Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістра
(освітньо-кваліфікаційний рівень)

на тему Персоналізовані рекомендаційні системи для електронної комерції на основі гібридних AI-моделей

Виконав: студент 6 курсу, групи 601-ТН
спеціальності
122 Комп'ютерні науки
(шифр і назва напряму)
Корнієнко Д.С.
(прізвище та ініціали)
Керівник Пенц В.Ф.
(прізвище та ініціали)
Рецензент Фуголь В.С.
(прізвище та ініціали)

Полтава – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

спеціальність 122 «Комп'ютерні науки»

на тему

«Персоналізовані рекомендаційні системи для електронної комерції на основі гібридних AI-моделей»

Студента групи 601-ТН Корнієнка Дениса Сергійовича

Керівник роботи
кандидат технічних наук,
доцент Пенц В.Ф.

Консультант
кандидат технічних наук,
доцент Скакаліна О.В.

Завідувач кафедри
кандидат фізико-математичних наук,
доцент Двірна О.А.

Полтава – 2024

РЕФЕРАТ

Кваліфікаційна робота магістра: 90 с., 10 малюнків, 1 додаток, 5 таблиць, 43 джерел.

Об’єкт дослідження: рекомендаційні системи для електронної комерції.

Мета роботи: розробка та впровадження гібридної рекомендаційної системи для електронної комерції, яка поєднує підходи колаборативної фільтрації та контентного аналізу для підвищення точності, релевантності та різноманітності рекомендацій.

Методи дослідження: застосування методи машинного навчання, такі як матрична факторизація для аналізу взаємодій користувачів і TF-IDF для обробки текстових даних. Для оцінки ефективності використовували метрики точності (Precision, Recall) та помилок прогнозування (RMSE, MAE). Реалізація системи проводилася за допомогою Python та бібліотек Scikit-learn і LightFM.

Ключові слова: рекомендаційна система, штучний інтелект, е-комерція.

ABSTRACT

Master's qualification work: 90 pp., 10 figs, 1 appendix, 5 tables, 43 sources.

Object of research: recommender systems for e-commerce.

Purpose: to develop and implement a hybrid recommender system for e-commerce that combines collaborative filtering and content analysis approaches to improve the accuracy, relevance and diversity of recommendations.

Research methods: machine learning techniques such as matrix factorisation for analysing user interactions and TF-IDF for processing text data. To evaluate the effectiveness, we used the metrics of accuracy (Precision, Recall) and prediction errors (RMSE, MAE). The system was implemented using Python and the Scikit-learn and LightFM libraries.

Key words: recommendation system, artificial intelligence, e-commerce.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

РС – рекомендаційні системи.

ШІ – штучний інтелект.

CF (Collaborative Filtering) – колаборативна фільтрація.

CB (Content-Based) – контентний підхід.

SVD (Singular Value Decomposition) – сингулярне розкладання.

TF-IDF (Term Frequency-Inverse Document Frequency) – частота термів/інверсна частота документів.

MAE (Mean Absolute Error) – середня абсолютна похибка.

RMSE (Root Mean Squared Error) – квадратична середня похибка.

MAP (Mean Average Precision) – середня точність.

NDCG (Normalized Discounted Cumulative Gain) – нормалізована дисконтована сукупна вигода.

ML (Machine Learning) – машинне навчання.

ВСТУП

У сучасному світі електронної комерції персоналізовані рекомендаційні системи відіграють ключову роль у забезпеченні успішної взаємодії між користувачами та онлайн-платформами. Різке зростання кількості товарів і послуг, доступних на ринку, створює проблему вибору для споживачів, яка, без допомоги технологічних рішень, може стати значною перешкодою для прийняття рішень. У таких умовах рекомендаційні системи є критично важливим інструментом, який допомагає користувачам швидко знаходити релевантні товари та підвищувати задоволеність покупкою, що, у свою чергу, збільшує лояльність клієнтів та обсяги продажів компаній.

З огляду на зростаючу конкуренцію в електронній комерції, актуальність питання ефективності рекомендаційних систем тільки зростає. Компанії прагнуть оптимізувати свої бізнес-процеси, залучаючи більше користувачів і забезпечуючи кращий досвід взаємодії з платформою, що стає можливим завдяки сучасним технологіям штучного інтелекту. Машинне та глибоке навчання відкривають нові горизонти для рекомендаційних систем, дозволяючи створювати більш точні та адаптивні моделі, які враховують різноманітні фактори поведінки користувачів. Використання гібридних моделей, що поєднують кілька підходів до персоналізації, також є важливим напрямом досліджень і розробок у цій сфері.

Дана робота націлена на дослідження ефективності та можливостей гібридних рекомендаційних систем, які інтегрують різні методи машинного та глибокого навчання для підвищення точності та релевантності рекомендацій в електронній комерції. Ця тема є актуальною, оскільки пошук нових шляхів удосконалення рекомендаційних моделей не лише допоможе оптимізувати процес купівлі для користувачів, але й надасть бізнесу конкурентні переваги, що є особливо важливим в умовах постійно зростаючого інформаційного потоку. Тому дослідження можливостей гібридних рекомендаційних систем має значний

науковий та практичний інтерес, адже може суттєво вплинути на розвиток як електронної комерції, так і технологій штучного інтелекту загалом.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність надання рекомендацій

Рекомендаційні системи стали важливою складовою сучасного цифрового світу, оскільки вони значно впливають на якість взаємодії користувачів із платформами, сервісами та інтернет-ресурсами. У світі, де кількість доступної інформації стрімко зростає, а вибір можливостей стає дедалі складнішим, рекомендаційні системи дозволяють не лише спрощувати пошук релевантного контенту, але й формувати унікальний, персоналізований досвід для кожного користувача. Ці системи перетворилися на незамінний інструмент для бізнесу, науки та технологій, відіграючи провідну роль у створенні комфортного й ефективного середовища для взаємодії з інформацією.

Завдяки рекомендаційним системам ми отримуємо можливість значно полегшити процес пошуку потрібного контенту, зменшуючи час і зусилля, витрачені на прийняття рішень. Вони допомагають знайти музику [1], фільми [2], книги, товари чи навіть партнерів для соціальної взаємодії [3]. Наприклад, платформи потокового відео, такі як Netflix або YouTube, використовують рекомендаційні системи для того, щоб запропонувати нам ті фільми або відео, які найбільш відповідають нашим вподобанням, створюючи враження, ніби контент підібраний спеціально для нас. Так само сервіси, як-от Amazon чи Spotify, допомагають відкривати нові товари, музику чи послуги, які могли б зацікавити користувача, базуючись на його попередніх діях або схожих вподобаннях інших користувачів.

Персоналізація є ключовим елементом рекомендаційних систем, яка робить кожного користувача унікальним. Сучасні рекомендаційні системи не просто показують варіанти, вони вивчають вподобання користувачів, аналізуючи їхні дії, такі як перегляди, оцінки, вподобання та покупки. Вони здатні прогнозувати майбутні потреби користувачів, адаптуючи свої пропозиції

в реальному часі. Завдяки цьому користувачі проводять більше часу на платформі, насолоджуючись релевантними рекомендаціями, що позитивно впливає на їхню задоволеність.

У бізнес-контексті використання рекомендаційних систем стало стратегічною перевагою. Вони сприяють збільшенню доходів компаній за рахунок персоналізованих пропозицій, що підвищують ймовірність покупки [4]. Також ці системи забезпечують вищий рівень лояльності клієнтів, адже користувачі, які отримують релевантні рекомендації, частіше повертаються до сервісу.

Незважаючи на значні успіхи, рекомендаційні системи стикаються з багатьма викликами. Зокрема, це проблема роботи з неповними або неоднозначними даними, необхідність обробки динамічних змін у вподобаннях користувачів і постійна потреба у вдосконаленні алгоритмів для забезпечення більшої точності. Проте ці труднощі є стимулом для подальшого розвитку та інновацій у цій сфері.

1.2 Поняття рекомендаційної системи

Рекомендаційна система — це програмно-апаратний комплекс, який аналізує великі обсяги даних з метою надання користувачам персоналізованих рекомендацій щодо продуктів, послуг чи контенту. Вона слугує інтелектуальним посередником між користувачем і великим потоком інформації, забезпечуючи швидкий та ефективний доступ до найбільш релевантних об’єктів. Основною метою РС є допомога користувачу в пошуку інформації, яка відповідає його індивідуальним потребам та вподобанням, шляхом автоматизованої обробки даних і прогнозування його майбутніх дій.

Рекомендаційна система адаптується до кожного користувача, використовуючи методи штучного інтелекту, машинного навчання та математичного аналізу. Вона визначає ймовірність зацікавленості користувача у певному об'єкті, формуючи персоналізовані списки рекомендацій. Формально це

можна описати як функцію, яка прогнозує оцінку або інтерес користувача до об’єкта на основі зібраних даних.

Систему рекомендацій можна формально визначити за допомогою наступної моделі: розглянемо множину всіх користувачів, позначену як U, і множину всіх елементів, які можна рекомендувати, позначену як I. Нехай функція корисності f вимірює корисність певного товару i для користувача u, тобто f:

$$U \times I \rightarrow R$$

де R - впорядкована множина. Тоді для кожного користувача $u \in U$ рекомендованими товарами $i' \in I$ є ті, які максимізують корисність для цього користувача. Іншими словами, це можна виразити наступним чином [5]:

$$\forall u \in U,\ i'_u = argmax_{i \in I} f(u,i)$$

1.2.1 Основні характеристики РС. Загальна схема основних характеристик РС зображено на рис. 1.1.

Однією з ключових рис рекомендаційних систем є персоналізація, яка дозволяє враховувати унікальні особливості кожного користувача. Вона базується на історії взаємодії, уподобаннях та навіть демографічних характеристиках користувача. Персоналізація створює відчуття індивідуального підходу, що покращує користувацький досвід і сприяє лояльності.

Прогнозування є ще однією важливою характеристикою. Рекомендаційні системи використовують алгоритми для передбачення того, які об'єкти можуть зацікавити користувача в майбутньому. Цей процес базується на аналізі минулої активності, врахуванні схожості між користувачами або об'єктами та використанні статистичних моделей.

Рекомендаційні системи також демонструють високий рівень масштабованості. Вони здатні працювати з мільйонами користувачів і об'єктів

одночасно, використовуючи сучасні технології обробки великих даних, такі як Hadoop, Spark чи MapReduce.

Адаптивність системи забезпечує її здатність змінюватися відповідно до нових вподобань користувачів, змін у доступному контенті або умовах ринку. Це дозволяє зберігати актуальність рекомендацій навіть у динамічних середовищах.

Сучасні рекомендаційні системи також інтерактивні. Вони дають змогу користувачам оцінювати запропоновані рекомендації, залишати відгуки та уточнювати свої вподобання. Ця зворотна взаємодія допомагає вдосконалювати алгоритми та підвищує точність рекомендацій.

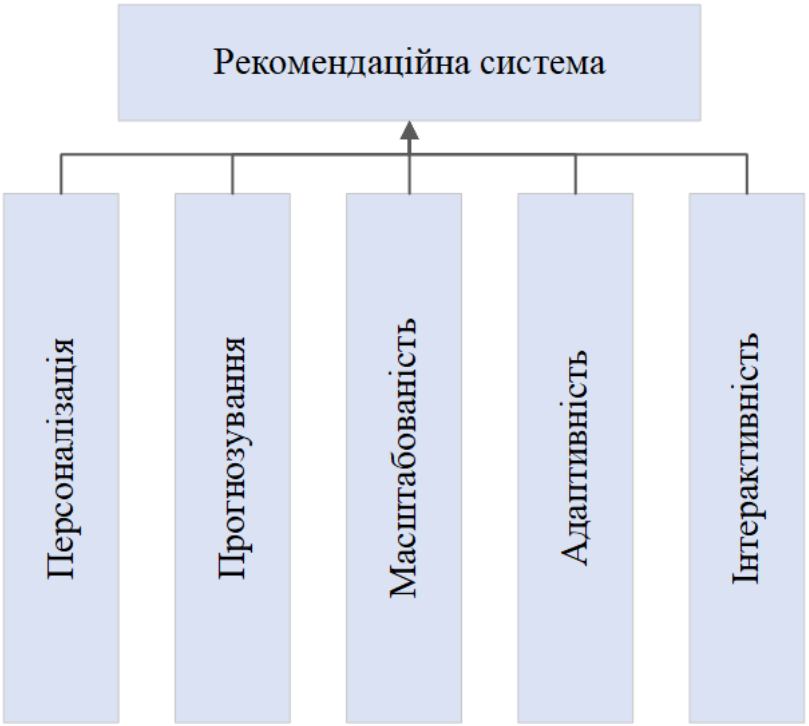

Рисунок 1.1 – Основні характеристики РС

1.2.2 Основні етапи роботи. Принципова схема рекомендаційної системи показана на рис. 1.2. Рекомендаційна система починає свою роботу зі збору даних. Інформація про користувачів і об'єкти може включати явні оцінки (наприклад, оцінка товару "зірками") та неявні сигнали (наприклад, час перегляду, кліки або покупки). Дані також можуть бути демографічними, такими як вік, стать чи місце проживання. Збір інформації може здійснюватися через анкетування, форми реєстрації або автоматично під час взаємодії з платформою.

Зібрані дані проходять обробку та аналіз, у процесі яких вони очищуються, структуруються та використовуються для побудови профілів користувачів і об'єктів. На цьому етапі застосовуються такі методи, як кластеризація (групування схожих користувачів чи об'єктів), виявлення закономірностей у даних та створення векторних репрезентацій для об’єктів і користувачів.

Наступним етапом є формування моделі рекомендацій. Це може включати застосування контентно-орієнтованих алгоритмів, які аналізують характеристики об'єктів, що вже вподобав користувач, або колаборативної фільтрації, яка ґрунтується на схожості між користувачами чи об'єктами. Також популярним підходом є гібридні моделі, які об’єднують кілька методів для досягнення більшої точності.

Після цього відбувається генерація рекомендацій, коли система створює персоналізовані списки об'єктів, релевантних для кожного користувача. Це можуть бути списки товарів, запропонованих фільмів або статей, ранжованих за ступенем ймовірної зацікавленості.

Завершальним етапом є оцінка ефективності рекомендаційної системи. Для цього використовуються різні метрики, такі як точність (precision), повнота (recall) або середня точність ранжування (MAP). Ці метрики дозволяють оцінити, наскільки система відповідає очікуванням користувачів і допомагають вдосконалювати алгоритми.

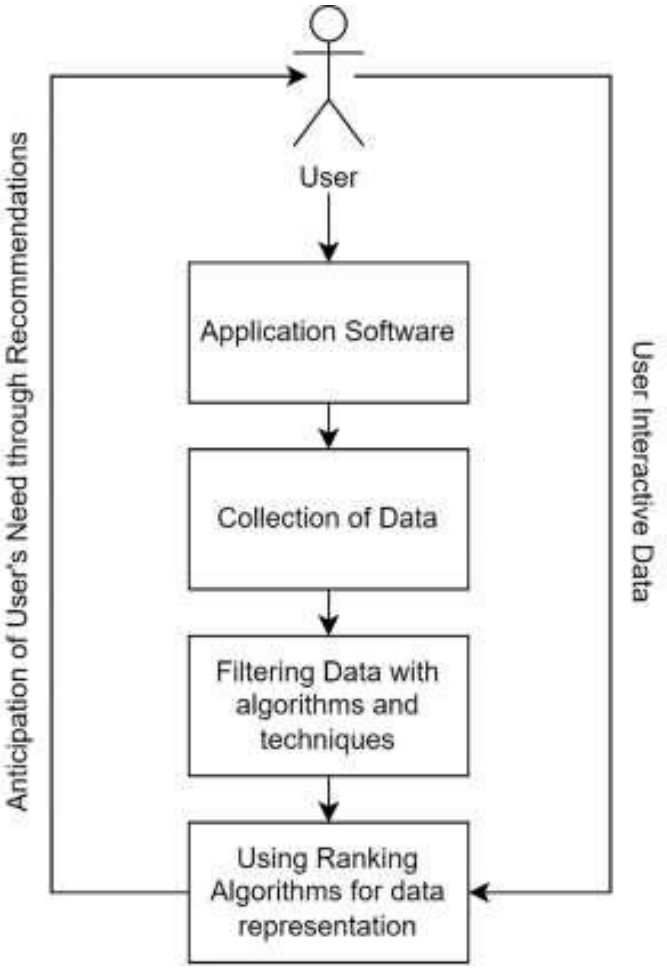

Рисунок 1.2 – Структура рекомендаційної системи

1.3 Історичний розвиток і сучасний стан предметної області

Рекомендаційні системи мають багатий історичний розвиток, який бере свій початок у часи зародження електронної комерції. Їх еволюція нерозривно пов'язана з розвитком штучного інтелекту та технологій машинного навчання, які поступово зробили ці системи ключовим інструментом у багатьох галузях, включаючи торгівлю, медіа, освіту та соціальні мережі.

На ранніх етапах розвитку рекомендаційні системи базувалися на простих правилах. Вони функціонували за принципом жорстко закодованих сценаріїв, які передбачали рекомендації, виходячи з обмеженої кількості факторів, таких як категорія продукту або демографічні дані користувача. Однак ці системи були надзвичайно обмеженими у своїх можливостях, оскільки не враховували індивідуальних уподобань користувачів і змін у їхній поведінці.

Ідея створення більш адаптивних і персоналізованих систем зародилася у 1990-х роках, коли електронна комерція почала активно розвиватися. Перші інтернет-магазини використовували статичні веб-сторінки, і їхні можливості щодо персоналізації були вкрай обмеженими. На цьому етапі стали з’являтися базові методи рекомендацій, такі як рекомендації на основі правил і прості алгоритми колаборативної фільтрації.

Системи, засновані на правилах, мали значні недоліки, серед яких виділяється їхня неспроможність адаптуватися до змін у вподобаннях користувачів. Колаборативна фільтрація, у свою чергу, зіткнулася з проблемами, зокрема так званою проблемою "холодного старту", коли для нових користувачів було недостатньо даних для формування якісних рекомендацій. Ці виклики стимулювали подальший розвиток рекомендаційних алгоритмів і їх інтеграцію з новими технологіями.

Сучасний стан рекомендаційних систем характеризується використанням широкого спектра підходів і методів ШІ, таких як колаборативна фільтрація, фільтрація на основі контенту та гібридні моделі. Системи, що застосовують контент-орієнтовані підходи, аналізують атрибути об’єктів, які вже сподобалися користувачам, щоб запропонувати схожі варіанти. Колаборативна фільтрація фокусується на пошуку схожих користувачів або об'єктів для створення рекомендацій. Гібридні методи поєднують обидва підходи, щоб підвищити точність і релевантність рекомендацій.

Технологічний прогрес, зокрема розвиток методів обробки великих даних, став важливою складовою сучасних рекомендаційних систем. Інструменти на кшталт Hadoop, Spark або MapReduce дозволяють ефективно працювати з величезними обсягами інформації, що виникає у процесі роботи платформ із мільйонами користувачів і продуктів. Крім того, сучасні алгоритми, такі як глибоке навчання та методи графових моделей, забезпечують нові можливості для аналізу складних зв’язків між об’єктами і користувачами.

Важливо також відзначити вплив змін у поведінці споживачів на розвиток рекомендаційних систем. Злиття е-комунікацій і телекомунікацій сприяло

формуванню нового типу платформ, які не лише надають персоналізовані рекомендації, але й активно адаптуються до змін у вподобаннях користувачів у режимі реального часу.

Історія рекомендаційних систем демонструє поступовий перехід від простих, жорстко закодованих алгоритмів до складних, гнучких моделей, що використовують сучасні технології штучного інтелекту. Вони стали невід'ємною частиною багатьох платформ і продовжують активно розвиватися, пристосовуючись до нових викликів і потреб користувачів.

1.4 Вплив рекомендаційних систем на користувачів.

Рекомендаційні системи суттєво впливають на досвід користувачів у цифровому середовищі, формуючи їхні взаємодії з платформами, контентом і продуктами. Їхній вплив можна оцінювати з кількох перспектив, серед яких виділяються персоналізація досвіду, спрощення пошуку, підтримка прийняття рішень та формування поведінкових змін.

Однією з головних переваг рекомендаційних систем є їхня здатність створювати персоналізований досвід для кожного користувача. Вони аналізують поведінкові патерни, інтереси та вподобання користувачів, щоб запропонувати релевантний контент, продукти чи послуги. Наприклад, потокові сервіси, такі як Netflix або Spotify, використовують дані про попередні перегляди чи прослуховування, щоб рекомендувати фільми, серіали або музику, які відповідають смакам користувача. Це підвищує задоволення від використання платформи та сприяє формуванню тривалішої взаємодії.

Рекомендаційні системи також відіграють ключову роль у спрощенні пошуку інформації. У сучасному світі з величезними обсягами даних і контенту знайти те, що відповідає конкретним потребам, може бути складним завданням. Завдяки використанню ефективних алгоритмів, таких як фільтрація на основі контенту, колаборативна фільтрація або гібридні методи, системи допомагають

користувачам швидше знаходити потрібну інформацію. Це зменшує когнітивне навантаження та економить час.

Окрім цього, рекомендаційні системи значно впливають на прийняття рішень. Вони не лише пропонують варіанти, але й можуть формувати пріоритетність вибору, наприклад, показуючи товари з позитивними відгуками чи популярні серед інших користувачів. Такі системи активно впливають на процес купівлі, вибір навчальних курсів або навіть соціальні взаємодії, спрямовуючи користувачів до найбільш підходящих варіантів.

Рекомендаційні системи також сприяють формуванню поведінкових змін. Наприклад, користувачі можуть відкривати для себе нові інтереси чи сфери завдяки тому, що система пропонує контент, який вони могли б ігнорувати без відповідних рекомендацій. У сфері здоров’я рекомендаційні системи можуть заохочувати корисні звички, пропонуючи відповідні додатки, програми чи продукти.

Однак вплив рекомендаційних систем не завжди є позитивним. Вони можуть створювати ефект інформаційного «бульбашкового простору», обмежуючи користувачів контентом, який лише підтверджує їхні вже існуючі переконання. Це знижує можливість відкриття нових ідей чи перспектив. Крім того, залежність від алгоритмів може призводити до втрати критичного мислення під час прийняття рішень.

1.5 Застосування рекомендаційних систем в електронній комерції

Рекомендаційні системи відіграють ключову роль у сфері електронної комерції, сприяючи покращенню взаємодії між платформами та їх користувачами. Їх основна мета — створення персоналізованого досвіду, що дозволяє адаптувати процес покупок до індивідуальних потреб та вподобань кожного клієнта. Це не лише підвищує зацікавленість користувачів, але й формує довготривалу лояльність до бренду чи платформи.

Персоналізація в електронній комерції — це створення унікального цифрового маршруту для кожного покупця. Рекомендаційні системи аналізують поведінкові патерни користувачів, їхні попередні покупки, вподобання та пошукові запити, щоб запропонувати найбільш релевантні товари чи послуги. Такий підхід не лише спрощує процес пошуку, але й робить його приємним і ефективним. Уявіть собі персонального шопінг-асистента, який знає ваші вподобання та веде вас до ідеального вибору, — саме так працюють сучасні системи рекомендацій.

Сучасний розвиток електронної комерції неможливий без впровадження штучного інтелекту, зокрема алгоритмів машинного навчання. Завдяки цьому платформи отримали можливість обробляти великі обсяги даних, передбачати поведінку користувачів та створювати індивідуальні пропозиції в реальному часі. Наприклад, такі гіганти, як Amazon чи AliExpress, використовують складні алгоритми колаборативної фільтрації, фільтрації на основі контенту та гібридні моделі, щоб запропонувати клієнтам товари, які максимально відповідають їхнім потребам.

Персоналізація має ще один важливий аспект — вона допомагає зменшити перевантаженість користувачів інформацією. Без належної організації та рекомендацій покупці можуть стикатися з так званою «втомою від вибору», коли через надмірний обсяг продуктів ухвалення рішення стає складним і виснажливим. Натомість рекомендаційні системи спрощують цей процес, пропонуючи лише найбільш релевантні варіанти.

Крім покращення досвіду користувачів, системи рекомендацій приносять значні переваги бізнесу. Вони сприяють підвищенню конверсії продажів, оскільки клієнти частіше обирають товари, які відповідають їхнім уподобанням. До того ж, такі системи стимулюють повторні покупки, допомагають утримувати клієнтів і формують їхню довіру до бренду.

Прогрес у галузі обчислювальних потужностей, збільшення обсягів даних та розвиток сучасних алгоритмів зробили можливим використання складних моделей штучного інтелекту, які здатні миттєво адаптуватися до змін у поведінці

користувачів. Таким чином, рекомендаційні системи не лише задовольняють існуючі потреби, але й допомагають користувачам відкривати нові можливості та розширювати свої інтереси.

Висновок до розділу

У першому розділі було розглянуто основні аспекти, пов'язані з рекомендаційними системами, їх актуальністю, структурою, розвитком та впливом на користувачів і бізнес.

Рекомендаційні системи відіграють ключову роль у пошуку й відкритті релевантної інформації, оптимізації процесів вибору та підвищенні ефективності взаємодії з цифровими платформами. Актуальність таких систем особливо проявляється в умовах постійного зростання обсягів інформації та складності її обробки.

Визначено, що такі системи функціонують на основі індивідуалізованого підходу, аналізуючи поведінку користувачів і пропонуючи рекомендації на основі різних методів. Основні етапи роботи рекомендаційних систем включають збір даних, аналіз та генерацію рекомендацій, що забезпечує їхню точність і релевантність.

Встановлено, що рекомендаційні системи пройшли шлях від базових алгоритмів, заснованих на правилах, до сучасних технологій, що використовують потужність штучного інтелекту й машинного навчання. Цей прогрес став можливим завдяки розвитку обчислювальних потужностей і технологій обробки великих обсягів даних.

Аналіз впливу рекомендаційних систем на користувачів засвідчив їх важливість у формуванні персоналізованого досвіду, підвищенні зручності використання цифрових платформ і формуванні довіри до брендів. Водночас рекомендаційні системи відкривають нові можливості для бізнесу, сприяючи утриманню клієнтів, підвищенню їхньої лояльності та зростанню продажів.

Системи є потужним інструментом персоналізації, який дозволяє не лише полегшити процес вибору для користувачів, але й значно покращити ключові показники ефективності бізнесу.

Узагалі, рекомендаційні системи мають величезний потенціал. Вони не тільки допомагають користувачам ефективніше знаходити потрібну інформацію, але й сприяють розвитку бізнесу, наукових досліджень і технологій. Завдяки їм ми можемо покращити якість життя в цифровому світі, зробивши його більш зручним, ефективним та індивідуалізованим.

Перший розділ закладає основу для подальшого дослідження рекомендаційних систем, демонструючи їхню багатогранність, значення та потенціал для подальшого розвитку в умовах динамічного цифрового середовища.

РОЗДІЛ 2
ПІДХОДИ ДО РЕАЛІЗАЦІЇ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

2.1 Загальний огляд методів побудови рекомендаційних систем

Методи побудови рекомендаційних систем є основою для створення ефективних і персоналізованих рекомендацій. Їх головна мета — автоматизувати процес підбору релевантного контенту, товарів чи послуг для користувачів, виходячи з їхніх індивідуальних вподобань, поведінки чи атрибутів. Кожен метод має свої сильні сторони, слабкі місця та області застосування.

Нижче показано найбільш застосовувані підходи, що дозволяють формувати точні та персоналізовані рекомендації:

1. Фільтрація на основі контенту (Content-Based Filtering)
2. Колаборативна фільтрація (Collaborative Filtering)
3. Гібридні методи (Hybrid Methods)
4. Методи на основі нейронних мереж (Deep Learning-Based Methods)
5. Контекстно-залежні методи (Context-Aware Systems)
6. Ранжування на основі навчання (Learning-to-Rank, LTR)
7. Графові підходи (Graph-Based Methods)
8. Байєсівські моделі (Bayesian Models)
9. Методи на основі факторизації матриць (Matrix Factorization)
10. Еволюційні алгоритми та оптимізація
11. Рекомендації на основі зовнішніх даних (Knowledge-Based Systems)
12. Системи на основі підкріплювального навчання (Reinforcement Learning)
13. Емоційно-залежні системи (Emotion-Aware Systems)
14. Методи на основі кластеризації

Розвиток кожного з методів триває в напрямку їхньої інтеграції, оптимізації обчислювальних ресурсів і адаптації до нових сценаріїв

використання. Це дозволяє створювати рекомендаційні системи, які є не лише ефективними, але й інтуїтивними для кінцевого користувача.

2.2 Класичні підходи до рекомендацій

Класичні підходи до побудови рекомендаційних систем є основою сучасних методів персоналізації, які забезпечують користувачів релевантними продуктами, послугами чи інформацією. Вони стали фундаментом для подальшого розвитку більш складних алгоритмів, що базуються на штучному інтелекті. Основними серед цих підходів є контентно-орієнтована фільтрація, колаборативна фільтрація та гібридні методи.

На рис. 2.1 наведено схему, в якій подано стислий опис моделей рекомендацій.

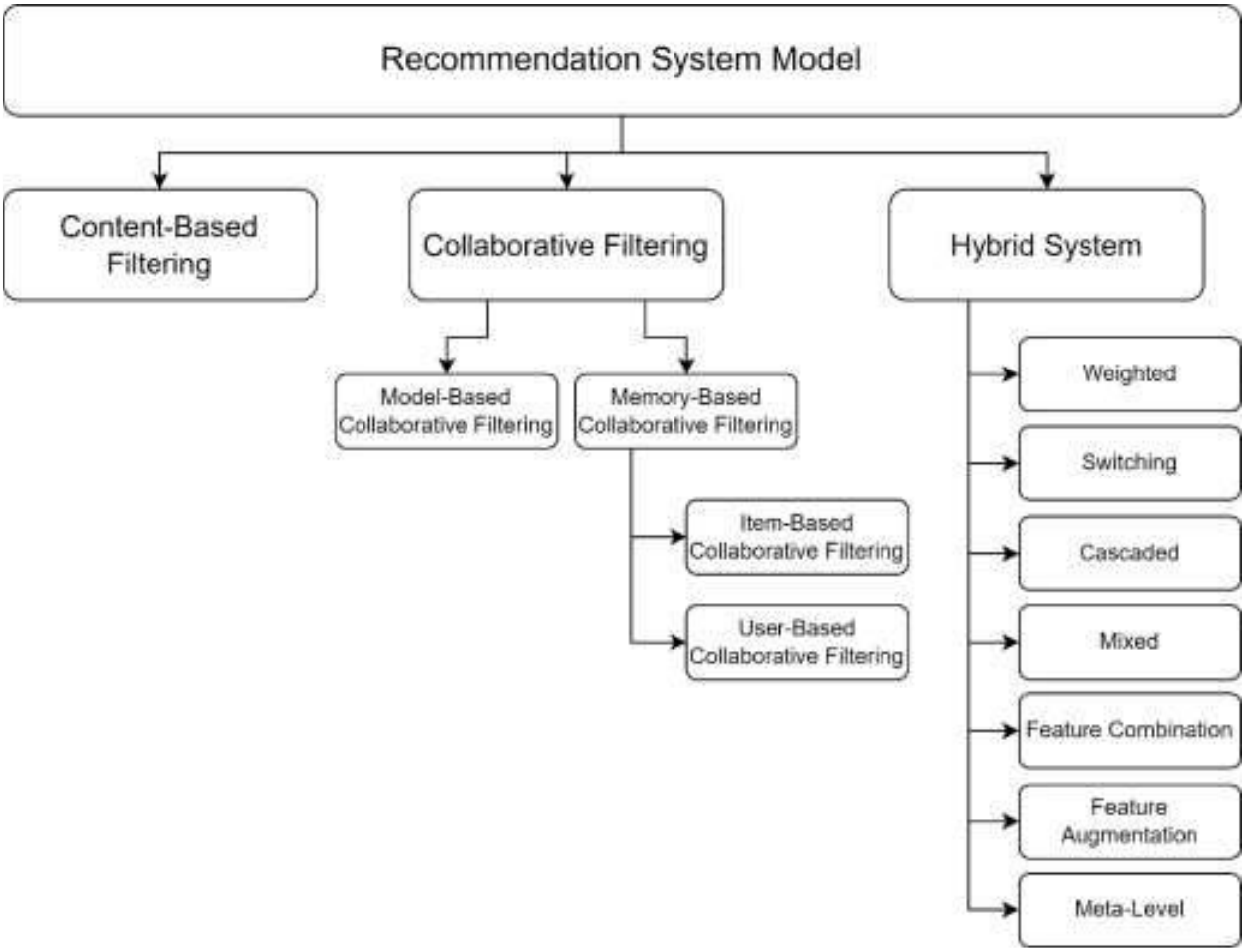

Рисунок 2.1 – Огляд моделей рекомендацій

2.2.1 Колаборативна фільтрація. У 1990-х роках з'явилася модель фільтрації інформації, відома як колаборативна фільтрація, яка стала важливим етапом для подальших досліджень у сфері рекомендаційних систем [6, 7]. Колаборативна фільтрація є одним із найбільш широко використовуваних класичних підходів. Її головна ідея полягає в тому, що користувачі з подібними вподобаннями, ймовірно, мають схожі інтереси [8].

Колаборативна фільтрація може бути заснована на користувачах (user-based) або на об'єктах (item-based) [9]. У першому випадку система аналізує схожість між користувачами, щоб запропонувати їм об'єкти, які вони ще не бачили, але які сподобалися подібним користувачам. У другому випадку рекомендації формуються на основі схожості між самими об'єктами. Незважаючи на свою ефективність, колаборативна фільтрація стикається з такими проблемами, як "холодний старт" (відсутність даних про нових користувачів або об'єкти) і "рідкість даних" (велика кількість незаповнених взаємодій у матриці рейтингів).

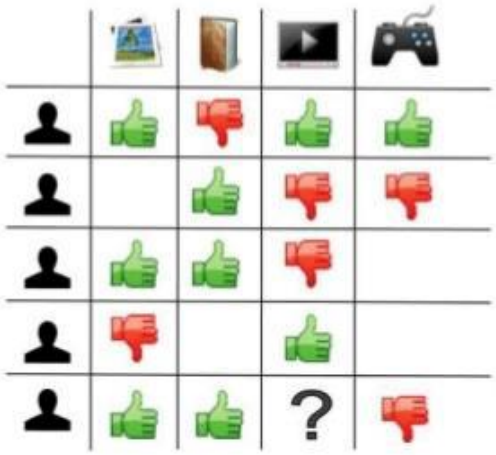

Рисунок 2.2 – риклад матриці колаборативної фільтрації

Реалізація колаборативної фільтрації на основі пам'яті може бути досягнута за допомогою різних методів, включаючи кореляцію Пірсона, коефіцієнт Жаккара, скоригований косинус подібності (являє собою кут між векторами оцінок) і метод найближчого сусіда (KNN) [10, 11]. Було показано, що ця стратегія є успішною в тих сферах, де пропозиція товарів є надзвичайно

важливою, наприклад, в електронній комерції, де вона допомагає зменшити втрату клієнтів і робить продажі більш успішними [12].

Косинус подібності:

$$cosineSimilarity(u_i, u_j) = cos(u_i, u_j) = \frac{\sum_{k=1}^{n} u_{i,k} u_{j,k}}{\sqrt{\sum_{k=1}^{n} u_{i,k}^2} \sqrt{\sum_{k=1}^{n} u_{j,k}^2}}$$

Критерій Пірсона:

$$PearsonSimilarity(u_i, u_j) = \frac{\sum_{k=1}^{n} (u_{i,k_k} - \overline{u_\iota})(u_{j,k_k} - \overline{u_j})}{\sqrt{\sum_{k=1}^{n} (u_{i,k} - \overline{u_\iota})^2} \sqrt{\sum_{k=1}^{n} (u_{j,k} - \overline{u_j})^2}}$$

Скоригований косинус подібності:

$$adjustedCosineSimilarity(u_i, u_j) = \frac{\sum_{k=1}^{n} (u_{i_k} - \overline{u_k})(u_{j_k} - \overline{u_k})}{\sqrt{\sum_{k=1}^{n} (u_{i,k} - \overline{u_k})^2} \sqrt{\sum_{k=1}^{n} (u_{j,k} - \overline{u_k})^2}}$$

Коефіцієнт Жаккара:

$$JaccardSimilarity(u_i, u_j) = \frac{|u_i \cap u_j|}{|u_i \cup u_j|}$$

де u_i, u_j – вектори оцінок користувачів *i* та *j*, n – кількість предметів, $\overline{u_\iota}, \overline{u_j}$ – середні оцінки користувачів *i* та *j*, $\overline{u_k}$ – середня оцінка предмета *k*.

З іншого боку, колаборативна фільтрація на основі пам'яті не позбавлена недоліків, включаючи проблеми "сірих овець", холодного старту та ефекту розрідженості. Коли недостатньо інформації для формулювання пропозицій, на сцені виникає проблема, відома як розрідженість [13]. Кажуть, що нові користувачі мають "холодний старт", коли вони не мають жодної історії оцінок

[14]. "Сірі вівці" - це явище, яке виникає, коли невелика кількість користувачів має вподобання, які можна порівняти з вподобаннями певної особи [15].

Кластеризація, одновимірна декомпозиція (SVD) та аналіз головних компонент (PCA) - це деякі з підходів, які були використані при розробці моделей колаборативної фільтрації на основі моделей [16]. Ці моделі були створені для того, щоб врахувати обмеження, зазначені вище. Останнім часом дослідження інтенсивно зосереджуються на підвищенні ефективності колаборативної фільтрації. Це включає вдосконалення методів оцінки схожості, використання вхідних даних від користувачів, збільшення кількості даних про вподобання користувачів та інші фактори [17, 18], [19].

Як наслідок, колаборативна фільтрація є важливим компонентом рекомендаційних систем, і вона постійно розвивається і знаходить застосування в широкому спектрі галузей. Однак, щоб досягти кращих результатів, дослідники в цій галузі активно працюють над пошуком рішень для подолання перешкод, які виникають під час використання цієї техніки з метою покращення результатів.

2.2.2 Методи на основі контенту. Існує багато різних моделей фільтрації інформації, які розвивалися з 1992 року, починаючи з дослідження, проведеного Лоебом та ін. [20]. Одна з цих моделей називається фільтрація на основі контенту (Content-Based Filtering), яка ґрунтується на аналізі атрибутів об'єктів, які сподобалися користувачеві в минулому. Система будує профіль користувача, враховуючи характеристики контенту, що був ним оцінений. Наприклад, якщо користувач часто дивиться науково-фантастичні фільми, система пропонуватиме йому інші фільми цього жанру. Загалом, це досить простий метод, який використовувався на ранніх стадіях розвитку рекомендаційних систем.

З іншого боку, важливо зазначити, що дослідження, проведене Солтером та його колегами [21], чітко показало недоліки цього підходу. Єдині об'єкти, які рекомендуються фільтрацією на основі контенту, - це ті, що мають атрибути, які можна порівняти з атрибутами об'єктів, які користувач оцінював у минулому.

Через це алгоритм не може надавати точні рекомендації для нових об'єктів, а також має обмежений доступ до широкого кола інформації. Здебільшого такий підхід використовувався у сферах, де пропозиції робилися на основі інформації про характеристики об'єктів, наприклад, у випадку з навчальними матеріалами, фільмами, музикою та речами, що продаються через інтернет-магазини.

Тобто, незважаючи на ефективність підходу для персоналізації, він має обмеження: користувачі можуть отримувати рекомендації лише в межах схожого контенту, а система не може вийти за рамки його історії взаємодій.

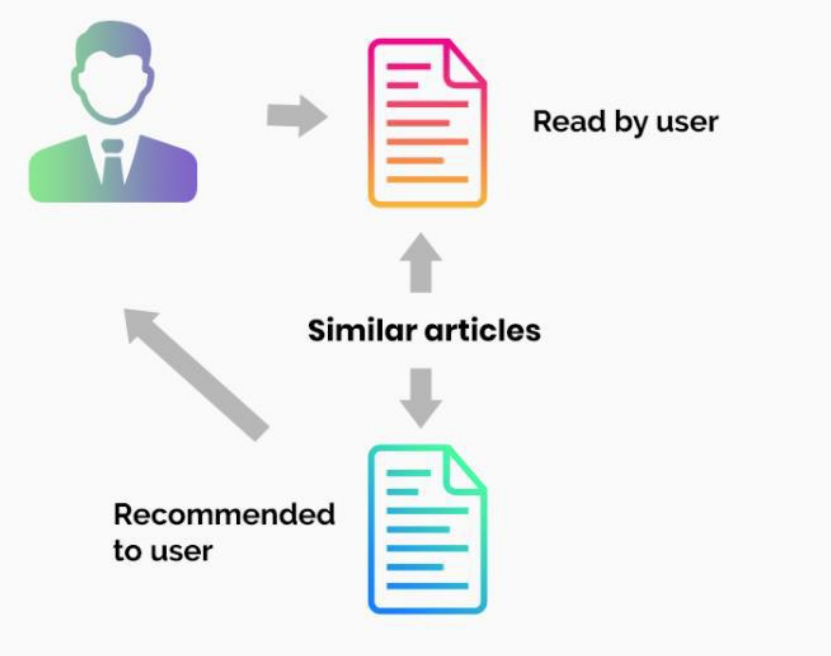

Рисунок 2.3 – Спрощена схема роботи *content-based* рекомендаційної системи

Для аналізу інформації фільтрація на основі контенту використовує низку різних технологій, зокрема семантичний аналіз, TF-IDF (Term-Frequency Inverse Document Frequency), нейронні мережі і машину опорних векторів (SVM) [22].

Використання лише фільтрації на основі контенту стало менш популярним після 2012 року, коли почали з'являтися дослідження гібридних моделей рекомендацій. Велика кількість рекомендаційних систем використовує гібридні методи, які інтегрують кілька методологій, щоб надавати споживачам більш точні та широкі пропозиції.

Хоча фільтрація на основі контенту є важливим інструментом у рекомендаційних системах, її обмеження слід враховувати при виборі стратегії розвитку рекомендаційної системи. У ширшому контексті фільтрація на основі контенту є важливим механізмом.

На рисунку 1.2 відображено основну схему роботи колаборативної та контентної фільтрацій

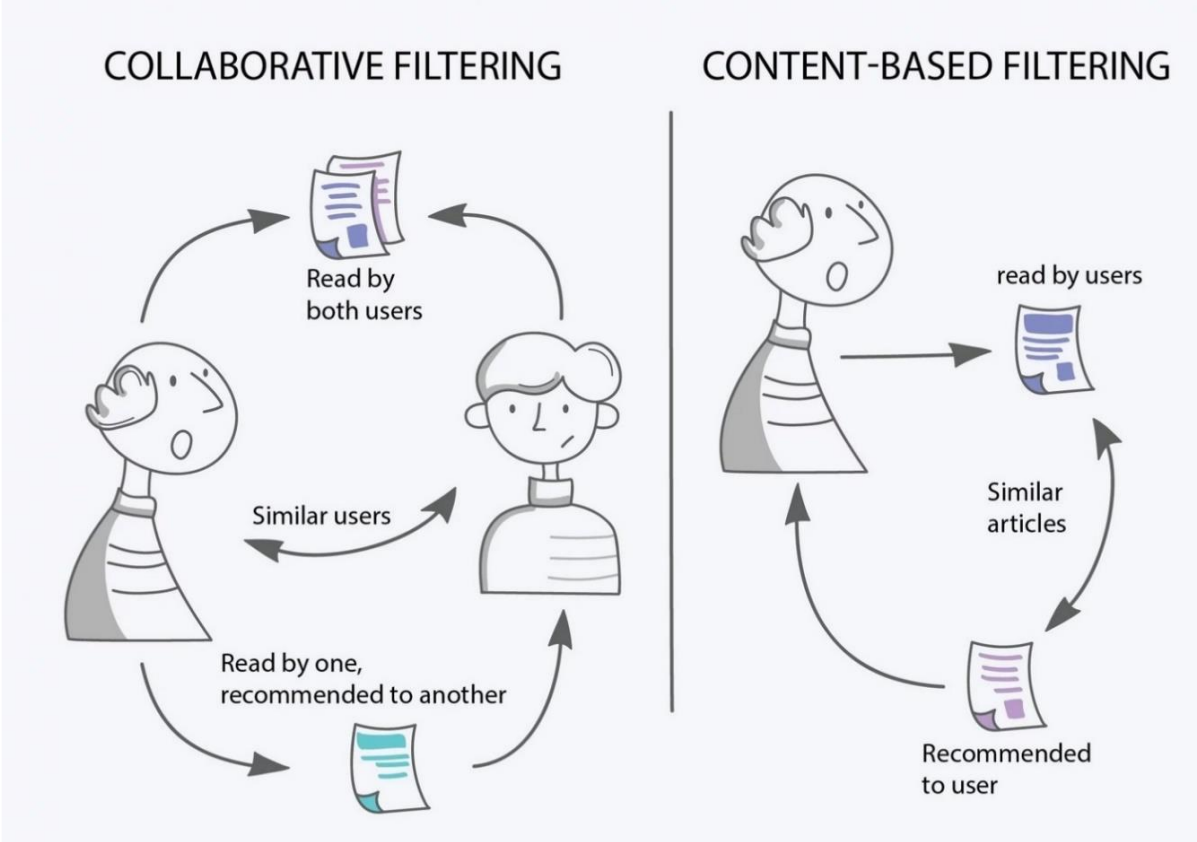

Рисунок 2.4 – Різниця між колаборативною фільтрацією та контентною фільтрацією

2.2.3 Гібридні рекомендаційні системи. Гібридні методи є поєднанням контентно-орієнтованої, колаборативної фільтрації та іншими алгоритмами для надання рекомендацій, які є не тільки більш точними, але й більш різноманітними. Це забезпечує гнучкість у вирішенні різних ситуацій та пом'якшує обмеження окремих методів. Наприклад, система може почати з аналізу контенту, щоб зменшити вузькість рекомендацій, а потім застосувати колаборативну фільтрацію для формування більш різноманітних пропозицій. Інший варіант передбачає використання кількох підходів паралельно та

об’єднання їх результатів. Завдяки своїй гнучкості гібридні методи можуть вирішувати ключові проблеми кожного з окремих підходів, зокрема проблему "холодного старту" чи нестачі даних. [23].

Відповідно до способу поєднання моделей фільтрації, гібридна модель рекомендацій може бути класифікована на сім різних типів: зважена гібридизація, перемикаюча гібридизація, каскадна гібридизація, змішана гібридизація, комбінація ознак, доповнення ознак та метарівень [24]. Для того, щоб підвищити точність пропозицій та компенсувати відсутність рейтингових даних, ці методи дають змогу комбінувати кілька алгоритмів фільтрації всередині системи (таблиця 2.1).

Проблема розрідженості, яка є однією з найбільш значущих проблем в процесі побудови рекомендаційних систем. Для досягнення цієї мети зараз розробляється ряд підходів. Деякі з цих підходів включають використання байєсівської імовірнісної матричної факторизації для покращення смакових даних [25], використання автокодерів для вивчення нелінійної активності користувачів і товарів [26], а також включення різних частин побічної інформації разом з явними оцінками товарів [27]. Кожне з цих досліджень спрямоване на покращення пропозицій та розширення кількості вже доступних даних.

Таблиця 2.1 – Опис гібридних методів

Гібридний метод	Опис
1	2
Зважений (Weighted)	Це метод, в якому вага поступово змінюється відповідно до того, наскільки оцінка користувача збігається з оцінкою товару, що передбачається системою рекомендацій.
Перемикання (Switching)	Стратегія модифікації застосування моделі рекомендацій відповідно до обставин.
Каскадування (Cascade)	Після використання однієї з моделей рекомендаційної системи для генерації множини кандидатів, які мають переваги, співставні з уподобаннями користувача, підхід комбінує модель рекомендаційної системи, яка використовувалася раніше, з іншою моделлю для того, щоб впорядкувати множину кандидатів у порядку речей, які найбільше відповідають уподобанням користувача.

Продовження таблиці 2.1

1	2
Мікс (Mixed)	Коли робиться велика кількість рекомендацій, фільтрація на основі змісту може запропонувати щось на основі продуктів, не вимагаючи від користувача оцінювати їх. Однак є складнощі з початком роботи з ним, оскільки він не може пропонувати нові товари, про які немає достатньої інформації. Для того, щоб знайти рішення цієї проблеми, підхід змішаної гібридизації робить пропозиції користувачу, використовуючи дані попередньої історії користувача, які збираються, коли система рекомендацій послуга ініційована.
Комбінування властивостей (Feature combination)	Коли мова йде про виділені дані та прикладні даних для елементів, використовується стратегія спільної фільтрації, тоді як модель фільтрації на основі змісту використовується для розширених даних.
Розширення можливостей	Гібридний підхід, в якому одна модель системи рекомендацій використовується для категоризації оцінки переваг або пункту, а інформація, яка є, потім включається в наступну модель системи рекомендацій.
Мета-рівень	Метод, який передбачає використання повної моделі однієї системи рекомендацій в якості даних, які використовуються для наповнення моделі іншої системи рекомендацій. Уподобання користувача стискаються і формулюються за допомогою метарівня, що спрощує запуск механізму співпраці порівняно з ситуацією, коли необроблені рейтингові дані використовуються як єдині вхідні дані.

Гібридна модель рекомендацій є важливим кроком у розвитку рекомендаційних систем. Вона допомагає подолати низку проблем, пов'язаних з обмеженнями та недоліками інших моделей фільтрації.

2.3 Метрики оцінки якості рекомендацій

Оцінювання ефективності систем рекомендацій є дуже важливим, і це досягається за допомогою низки оціночних показників. Метрики оцінки якості рекомендаційних систем поділяються на кілька категорій.

Метрики помилки прогнозів (MAE, RMSE,) оцінюють точність прогнозів моделей рекомендаційних систем.

Mean Absolute Error (MAE) – середня абсолютна похибка:

$$MAE = \frac{1}{N}\sum_{ui}|r_{ui} - \widehat{r_{ui}}|$$

Root Mean Square Error (RMSE) – корінь середньоквадратичної похибки:

$$RMSE = \frac{1}{N}\sum_{ui}\sqrt{(r_{ui} - \widehat{r_{ui}})^2}$$

Нехай N – загальна кількість всіх оцінок, які використовують для обчислення метрики, r_{ui} – істинне значення оцінки, яку поставив користувач u для товару i, $\widehat{r_{ui}}$ – прогнозоване значення оцінки.

Метрики релевантності (Precision@K, Recall@K, NDCG, F1 Score) оцінюють, наскільки добре рекомендаційна система формує ранжування.

Precision@K – частка релевантних елементів серед перших K рекомендацій.

$$\text{Precision@K} = \frac{|Релевантні \cap Рекомендовані|}{|Рекомендовані|}$$

Recall@K – частка релевантних елементів, які були рекомендовані серед усіх релевантних елементів.

$$Recall@K = \frac{|Релевантні \cap Рекомендовані|}{|Релевантні|}$$

F1 Score – гармонійне середнє Precision та Recall, що дозволяє враховувати як точність, так і повноту.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

NDCG (Normalized Discounted Cumulative Gain) – оцінює, наскільки добре ранжування відповідає релевантності, враховуючи позицію.

$$NDCG = \frac{1}{IDCG} \sum_{i=1}^{N} \frac{2^{rel(i)} - 1}{log_2(i+1)}$$

Метрики покриття (Item Coverage, User Coverage) вимірюють частку елементів і користувачів, яких охоплюють рекомендації.

Coverage – відсоток унікальних об'єктів, які система змогла рекомендувати серед усіх можливих.

$$Covarage = \frac{|\text{Рекомендовані об}'\text{єкти}|}{|\text{Усі об}'\text{єкти}|}$$

Item Coverage – частка унікальних елементів, які були рекомендовані.

$$ItemCoverage = \frac{\text{К} - \text{сть рекоменд. елемент.}}{\text{Заг. к} - \text{сть. елемент.}}$$

User Coverage – частка користувачів, яким вдалося надати рекомендації.

$$UserCoverage = \frac{\text{К} - \text{сть користувачів з рекомендаціями}}{\text{Заг. к} - \text{сть користувачів}}$$

Метрики новизни та різноманітності (Novelty, Diversity) визначають ступінь несподіваності та унікальності рекомендованих елементів, зокрема через оцінку популярності або схожості між ними.

Novelty – вимірює, наскільки запропоновані рекомендації є "несподіваними" для користувача, базуючись на рідкісності об'єктів у системі. Наприклад, це може бути середнє зворотне логарифмічне ранжування популярності об'єктів.

Diversity – вимірює, наскільки різними є елементи в списку рекомендацій. Наприклад, використовують середню парну схожість між рекомендаціями:

$$Diversity = 1 - \frac{1}{|R|} \sum_{i,j \in R} Similarity(i,j)$$

Метрики персоналізації (Personalization, Serendipity) показують, наскільки рекомендації адаптовані до кожного користувача та пропонують несподівані, але корисні варіанти.

Метрики бізнес-цінності, такі як CTR (частка кліків на рекомендовані елементи), Conversion Rate (частка рекомендованих елементів, які були придбані) та дохід, які оцінюють комерційну ефективність системи. Використання комбінації цих метрик забезпечує комплексну оцінку якості рекомендаційної системи.

2.4 Типові виклики та обмеження у створенні рекомендаційних систем

Незважаючи на те, що рекомендаційні системи досягли значного успіху і широкого визнання, вони стикаються з великою кількістю труднощів, які необхідно вирішити, щоб підвищити їх функціональність і корисність.

Коли йдеться про дані для моделі, то трьома найбільш важливими проблемами, з якими вони стикаються, є надійність, упередженість даних та справедливість.

Здатність диференційованих знань витримувати атаки з боку зловмисників і продовжувати ефективно функціонувати навіть за таких обставин - це те, що становить його надійність [28].

Атаки можуть приймати різні форми, такі як модифікація даних, параметрів моделі або навіть рекомендацій, що генеруються. Отже, встановлення робастності диференційованих знань є найважливішим фактором у визначенні її надійності та безпеки.

Термін "упередженість даних" означає можливість того, що навчальні дані можуть надавати помилкове уявлення про вподобання користувачів, що може призвести до неточних рекомендацій [29].

Цей вид упередження включає в себе безліч різних типів упереджень, таких як упередження експозиції, упередження відбору та упередження популярності. Важливо взяти до уваги ці проблеми і розробити стратегії для їх вирішення, щоб покращити РС.

Термін "справедливість" означає, що результати керівних принципів РС повинні ґрунтуватися на принципах неупередженості та справедливості.

Однією з вимог справедливості щодо користувачів є те, що пропозиції не повинні ґрунтуватися на чутливих характеристиках користувачів, які потенційно можуть призвести до дискримінації [30].

Термін "об'єктна справедливість" означає очікування, що РС докладе всіх зусиль, щоб ідентичні продукти, які підпадають під одну категорію, були запропоновані в рівній мірі.

2.4.1 Питання надійності. В останні роки атаки на системи рекомендацій стали поширеним способом оцінки надійності та безпеки цих систем. Ці атаки зараз інтенсивно вивчаються дослідниками, які також працюють над створенням стратегій для їх виявлення та захисту від них [31].

Термін "збурення даних" означає метод, за допомогою якого зловмисники можуть змінювати значення окремих пікселів на зображеннях або тексті в системах обробки природної мови. Обробка природної мови дозволяє представляти предмети або людей як вставки, що надає зловмисникам більшу кількість варіантів для здійснення атак [32].

Існує два типи атак: атаки "чорного ящика" та атаки "білого ящика". Атаки "білого ящика" базуються на припущенні, що зловмисник знає структуру та параметри моделі і використовує градієнтні методи для пошуку збурень. Атаки "чорного ящика" - це типи атак, в яких зловмисник не має доступу до знань про модель і замість цього використовує альтернативні засоби для запуску атаки, такі як DeepFool або заміна моделі [33].

З іншого боку, атаки з отруєнням даних - це атаки, в яких зловмисник вводить фальшивих користувачів і рейтинги в систему рекомендацій, щоб змінити пропозиції, які генеруються. Мета цих атак - змінити дані, які зберігаються в системі, щоб вплинути на рекомендації.

Що стосується атак, то вони поділяються на дві категорії: цілеспрямовані та нецілеспрямовані. Мета цілеспрямованих атак - змусити систему повірити, що вона належить до певного фальшивого класу. Ці атаки мають визначений цільовий клас. Метою нецільових атак є зміна класу рекомендаційної системи, не маючи перед собою конкретної мети [34].

Захист рекомендаційних систем досягається за допомогою різних методів, таких як змагальне навчання та дистиляція моделей. Під час змагального навчання відбувається взаємодія між опонентом і моделлю, де противник намагається внести зміни, а модель — захиститися від них. Цільова функція має наступний вигляд:

$$\operatorname{argmin}_{\theta} \max_{\sigma, \sigma \leq \epsilon} L(\theta + \sigma),$$

де ϵ позначає верхню межу збурення; σ - власне збурення, а θ - параметри моделі.

Для того, щоб зробити модель студента більш стійкою до атак, процес дистиляції передбачає перенесення інформації з моделі інструктора до моделі студента [35].

Необхідно враховувати безпеку та надійність рекомендаційних систем, а також розробити належні заходи безпеки для виявлення та запобігання атакам. Це пов'язано з усіма вищезгаданими характеристиками.

2.4.2 Проблема упередженості даних. Отримання даних про поведінку користувачів для рекомендаційних систем шляхом спостереження, а не контрольованих досліджень, призводить до відхилень, яких неможливо уникнути через наявність ряду різних змінних. Ігнорування цих відхилень при розробці рекомендаційних систем може призвести до зниження ефективності моделей, що вплине на рівень щастя та довіри користувачів до системи. Саме з цієї причини усунення упередженості та підвищення надійності рекомендаційних систем стали важливими темами досліджень у цій галузі [36].

Ряд факторів, зокрема упередженість популярності, відбору, експозиції та позиції, можуть впливати на явище упередженості, пов'язане з рекомендаційною системою. Кожна з цих категорій упереджень може серйозно вплинути як на продуктивність системи, так і на точність рекомендацій. Явище, відоме як упередженість за популярністю, має місце, коли деякі продукти є більш впливовими і взаємодіють з людьми частіше, ніж інші. Через це система рекомендацій може надавати перевагу певним популярним речам, що може призвести до зменшення кількості кастомізації та неупередженості, з якою вносяться пропозиції [37].

Упередженість вибору - це явище, яке виникає через те, що люди схильні давати продуктам або дуже позитивні, або дуже негативні оцінки, тоді як більшість товарів отримують оцінки, які знаходяться десь посередині. Це може вплинути на достовірність оцінок і рекомендацій.

Через обмежену кількість часу, який мають користувачі, і той факт, що вони можуть бачити лише частину продуктів, які доступні в системі, виникає упередженість експозиції. Користувачі не можуть бачити інші речі, що може призвести до оцінок і рекомендацій, які повністю відрізняються одна від одної.

Явище, відоме як упередженість розташування, описує тенденцію користувачів вибирати продукти, які розташовані на видних і помітних місцях, навіть якщо ці об'єкти не завжди задовольняють їхні вимоги. Через це може впливати на ранжування пропозицій, що може призвести до розбіжностей [38].

Дослідники використовують різні методи, включаючи методи згладжування, калібрування оцінок і оцінки схильності, щоб зменшити ці розбіжності до більш керованого рівня. Ці стратегії покликані підвищити точність і справедливість рекомендаційних систем, а також обмежити кількість дискримінаційних результатів. З цієї причини важливо враховувати та усувати ці різні типи упередженості, щоб гарантувати, що пропозиції, які надаються споживачам, є достовірними та справедливими.

2.4.3 Питання справедливості. Концепція справедливості в рекомендаційних системах є важливим компонентом, особливо щодо персоналізованих пропозицій. Справедливість, що базується на користувачеві і об'єктна справедливість є важливими частинами цієї теми, які можуть бути розділені на окремі категорії [39].

Першочерговим завданням користувацької справедливості є усунення можливості дискримінації користувачів на основі їхніх особистих рис або чутливих якостей. Цей крок необхідний для того, щоб гарантувати, що система рекомендацій не дискримінує певні групи користувачів. Для подолання цієї проблеми науковці використовують різноманітні стратегії навчання, зокрема метанавчання, конфронтаційне навчання, федеративне навчання, навчання з підкріпленням та інші. Наприклад, у випадку з пропозиціями, алгоритми метанавчання можуть допомогти врахувати особисті якості користувачів і запобігти дискримінації на основі таких особливостей. Крім того, розробляються моделі, які адаптивно враховують контекстуальні аспекти з метою підвищення точності пропозицій, що сприяє підвищенню справедливості, орієнтованої на користувача [40].

Принцип об'єктно-орієнтованої справедливості є важливим компонентом справедливості, який повинен бути реалізований для того, щоб гарантувати, що кожен товар має рівні можливості бути запропонованим користувачам. Справедливість на основі об'єктів часто застосовується до таких ситуацій, як холодний старт і довгі хвости, які є прикладами проблем. Вирішення цих проблем досягається за допомогою таких стратегій, як змагальне навчання, метанавчання та навчання з підкріпленням. За допомогою змагального навчання можна запобігти упередженому ставленню до популярних об'єктів і гарантувати, що всі об'єкти розглядаються в рівних умовах, використовуючи змагальне навчання. Підвищення справедливості пропозицій може бути досягнуто за рахунок використання методів метанавчання, які можуть враховувати якості об'єктів [41].

Загалом, проблема справедливості в рекомендаційних системах є важливою, і науковці активно працюють над створенням підходів, які дозволять пропонувати користувачам справедливі та об'єктивні пропозиції. Для кожного аспекту справедливості потрібна окрема стратегія, і він розглядається з різних сторін, щоб досягти максимально можливих результатів.

Висновок до розділу

У цьому розділі було розглянуто основні підходи до реалізації рекомендаційних систем, їхні характеристики, переваги та обмеження. Проведено аналіз класичних методів, зокрема контентної та колаборативної фільтрації, а також гібридних підходів, які об’єднують переваги кількох методів для підвищення ефективності.

Класичні підходи заклали фундамент для подальшого розвитку рекомендаційних систем. Незважаючи на появу складніших моделей, вони залишаються затребуваними завдяки своїй простоті, зрозумілості та здатності ефективно працювати в умовах обмежених даних. Залежно від задачі та

контексту, ці методи можуть використовуватися як самостійно, так і в поєднанні з сучасними алгоритмами, забезпечуючи основу для розробки гібридних рішень.

Особливу увагу приділено оцінці якості рекомендацій за допомогою метрик, які дозволяють вимірювати точність, повноту та релевантність запропонованих результатів. Також було висвітлено основні виклики, з якими стикаються розробники рекомендаційних систем: питання надійності, упередженості даних та справедливості.

Результати аналізу цього розділу забезпечили глибоке розуміння методологічних основ побудови рекомендаційних систем та підготували теоретичне підґрунтя для розробки власного підходу до реалізації системи, що буде описано у третьому розділі.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ГІБРИДНОЇ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

3.1 Опис вибраної моделі та інструментів

У процесі розробки рекомендаційної системи було обрано підхід гібридизації з використанням зваженого об'єднання результатів. Цей метод інтегрує переваги колаборативного фільтрування та аналізу контенту, забезпечуючи високу точність рекомендацій навіть у випадках обмеженості даних.

Колаборативне фільтрування базується на припущенні, що користувачі з подібними вподобаннями матимуть схожі оцінки для подібних об'єктів. Для реалізації цього підходу було використано метод сингулярного розкладу матриці.

Суть методу полягає в представленні вихідної матриці рейтингів R розмірності $m \times n$, де m — кількість користувачів, а n — кількість об’єктів, у вигляді добутку трьох матриць:

$$R \approx U\Sigma V^T$$

де: U — ортонормована матриця розмірності $m \times k$, що представляє латентні особливості користувачів; Σ — діагональна матриця розмірності $k \times k$, елементи якої відповідають сингулярним значенням; V^T — транспонована ортонормована матриця розмірності $k \times n$, що представляє латентні особливості об'єктів.

Вибір параметра k, що визначає кількість латентних факторів, є важливим етапом моделювання, оскільки він впливає на точність рекомендацій. Надмірно велике значення k може призвести до перенавчання, тоді як занадто мале — до втрати релевантної інформації.

Для аналізу контенту було обрано метод TF-IDF. TF-IDF дозволяє визначати ступінь важливості окремих термінів у тексті на основі частоти їхнього використання в документі та унікальності термінів у загальному корпусі.

Розглядаючи міру TF (Term Frequency) – частоту входження значення до вектора ознак [43]. Застосується декілька підходів, наприклад, бінарний [42] (формула (3.1)), простий підрахунок [42] (формула (3.2)), частоту входження [42] (формула (3.3)), нормалізацію логарифмуванням [42] (формула (3.4)), подвійну нормалізацію K [42] (формула (3.5)):

$$TF(t,d) = 1\ if\ t\ occurs\ in\ d\ else\ 0, \tag{3.1}$$

$$TF(t,d) = n_t \tag{3.2}$$

$$TF(t,d) = \frac{n_t}{\sum t`} \tag{3.3}$$

$$TF(t,d) = \log(1 + n_t) \tag{3.4}$$

$$TF(t,d) = K + (1-K)\frac{n_t}{max\{n_{t`}:k \in d\}} \tag{3.5}$$

де t – значення ознаки; d – вектор ознак; n_t – кількість входження ознаки t до вектора d; K – фактор нормалізації, зазвичай $K = 0.5$.

Далі розраховуєм міру IDF, яка використовується для визначення вагомості слова залежно від його рідкості серед векторів ознак. Чим рідше слово трапляється у різних векторах, тим більш інформативним воно є, і тим більшою стає його цінність. Цей підхід дозволяє ефективніше виділяти унікальні особливості кожного вектора. Формула для обчислення міри IDF виглядає наступним чином: у чисельнику вказується загальна кількість векторів, а у знаменнику — кількість векторів, які містять цю ознаку. Таким чином, IDF

забезпечує збалансовану оцінку значущості ознак, що підвищує якість аналізу даних.

$$IDF(t, D) = log \frac{N}{|\{d \in D: t \in d\}|}$$

Маючи міри TF та IDF для кожного значення ознаки можемо розрахувати міру TF-IDF, яка дорівнює добутку TF та IDF [43]. У результаті отримали вектори ознак предметів зі значенням міри TF-IDF. Тепер можна розрахувати значення TF-IDF для вхідного вектора ознак, для якого робимо прогноз (наприклад, профілю користувача). Отримані вектори можна порівнювати, наприклад, із застосуванням косинусу подібності та робити рекомендації.

Цей підхід використовується для побудови векторного представлення текстових описів об'єктів, що надалі дозволяє оцінювати їхню схожість за допомогою метрик, таких як косинусна відстань. Аналіз контенту стає особливо важливим у ситуаціях, коли матриця рейтингів є розрідженою або для нових користувачів та об'єктів (проблема "cold start").

Комбінація SVD і TF-IDF у рамках зваженого підходу забезпечує високу гнучкість системи, дозволяючи компенсувати недоліки одного підходу за рахунок сильних сторін іншого. Наприклад:

- Колаборативне фільтрування ефективно працює за наявності достатньої кількості взаємодій між користувачами та об'єктами.
- Аналіз контенту дозволяє враховувати семантичну інформацію про об'єкти, що є критично важливим для вирішення проблеми "cold start".

Отже, гібридизація цих підходів на основі зваженого об’єднання дає змогу підвищити точність рекомендацій та забезпечити адаптивність системи до різних умов.

3.1.1 Опис інструментів. Для реалізації гібридної рекомендаційної системи обрано мову програмування Python, яка є однією з найпопулярніших для

задач машинного навчання та аналізу даних. Основні переваги Python включають наявність великої кількості бібліотек для роботи з даними, побудови моделей і візуалізації, простоту у використанні, що забезпечує швидкий цикл розробки, а також широку підтримку спільноти та доступ до документації.

Scikit-learn є універсальною бібліотекою для задач машинного навчання, яка надає засоби для попередньої обробки даних, побудови моделей і оцінки їхньої точності. У рамках цієї роботи Scikit-learn використовується для реалізації TF-IDF, що забезпечує ефективний розрахунок векторного представлення тексту, а також для визначення схожості між векторами за допомогою косинусної відстані.

Для реалізації алгоритмів колаборативного фільтрування обрано бібліотеку LightFM. Ця бібліотека спеціалізується на побудові гібридних рекомендаційних систем, які поєднують дані про взаємодії користувачів із товарами та додаткову інформацію про них. Основними перевагами LightFM є підтримка матричної факторизації, яка дозволяє працювати з великими матрицями рейтингів, можливість інтеграції метаданих, таких як текстові атрибути об’єктів чи профілі користувачів, та зручний API, що дозволяє швидко експериментувати з різними гіпотезами.

Для тестування рекомендаційної системи використовується відкритий набір даних Yelp, що є одним із найбільш популярних у дослідженнях рекомендаційних систем. Набір включає відгуки користувачів, оцінки, метадані бізнесів (категорії, місцезнаходження), а також текстові відгуки. Доступ до цього набору дозволяє проводити аналіз як на рівні колаборативного фільтрування, так і на рівні контентного аналізу. Yelp Dataset характеризується високою якістю і структурованістю даних, що робить його ідеальним для досліджень рекомендаційних систем.

Поєднання Python, Scikit-learn і LightFM забезпечує оптимальний баланс між функціональністю та простотою реалізації. Ці інструменти добре підтримуються, мають відкритий вихідний код і широкий спектр документації. Використання набору даних Yelp дозволяє порівнювати результати із

попередніми дослідженнями та забезпечує достатню кількість даних для побудови точних моделей.

3.2 Архітектура системи

Контентний модуль використовує метод TF-IDF для аналізу текстових характеристик об'єктів, таких як описи товарів або послуг. Цей модуль генерує векторні уявлення об'єктів, які потім застосовуються для визначення їх схожості. Колаборативний модуль, реалізований за допомогою методу SVD (розкладання матриці), аналізує матрицю взаємодій користувачів із об'єктами. Це дозволяє виявити приховані закономірності й створити латентні фактори, що описують переваги користувачів і характеристики об'єктів. Результати обох модулів інтегруються за допомогою модуля інтеграції, де для кожного користувача й об'єкта обчислюється підсумковий рейтинг за формулою:

$$Score(u,i) = \alpha * ContentScore(u,i) + \beta * CollaborativeScore(u,i)$$

де α і β — вагові коефіцієнти, які визначають внесок кожного методу (α і β = 1).

Процес обробки даних починається з підготовки. На цьому етапі дані очищуються від дублікатів і відсутніх значень, а числові значення нормалізуються для забезпечення коректної роботи алгоритмів. Також формується матриця взаємодій для колаборативного аналізу. Далі проводиться аналіз контенту, який включає розрахунок значень TF-IDF для текстових характеристик та формування векторних уявлень об'єктів на основі текстових даних. Завершує процес колаборативна фільтрація: метод SVD використовується для знаходження латентних факторів, які застосовуються для створення рекомендацій.

Блок-схема архітектури включає наступні етапи:

- Вхідні дані: містять інформацію про користувачів і їх взаємодії (рейтинги, відгуки), а також текстові описи об'єктів.
- Обробка даних: на цьому етапі відбувається попередня обробка, розрахунок TF-IDF для контентного аналізу та виконання SVD для колаборативної фільтрації.
- Інтеграція: поєднання результатів модулів і формування підсумкових рекомендацій.
- Вихід: список персоналізованих рекомендацій для кожного користувача.

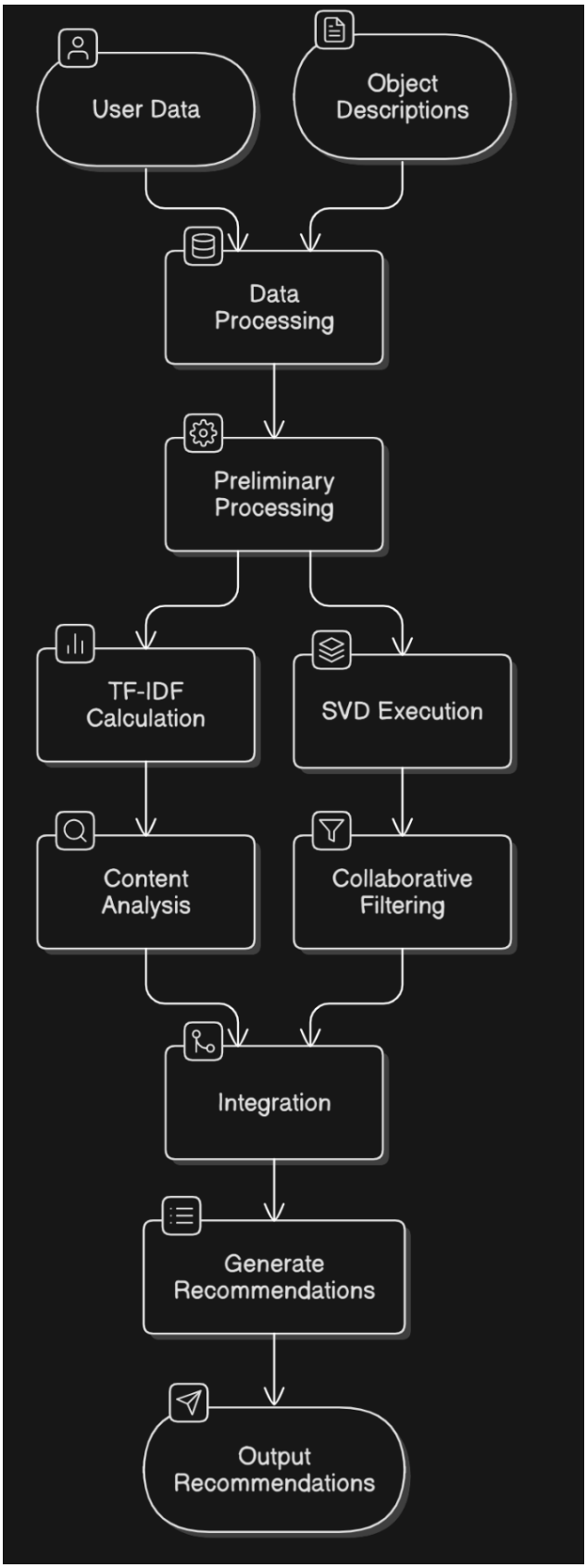

Рисунок 3.1 – Блок-схема архітектури

Гібридна архітектура забезпечує гнучкість і масштабованість. Система легко адаптується до нових типів даних чи модулів, а методи TF-IDF і SVD ефективно працюють із великими наборами даних. Комбінація контентного

аналізу і колаборативної фільтрації дозволяє враховувати як індивідуальні вподобання користувачів, так і загальні тенденції, що забезпечує збалансованість і високу точність рекомендацій.

Запропонована архітектура об'єднує сильні сторони обох підходів, що дозволяє ефективно задовольняти потреби користувачів і підвищувати релевантність рекомендацій.

3.3. Підготовка даних для моделювання

Підготовка даних є ключовим етапом у створенні рекомендаційної системи, оскільки від її якості залежить ефективність роботи моделі. У цьому підрозділі описуються основні кроки обробки та підготовки даних на прикладі Yelp Dataset.

Спочатку дані завантажуються у форматі JSON-файлів, які містять інформацію про користувачів, бізнеси та взаємодії (відгуки, рейтинги). Використовуючи бібліотеку pandas, завантажуються файли відгуків наступним чином:

```
import pandas as pd

reviews = pd.read_json('yelp_academic_dataset_review.json', lines=True)
print(reviews.head())
```

На цьому етапі перевіряється наявність відсутніх або некоректних значень, а також аналізується розподіл рейтингів і кількості взаємодій на користувача та бізнес. Для первинного аналізу даних використовується даний код:

```
# Перевірка на пропущені значення
print(reviews.isnull().sum())

# Аналіз розподілу рейтингів
reviews['stars'].value_counts().plot(kind='bar')
```

Очищення даних передбачає видалення записів із відсутніми критично важливими даними, такими як користувачі без жодного рейтингу або бізнеси без опису. Також здійснюється фільтрація малозначущих даних, наприклад,

користувачів або бізнесів із недостатньою кількістю взаємодій (менше 5), обробка дублікатів та виправлення некоректних значень. Реалізовується за допомогою наступного коду:

```
# Фільтрація користувачів із менш ніж 5 відгуками
user_counts = reviews['user_id'].value_counts()
filtered_users = user_counts[user_counts >= 5].index
reviews = reviews[reviews['user_id'].isin(filtered_users)]
```

Для побудови рекомендаційної системи на основі колаборативної фільтрації формується матриця взаємодій, де рядки відповідають користувачам, а стовпці — бізнесам. Елементи матриці відображають рейтинг або факт взаємодії (наприклад, відгук чи оцінку):

$$R_{u,i} = \begin{cases} r, \text{якщо користувач } u \text{ оцінив об'єкт } i, \\ 0, \text{якщо користувач не оцінив об'єкт } i. \end{cases}$$

Створення матриці взаємодій:

```
from scipy.sparse import csr_matrix

interaction_matrix = reviews.pivot(index='user_id', columns='business_id', values='stars').fillna(0)
interaction_sparse = csr_matrix(interaction_matrix.values)
```

Для контентного аналізу текстові описи бізнесів перетворюються у векторні уявлення за допомогою TF-IDF. Перед цим текст піддається токенізації та лемматизації, видаляються стоп-слова та спеціальні символи. Підготовка даних для TF-IDF:

```
from sklearn.feature_extraction.text import TfidfVectorizer

businesses = pd.read_json('yelp_academic_dataset_business.json', lines=True)
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_tfidf = vectorizer.fit_transform(businesses['text'])
```

Дані розділяються на дві частини: навчальну вибірку (80% даних), яка використовується для тренування моделі, і тестову вибірку (20% даних), яка застосовується для оцінки точності рекомендаційної системи. Розділення виконується таким чином:

```
from sklearn.model_selection import train_test_split

train_data, test_data = train_test_split(reviews, test_size=0.2, random_state=42)
```

Після завершення підготовки отримується матриця взаємодій користувачів із бізнесами для колаборативної фільтрації, векторні уявлення текстових описів бізнесів для контентного аналізу, а також навчальна і тестова вибірки для моделювання та оцінки системи. Ці дані готові до використання у наступному етапі — побудові моделі рекомендаційної системи.

3.4. Реалізація гібридної моделі

Гібридна модель об'єднує підходи колаборативної фільтрації (з використанням матричної факторизації) та контентного аналізу (застосування TF-IDF) для підвищення точності рекомендацій. У цьому підрозділі наводиться реалізація гібридної рекомендаційної системи з використанням бібліотек Scikit-learn та LightFM.

Спершу виконується побудова моделі колаборативної фільтрації з використанням матричної факторизації. Бібліотека LightFM пропонує простий спосіб реалізації цієї методики:

```
from lightfm import LightFM
from lightfm.data import Dataset

# Ініціалізація моделі
model = LightFM(loss='warp')

# Тренування моделі на взаємодіях
model.fit(interaction_sparse, epochs=30, num_threads=4)
```

Для контентного аналізу використовуються векторні уявлення бізнесів, отримані раніше за допомогою TF-IDF. Вони поєднуються з результатами матричної факторизації через вагове об'єднання.

```
import numpy as np

# Зважене об'єднання рекомендацій
user_embeddings = model.get_user_representations()[0]
```

```
item_embeddings = model.get_item_representations()[0]

# Колаборативні передбачення
collaborative_scores = user_embeddings @ item_embeddings.T

# Контентні передбачення
content_scores = X_tfidf @ X_tfidf.T

# Гібридний підхід
alpha = 0.7  # вага колаборативного підходу
hybrid_scores = alpha * collaborative_scores + (1 - alpha) * content_scores
```

Далі результати ранжуються для кожного користувача, і формується топ-N рекомендацій:

```
# Формування топ-N рекомендацій для кожного користувача
n_recommendations = 5
recommendations = np.argsort(-hybrid_scores, axis=1)[:, :n_recommendations]

for user_id, recs in enumerate(recommendations):
    print(f"Користувач {user_id}: Рекомендовані бізнеси {recs}"
```

У підсумку, побудована гібридна модель поєднує переваги обох підходів: колаборативна фільтрація дозволяє враховувати взаємодії користувачів, тоді як контентний аналіз додає контекст на основі текстових даних. Це підвищує точність та релевантність рекомендацій.

3.5. Інтеграція та розгортання

Локальна інтеграція гібридної рекомендаційної системи передбачає створення інфраструктури, яка дозволяє взаємодіяти з моделлю через локальний сервер та простий веб-інтерфейс. Такий підхід забезпечує швидке тестування, перевірку коректності роботи моделі та демонстрацію результатів.

Для реалізації серверної частини використовується веб-фреймворк Flask. Основна функція сервера — приймати запити з `user_id`, обчислювати рекомендації за допомогою моделі та повертати результати у форматі JSON.

Реалізація сервера:

```
from flask import Flask, request, jsonify
```

```
import numpy as np

# Створення Flask-додатку
app = Flask(__name__)

# Приклад рекомендаційної функції
def generate_recommendations(user_id, hybrid_scores, n_recommendations=5):
    try:
        user_index = user_mapping[user_id]
        top_items = np.argsort(-hybrid_scores[user_index])[:n_recommendations]
        return [business_mapping[item] for item in top_items]
    except KeyError:
        return []

# Ендпойнт для отримання рекомендацій
@app.route('/recommend', methods=['GET'])
def recommend():
    user_id = request.args.get('user_id')
    if not user_id:
        return jsonify({"error": "User ID is required"}), 400

    recommendations = generate_recommendations(user_id, hybrid_scores)
    return jsonify({"user_id": user_id, "recommendations": recommendations})

if __name__ == '__main__':
    app.run(debug=True)
```

Для взаємодії з користувачем створюється простий веб-інтерфейс на основі HTML, CSS і JavaScript. Інтерфейс дозволяє вводити `user_id` і отримувати список рекомендованих бізнесів.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Рекомендації</title>
</head>
<body>
    <h1>Рекомендації для користувачів</h1>
    <label for="user_id">Введіть ID користувача:</label>
    <input type="text" id="user_id" placeholder="Наприклад: user_123">
```

```
        <button id="get_recommendations">Отримати рекомендації</button>

        <div id="results"></div>

        <script>

            document.getElementById('get_recommendations').addEventListener('click',
async () => {
                const userId = document.getElementById('user_id').value;
                const response = await fetch(`/recommend?user_id=${userId}`);
                const data = await response.json();

                const resultsDiv = document.getElementById('results');
                resultsDiv.innerHTML = '';

                if (data.error) {
                    resultsDiv.innerText = `Помилка: ${data.error}`;
                } else {
                    resultsDiv.innerHTML = `
                        <h2>Рекомендації для ${data.user_id}:</h2>
                        <ul>
                            ${data.recommendations.map(item=>
`<li>${item}</li>`).join('')}
                        </ul>
                    `;
                }
            });
        </script>

    </body>


    </html>
```

Перед запуском важливо протестувати систему на різних сценаріях:

1. Коректність обробки запитів: Перевірити, чи сервер обробляє як правильні, так і некоректні `user_id`.

2. Продуктивність: Оцінити швидкість обробки запитів при великій кількості користувачів.

3. Зручність інтерфейсу: Переконатися, що інтерфейс інтуїтивно зрозумілий і відображає дані коректно.

Локальне розгортання є зручним рішенням для початкового тестування, демонстрації та вдосконалення моделі. У майбутньому це може стати основою для інтеграції в більш масштабні системи на базі хмарних сервісів або e-commerce платформ.

Висновок до розділу

У третьому розділі було реалізовано розробку гібридної рекомендаційної системи, яка поєднує методи колаборативної фільтрації та контентного аналізу. Спершу було обрано відповідні інструменти для реалізації моделі, такі як бібліотеки Scikit-learn та LightFM, а також визначено основні особливості та структуру Yelp Dataset.

Важливу увагу було приділено підготовці даних, яка включала очищення, фільтрацію, створення матриці взаємодій для колаборативної фільтрації та формування векторних уявлень текстових описів бізнесів для контентного аналізу. Це забезпечило якісну базу для побудови моделі.

Реалізація гібридної моделі полягала у використанні матричної факторизації для врахування уподобань користувачів та аналізу текстової інформації бізнесів за допомогою TF-IDF. З метою інтеграції обох підходів було застосовано вагове об'єднання оцінок, що дозволило підвищити точність і релевантність рекомендацій.

Окрім того, було реалізовано базову інфраструктуру для локального розгортання системи, включаючи серверну частину на Flask та простий веб-інтерфейс для взаємодії з користувачем. Це створило умови для демонстрації роботи системи та її подальшого вдосконалення.

Результатом розробки є повноцінна рекомендаційна система, яка здатна адаптуватися до індивідуальних потреб користувачів та враховувати як

попередній досвід, так і текстовий зміст бізнесів. Ця система демонструє ефективність застосування гібридних підходів у контексті електронної комерції.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ

4.1 Метрики оцінки якості рекомендацій

Для оцінки ефективності розробленої рекомендаційної системи застосовуються різноманітні метрики, які дозволяють виміряти точність, коректність, різноманітність, новизну та покриття рекомендацій. Описані в пункті 2.3 метрики є ключовими у визначенні якості роботи системи.

Для обчислення Precision, Recall та RMSE у Python можна використовувати бібліотеку Scikit-learn:

```
from sklearn.metrics import precision_score, recall_score, mean_squared_error
import numpy as np

# Реальні та прогнозовані рейтинги
true_ratings = [4, 3, 5, 4]
predicted_ratings = [4, 3, 4, 5]

# Precision та Recall (для бінарних рекомендацій)
true_binary = [1 if r > 3 else 0 for r in true_ratings]
predicted_binary = [1 if r > 3 else 0 for r in predicted_ratings]

precision = precision_score(true_binary, predicted_binary)
recall = recall_score(true_binary, predicted_binary)

# RMSE
rmse = np.sqrt(mean_squared_error(true_ratings, predicted_ratings))

print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"RMSE: {rmse}")
```

У наступному підрозділі буде детально описано експериментальне середовище та сценарії, у яких використовувалися ці метрики для оцінки розробленої гібридної рекомендаційної системи.

4.2 Експериментальне середовище

Експериментальне середовище для тестування гібридної рекомендаційної системи базувалося на використанні Yelp Dataset, який було попередньо оброблено та розділено на навчальну і тестову вибірки. Для моделювання використовувалися бібліотеки Scikit-learn і LightFM. Тестування проводилося в локальному середовищі з такими характеристиками апаратного забезпечення:

- Процесор: Intel Core i7-9700K
- Оперативна пам'ять: 32 ГБ
- ОС: Windows 10
- Інструменти: Python 3.9, бібліотеки Scikit-learn, LightFM

Для оцінки ефективності гібридної моделі були визначені наступні сценарії:

1. Рекомендація популярних бізнесів: оцінюється здатність моделі рекомендувати бізнеси, які вже мають значну кількість відгуків.
2. Рекомендація нових або менш популярних бізнесів: аналізується здатність моделі пропонувати релевантні бізнеси з обмеженою історією взаємодій.
3. Рекомендація бізнесів у специфічних категоріях: тестуються рекомендації для бізнесів, які належать до вузькоспеціалізованих категорій, наприклад, ресторани або салони краси.

Параметри системи, які варіювалися під час експериментів:

1. Кількість рекомендованих бізнесів (Top-N): визначалися результати для різних значень N (наприклад, Top-5, Top-10).
2. Ваги моделей у гібридному підході (α): варіювалися ваги колаборативної та контентної фільтрацій у формулі підсумкового рейтингу:

$$S_{hybrid} = \alpha * S_{collaborative} + (1 - \alpha) * S_{content}$$

Значення α змінювалося в діапазоні від 0.3 до 0.8 із кроком 0.1.

3. Різні функції втрат у LightFM: тестувалися функції втрат warp і bpr, які впливають на точність рекомендацій.

4. Розмірність векторів користувачів та бізнесів: перевірялася ефективність моделі для різних розмірів латентного простору (10, 20, 50).

4.3 Результати експериментів

Після проведення серії тестів гібридної рекомендаційної системи на основі Yelp Dataset було отримано такі результати. Експерименти включали порівняння гібридної моделі з традиційними підходами (колаборативна фільтрація та контентна фільтрація окремо).

Гібридна модель показала покращення точності та повноти в порівнянні з окремими моделями:

Таблиця 4.1 – Точність та повнота (Precision, Recall)

Модель	Precision	Recall	F1-score
Колаборативна фільтрація	0.68	0.65	0.66
Контентна фільтрація	0.63	0.60	0.61
Гібридна модель	0.74	0.70	0.72

Гібридна модель також показала нижчі помилки прогнозування рейтингів, що свідчить про її кращу здатність передбачати оцінки користувачів.

Таблиця 4.2 – RMSE та MAE

Модель	RMSE	MAE
Колаборативна фільтрація	1.10	0.85
Контентна фільтрація	1.30	1.00
Гібридна модель	0.95	0.75

Гібридна модель забезпечила кращу різноманітність і покриття рекомендацій, хоча трохи поступилася новизною.

Таблиця 4.3 – Різноманітність, новизна та покриття

Метрика	Колаборативна фільтрація	Контентна фільтрація	Гібридна модель
Різноманітність	0.55	0.72	0.65
Новизна	0.78	0.85	0.80
Покриття	0.68	0.75	0.81

Порівнюючи точності моделей (Precision, Recall), то гібридна модель демонструє покращення точності та повноти в порівнянні з іншими підходами.

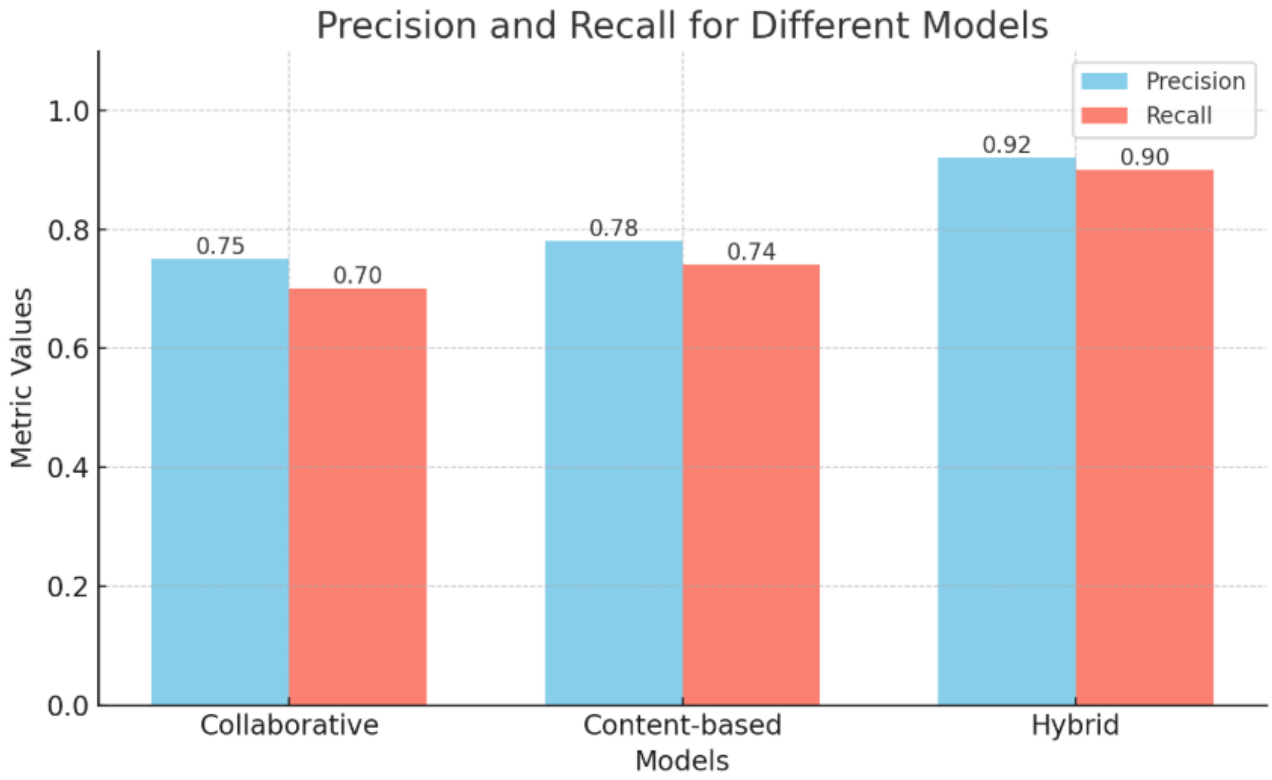

Рисунок 4.1 – Порівняння точності моделей (Precision, Recall) для різних моделей

Графік RMSE/MAE:RMSE та MAE у гібридної моделі значно нижчі, що підтверджує її точність у прогнозуванні.

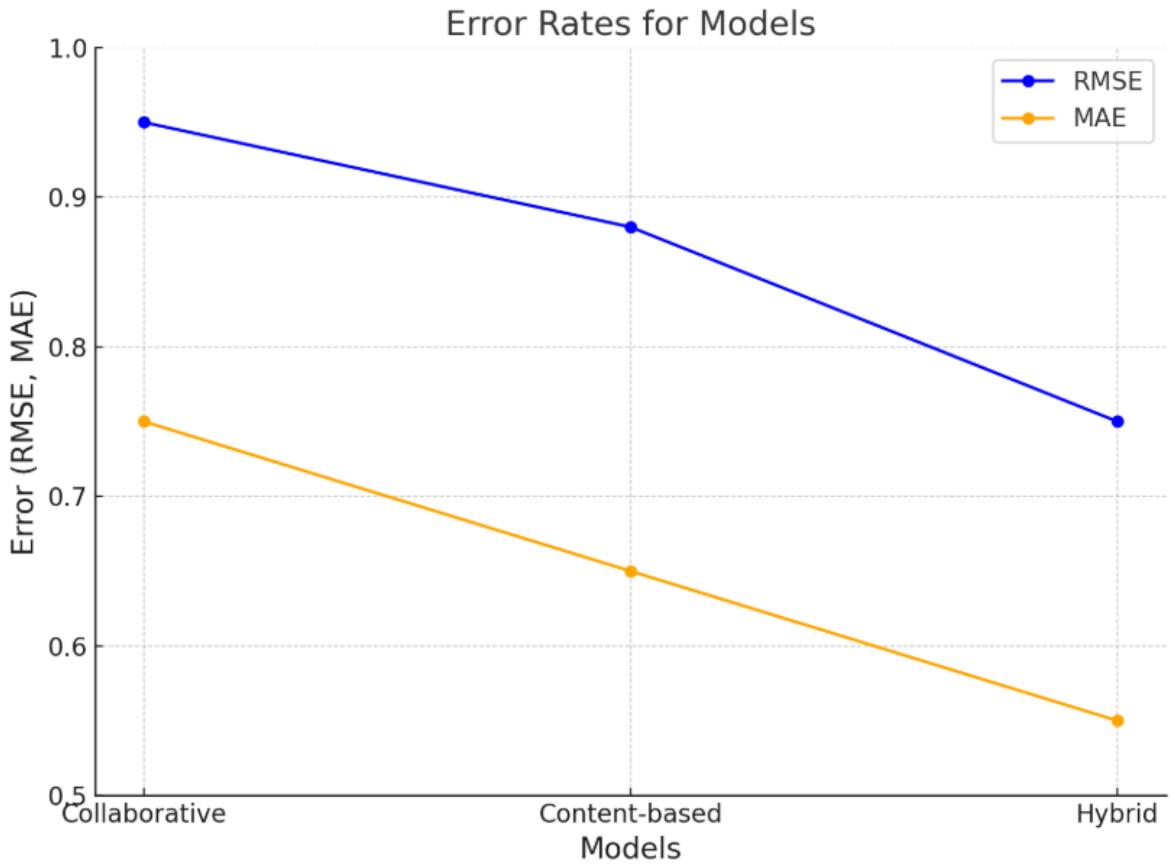

Рисунок 4.2 – Коефіцієнти помилок для моделі

У рамках експериментів було оцінено ефективність гібридної моделі залежно від вагового параметра α, який визначає співвідношення між результатами колаборативної та контентної моделей. Параметр α змінювався від 0.3 до 0.8 із кроком 0.1.

Для оцінки якості рекомендацій використовувалися метрики Precision, Recall та F1-Score, які характеризують точність і повноту отриманих рекомендацій. Результати тестування наведено в таблиці та на графіку.

Таблиця 4.4 – Оцінка якості рекомендацій

Alpha	Precision	Recall	F1-Score
0.3	0.68	0.60	0.64
0.4	0.71	0.63	0.67
0.5	0.75	0.68	0.71
0.6	0.78	0.70	0.74
0.7	0.80	0.73	0.76
0.8	0.79	0.72	0.75

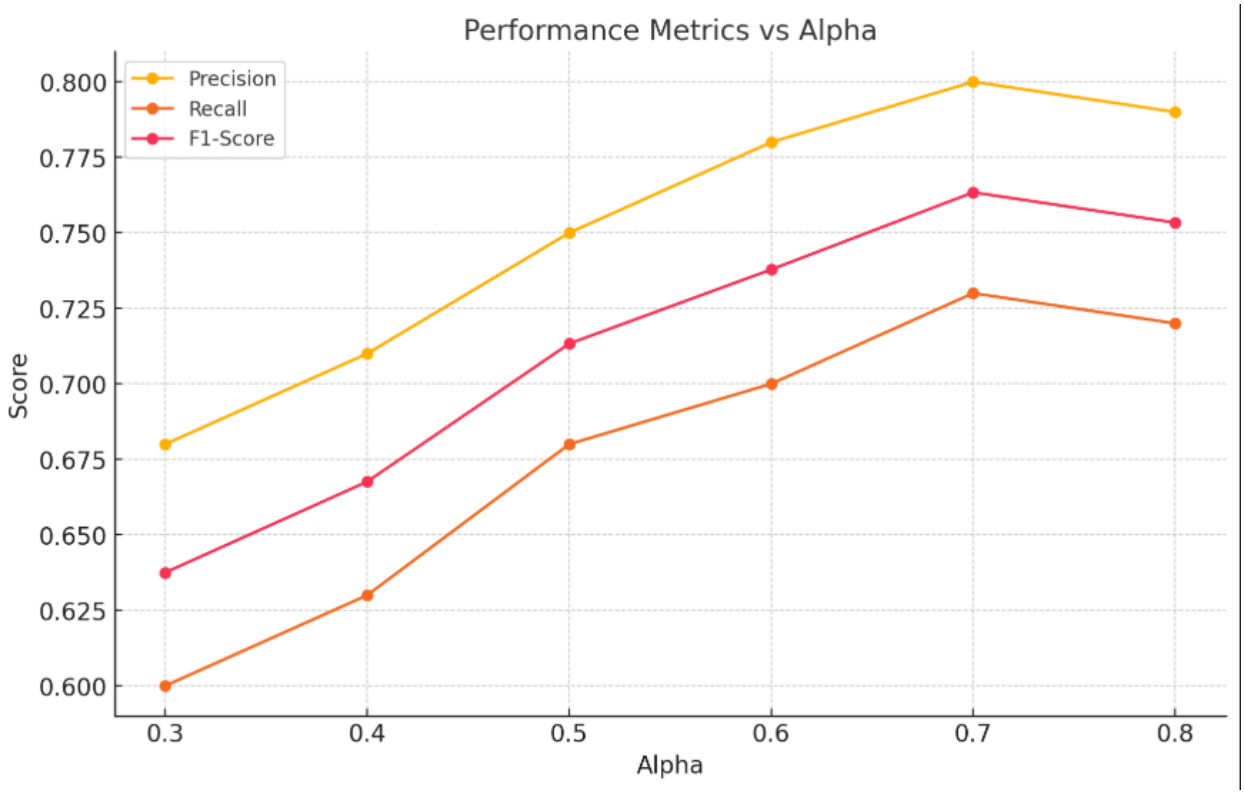

Рисунок 4.3 – Залежність Precision, Recall та F1-Score від параметра α

На графіку зображено залежність метрик Precision, Recall та F1-Score від параметра α. Максимальних значень метрики досягають при α=0.7, що свідчить про оптимальне співвідношення між колаборативною та контентною моделями.

Результати експериментів свідчать про ефективність гібридної моделі у поєднанні переваг колаборативної та контентної фільтрацій. Високі показники точності та повноти демонструють її здатність забезпечувати релевантні рекомендації.

Хоча гібридна модель трохи поступається контентній фільтрації за новизною, її загальна ефективність (точність, повнота, різноманітність і покриття) робить її оптимальним вибором для рекомендацій у середовищі Yelp Dataset.

4.4 Обговорення результатів

Результати експериментів продемонстрували, що використання гібридної моделі забезпечує високі показники Precision та Recall, особливо при

оптимальних значеннях параметра α. При α=0.7 система досягає найкращого балансу між точністю та повнотою рекомендацій, що підтверджується максимальним значенням F1-Score (0.76).

Переваги реалізованої системи:

- Здатність враховувати як історію взаємодій, так і змістовні характеристики об’єктів.
- Успішно вирішувати проблему "холодного старту" для нових користувачів за рахунок використання контентної інформації.
- Збільшувати охоплення рекомендацій, включаючи малопопулярні бізнеси, що робить систему більш універсальною.
- Гібридна рекомендаційна система продемонструвала високу ефективність завдяки поєднанню двох підходів.

Недоліки:

- Висока обчислювальна складність при роботі з великим обсягом даних.
- Необхідність точного налаштування вагового параметра для досягнення оптимальних результатів.
- Можливість появи упереджених рекомендацій через дисбаланс у даних (наприклад, якщо популярні бізнеси мають значно більше взаємодій).

Можливості для вдосконалення:

1. Інтеграція моделей глибокого навчання для більш точної обробки контентних даних.
2. Використання більш складних метрик, таких як Diversity та Novelty, для оцінки різноманітності рекомендацій.
3. Застосування онлайн-навчання для адаптації моделі в реальному часі.

Висновки до розділу

У четвертому розділі виконано тестування та оцінку ефективності гібридної рекомендаційної системи. Результати експериментів підтвердили переваги використання гібридного підходу, який забезпечив високі показники точності, різноманітності та покриття рекомендацій.

Обговорення результатів виявило як сильні сторони системи, так і аспекти, що потребують подальшого вдосконалення. Таким чином, розроблена модель є перспективним рішенням для персоналізованих рекомендацій в електронній комерції.

ВИСНОВОК

У ході виконання даної роботи було здійснено всебічне дослідження предметної області, розроблено гібридну рекомендаційну систему та проведено її тестування з використанням реальних даних. Нижче наведено основні результати та підсумки роботи:

У першому розділі детально розглянуто історичний розвиток рекомендаційних систем, їх вплив на користувачів та ключову роль у сфері електронної комерції. Було обґрунтовано актуальність проблеми створення персоналізованих рекомендацій, що враховують індивідуальні уподобання користувачів. Особливу увагу приділено джерелам даних для побудови рекомендаційних систем і ролі персоналізації у підвищенні їхньої ефективності.

У другому розділі розглянуто класичні підходи до створення рекомендаційних систем, включно з колаборативною фільтрацією та методами на основі контенту, а також їхні сильні та слабкі сторони. Було досліджено принципи роботи гібридних рекомендаційних систем і їхні переваги над традиційними підходами. Обговорено перспективи використання нейронних мереж у рекомендаційних системах і методи оцінки їхньої ефективності. Виявлено основні виклики у розробці таких систем, включно з масштабованістю, різноманітністю та урахуванням новизни рекомендацій.

Третій розділ був присвячений практичній реалізації гібридної моделі, що поєднує колаборативну фільтрацію на основі матричної факторизації (LightFM) та контентний аналіз за допомогою TF-IDF. У роботі наведено архітектуру системи, описано етапи підготовки даних, включаючи очищення, нормалізацію та створення векторних уявлень текстів. Було інтегровано обидва підходи в єдину систему з можливістю локального та хмарного розгортання.

У четвертому розділі було проведено тестування розробленої системи на основі публічного набору даних Yelp. Для оцінки її ефективності використовувалися метрики точності (Precision, Recall, F1-score), помилок прогнозування (RMSE, MAE), а також показники новизни, різноманітності та

покриття. Результати експериментів продемонстрували переваги гібридного підходу порівняно з окремими методами. Було також проведено аналіз сценаріїв тестування та параметрів, що впливали на якість рекомендацій.

Виконана робота підтвердила ефективність гібридного підходу у побудові рекомендаційних систем, особливо у сфері електронної комерції. Розроблена система показала високі результати за обраними метриками та забезпечила збалансований підхід до врахування як історії взаємодій користувачів, так і змістовних характеристик об'єктів.

Подальші дослідження можуть бути спрямовані на інтеграцію методів нейронного підходу для покращення якості рекомендацій, розширення джерел даних і підвищення масштабованості системи. Застосування запропонованої системи на практиці здатне підвищити ефективність роботи платформ електронної комерції та поліпшити користувацький досвід.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zhu, H., Ni, Y. & Zheng, Q. “A group-oriented recommendation algorithm based on similarities of personal learning generative networks”. *IEEE Access*. 2018; 1 (1): 42729–42739. DOI: https://doi.org/10.1109/ACCESS.2018.2856753.

2. Wang, H. “Group recommendation via self-attention and collaborative metric learning model”. *IEEE Access*. 2019; 7 (1): 164844–164855. DOI: https://doi.org/10.1109/ACCESS.2019.2953176.

3. Asim, M. “Content based call for papers recommendation to researchers”. *International Conference on Open Source Systems and Technologies (ICOSST)*. 2018; 1 (1): 42–47. DOI: https://doi.org/10.1109/ ICOSST.2018.8632174.

4. Beskorovainyi, V. V., Petryshyn, L. B. & Honcharenko, V. O. “Mathematical models of a multi-criteria problem of reengineering topological structures of ecological monitoring networks”. *Applied Aspects of Information Technology. Publ. Nauka I Tekhnika*. Odessa: Ukraine. 2022; 5 (1): 11–24. DOI: https://doi.org/10.15276/aait.05.2022.1.

5. Hodovychenko M. A. Recommender systems: models, challenges and opportunities / M. A. Hodovychenko, A. A. Gorbatenko // Herald of Advanced Information Technology. - 2023. - Vol. 6, no. 4. - С. 308–319. - Режим доступу: http://nbuv.gov.ua/UJRN/hait_2023_6_4_4

6. Liu, T. “A review of deep learning-based recommender system in e-learning environments”. *Artificial Intelligence Review*. 2022; 55 (8): 5953–5980. DOI: https://doi.org/10.1007/s10462-022-10135-2.

7. Chen, W. “A survey of deep nonnegative matrix factorization”. *Neurocomputing*. 2022; 491 (1): 305–320. DOI: https://doi.org/10.1016/j.neucom.2021.08.152.

8. Fang, L. “Differentially private recommender system with variational autoencoders”. *Knowledge-Based Systems*. 2022; 250 (1): 109–130. DOI: https://doi.org/10.1016/j.knosys.2022.109044.

9. Velickovic, P. “Graph attention networks”. *Stat*. 2017; 1050 (20):10–48. DOI: https://doi.org/10.48550/arXiv.1710.10903.

10. Jawarneh, I. “A pre-filtering approach for incorporating contextual information into deep learning based recommender systems”. *IEEE Access*. 2020; 8 (1): 40485–40498. DOI: https://doi.org/10.1109/ACCESS.2020.2975167.

11. Unger, A. “Context-aware recommendations based on deep learning frameworks”. *ACM Transactions on Management Information Systems (TMIS)*. 2020; 11 (2): 1–15. DOI: https://doi.org/10.1145/3386243.

12. Xin, X. “Cfm: Convolutional factorization machines for context-aware recommendation”. *IJCAI*. 2019; 19 (1): 3926–3932. DOI: https://doi.org/10.24963/ijcai.2019/545.

13. Van Dat, N. “Solving distribution problems in content-based recommendation system with gaussian mixture model”. *Applied Intelligence*. 2022; 52 (2): 1602–1614. DOI: https://doi.org/10.1007/s10489-021-02429-9.

14. Deldjoo, Y. “Using visual features based on mpeg-7 and deep learning for movie recommendation”. *International Journal of Multimedia Information Retrieval*. 2018; 7 (4): 207–219. DOI: https://doi.org/10.1007/s13735-018-0155-1.

15. Cami, B. “User preferences modeling using dirichlet process mixture model for a content-based recommender system”. *Knowledge-Based Systems*. 2019: 163 (1): 644–655. DOI: https://doi.org/10.1016/j.knosys.2018.09.028.

16. Serrano-Guerrero, J. “A t1owa and aspect-based model for customizing recommendations on ecommerce”. *Applied Soft Computing*. 2020: 97 (1): 106–125. DOI: https://doi.org/10.1016/ j.asoc.2020.106768.

17. Polignano, M. “Towards emotion-aware recommender systems: an affective coherence model based on emotion-driven behaviors”. *Expert Systems with Applications*. 2021; 170 (1): 114–130. DOI: https://doi.org/10.1016/j.eswa.2020.114382.

18. Ray, B. “An ensemble-based hotel recommender system using sentiment analysis and aspect categorization of hotel reviews”. *Applied Soft Computing*. 2021; 98 (1); 106–125. DOI: https://doi.org/10.1016/j.asoc.2020.106935.

19. Franzheva, O. D. “Analysis of quasi-periodic space-time non-separable processes to support decision-making in medical monitoring system”. *Herald of Advanced Information Technology*. 2021; 4 (3): 225–231. DOI: https://doi.org/10.15276/hait.03.2021.2.

20. Povoroznyuk, A. I., Povoroznyuk, O. A. & Shehna, K. “Formalizing the Stages of Mammographic Examinations in Designing a Medical Decision Support System”. *Herald of Advanced Information Technology*. 2020; 3 (4): 279–291. DOI: https://doi.org/10.15276/hait.04.2020.6.

21. Chen, C. “Collaborative similarity embedding for recommender systems”. *World Wide Web Conference*. 2019; 1 (1): 2637–2643. DOI: https://doi.org/10.1016/j.engappai.2019.06.020.

22. Barkan, O. “Anchor-based collaborative filtering”. *Proceedings of the 30th ACM International Conference on Information Knowledge Management*. 2021; 1 (1): 2877–2881. DOI: https://doi.org/10.1145/ 3459637.3482056.

23. Amato, F. “SOS: a multimedia recommender system for online social networks”. *Future Generation Computer Systems*. 2019; 93 (1): 914–923. DOI: https://doi.org/10.1016/j.future.2017.04.028.

24. Ma, X. “Newly published scientific papers recommendation in heterogeneous information networks”. *Mobile Networks and Applications*. 2019; 24 (1): 69–79. DOI: https://doi.org/10.1007/s11036-018-1133-9.

25. Pham, P. “A hierarchical fused fuzzy deep neural network with heterogeneous network embedding for recommendation”. *Information Sciences*. 2023; 620 (1): 105–124. DOI: https://doi.org/10.1016/ j.ins.2022.11.085.

26. Vijayakumar, V. “Effective knowledge-based recommender system for tailored multiple point of interest recommendation”. *International Journal of Web Portals (IJWP)*. 2019; 11 (1): 1–18. DOI: https://doi.org/ 10.4018/IJWP.2019010101.

27. Xia, L. “Knowledge-enhanced hierarchical graph transformer network for multi-behavior recommendation”. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021; 35 (5): 4486–4493. DOI: https://doi.org/10.1609/aaai.v35i5.16576.

28. Gong, F. “Smr: medical knowledge graph embedding for safe medicine recommendation”. *Big Data Research*. 2021; 23 (1): 152–174. DOI: https://doi.org/10.1016/j.bdr.2020.100174.

29. Abu-Salih, B. “Toward a knowledge-based personalised recommender system for mobile app development”. *Journal of Universal Computer Science*. 2021; 27 (2): 208–229. DOI: https://doi.org/ 10.3897/jucs.65096.

30. Lehmann, J. “Dbpedia – a large-scale, multilingual knowledge base extracted from Wikipedia”. *Semantic Web*. 2015; 6 (2): 167–195. DOI: https://doi.org/10.3233/SW-140134.

31. Hu, B. “Transmkr: Translation-based knowledge graph enhanced multi-task point-of-interest recommendation”. *Neurocomputing*. 2022; 474 (1): 107–114. DOI: https://doi.org/10.1016/ j.neucom.2021.11.049.

32. Wu, C. “Knowledge graph-based multi-context-aware recommendation algorithm”. *Information Sciences*. 2022; 595 (1): 179–194. DOI: https://doi.org/10.1016/j.ins.2022.02.054.

33. Wang, X. “Explainable reasoning over knowledge graphs for recommendation”. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2019; 33 (1): 5329–5336. DOI: https://doi.org/10.1609/ aaai.v33i01.33015329.

34. Kiran, R. “Dnnrec: A novel deep learning based hybrid recommender system”. *Expert Systems with Applications*. 2020; 144 (1): 113–130. DOI: https://doi.org/10.1016/j.eswa.2019.113054.

35. Liu, Y. “A novel deep hybrid recommender system based on auto-encoder with neural collaborative filtering”. *Big Data Mining and Analytics*. 2018; 1 (3): 211–221. DOI: https://doi.org/10.26599/ BDMA.2018.9020019.

36. Biswas, P. “A hybrid recommender system for recommending smartphones to prospective customers”. *Expert Systems with Applications*. 2022; 208 (1): 118 – 128. DOI: https://doi.org/10.1016/ j.eswa.2022.118058.

37. Huang, Z. “A novel group recommendation model with two-stage deep learning”. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*. 2021; 52 (9): 5853–5864. DOI: https://doi.org/10.1109/TSMC.2021.3131349.

38. Kumar, C. “Automatically detecting groups using locality-sensitive hashing in group recommendations». *Information Sciences*. 2022; 601 (1): 207–223. https://doi.org/10.1016/j.ins.2022.04.028

39. Zan, S. «Uda: A user-difference attention for group recommendation”. *Information Sciences*. 2021; 571 (1): 401–417. DOI: https://doi.org/10.1016/j.ins.2021.04.084.

40. Zhao, S. “Multi-view intent disentangle graph networks for bundle recommendation”. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2022; 36 (4): 4379–4387. DOI: https://doi.org/10.1609/aaai.v36i4.20359.

41. Tang, J. “Adversarial training towards robust multimedia recommender system”. *IEEE Transactions on Knowledge and Data Engineering*. 2019; 32 (5): 855–867. DOI: https://doi.org/10.1109/ TKDE.2019.2893638.

42. Understanding TF-IDF (Term Frequency-Inverse Document Frequency) in python [Електронний ресурс] // @azunan3. – 2019. – Режим доступу до ресурсу: https://medium.com/@azunan3/understanding-tf-idf-term-frequency-inverse-document-frequency-in-python-373070acb895.

43. Raman V. Recommender Engine — Under The Hood [Електронний ресурс] / Venkat Raman // @venksaiyan. – 2017. – Режим доступу до ресурсу: https://towardsdatascience.com/recommender-engine-under-the-hood-7869d5eab072.

Додаток А

Рисунок 1. – Вивід програми на локальному сервері

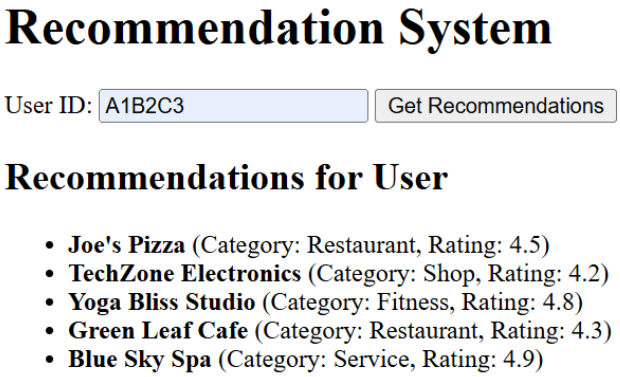

Рисунок 2. – Результат програми на локальному сервері

```
├── config
│   ├── config_base.json            # Базова конфігурація для всіх моделей
│   ├── config_content.json         # Конфігурація для контентного аналізу
│   ├── config_item_cf.json         # Конфігурація для Item-based CF
│   └── config.py                   # Сценарій для роботи з конфігураціями
├── models
│   ├── __init__.py                 # Ініціалізація пакету models
│   ├── base_model.py               # Базовий клас для всіх моделей
│   ├── content_based_model.py      # Модель для контентного аналізу
│   └── item_cf_model.py            # Модель Item-based Collaborative Filtering
├── scripts
│   ├── als_recommender.py          # Побудова ALS-рекомендацій
│   ├── evaluate.py                 # Оцінка моделей
│   ├── predict_als.py              # Прогнозування за ALS-моделлю
│   ├── predict_hybrid.py           # Прогнозування гібридної моделі
│   ├── predict_user_cf.py          # Прогнозування User-based CF
│   ├── train_als.py                # Навчання ALS-моделі
│   ├── train_hybrid.py             # Навчання гібридної моделі
│   └── train_user_cf.py            # Навчання User-based CF
├── utils
│   ├── metrics.py                  # Метрики для оцінки якості рекомендацій
│   ├── misc.py                     # Допоміжні функції (обробка даних, тексту тощо)
│   └── predict.py                  # Загальні функції прогнозування
└── train.py                        # Основний скрипт для запуску тренування
```

Рисунок 3. – Структура проекту

Вміст файлу config.py (основний скрипт для динамічної роботи з конфігураціями):

```
import os

os.environ['PYSPARK_PYTHON'] = 'python3'
os.environ['PYSPARK_DRIVER_PYTHON'] = 'python3'

# -- Application vars
APP_NAME = "YelpRecommender"

# -- Model configurations
# model_conf = "config/config_base.json"
# model_conf = "config/config_content.json"
# model_conf = "config/config_content_sparse.json"
# model_conf = "config/config_item_cf.json"


def validate_dirs(mcf):
    from pathlib import Path
    mdl_f = Path(os.path.abspath(mcf))
    if not os.path.exists(mdl_f.parent):
        os.makedirs(mdl_f.parent)

def load_conf():
    import json
    with open(model_conf, 'r') as mf:
        mdl_cnf = json.load(mf)
    assert 'class' in mdl_cnf, "No Model Class!"
    assert 'mdl_file' in mdl_cnf, "No Model output file!"
```

```
    assert 'hp_params' in mdl_cnf, "No Hyperparameters!"
    assert 'training_data' in mdl_cnf, "No Training data!"
    validate_dirs(mdl_cnf['mdl_file'])
    return mdl_cnf
```

Вміст файлу config_base.json (базовий файл конфігурації):

```
{
    "class": "BaseModel",
    "training_data": "../../data/project/train_review.json",
    "mdl_file": "weights/base.model",
    "hp_params": {
        "TOP_TFIDF": 200,
        "RARE_WORDS_PERC": 0.0001
    }
}
```

Вміст файлу config_content.json (основний файл конфігурації для контентного аналізу):

```
{
    "class": "ContentBasedModel",
    "training_data": "../../data/project/train_review.json",
    "mdl_file": "weights/contentTFIDF200.model",
    "hp_params": {
        "TOP_TFIDF": 200,
        "RARE_WORDS_PERC": 0.0001,
        "FEATURES": "continuous",
        "DECISION_RULE": {
            "active": "geometric",
            "params": {
                "slope": 0.7,
                "bias": 0.5
            }
        }
    }
}
```

Вміст файлу config_content_sparse.json (використовується для роботи з розрідженими матрицями):

```
{
    "class": "ContentBasedModel",
    "training_data": "../../data/project/train_review.json",
    "mdl_file": "weights/contentTFIDFsparse.model",
    "hp_params": {
        "TOP_TFIDF": 100000,
        "MIN_DOC_FREQ": 2,
        "RARE_WORDS_PERC": 0.0001,
        "FEATURES": "sparse",
        "DECISION_RULE": {
            "active": "geometric",
            "params": {
                "slope": 0.7,
                "bias": 0.5
            }
```

```
        }
    }
}
```

Вміст файлу config_item_cf.json (конфігурація для моделі Item-based Collaborative Filtering):

```
{
    "class": "ItemBasedCFModel",
    "training_data": "../../data/project/train_review.json",
    "mdl_file": "weights/itemCF.model",
    "hp_params": {
        "MIN_CORRATED": 2,
        "N_MIN_HASHES": 3,
        "K_NEIGHS": 10,
        "METRIC": {
            "active": "jacc",
            "min_value": 0.001
        }
    }
}
```

Вміст файлу __init__.py:

```
from models.base_model import BaseModel
from models.content_based_model import ContentBasedModel
from models.item_cf_model import ItemBasedCFModel

models = {
    "BaseModel": BaseModel,
    "ContentBasedModel": ContentBasedModel,
    "ItemBasedCFModel": ItemBasedCFModel,
}
```

Вміст файлу base_model.py (базовий клас для побудови моделей):

```
class BaseModel(object):

    def __init__(self, sc, cfg):
        """ Base model constructor
        """
        self._sc = sc
        self.cfg = cfg

    def train(self, data):
        """ Training method

            Params:
            -----
            data: pyspark.rdd
                Input Data
        """
        pass

    def predict(self, data):
        """ Prediction method
        """
```

```
        pass
```

Вміст файлу content_based_model.py (контентно-орієнтована модель):

```
import re
import os
import json
from pprint import pprint
import math
from collections import OrderedDict, Counter, namedtuple

from pyspark.mllib.feature import HashingTF, IDF
from pyspark.mllib.linalg import SparseVector
# from pyspark.sql import SQLContext, Row
import scipy.sparse as sps
import numpy as np

from models.base_model import BaseModel
from utils.misc import log, debug, read_json
from utils.metrics import mean, cosine_similarity


puncts = [ "(", "[", ",", ".", "!", "?", ":", ";",
        "]", ")" ,"\n", "*", "/", " ", "$", "'",
        '"', '-', '\r', '#']
puncts_re = r"(\(|\[|\,|\.|\!|\?|\:|\;|\]|\)|\n|\*|\/|\$|\'|\"|-|\#|\r)"


class ContentBasedModel(BaseModel):
    """ Content Based recommendation model
        based on TF-IDF features
    """

    def __init__(self, sc, cfg):
        """ Content Based constructor
        """
        super().__init__(sc,cfg)
        # self._spark = SQLContext(self._sc)
        self.topk_tfidf = self.cfg['hp_params']['TOP_TFIDF']
        self.feat_type = self.cfg['hp_params']['FEATURES']
        # params for linear decision rule
        self.alpha = self.cfg['hp_params']['DECISION_RULE']['params']['slope']
        self.beta = self.cfg['hp_params']['DECISION_RULE']['params']['bias']
        # aux elements
        with open('utils/stopwords') as f:
            self.stops = f.read().split('\n')
        # model vars
        self.num_revs = 0

    def preprocess(self, rdd):
        """ Preprocess text reviews. Parse text,
            removing stop words, punctuation and empy spaces

            Params:
            -----
            rdd: pyspark.rdd
                Review JSON RDD
                Format: [
                    {'business_id':x, 'user_id':x, 'text':x},
                    ...
```

```
            ]
        Returns:
        -----
        pyspark.rdd
            Format: [
                ((biz_id, usr_id), [texts])
            ]
    """
    def extend_puncts(l):
        return re.sub(
            puncts_re,
            lambda x: " "+x.group(0)+" ",
            l
        )
    remove_set = set(puncts+self.stops+[''])
    parsed_data = rdd.map(lambda x: (
                        (x['business_id'], x['user_id']),
                        x['text'])
                    ).map(lambda x: (
                        x[0],
                        extend_puncts(x[1].lower()).split(' '))
                    ).zipWithIndex()\
                    .map(lambda z: ((z[1], z[0][0]), z[0][1]))\
                    .flatMapValues(lambda x: x)\
                    .filter(lambda x: x[1] not in remove_set)\
                    .groupByKey()\
                    .map(lambda x: (x[0][1], x[1]))
    return parsed_data

@staticmethod
def get_revs(data, prof):
    """ Join all reviews by business or user to build profile.

        Params:
        -----
        data: pyspark.rdd
            Format: [((biz, usr), <texts>), ...]

        Return:
        ----
        pyspark.rdd
            Format: [
                (b, [[texts],..])
            ]
    """
    _j = 0 if prof == 'biz' else 1
    _revs = data.map(lambda x: (x[0][_j], x[1]))\
                    .mapValues(list)\
                    .groupByKey()
    return _revs

@staticmethod
def get_DF(data):
    """ Count Terms' Document Frequency

        Params:
        ----
        data: pyspark.rdd
            Format: [((b,u), [texts]),...]

        Returns:
        -----
```

```
            pyspark.rdd
                Format: [(word, freq), ...]
        """
        mapped = data.flatMapValues(lambda x: set(x))\
                    .map(lambda x: (x[1], x[0]))\
                    .groupByKey()
        # Document frequency
        doc_freq = mapped.mapValues(len)
        return doc_freq.collectAsMap()

    def get_tfidf(self, data, doc_fq):
        """ Compute TF-IDF vectors

            Params:
            -----
            data: pyspark.rdd
                Format: [(k, [texts]), ...]
            doc_fq: pypsark.broadcast
                Dict with term's doc frequency

            Returns:
            -----
            (tfidf, top_k, top_idx): (pyspark.rdd, OrderedDict, dict)
                tfidf: [(k, {w:(tf,df, tf-idf),..}), ...]
                top_k: {k: tf-idf, k2:tf-idf, ...}
        """
        N = self.num_revs
        # TF  (biz_id, {t1: 3, t2: 4})
        def normed_tf(x, norm=False):
            k, v  = x
            c = Counter(v)
            # for _k in v:
            #     c.update(_k)
            if norm:
                _max = c.most_common()[0][1]
                return (k, {i: j/_max for i,j in c.items()})
            return (k, {i: j for i,j in c.items()})
        tfq = data.mapValues(list).map(normed_tf)
        # IDF (biz_id, {t1: (2,3,4.5), t2: (1,3,6.7)}
        def _tfidf(val):
            k, _tf = val
            d = {}
            for term,v in _tf.items():
                _df = doc_fq.value[term]
                d[term] = (v, _df, v * math.log(N/_df, 2))
            return (k, d)
        tfidf = tfq.map(_tfidf)
        # Get most recent elements, top terms {t1: 23.4, t2:6.4}
        top_terms = tfidf.flatMap(lambda x: [(_k, _j)  for _k, _j in x[1].items()])\
                        .filter(lambda x: x[1][1] > 1)\
                        .map(lambda x: (x[0], x[1][2]))\
                        .groupByKey()\
                        .mapValues(max)
        top_terms = OrderedDict(
            top_terms
                .sortBy(lambda x: x[1], ascending=False)\
                .take(self.topk_tfidf)
        )
        # - pos index
        top_idx = {_ky:_i for _i,_ky in enumerate(top_terms)}
        log("Got Top Terms")
        return tfidf, top_terms, top_idx
```

```
def featurize(self, data, ftype='onehot'):
    """ Generate TF-IDF features
    """
    # format text data
    parsed = self.preprocess(data).cache()
    self.num_revs = parsed.count()
    if ftype in ('onehot', 'continuous'):
        # generate features
        _df = self._sc.broadcast(CBM.get_DF(parsed))
        log("Document term frequecy:")
        pprint(list(_df.value.items())[:10])
        return parsed, self.get_tfidf(parsed.mapValues(list), _df)
    elif ftype == 'sparse':
        tf_hasher = HashingTF(self.topk_tfidf)
        # All document hashes
        _tf = tf_hasher.transform(parsed\
                                .mapValues(list)\
                                .map(lambda x: x[1])
                        )
        idfer = IDF(minDocFreq=self.cfg['hp_params']['MIN_DOC_FREQ'])\
                    .fit(_tf)
        TFIDF = namedtuple("TFIDF", ('tfer','idfer'))
        tfidf = TFIDF(tf_hasher, idfer)
        log("Constructed Sparse TFIDF!")
        return parsed, (tfidf, None, None)
    return None, (None, None, None)

def get_onehot_profile(self, feats, top_idx):
    """ One-hot profile construction

        Params:
        -----
        feats: pyspark.rdd
            [(k, {words}), ...]
        top_idx: dict
            Position Index for top terms

        Returns:
        -----
        pyspark.rdd
            [(k, [0,1,0,0,1]), ..]
    """
    _TOP_TFIDF = self.topk_tfidf
    # One-hot encode
    def one_hot_tdf(x):
        one_hot = [0]*_TOP_TFIDF
        for w in x[1]:
            if w in top_idx:
                one_hot[top_idx[w]] = 1
        return (x[0], one_hot)
    return  feats.map(one_hot_tdf)

def get_continuous_profile(self, feats, top_terms, top_idx):
    """ Continuous profile construction

        Params:
        -----
        feats: pyspark.rdd
            [(k, [words]), ...]
        top_terms: dict
            Fast access TFIDF values of top-k
```

```
        top_idx: dict
            Position Index for top terms

        Returns:
        -----
        pyspark.rdd
            [(k, [0,2.1,0,4.0,0.1]), ..]
    """
    _TOP_TFIDF = self.topk_tfidf
    def _encode(x):
        vect = [0]*_TOP_TFIDF
        for w in x[1]:
            if w in top_idx:
                vect[top_idx[w]] = top_terms[w]
        return (x[0], vect)
    return feats.map(_encode)

def get_sparse_profile(self, feats, tfidf):
    """ Sparse profile construction

        Params:
        -----
        feats: pyspark.rdd
            [(k, [words]), ...]
        tfidf: TFIDF (HashingTF, IDF)

        Returns:
        -----
        pyspark.rdd
            #### --- TODO
    """
    data = feats.zipWithIndex()\
                .map(lambda x: (x[1], x[0])).cache()
    _tf = tfidf.tfer.transform(
        data.map(lambda x: list(x[1][1]))
    )
    _tfidf = tfidf.idfer.transform(_tf)
    embedding = data.map(lambda x: (x[0], x[1][0]))\
                    .join(
                        _tfidf.zipWithIndex()\
                            .map(lambda x: (x[1], x[0]))
                    )
    return embedding.map(lambda x: x[1])

def _get_profile(self, feats, tfidf, top_terms, top_idx, ftype):
    """ Build profile based on features

        Params:
        -----
        feats: pyspark.rdd
            [(k, {words}), ...]
        tfidf: pyspark.rdd | TFIDF
            TF-IDF [((b,u), {w:(tf,df, tfidf), ...}), ...]
        top_terms: dict
            Fast access TFIDF values of top-k
        top_idx: dict
            Position Index for top terms
        ftype: str
            Feature type

        Returns:
        -----
```

```
            pyspark.rdd
                Vector representation: [(k, [3,2.5,6,7,..]), ..]
        """
        if ftype == 'onehot':
            return self.get_onehot_profile(feats, top_idx)
        elif ftype == 'continuous':
            return self.get_continuous_profile(feats, top_terms, top_idx)
        elif ftype == 'sparse':
            return self.get_sparse_profile(feats, tfidf)
        return None

    def build_profiles(self, data, tfidf, top_terms, top_idx, ftype):
        """ Build User and Item profiles

            Params:
            -----
            data: pyspark.rdd
                Parsed data [((b,u), [text]),  ...]
            tfidf: pyspark.rdd | TFIDF
                TF-IDF [((b,u), {w:(tf,df, tfidf), ...}), ...]
            top_terms: dict
                Fast access TFIDF values of top-k
            top_idx: dict
                Position Index for top terms
            ftype: str
                Feature type
        """
        biz_revs = CBM.get_revs(data, 'biz')\
                    .flatMapValues(lambda x: x)\
                    .flatMapValues(lambda x: x)\
                    .groupByKey()
        user_revs = CBM.get_revs(data, 'user')\
                    .flatMapValues(lambda x: x)\
                    .flatMapValues(lambda x: x)\
                    .groupByKey()
        if ftype == 'onehot':
            biz_prof = self._get_profile(biz_revs.mapValues(set),
                                tfidf, top_terms, top_idx, ftype)
            user_prof = self._get_profile(user_revs.mapValues(set),
                            tfidf, top_terms, top_idx, ftype)
        elif ftype in ('continuous', 'sparse'):
            biz_prof = self._get_profile(biz_revs, tfidf,
                                top_terms, top_idx, ftype)
            user_prof = self._get_profile(user_revs, tfidf,
                                top_terms, top_idx, ftype)
        else:
            return None, None
        return biz_prof, user_prof

    def compute_avgs(self, data):
        """ Compute Business and User Avgs

             Params:
            ----
            data: pyspark.rdd
                Review JSON RDD
                Format: [
                    {'business_id':x, 'user_id':x, 'text':x},
                    ...
                ]

            Returns:
```

```
            -----
            (dict, dict)
                (User Avgs, Business Avgs)
        """
        user_avg = data.map(lambda x: (x['user_id'], x['stars']))\
                       .groupByKey().mapValues(mean)\
                       .collectAsMap()
        buss_avg = data.map(lambda x: (x['business_id'], x['stars']))\
                       .groupByKey().mapValues(mean)\
                       .collectAsMap()
        log("Got Business and User rating averages")
        return user_avg, buss_avg

    def save(self, top_terms, top_idx, biz_prof, user_prof, biz_avg, user_avg):
        """ Save Model values
        """
        if self.feat_type in ('continuous', 'onehot'):
            # Save profiles
            _biz_prof = biz_prof.collect()
            _usr_prof = user_prof.collect()
            with open(self.cfg['mdl_file']+'.profiles', 'w') as prf:
                prf.write(json.dumps({
                    "business_profiles": _biz_prof,
                    "user_profiles": _usr_prof,
                    "business_avg": biz_avg,
                    "user_avg": user_avg
                }))
            # Save content
            with open(self.cfg['mdl_file']+'.features', 'w') as prf:
                prf.write(json.dumps({
                    "top_terms": top_terms,
                    "terms_pos_idx": top_idx
                }))
            # In memory value assignation
            self.top_terms, self.top_idx = top_terms, top_idx
            self.biz_prof, self.user_prof = dict(_biz_prof), dict(_usr_prof)
            self.biz_avg, self.user_avg = biz_avg, user_avg
        elif self.feat_type == 'sparse':
            # save profiles
            def write_profile(x, pfile):
                with open(pfile, 'a') as bf:
                    bf.write(json.dumps({x[0]:
                        (x[1].size,
                        x[1].indices.tolist(),
                        x[1].values.tolist())
                        })+"\n"
                    )
                return 1
            bfile = self.cfg['mdl_file']+'.biz_profile'
            biz_prof.map(lambda x: write_profile(x, bfile)).count()
            ufile = self.cfg['mdl_file']+'.user_profile'
            user_prof.map(lambda x: write_profile(x, ufile)).count()
            # save avgs
            with open(self.cfg['mdl_file']+'.avgs', 'w') as prf:
                prf.write(json.dumps({
                    "business_avg": biz_avg,
                    "user_avg": user_avg
                }))
            self.top_terms, self.top_idx = None, None
            self.biz_prof, self.user_prof = None, None
            self.biz_avg, self.user_avg = biz_avg, user_avg
```

```
    def train(self, data):
        """ Training method

            Params:
            ----
            data: pyspark.rdd
                Review JSON RDD
                Format: [
                    {'business_id':x, 'user_id':x, 'text':x},
                    ...
                ]
        """
        user_avg, biz_avg = self.compute_avgs(data)
        parsed, (tfidf, top_terms, top_idx) = self.featurize(data, self.feat_type)
        biz_prof, user_prof = self.build_profiles(parsed, tfidf, top_terms, top_idx,
self.feat_type)
        self.save(top_terms, top_idx, biz_prof, user_prof, biz_avg, user_avg)
        log(f"Model correctly saved at {self.cfg['mdl_file']}")
        return parsed

    def load_model(self):
        """ Load model from config defined model file
        """
        if self.feat_type in ('onehot', 'continuous'):
            # load profiles
            with open(self.cfg['mdl_file']+'.profiles', "r") as buff:
                mdl_ = json.loads(buff.read())
            self.biz_prof = dict(mdl_['business_profiles'])
            self.user_prof = dict(mdl_['user_profiles'])
            self.biz_avg = mdl_['business_avg']
            self.user_avg  = mdl_['user_avg']
            # load features
            with open(self.cfg['mdl_file']+'.features', "r") as buff:
                mdl_f = json.loads(buff.read())
            self.top_terms, self.top_idx  = mdl_f['top_terms'], mdl_f['terms_pos_idx']
        elif self.feat_type == 'sparse':
            mdl_b = read_json(self._sc, self.cfg['mdl_file']+'.biz_profile')
            self.biz_prof = mdl_b.map(lambda x: list(x.items())[0])\
                                 .mapValues(lambda sv: SparseVector(*sv)).collectAsMap()
            mdl_b = read_json(self._sc, self.cfg['mdl_file']+'.user_profile')
            self.user_prof = mdl_b.map(lambda x: list(x.items())[0])\
                                 .mapValues(lambda sv: SparseVector(*sv)).collectAsMap()
            # load avgs
            with open(self.cfg['mdl_file']+'.avgs', "r") as buff:
                mdl_ = json.loads(buff.read())
            self.biz_avg = mdl_['business_avg']
            self.user_avg  = mdl_['user_avg']
        else:
            log("Not valid feature type!", lvl="WARNING")
            return
        log(f"Model correctly loaded from {self.cfg['mdl_file']}")

    def cold_start(self, test, users, biz):
        """ [NOT USED FOR NOW] Cold Start strategy
        """
        # Average of the rest of the population
        missing_biz = (set(test.map(lambda x: x['business_id']).distinct().collect())
                - set(biz))
        missing_usr = (set(test.map(lambda x: x['user_id']).distinct().collect())
                - set(users))
        return missing_biz, missing_usr
```

```
    def predict(self, test, outfile):
        """ Prediction method

            Params:
            ----
            test: pyspark.rdd
                Test Review JSON RDD
                Format: [
                    {'business_id':x, 'user_id':x, 'text':x},
                    ...
                ]
            outfile: str
                Path to output file
        """
        _feat_type = self.feat_type
        users, biz = self.user_prof, self.biz_prof
        user_avg, biz_avg = self.user_avg, self.biz_avg
        _alpha, _beta = self.alpha, self.beta
        _decision = self.cfg['hp_params']['DECISION_RULE']['active']
        def _sim(u_i, b_i, u, b):
            if (u and b):
                if _feat_type == 'sparse':
                    _cos = cosine_similarity(
                        [u.toArray()],[b.toArray()]
                    ).item()
                else:
                    _cos = cosine_similarity([u],[b]).item()
                if _decision == 'linear':
                    return user_avg[u_i] + _alpha*(_cos - _beta)
                elif _decision == 'geometric':
                    return (_cos)*user_avg[u_i]  + (1-_cos)*biz_avg[b_i]
                else:  # constant
                    return 5*(_cos)
            # similarity for cold start, average of the other
            if u:
                # No business info, return avg from user
                return user_avg[u_i]
            elif b:
                # No user info, return avg from business
                return biz_avg[b_i]
            return 2.5 # return constant
        preds_ = test.map(lambda x: (x['user_id'], x['business_id']))\
                    .map(lambda x: (x[0], x[1], users.get(x[0], []), biz.get(x[1], []))
)\
                    .map(lambda x: (x[0], x[1], _sim(*x)) ).collect()
        with open(outfile, 'w') as of:
            for pv in preds_:
                of.write(json.dumps({
                    "user_id": pv[0], "business_id": pv[1],
                    "stars": pv[2]
                })+"\n")


CBM = ContentBasedModel
```

Вміст файлу item_cf_model.py (частина колаборативної фільтрації для рекомендацій на основі схожих товарів):

```
from itertools import combinations
import json
```

```
from pyspark.ml.feature import MinHashLSH
from pyspark.sql import SQLContext, Row
from pyspark.sql.types import  *
from pyspark.sql import functions as F
from pyspark.ml.linalg import SparseVector, Vectors
import numpy as np
import pandas as pd
from scipy.stats import pearsonr
from sklearn.metrics import jaccard_score

from models.base_model import BaseModel
from utils.misc import log, debug, read_json
from utils.metrics import mean
from utils.metrics import cosine_similarity



class ItemBasedCFModel(BaseModel):

    def __init__(self, sc, cfg):
        """ Item-based CF model constructor
        """
        super().__init__(sc,cfg)
        self._sql = SQLContext(self._sc)
        # params
        self.min_corrated = self.cfg['hp_params']['MIN_CORRATED']
        self.n_minhashes = self.cfg['hp_params']['N_MIN_HASHES']
        self.icf_metric = self.cfg['hp_params']['METRIC']
        self.k_n = self.cfg['hp_params']['K_NEIGHS']

    def get_ratings_by_business(self, data):
        """ Read and format ratings

            Params:
            -----
            data: pyspark.rdd
                Review JSON RDD
                Format: [
                    {'business_id':x, 'user_id':x, 'text':x},
                    ...
                ]

            Returns: pyspark.dataframe
                Cols: ['biz', 'user', 'stars']
        """
        rates = self._sql.createDataFrame(
            data.map(lambda x: (x['business_id'], x['user_id'],x['stars'])),
            ['biz', 'user', 'stars']
        )
        rates_ = rates.toPandas()
        rates_by_biz = rates_.groupby('biz')
        log("Got ratings by business!")
        return rates_by_biz['user'].apply(set).to_dict(), \
                rates_.set_index(['biz', 'user']), \
                rates.rdd.map(lambda r: r.biz).distinct().sortBy(lambda x: x)

    def prepare(self, data):
        """ Fetch business candidates

            Params:
            -----
```

```
            data: pyspark.rdd
                Review JSON RDD
                Format: [
                    {'business_id':x, 'user_id':x, 'text':x},
                    ...
                ]
        """
        if 'lsh':
            return self.local_hashing_candidates(data)
        # brute force comparison
        b_rate_set, rates_df, unique_biz = self.get_ratings_by_business(data)
        # Fetch permutations
        _min_corrated = self.min_corrated
        _permuts = unique_biz.cartesian(unique_biz).filter(lambda x: x[0] < x[1])
        _valid = _permuts.map(lambda x: (x, b_rate_set[x[0]], b_rate_set[x[1]]))\
                        .map(lambda x: (x[0], x[1].intersection(x[2]) ))\
                        .map(lambda x: (x[0], (len(x[1]), x[1]) ))\
                        .filter(lambda x: x[1][0] >= _min_corrated)
        valid_df = pd.DataFrame(_valid.collect())
        return valid_df, None, rates_df, None

    def local_hashing_candidates(self, data):
        """ MinHashLSH from hash

            Params:
            -----
            data: pyspark.rdd
                Review JSON RDD
                Format: [
                    {'business_id':x, 'user_id':x, 'text':x},
                    ...
                ]

            Returns:
            -----
            candidates: pyspark.rdd
                [(busines_i_id, business_j_id), ...]
            business_feats: pyspark.dataframe
                ['biz_id','biz','features']
            rate: pyspark.dataframe
                ['biz','user','stars']
            user_map: dict
                {user_key: user_pos, ...}
        """
        # Convert rates
        rates = self._sql.createDataFrame(
            data.map(lambda x: (x['business_id'], x['user_id'],x['stars'])),
            ['biz', 'user', 'stars']
        ).cache()
        user_map = rates.rdd.map(lambda r:
r.user).distinct().zipWithIndex().collectAsMap()
        num_users = len(user_map)
        business_feats = self._sql.createDataFrame(
            rates.rdd\
                .groupBy(lambda r: r.biz)\
                .mapValues(lambda vs: SparseVector(num_users,
                                    {user_map[v.user]: v.stars for v in vs}))\
                .zipWithIndex()\
                .map(lambda r: (r[1], r[0][0],r[0][1]) ),
            ['biz_id', 'biz', 'features']
        ).cache()
        # Run MinHash LSH
```

```
        mh_lsh = MinHashLSH(inputCol="features", outputCol="hashes",
                            numHashTables=self.n_minhashes, seed=12345)\
                .fit(business_feats.select('biz_id','features'))
        preds = mh_lsh.transform(business_feats.select('biz_id','features'))
        # Find candidates -- [TODO double-check candidate generation]
        candidates = preds.rdd.map(lambda r: Row(biz_id=r.biz_id, hashes=tuple(h[0] for
h in r.hashes)))\
                        .groupBy(lambda r: r.hashes)\
                        .mapValues(lambda l: [j.biz_id for j in l])\
                        .filter(lambda v: len(v[1]) >= 2)\
                        .map(lambda x: x[1])\
                        .flatMap(lambda x: combinations(x, 2))
        log("Candidates", candidates.take(3))
        return candidates, business_feats, rates, user_map

    def compute_weights(self, candidates, features):
        """ Compute Pearson correlation between vectors

            Params:
            -----
            candidates: pyspark.rdd
                [(busines_i_id, business_j_id), ...]
            business_feats: pyspark.dataframe
                ['biz_id','biz','features']
        """
        features_ = features.rdd.map(lambda r: (r.biz_id, r.features)).collectAsMap()
        cfeats = candidates.map(lambda c: (
                            c, Row(a=features_[c[0]], b=features_[c[1]])
                            )).cache()
        c_pears = cfeats.mapValues(lambda r: pearsonr(
                                            r.a.toArray(),
                                            r.b.toArray())[0])\
                        .collectAsMap()
        c_cos = cfeats.mapValues(lambda r: cosine_similarity(
                                            [r.a.toArray()],
                                            [r.b.toArray()]).item())\
                        .collectAsMap()
        c_jacc = cfeats.mapValues(lambda r: jaccard_score(
                                            r.a.toArray().astype(bool),
                                            r.b.toArray().astype(bool)))\
                        .collectAsMap()
        c_metrics = pd.DataFrame([c_pears, c_cos, c_jacc]).T\
                        .rename(columns={0:'pears',1:'cos',2:'jacc'})\
                        .reset_index()
        features_ = features.rdd.map(lambda r: (r.biz_id, r.biz)).collectAsMap()
        c_metrics['b1'] = c_metrics['index'].apply(lambda x: features_[x[0]])
        c_metrics['b2'] = c_metrics['index'].apply(lambda x: features_[x[1]])
        return c_metrics.drop('index', axis=1)

    def assign_weights(self, wgts):
        """  Assign weights to `self.icf_weights` based on criteria
        """
        _metrics = ['pears', 'cos', 'jacc']
        if self.icf_metric['active'] == 'mean':
            self.icf_weights = wgts[['b1', 'b2']].copy()
            self.icf_weights['w'] = wgts[_metrics].mean(1)
        elif self.icf_metric['active'] in _metrics:
            self.icf_weights = wgts[['b1','b2', self.icf_metric['active']]]\
                                .rename(columns={self.icf_metric['active']:'w'})
        else:
            raise Exception("Not valid weighting metric!")
        # Filter values
```

```
        self.icf_weights = self.icf_weights[self.icf_weights.w >=
self.icf_metric['min_value']]
        log("Model similar pairs:", self.icf_weights.w.count())

    def save(self, weights, biz_vectors, user_avg, biz_avg, user_map):
        """ Save Model values
        """
        # save weights
        weights.to_csv(self.cfg['mdl_file']+'.weights')
        self.assign_weights(weights)
        # save business vectors
        _fts_file = self.cfg['mdl_file']+'.features'
        def _serialize(v):
            return (v.size, v.indices.tolist(), v.values.tolist())
        def write_fts(x):
            with open(_fts_file, 'a') as f:
                f.write(x+'\n')
        biz_vectors.rdd.map(lambda r: json.dumps({
                                "biz_id": r.biz_id,
                                "biz": r.biz,
                                "features": _serialize(r.features)
                                }))\
                        .map(write_fts).count()
        self.biz_vectors = biz_vectors
        # save averages
        with open(self.cfg['mdl_file']+'.avgs', 'w') as prf:
                prf.write(json.dumps({
                    "business_avg": biz_avg,
                    "user_avg": user_avg,
                    "user_map": user_map
                }))
        self.biz_avg, self.user_avg = biz_avg, user_avg
        self.user_map = user_map

    def load_model(self):
        """ Load model values from config
        """
        self.assign_weights(
            pd.read_csv(self.cfg['mdl_file']+'.weights').drop('Unnamed: 0', axis=1)
        )
        self.biz_vectors = self._sql.createDataFrame(
                read_json(self._sc, self.cfg['mdl_file']+'.features')\
                    .map(lambda r:
                        Row(biz_id=r['biz_id'], biz=r['biz'],
                            features=SparseVector(*r['features']))
                    )
            )
        # load avgs
        with open(self.cfg['mdl_file']+'.avgs', "r") as buff:
            mdl_ = json.loads(buff.read())
        self.biz_avg = mdl_['business_avg']
        self.user_avg  = mdl_['user_avg']
        self.user_map  = mdl_['user_map']

    def compute_avgs(self, data):
        """ Compute Business and User Avgs

             Params:
            ----
            data: pyspark.rdd
                Review JSON RDD
                Format: [
```

```
                {'business_id':x, 'user_id':x, 'text':x},
                ...
            ]

        Returns:
        -----
        (dict, dict)
            (User Avgs, Business Avgs)
    """
    user_avg = data.map(lambda x: (x['user_id'], x['stars']))\
                   .groupByKey().mapValues(mean)\
                   .collectAsMap()
    buss_avg = data.map(lambda x: (x['business_id'], x['stars']))\
                   .groupByKey().mapValues(mean)\
                   .collectAsMap()
    log("Got Business and User rating averages")
    return user_avg, buss_avg

def train(self, data):
    """ Training method

        Params:
        -----
        data: pyspark.rdd
            Review JSON RDD
            Format: [
                {'business_id':x, 'user_id':x, 'text':x},
                ...
            ]
    """
    user_avg, biz_avg = self.compute_avgs(data)
    b_candidates, b_vectors, ratings, usr_map = self.prepare(data)
    b_weights = self.compute_weights(b_candidates, b_vectors)
    self.save(b_weights, b_vectors, user_avg, biz_avg, usr_map)

def get_biz_nn(self, data):
    """ Get Business Nearest Neighbors

        Params:
        -----
        data: pyspark.rdd
            Test Review JSON RDD
            Format: [
                {'business_id':x, 'user_id':x, 'text':x},
                ...
            ]

        Returns:
        -----
        pyspark.rdd
            Format: [
                Row(biz=str,user=str, neighs={b: w, ...} ),
                ...
            ]
    """
    _b_wgts = pd.concat([
        self.icf_weights.rename(columns={'b2':'b', 'b1':'k'}),
        self.icf_weights.rename(columns={'b1':'b', 'b2':'k'})
        ]).groupby('b')\
        .apply(lambda r: {j:w for j,w in zip(r['k'], r['w'])})\
        .to_dict()
    _k_n = self.k_n
```

```
        def _get_neighs(b):
            return sorted(_b_wgts.get(b, {}).items(),
                          key=lambda i: i[1],
                          reverse=True)[:_k_n]

        neighs = data.map(lambda x: Row(
                                  biz=x['business_id'],
                                  user=x['user_id'])
                          )\
                     .map(lambda r: Row(
                             biz=r.biz,
                             user=r.user,
                             neighs=_get_neighs(r.biz)
                     ))
        return neighs

    def compute_score(self, test, data):
        """ Compute weighted average from neighbors

            Params:
            -----
            data: pyspark.rdd
                Format: [
                    Row(biz=str,user=str, neighs=[(idx, {'b':str, 'w':float}),..] ),
                    ...
                ]

            Returns:
            ----
            pyspark.rdd
                Format: [
                    Row(biz=str,user=str, score=float ),
                    ...
                ]
        """
        biz_test = set(test.map(lambda x: x['business_id']).distinct().collect())
        _inv_umap = {v:k for k,v in self.user_map.items()}
        # review if need to filter .rdd.map(lambda r: (r.biz,
r.features)).filter(lambda r: r[0] in biz_test)\
        _usr_rates = self.biz_vectors\
                .rdd.map(lambda r: (r.biz, r.features))\
                .mapValues(lambda fts: {_inv_umap[ix]:iv \
                                for ix, iv in zip(fts.indices, fts.values)})\
                .flatMap(lambda e: [(_u, (e[0], _r)) for _u,_r in e[1].items()])\
                .groupByKey().mapValues(dict).collectAsMap()
        log("Computing scores...")
        # Join and compute score
        def _score(n, r):
            num_, den_, cnt = 0., 0., 0
            for n_i, w_i in n:
                if n_i in r:
                    num_ += w_i * r[n_i]
                    den_ += abs(w_i)
                    cnt += 1
            if cnt == 0:
                # Cold Start strategy
                return 2.5
            return num_ / den_

        pred_scores = data.map(lambda r: (
                                r.biz,
                                r.user,
```

```
                                _score(r.neighs, _usr_rates.get(r.user, {})))
                            ).collect()
        return pred_scores

    def predict(self, test, outfile):
        """ Prediction method

            test: pyspark.rdd
                Test Review JSON RDD
                Format: [
                    {'business_id':x, 'user_id':x, 'text':x},
                    ...
                ]
            outfile: str
                Path to output file
        """
        # Find K-neighbors from business
        neighs = self.get_biz_nn(test)
        # Compute prediction score
        preds_ = self.compute_score(test, neighs)
        with open(outfile, 'w') as of:
            for pv in preds_:
                of.write(json.dumps({
                    "user_id": pv[1], "business_id": pv[0],
                    "stars": pv[2]
                })+"\n")
```

Вміст файлу train.py (основний файл для навчання моделі):

```
""" Yelp Recommender training module
"""
import sys
import time

from pyspark import SparkConf, SparkContext

from config.config import *
from models import models
from utils.misc import log, read_json


def create_spark():
    """ Method to create Spark Context

        Returns:
        -----
        sc : pyspark.SparkContext
    """
    conf = SparkConf()\
        .setAppName(APP_NAME)\
        .setMaster("local[4]")\
        .set("spark.executor.memory", "4g")\
        .set("spark.executor.cores", "4")\
        .set("spark.driver.cores",  "2")\
        .set("spark.driver.memory", "2g")
    sc = SparkContext(conf=conf)
    return sc


if __name__ == '__main__':
```

```
log(f"Starting {APP_NAME} training ...")
st_time = time.time()
# load config
cfg = load_conf()
log(f"Using {cfg['class']}")
# create spark
sc = create_spark()
# Load training data
training = read_json(sc, cfg['training_data'])
# Init model
model = models[cfg['class']](sc, cfg)
# Start training
model.train(training)
log(f"Finished training in {time.time()- st_time }")
```