант 2 складається з основних та додаткового завдання 3.2

Завдання 1 та завдання 3.1 і 3.2 за ступенем новизни відноситься до групи В. Завдання 2 – до групи Б. За складністю алгоритми, що використовуються в завданні 1 та завданнях 3.1 і 3.2 належать до групи 2, а в завданні 2 – до групи 1.

Для реалізації завдання 1 використовує довідкову інформацію, а завдання 2 – інформацію з даних.

Спираючись на норми розрахункового часу визначимо трудомісткість. Для першого завдання трудомісткість складає $T_p = 43$ людино-днів. Поправочний коефіцієнт складає $К_п = 1.35$. Оскільки для виконання даного завдання використовуються стандартні модулі, це буде враховано за допомогою коефіцієнта $K_{ст} = 0.8$. Поправочний коефіцієнт, що враховує програмування на високорівневій мові програмування та розробку програмного забезпечення для всіх завдань складатиме 1. Повна трудомісткість першого завдання складає (складність – 2, новизна - В)

$$T_1 = 43 * 1.35 * 0.8 = 46.44 \text{ людино} - \text{днів}$$

Аналогічно для завдання 2 (складність – 1, новизна - Б)

$$T_2 = 27 * 1.08 * 0.9 = 26.24$$

Аналогічно для завдання 3.1 (складність - 2, новизна - В):

$$T_3 = 43 * 1.35 * 0.6 = 34.83$$

Аналогічно для завдання 3.2 (складність - 2, новизна - В)

$$T_4 = 43 * 1.35 * 0.7 = 40.64$$

Повна трудомісткість:

1. $T_1 = 46.44 + 26.24 + 34.83 = 107.51$
2. $T_2 = 46.44 + 26.24 + 40.64 = 113.32$

Найбільшу трудомісткість має варіант 2.

В розробці приймають участь програміст з окладом 12000 грн., аналітик даних з окладом 10000 грн та тестувальник з окладом 8000 грн. Робочий день складає 8 годин. В тижні 5 робочих днів. Розрахуємо середню заробітну плату:

$$\mathrm{C_ч} = \frac{12000 + 10000 + 8000}{22 * 8 * 3} = 56.82$$

В такому випадку заробітна плата працівників для кожного варіанту розробки сладатиме:

1. $C_{\mathrm{зп}} = 56.82 * 8 * 107.51 = 48869.75$
2. $\mathrm{C_{зп}} = 56.82 * 8 * 113.32 = 51510.74$

Відрахування на соціальне страхування складатиме 22%

1. $C_{від} = 48869.75 * 0.22 = 10751.34$
2. $C_{від} = 526919.08 * 0.22 = 11332.36$

Розрахуємо витрати на оплату однієї машино-години. Враховуючи той факт, що вона обслуговує одного спеціаліста з окладом 12000 грн., одного з окладом 10000 грн та одного з окладом 8000 грн з коефіцієнтом зайнятості 0.4, то отримуємо для трьох машин наступне

$$C_r = 12 * 12000 * 0.4 + 12 * 10000 * 0.4 + 12 * 8000 * 0.4 = 144000 \text{ грн}$$

Враховуючи додаткову заробітну плату, що рівна 20%

$$C_{зп} = 144000 * 1.2 = 172800 \text{ грн}$$

Відрахування на соціальне страхування 22%

$$C_{від} = 172800 * 0.22 = 38016 \text{ грн}$$

Розрахуємо амортизаційні відрахунки при амортизації 25% та вартості ЕОМ – 40000 грн:

$$C_A = 1.15 * 0.25 * 40000 = 11500 \text{ грн}$$

Витрати на ремонт і профілактику:

$$C_p = 1.15 * 40000 * 0.05 = 2300 \text{ грн}$$

Ефективний годинний фонд часу ПК за рік:

$$T_{еф} = (365 - 104 - 8 - 16) * 8 * 0.6 = 1137.6 \text{ год}$$

Витрати на електроенергію:

$$C_{ел} = 1137.6 * 0.156 * 0.3 * 1.75 = 308.17 \text{ грн}$$

Накладні витрати:

$$C_{н} = 40000 * 0.67 = 26800 \text{ грн}$$

Експлуатаційні витрати:

$$C_{екс} = 172800 + 38016 + 11500 + 2300 + 308.17 + 26800 = 251725.17$$

Собівартість однієї машино-години ЕОМ:

$$C_{м-г} = \frac{251725.17}{1137.6} = 221.28 \text{ грн/год}$$

Враховуючи, що вся робота ведеться на ЕОМ, то витрати на оплату машинного часу складатимуть:

1. $C_{м} = 221.28 * 8 * 107.51 = 190318.50$
2. $C_{м} = 221.28 * 8 * 113.32 = 200603.60$

Накладні витрати відповідно складатимуть

1. $C_{н} = 190318.50 * 0.67 = 127513.40$

2. $C_н = 200603.60 * 0.67 = 134404.41$

Повна вартість розробки:

1. $C_{пп} = 48869.75 + 10751.34 + 190318.50 + 127513.40 = 377453$
2. $C_{пп} = 51510.74 + 11332.36 + 200603.60 + 134404.41 = 397851.1$

4.4. Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня

$$K_{тер} = \frac{5.68}{377453} = 1.5 * 10^{-5}$$

$$K_{тер} = \frac{5.56}{397851.1} = 1.3 * 10^{-5}$$

Висновки до розділу 4

Отже, враховуючи проведені дослідження, ми доходимо висновку, що перший варіант реалізації оптимальніший з точки зору якісно-економічного дослідження. Його коефіцієнт техніко-економічного рівня складає $1.5 * 10^{-6}$.

Розробка цього варіанту передбачає

1. Обробку HTTP запитів
2. Перерахунок моделі

Та серед завдань між якими був вибір даний варіант реалізує

3. Кешування розрахунків у незалежній БД

ВИСНОВКИ

В першому розділі було розглянуто поняття рекомендаційної системи та основні алгоритмічні підходи до її побудови. Також було розглянуто основні проблеми рекомендаційних систем, та можливі способи їх розв’язання.

У другому розділі було детальніше розглянуто метод колаборативної фільтрації та його модифікацію – гібридну контентно-колаборативну модель. Для останньої було проаналізовано її характеристики та проведено порівняльний аналіз з існуючими альтернативними підходами. Також було представлено аналіз існуючих методів до розв’язання проблем розрідженості даних та проблеми холодного старту.

У наступному розділі представлено власне розробку програмного модулю. По-перше, було знайдено датасет, що відповідав вимогам та поставленій задачі. Далі було обрано та протестовано програмну реалізацію гібридної контентно-колаборативної моделі. Ця реалізація виявила себе ефективною на тестових даних та мала досить ефективні показники процесу навчання. Також вона була перевірена на можливість інтеграції в повноцінний сервіс. Далі було проведено проектування програмного інтерфейсу самого сервісу та логіки роботи з даними. Було побудовано реляційну модель даних, що забезпечує ефективну роботу сервісу. Наступним кроком було проведено аналіз існуючих альтернатив для зберігання постійних та тимчасових даних. В якості БД для постійного зберігання інформації було обрано реляційну БД PostgreSQL. Для тимчасового зберігання даних було обрано базу типу ключ-значення Redis. Програмна реалізація сервісу була створена за допомогою мови програмування Python з використанням асинхронного підходу програмування. Використання сервісу відбувається за допомогою протоколу HTTP.

В останньому розділі було проведено функціонально-вартісний аналіз, у ході якого було виявлено, що оптимальними технологіями для побудови

сервісу рекомендаційної системи стануть мови програмування Python та асинхронний підхід до програмування.

Для подальшого розвитку проекту можна виділити три основних напрямки. Першим є покращення існуючої гібридної контентно-колаборативної моделі за допомогою експериментів з характеристичними ознаками користувачів та об’єктів. Другим є додавання нових рекомендаційних алгоритмів у сервіс та побудова гібридної рекомендаційної системи. Останнім напрямком роботи є розширення фунціоналу продукту з метою побудови гнучкої системи з широким спектром налаштувань.

СПИСОК ПОСИЛАННЬ

1. F. Ricci, L. Rokach, B. Shapira. Introduction to recommender systems, 2011. P. 1-35.

2. H. Chen, A. Ororbia, C. Giles. A keyphrase based expert recommender for digital libraries, 2015. P. 1-9.

3. J. Breese; D. Heckerman, C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering, 1998. P. 3-11.

4. D. Wang, Y. Liang, D. Xu. A content-based recommender system for computer science publications, 2018. P. 1-9.

5. G. Adomavicius, A. Tuzhilin. Toward the Next Generation of Recommender Systems, 2005. P. 734–749.

6. Netflix Prize // Wikipedia. URL: https://en.wikipedia.org/wiki/Netflix_Prize (дата звернення 19.04.2020).

7. С. Писсанецки. Технология разреженных матриц, 1988. С. 20-40

8. Y. Koren, R. Bell, C. Volinsky. Matrix Factorization Techniques for Recommender Systems, 2009. P. 30–37.

9. D. Jannach, L. Lerche, F. Gedikli. What Recommenders Recommend – An Analysis of Accuracy, Popularity, and Sales Diversity Effects, 2013. P. 25–37.

10. J. Cao, H. Hu, T. Luo. Distributed Design and Implementation of SVD++ Algorithm for E-commerce Personalized Recommender System, 2015. P. 30–44.

11. L. Pu, B. Faltings. Understanding and improving relational matrix factorization in recommender systems, 2013. P. 41–48.

12. C. Zhao, S. Sun, L. Han. Hybrid Matrix Factorization for Recommender Systems in Social Networks, 2016. P. 559–569.

13. X. He, L. Liao, H. Zhang. Neural Collaborative Filtering, 2017. P. 173–182.

14. P. Lops, M. Gemmis, G. Semeraro. Content-based recommender systems, 2011. P. 73–105.

15. S. Rendle, C. Freudenthaler, Z. Gantner. Bayesian personalized ranking from implicit feedback, 2009. P. 452–461.

16. J. San Pedro, A. Karatzoglou. Question recommendation for collaborative question answering systems with Rank, 2014. P. 193–200.

Додаток А

Лістинг програмного продукту

```
# recommender/alembic.ini
# A generic, single database configuration.

[alembic]
# path to migration scripts
script_location = db/alembic

# template used to generate migration files
# file_template = %%(rev)s_%%(slug)s

# timezone to use when rendering the date
# within the migration file as well as the filename.
# string value is passed to dateutil.tz.gettz()
# leave blank for localtime
# timezone =

# max length of characters to apply to the
# "slug" field
# truncate_slug_length = 40

# set to 'true' to run the environment during
# the 'revision' command, regardless of autogenerate
# revision_environment = false

# set to 'true' to allow .pyc and .pyo files without
# a source .py file to be detected as revisions in the
# versions/ directory
# sourceless = false

# version location specification; this defaults
# to alembic/versions.  When using multiple version
# directories, initial revisions must be specified with --version-path
# version_locations = %(here)s/bar %(here)s/bat alembic/versions

# the output encoding used when revision files
# are written from script.py.mako
# output_encoding = utf-8

[post_write_hooks]
# post_write_hooks defines scripts or Python functions that are run
# on newly generated revision scripts.  See the documentation for further
# detail and examples

# format using "black" - use the console_scripts runner, against the "black" entrypoint
# hooks=black
# black.type=console_scripts
# black.entrypoint=black
# black.options=-l 79

# Logging configuration
[loggers]
keys = root,sqlalchemy,alembic

[handlers]
keys = console

[formatters]
keys = generic
```

```
[logger_root]
level = WARN
handlers = console
qualname =

[logger_sqlalchemy]
level = WARN
handlers =
qualname = sqlalchemy.engine

[logger_alembic]
level = INFO
handlers =
qualname = alembic

[handler_console]
class = StreamHandler
args = (sys.stderr,)
level = NOTSET
formatter = generic

[formatter_generic]
format = %(levelname)-5.5s [%(name)s] %(message)s
datefmt = %H:%M:%S


# recommender/utils/pg.py
import logging
import os
from collections import AsyncIterable
from pathlib import Path
from types import SimpleNamespace
from typing import Union

from aiohttp.web_app import Application
from alembic.config import Config
from asyncpgsa import PG
from asyncpgsa.transactionmanager import ConnectionTransactionContextManager
from configargparse import Namespace
from sqlalchemy import Numeric, cast, func
from sqlalchemy.sql import Select

CENSORED = "***"
DEFAULT_PG_URL = "postgresql://user:hackme@localhost:5432/recommender"
MAX_QUERY_ARGS = 32767
MAX_INTEGER = 2147483647

PROJECT_PATH = Path(__file__).parent.parent.resolve()

log = logging.getLogger(__name__)


async def setup_pg(app: Application, args: Namespace) -> PG:
    db_info = args.pg_url.with_password(CENSORED)
    log.info("Connecting to database: %s", db_info)

    app["pg"] = PG()
    await app["pg"].init(
        str(args.pg_url),
        min_size=args.pg_pool_min_size,
        max_size=args.pg_pool_max_size
    )
    await app["pg"].fetchval("SELECT 1")
```

```
    log.info("Connected to database %s", db_info)

    try:
        yield
    finally:
        log.info("Disconnecting from database %s", db_info)
        await app["pg"].pool.close()
        log.info("Disconnected from database %s", db_info)


def rounded(column, fraction: int = 2):
    return func.round(cast(column, Numeric), fraction)


def make_alembic_config(cmd_opts: Union[Namespace, SimpleNamespace],
                        base_path: str = PROJECT_PATH) -> Config:
    if not os.path.isabs(cmd_opts.config):
        cmd_opts.config = os.path.join(base_path, cmd_opts.config)

    config = Config(file_=cmd_opts.config, ini_section=cmd_opts.name,
                    cmd_opts=cmd_opts)

    alembic_location = config.get_main_option("script_location")
    if not os.path.isabs(alembic_location):
        config.set_main_option("script_location",
                               os.path.join(base_path, alembic_location))
    if cmd_opts.pg_url:
        config.set_main_option("sqlalchemy.url", cmd_opts.pg_url)

    return config


class SelectQuery(AsyncIterable):
    PREFETCH = 1000

    __slots__ = (
        "query", "transaction_ctx", "prefetch", "timeout"
    )

    def __init__(self, query: Select,
                 transaction_ctx: ConnectionTransactionContextManager,
                 prefetch: int = None,
                 timeout: float = None):
        self.query = query
        self.transaction_ctx = transaction_ctx
        self.prefetch = prefetch or self.PREFETCH
        self.timeout = timeout

    async def __aiter__(self):
        async with self.transaction_ctx as conn:
            cursor = conn.cursor(self.query, prefetch=self.prefetch,
                                 timeout=self.timeout)
            async for row in cursor:
                yield row


# recommender/utils/__init__.py


# recommender/utils/redis.py
import logging

from aiohttp.web_app import Application
```

```
from aioredis import create_redis, Redis
from configargparse import Namespace

CENSORED = "***"
DEFAULT_REDIS_URL = "redis://:hackme@localhost:6379/0"

log = logging.getLogger(__name__)


async def setup_redis(app: Application, args: Namespace) -> Redis:
    db_info = args.redis_url.with_password(CENSORED)
    log.info("Connecting to cache: %s", db_info)

    app["redis"] = await create_redis(str(args.redis_url))
    await app["redis"].ping()
    log.info(f"Connected to cache %s", db_info)

    try:
        yield
    finally:
        log.info("Disconnecting from cache %s", db_info)
        app["redis"].close()
        await app["redis"].wait_closed()
        log.info("Disconnected from cache %s", db_info)


# recommender/utils/argparse.py
import os
from argparse import ArgumentTypeError
from typing import Callable


def validate(type: Callable, constrain: Callable):
    def wrapper(value):
        value = type(value)
        if not constrain(value):
            raise ArgumentTypeError
        return value

    return wrapper


positive_int = validate(int, constrain=lambda x: x > 0)


def clear_environ(rule: Callable):
    for name in filter(rule, tuple(os.environ)):
        os.environ.pop(name)


# recommender/scripts/cache.py
import argparse
import logging

from aiomisc.log import LogFormat, basic_config
from configargparse import ArgumentParser
from lightfm import LightFM
from lightfm.datasets import fetch_stackexchange
from lightfm.evaluation import auc_score
from sqlalchemy import create_engine, select
from redis import Redis
from yarl import URL
```

```
from recommender.utils.argparse import clear_environ, positive_int
from recommender.utils.pg import DEFAULT_PG_URL
from recommender.utils.redis import DEFAULT_REDIS_URL

ENV_VAR_PREFIX = 'RECOMMENDER_'

NUM_THREADS = 1
NUM_COMPONENTS = 30
NUM_EPOCHS = 30
ITEM_ALPHA = 1e-6

log = logging.getLogger(__name__)

parser = ArgumentParser(
   auto_env_var_prefix=ENV_VAR_PREFIX, allow_abbrev=False,
   formatter_class=argparse.ArgumentDefaultsHelpFormatter
)

group = parser.add_argument_group('PostgreSQL options')
group.add_argument('--pg-url', type=URL, default=URL(DEFAULT_PG_URL),
                help='URL to use to connect to the database')

group = parser.add_argument_group('Redis options')
group.add_argument('--redis-url', type=URL, default=URL(DEFAULT_REDIS_URL),
                help='URL to use to connect to the cache')
group.add_argument("--redis-ttl", type=positive_int, default=3600,
                help="TTL for cached values")

group = parser.add_argument_group('Logging options')
group.add_argument('--log-level', default='info',
                choices=('debug', 'info', 'warning', 'error', 'fatal'))
group.add_argument('--log-format', choices=LogFormat.choices(),
                default='color')


def fetch_training_data():
   log.info("Fetching training data")

   data = fetch_stackexchange("crossvalidated",
                           test_set_fraction=0.1,
                           indicator_features=False,
                           tag_features=True)
   train = data["train"]
   test = data["test"]
   user_features = None
   item_features = data["item_features"]

   # Remove duplicated train values from test set
   t1 = set(zip(*train.nonzero()))
   t2 = set(zip(*test.nonzero()))
   test = test.tocsr()
   for idx in t1 & t2:
      test[idx] = 0
   test = test.tocoo()

   return train, test, user_features, item_features


def train_model(train, test, user_features, item_features):
   log.info("Initializing model")
   model = LightFM(loss="warp",
                   item_alpha=ITEM_ALPHA,
                   no_components=NUM_COMPONENTS)
```

```
    log.info("Training model")
    model = model.fit(train,
                      user_features=user_features,
                      item_features=item_features,
                      epochs=NUM_EPOCHS,
                      num_threads=NUM_THREADS)

    log.info("Scoring")
    train_auc = auc_score(model,
                          train,
                          user_features=user_features,
                          item_features=item_features,
                          num_threads=NUM_THREADS).mean()
    log.info(f"Training set AUC: {train_auc}")
    test_auc = auc_score(model,
                         test,
                         train_interactions=train,
                         user_features=user_features,
                         item_features=item_features,
                         num_threads=NUM_THREADS).mean()
    log.info(f"Test set AUC: {test_auc}")
    return model


def fetch_latest_items():
    log.info("Fetching latest items")
    data = fetch_stackexchange("crossvalidated",
                               test_set_fraction=0.1,
                               indicator_features=False,
                               tag_features=True)
    train = data["train"]
    return list(reversed(range(train.shape[1])))


def fetch_active_users():
    log.info("Fetching active users")
    data = fetch_stackexchange("crossvalidated",
                               test_set_fraction=0.1,
                               indicator_features=False,
                               tag_features=True)
    train = data["train"]
    return list(reversed(range(train.shape[0])))


def predict(model, user_ids, item_ids, user_features, item_features, limit=100):
    log.info("Predicting")
    predictions = {}
    for user_id in user_ids:
        pred = model.predict(user_ids=user_id,
                             item_ids=item_ids,
                             user_features=user_features,
                             item_features=item_features)
        pred = pred.argsort()[-limit:][::-1]
        predictions[user_id] = pred.tolist()
        log.debug(f"Predicted for user {user_id}")
    log.info(f"Predicted for {len(predictions)} users")
    return predictions


def cache_predictions(conn, predictions, ttl):
    log.info("Caching predictions")
    pipe = conn.pipeline()
```

```
    for user_id, pred in predictions.items():
        pipe.rpush(f"{user_id}", *pred)
        pipe.expire(f"{user_id}", ttl)
    pipe.execute()
    log.info(f"Cached predictions for {len(predictions)} users")


def cache_latest_items(conn, items, ttl):
    log.info("Caching latest items")
    pipe = conn.pipeline()
    pipe.delete("latest")
    pipe.sadd("latest", *items)
    pipe.expire("latest", ttl)
    pipe.execute()
    log.info(f"Cached {len(items)} latest items")


def main():
    args = parser.parse_args()
    clear_environ(lambda i: i.startswith(ENV_VAR_PREFIX))

    basic_config(args.log_level, args.log_format, buffered=True)

    # engine = create_engine(args.pg_url)
    # try:
    #     with engine.begin() as conn:
    users = fetch_active_users()
    items = fetch_latest_items()
    train, test, user_features, item_features = fetch_training_data()
    # finally:
    #     engine.dispose()

    model = train_model(train, test, user_features, item_features)
    predictions = predict(model, users, items, user_features, item_features)

    with Redis.from_url(str(args.redis_url)) as r:
        cache_predictions(r, predictions, args.redis_ttl)
        cache_latest_items(r, items[:100], args.redis_ttl)


if __name__ == "__main__":
    main()


# recommender/scripts/data.py
import argparse
import logging

from aiomisc.log import LogFormat, basic_config
from configargparse import ArgumentParser
from lightfm import LightFM
from lightfm.datasets import fetch_stackexchange
from lightfm.evaluation import auc_score
from sqlalchemy import create_engine, select
from redis import Redis
from yarl import URL

from recommender.db.schema import (
    users_table,
    items_table,
    interactions_table,
    item_features_table,
    item_description_table,
```

```
)
from recommender.utils.argparse import clear_environ, positive_int
from recommender.utils.pg import DEFAULT_PG_URL, MAX_QUERY_ARGS
from recommender.utils.redis import DEFAULT_REDIS_URL

ENV_VAR_PREFIX = 'RECOMMENDER_'

NUM_THREADS = 1
NUM_COMPONENTS = 30
NUM_EPOCHS = 30
ITEM_ALPHA = 1e-6

log = logging.getLogger(__name__)

parser = ArgumentParser(
    auto_env_var_prefix=ENV_VAR_PREFIX, allow_abbrev=False,
    formatter_class=argparse.ArgumentDefaultsHelpFormatter
)

group = parser.add_argument_group('PostgreSQL options')
group.add_argument('--pg-url', type=URL, default=URL(DEFAULT_PG_URL),
                   help='URL to use to connect to the database')

group = parser.add_argument_group('Logging options')
group.add_argument('--log-level', default='info',
                   choices=('debug', 'info', 'warning', 'error', 'fatal'))
group.add_argument('--log-format', choices=LogFormat.choices(),
                   default='color')


def fetch_dataset():
    log.info("Fetching dataset")
    data = fetch_stackexchange("crossvalidated",
                               test_set_fraction=0.1,
                               indicator_features=False,
                               tag_features=True)
    train = data["train"]
    test = data["test"]
    item_features = data["item_features"]
    item_feature_labels = data["item_feature_labels"]

    # Remove duplicated train values from test set
    t1 = set(zip(*train.nonzero()))
    t2 = set(zip(*test.nonzero()))
    test = test.tocsr()
    for idx in t1 & t2:
        test[idx] = 0
    test = test.tocoo()

    return train, test, item_features, item_feature_labels


def fill_users_table(conn, matrix):
    log.info(f"Filling 'users' table")
    users = [{"id": i} for i in range(matrix.shape[0])]
    batch_size = MAX_QUERY_ARGS // len(users_table.c)
    for i in range(0, len(users), batch_size):
        query = users_table.insert().values(users[i:i+batch_size])
        conn.execute(query)


def fill_items_table(conn, matrix):
    log.info(f"Filling 'items' table")
```

```
    items = [{"id": i} for i in range(matrix.shape[1])]
    batch_size = MAX_QUERY_ARGS // len(items_table.c)
    for i in range(0, len(items), batch_size):
        query = items_table.insert().values(items[i:i+batch_size])
        conn.execute(query)


def fill_interactions_table(conn, matrix):
    log.info(f"Filling 'interactions' table")

    interactions = []
    rows, cols = matrix.nonzero()
    for u, i in zip(rows, cols):
        interactions.append({"user_id": int(u), "item_id": int(i)})

    batch_size = MAX_QUERY_ARGS // len(interactions_table.c)
    for i in range(0, len(interactions), batch_size):
        query = interactions_table.insert().values(interactions[i:i+batch_size])
        conn.execute(query)


def fill_item_features_table(conn, matrix, labels):
    log.info(f"Filling 'item_features' table")

    item_features = [
        {
            "id": i,
            "description": labels[i],
            "embedding": [0 for _ in range(i)] + [1] + [0 for _ in range(i + 1, matrix.shape[1])]
        }
        for i in range(matrix.shape[1])
    ]
    batch_size = MAX_QUERY_ARGS // len(item_features_table.c)
    for i in range(0, len(item_features), batch_size):
        query = item_features_table.insert().values(item_features[i:i + batch_size])
        conn.execute(query)


def fill_item_descriptions_table(conn, matrix):
    log.info(f"Filling 'item_description' table")

    item_descriptions = []
    rows, cols = matrix.nonzero()
    for i, f in zip(rows, cols):
        item_descriptions.append({"item_id": int(i), "feature_id": int(f)})

    batch_size = MAX_QUERY_ARGS // len(item_description_table.c)
    for i in range(0, len(item_descriptions), batch_size):
        query = item_description_table.insert().values(item_descriptions[i:i + batch_size])
        conn.execute(query)


def main():
    args = parser.parse_args()
    clear_environ(lambda i: i.startswith(ENV_VAR_PREFIX))
    basic_config(args.log_level, args.log_format, buffered=True)

    train, test, item_features, item_feature_labels = fetch_dataset()

    engine = create_engine(str(args.pg_url))
    try:
        with engine.begin() as conn:
            fill_users_table(conn, train)
```

```
            fill_items_table(conn, train)
            fill_interactions_table(conn, train)
            fill_interactions_table(conn, test)
            fill_item_features_table(conn, item_features, item_feature_labels)
            fill_item_descriptions_table(conn, item_features)
    finally:
        engine.dispose()


# recommender/db/__init__.py


# recommender/db/__main__.py
import argparse
import logging
import os

from alembic.config import CommandLine

from recommender.utils.pg import DEFAULT_PG_URL, make_alembic_config


def main():
    logging.basicConfig(level=logging.DEBUG)

    alembic = CommandLine()
    alembic.parser.formatter_class = argparse.ArgumentDefaultsHelpFormatter
    alembic.parser.add_argument(
        '--pg-url', default=os.getenv('RECOMMENDER_PG_URL', DEFAULT_PG_URL),
        help='Database URL [env var: RECOMMENDER_PG_URL]'
    )

    options = alembic.parser.parse_args()
    if 'cmd' not in options:
        alembic.parser.error('too few arguments')
        exit(128)
    else:
        config = make_alembic_config(options)
        exit(alembic.run_cmd(config, options))


if __name__ == '__main__':
    main()


# recommender/db/schema.py
from sqlalchemy import (
    MetaData, Table, Column, ForeignKey,
    ARRAY, Float, Integer, Text
)

convention = {
    'all_column_names': lambda constraint, table: '_'.join([
        column.name for column in constraint.columns.values()
    ]),
    'ix': 'ix__%(table_name)s__%(all_column_names)s',
    'uq': 'uq__%(table_name)s__%(all_column_names)s',
    'ck': 'ck__%(table_name)s__%(constraint_name)s',
    'fk': 'fk__%(table_name)s__%(all_column_names)s__%(referred_table_name)s',
    'pk': 'pk__%(table_name)s'
}

metadata = MetaData(naming_convention=convention)
```

```
users_table = Table(
    "users",
    metadata,
    Column("id", Integer, primary_key=True),
)
items_table = Table(
    "items",
    metadata,
    Column("id", Integer, primary_key=True),
)
interactions_table = Table(
    "interactions",
    metadata,
    Column("id", Integer, primary_key=True, autoincrement=True),
    Column("user_id", Integer, ForeignKey("users.id", ondelete="CASCADE"), primary_key=True),
    Column("item_id", Integer, ForeignKey("items.id", ondelete="CASCADE"), primary_key=True),
)
user_features_table = Table(
    "user_features",
    metadata,
    Column("id", Integer, primary_key=True),
    Column("description", Text, nullable=True),
    Column("embedding", ARRAY(Float), nullable=False),
)
item_features_table = Table(
    "item_features",
    metadata,
    Column("id", Integer, primary_key=True),
    Column("description", Text, nullable=True),
    Column("embedding", ARRAY(Float), nullable=False),
)
user_description_table = Table(
    "user_description",
    metadata,
    Column("id", Integer, primary_key=True, autoincrement=True),
    Column("user_id", Integer, ForeignKey("users.id", ondelete="CASCADE"), primary_key=True),
    Column("feature_id", Integer, ForeignKey("user_features.id", ondelete="CASCADE"), primary_key=True),
)
item_description_table = Table(
    "item_description",
    metadata,
    Column("id", Integer, primary_key=True, autoincrement=True),
    Column("item_id", Integer, ForeignKey("items.id", ondelete="CASCADE"), primary_key=True),
    Column("feature_id", Integer, ForeignKey("item_features.id", ondelete="CASCADE"), primary_key=True),
)


# recommender/db/alembic/script.py.mako
"""${message}

Revision ID: ${up_revision}
Revises: ${down_revision | comma,n}
Create Date: ${create_date}

"""
from alembic import op
import sqlalchemy as sa
${imports if imports else ""}

# revision identifiers, used by Alembic.
revision = ${repr(up_revision)}
down_revision = ${repr(down_revision)}
```

```
branch_labels = ${repr(branch_labels)}
depends_on = ${repr(depends_on)}


def upgrade():
    ${upgrades if upgrades else "pass"}


def downgrade():
    ${downgrades if downgrades else "pass"}


# recommender/db/alembic/env.py
from logging.config import fileConfig

from alembic import context
from sqlalchemy import engine_from_config, pool

from recommender.db import schema


# this is the Alembic Config object, which provides
# access to the values within the .ini file in use.

config = context.config

# Interpret the config file for Python logging.
# This line sets up loggers basically.
fileConfig(config.config_file_name)

# add your model's MetaData object here
# for 'autogenerate' support
# from myapp import mymodel
# target_metadata = mymodel.Base.metadata
target_metadata = schema.metadata

# other values from the config, defined by the needs of env.py,
# can be acquired:
# my_important_option = config.get_main_option("my_important_option")
# ... etc.


def run_migrations_offline():
    """Run migrations in 'offline' mode.

    This configures the context with just a URL
    and not an Engine, though an Engine is acceptable
    here as well.  By skipping the Engine creation
    we don't even need a DBAPI to be available.

    Calls to context.execute() here emit the given string to the
    script output.

    """
    url = config.get_main_option("sqlalchemy.url")
    context.configure(
        url=url,
        target_metadata=target_metadata,
        literal_binds=True,
        dialect_opts={"paramstyle": "named"},
    )

    with context.begin_transaction():
```

```
        context.run_migrations()


def run_migrations_online():
    """Run migrations in 'online' mode.

    In this scenario we need to create an Engine
    and associate a connection with the context.

    """
    connectable = engine_from_config(
        config.get_section(config.config_ini_section),
        prefix="sqlalchemy.",
        poolclass=pool.NullPool,
    )

    with connectable.connect() as connection:
        context.configure(
            connection=connection, target_metadata=target_metadata
        )

        with context.begin_transaction():
            context.run_migrations()


if context.is_offline_mode():
    run_migrations_offline()
else:
    run_migrations_online()


# recommender/db/alembic/versions/e6659ab3464d_add_cascade_deletions.py
"""Add cascade deletions

Revision ID: e6659ab3464d
Revises: ebf43a9c712b
Create Date: 2020-05-31 18:27:26.515730

"""
from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision = 'e6659ab3464d'
down_revision = 'ebf43a9c712b'
branch_labels = None
depends_on = None


def upgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_constraint('fk__interactions__item_id__items', 'interactions', type_='foreignkey')
    op.drop_constraint('fk__interactions__user_id__users', 'interactions', type_='foreignkey')
    op.create_foreign_key(op.f('fk__interactions__item_id__items'), 'interactions', 'items', ['item_id'], ['id'], ondelete='
CASCADE')
    op.create_foreign_key(op.f('fk__interactions__user_id__users'), 'interactions', 'users', ['user_id'], ['id'], ondelete='
CASCADE')
    op.drop_constraint('fk__item_description__feature_id__item_features', 'item_description', type_='foreignkey')
    op.drop_constraint('fk__item_description__item_id__items', 'item_description', type_='foreignkey')
    op.create_foreign_key(op.f('fk__item_description__item_id__items'), 'item_description', 'items', ['item_id'], ['id'],
ondelete='CASCADE')
```

```
    op.create_foreign_key(op.f('fk__item_description__feature_id__item_features'), 'item_description', 'item_features
', ['feature_id'], ['id'], ondelete='CASCADE')
    op.drop_constraint('fk__user_description__feature_id__user_features', 'user_description', type_='foreignkey')
    op.drop_constraint('fk__user_description__user_id__users', 'user_description', type_='foreignkey')
    op.create_foreign_key(op.f('fk__user_description__user_id__users'), 'user_description', 'users', ['user_id'], ['id'], o
ndelete='CASCADE')
    op.create_foreign_key(op.f('fk__user_description__feature_id__user_features'), 'user_description', 'user_features',
['feature_id'], ['id'], ondelete='CASCADE')
    # ### end Alembic commands ###


def downgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_constraint(op.f('fk__user_description__feature_id__user_features'), 'user_description', type_='foreignkey'
)
    op.drop_constraint(op.f('fk__user_description__user_id__users'), 'user_description', type_='foreignkey')
    op.create_foreign_key('fk__user_description__user_id__users', 'user_description', 'users', ['user_id'], ['id'])
    op.create_foreign_key('fk__user_description__feature_id__user_features', 'user_description', 'user_features', ['feat
ure_id'], ['id'])
    op.drop_constraint(op.f('fk__item_description__feature_id__item_features'), 'item_description', type_='foreignke
y')
    op.drop_constraint(op.f('fk__item_description__item_id__items'), 'item_description', type_='foreignkey')
    op.create_foreign_key('fk__item_description__item_id__items', 'item_description', 'items', ['item_id'], ['id'])
    op.create_foreign_key('fk__item_description__feature_id__item_features', 'item_description', 'item_features', ['fe
ature_id'], ['id'])
    op.drop_constraint(op.f('fk__interactions__user_id__users'), 'interactions', type_='foreignkey')
    op.drop_constraint(op.f('fk__interactions__item_id__items'), 'interactions', type_='foreignkey')
    op.create_foreign_key('fk__interactions__user_id__users', 'interactions', 'users', ['user_id'], ['id'])
    op.create_foreign_key('fk__interactions__item_id__items', 'interactions', 'items', ['item_id'], ['id'])
    # ### end Alembic commands ###


# recommender/db/alembic/versions/ebf43a9c712b_create_aggregation_function.py
"""Create aggregation function

Revision ID: ebf43a9c712b
Revises: d4c028ee7223
Create Date: 2020-05-31 12:04:45.024497

"""
from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision = 'ebf43a9c712b'
down_revision = 'd4c028ee7223'
branch_labels = None
depends_on = None


def upgrade():
    conn = op.get_bind()
    conn.execute(
        sa.text("""
            CREATE OR REPLACE FUNCTION vec_add(arr1 double precision[], arr2 double precision[])
            RETURNS double precision[] AS
            $$
            SELECT array_agg(result)
            FROM (SELECT coalesce(tuple.val1, 0) + coalesce(tuple.val2, 0) AS result
                FROM (SELECT UNNEST($1) AS val1,
                             UNNEST($2) AS val2,
                             generate_subscripts($1, 1) AS ix) tuple
```

```
            ORDER BY ix) inn;
        $$ LANGUAGE SQL IMMUTABLE STRICT;

        CREATE AGGREGATE vec_sum(double precision[]) (
          SFUNC = vec_add,
          STYPE = double precision[]
        );
      """)
  )


def downgrade():
  conn = op.get_bind()
  conn.execute(
    sa.text("""
      DROP AGGREGATE vec_sum(double precision[]);
      DROP FUNCTION vec_add(arr1 double precision[], arr2 double precision[]);
    """)
  )


# recommender/db/alembic/versions/d4c028ee7223_intitial.py
"""Intitial

Revision ID: d4c028ee7223
Revises:
Create Date: 2020-05-31 11:13:32.071954

"""
from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision = 'd4c028ee7223'
down_revision = None
branch_labels = None
depends_on = None


def upgrade():
  # ### commands auto generated by Alembic - please adjust! ###
  op.create_table(
    'item_features',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('description', sa.Text(), nullable=True),
    sa.Column('embedding', sa.ARRAY(sa.Float()), nullable=False),
    sa.PrimaryKeyConstraint('id', name=op.f('pk__item_features'))
  )
  op.create_table(
    'items',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.PrimaryKeyConstraint('id', name=op.f('pk__items'))
  )
  op.create_table(
    'user_features',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('description', sa.Text(), nullable=True),
    sa.Column('embedding', sa.ARRAY(sa.Float()), nullable=False),
    sa.PrimaryKeyConstraint('id', name=op.f('pk__user_features'))
  )
  op.create_table(
    'users',
```

```
        sa.Column('id', sa.Integer(), nullable=False),
        sa.PrimaryKeyConstraint('id', name=op.f('pk__users'))
    )
    op.create_table(
        'interactions',
        sa.Column('id', sa.Integer(), nullable=False, autoincrement=True),
        sa.Column('user_id', sa.Integer(), nullable=False),
        sa.Column('item_id', sa.Integer(), nullable=False),
        sa.ForeignKeyConstraint(['item_id'], ['items.id'], name=op.f('fk__interactions__item_id__items')),
        sa.ForeignKeyConstraint(['user_id'], ['users.id'], name=op.f('fk__interactions__user_id__users')),
        sa.PrimaryKeyConstraint('id', 'user_id', 'item_id', name=op.f('pk__interactions'))
    )
    op.create_table(
        'item_description',
        sa.Column('id', sa.Integer(), nullable=False, autoincrement=True),
        sa.Column('item_id', sa.Integer(), nullable=False),
        sa.Column('feature_id', sa.Integer(), nullable=False),
        sa.ForeignKeyConstraint(['feature_id'], ['item_features.id'], name=op.f('fk__item_description__feature_id__ite
m_features')),
        sa.ForeignKeyConstraint(['item_id'], ['items.id'], name=op.f('fk__item_description__item_id__items')),
        sa.PrimaryKeyConstraint('id', 'item_id', 'feature_id', name=op.f('pk__item_description'))
    )
    op.create_table(
        'user_description',
        sa.Column('id', sa.Integer(), nullable=False, autoincrement=True),
        sa.Column('user_id', sa.Integer(), nullable=False),
        sa.Column('feature_id', sa.Integer(), nullable=False),
        sa.ForeignKeyConstraint(['feature_id'], ['user_features.id'], name=op.f('fk__user_description__feature_id__use
r_features')),
        sa.ForeignKeyConstraint(['user_id'], ['users.id'], name=op.f('fk__user_description__user_id__users')),
        sa.PrimaryKeyConstraint('id', 'user_id', 'feature_id', name=op.f('pk__user_description'))
    )
    # ### end Alembic commands ###


def downgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_table('user_description')
    op.drop_table('item_description')
    op.drop_table('interactions')
    op.drop_table('users')
    op.drop_table('user_features')
    op.drop_table('items')
    op.drop_table('item_features')
    # ### end Alembic commands ###


# recommender/api/__init__.py


# recommender/api/payloads.py
import json
from decimal import Decimal
from functools import partial, singledispatch
from typing import Any

from aiohttp.payload import JsonPayload as BaseJsonPayload, Payload
from aiohttp.typedefs import JSONEncoder
from asyncpg import Record


@singledispatch
def convert(value):
```

```
    raise TypeError(f"Unserializable value: {value!r}")


@convert.register(Record)
def convert_asyncpg_record(value: Record):
    return dict(value)


@convert.register(Decimal)
def convert_decimal(value: Decimal):
    return float(value)


dumps = partial(json.dumps, default=convert, ensure_ascii=False)


class JsonPayload(BaseJsonPayload):
    def __init__(self,
                 value: Any,
                 encoding: str = "utf-8",
                 content_type: str = "application/json",
                 dumps: JSONEncoder = dumps,
                 *args: Any,
                 **kwargs: Any) -> None:
        super().__init__(value, encoding, content_type, dumps, *args, **kwargs)


class AsyncGenJSONListPayload(Payload):
    def __init__(self, value, encoding: str = "utf-8",
                 content_type: str = "application/json",
                 root_object: str = "data",
                 *args, **kwargs):
        self.root_object = root_object
        super().__init__(value, content_type=content_type, encoding=encoding,
                         *args, **kwargs)

    async def write(self, writer):
        # Начало объекта
        await writer.write(
            ('{"%s":[' % self.root_object).encode(self._encoding)
        )

        first = True
        async for row in self._value:
            # Перед первой строчкой запятая не нужнаа
            if not first:
                await writer.write(b",")
            else:
                first = False

            await writer.write(dumps(row).encode(self._encoding))

        # Конец объекта
        await writer.write(b"]}")


__all__ = ("JsonPayload", "AsyncGenJSONListPayload")


# recommender/api/app.py
import logging
from functools import partial
from types import MappingProxyType
```

```
from typing import Mapping

from aiohttp import PAYLOAD_REGISTRY
from aiohttp.web_app import Application
from aiohttp_apispec import setup_aiohttp_apispec, validation_middleware
from configargparse import Namespace

from recommender.api.handlers import HANDLERS
from recommender.api.middleware import error_middleware, handle_validation_error
from recommender.api.payloads import JsonPayload
from recommender.utils.pg import setup_pg
from recommender.utils.redis import setup_redis

log = logging.getLogger(__name__)


def create_app(args: Namespace) -> Application:
    app = Application(middlewares=[error_middleware, validation_middleware])
    app.cleanup_ctx.append(partial(setup_pg, args=args))
    app.cleanup_ctx.append(partial(setup_redis, args=args))

    for handler in HANDLERS:
        log.debug('Registering handler %r as %r', handler, handler.URL_PATH)
        app.router.add_route('*', handler.URL_PATH, handler)

    setup_aiohttp_apispec(app=app, title='Recommender API',
                          error_callback=handle_validation_error)

    PAYLOAD_REGISTRY.register(JsonPayload, (Mapping, MappingProxyType))
    return app


# recommender/api/__main__.py
import argparse
import logging
import os
import pwd
from sys import argv

from aiohttp.web import run_app
from aiomisc import bind_socket
from aiomisc.log import LogFormat, basic_config
from configargparse import ArgumentParser
from setproctitle import setproctitle
from yarl import URL

from recommender.api.app import create_app
from recommender.utils.argparse import clear_environ, positive_int
from recommender.utils.pg import DEFAULT_PG_URL
from recommender.utils.redis import DEFAULT_REDIS_URL


ENV_VAR_PREFIX = 'RECOMMENDER_'


parser = ArgumentParser(
    auto_env_var_prefix=ENV_VAR_PREFIX, allow_abbrev=False,
    formatter_class=argparse.ArgumentDefaultsHelpFormatter
)
parser.add_argument('--user', required=False, type=pwd.getpwnam,
                    help='Change process UID')

group = parser.add_argument_group('API Options')
```

```
group.add_argument('--api-address', default='0.0.0.0',
                   help='IPv4/IPv6 address API server would listen on')
group.add_argument('--api-port', type=positive_int, default=8081,
                   help='TCP port API server would listen on')

group = parser.add_argument_group('PostgreSQL options')
group.add_argument('--pg-url', type=URL, default=URL(DEFAULT_PG_URL),
                   help='URL to use to connect to the database')
group.add_argument('--pg-pool-min-size', type=int, default=10,
                   help='Minimum database connections')
group.add_argument('--pg-pool-max-size', type=int, default=10,
                   help='Maximum database connections')

group = parser.add_argument_group('Redis options')
group.add_argument('--redis-url', type=URL, default=URL(DEFAULT_REDIS_URL),
                   help='URL to use to connect to the cache')

group = parser.add_argument_group('Logging options')
group.add_argument('--log-level', default='info',
                   choices=('debug', 'info', 'warning', 'error', 'fatal'))
group.add_argument('--log-format', choices=LogFormat.choices(),
                   default='color')


def main():
    args = parser.parse_args()
    clear_environ(lambda i: i.startswith(ENV_VAR_PREFIX))

    basic_config(args.log_level, args.log_format, buffered=True)

    sock = bind_socket(address=args.api_address, port=args.api_port,
                       proto_name='http')

    if args.user is not None:
        logging.info('Changing user to %r', args.user.pw_name)
        os.setgid(args.user.pw_gid)
        os.setuid(args.user.pw_uid)
    setproctitle(os.path.basename(argv[0]))

    app = create_app(args)
    run_app(app, sock=sock)


if __name__ == '__main__':
    main()


# recommender/api/middleware.py
import logging
from http import HTTPStatus
from typing import Mapping, Optional

from aiohttp.web_exceptions import (
    HTTPBadRequest, HTTPException, HTTPInternalServerError,
)
from aiohttp.web_middlewares import middleware
from aiohttp.web_request import Request
from marshmallow import ValidationError

from recommender.api.payloads import JsonPayload


log = logging.getLogger(__name__)
```

```
def format_http_error(http_error_cls, message: Optional[str] = None,
                      fields: Optional[Mapping] = None) -> HTTPException:
    status = HTTPStatus(http_error_cls.status_code)
    error = {
        "code": status.name.lower(),
        "message": message or status.description
    }

    if fields:
        error["fields"] = fields

    return http_error_cls(body={"error": error})


def handle_validation_error(error: ValidationError, *_):
    raise format_http_error(HTTPBadRequest, "Request validation has failed",
                            error.messages)


@middleware
async def error_middleware(request: Request, handler):
    try:
        return await handler(request)
    except HTTPException as err:
        if not isinstance(err.body, JsonPayload):
            err = format_http_error(err.__class__, err.text)
        raise err
    except ValidationError as err:
        raise handle_validation_error(err)
    except Exception:
        log.exception("Unhandled exception")
        raise format_http_error(HTTPInternalServerError)


# recommender/api/schema.py
from marshmallow import Schema
from marshmallow.fields import Dict, Int, List, Nested, Str
from marshmallow.validate import Range


class ItemSchema(Schema):
    feature_ids = List(Int(validate=Range(min=1), strict=True), default=[])


class UserSchema(Schema):
    feature_ids = List(Int(validate=Range(min=1), strict=True), default=[])


class HistoryQSSchema(Schema):
    limit = Int(validate=Range(min=1, max=100), default=10)


class RecommendationQSSchema(Schema):
    limit = Int(validate=Range(min=1, max=100), default=10)


class ErrorSchema(Schema):
    code = Str(required=True)
    message = Str(required=True)
    fields = Dict()
```

```
class ErrorResponseSchema(Schema):
    error = Nested(ErrorSchema(), required=True)


# recommender/api/handlers/user.py
from aiohttp.web_exceptions import HTTPOk, HTTPAccepted, HTTPNoContent
from aiohttp_apispec import request_schema
from sqlalchemy import func, select

from recommender.api.schema import UserSchema
from recommender.db.schema import (
    users_table, user_features_table, user_description_table
)

from .base import BaseUserView


class UserView(BaseUserView):
    URL_PATH = r"/users/{user_id:\d+}"

    @staticmethod
    async def get_user(conn, user_id):
        query = select([
            users_table.c.id,
            func.vec_sum(
                user_features_table.c.embedding
            ).label("embedding"),
            func.array_remove(
                func.array_agg(user_features_table.c.id),
                None
            ).label("feature_ids")
        ]).select_from(
            users_table.outerjoin(
                user_description_table,
                users_table.c.id == user_description_table.c.user_id
            ).outerjoin(
                user_features_table,
                user_features_table.c.id == user_description_table.c.feature_id
            )
        ).where(
            users_table.c.id == user_id
        ).group_by(
            users_table.c.id
        )
        return await conn.fetchrow(query)

    @staticmethod
    async def get_features(conn, feature_ids):
        if not feature_ids:
            return []
        query = select([
            user_features_table.c.id,
            user_features_table.c.description,
            user_features_table.c.embedding,
        ]).where(
            user_features_table.c.id.in_(feature_ids)
        )
        return await conn.fetch(query)

    # @staticmethod
    # def compute_embedding(features):
    #     if not features:
```

```
    #         return None
    #     embeddings = [f["embedding"] for f in features]
    #     return np.sum(np.array(embeddings), axis=0).tolist()

    @staticmethod
    async def create_user(conn, user_id, data):
        values = {k: v for k, v in data.users() if k != "features"}
        values["id"] = user_id
        query = users_table.insert().values(values)
        await conn.execute(query)

    @staticmethod
    async def create_user_description(conn, user_id, feature_ids):
        values = []
        for feature_id in feature_ids:
            values.append({"user_id": user_id,
                           "feature_id": feature_id})
        query = user_description_table.insert().values(values)
        await conn.execute(query)

    @staticmethod
    async def delete_user(conn, user_id):
        query = users_table.delete().where(
            users_table.c.id == user_id
        )
        await conn.execute(query)

    async def get(self):
        await self.check_user_exists()
        async with self.pg.transaction() as conn:
            user = await self.get_user(conn, self.user_id)
            # TODO Get features via one query
            user = dict(user)
            feature_ids = user.pop("feature_ids", [])
            user["features"] = await self.get_features(conn, feature_ids)
        return HTTPOk(body={"data": user})

    @request_schema(UserSchema)
    async def put(self):
        async with self.pg.transaction() as conn:
            feature_ids = self.request["data"].pop("feature_ids")
            await self.create_user(conn, self.user_id, self.request["data"])
            await self.create_user_description(conn, self.user_id, feature_ids)
        return HTTPAccepted()

    async def delete(self):
        await self.check_user_exists()
        async with self.pg.transaction() as conn:
            await self.delete_user(conn, self.user_id)
        return HTTPNoContent()


# recommender/api/handlers/user_history.py
from aiohttp.web_exceptions import HTTPOk
from aiohttp_apispec import querystring_schema
from sqlalchemy import select, desc

from recommender.api.schema import HistoryQSSchema
from recommender.db.schema import (
    users_table, interactions_table, items_table
)

from .base import BaseUserView
```

```
class UserHistoryView(BaseUserView):
    URL_PATH = r"/users/{user_id:\d+}/history"

    @property
    def limit(self):
        return int(self.request["querystring"]["limit"])

    @staticmethod
    async def get_history(conn, limit):
        query = select([
            items_table.c.id,
        ]).select_from(
            users_table.outerjoin(
                interactions_table,
                users_table.c.id == interactions_table.c.user_id
            ).outerjoin(
                items_table,
                items_table.c.id == interactions_table.c.item_id
            )
        ).group_by(
            interactions_table.c.id,
            items_table.c.id,
        ).order_by(
            desc(interactions_table.c.id)
        ).limit(limit)
        return await conn.fetch(query)

    @querystring_schema(HistoryQSSchema)
    async def get(self):
        await self.check_user_exists()

        async with self.pg.transaction() as conn:
            history = await self.get_history(conn, self.limit)
        return HTTPOk(body={"data": history})


# recommender/api/handlers/__init__.py
from .item import ItemView
from .interactions import InteractionView
from .user import UserView
from .user_history import UserHistoryView
from .user_recommendations import UserRecommendationsView

HANDLERS = (
    ItemView, UserView, UserHistoryView,
    UserRecommendationsView, InteractionView,
)


# recommender/api/handlers/user_recommendations.py
from aiohttp.web_exceptions import HTTPOk
from aiohttp_apispec import querystring_schema

from recommender.api.schema import RecommendationQSSchema

from .base import BaseUserView


class UserRecommendationsView(BaseUserView):
    URL_PATH = r"/users/{user_id:\d+}/recommendations"
```

```
    @property
    def limit(self):
        return int(self.request["querystring"]["limit"])

    @querystring_schema(RecommendationQSSchema)
    async def get(self):
        await self.check_user_exists()

        items = await self.redis.lrange(f"{self.user_id}", 0, self.limit)
        if not items:
            items = await self.redis.srandmember("latest", self.limit)

        items = [{"id": int(i.decode())} for i in items]
        return HTTPOk(body={"data": items})


# recommender/api/handlers/interactions.py
from aiohttp.web_exceptions import HTTPAccepted
from recommender.db.schema import interactions_table

from .base import BaseUserView, BaseItemView


class InteractionView(BaseUserView, BaseItemView):
    URL_PATH = r"/users/{user_id:\d+}/interact/{item_id:\d+}"

    @staticmethod
    async def save_interaction(conn, user_id, item_id):
        value = {"user_id": user_id, "item_id": item_id}
        query = interactions_table.insert().values(value)
        await conn.execute(query)

    async def post(self):
        await self.check_user_exists()
        await self.check_item_exists()

        async with self.pg.transaction() as conn:
            await self.save_interaction(conn, self.user_id, self.item_id)

        return HTTPAccepted()


# recommender/api/handlers/base.py
from aiohttp.web_exceptions import HTTPNotFound
from aiohttp.web_urldispatcher import View
from aioredis import Redis
from asyncpgsa import PG
from sqlalchemy import select, exists

from recommender.db.schema import items_table, users_table


class BaseView(View):
    URL_PATH: str

    @property
    def pg(self) -> PG:
        return self.request.app["pg"]

    @property
    def redis(self) -> Redis:
        return self.request.app["redis"]
```

```
class BaseItemView(BaseView):
    @property
    def item_id(self):
        return int(self.request.match_info.get("item_id"))

    async def check_item_exists(self):
        query = select([
            exists().where(items_table.c.id == self.item_id)
        ])
        if not await self.pg.fetchval(query):
            raise HTTPNotFound()


class BaseUserView(BaseView):
    @property
    def user_id(self):
        return int(self.request.match_info.get("user_id"))

    async def check_user_exists(self):
        query = select([
            exists().where(users_table.c.id == self.user_id)
        ])
        if not await self.pg.fetchval(query):
            raise HTTPNotFound()


# recommender/api/handlers/item.py
from aiohttp.web_exceptions import HTTPOk, HTTPAccepted, HTTPNoContent
from aiohttp_apispec import request_schema
from sqlalchemy import func, select

from recommender.api.schema import ItemSchema
from recommender.db.schema import (
    items_table, item_features_table, item_description_table
)

from .base import BaseItemView


class ItemView(BaseItemView):
    URL_PATH = r"/items/{item_id:\d+}"

    @staticmethod
    async def get_item(conn, item_id):
        query = select([
            items_table.c.id,
            func.vec_sum(
                item_features_table.c.embedding
            ).label("embedding"),
            func.array_remove(
                func.array_agg(item_features_table.c.id),
                None
            ).label("feature_ids")
        ]).select_from(
            items_table.outerjoin(
                item_description_table,
                items_table.c.id == item_description_table.c.item_id
            ).outerjoin(
                item_features_table,
                item_features_table.c.id == item_description_table.c.feature_id
            )
        ).where(
```

```
            items_table.c.id == item_id
        ).group_by(
            items_table.c.id
        )
        return await conn.fetchrow(query)

    @staticmethod
    async def get_features(conn, feature_ids):
        if not feature_ids:
            return []
        query = select([
            item_features_table.c.id,
            item_features_table.c.description,
            item_features_table.c.embedding,
        ]).where(
            item_features_table.c.id.in_(feature_ids)
        )
        return await conn.fetch(query)

    # @staticmethod
    # def compute_embedding(features):
    #     if not features:
    #         return None
    #     embeddings = [f["embedding"] for f in features]
    #     return np.sum(np.array(embeddings), axis=0).tolist()

    @staticmethod
    async def create_item(conn, item_id, data):
        values = {k: v for k, v in data.items() if k != "features"}
        values["id"] = item_id
        query = items_table.insert().values(values)
        await conn.execute(query)

    @staticmethod
    async def create_item_description(conn, item_id, feature_ids):
        values = []
        for feature_id in feature_ids:
            values.append({"item_id": item_id,
                           "feature_id": feature_id})
        query = item_description_table.insert().values(values)
        await conn.execute(query)

    @staticmethod
    async def delete_item(conn, item_id):
        query = items_table.delete().where(
            items_table.c.id == item_id
        )
        await conn.execute(query)

    async def get(self):
        await self.check_item_exists()
        async with self.pg.transaction() as conn:
            item = await self.get_item(conn, self.item_id)
            # TODO Get features via one query
            item = dict(item)
            feature_ids = item.pop("feature_ids", [])
            item["features"] = await self.get_features(conn, feature_ids)
        return HTTPOk(body={"data": item})

    @request_schema(ItemSchema)
    async def put(self):
        async with self.pg.transaction() as conn:
            feature_ids = self.request["data"].pop("feature_ids")
```

```
            await self.create_item(conn, self.item_id, self.request["data"])
            await self.create_item_description(conn, self.item_id, feature_ids)
        return HTTPAccepted()

    async def delete(self):
        await self.check_item_exists()
        async with self.pg.transaction() as conn:
            await self.delete_item(conn, self.item_id)
        return HTTPNoContent()
```

Додаток Б

Ілюстративний матеріал

Зображення слайдів з презентації

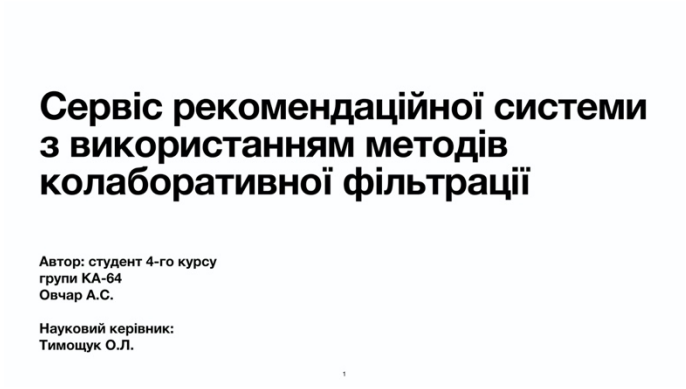

Рисунок Б.1 – Слайд №1

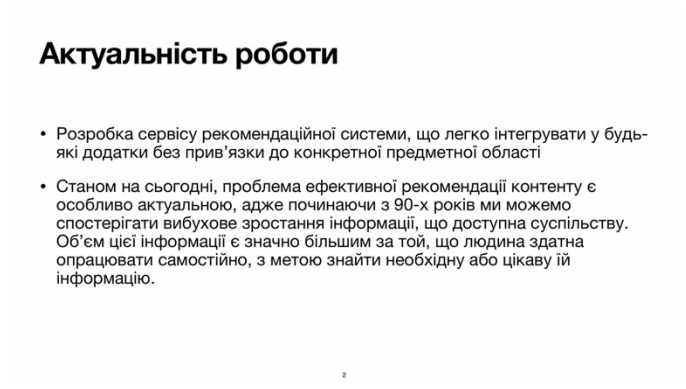

Рисунок Б.2 – Слайд №2

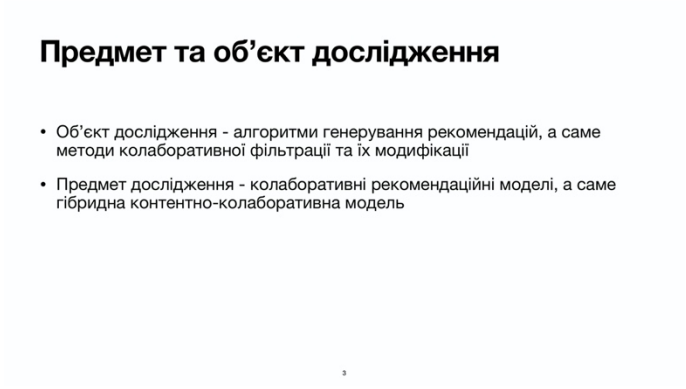

Рисунок Б.3 – Слайд №3

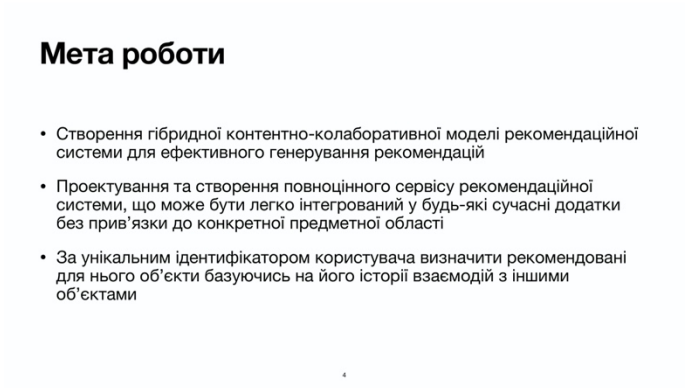

Рисунок Б.4 – Слайд №4

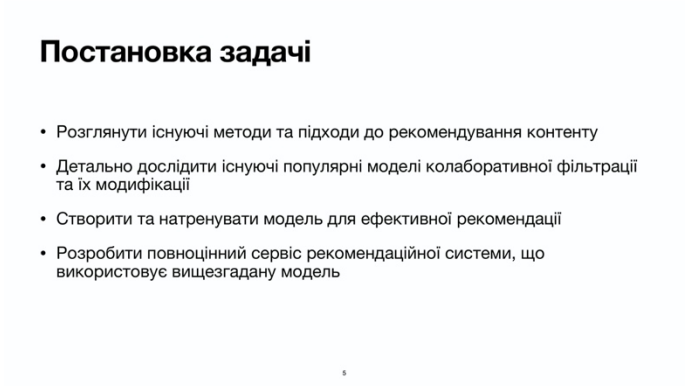

Рисунок Б.5 – Слайд №5

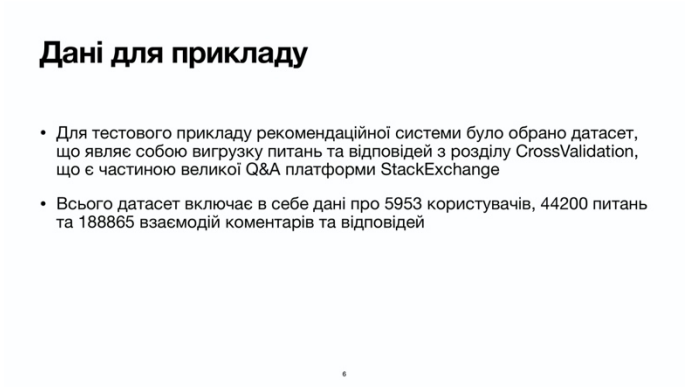

Рисунок Б.6 – Слайд №6

Гібридна контентно-колаборативна модель

$$q_u = \sum_{j \in f_u} e_j^U \qquad p_i = \sum_{j \in f_i} e_j^I \qquad b_u = \sum_{j \in f_u} b_j^U \qquad b_i = \sum_{j \in f_i} b_j^I$$

$$\hat{r} = f(q_u \times p_i + b_u + b_i)$$

$$f(x) = \frac{1}{1 + \exp(-x)}$$

$$L(e^U, e^I, b^U, b^I) = \prod_{(u,i) \in S^+} \hat{r}_{ui} \times \prod_{(u,i) \in S^-} (1 - \hat{r}_{ui})$$

7

Рисунок Б.7 – Слайд №7

Гібридна контентно-колаборативна модель

	CrossValidated	
	Warm	Cold
LSI-LR	0.662	0.660
LSI-UP	0.636	0.637
MF	0.541	0.508
LightFM (tags)	0.675	0.675
LightFM (tags + ids)	0.682	0.674
LightFM (tags + about)	**0.695**	**0.696**

8

Рисунок Б.8 – Слайд №8

Технології, що були використані в роботі

Python LightFM PostgreSQL Redis

9

Рисунок Б.9 – Слайд №9

Рисунок Б.10 – Слайд №10

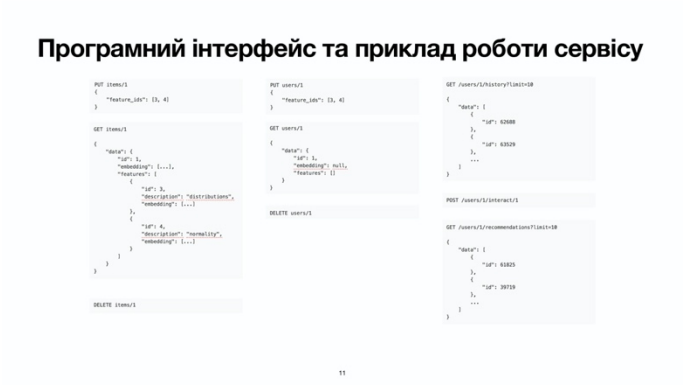

Рисунок Б.11 – Слайд №11

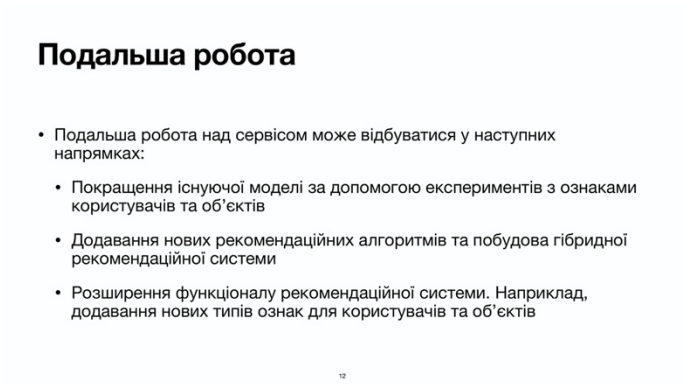

Рисунок Б.12– Слайд №12

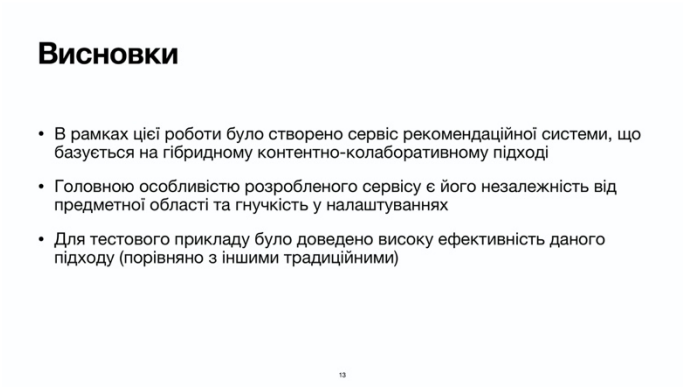

Рисунок Б.13 – Слайд №13

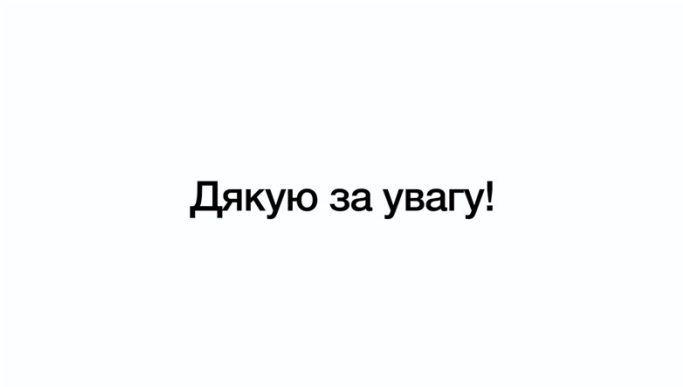

Рисунок Б.14 – Слайд №14