Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Системотехніки
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розробка рекомендаційної системи з розподіленою обробкою даних на основі метода колаборативної фільтрації
(тема)

Виконав: студент 2 курсу, групи ІТПм-18-1
Російчук Д.О.
(прізвище, ініціали)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Інформаційні технології проектування
(повна назва освітньої програми)
Керівник доц. Імангулова З.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри СТ _____ проф. Гребеннік І.В.
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра системотехніки
Освітньо-кваліфікаційний рівень другий (магістерський)
Спеціальність 122 Комп'ютерні науки
Тип програми освітньо-професійна

Освітня програма Інформаційні технології проектування

ЗАТВЕРДЖУЮ:
Зав. кафедри СТ _____
(підпис)
"____" _____ 2019 р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Російчуку Данилу Олексійовичу
(прізвище, ім'я, по батькові)

1.Тема роботи (проекту) Розробка рекомендаціної системи з розподіленою обробкою даних на основі метода колаборативної фільтрації

затверджена наказом по університету від "04" листопада 2019 р. № 1627 Ст
2. Термін здачі студентом закінченого роботи 12.12.2019

3. Вихідні дані до роботи *Перелік використовуваних програмних засобів: ОС Ubuntu 18.04, середовище розробки Intellij Idea, локальний сервер*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) _____
4 Вступ. 4.1 Огляд і аналіз сучасного стану проблеми, що розглядається. 4.2 Аналіз методів колаборативної фільтрації на основі обчислень у пам'яті. 4.2.1 Опис основних підходів на основі обчислень у пам'яті. 4.2.2 Порівнення методів "користувач-користувач" та "продукт-продукт". 4.2.3 Складність та побічні ефекти методів основанних на обчислені у пам'яті. 4.3 Аналіз методів колаборативної фільтрації на основі моделі. 4.3.1 Матрична факторизація. 4.3.2 Алгоритм SLOPE ONE 4.3.3 Зважений SLOPE ONE 4.3.4 Двополюсний SLOPE ONE 4.4 Постановка задачі. 4.4.1 Вихідна інформація 4.4.2 Вхідна інформація 4.4.3 Визначення основних варіантів використання клієнтського додатуку 4.5 Розробка інформаційного забезпечення. 4.5.1 Сховища даних. 4.5.2 Опис архітектури системи. 4.6 Розробка програмного забезпечення 4.7 Висновки. 4.8 Перелік джерел посилання

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, схем, плакатів) *5.1 Приклади рекомендацій, створені сторонніми сервісами (2 аркуши А4). 5.2 Методи колаборативної фільтрації (2 аркуши А4). 5.3 Порівняння роботи методів (1 аркуш А4). 5.4 Процес матричної факторизації (2 аркуши А4). 5.4 Ілюстрація роботи алгоритмів (1 аркуш А4). 5.5. Use-case діаграми клієнтського додатку (1 аркуш А4). 5.6 Діаграма архітектури системи (1 аркуш А4). 5.7. Приклади роботи з Apache Spark (2 аркуши А4). 5.8 Створення наборів даних для тестування (2 аркуши А4). 5.9. Результати роботи рекомендаційної системи (2 аркуши А4).*

6. Консультанти розділів роботи

Найменування Розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Розділи спеціальної частини	доц. Імангулова З.А.		

КАЛЕНДАРНИЙ ПЛАН

Номер	Назва етапів роботи (проекту)	Термін виконання етапів роботи (проекту)	Примітка
1	*Отримання завдання атестаційної роботи*	01.09.2019	виконано
2	*Аналіз завдання, літератури та аналогів з теми атестаційної роботи*	03.09.2019 – 25.09.2019	виконано
3	*Вибір засобів для розробки, технічних вимог до програми*	25.09.2019 – 30.09.2019	виконано
4	*Структурне проектування*	30.09.2019 – 20.10.2019	виконано
5	*Вибір середовища розробки програми*	*20.10.2019 – 25.10.2019*	виконано
6	*Розробка програми*	*25.10.2019 – 15.11.2019*	виконано
7	*Тестування програми*	*15.11.2018 – 17.11.2019*	виконано
8	*Оформлення пояснювальної записки та програмної документації*	*17.11.2019 – 28.11.2019*	виконано
9	*Оформлення графічної частини та презентаційних матеріалів*	*30.11.2019*	виконано
10	*Представлення атестаційної роботи ДЕК*	*16.12.2019*	виконано

Дата видачі завдання _____

Студент _____
(підпис)

Керівник роботи _____ доц. Імангулова З.А.
(підпис) (посада, прізвище, ім'я, по батькові)

Атестаційна робота оформлена у відповідності до вимог діючих стандартів та методичних вказівок.

Матеріали атестаційної роботи не містять відомостей, що заборонені для опублікування у відкритих виданнях.

Попередній захист проведено.

Керівник атестаційної роботи Імангулова З.А.

РЕФЕРАТ

РЕКОМЕНДАЦІЙНІ СИСТЕМИ, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, APACHE SPARK, PYSPARK, МАТРИЧНА ФАКТОРИЗАЦІЯ, ALS

Атестаційна робота містить: 95 сторінок, 23 рисунків, 7 таблиць, 3 додатки, 25 джерел; графічний матеріал атестаційної роботи містить 19 плакатів.

Предмет дослідження – рекомендаційна система основана на колаборативній фільтрації.

Мета роботи – розробка рекомендаційної системи, знатної працювати на великих обсягах даних..

Методи розробки – методи об’єктно-орієнтованого і структурно-функціонального аналізу і проектування автоматизованих систем.

Засоби розробки програмного забезпечення – середовище розробки IntelliJ IDEA 2019.1, мова програмування Java та Python. В якості сховищ для зберігання інформації використовуються СУБД PostgreSQL та MongoDB.

Результати роботи – рекомендаційна система для підбору музичних виконавців для користувачів, здатна функціонувати з великими обсягами даних.

Область застосування – розроблений програмний засіб призначений для використання в сервісах надання контенту або магазинах.

ABSTRACT

RECOMMENDER SYSTEMS, COLLABORATIVE FILTERING, APACHE SPARK, PYSPARK, MATRIX FACTORIZATION, ALS

Explanatory note: 95 p., 23 fig., 7 tables, 3 applications, 25 sources, graphic material 19 p.

The purpose of the work is to develop a recommender system based on collaborative filtering that is able to process high amounts of data.

Methods of development - methods of object-oriented and structural-functional analysis and design of automated systems, analysis and use of algorithms.

Software development tools - IntelliJ IDEA 2019.1 development environment, Java and Python programming languages. PostgreSQL and MongoDB are used as data warehouse.

The results of the work - a recommender system that will recommend new musical artists to a user based on their listening history.

Scope - The developed software tool is intended for use in a media service providers or e-commerce.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

Collaborative filtering – колаборативна фільтрація;
Content-based filtering – фільтрація на основі контенту;
Memory based – заснована на обчислені у пам’яті;
БД – база даних;
Model based – фільтрація заснована на моделі;
MVC – Model-View-Controller – архітектурний шаблон;
MVVM – Model-View-ViewModel – архітектурний шаблон.

ВСТУП

Проблема створення рекомендацій для користувача є актуальною для будь-яких систем, що базуються на продажу або наданні доступу до контенту кінцевому користувачу, наприклад фільмів, музики, книг, новин або веб-сайтів. Такі сервіси як «Youtube», інтернет магазин «Amazon», сервіс відеострімінгу «Netflix» та музикальний сервіс «Last.fm» захопили значну частинку ринку саме завдяки вдалому використанню рекомендаційних систем. Ця галузь почала бурхливо розвиватися із зростанням кількості популярних інтернет сервісів у два останніх дисятиріччя.

Раніше для формування рекомендацій використовували список найбільш популярних продуктів, але с часом такі рекомендації почали витіснятися таргетованими (цільовими) пропозиціями: користувачам пропонуються не просто популярні пропозиції, а ті продукти, що напевно мають сподобатися саме їм. Для них життєво важливо надавати можливі варіанти, які або доповнюють товари (у випадку магазинів), або ті, що мають схожу тематику або відгуки, тому що від цього безпосередньо залежить їх прибуток.

У загальному розумінні рекомендаційні системи являють собою алгоритми направлені на навіювання важливих елементів для користувачів (фільми для перегляду, книги або тексти для читання, продукти для покупки або щось інше в залежності від індустрії). У деяких індустріях впровадження рекомендаційних систем є дійсно критичним, тому що вони можуть згенерувати значну частину доходу у разі їх ефективного використання та надати перевагу над конкурентами.

Метою даної роботи є дослідження існуючих методів створення рекомендаційних систем для великого обсягу даних.

Поставлена мета роботи обумовила наступні завдання дослідження:

– огляд існуючух методів створення рекомендаційних систем та їх аналіз;

– аналіз можливості застосування розглянутих методів для обробки великих обсягів даних;

– розробка рекомендаційної системи, що здатна оброблювати великі обсяги даних з використанням розглянутих методів;

– розробка веб-сервісу для однієї з таких предметних областей;

– реалізація та впровадження розробленої рекомендаційної системи до створеного сервісу.

Об’єктом дослідження є веб-сервіси, що базуються на наданні доступу до медіа-контенту або продажу товарів.

Предметом дослідження є алгоритми та методи створення рекомендацій та можливість застосування розподіленої обробки даних для підвищення їхньої ефективності.

Методи дослідження. Досягнення мети атестаційної роботи базується на комплексному використанні методів теорії фільтрації інформації, що стосуються створення рекомендаційних систем.

1 **ОГЛЯД І АНАЛІЗ СУЧАСНОГО СТАНУ ПРОБЛЕМИ, ЩО РОЗГЛЯДАЄТЬСЯ**

1.1 Аналіз рекомендаційних систем

Рекомендаційні системи являють собою підклас систем фільтрування інформації, направлені на передбачення рейтингу або вибору продукту користувачем. В загальному сенсі рекомендаційні системи це алгоритми, направлені на пропонування важливих для користувача продуктів. Існують дві основні категорії методів для вирішення цієї проблеми: методи колаборативної фільтрації та методи, основані на контенті.

Алгоритми колаборативної фільтрації базуються на припущені, що користувачі, які мали схожі вподобання в минулому будуть мати схожі вподобання і в майбутньому та їм подобатимуться елементи, схожі на ті, що побалися їм в минулому. Для створення нових рекомендацій алгоритми колаборативної фільтрації базуються лише на попередній інформації про взаємодії між користувачами та елементами. Такі взаємодії зберігаються у так званій «матриці взаємодій користувачів до елементів». Ці минулі взаємодії між користувачем та елементом достатні для того щоб знаходити схожих користувачів та/або схожі елементи та створювати передбачення основані на оцінках їхньої схожості [1].

Основна перевага методів колаборативної фільтрації полягає в тому, що вони не потребують жодної додаткової інформації про користувачів та продукти, тому можуть бути використані майже у будь-якій ситуації [2]. Більш того, чим більше користувачі взаємодіють з продуктами, тим більш точними є нові рекомендації: для фіксованого набору користувачів і продуктів нові записані взаємодії з часом дають нову інформацію і роблять систему все більш ефективною.

Методи, основані на контенті, напроти, базуються на використанні додаткової інформації про користувачів та/або продукти (наприклад вік, стать або професію або іншу персональну інформацію). Основна ідея цих методів полягає в створенні моделі яка базується на наявних властивостях та пояснює спостережені взаємодії. Методи, основані на контенті мають деяки переваги перед методами колаборативної фільтрації, але необхідність створення моделі на основі додаткових даних робить їх значно менш універсальними [1].

В атестаційній роботі для створення рекомендаційної системи було обрано підхід на основі колаборативної фільтрації. Такі сервіси як «Youtube», інтернет магазин «Amazon», сервіс відеострімінгу «Netflix» та музикальний сервіс «Spotify» вдало використовують методи колаборативної фільтрації. На рисунках 1.1 та 1.2 наведено приклад рекомендації музичних альбомів, створений сервісом «Spotify» та приклад рекомендації фільмів та серіалів, створений сервісом «Netflix».

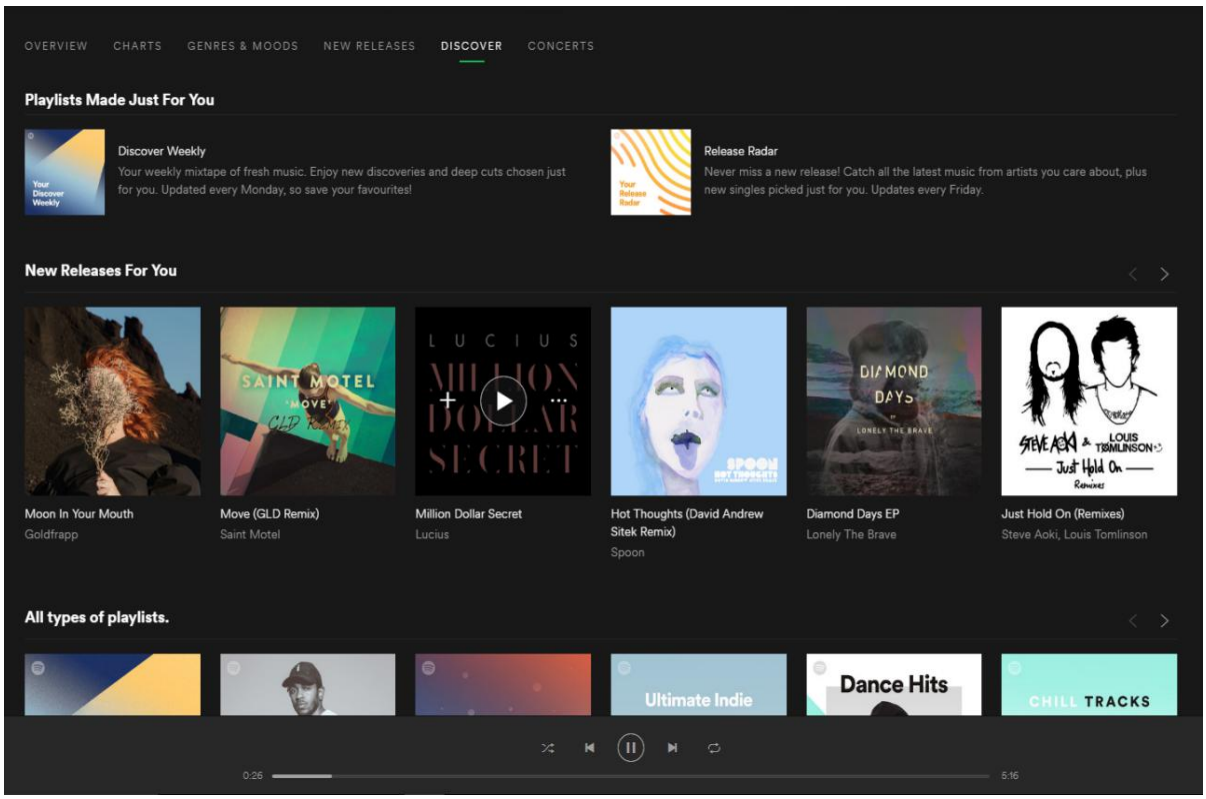

Рисунок 1.1 — Приклад рекомендації, створеної сервісом «Spotify»

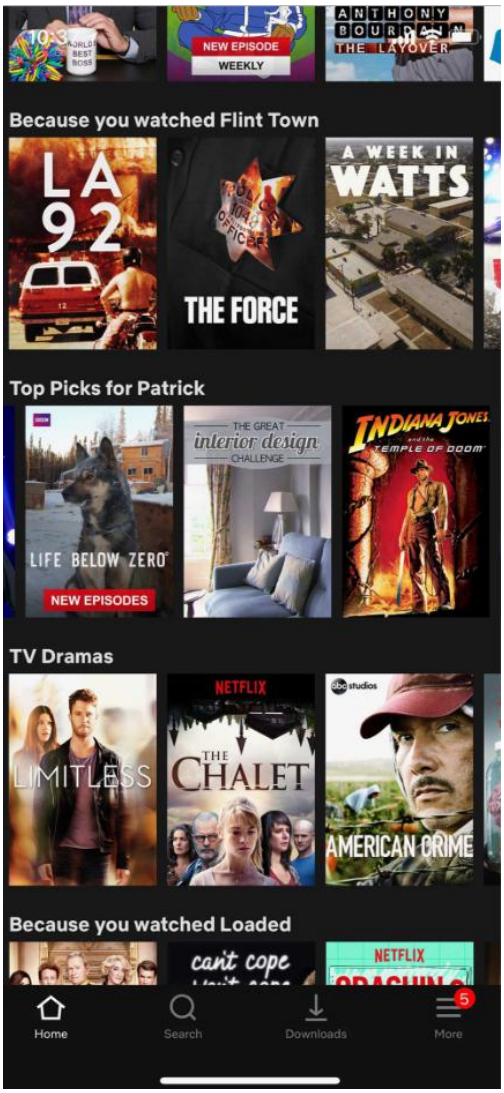

Рисунок 1.2 — Приклад рекомендації, створеної сервісом «Netflix»

До основних способів колаборативної фільтрації відносяться: користувач-користувач, продукт-продукт та матрична факторизація.

Алгоритми колаборативної фільтрації поділені на дві підкатегорії які загалом звуться «memory based» та «model based» [1]. Алгоритми першої категорії працюють зі значеннями записаних взаємодій напряму, без будь-якої моделі, та базуються на пошуку найближчих сусідів (наприклад, знайти найближчих користувачів для якогось користувача та рекомендувати їм найбільш популярні серед них елементи). Підхід оснований на моделі передбачає наявність «генеративної моделі», яка пояснює взаємодію

користувача та елемента та намагається знайти залежність для формування нових передбачень.

Тому що колаборативна фільтрація основувається на попередніх взаємодіях для створення рекомендацій, вона має суттєвий недолік - так звану проблему «холодного старту» [1]: неможливо рекомендувати щось новому користувачу або рекомендувати новий продукт. Також дуже складно створювати рекомендації, якщо багато користувачів або продуктів мають недостатню кількість взаємодій для ефективного формування рекомендації. Цього недоліку можна уникнути декількома способами: рекомендувати випадкові продукти новим користувачам або нові продукти випадковим користувачам (так звана «випадкова стратегія»), рекомендувати популярні продукти новим користувачам або нові продукти найбільш активним користувачам («стратегія максимального очікування»), рекомендувати набір різноманітних продуктів новим користувачам або нові продукти набору різноманітних користувачів («розвідувальна стратегія»); використання не-колаборативного методу для початку існування користувача або продукта.

Одним з найважливіших факторів у виборі методу колаборативної фільтрації для застосування є обсяг наявних даних о взаємодіях між користувачами та продуктами, оскільки всі ці методи так чи інакше мають використовувати ці дані для обчислень у пам’яті або при формуванні спеціалізованої моделі.

2 **АНАЛІЗ МЕТОДІВ КОЛАБОРАТИВНОЇ ФІЛЬТРАЦІЇ НА ОСНОВІ ОБЧИСЛЕНЬ У ПАМ'ЯТІ**

2.1 Опис основних підходів на основі обчислень у пам'яті

Першим буде розглянутий метод так званий «користувач-користувач». Для створення нових рекомендацій для користувача цей метод намагається приблизно ідентифікувати користувача із найбільш схожим «профілем взаємодій» (найближчі сусіди) для пропонування продуктів що є найбільш популярними серед цих сусідів (та ті що є новими для користувача). Цей метод фокусується на користувачах тому, що він представляє користувачів в залежності від їх взаємодій з продуктами та обчислює дистанцію між ними.

Представимо що ми хочемо створити рекомендацію для обраного користувача. По перше, кожний користувач може бути представлений як його вектор взаємодій з різними продуктами (його ліній у матриці взаємодій). Потім, ми можемо обчислити меру схожості між обраним користувачем та кожним іншим. Ця міра схожості полягає у тому, що два користувача зі схожими діями на тих самих продуктах мають бути визнані як схожі. Коли міри схожості для кожного користувача обчислені система може зберігати k-схожих-сусідів для користувачів та пропонувати їм найбільш популярні продукти серед них, обираючи лише з тих, з якими обраний користувач ще не робив взаємодій [3]. Ілюстрація роботи цього методу наведена на рисунку 2.1.

Треба зауважити, що при обчислені міри схожості кількість «спільних дій» має бути обрана обережно, бо у багатьох ситуаціях, наприклад, коли якийсь користувач має лише одну спільну дію з обраним буде мати 100% схожість і буде обраним ближчим, ніж інший, що має 100 спільних взаємодій і погоджуються «лише» у 98%. Таким чином, два користувача вважаються схожими якщо вони взаємодіяли з багатьма продуктами схожим чином (схожі оцінки, схожий час просмотру та ін.) [4].

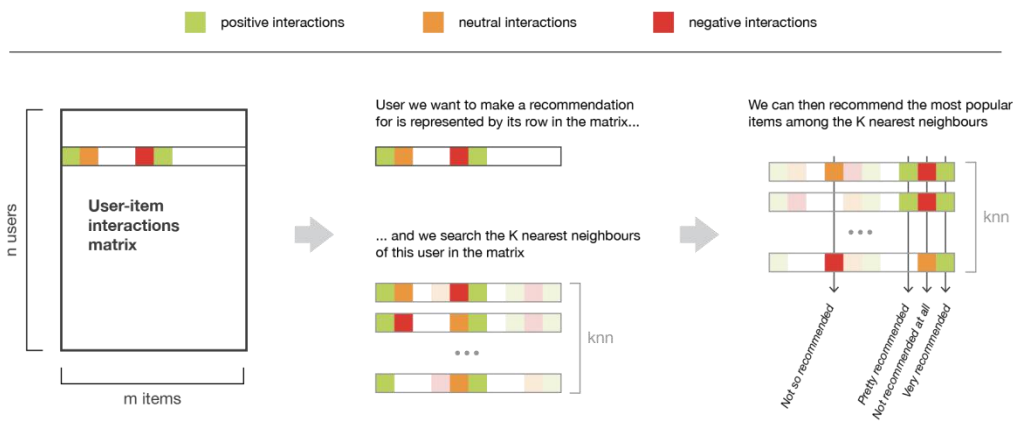

Рисунок 2.1 — Ілюстрація методу користувач-користувач

Наступним буде розглянутий так званий метод «продукт-продукт». Цей метод полягає у тому, що для створення нових рекомендацій потрібно знайти продукти схожі на ті, з якими користувач вже «позитивно» взаємодіяв. Два продукти вважаються схожими якщо більшість користувачів, що взаємодіяли з кожним з них робили це схожим чином (однаково оцінювали, переглядали приблизно однакову кількість разів та ін.). Цей метод є орієнтованим на продукт, тому що він представляє продукти в залежності від дій користувачів стосовно цього продукту та використовує дистанцію між продуктами для створення рекомендацій [5]. Ілюстрація роботи методу «продукт-продукт» наведена на рисунку 2.2.

Представимо що нам потрібно зробити рекомендацію для заданого користувача. Спочатку ми розглядаємо продукт, який найбільше сподобався цьому користувачу і представляємо його як вектор взаємодій з кожним користувачем (його колонка у матриці взаємодій). Потім, ми обчислюємо схожості між «найкращим продуктом» та усіма іншими продуктами. Після обчислення мір схожості к-найближчих-сусідів до обраного продукту зберігаються і з‘являються у рекомендаціях. Для того, щоб робити більш релевантні рекомендації можливо використати цей підхід з більш ніж одним

продуктом, обравши замість одного якусь кількість «найкращих» продуктів. Цей спосіб дозволяє рекомендувати продукти, схожі на ці обрані.

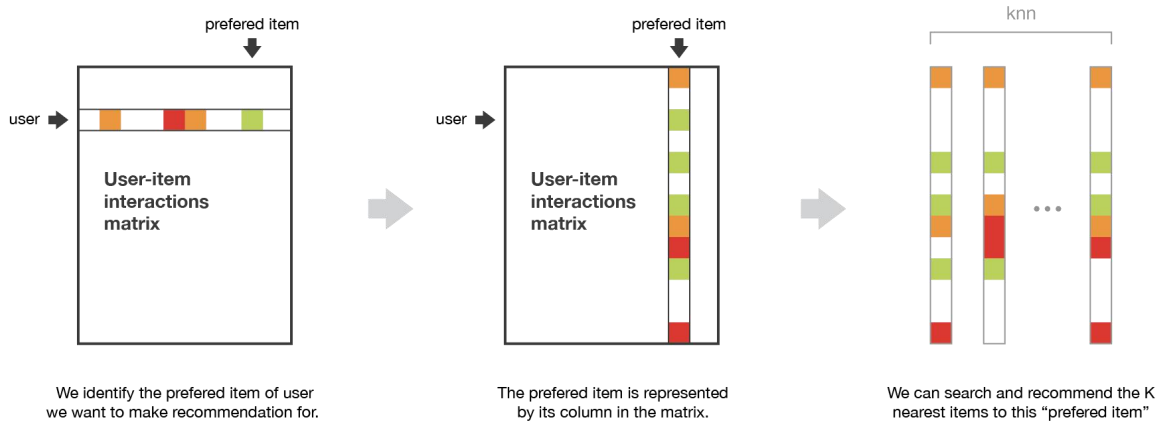

Рисунок 2.2 — Ілюстрація методу продукт-продукт

2.2 Порівняння методів «користувач-користувач» та «продукт-продукт»

Метод користувач-користувач базується на пошуку схожих користувачів на основі взаємодій з продуктами. Якщо кожен користувач взаємодіяв лише з декількома продуктами, це робить метод достатньо чутливим до будь-яких збережених взаємодій (висока дисперсія). З іншого боку, якщо фінальна рекомендація базується лише на збережених взаємодіях користувачів із схожими інтересами, це дозволяє отримати більш персональний результат (низька похибка).

Метод користувач-користувач, навпаки, базується на пошуку схожих продуктів на основі взаємодій користувача з продуктом. Загалом, якщо багато користувачів взаємодіяли з продуктом, пошук сусідів є значно менш чутливим до одиничних взаємодій (низька дисперсія). Аналогічно, дії будь-яких користувачів (навіть тих, що достатньо відрізняються від обраного) розглядаються у рекомендації, роблячи цей метод менш персональним (більша похибка). Таким чином, цей підхід є менш персональним ніж підхід

«користувач-користувач», але є більш надійним. Візуальне порівняння роботи цих методів наведене на рисунку 2.3.

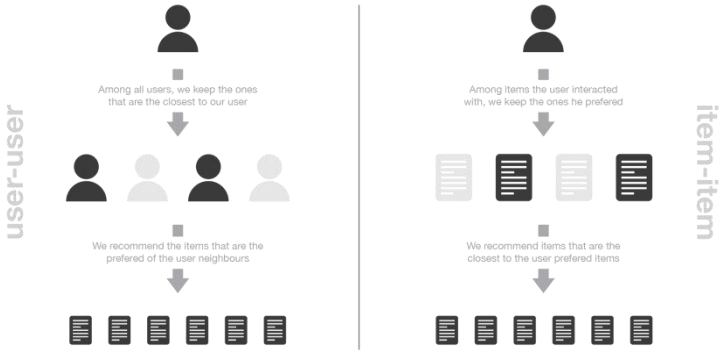

Рисунок 2.3 — Ілюстрація порівняння роботи методів

Найбільш поширеною метрикою дистанції, що використовується у методах, основаних на пам’яті є «косинусна схожість». Ця метрика являє собою геометричне представлення рядку для користувача або колонки для продукту як вектору. Для підходу що базується на користувачах схожість двох будь-яких з них вимірюється як косинус між відповідними їм векторами [3]:

$$sim(u,u') = \cos(\theta) = \frac{r_u \cdot r_{u'}}{\| r_u \| \| r_{u'} \|} = \sum_i \frac{r_{ui} r_{u'i}}{\sqrt{\sum_i r_{ui}^2} \sqrt{\sum_i r_{u'i}^2}} \qquad (2.1)$$

де r_u та $r_{u'}$ — відповідні вектори для порівнюваних продуктів u та u' ;

r_{ui} та $r_{u'i}$ — компоненти цих векторів.

Завдяки цьому можна передбачити рейтинг користувача u’ для і-того продукта на основі зваженої суми рейтингу всіх інших користувачів, де значення схожості між обраним користувачем та всіма іншими використовується як вага:

$$\hat{r}_{ui} = \sum_{u'} sim(u,u') r_{u'i} \qquad (2.2)$$

Також має сенс зробити нормалізацію отриманих значень рейтингу за допомогою сумарної кількості всіх оцінок [5]:

$$\hat{r}_{ui} = \frac{\sum_{u'} sim(u,u') r_{u'i}}{\sum_{u'} |sim(u,u')|} \qquad (2.2)$$

2.3 Складність та побічні ефекти методів основаних на обчислені у пам’яті

Одним з найбільших недоліків колаборативної фільтрації на основі обчислень у пам’яті є те, що вона не масштабується легко: створення нової рекомендації може займати надзвичайно багато часу для великих систем. У разі використання методів методів основаних на обчислені у пам’яті для вирішення проблеми холодного старту часто використовують алгоритми пошуку найближчих сусідів, але через складність цих алгоритмів у найгіршому сценарії вони не можуть бути використані для значного обсягу даних. Однак, етап пошуку найближчих сусідів може бути оптимізований за допомогою використання приблизного методу найближчих сусідів або використання алгоритму, що враховує той факт, що матриця взаємодій є розрідженою замість використання звичайного методу к-найближчих-сусідів.

У більшості рекомендаційних алгоритмів обов’язково потрібно бути надзвичайно обережними для уникнення ефекту «багатий-стає-багатшим» для популярних продуктів та уникнення ситуації в якій користувач застряг у так званій «зоні обмеження інформації», тому ще це має величезний негативний

вплив на потенційних прибуток від продажу нового товару або нового медіа-контенту [6].

Інакше кажучи, система не повинна намагатися рекомендувати лише популярні продукти все більше і більше, як і не повинна рекомендувати користувачам лише надзвичайно близькі продукти до тих, що вони вже оцінили позитивно без жодного шансу для них дізнатись про нові продукти які також мають сподобатися їм (усі ці продукти «недостатньо близьки» для того, щоб бути рекомендованими).

Якщо, як було згадано, ці проблеми можуть виникати у більшості рекомендаційних алгоритмів, це особливо стосується саме колаборативних методів на основі обчислень в пам’яті. При відсутності моделі для упорядкування цей феномен спостерігається частіше, тому що використані у створенні алгоритму залежності мають не бути оптимальними у порівнянні з тими, що модель знайде самостійно.

3 **АНАЛІЗ МЕТОДІВ КОЛАБОРАТИВНОЇ ФІЛЬТРАЦІЇ НА ОСНОВІ МОДЕЛІ**

Колаборативні підходи, основанні на моделі залежать лише від інформації о взаємодіях користувачів та продуктів та припускають наявність моделі, яка пояснює ці взаємодії. Наприклад, алгоритм матричної факторизації складається у декомпозиції величезної та розрідженої матриці взаємодій користувача та продукта у добуток двох менших та щільних матриць: матриці користувач-фактор (складається з представлень користувачів) яка примножує матрицю фактор-продукт (складається з представлень продуктів). На відміну від алгоритмів, що базуються на обчисленнях у пам’яті, методи на основі моделі самостійно визначають характеристики для пошуку схожих елементів, що виключає можливість помилки при створенні алгоритму, але мають велику залежність від даних для навчання.

3.1 Матрична факторизація

Метод матричної факторизації заснований на припущенні, що існує низьковимірний простір характеристик, у якому можуть бути представлені як користувачі, так і продукти, а взаємодії між ними можуть бути отримані за допомогою скалярного добутку між відповідними векторами у цьому просторі.

Наприклад, представимо що існує матриця оцінок фільмів користувачами. Для моделювання взаємодій між користувачами і фільмами, припустимо що:

– існує деякий набір характеристик, що описують фільми;

– цей набір характеристик може бути використаний для ідентифікації фільмів;

– ці характеристики можуть бути використані для опису вподобань користувача (високі значення для характеристик що подобаються користувачі та низьки в протилежному випадку).

Однак, замість того, щоб явно задати ці характеристики для моделі, система повинна знайти ці характеристики самостійно та створити власні представлення як користувачів, так і продуктів. Оскільки вони є вивченими, а не заданими, витягнуті якості мають лише математичне значення та не мають жодної інтуїтивної інтерпретації, тому фактично не можуть бути зрозумілими людиною.

Однак, іноді виникаючи структури цього типу алгоритмів є надзвичайно близькими до інтуїтивної декомпозиції того, про що мала би подумати людина. Результатом цієї факторизації є те, що люди, схожі за перевагами та продукти, схожі за характеристиками мають близьке представлення у цьому просторі [7]. Візуалізація цього процесу представлена на рисунку 3.1:

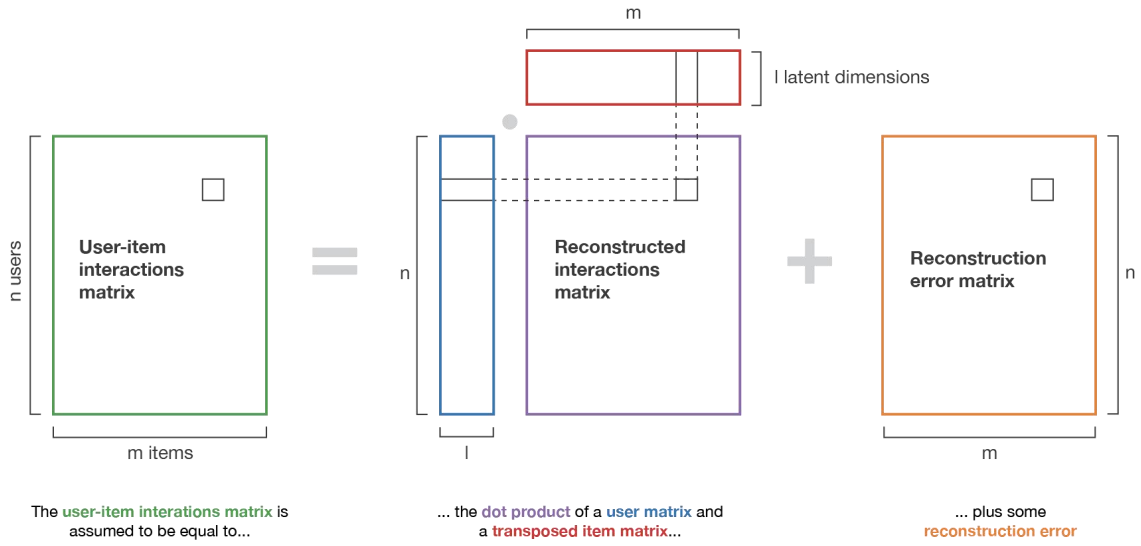

Рисунок 3.1 — Візуалізація процесу факторизації

Матрична факторизація використовує класичний ітеративний підхід, оснований на градієнтному спуску, який дозволяє отримувати факторизації для

дуже великих матриць без завантаження водночас всіх даних у пам’ять, що дозволяє уникнути головного недоліку методів, що базуються на обчисленнях у пам’яті.

Представимо матрицю взаємодій M розміром m на n рейтингу, в якій тільки деякі з продуктів були оцінені кожним користувачем (більшість взаємодій відсутні, щоб виразити недостачу рейтингу). Потрібно факторизувати таку матрицю наступним чином:

$$M \approx XY^T \tag{3.1}$$

де X - матриця користувачів, строки якої представляють n користувачів;

Y - матриця продуктів, строки якої представляють m продуктів:

$$\begin{aligned} user_i &\equiv X_i \quad \forall_i \in \{1,...,n\} \\ item_j &\equiv Y_j \quad \forall_j \in \{1,...,m\} \end{aligned} \tag{3.2}$$

де l - вимір у просторі, в якому користувач та продукт будуть представлені.

Таким чином, потрібно виконати пошук матриць X та Y, скалярний добуток яких буде наближеним до існуючих взаємодій. Позначимо пари (i, j), для яких відсутнє значення M_{ij} як E, таким чином задача пошуку X та Y представляється у вигляді:

$$(X,Y) = \underset{X,Y}{\operatorname{argmin}} \sum_{(i,j)\in E} \left[(X_i)(Y_j)^T - M_{ij}\right]^2 \tag{3.3}$$

Додавши регуляризацію, отримаємо:

$$(X,Y) = \underset{X,Y}{\operatorname{argmin}} \frac{1}{2} \sum_{(i,j)\in E} \left[(X_i)(Y_j)^T - M_{ij} \right]^2 + \frac{\lambda}{2} \left(\sum_{i,k} (X_{ik})^2 + \sum_{j,k} (Y_{jk})^2 \right) \quad (3.4)$$

Матриці X та Y можуть бути отримані завдяки процесу оптимізації градієнтного спуску, для якого справедливі дві речі:

– градієнт не має бути обчисленим за всіма парами E на кожному кроці, що надає можливість використати лише підмножину цих пар, що надає можливість для оптимізації;

– значення у X та Y не мають бути оновлені водночас, тому градієнтний спуск може бути виконаним роздільно для X та Y на кожному кроці (у цьому разі вважається, що одна матриця оптимізована та зафіксована для іншої перш ніж робити зворотню операцію на наступній ітерації) [8].

Після того, як матриця буде факторизована, кількість інформації для створення рекомендації значно зменшується: єдине, що можна зробити для отримання - помножити вектор користувача на будь-який векторний елемент, щоб оцінити відповідний рейтинг. Треба зауважити, що у цьому разі також можливо використовувати методи користувача-користувача та продукт-продукт з цими новими представленнями користувачів і елементів: (приблизний) пошук найближчих сусідів здійснюватиметься не за величезними розрідженими векторами, а над невеликими щільними, що робить деякі методи наближення більш простежуваними.

Щоб побачити, як матриця фактуризується, перше, що слід зрозуміти, це сингулярне розкладання значення (SVD). Виходячи з лінійної алгебри, будь-яка реальна матриця R може бути розкладена на 3 матриці U, Σ та V [9]. Нехай U - матриця функцій, що латерує користувачем, n × r, V - матрична латентна функція, Σ - діагональна матриця r × r, що містить сингулярні значення вихідної матриці, просто представляючи, наскільки важливою особливістю є прогнозування переваг користувача.

$$R = U \sum V^T$$
$$U \in IR^{n \times r}, \quad \sum \in IR^{r \times r}, \quad V \in IR^{r \times m} \tag{3.5}$$

Для сортування значень Σ шляхом зменшення абсолютного значення та укорочення матриці Σ до перших k розмірів (k сингулярних значень) можливо відтворити цю матрицю як матрицю A. Вибір k повинен переконатися, що A здатний захопити найбільшу кількість дисперсії в межах вихідної матриці R, так що A - наближення R, A ≈ R. Різниця між A і R - похибка, яка, як очікується, буде мінімізована. Візуалізація методу SVD представлена на рисунку 3.2.

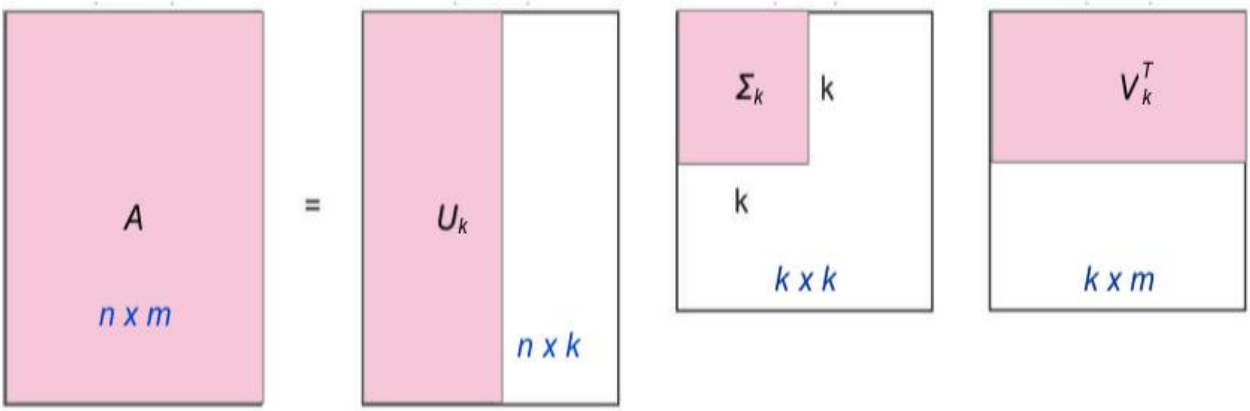

Рисунок 3.2 — Візуалізація методу SVD

Коли матриця R щільна, U і V можна легко аналізувати аналітично. Однак матриця рейтингів фільмів ймовірно буте розрідженною. Хоча існують деякі методи імпутації для заповнення пропущених значень, краще звернутися до іншого підходу. Замість факторизації R через SVD пошук U і V має бути виконаний безпосередньо за допомогою того, що коли U і V будуть помножені, отримана вихідна матриця R' є дейким наближенням до R і більше не э розрідженою матрицею. Це числове наближення, як правило, досягається за допомогою невід'ємної матричної факторизації для систем рекомендування, оскільки рейтинги не мають негативних значень [10].

Дивлячись на прогнозований рейтинг для конкретного користувача та елемента, елемент і позначається як вектор q_i, а користувач u позначається як вектор p_u, таким чином, що скалярний добуток цих двох векторів є прогнозованою оцінкою для користувача u на продукт і. Це значення представлене в матриці R у рядку u та стовпці і:

$$\text{Predicted Ratings}: r'_{ui} = p_u^T q_i \tag{3.5}$$

Оптимальні значення q_i і p_u знаходяться завдяки функції втрат, визначеної для мінімізації витрат на помилки:

$$\min_{q,p} \sum \left(r_{ui} - p_u^T q_i\right)^2 + \lambda\left(\| p_u \|^2 + \| q_i \|^2\right) \tag{3.7}$$

де r_{ui} - справжні оцінки з оригінальної матриці елементів користувача.

Процес оптимізації полягає у пошуку оптимальної матриці P, складеної вектором p_u та матрицею Q, складеною вектором q_i, щоб мінімізувати помилку квадратної суми між передбачуваними рейтингами r_{ui} та справжніми оцінками r_{ui}. Крім того, необхідно добавити регуляризацію L2, щоб запобігти переналагодженню векторів користувачів та елементів. Також має сенс додати деяке зміщення, яке зазвичай містить 3 основні компоненти: середній рейтинг усіх елементів μ, різницю між середнім рейтингом елемента і та μ (зазначений як b_u), різниця між середнім рейтингом, наданим користувачами та рейтингом, наданим довільним корисутвачем (позначений як b_i)

$$\min_{p,q,b_u,b_i} \sum \left(r_{ui} - \left(p_u^T q_i + \mu + b_u + b_i\right)\right)^2 + \lambda\left(\| p_u \|^2 + \| q_i \|^2 + b_u^2 + b_i^2\right) \tag{3.8}$$

Оскільки матрична факторизація фактично представляє собою скалярний добуток двох масивів та всі її значення мають бути рівними або більшими за нуль, деякі алгоритми оптимізації вирішують цю проблему можливої негативної факторизації [12].

Альтернативна найменша площа (ALS) - один з них. Оскільки функція втрат в цьому випадку невипукла, немає можливості досягти глобального мінімуму, хоча вона все ще може досягти великого наближення шляхом пошуку локальних мінімумів. Альтернативно найменшим квадратом є утримування постійної матриці фактора користувача, коригування матриці коефіцієнта елемента шляхом виведення похідних функції втрати та встановлення її рівним 0, а потім встановлення постійної матриці коефіцієнта елемента під час коригування матриці фактора користувача. Як і всі методи матричної факторизації, Цей процесс має бути повторeним, перемикаючи та регулюючи матриці вперед і назад до зближення. ALS широко використовується у бібліотеках машиного навчання та аналізу даних.

3.2 Алгоритм SLOPE ONE

Алгоритм «SLOPE ONE» вперше запропонований доктором Даніелем Леміром у 2005 р. [7]. Основна його форма предиктора - $f(x) = x + b$ Припустимо, що U_i і U_j являють собою набори користувачів, які оцінюють елемент і та елемент j відповідно, процес обчислення R_{U_j} можна розділити на два етапи:

– припустимо, що рейтинги елемента і та j від користувача v відповідають лінійному співвідношенню:

$$R_{v_j} = R_{v_i} + d_{ji} \tag{3.9}$$

де d_{ji} можна сформулювати як:

$$d_{ji} = \sum R_{v_j} - R_{v_i} \mid U_i \cap U_j \mid v \in U_i \cap U_j \qquad (3.10)$$

– припустимо, що рейтинг елемента i від користувача u представлений r_{u_i}, ми можемо скористатись рівнянням для прогнозування рейтингу від u до j як $R_{u_j} = r_{u_i} + d_{ji}$. З вищенаведених кроків ми бачимо, що алгоритм Slope One - це додаткова модель, яка має прості, ефективні функції і може продовжувати самонавчання з приєднанням користувачів та елементів [8].

Алгоритм Slope One може певною мірою знизити розрідженість матриці та підвищити точність рекомендацій, але витрата часу збільшується зі збільшенням системи.

Більше того, процес прогнозування враховує лише подібність між уподобаннями користувачів, не враховує схожість між атрибутами елемента та призводить до відсутності кореляції прогнозу. Отже, у цій роботі на основі зваженого алгоритму Slope One, вводячи особливості атрибутів елемента, ми вибираємо лише рейтингові дані для елементів, всеосяжна схожість яких значно вища для об'єкта, щоб виключити втручання з боку сусідів; крім того, алгоритм може зменшити розмір сусідів елементів, зменшивши трудомісткість. Матриця атрибутів елемента може бути записана внизу.

Рядок матриці представляє елемент, стовпець представляє атрибут, значення 1 являє собою елемент, має атрибут і 0 представляє, що не має. Так що кожен елемент i можна позначити як:

$$i = \{p_1, p_2, ..., p_l\} \qquad (3.11)$$

де $p_t \in \{0,1\}, 1 \le t \le l$.

Процес нахилу першого на основі подібності атрибутів елемента складається з наступного:

– обчислити подібність атрибута між елементом i та j відповідно до рівняння, позначається як $sims(i, j)$. З формули видно, чим більше однакових атрибутів, тим вище схожість.

– обчислити схожість рейтингу між елементами на основі матриці R, що позначаються як $simr(i, j)$.

– використовуючи результати, отримані на кроку 1 та кроку 2, для обчислення всебічної подібності $sim(i, j)$, сформульованої нижче:

$$sim(i, j) = \alpha sims(i, j) + (1 - \alpha) simr(i, j) \qquad (3.12)$$

– сортувати $sim(i, j)$ у порядку зменшення, обираючи найкращі елементи у якості сусідів.

– на підставі сусідів по пунктах застосувати зважений SLOPE ONE, після чого заповнити R прогнозованими значеннями.

SLOPE ONE бере до уваги як інформацію інших користувачів, які оцінили той самий предмет, так і інші елементи, оцінені тим самим користувачем [8]. Однак схеми також покладаються на точки даних, які не потрапляють ні в масив користувача, ні в масив елементів (наприклад, рейтинг користувача A на пункт 1), але, тим не менш, є важливою інформацією для прогнозування рейтингу. Значна частина підходу припадає на дані, які не враховуються. Зокрема, у прогнозування входять лише ті рейтинги користувачів, які оцінили якийсь звичайний предмет із передбачуваним користувачем, і лише ті рейтинги предметів, які оцінив також передбачуваний користувач рейтингів за нахилом однієї схеми. Формально, задавши два масиви

оцінювання v_i та w_i з $i = 1..n$, ми шукаємо найкращого предиктора виду $f(x) = x + b$ для прогнозу w від v шляхом мінімізації $\sum_i (v_i + b - w_i)^2$. Отримавши відносно b і встановивши похідну до нуля, отримаємо $b = \frac{\sum_i w_i - v_i}{n}$. Іншими словами, константа b повинна бути обрана як середня різниця між двома масивами. Цей результат мотивує наступну схему. Враховуючи навчальний набір χ, і будь-які два пункти j та i з рейтингами u_j та u_i відповідно в деякій оцінці користувача u (позначається як $u \in S_j, i (\chi)$), ми вважаємо середнє відхилення елемента i стосовно елемента j як:

$$dev_{j,i} = \sum_{u \in S_{j,i}(\chi)} \frac{u_j - u_i}{card\left(S_{j,i}(\chi)\right)} \qquad (3.13)$$

Треба зауважити, що будь-яка оцінка користувача u, яка не містить ні u_j, ні u_i не включається до підсумовування. Симетричну матрицю, визначену $dev_{j,i}$, можна обчислити один раз і швидко оновити, коли вводяться нові дані. Враховуючи, що $dev_{j,i} + u_i$— це передбачення для u_j, заданого u.

$$P(u)_j = \frac{1}{card\left(R_j\right)} \sum_{i \in R_j} \left(dev_{j,i} + u_i\right) \qquad (3.14)$$

де $R_i = \{i | i \in S(u), i \neq j, card(S_{j,i}(\chi)) > 0\}$ - це набір усіх відповідних елементів. Існує наближення, яке може спростити обчислення цього прогнозу. Для достатньо щільного набору даних, де майже всі пари предметів мають рейтинги, тобто де $card(S_{j,i}(\chi)) > 0$ майже для всіх i, j, більшість часу $R_i = S(u)$ *для* $j \in S(u)$ і $R_i = S(u) - \{j\}$ *для* $j \in S(u)$. Оскільки $\underline{u} = \Sigma_{i \in S(u)} \frac{u_i}{card(S(u))} \simeq \Sigma_{i \in R_j} \frac{u_i}{card(R_j)}$, ми можемо спростити формулу прогнозування схеми SLOPE ONE до

$$P^{S1}(u)_j = \bar{u} + \frac{1}{card(R_j)} \sum_{i \in R_j} dev_{j,i} \qquad (3.15)$$

Цікаво зауважити, що реалізація SLOPE ONE не залежить від того, як користувач оцінив окремі товари, а лише від середнього рейтингу користувача та від того, які саме товари оцінив користувач [8]. Ілюстрація роботи алгоритму SLOPE ONE наведена на рисунку 3.3.

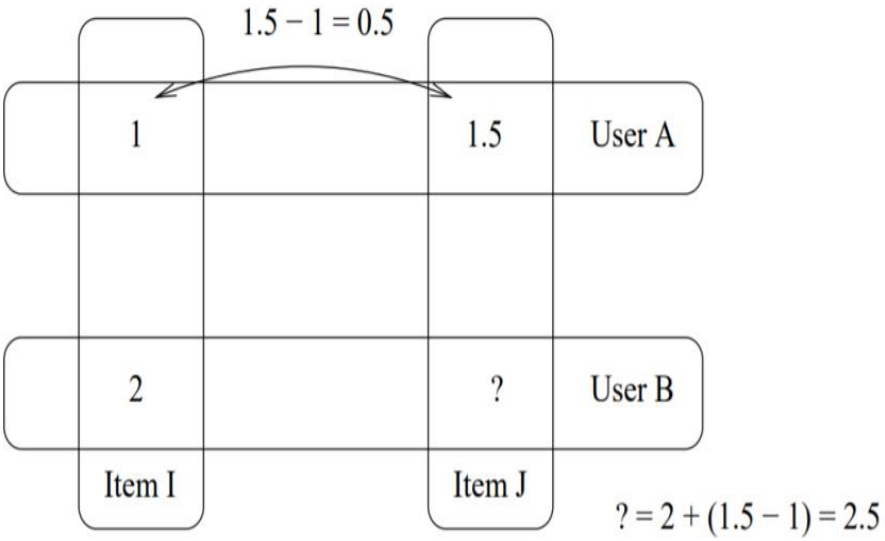

Рисунок 3.3 — Ілюстрація роботи алгоритму SLOPE ONE

Розглянемо обчислення SLOPE ONE на наступному прикладі, наведеному у таблиці 3.1.

Таблиця 3.1 – Приклад оцінок користувачів

Customer	Item A	Item B	Item C
John	5	3	2
Mark	3	4	-
Lucy	-	2	5

У цьому разі, середня різниця у рейтингу між елементами A та B дорівнює *(2+(-1))/2=0.5*. Звідси, у середньому, оцінка A вища за оцінку B на 0.5. Таким чином, якщо спробувати передбачити оцінку користувача Lucy для елементу A, отримаємо *2 + 0.5 = 2.5*. Також спробуємо передбачити її оцінку для елементу A використовуючи рейтинг елементу C: *5 + 3 = 8*. Якщо користувач оцінив декілька елементів, передбачення просто комбінуються з використанням зваженого середнього, де вага дорівнює кількості користувачів, що оцінили обидва елементи. На цьому прикладі, якщо обидва John та Mark оцінили A та B, то вага дорівнює 2, та тільки John оцінив A та C, тому вага дорівнює 1. Таким чином, передбачення оцінки A для користувача Lucy дорівнює $\frac{2*2.5+1*8}{2+1}=\frac{13}{3}=4.33$.

3.3 Зважений SLOPE ONE

Недоліком SLOPE ONE є те, що кількість рейтингових оцінок не береться до уваги. Інтуїтивно зрозуміло, що для прогнозування оцінки користувача A для продукту L, маючи його оцінки для продуктів J та K, якщо, наприклад, дві тисячі користувачів оцінили пару продуктів J та L та тільки двадцять оцінили пару K та L, оцінка користувачем A рейтингу продукту J швидше за все буде набагато кращим предиктором для продукту L ніж його оцінка продукту K [16].

Таким чином, зважений алгоритм SLOPE ONE визначається так:

$$P^{wS1}(u)_j = \frac{\sum_{i \in S(u)-\{j\}} (dev_{j,i} + u_i) c_{j,i}}{\sum_{i \in S(u)-\{j\}} c_{j,i}} \qquad (3.16)$$

3.4 Двополюсний SLOPE ONE

Розглянемо ще один шаблон оцінки рейтингу, коли зважження використовується для обрання найчастіше трапляючихся шаблонів оцінки замість нечасто трапляючихся шаблонів. Це може бути досягнуто за допомогою поділення формування прогнозу на дві частини. Використовуючи зважений алгоритм SLOPE ONE спочатку отримується одне передбачення з предметів, які сподобалися користувачам, і інше прогнозування, використовуючи предмети, які користувачам не сподобалися. Враховуючи рейтингову шкалу, скажімо, від 0 до 10, може бути доцільним використовувати середину шкали 5, як поріг, і сказати, що предмети, оцінені вище 5, подобаються, а ті, що оцінюються нижче 5 - ні. Цей підхід спрацює, якщо рейтинги користувачів розподіляються рівномірно, однак найчастіше це не так.

Оскільки алгоритм має підтримувати всі типи користувачів, включаючи збалансованих, оптимістичних, песимістичних та бімодальних, він застосовує середнє значення користувача як поріг між предметами що сподобились, та тими, які не подобаються. Наприклад, для оптимістичних користувачів, яким подобається кожен предмет, який вони оцінюють, вважається що їм не сподобаються предмети, оцінені нижче середнього рейтингу. Цей поріг гарантує, що алгоритм має достатню кількість елементів що подобаються та не подобаються для кожного користувача [17].

Як і у разі звичайного SLOPE ONE, прогноз для продукту J користувачем B базується на відхиленні від оцінки продукту I для користувачів, які оцінили обидва продукти. Двополярний SLOPE ONE обмежує набір рейтингів, який бере участь у прогнозуванні.

По-перше, щодо предметів, враховуються лише відхилення між двома вподобаними предметами або відхилення між двома предметами, які не подобаються.

По-друге, з точки зору користувачів, для прогнозування рейтингів для елемента J використовуються лише відхилення від пар користувачів, які оцінили як I, так і J та які поділяють подібний або неприязний елемент I [18].

Поділення кожного користувача на те, що йому подобається та на те, що не подобається ефективно подвоює кількість користувачів. Однак треба зауважити, що ці обмеження зменшують загальну кількість рейтингів при обчисленні прогнозів, враховуючи лише релевантні. Хоча будь-яке підвищення точності у світлі такого зменшення може здатися контрінтуїтивним, коли розрідженість даних є проблемою, неможливість відфільтрування рейтинги, які не мають значення для обчислення виявитися ще більш проблематичною.

Найважливіше те, що модель біполярного SLOPE ONE нічого не передбачає з факту того, що користувачеві A подобається елемент K, а користувачеві B не подобається цей самий K. Формально кожну оцінку можна поділити на два набори оцінок:

$$S^{like}(u) = \{i \in S(u) | u_i > u\}, \qquad (3.17)$$
$$S^{dislike}(u) = \{i \in S(u) | u_i < u\}$$

І для кожної пари елементів i, j розділіть набір усіх оцінок χ на $S_{i,j}^{like} = \{u \in \chi \mid i, j \in S^{like}(u)\}$ та $S_{i,j}^{dislike} = \{u \in \chi \mid i, j \in S^{dislike}(u)\}$. Використовуючи ці два набори, ми обчислимо наступну матрицю відхилень для елементів, що сподобалися, а також матрицю деривації $dev_{j,i}^{dislike}$.

$$dev_{j,i}^{like} = \sum_{u \in S_{j,i}^{like}(\chi)} \frac{u_j - u_i}{card\left(S_{j,i}^{like}(\chi)\right)} \qquad (3.18)$$

Прогноз для рейтингу елемента j, заснований на оцінці елемента i, або $p_{j,i}^{like} = dev_{j,i}^{like} + u_i$, або $p_{j,i}^{dislike} = dev_{j,i}^{dislike} + u_i$ залежно від того, належить i до $S^{like}(u)$ або $S^{dislike}(u)$ відповідно.

Модель біполярного SLOPE ONE має наступний вигляд:

$$P^{bpS1}(u)_j = \frac{\sum_{i \in S^{like}(u)-\{j\}} p_{j,i}^{like} c_{j,i}^{like} + \sum_{i \in S^{dislike}(u)-\{j\}} p_{j,i}^{dislike} c_{j,i}^{dislike}}{\sum_{i \in S^{like}(u)-\{j\}} c_{j,i}^{like} + \sum_{i \in S^{dislike}(u)-\{j\}} c_{j,i}^{dislike}} \qquad (3.19)$$

де ваги $c_{j,i}^{like} = card(S_{j,i}^{like})$ та $c_{j,i}^{dislike} = card(S_{j,i}^{dislike})$ є подібними до ваги у зваженій моделі SLOPE ONE.

Базуючись на виконаному аналізі, можливо зробити висновок, що найкращою моделю для використання у рекомендаційних системах є матрична факторизація на основі алгоритму ALS. Розглянутий алгоритм SLOPE ONE є досить точним, але його алгоритмічна складність є надто високою для використання при наявності значних обсягів даних. Окрім цього, ALS має велику кількість реалізацій у бібліотеках машиного навчання, таких як scikit-learn та pyspark.

4 ПОСТАНОВКА ЗАДАЧІ

Розглянемо важливість рекомендаціної системи на прикладі сервісу Netflix, що спеціалізується на наданні доступу до фільмів і серіалів користувачам. Основною частиною функціоналу цього сервісу є надання рекомендацій користувачам на основі попередньо переглянутого або популярного серед інших користувачів, тому що для нього важливо змусити користувача продивитися так багато контенту, як тількі можливо, бо саме це забезпечує йому прибуток. Netflix характеризується величезним обсягом контенту, доступного користувачам. В якості вхідних даних цей сервіс використує:

– існуючи взаємодії обраного користувача;

– вибір інших користувачів із схожим смаком та уподобаннями;

– іншу додаткову інформацію стосовно назв, жанрів, категорій та іншого.

Основна функція цього сервісу — створення рекомендацій, які підказують користувачам те, що має їм сподобитися. Критерії обрання ціх рекомендацій цілком залежать від обраної моделі, що використовується цим сервісом. Головною особливостю рекомендаціного сервісу є його спроможність оброблювати величезні обсяги даних, що є важливим для систем цього класу.

Постановка задачі дослідження полягає у аналізі існуючих методів створення рекомендацій з цілю виявлення найкращих з них для обробки значних обсягів даних та створення рекомендаційної системи, на його основі.

Використання систем розподілених обчислень, таких як Apache Spark [3] стає необхідним при зростанні кількості інформації. Таким чином, постановка задачі дослідження зводиться до наступного:

– проведення аналізу методів колаборативної фільтрації та розробка стратегії іх вибору в залежності від початкових даних та іх прогнозованих обсягів;

– розробка рекомендаційної системи що базується на розподілених обчисленнях та сервісу, що її використає;

– впровадження розробленої системи до цього сервісу.

4.1 Вихідна інформація

Вихідною інформацією модулю є інформація про результат рекомендації, відображена у виді тексу на екраній формі. Ця інформація має містити наступні дані:

– список рекомендованих елементів із заданою кількістю.

4.2 Вхідна інформація

В якості вхідної інформації система використовує:

– список користувачів у системі:

– список елементів системи, що можуть бути рекомендовані;

– список зареєстрованих взаємодій між користувачами та елементами.

4.3 Визначення основних варіантів використання клієнтського додатку

Будь-яка рекомендаційна система призначена для створення рекомендацій її кінцевим користувачам. Розроблювана система має складатися з клієнтського додатку, веб-серверу, що обслуговує запити користувачів та обчислювальної частини, що здатна створювати рекомендації на основі великого обсягу даних.

Для визначення варіантів використання клієнтського додатку потрібно визначити основні вимоги до системи з точки зору функціональності. Розроблювана система має виконувати наступні функції:

– додавання і редагування інформації про користувачів системи, авторизація користувачів;

– додавання та редагування інформації щодо можливих рекомендаційних елементів у системі, наприклад фільмів або продуктів.

– відстежуваввння зроблених користувачем дій стосовно продуктів та зберігання цих дій;

– створення рекомендацій користувачам.

Розроблена система може бути використана будь-яким сервісом надання контенту, наприклад відеострімінгу або електроним магазином для збільшення прибутку на основі більш точних рекомендацій позицій, що мають сподобатися користувачеві.

Відповідна до заданого аналізу функціоналу клієнтського додатку та ролей системи Use-Case діаграма представлена рисунку 4.1.

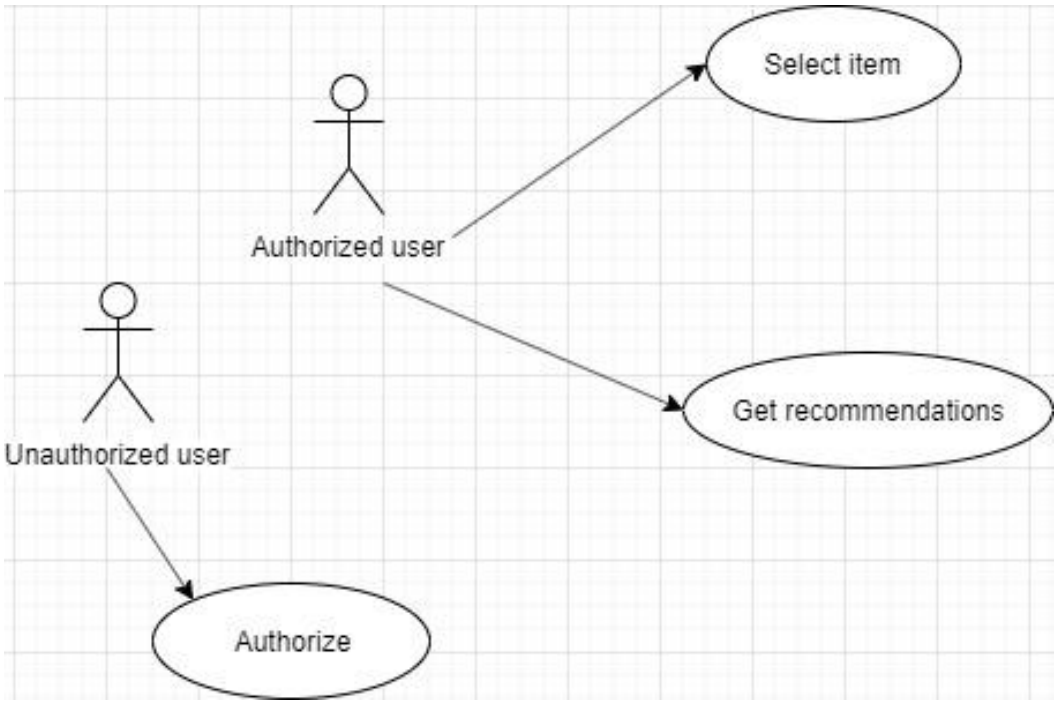

Рисунок 4 .1 — Use-Case діаграма для клієнтського додатку

Як і для будь-якої рекомендаційної системи, основною задачею розроблюваної рекомендаційної системи є створення рекомендації для її користувачів за заданими параметрами. Окрім цього система має зберігати дані про користувачів та їх взаємодії з елементами, що можуть бути рекомендовані

5 РОЗРОБКА ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Сховища даних

Для розробки рекомендаційної системи, що здатна опрацювувати великий обсяг даних необхідно використати сховища, які здатні зберігати ці обсяги та надавати відносно швидкий доступ до них. Для вирішення даної задачі сховище має підтримувати лише дві операції – запис та читання, оскільки записи щодо взаємодій між користувачами та елементами ніколи не будуть змінені та бути здатним масштабуватися горізонтально.

База даних (БД) – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами. Внутрішній устрій різних систем управління базами даних впливає на особливості роботи з ними. Наприклад, нереляційні бази краще піддаються масштабуванню.

Реляційний підхід є найбільш поширеним підходом до організації баз даних. Саме на ньому заснована переважна більшість сучасних комерційних системах управління базами даних. У реляційних системах інформація представляється користувачеві у вигляді таблиць. Таке представлення є досить зручним і досить зрозумілим для будь-якої людини. Можливо, саме завдяки цьому реляційний підхід і здобув таку популярність.

Однак, хотілося б відзначити, що в сучасних СУБД часто порушується класичні принципи реляційного підходу в догоду розширенню функціональності (PostgreSQL, MySQL та Microsoft SQL Server мають підтримку JSON полей та запитів з використанням json-path).

Об'єктно-орієнтована база даних (ООБД) – база даних, в якій дані моделюються у вигляді об'єктів, їх атрибутів, методів і класів (як в об'єктно-орієнтованому програмуванні). Об'єктно-орієнтований підхід підтримує всі

принципи об'єктно-орієнтованого програмування – інкапсуляцію, успадкування, поліморфізм. В БД містяться класи, створені користувачем.

В об'єктно-орієнтованої моделі даних будь-яка сутність реального світу представляється всього одним поняттям - об'єктом. З об'єктом асоціюється стан і поведінка. Стан об'єкта визначається значеннями його властивостей – атрибутів [22].

Документо-орієнтовані бази даних призначені для зберігання і управління документо-орієнтованої або напів-структурованої інформації. На відміну від реляційних баз даних з їх поняттями «відносин», ці системи побудовані навколо абстрактного поняття «документа». Документ містить довільне число властивостей. Документи в ДОБД схожі на записи в РБД, але до них не такі суворі вимоги. Вони не зобов'язані мати однакову структури і кількість атрибутів. Порівняння підходів наведено у таблиці 5.1.

Таблиця 5.1 – Основні переваги і недоліки підходів до організації баз даних

Підхід	Переваги	Недоліки
Реляційний	Заснований на розвиненому математичному апараті, який дозволяє досить лаконічно описати основні операції над даними, можливість використання операцій реляційної алгебри	Відносно низька швидкість доступу і великий обсяг зовнішньої пам'яті, важкість розуміння структури даних через появу великої кількості таблиць в результаті логічного проектування.

Продовження таблиці 5.1

Підхід	Переваги	Недоліки
		Предметну область не завжди можна представити у вигляді сукупності таблиць, практично не дозволяє зберігати великі обсяги даних, важко масштабується лише через написання великовагових міграцій бази даних до нової схеми
Об’єктно-орі єнтований	У порівнянні з реляційними базами даних найкраща продуктивність при індексуванні великих обсягів даних і великим кількості запитів на читання. Легше масштабуються в порівнянні з SQL рішеннями, легко міняти "схему" даних:не потрібно виконувати	Відсутність інтероперабельності між РБД і ООБД, відсутність стандартної алгебри запитів та коштів забезпечення запитів

Продовження таблиці 5.1

Підхід	Переваги	Недоліки
	ніяких операцій оновлення для додавання нових полів, немає проблем із зберіганням неструктурованих даних, єдине місце зберігання всієї інформації про об'єкт: менше операцій виду "join". Дозволяють користувачам визначати абстракції, полегшують проектування деяких зв'язків та усувають потреба в обумовлених користувачами ключах, у деяких ситуаціях забезпечують більш високу продуктивність	
Документо-орієнтований	У порівнянні з реляційними базами даних найкраща продуктивність при індексуванні великих	Відсутність транзакціонної логіки і контролю цілісності в більшості реалізацій: необхідно

Продовження таблиці 5.1

Підхід	Переваги	Недоліки
	обсягів даних і великим кількості запитів на читання. Легше масштабуються в порівнянні з SQL рішеннями, легко міняти "схему" даних: не потрібно виконувати ніяких операцій оновлення для додавання нових полів. Немає проблем із зберіганням неструктурованих даних. Єдине місце зберігання всієї інформації про об'єкт: менше операцій виду "join". Простий інтерфейс спілкування з БД	реалізовувати її в логіці програми. Для обробки даних необхідно використання додаткового мови програмування
Мережева	Досить легке горизонтальне масштабування у ситуації, коли можлива	При горизонтальному масштабуванні у ситуації, коли дані не можуть бути

Продовження таблиці 5.1

Підхід	Переваги	Недоліки
	кластеризація даних. Для деяких типів даних є нативною, наприклад для створення бази даних друзів у соціальній мережі, висока продуктивність запитів, що не пов’язані з перетинанням багатьох ребер графу і не потребують мережевих запитів	кластеризовані має невелику продуктивність внаслідок необхідності мереживих запитів між вузлами. Перетинання графу є дорогою операцією

Таким чином можно зробити висновок, що для збереження інформації про взаємодії між користувачами та елементами системи достатньо використати документо-орієнтоване сховище, тому що воно має забезпечити найкращу швидкість читання, легке масштабування, а використання транзакціональної логіки не має сенсу для даної задачі.

В якості документоорієнтованого сховища була обрана MongoDB. MongoDB – документо-орієнтована система керування базами даних з відкритим сирцевим кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СУБД, функціональними і зручними у формуванні запитів [16].

Код MongoDB написаний на мові C++ і поширюється в рамках ліцензії AGPLv3. Вона підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce [16], підтримує реплікацію і побудову відмовостійких конфігурацій. У MongoDB є вбудовані засоби із забезпечення шардінгу (розподіл набору даних по серверах на основі певного ключа), комбінуючи який реплікацією даних можна побудувати горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера на кластер проводиться без зупинки роботи БД простим додаванням нових машин.

Оскільки MongoDB є документоорієнтованим сховищем, через специфіку предметної області, а саме відсутність поновлення даних, які, по суті, використовуються виключно для читання використання реляційної бази даних не виправдано, то саме вона була обрана для зберігання інформації щодо взаємодій користуваів та елементів. Дана СУБД не реалізує властивості ACID, тому її не варто використовувати для зберігання інформації, яка може оновлюватися паралельно. MongoDB використовує блокування тільки на рівні документа перед кожною операцією запису, що дозволяє їй значно випереджати по продуктивності класичні SQL рішення. Незважаючи на те, що основна частина даних, необхідних для розрахунків буде зберігатися у документоорієнтованому сховищі, існує необхідність у використанні реляційної СУБД. Частина системи, що пов’язана з даними про користувачів та чимось, що потребує можливості модифікації зі зберіганням консистентності має використати реляційну базу даних, що задовольняє ACID.

5.2 Опис архітектури системи

Для розроблюваної системи була обрана архітектура Клієнт-сервер, при якій взаємодія модулів відбувається по протоколу HTTP шляхом передачі повідомлень (запитів) від клієнта до сервера.

Архітектура програмного забезпечення – сукупність найважливіших рішень про організацію програмної системи. Архітектура включає:

– вибір структурних елементів і їх інтерфейсів, за допомогою яких складена система, а також їх поведінки в рамках взаємодії структурних елементів;

– з'єднання обраних елементів структури і поведінки в більш великі системи;

– архітектурний стиль, який направляє всю організацію – все елементи, їх інтерфейси, їх взаємодія і їх з'єднання.

Документування архітектури програмного забезпечення (ПО) спрощує процес комунікації між розробниками, дозволяє зафіксувати прийняті проектні рішення і надати інформацію про них експлуатаційного персоналу системи, повторно використовувати компоненти і шаблони проекту в інших [23].

У підсумку, підтримка добре спроектованих і перевірених архітектурних рішень наділяє програмне забезпечення наступними властивостями:

– масштабованість;

– придатність до тестування;

– розподіл обов'язків елементів системи;

– придатність до підтримки;

– модульність.

Архітектура системи покликана створювати структурованість не по розташуванню в просторі деякого типу, але по взаємодії елементів деякої певної системи. Існує ряд архітектурних шаблонів (патернів), рекомендацій і

повноформатних моделей, кожна з яких вирішує свої задачі, а, отже, має свої переваги і недоліки. Також архітектура класифікується залежно від сфери застосування, тобто вона може стосуватися різнорідних оточень, прикладами яких можна вважати архітектуру клієнт-серверної взаємодії, архітектуру взаємодії системи з графічними елементами системи, архітектуру взаємодії програмного засобу з областю постійного зберігання даних (persistence), і т.д.

Першим винайденим архітектурним шаблоном є Model-View-Controller. Це схема поділу даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: модель, представлення та контролер – таким чином, що модифікація кожного компонента може здійснюватися незалежно.

Модель (Model) надає дані і реагує на команди контролера, змінюючи свій стан. Модель надає дані і методи роботи з ними: запити до бази даних, перевірка на коректність. Модель не залежить від представлення (не знає як дані візуалізувати) і контролера (не має точок взаємодії з користувачем) просто надаючи доступ до даних і управління ними. Модель будується таким чином, щоб відповідати на запити, змінюючи свій стан, при цьому може бути вбудовано повідомлення «спостерігачів». Модель, за рахунок незалежності від візуального представлення, може мати кілька різних уявлень для однієї «моделі».

Представлення (View) відповідає за відображення даних моделі користувачеві, реагуючи на зміни моделі.

Контролер (Controller) інтерпретує дії користувача, сповіщаючи модель про необхідність змін. Контролер забезпечує «зв'язок» між користувачем і системою. Контролює і направляє дані від користувача до системи і навпаки. Використовує модель і представлення для реалізації необхідних дій [25].

З розвитком об'єктно-орієнтованого програмування і поняття про шаблони проектування – був створений ряд модифікацій концепції MVC, які при реалізації у різних авторів можуть відрізнятися від оригінальної. І оскільки

цей архітектурний шаблон в результаті перетворився більше в опис абстракцій, ніж у суворий опис реалізації, то реалізований він може бути по-різному. Немає загальноприйнятого визначення, де повинна розташовуватися бізнес-логіка. Вона може знаходитися як в контролері, так і в моделі. В останньому випадку, модель буде містити всі бізнес-об'єкти з усіма даними і функціями. Також не вказано, де повинна знаходитися перевірка введених користувачем даних. Проста валідація може зустрічатися навіть у представлені, але частіше вони зустрічаються в контролері чи моделі.

Для реалізації схеми «Model-View-Controller» використовується досить велика кількість шаблонів проектування (в залежності від складності архітектурного рішення), основні з яких – «спостерігач», «стратегія», «компоновщик». Найбільш типова реалізація – в якій представлення відокремлено від моделі шляхом встановлення між ними протоколу взаємодії, що використовує «апарат подій» (позначення «подіями» певних ситуацій, що виникають в ході виконання програми, і розсилка повідомлень про них всім тим, хто підписався на отримання) : при кожній особливій зміні внутрішніх даних в моделі (позначеному як «подія»), вона сповіщає про це ті залежні від неї представлення, які підписалися на такі оповіщення – і представлення оновлюється. Так використовується шаблон «спостерігач». При обробці подій, які були надіслані користувачем за допомогою графічного інтерфейсу, представлення вибирає, в залежності від події, потрібний контролер, який забезпечить обробку події зі зміною стану моделі. Для цього використовується шаблон «стратегія», або замість цього може бути модифікація з використанням шаблону «команда». Крім того, можуть використовуватися і інші шаблони проектування – наприклад, «фабричний метод», який дозволить задати за замовчуванням тип контролера для відповідного виду [23].

У контексті розробки даної системи немає необхідності у впровадженні концепції зв'язування даних, тому що зв'язування даних в основному пов'язано з системами, в яких існує величезний обсяг різнорідних даних і набір

графічних компонентів, які покликані відображати зміни цих даних, причому можуть існувати компоненти, покликані відображати зміну одних і тих же даних, але в різних форматах і контекстах. Цей архітектурний шаблон не дасть ніяких переваг при розробці даної системи.

Отже, враховуючи переваги та недоліки описаних шаблонів для розробки серверної частини було прийнято рішення використовувати архітектурний шаблон MVC, а для клієнтського додатку – MVVM.

Найбільш часто архітектура клієнт-сервер застосовується для додатків, створених з використанням систем управління базами даних (СУБД). При цьому на сервері розміщується ядро СУБД і виконуються найбільш складні операції по введенню, зберіганню, модифікації, вилученню та первинній обробці даних. Наприклад, в реляційних БД на сервері виконуються такі трудомісткі операції, як з'єднання таблиць, сортування, виконання складних запитів і т.д. Централізоване зберігання БД дозволяє полегшити роботи по адмініструванню БД, оскільки вони виконуються локально на серверах, забезпечити загальні для всіх додатків правила контролю цілісності і несуперечності БД, а також реалізувати контроль прав доступу до даних.

Архітектура розроблюваної системи складається з двох частин – частини що відповідає за створення акаунтів користувачів в та збереження іх дій та частини, що створює рекомендації на основі тренованої моделі. Перша частина також відповідає за передачу результату рекомендації до клієнту через програмний інтерфейс. Таким чином клієнт не взаємодіє з рекомендаційною частиною системи безпосередньо, що є гарною практикою і дозволяє гнучкіше налаштовувати та замінювати частини системи або використати кешування результатів.

Саме ця остання частина є ключовою, і буде розглянута більш детальніше. Загальна ілюстрація цієї архітектури наведена на рисунку 5.1.

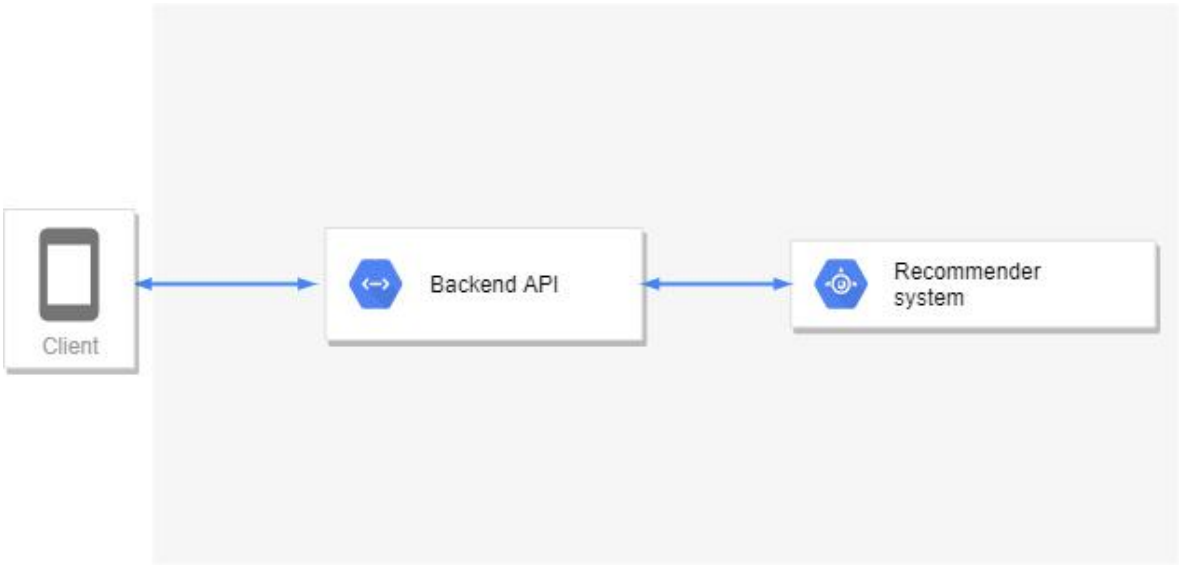

Рисунок 5.1 — Загальна ілюстрація архітектури системи

Оскільки система повинна бути здатною оброблювати великі обсяги даних, необхідно використати інструменти для виконання розроділених обчислень. Особливостю розподілених обчислень є те, що у розподіленій системі, компоненти якої знаходяться на різних комп'ютерах у мережі, ці компоненти здатні виконувати окремі частини задачі, координуючи свої дії для досягнення спільної мети – вирішення цієї задачі. Сукупність цих компонентів називають обчислювальним кластером.

Головною перевагою цього підходу є можливість майже необмеженного горизонтального масштабування - для того, щоб збільшити потужність системи необхідно лише додати більше комп’ютерів до мережі – так званих “вузлів” кластеру, що є набагато дешевшим ніж вертикальне масштабування – збільшення продуктивності кожного компонента системи з метою підвищення загальної продуктивності, що є набагато дорожчим. Огляд популярних інструментів виконання розподілених обчислень наведений у таблиці 5.2.

Таблиця 5.2 – Основні переваги і недоліки інтрументів виконання розподілених обчислень

Назва	Переваги	Недоліки
Apache Hadoop	Дозволяє виконувати розподілену обробку великого обсягу даних у кластері, використаючи прості програмні моделі. Може масштабуватися від одного серверу до тисяч машин, кожна з яких виконує локальні обчислення та зберігає дані, заснований на парадигмі map-reduce. Має модулі розподіленої файлової системи та планувальник задач, відповідальний за автоматичне виділення ресурсів, має велику екосистему та гарну підтримку.	Використовує дискове пространтство для збереження проміжних результатів, що є досить повільним у порівнянні із збереженням в пам’яті.
Apache Storm	Дозволяє виконувати обчислення у режимі реального часу, має підтримку багатьох мов програмування, направлених на потокову обробку даних. Може бути використаний для обчислення аналітичних показників, машиного навчання, розподіленого віддаленого визову процедур, Extract-transform-load	Не має вбудованої підтрики для бібліотек машиного навчання

Подовження таблиці 5.2.

Назва	Переваги	Недоліки
	операцій та іншого.	
Apache Spark	Дозволяє зберігати проміжні результати обчислень у пам’яті, якщо це можливо, має вбудовану підтримку SQL-сумісного інтрефейсу, підтримує велику кількість алгорітмів та методів машиного навчання, може бути інтегрований з модулями Hadoop. Має підтримку майже будь-яких джерел отримання даних, в тому числі реляційні та документо-орієнтовані сховища, розподілені файлові системи, колоночні сховища та ін. Має підтримку декількохі мов програмування, у тому числі Python та R.	Має відносно складний інтерфейс взаємодії з розподіленими абстракціями над даними (RDD, DataFrame).

Таким чином, на основі порівняння інструментів розподіленої обробки для створення рекомендаційної системи має бути використаний саме Apache Spark. Apache Spark – високорозвинутий двигун для обробки даних великого масштабу, здатний працювати на тисячах машин паралельно. Це дозволяє максимізувати процесорну обчислювальную здатність цих машин для збільшення продуктивності. Spark має здатність оброблювати декілька задач по обробці даних, включаючи складний аналітіз даних, потоковий аналіз,

графовий аналіз так само, як масштабоване машине навчання на величезних обсягах даних порядку терабайтів, зеттабайтів та навіть більших.

Apache Spark базується на парадигмі map-reduce, але, на відмінну від свого попередника Hadoop, що базується на тій ж самій парадишмі, який виконує обчислення, використовуючи дисковий простір, Spark здатний виконувати обчислювання цілком у оперативній пам’яті або використати комбінований спосіб збереження, що дозволяє уникнути безперервних операцій вводу/виводу [20].

Для виконання ефективних обчислень у пам’яті Spark використовує спеціалізовану модель даних, що має назву Resilient Distributed Dataset (RDD). Ця модель може бути ефективно збережена в пам'яті та має підтримку багатьох видів операцій, основні з яких:

– map — трансформація елементів;

– reduce — групова операція для знаходження загальних метрик;

– filter — фільтрування елементів;

– distinct — створює датасет з лише унікальними елементами початкового датасету;

– union — об’єднання датасетів;

– intersection — створення датасету, що складається з елементів, які зустрічаються в обох датасетах;

– collect — збір проміжних даних на master вузлі;

– count — кількість елементів у RDD;

– take — отримання перших n елементів;

– cache —кешування проміжних результатів у оперативній пам’яті;

– groupBy — групування за вказаними властивостями.

Усі RDD є незмінними, тобто будь-яка операція над RDD призведе до створення гового RDD. Цей формат також ефективно розподіляється по кластерам машин, кожна з яких зберігає лише певну частину RDD - так звану партицію. В загальному сенсі RDD можна розглядати як абстракцію даних над

необробленим форматом даних. Загальний приклад використання RDD наведений на рисунку 5.2.

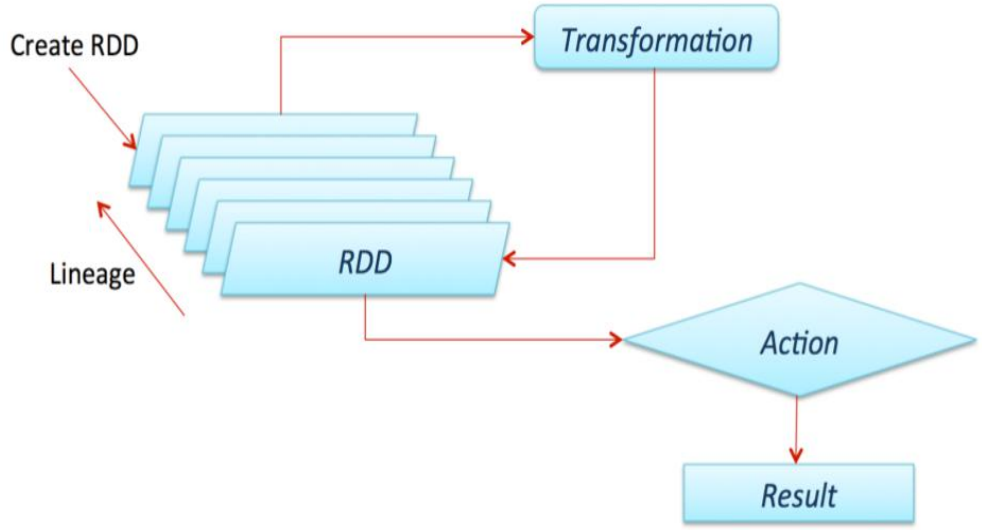

Рисунок 5.2 — Приклад роботи RDD

Крім RDD, Spark також використовує Direct Acyclic Graph (DAG) для відстеження обчислень на RDD, цей підхід оптимізує обробку даних, використовуючи потоки завдань для належного призначення оптимізації продуктивності, це також має додаткову перевагу, яка допомагає Spark керувати помилками, коли є збої в роботі або роботі через ефективний механізм відкату. Отже, у випадках помилок, Spark не потрібно починати обчислення з самого початку, він може легко використовувати обчислені RDD перед помилкою та передавати її через фіксовану операцію [22]. Приклад розподілення RDD на партиції наведений на рисунку 5.3.

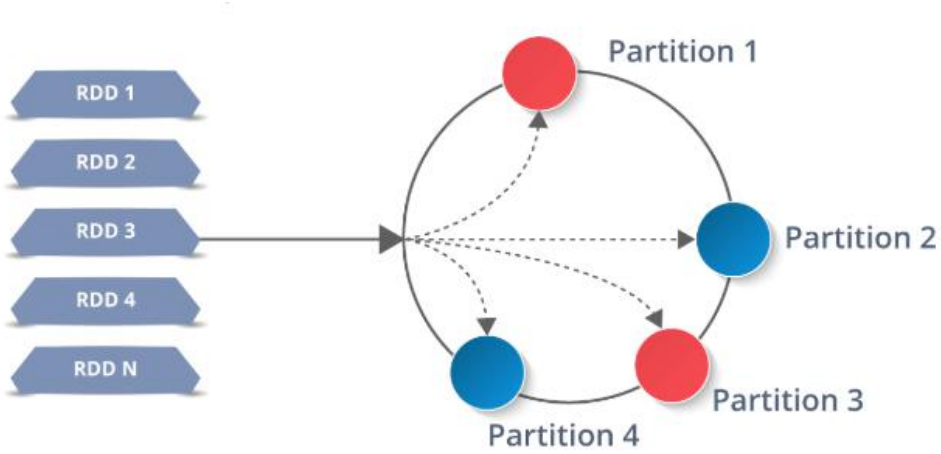

Рисунок 5.3 — Ілюстрація розподілення RDD на партиції

Spark також використовує диспетчера кластерів, щоб правильно виконувати свою роботу по кластеру машин, менеджер кластерів допомагає в розподілі ресурсів і плануванні роботи в режимі master - worker. Діаграма взаємодії у кластері наведена на рисунку 5.4. Майстер розподіляє робочі місця та розподіляє необхідні ресурси для працівників кластеру та координує діяльність працівника таким чином, що у випадках, коли працівник недоступний, це завдання перепризначається іншому працівникові.

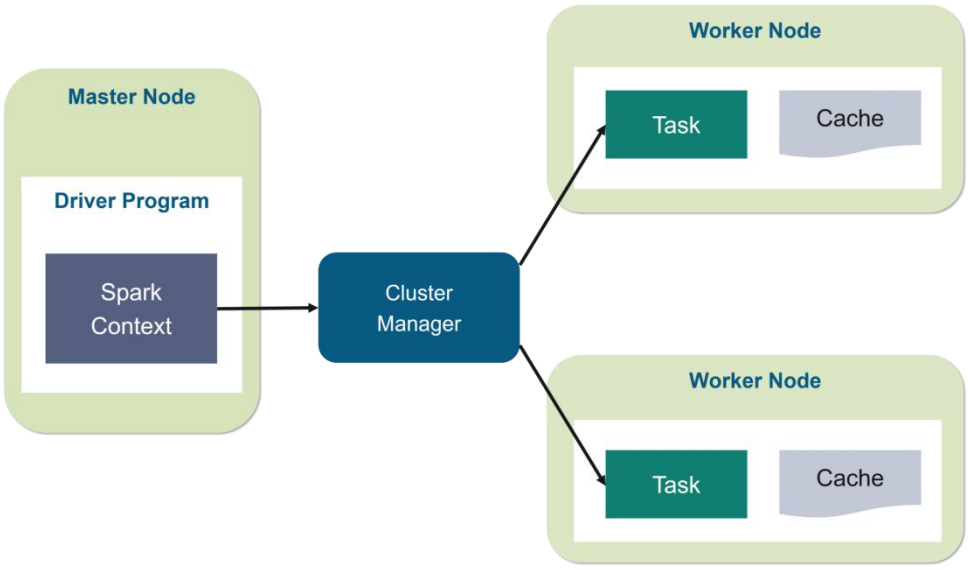

Рисунок 5.4 — Загальна архітектура Apache Spark

Ідея обробки оперативної пам'яті за допомогою абстрагування RDD, парадигми обчислень DAG, розподілу ресурсів та планування роботи менеджером кластерів Spark перетворилася на постійно прогресуючий двигун у світі швидкої великої обробки даних.

Apache Spark підтримує декілька мов програмування, серед них – Scala, Java, Python та R. Окрім цього, він має велику кількість бібліотек з реалізованими функціями - mllib, spark streaming та інші. Mllib представлє собою бібліотеку для машиного навчання, яка взаємодіє з популярною python бібліотекою NumPy. Mllib включає наступні алгоритми [18]:

- методи класифікації: логістична регресія, наївний баєсів класифікатор;
- методи регресії;
- дерево ухвалення рішень, Random forest, gradient-boosted trees;
- рекомендаційні: ALS;
- кластерізаційні: K-means, GMMs.

Для розробки рекомендаційної системи була обрана мова програмування Python, що є стандартом у сфері машиного навчання та штучного інтелекту та має велику кількість бібліотек та інструментів для цього.

6 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Вибір платформи та мови програмування

В якості програмного середовища для серверної частини була обрана JVM – ця платформа має багату екосистему і велику кількість готових рішень для широкого спектру задач, фреймворків та бібліотек. Це дозволяє значно скоротити час на розробку програмної системи шляхом вибору фреймворку для розробки [6].

Java – об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

«Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Для розробки серверної частини системи, що відвовідає за операції пов’язані з користувачами (регістрація, збір даних та ін.) була використана мова програмування Java і веб-фреймворк Spring, що надає велику кількість функціональних модулів і дозволяє знизити час на початкову конфігурацію програми та веб-сервера.

Як сервер додатка був використаний Apache Tomcat - один з найпопулярніших сервлет-контейнерів, використовуваних для розробки на Java, що має багато корисних можливостей, таких як:

- встроєний механізм підтримки TLS протоколу для забезпечення захищеної передачі даних у мережі;

- неблокуюче введення/виведення для покращення продуктивності;

- підтримка веб-сокетів;

- гнучка конфігурація пула потоків для обробки HTTP-запитів, максимального розміру завантажених файлів та механізмів авторизації;

- гнучка конфігурація систем створення логів.

Для реалізації клієнтського додатку була обрана мова програмування Javascript. JavaScript (JS) – динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.

Для створення рекоммендаційної частини на основі Apache Spark був обраний модуль PySpark, оснований на мові програмування Python. Для реалізації була обрана предметна область «стрімінговий аудіо-сервіс». Ця рекомендаціна система має рекомендувати користувачам системи виконавців, що мають їм сподобатися в залежності від вже прослуханих ними композицій. В якості початкових даних був використан датасет стримінгового музикального сервісу audioscrobbler, що містить 141 тисячу унікальних користувачів та 24.2 мільйона записів прослуханих композицій виконавців з їх кількістю. В якості моделі колаборативної фільтрацій була обрана модель ALS з пакету Apache Spark mllib. Структура даних представлена у таблиці 6.1.

Таблиця 6.1 структура датасету audioscrobbler

Назва	Значення
user_id	Унікальний ідентифікатор користувача
artist_id	Унікальний ідентифікатор виконавця
quantity	Кількість прослухань композицій виконавця відповідним користувачем

Обрана модель для тренування є моделю з неявним зворотнім зв'язком. Функція ALS з пакету mllib має декілька параметрів, що впливають на створення моделі та можуть бути настроєні. Для того, щоб створити найкращу модель необхідно підібрати значення цих параметрів, при яких вона буде показувати найкращі результати на валідаційному наборі даних. Таким чином, по-перше необхідно створити метод перевірки якості моделі. Після цього кожна модель з комбінації параметрів має бути оцінена з метою вибору найракщої комбінації.

Для оцінки якості моделі був обраний наступний метод: представимо, що існує модель та відповідний набір даних прослуханих виконавців для деякого набору користувачів. Ця модель може бути використана для передбачення наступних перших *X* рекомендацій для користувачів та ці рекомендації можуть бути порівняні з виконавцями, яких користувач дійсно слухав. Таким чином, частка, яка співпадає у перших *X* передбаченнях моделі та *X* виконавців, яких користувач дійсно переслухав може бути обчислена. Цей процес має бути півторений для всіх користувачів з метою обчислення середнього значення для запобігання підбору параметрів, що прив'язані до цього тестового датасету - так звана проблема «overfitted» моделей [24]. Поділення даних на валідаційний та тестувальний набори представлене на рисунку 6.1.

```
trainData, validationData, testData = userArtistData.randomSplit([4, 4, 2], 13)

print trainData.take(3)
print validationData.take(3)
print testData.take(3)
print trainData.count()
print validationData.count()
print testData.count()

#Caching and creating ratings object
trainData = trainData.map(lambda l: Rating(*l)).cache()
validationData = validationData.map(lambda l: Rating(*l)).cache()
testData = testData.map(lambda l: Rating(*l)).cache()
```

Рисунок 6.1 — Створення наборів даних для тестування та перевірки

Приклад створеного набору даних наведений на рисунку 6.2:

```
[(1059637, 1000049, 1), (1059637, 1000056, 1), (1059637, 1000113, 5)]
[(1059637, 1000010, 238), (1059637, 1000062, 11), (1059637, 1000112, 423)]
[(1059637, 1000094, 1), (1059637, 1000130, 19129), (1059637, 1000139, 4)]
19817
19633
10031
```

Рисунок 6.2 — Приклад структури створеного набору даних та загальнальна кількість елементів у наборі

Наприклад, припустимо що модель створила передбачення зі значеннями *[1, 2, 4, 8]* для *X=4* виконавців для цього користувача. Припустимо, що користувач дійсно прослухав виконавців *[1, 3, 7, 8]*. Таким чином, ця модель має результат, що дорівнює *2/4=0.5*. Для того, щоб знайти загальний результат, ця операція має бути виконана для всіх користувачів із обчисленням середнього значення. Потрібно зазначити, що при використанні моделі для передбачення виконавців для користувача тренувальні дані не мають містити даних щодо прослухань виконавців для цього користувача. ALS має декілька параметрів, один з найважливіших з яких “rank”, що відповідає кількості так званих “латентних факторів” [25]. Метод оцінки наведений на рисунку 6.3.

```
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
from collections import defaultdict

#model evaluation function
def modelEval(model, dataset):
    global trainData
    global allArtists

    #Getting nonTrainArtists for each user
    userArtists = defaultdict(list)

    for data in trainData.collect():
        userArtists[data[0]].append(data[1])

    cvList = []

    for key in userArtists.keys():
        userArtists[key] = list(set(allArtists) - set(userArtists[key]))
        for artist in userArtists[key]:
            cvList.append((key, artist))

    #Creating user,nonTrainArtists RDD
    cvData = sc.parallelize(cvList)

    userOriginal = dataset.map(lambda x:(x.user, (x.product, x.rating))).groupByKey().collect()

    #prediction on the user, nonTrainArtists RDD
    predictions = model.predictAll(cvData)
    userPredictions = predictions.map(lambda x:(x.user, (x.product, x.rating))).groupByKey().co
    original = {}
    predictions = {}
```

Рисунок 6.3 — Метод оцінки якості моделі

Для будування моделі були використані значення *rank = [2, 10, 20]*. Оцінки для моделей з цими значеннями представлені на рисунку 6.4.

```
rank_list = [2, 10, 20]

for rank in rank_list:
    model = ALS.trainImplicit(trainData, rank, seed=345)
    modelEval(model,validationData)
```

```
The model score for rank 2 is 0.0909391661474
The model score for rank 10 is 0.0957125879247
The model score for rank 20 is 0.09047041725
```

Рисунок 6.4 — Результати оцінок моделей для значень “rank”

Таким чином, на основі найкращої оцінки можна обрати параметри моделі, що створює найбільш точні рекомендації — у цьому випадку це значення *rank = 10*. Наступним кроком знайдемо перші 5 рекомендованих виконавців для довільного користувача. Пошук цих елементів для користувача з ідентифікатором 1059637 представлений на рисунку 6.5.

```
ratings = bestModel.recommendProducts(1059637, 5)

import re
artist_data = artistData.collect()

artist_names_dict = {}

for line in artist_data:
    pattern = re.match( r'(\d+)(\s+)(.*)', line)
    artist_names_dict[str(pattern.group(1))] = pattern.group(3)

for i in range(0,5):
    if str(ratings[i].product) in artist_canonical_dict:
        artist_id = artist_canonical_dict[str(ratings[i].product)]
        print "Artist " + str(i) + ": "  + str(artist_names_dict[str(artist_id)])
    else:
        print "Artist " + str(i) + ": " + str(artist_names_dict[str(ratings[i].product)])
```

Рисунок 6.5 — Приклад створення рекомендації для довільно обраного користувача

На основі даного прикладу отримаємо перші п’ять виконавців що мають бути рекомендовані обраному користувачу. Приклад згенерованої рекомендації наведений на рисунку 6.6.

```
Artist 0: Brand New
Artist 1: Taking Back Sunday
Artist 2: Evanescence
Artist 3: Elliott Smith
Artist 4: blink-182
```

Рисунок 6.6 — Рекомендація для обраного користувача

Таким чином, для кожного з користувачів системи за допомогою рекомендаціної системи створюється набір виконавців, що мають сподобатися їм з вказаним лімітом (перші п’ять у даному випадку).

ВИСНОВКИ

У ході виконання атестаційної роботи була описана проблема необхідності створення рекомендацій для різноманітних систем, що взаємодіють з користувачами шляхом продажу продуктів або надання їм якогось контенту. Були розглянуті дві основних категорії методів колаборативної фільтрації для створення рекомендацій та оцінені їх переваги та недоліки. Ті, що базуються на обчисленнях у пам’яті значно простіші для реалізації та використання на відносно невеликих обсягах даних і можуть бути використані для розподілених систем, а ті, що базуються на моделі — значно кращі у ситуаціях, коли обсяги даних дуже великі або кількість обчислювальних ресурсів недостатня для оперування повним набором даних - вони так чи інакше оперують іншими формами даних, значно меншими за обсягом ніж початкові, але набагато складнішими для людського розуміння. Основним їх недоліком є необхідність попередніх тренувань моделей. Таким чином, обрання методу колаборативної фільтрації має залежати від обсягу інформації та наявності обчислювальних ресурсів.

Використання рекомендаційних систем актуальне для будь-яких сфер бізнесу, що базуються на надані доступу до медіа-контенту або продажу товарів. Основною складністю впровадження рекомендаціних систем у цих сферах є обсяг даних, на основі яких мають бути створені рекомендації. Саме ця проблема була детально розглянута в роботі.

В ході виконання роботи були проаналізовані існуючи методи створення рекомендацій на основі колаборативної фільтрації з метою їх використання або впровадження до систем, що оперують великими обсягами даних. Створена рекомендаційна система здатна горізонтально масштабуватися для опрацювування значного обсягу даних завдяки використанню розподілених обчислень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Charu C. Aggarwal. Recommender Systems: The Textbook, 2016.

2. S. H. S.g Chee, J. H., and K. Wang. Rectree: An efficient collaborative filtering method. Lecture Notes in Computer Science, 2114, 2017.

3. Big Data. Часть 3: Spark. [Електронний ресурс]. 05.10.2015. — Режим доступу: https://habr.com/ru/company/dca/blog/268277/

4. Kim Folk. Practical Recommender Systems, 2019.

5. Deepak K. Agarwal and Bee-Chung Chen. Statistical Methods for Recommender Systems. 2016.

6. Shlomo Berkovsky and Ivan Cantador. Collaborative Recommendations: Algorithms, Practical Challenges and Applications. 2019.

7. Bill Chambers and Matei Zaharia, Spark: The Definitive Guide: Big Data Processing Made Simple. 2018.

8. Oliver Theobald. Machine Learning: Make Your Own Recommender System (Machine Learning From Scratch Book 3).

9. Frank Kane. Building Recommender Systems with Machine Learning and AI: Help people discover new products and content with deep learning, neural networks, and machine learning recommendations. Aug 11, 2018.

10. Recommender Systems in Practice [Електронний ресурс]. 05.07.2019. — Режим доступу: https://towardsdatascience.com/recommender-systems-in-practice-cef9033bb23a

11. Dietmar Jannach, Markus Zanker. Recommender Systems: An Introduction. 2010.

12. Introduction to recommender systems [Електронний ресурс]. 05.07.2019. — Режим доступу: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada

13. Frank Kane. Frank Kane's Taming Big Data with Apache Spark and Python. Jun 30, 2017.

14. Jeffrey Aven. Data Analytics with Spark Using Python (Addison-Wesley Data & Analytics Series). Jun 16, 2018.

15. Collaborative Recommendations: Algorithms, Practical Challenges and Applications by Shlomo Berkovsky, Ivan Cantador, et al. May 15, 2019

16. Jean-Georges Perrin. Spark in Action. March 17, 2019.

17. Romeo Kienzler. Mastering Apache Spark 2.x - Second Edition: Scale your machine learning and deep learning systems with SparkML, DeepLearning4j and H2O. July 26, 2017.

18. Inversion of Control Containers and Dependency Injection pattern. URL: https://www.martinfowler.com/articles/injection.html (дата звернення 05.11.2019)

19. Брюс Экель. Философия Java, 4-е издание. [Текст] // Брюс Экель — М.: «President, MindView, Inc», 2006. — 9с. Bruce Eckel, Thinking in Java, 4th edition – Introduction – 9с.

20. JS. URL: https://en.wikipedia.org/wiki/JavaScript (дата звернення 18.05.2018).

21. Raju Kumar Mishra. PySpark Recipes: A Problem-Solution Approach with PySpark2. Dec 10, 2017

22. Wikipedia [Електронний ресурс] // Separation of Concerns – Електронні данні. – Режим доступу: https://en.wikipedia.org/wiki/Separation_of_concerns (Дата доступу – 18.05.2018).

23. Rounak Banik. Hands-On Recommendation Systems with Python: Start building powerful and personalized, recommendation engines with Python

24. Model-View-ViewModel. URL: https://en.wikipedia.org/wiki/Model-view-viewmodel (дата звернення 05.11.2019).

25. Wikipedia [Електронний ресурс] // Model-View-Controller – Електронні данні. – Режим доступу: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller (Дата доступу 18.05.2018).