МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

15.03 — КМР. 1431 - "С" 2022.11.10.013 ПЗ

Івченка Олександра Володимировича

2022 р.

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.9:323

«ПОГОДЖЕНО»
Декан факультету
інформаційних технологій
Глазунова О.Г., д.п.н., професор
2022 р.

«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»
Завідувач кафедри комп'ютерних наук
Голуб Б.Л., к.т.н., доцент
2022 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему «Дослідження алгоритмів колаборативної фільтрації та розробка рекомендаційної системи на їх основі»

Спеціальність 122 «Комп'ютерні науки»

Освітня програма «Інформаційні управляючі системи та технології»

(назва)

Орієнтація освітньої програми освітньо-професійна

Гарант освітньої програми
к.е.н., доцент Густера О.М.

Керівник магістерської кваліфікаційної роботи
к.т.н., доцент Ільїн О.О.

Виконав Івченко О.В.

КИЇВ-2022

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент
Голуб Б.Л. _____
(ПІБ) (науковий ступінь, вчене звання) (підпис)
"01" листопада 2021 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
СТУДЕНТУ

Івченку Олександру Володимировичу
(прізвище, ім'я, по батькові)

Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи та технології»
Орієнтація освітньої програми: освітньо-професійна
Тема магістерської кваліфікаційної роботи «Дослідження алгоритмів колаборативної фільтрації та розробка рекомендаційної системи на їх основі»
затверджена наказом ректора НУБіП України від "01" листопада 2021 р. №1862 «С»

Термін подання завершеної роботи на кафедру 27 жовтня 2022 р.
Перелік питань, що підлягають дослідженню:

1. Аналіз предметної області
2. Дослідження алгоритмів колаборативної фільтрації
3. Дослідження сучасних засобів розробки
4. Проектування системи

Дата видачі завдання "01" листопада 2021 р.

Керівник магістерської кваліфікаційної роботи _____ Ільїн О.С.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Івченко О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

ПЗ: 52 сторінка, 39 рисунків, 4 таблиці, 12 джерел.

РЕКОМЕНДАЦІЙНА СИСТЕМА,КОЛАБОРАТИВНИЙ АНАЛІЗ.

Об'єкт розробки – Розроблення системи рекомендацій на основі колаборативної фільтрації.

Мета роботи – забезпечити користувачів системи рекомендаціями на основі їх інтересів.

Проект складається з п'яти розділів.

Перший розділ присвячено огляду існуючих проблем, аналізу ТЗ і огляду рекомендаційних систем.

У другому розділі дослідження сучасних рекомендаційних систем на основі колаборативного аналізу

Третій розділ – вибір середовища розробки

Четвертий розділ – опис розробки прототипу системи

П’ятий розділ присвячено опису реалізації системи, огляду інтерфейсу і тестування усього функціоналу розробленої програми.

Результатом виконання дипломної роботи є розроблена система рекомендацій на основі колаборативної фільтрації.

Зміст

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

Атрибут – властивість поведінки користувача;
Інтерфейс – засіб взаємодії системи з користувачем;
Кластер – об'єднані об'єкти, що мають спільні характеристики;
Кластеризація – групування кластерів;
Колаборативний – спільний;
Контент – вміст, інформаційне наповнення;
РС – рекомендаційна система;
БД – база даних;
ПК – персональний комп'ютер;
Клікбейт — зазвичай, заголовок, який змушує користувача інтернету натиснути на нього
API - application programming interface - опис способів, які одна програма може взаємодіяти з іншою програмою, або використання бібліотеки методів.

ВСТУП

В сучасному світі, коли люди неймовірно взаємопов'язані з технікою та всіма благами, що вона надає, гостро виникає попит на автоматизацію всіх технічних моментів, що можуть виникнути.

Рекомендаційні системи в сучасному світі непомірними кроками пішли в майбутнє. Важко собі уявити будь-який сучасний сервіс, де б не використовувалася РС. Будь то онлайн магазин, онлайн бібліотека, відеохостинг, форуми, персональні сторінки, соцмережі.

Процес розробки РС – є доволі індивідуальним процесом, адже немає "єдиного рецепту", щоб розробити ідеальну систему. До кожної системи має бути свій підхід та методи розробки.

Таким чином, існує велика кількість варіантів побудови РС, що провокує до необхідності створення різноманітних варіантів, враховуючи індивідуальні потреби кожної з них. В той же час, наявність багатьох методів та великого об'єму інформації ускладнює пошуки механізмів створення рекомендаційної системи для конкретного прикладу з урахуванням усіх притаманних їй особливостей.

Для уніфікації процесу необхідно правильно оцінити всі особливості, а також раціонально підібрати способи для створення рекомендаційної системи.

1. ДОСЛІДЖЕННЯ ОБ'ЄКТНОЇ БАЗИ ІСНУЮЧИХ ПРОБЛЕМ

1.1 Поняття, що таке рекомендаційна система

За останні десятиліття кількість інформації в інформаційних системах світу значно зросла. Широке впровадження Інтернет-технологій у всіх сферах суспільного життя, доступність інформації - сприяло виникненню необхідності розробити нові методи пошуку інформації. Навіть пошукові системи Google, Yahoo, Bing розглядають персоналізацію інформації.

Рекомендаційна система - є такою системою, використовуючи певний тип інформації, систему фільтрів, яка рекомендує інформаційні елементи, в яких користувач може бути зацікавлений. Типова система рекомендацій приймає рекомендації користувача як вхідні дані, агреговані та надіслані у формі відповідним користувачам. Технологія дозволяє користувачам витрачати найменшу кількість часу на пошук необхідної інформації в інтернеті. Системи рекомендацій порівнюють зібрані дані користувачів і створюють список елементів, рекомендованих користувачам. Це ще один варіант алгоритмів пошуку, оскільки вони допомагають користувачам швидко знаходити предмети та інформацію, якщо вони не можуть знайти її [1].

Використання систем рекомендацій останнім часом поширилося на стаціонарну роздрібну торгівлю, довідкові центри, пошук програмного забезпечення, наукові статті. Ця програма або сайт містять автоматичні рекомендації користувачам на основі виконаних дій (покупки, опубліковані оцінки, відвідування, тощо) та отримання від них відгуків (замовлення в магазині, натискання на посилання).

Системи рекомендацій існують із середини 1990-х років. Основне завдання системи рекомендацій – надати користувачеві персоналізовані

рекомендації з урахуванням його переваг при виборі товару (товару, об'єкта, послуги).

У більшості випадків розробка рекомендаційних систем включає покращення алгоритму рекомендацій. Мета цього вдосконалення — надати відвідувачам сайтів чи користувачам програм найточніші рекомендації щодо виконання їхніх запитів у певний момент. Для цього необхідно постійно вдосконалювати математичні алгоритми, що стоять за рекомендаційними службами. Саме тут вступає в дію інтелектуальний аналіз даних Data Mining.

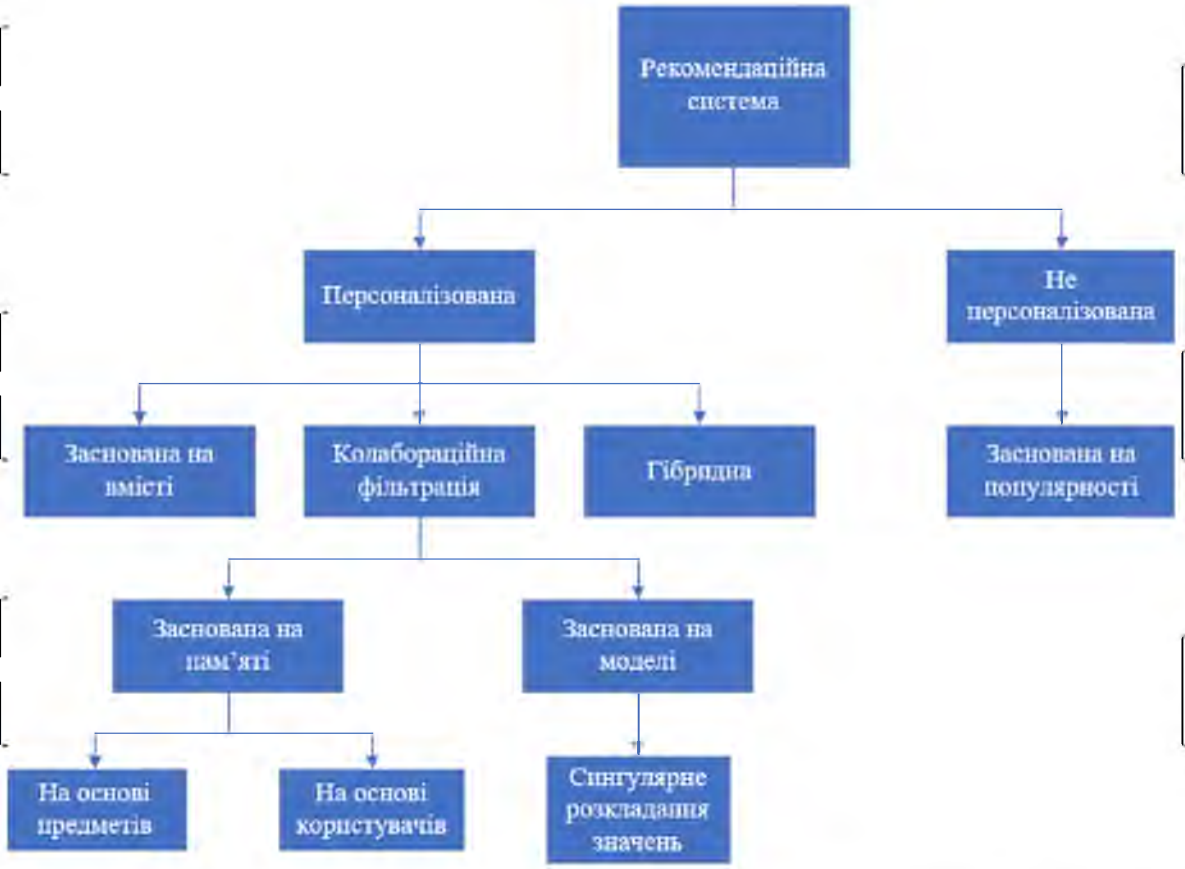

Рисунок 1.1 Класифікація рекомендаційних систем

Це можна приблизно описати так: служба веб-сайту надає набір рекомендацій користувачеві, потім розробник отримує відгук, аналізує дані, перенавчає математичну модель, а потім знову надає

рекомендацію. Математичний алгоритм системи рекомендацій складається з наступних аспектів:

- Кількість рекомендованого контенту – його має бути достатньо, щоб було що запропонувати;
- Обсяг інформації про рекомендований контент (наприклад, хто автор книги, хто перекладач, хто ілюстратор і скільки в ньому символів) – чим більше інформації, тим точніше і правильніше можна вибрати;
- Обсяг інформації про користувача (стать, ім'я, вік, країна проживання) – знову ж таки, чим більше інформації, тим більш конкретні рекомендації вибираються;
- Зручність для користувача - чим комфортніше відвідувачу, тим більше інформації ми можемо отримати про його реакцію на наші пропозиції.[2]

Так як система рекомендацій має допомогти користувачу зекономити свій час та покращити користувацький досвід, а для власника збільшити кількість часу яку проводить користувач на ресурсі або збільшити середній чек, не слід перенасичувати інтерфейс з рекомендаціями. За дослідженнями людина сприймає від 6 до 9 рекомендаційних предметів чи товарів, а коли більше то починає губитися в перенасиченому інтерфейсі. Якщо важко дослідити, що конкретно буде цікаво чи необхідно в даний момент - то як альтернативний спосіб запроваджується система фільтрації (По ціні, кольору, характеристикам).

Методи збору інформації, які надає Інтернет, значно спрощує використання громадської думки за допомогою колаборативної фільтрації. Але, з іншого боку, велика кількість інформації ускладнює реалізацію цієї можливості. Наприклад, поведінка одних людей явно піддається моделюванню, а поведінка інших абсолютно непередбачувана. Саме останнє впливає на зсув результатів системи

рекомендацій і знижує її ефективність. Інший приклад: користувач може використовувати систему рекомендацій, щоб навмисно просувати свій продукт. Вони можуть не тільки залишати позитивні відгуки про продукти, які їм подобаються, але й негативні відгуки про своїх конкурентів. Ще одна проблема, властива великим системам рекомендацій – розширюваність. Традиційні алгоритми добре працюють на невеликих обсягах даних. Але чим більше інформації потрібно обробити, тим важче отримати точні результати.

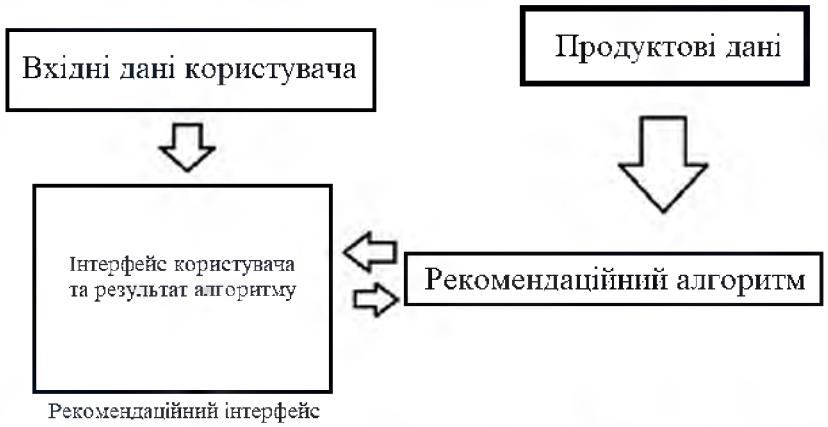

Рисунок 1.2 Архітектура рекомендаційної системи

Сфери застосування систем рекомендацій в Інтернеті:

- Рекомендації товарів в інтернет-магазинах. Очевидно, цей варіант найактуальніший на сьогодні. Система рекомендацій Інтернет-магазинів пропонують користувачам те, що їм потрібно купити на думку алгоритмів, зосереджуючись на продуктах на основі різних факторів.
- Рекомендації щодо фільмів/музики/відео. Сервіси, такі як Netflix, YouTube або Spotify, рекомендують певний контент зареєстрованим відвідувачам, в залежності від їх попереднього перегляду чи прослуховування та/рейтинг того чи іншого відео, фільму або виконавця;

– Рекомендація новин. Система рекомендує «схожі на» раніше прочитані новини, або на основі подібності ключових слів, або також на основі вибору користувачів зі схожими інтересами.

Існують дві основні стратегії створення рекомендаційних систем:

– Фільтрація вмісту
– Колаборативна фільтрація.

При фільтрації вмісту створюються профілі користувачів і об'єктів:

– Профілі користувачів можуть містити демографічну інформацію або відповіді на певний набір питань.
– Профілі об'єктів можуть містити назви жанрів, імена акторів, імена виконавців, тощо. Або якусь іншу інформацію в залежності від типу об'єкта.

Колаборативна фільтрація використовує інформацію про минулу поведінку користувачів, наприклад інформацію про покупки чи оцінки. У цьому випадку не має значення, який тип об’єкта ви використовуєте, але ви можете взяти до уваги неявні особливості, які важко враховувати під час створення файлу конфігурації. Основною проблемою цього типу системи рекомендацій є «холодний старт»: відсутність даних про нових користувачів або об’єктів в системі.[3]

У роботі системи рекомендацій використовують комбінацію явних і неявних методів для збору даних про користувачів.

Явний приклад збору даних:

– Користувачі оцінюють запропоновані об’єкти за різними шкалами;
– Користувач ранжує набір об’єктів від найкращого до гіршого;

– Користувач вибирає найкращий з двох запропонованих об'єктів;
– Можливість користувачеві створити список улюблених об'єктів.

Не явний приклад збору даних:

– Спостереження за тим, що користувач оглядає в інтернет-магазинах або базах даних іншого типу;
– Ведення записів про поведінку користувача онлайн;
– Відстеження вмісту комп'ютера користувача;

1.2 Види та типи алгоритмів колаборативної фільтрації

Спільна фільтрація – це триступеневий процес, що починається зі збору користувацької інформації, потім будується матриця для розрахунку асоціацій і, нарешті, дається дуже вірогідна рекомендація. Її основне припущення полягає в наступному: ті, хто однаково оцінював будь-які предмети в минулому, схильні давати схожі оцінки інших предметів і в майбутньому. Наприклад, за допомогою колаборативної фільтрації музичний додаток здатен прогнозувати, яка музика сподобається користувачу.

Хоча використана інформація збирається від багатьох учасників, прогнози зроблені індивідуально для кожного користувача. Це відрізняє спільну фільтрацію від простішого підходу, який надає середній бал для кожного об'єкта інтересу, наприклад оцінку на основі голосування. Дослідження в цій області активно тривають і в наш час, особливо через невирішеність проблем методів спільної фільтрації.

Технології персоналізованих рекомендацій, такі як спільне фільтрування, дуже корисні в епоху інформаційного вибуху, оскільки кількість елементів навіть в одній категорії (наприклад фільми, музика,

книги, новини, веб-сайти) стала настільки великою, що люди не можуть переглядати їх всі, щоб вибрати необхідний.[4]

Спільні системи фільтрації зазвичай використовують двоступеневу схему:

– Знаходять людей, які поділяють оцінки «активних» (прогнозованих) користувачів.
– Оцінки однодумців, знайдених на першому кроці, використовуються для розрахунку прогнозів.

Існує ще одна форма спільної фільтрації, заснована на прихованому спостереженні за звичайною поведінкою користувача (на відміну від відкритого спостереження зі збору оцінок). У цих системах ви можете спостерігати, як поводиться цей користувач і як поводяться інші (яку музику вони слухають, які відео вони дивляться, що купують) і використовувати отримані дані, щоб спрогнозувати майбутню поведінку користувача або спрогнозувати, як користувач діятиме за такої можливості. Ці прогнози слід робити відповідно до бізнес-логіки, оскільки марно радити споживачеві купувати музичні файли, якими він уже володіє.

Типи спільної фільтрації, котрі базуються на пам’яті – безпосередньо обробляють значення взаємодій, не припускаючи жодної моделі, і, по суті, базуються на пошуку найближчих сусідів (пошук схожого на користувача, і пропозиції йому найпопулярнішого елемента серед “сусідів”). Методи, засновані на моделі, припускають існування узагальненої моделі, яка пояснює взаємодію між елементами та користувачами, і намагається виявити її, щоб виробити нові рекомендації.

Головною перевагою спільної фільтрації є необов’язковість детальної інформації про користувача або об'єкт, тому її можна використовувати в більшій кількості ситуацій. Крім того, чим більше

користувачів взаємодіяли з елементом, тим точнішою буде нова рекомендація (для постійних користувачів і елементів, нові взаємодії, зареєстровані з часом, приносять нову інформацію і роблять систему все більш ефективною). Оскільки спільна фільтрація враховує лише попередні взаємодії, спільна фільтрація має проблеми «холодного запуску» — неможливості рекомендувати новим користувачам або рекомендувати нові елементи іншим користувачам, оскільки багато користувачів або елементів мають занадто мало взаємодій.

Цей недолік можна усунути різними способами:

– Рекомендувати рандомні предмети новим користувачам або нові предмети звичайним користувачам,
– Рекомендувати популярні предмети новим користувачам або новинки найактивнішим користувачам.

Колаборативна фільтрація заснована на моделі - оцінює параметри статистичних моделей шляхом вимірювання користувачів, створених за допомогою таких методів, як байєсовські мережі, кластеризація, латентні семантичні моделі, такі як сингулярна декомпозиція, ймовірнісний латентний семантичний аналіз, приховані розподіли Діріхле та процеси прийняття рішень Маркова для надання рекомендацій на основі модель. Моделі розробляються з використанням інтелектуального аналізу даних, алгоритмів машинного навчання для пошуку шаблонів на основі навчальних даних. Методи, які використовують основні компоненти, можуть зменшити кількість параметрів у моделі. Цей підхід є більш комплексним і точнішим у прогнозах, оскільки він допомагає виявити основні фактори, які пояснюють спостережувані оцінки. Такий підхід має низку переваг. Він обробляє розріджені матриці краще, ніж методи на основі сусідства, що, у свою чергу, сприяє масштабованості великих наборів даних.

Гібридний підхід – цей підхід поєднує підходи на основі сусідства та моделі. Гібридний метод є найпоширенішим методом у розробці систем рекомендацій для комерційних веб-сайтів, оскільки він допомагає подолати обмеження оригінального методу (на основі сусідства) і покращити якість прогнозування. Цей підхід також може подолати проблеми розрідженості даних і втрати інформації. Однак цей метод складний у реалізації та застосуванні, і вартість висока. Зі збільшенням кількості користувачів системи також виникають проблеми з масштабованістю. Наприклад, з 10 мільйонами покупців і 1 мільйоном товарів алгоритм спільної фільтрації з таким рівнем складності є надто складним для обчислення. Крім того, багато систем повинні миттєво відповідати на всі онлайн-запити користувачів, незалежно від історії їх покупок і рейтингів, що вимагає більшої масштабованості.[5]

У системі рекомендацій, де кожен може оцінювати, люди можуть оцінювати свої теми\товари позитивно, а своїх конкурентів — негативно. Крім того, системи рекомендацій починають сильно впливати на продажі та прибутки, оскільки вони широко використовуються на комерційних веб-сайтах. Це призвело до того, що недобросовісні постачальники намагалися шахрайським шляхом підвищити рейтинги своїх продуктів і знизити рейтинги своїх конкурентів. Спільна фільтрація спочатку була задумана для збільшення різноманітності, дозволяючи користувачам відкривати нові продукти з нескінченного пулу. Однак деякі алгоритми, особливо основні з продажів і рейтингів, створюють дуже складні умови для просування нових і маловідомих товарів, оскільки їх замінюють давно популярні на ринку товари. Це, у свою чергу, лише посилює ефект «багаті стають ще багатшими», що призводить до зменшення різноманітності.

Змішана система рекомендацій - на відміну від методів спільної фільтрації, які базуються лише на взаємодії між елементами та користувачами, методи на основі вмісту використовують додаткову інформацію про користувачів та елементи. Наприклад, у системі рекомендацій фільмів додатковою інформацією може бути вік, стать, адреса електронної пошти, країна та інша особиста інформація користувача. Щодо елементів, то це може бути жанр фільму, актори, режисер, сценарист, тривалість тощо.

Основна ідея методів, заснованих на вмісті, полягає в створенні моделей на основі існуючих методів характеристики (ознаки), що пояснюють взаємодію між елементами та користувачем. Наприклад, жінки віддають перевагу драм, а чоловіки — трилери. У цьому випадку модель робитиме прогнози для користувача на основі інформації в профілі користувача та даватиме рекомендації на основі цієї інформації. На методи, засновані на вмісті, набагато менше впливає проблема «холодного запуску», ніж на спільну фільтрацію. Ідея змішаного типу полягає в тому, щоб поділяти користувачів на підгрупи(Стать, вік), а потім уже на основі даних груп людей пропонувати рекомендації, що більш конкретно відповідають їх інтересам.[6]

1.3 Дослідження алгоритмів колаборативної фільтрації

У світі часто доводиться стикатися з проблемою рекомендації товарів чи послуг користувачам будь-якої інформаційної системи. Раніше рекомендації складалися шляхом узагальнення найпопулярніших продуктів: це можна спостерігати навіть зараз, відкриваючи той же Google Play. Але потім такі рекомендації стали замінюватися адресними пропозиціями: користувачам рекомендували не просто популярні товари, а ті, які їм сподобалися б.

Таблиця 1.1 – порівняльна характеристика переваг та недоліків рекомендаційних алгоритмів

	Переваги	Недоліки	Застосування
Колаборативна фільтрація	1. Використовується оцінка та смак користувачів 2. Не залежить від контенту сервісу	1. Проблема холодного старту (нового користувача) 2. Розрідженість матриці оцінок	• Інформаційні портали • Блоги
Контентна фільтрація	1. Немає проблем з холодним стартом 2. Немає проблеми розрідженості даних	1. Залежить від контенту 2. Вузько спрямована 3. Не має можливості спиратися на оцінку та смаки користувача	• Інтернет-магазин • Блоги • Інтернет-курси
Гібридна фільтрація	1. Майже відсутні проблеми Колаборативної та Контентно базової фільтрації 2. Висока швидкість алгоритму	1. Висока складність розробки 2. Вузько спрямована	• Складні соціальні мережі • Великі інтернет-магазини

Рекомендаційні системи існують досить давно. Youtube, Facebook, Амазон та багато інших надають своїм користувачам ті чи інші рекомендації. Це не тільки допомагає їм показувати користувачам релевантні продукти, а й дозволяє виділитися на тлі конкурентів.

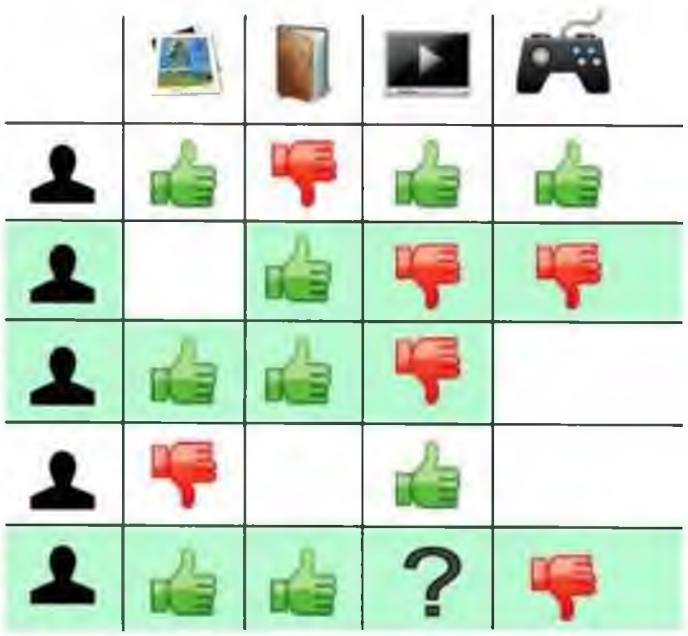

Рисунок 1.3 Приклад колаборативної фільтрації

Основна ідея алгоритмів колаборативної фільтрації полягає в пропозиції нових елементів для конкретного користувача на основі попередніх переваг користувача або думки інших однодумців користувача. На сьогоднішній день дослідники розробили цілий ряд алгоритмів КФ, які можна розділити на дві основні категорії:

1. Методи, засновані на аналізі наявних оцінок, – анамнестичні методи (Memory-based). Ці алгоритми грунтуються на статистичних методах, щоб знайти групу користувачів близьких до цільового користувача. Цей підхід ще називають метод найближчих сусідів: використання попередніх оцінок, зроблених клієнтом, і аналіз оцінок інших користувачів, які мають подібні ознаки. Тоді рекомендації (прогноз) для цільового користувача формуються на підставі обчислення якоїсь міри схожості за всіма накопиченими даними.

Формально роботу рекомендаційної системи на основі колаборативної фільтрації можна зобразити наступним чином

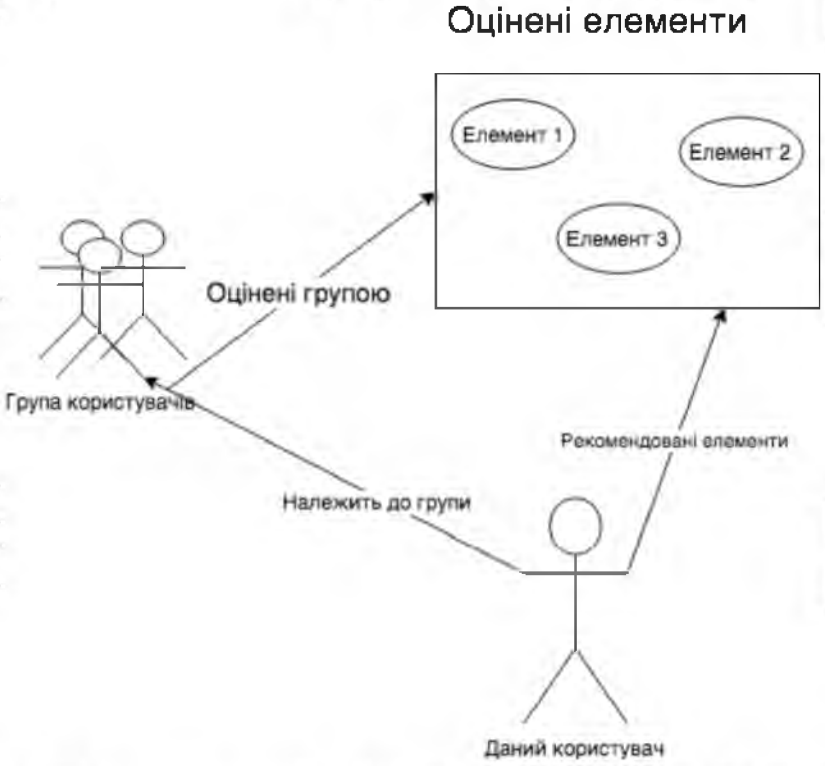

Рисунок 1.4 Принцип роботи рекомендаційної системи на основі колаборативної фільтрації

2. Методи, засновані на аналізі моделі даних, – модельні методи (Modelbased). У цьому випадку спочатку за сукупністю оцінок формується описова модель переваг користувачів, товарів і взаємозв'язку між ними, а потім формуються рекомендації на підставі отриманої моделі. Процес формування рекомендацій розбитий на два етапи: ресурсномістке навчання моделі у відкладеному режимі і досить просте обчислення рекомендацій на основі існуючої моделі в реальному часі. Ці алгоритми можуть бути засновані на імовірнісному підході, кластерному аналізі, аналізі прихованих чинників.[7]

3. Методи, засновані на об'єднанні попередніх алгоритмів, – гібридні методи. Ці підходи в свою чергу можуть бути розбиті далі на групи методів.

Так, методи на основі сусідства (близькості) поділяються на аналіз:

– подібності користувачів (User-based);

– подібності елементів (Item-based).

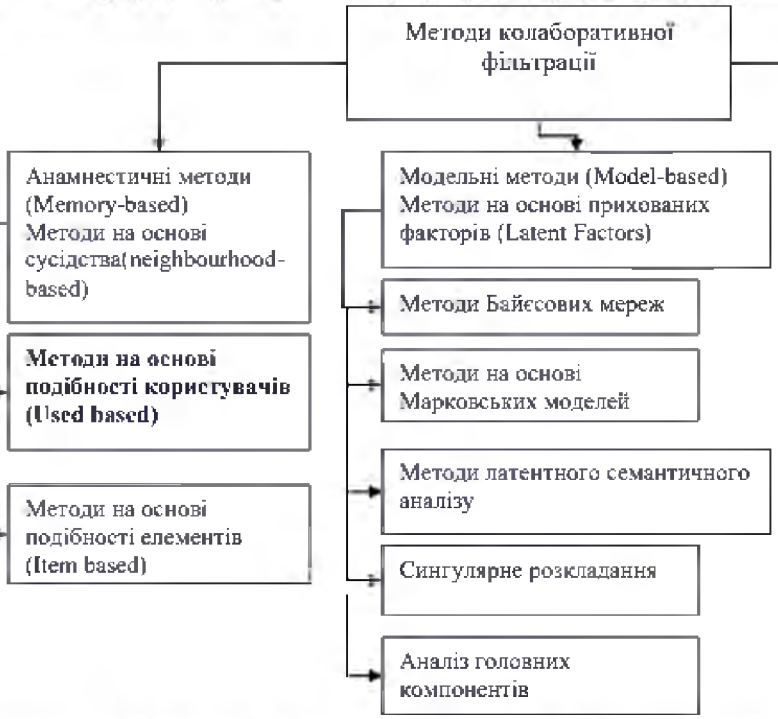

Рисунок 1.5 Класифікація методів колаборативної фільтрації

Колаборативна фільтрація має недоліки в холодним стартом, та тривалістю результат в рекомендацій. Алгоритм SVD (Singular Value Decomposition), створений для покращення результатів фільтрації.

Теорема про сингулярний розклад стверджує, що будь-якої матриці A розміру n на m, існує розкладання в добуток матриць : U, Σ та V^t :

$$\underset{n\times m}{A} = \underset{n\times n}{U} \times \underset{n\times m}{\Sigma} \times \underset{m\times m}{V^T}$$

$$UU^T = I_n, VV^T = I_m$$

$$\Sigma = diag(\lambda_1, \dots, \lambda_{\min(n,m)}), \lambda_1 \geq \cdots \geq \lambda_{\min(n,m)} \geq 0$$

Матриці U і V - ортогональні, а Σ - діагональна (хоч і не квадратна).

Лямбди в формулі розташовані за спаданням. Окрім звичайного розкладання існує ще усічене, коли з усіх лямбд залишаються лише перші ***d*** чисел, а інші вважаються рівними нулю.

$$\lambda_1, \ldots, \lambda_{\min(n,m)}=0$$

Це рівносильно тому, що в матрицях ***U*** і ***V*** залишати лише ***d*** стовпців, а матрицю **Σ** обрізати до квадратної розмірності ***d*** × ***d.***

$$\underset{n\times m}{A'} = \underset{n\times d}{U'} \times \underset{d\times d}{\Sigma'} \times \underset{d\times m}{V'^T}$$

На практиці нова матриця *A'* дуже добре наближає вихідну матрицю *A* та, тим більше, є найкращим наближенням.

Формула спрощується, вважаючи добуток перших двох матриць за одну:

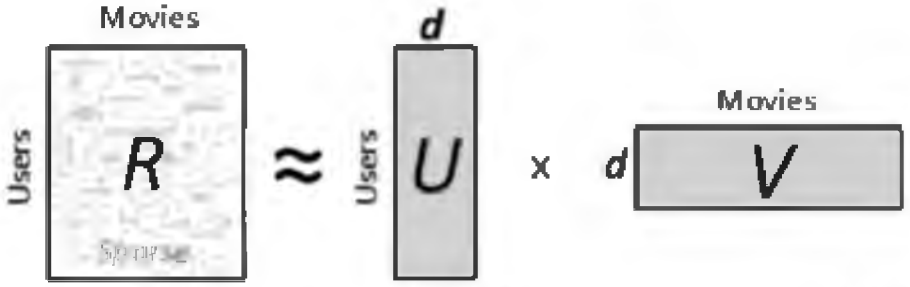

Рисунок 1.6 Приклад застосування SVD для рекомендацій

Тобто отримано наступний алгоритм: щоб передбачити оцінку користувача ***U*** для фільма *V*, береться деякий вектор ***p*** (набір параметрів користувача) та вектор ***q*** (набір параметрів фільму). Їх скалярний добуток і буде необхідним передбаченням.

Наприклад, у векторі користувача на першому місці буде стояти параметр, який відповідає за стать користувача (хлопчик чи дівчинка), а на другому - його вік. У фільмів на аналогічному першому місці буде стояти параметр, який вказує на те, чи цей фільм подобається більше хлопчикам/дівчатам, а інший - для якої вікової категорії цей фільм

більше підходить. З цього видно, що цей алгоритм дозволяє не тільки передбачувати оцінки, також можливо передбачувати скриті інтереси користувачів та параметри об’єктів.

Матриця оцінок невідома, тому однозначний SVD розклад знайти неможливо. Для того щоб створити рекомендаційну систему, необхідно знайти деякий вектор користувача та вектор фільму з різними параметрами. Оскільки задані попередні оцінки користувачів, їх можна використати як навчальну вибірку. А для знаходження результатів, близьких до існуючих, необхідно обрати оптимальні параметри моделей фільмів та користувачів. [8]

Пошук оптимальних параметрів можна прискорити використавши вже відомі алгоритми градієнтного спуску та метод найменших квадратів.

Для сервісів, де кількість користувачів набагато більше, ніж об'єктів (товарів, музики, статей та іншого) доцільно використовувати Item-Base метод.

Алгоритм на основі елементів – користувач характеризується об’єктами які він подивився або оцінив. Це тип алгоритму системи рекомендацій, який використовує схожість елементів для складання рекомендацій щодо продуктів. Для кожного такого об’єкта алгоритм знаходить найбільш ревалентні об’єкти по переглядам та оцінкам користувачів. І в наступному етапі всі об’єкти “сусіди” об’єднуються в один список, виключаючи з результатів той об’єкт який уже був переглянутий або оцінений раніше. І уже з фінального списку формується рекомендація. Таким чином участь в формуванні

рекомендацій беруть всі користувачі, котрим сподобався той чи інший об'єкт.

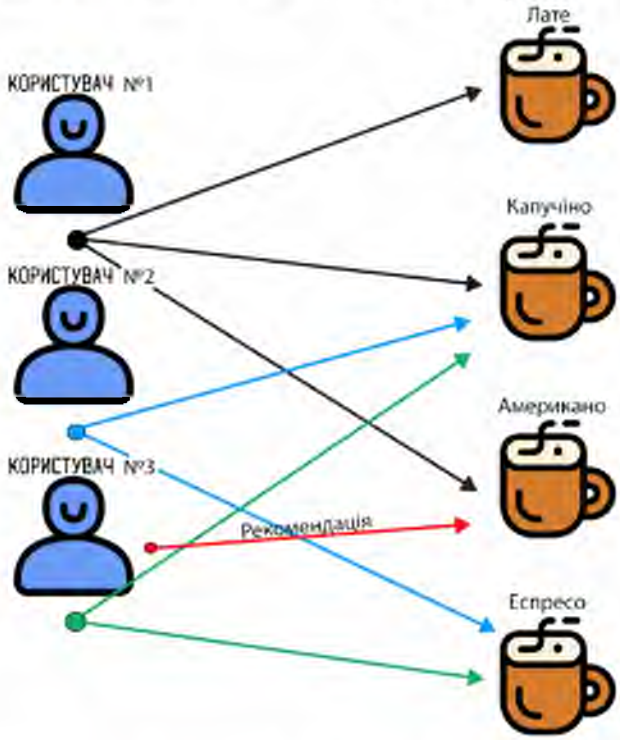

Рисунок 1.7 Робота колаборативної фільтрації на основі елементів на спрощеному прикладі

На сьогоднішній день при створенні рекомендаційних систем використовуються дві основні стратегії фільтрація вмісту та колаборативна фільтрація. Варто відзначити, що на практиці зазвичай використовуються гібридні методи, що поєднують в собі переваги приведених вище підходів. Комбінування методів в рекомендаційних системах гібридного типу останнім часом дуже поширений. Основний принцип методу в тому, щоб використовувати характеристики фільтрації вмісту і колаборативної фільтрації (наприклад, жанр музичної композиції і професію користувача) в одній рекомендаційній системі. До таких підходів належить уніфікований імовірнісний метод, який застосовує латентно-семантичний аналіз. Такі гібридн

рекомендаційні системи побудовані на колаборативній складовій, але також включають в призначений для користувача профіль деякі дані фільтрації вмісту. Після цього дані служать підставою для обчислення схожості користувачів замість загальних оцінених об'єктів. Такий підхід дозволяє уникнути проблеми з недостатньою кількістю даних, що виникає через малу кількість пар користувачів з досить великим числом загальних оцінених об'єктів. Також, метод дозволяє рекомендувати користувачеві не тільки об'єкти з позитивними відгуками від інших користувачів, а й ті об'єкти, які користувачеві можуть сподобатися виходячи з його особистих переваг.

Гібридні рекомендаційні системи іноді доповнюють методами, які використовують базу знань для поліпшення якості рекомендацій і розв’язання основних проблем більшості рекомендаційних систем (холодний старт і інші).

2. АНАЛІЗ СУЧАСНИХ РЕКОМЕНДАЦІЙНИХ СИСТЕМ НА ОСНОВІ КОЛАБОРАТИВНОЇ ФІЛЬТРАЦІЇ

При створенні сучасного програмного забезпечення слід детально вивчити предметну область розробки та ознайомитися з уже наявними рішеннями на ринку. Це потрібно для того, щоб уникнути поширених проблем, а також для прискорення розробки і використовувати найбільш новітні технології.

Як уже і було зазначено раніше існують явні і не явні типи збору інформації про користувача. Якщо розглянути найсучасніші, найпрогресивніші продукти від світових компаній то можна сказати, що вони використовують змішану систему і навіть в них не все ідеально через те, що важко догодити абсолютно всім користувачам. Через цю проблему, все більш популярних сервісів проводить так би мовити анкетування. Розглядаючи на приклад ютуб, можна досконально прослідити процес еволюції рекомендаційної системи: спочатку зазвичай рекомендувався лише той відеоматеріал, котрий набирав найбільше переглядів, а також лайки/коментарі. Дане базове ядро залишилося, проте зазнавало змін. Наприклад, в один із не найкращих періодів для ютубу рекомендації були більше засновані на тому, що чим більше користувачів відкриє певне відео, тобто чим більше click-through rate(відношення кількості кліків вашого оголошення до кількості його показів), тим краще воно продвигається алгоритмами і власне рекомендується людям. Через це виникла проблема з тим, що почалась епоха клікбейту, адже для кожного відео, перед тим як його переглянути є прев'ю - невелике зображення, яке в цілому відображає суть відеоролику. Прев' ю повністю перестало відображати суть відеоролику і в багатьох випадках мова абсолютно була про інші речі, через це дуже страждали алгоритми ютуб. Загальновідомо, відеохостинг отримує свій заробіток через вбудовану рекламу, яка інколи прям зовсім

недокучлива, але все ж таки дозволяє розвиватися самій платформі. В подальшому розвитку, доволі логічна модифікація була зроблена: рекомендувати не тільки той контент, котрий має шалену "клікабельність", але ще враховувати середню тривалість перегляду. Завдяки цьому в рекомендації ютуб може потрапити абсолютно будь-який відеоролик і рекомендуватися людям які дійсно в цьому зацікавлені.

Зазвичай анкетування проводиться з найбільш активними користувачами, наприклад ті, хто залишає коментарі, лайки, скарги. Це допомагає навчати рекомендаційну систему і вберегти систему від спаму.

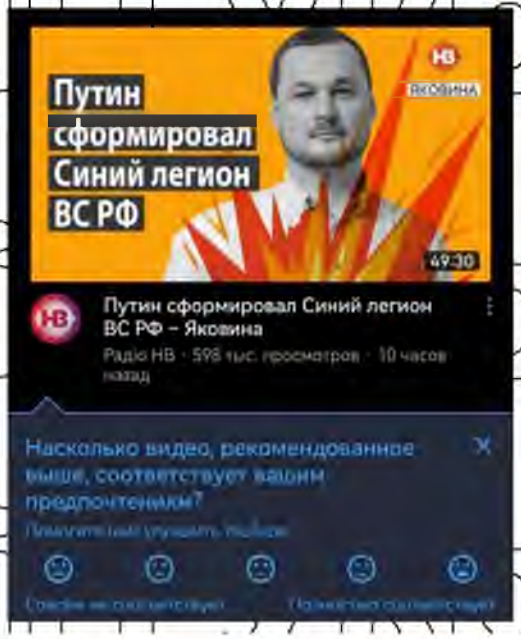

Рисунок 2.1 Приклад анкетування від YouTube

Даний метод застосовують все більше компаній “гігантів”, бо якими б досконалими не були алгоритми чи продвинуті нейронні мережі, важливо дослідити вподобання в певних проміжках часу, таким чином, щоб не втрачалася актуальність та зацікавленість в рекомендованих продуктах.

Для повноцінності огляду сучасних рекомендаційних систем необхідно вивчити достатню кількість прикладів, щоб увібрати в свій продукт найпередовіші технології.

YouTube вже більше можна позиціонувати як соціальну мережу, але більш спрощену та зосереджену на відеоконтенті. Найбільш прямий конкурент YouTube по популярності та типу контенту є TikTok. TikTok – це та платформа, яка змогла досягти надзвичайної популярності, завдячуючи цьому своїм алгоритмам. Даний сервіс зміг організувати рекомендаційні алгоритми настільки, що цьому може позаздрити будь-яка інша компанія зі своїми продуктами. В чому ж інновація TikTok? Насправді досить складно відповісти на питання, тому що було ввібрано всі найсучасніші методи створення рекомендацій і вдало скомпоновано. Колаборативна фільтрація в цьому відіграє не останню роль. Для більшої персоналізації пулу безкінечного потоку відео на початку роботи з програмою є вибір власних інтересів.

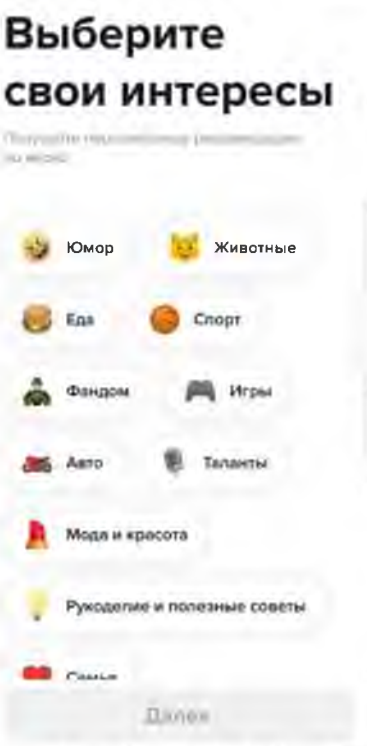

Рисунок 2.2 Вікно TikTok при першому вході в додаток

Через деякий час відбувається ще більша персоналізація по інтересам, за допомогою іншого опитування, де користувач вибирає мови, які він розуміє, що досить цілком логічно – рекомендувати

матеріали користувачам тільки тими мовами, які йому зрозумілі. Попри це, продумана концепція глобальних рекомендацій, коли відео контент потрапляє до користувачів зі всього світу, це в основному контент, який не “прив’язаний” ні до якої мови і в переважній більшості показує красу природи або життєві жарти. Для збереження актуальності рекомендаційної системи час від часу проводиться анкетування з питаниям: “Чи сподобався вам даний відеоматеріал?”

Головний показник того, чи сподобався контент користувачу – перегляд до самого закінчення відео та чи було залишено лайк\коментар, а також поширення серед друзів.

Ще один ресурс, який набув широкої популярності – Instagram. На початковому етапі це було місце, де поширювався контент звичайних користувачів(Фото, відео). З плином часу сервіс перетворився у великий бізнес агломерат, стали доступні бізнес акаунти і дуже багато зазвичай невеликих бізнесів почали створювати та розвивати свої акаунти. Інтегрована система реклами, рекомендацій, безкоштовності дозволила використовувати свої інстаграм сторінки замість персонального сайту і мати шалені охоплення та систему розвитку.

Рекомендації створюються завдяки штучному інтелекту, котрий аналізує кожне зображення, та має представлення про вміст публікації і додатково використовує хештеги та опис для персоналізації контенту. Якщо швидкість інтернету замала, користувач все рівно може дізнатися про саму суть контенту, з текстового опису.

Також доволі цікава стрічка новин в додатку, спочатку йдуть пости ваших друзів, бо людей безпосередньо більше цікавлять речі, які відбуваються довкола них і все це подається не в хронологічному порядку, а по мірі цікавості. Тому для цього в Instagram також є анкетування, яке дозволяє ставати алгоритмам краще.

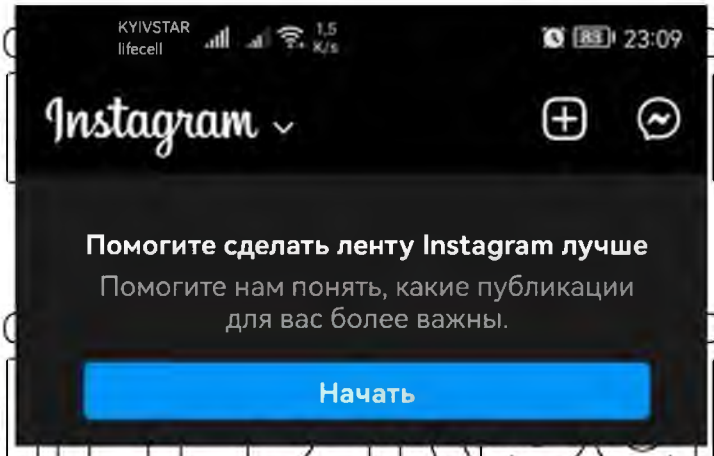

Рисунок 2.3 Приклад запрошення до опитування від Instagram

Рисунок 2.4 Приклад опитування від Instagram

З принципами того як відео та фото вміст опрацьовувати, щоб система рекомендацій показувала гарний результат розглянуто, а як щодо аудіоконтенту?

Одним із найбільш відомих стрімінгових сервісів із музикою та подкастами є Spotify. Для створення рекомендацій тижня сервіс використовує три моделі:

— Колаборативна фільтрація, котра аналізує вашу модель поведінки та інших користувачів;

— Обробку природної мови для аналізу тексту;

— Штучний інтелект, котрий аналізує аудіо файли.

Так як в Spotify немає конкретної системи оцінок то користувачі залишають фідбек у вигляді метаданих: кількість прослуховувань, лайк або пропуск трека, відвідування сторінки артиста, прослуховування конкретного альбому, тощо.

Обробка природної мови – здатність комп'ютера розуміти людську мову. Spotify постійно шукає пости, а також тексти про ту чи іншу музику. Це робиться для того, щоб зрозуміти якими словами говорять люди про конкретного артиста чи пісню. Крім того, сервіс дивиться які ще артисти і пісні виникають в таких описах. Подібно колаборативній фільтрації обробка природної мови використовує опис і їх значимість для створення вектора, який є зборомом всіх даних про пісню. Це і дозволяє зрозуміти ступінь схожості двух чи більше треків.

Не дивлячись на ефективну роботу цих двох методів, штучний інтелект працює для покращення точності музичних рекомендацій. На відміну від перших двох алгоритмів, штучний інтелект дозволяє враховувати свіжо завантажені пісні. Це зроблено наприклад для того, якщо трек має умовно мало прослуховувань та оцінок і мало згадувань в мережі, що само собою виключає можливість застосовувати колаборативну фільтрацію або обробку природної мови.

Але ж як аналізуються такі аудіо дані? Тут на допомогу приходить згорткова нейронна мережа, яка використовується зазвичай в програмах по розпізнаванню обличчя. Для своїх завдань Spotify перенавчили нейронку під роботу з аудіо даними замість пікселів. Після обробки нейронна мережа видає характеристику про пісню: розмір, гармонічність, форма, темп, гучність. Зрівнюючи ці ключові характеристики, система рекомендацій може зрозуміти ці фундаментальні схожості між різними треками.

Якщо брати лідера на українському ринку по роздрібній торгівлі то безумовним чемпіоном є інтернет-магазин Rozetka. Це зумовлено не тільки масштабними рекламними кампаніями, інтуїтивному інтерфейсу та швидкості. Rozetka змогла агломерувати в собі багато інших магазинів, що в свою чергу дозволило вийти магазину на ринок з абсолютно будь-якими товарами, не обмежуючись лише побутовою технікою та електронікою, а досягти такого результату в першу чергу допомогли рекомендаційні системи. Яскравим прикладом застосування колаборативної фільтрації є розділ “Разом з цим товаром купують”, магазин використовуючи дані поведінки користувачів, підбирає найбільш ревалентні товари, котрі можуть зацікавити.

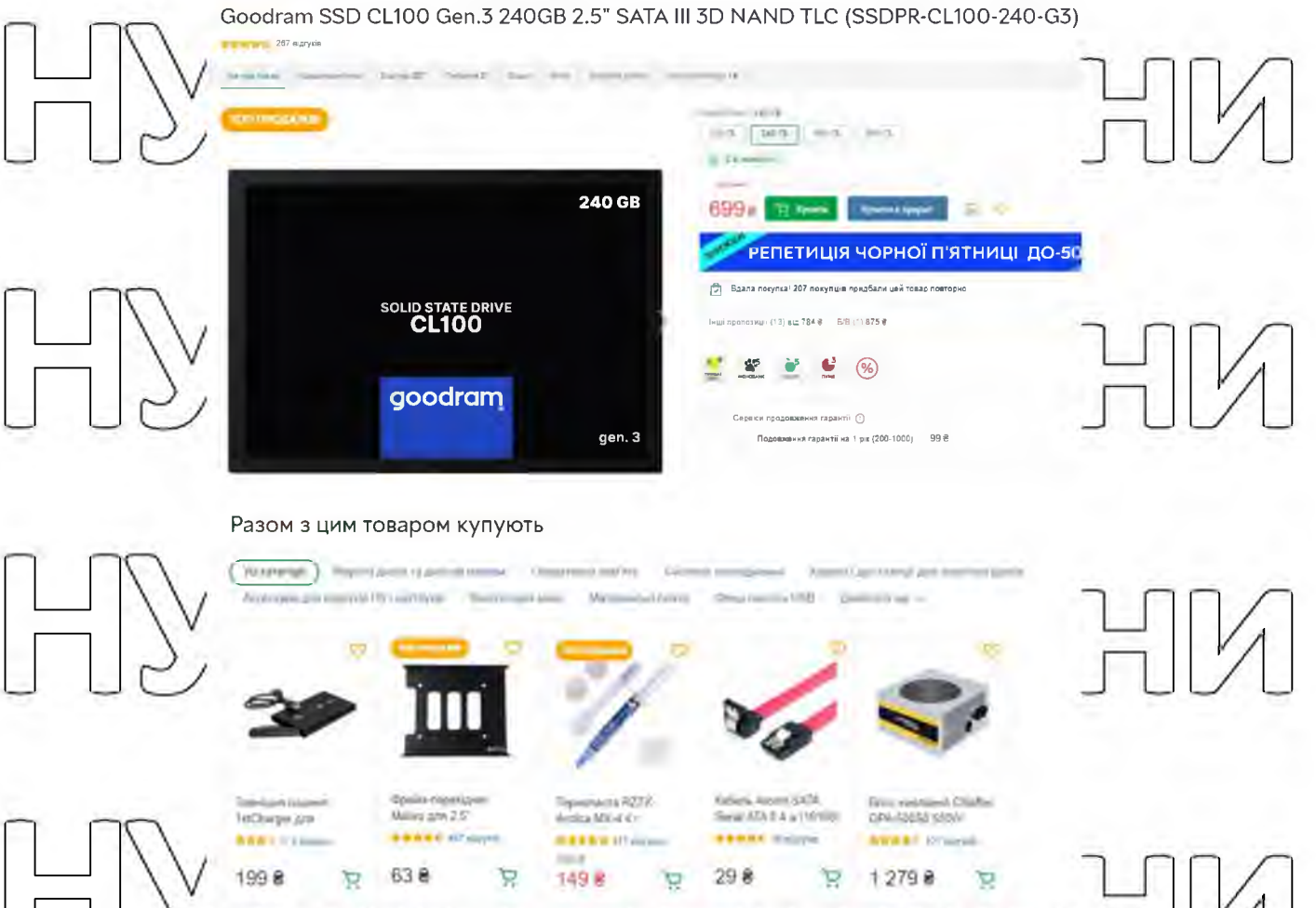

Рисунок 2.5 Ревалентна рекомендація до товару магазину Rozetka

Як можна спостерігати на рис. 2.5, до швидкісного жорсткого накопичувача пропонують придбати зовнішню кишеню, кабелі, перехідники, що цілком відповідає потребам користувачів.

Також на основі даних, про те якими товарами ви цікавились, надсилаються комерційні персоналізовані email розсилки. (Рис. 2.6) Що в свою чергу збільшує конверсію придбаних товарів на сайті, покращує лояльність та продвигає бренд.

Рисунок 2.7 Приклад комерційної персоналізованої Email розсилки

3. ОПИС МОВИ JAVA ТА ПРОЦЕС РОЗРОБКИ ПРОГРАМ

При створені програми, послідовно створюємо підходи для вирішення задач, які потрібно виконати для того щоб створити програму. З першу потрібно визначитись з технічним планом :

- Створення макету програми.
- Визначення мови програмування та алгоритмів розробки
- Захист створеного застосунку
- Розробка прототипу алгоритму та самої програми.

Для розробки додатку потрібно також розуміння самої мови програмування в даному прикладі будемо використовувати мову програмування Java і створювати мобільний додаток на її основі.

3.1 Розгляд захисту інтернет-додатків

Для того, щоб зрозуміти, які найбільш вразливі місця загалом при створенні будь-якого Інтернет застосунку, необхідно розглянути рівні мережевих атак згідно моделі OSI. Мережева модель OSI (Open Systems Interconnection Basic Reference Model) - це модель мережевого стека (сховища) мережевого протоколу OSI / ISO. За допомогою цієї моделі різні мережеві пристрої можуть бути підключені між собою. Модель визначає різні рівні взаємодії системи, а кожен рівень виконує певні функції в цій взаємодії. Найпоширенішим є опис рівня моделі OSI, починаючи з сьомого рівня (так званий прикладний), в якому користувальницькі програми отримують доступ до мережі. Модель OSI закінчується рівнем 1 (фізичний), який визначає стандарти, встановлені незалежними виробниками для носіїв передачі даних. Будь-який протокол моделі OSI повинен взаємодіяти з протоколом на його рівні, або протолоком вище та / або нижче від нього. Взаємодія на рівні

називається горизонтальною, а хоч один рівень вгору-вниз - вертикальною.

Таблиця 3.1 Модель OSI

Дані	7 Прикладний application	Доступ до мережевих служб
	6 Представлень presentation	Представлення і кодування даних
	5 Сеансовий session	Управління сеансом зв’язку
Сегменти	4 Транспортний transport	Прямий зв’язок між кінцевими пунктами і надійність
Пакети	3 Мережевий network	Визначення маршруту і логічна адресація
Кадри	2 Канальний data link	Фізична адресація
Біти	1 Фізичний physical	Робота з середовищем передачі, сигналами і двійковими даними

Після дослідження можна зрозуміти, які місця моделей OSI є найбільш вразливими. (Табл. 3.2)

Таблиця 3.2 Види мережевих загроз в залежності від типу OSI

Рівень	Вид мережевої загрози
Фізичний	DDoS атака
Канальний	DDoS атака
Мережевий	Свідомо неправильна маршрутизація пакетів, MITMARP Poisoning
Транспортний	Переповнення буферу пакетів ОС послідовністю великих і неповних пакетів, IP Spoofing
Сеансовий	SYN-Flood(Спам TCP запитами без очікування відповіді)
Представлень	SSL Hijack
Прикладний	Віруси, MItB, Cookie Injection

Захист від стороннього втручання є дуже великим пріоритетом, адже нікому не подобається, коли ресурс не працює або ж коли конфіденційні дані потрапляють в мережу. Є чудові ресурси по типу Cloudflare, котрі надають мережеві послуги доставки контенту, щоб людина з будь-якої точки світу могла повноцінно користуватися вашим продуктом, та захистом від DDOS атак. Підключення такого сервісу відбувається доволі легко – доменне ім'я прив'язується до ip адреси Cloudflare, а в самих налаштуваннях вказується на яку ip адресу спрямовувати користувача після перевірки запиту на легітимність. Це доволі зручно та швидко, проте не безкоштовно, тому для дипломної роботи буде доцільніше використовувати не сайт, а створення мобільного додатку, де визначити на які ip адреси йдуть запити доволі проблематично.

3.2 Опис мови програмування java

Невеликий опис мови Java – це технологія яка використовується для розробки додатків які роблять роботу в мережі Інтернет більш захоплюючою і зручною та не тільки в інтернеті. Java відрізняється від інших мов програмування простим навчанням самої мови так і об'єктно-орієнтованому програмуванні, простіше думати з точки зору класів та об'єктів. Також Java має велику кількість API, з цими пакетами даних можливо працювати як з графікою, звуком та іншими.

В мові Java є вбудованим Garbage collection (автоматичне керування пам'яттю, що займають невикористовувані об'єкти), що є великим плюсом для абсолютних новачків. Управління пам'яттю - це велика справа на початку навчання програмуванню.

3.3 Основні особливості мови

Дизайнери Java вирішили використовувати комбінацію компіляції та інтерпретації. Програми, написані на Java, компілюються в машинний мову, але це машина мова для комп'ютера, якого насправді не існує. Цей так званий «віртуальний» комп'ютер відомий як Java Віртуальна машина або JVM. Машинна мова для віртуальної машини Java називається Java байт-код. Перевагою подібного способу виконання програм є повна незалежність байт-коду від операційної системи і устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина. Іншою важливою особливістю технології Java є гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання.

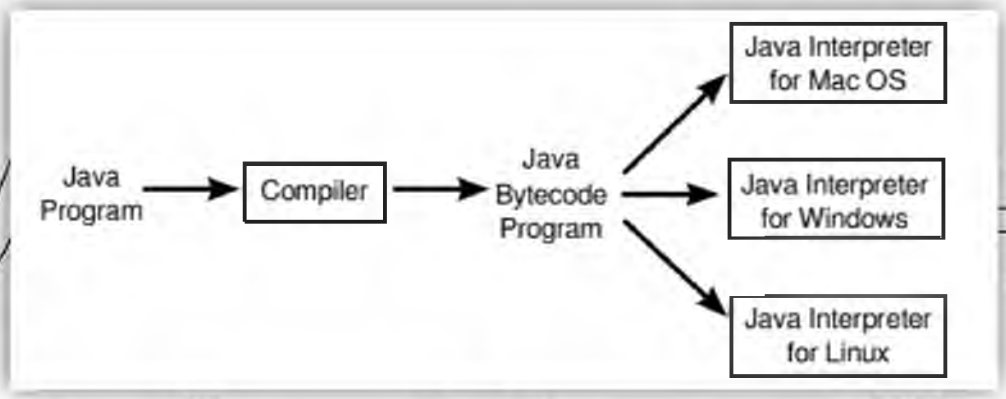

Рис 3.1 – Приклад Java програми її компіляції на різних системах.

3.4 Розробка програм на мові Java

Базова система розробки для Java-програмування зазвичай називається JDK (Java Development Kit). Він є частиною Java SE, стандартної версії Java (на відміну від Java EE для серверів або Java ME для мобільних пристроїв). Зверніть увагу, що Java SE поставляється в двох версіях: версія комплекту розробника (JDK) і версія середовища виконання (JRE). Середовище виконання можна використовувати для запуску програм на Java, але вона не дозволяє вам компілювати ваші власні програми на Java. Комплект розробника включає в себе середовище виконання, але також дозволяє вам компілювати програми. Java була розроблена Sun Microsystems, Inc., яка в даний час є частиною корпорації Oracle. Oracle робить JDK для Windows, Mac OS і Linux доступною для безкоштовного скачування на своєму веб-сайті Java. Багато комп'ютерів Windows поставляються з уже встановленим середовищем виконання Java, але вам може знадобитися встановити JDK. Деякі версії Linux поставляються з JDK, встановленим за замовчуванням або на установчому носії. Mac OS в даний час не поставляється з попередньо встановленою Java. [9]

Багато сучасних користувачів комп'ютерів знаходять середовище командного рядка досить дивною і неінтуїтивним. Це безумовно, дуже відрізняється від графічного інтерфейсу користувача, який використовується більшістю людей. Однак для вивчення основ середовища командного рядка потрібна лише невелика практика його використання.

Основними командами для використання Java в командному рядку є javac і java; Javac використовується для компіляції вихідного коду Java, а java використовується для запуску автономних додатків Java. На рис. 3.2 приклад програми для виводу рандомних чисел.

```
package StartSystem;

import java.util.Random;

public class StartMain {

    static int[] product = {1,2,3,4,5,6,7,8,9,10};

    static String TEXT_product;

    public static void main(String[] args) {
        Random random = new Random();

        TEXT_product = "Товар : ";

        // Вывод рандомного товара
        int ran = random.nextInt(product.length);

        System.out.print(TEXT_product + product[ran] + '\n');

    }
}
```

```
<terminated> StartMain
Товар : 9
```

Рисунок 3.2 – Програма для виводу рандомних чисел на екран.

4. ОПИС РОЗРОБКИ ПРОТОТИПУ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

При розробці алгоритму колаборативної фільтрації для рекомендаційної системи, проаналізували математичну модель яку можливо застосувати. В даному етапі створюємо прототип на моделі алгоритму Slope One – який входить до сімейства алгоритмів для колаборативної фільтрації для аналізу різних думок, і побажань користувачів, та вироблення персональних рекомендацій.

4.1 Використання алгоритму Slope One

Короткий опис роботи алгоритму Slope One - для істотного зменшення ефекту перенавчання, підвищення продуктивності і полегшення впровадження було запропоновано сімейство алгоритмів Slope One. Основна відмінність від регресивного аналізу відносини рейтингів двох предметів

$$(f(x) = ax + b)$$

полягає у використанні спрощеної форми регресії з одним предиктором

$$(f(x) = x + b)$$

Таким чином, предиктор - це просто середня різниця між оцінками обох предметів. Автори продемонстрували, що такий підхід в деяких випадках більш точний, ніж лінійна регресія і вимагає в 2 рази менше пам'яті.

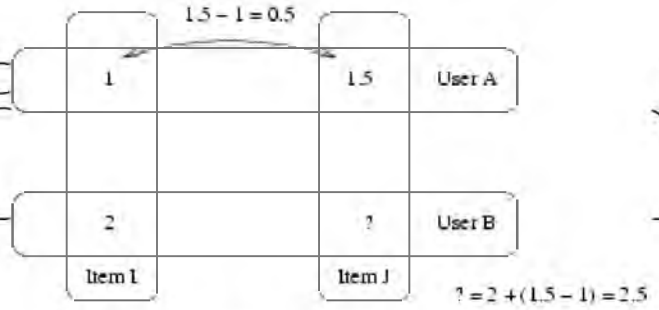

Рисунок 4.1 Приклад аналізу алгоритму Slope One

Таблиця оцінок 4.1 – алгоритму Slope One

Користувач	Продукт 1	Продукт 2	Продукт 3
User 0	5	3	2
User 1	3	4	-
User 2	-	2	5

Згідно цієї таблиці середня різниця в оцінках продукту 1 і 2 дорівнює (2 + (- 1)) / 2 = 0.5. Таким чином, в середньому продукт 1 оцінюється на 0.5 бали вище, ніж продукт 2. Те саме і для продуктів 3 і 1: середня різниця в оцінках 3.

Якщо зараз ми спробуємо передбачити оцінку **User 2** для продукту 1, використовуючи її оцінку для продукта 2, ми отримаємо 2 + 0.5 = 2.5. Аналогічно передбачаємо її оцінку для продукт 1, використовуючи оцінку, дану Продукту 3: 5 + 3 = 8. Оскільки у нас є кілька передбачуваних оцінок (**User 2** голосувала 2 рази), підсумкову оцінку ми отримаємо як зважене середнє. Для вагових коефіцієнтів будемо використовувати кількість користувачів, які дали оцінку предмету:

$$\frac{2 \times 2.5 + 1 \times 8}{2 + 1} = \frac{13}{3} = 4.33$$

Щоб застосувати алгоритм Slope One для заданих n предметів, треба розрахувати і зберегти середню різницю і кількість голосів для кожної з n^2 пар предметів.

4.2 Відтворення алгоритму Slope One – на Java

Для створення алгоритму потрібно створити 2 моделі даних. Клас User містити ім'я користувача. Клас Item містити назву елементга. На рис 4.2 та 4.3, 2 класи моделі.

```
public class User {

    private String user_name;


    public User(String user_name) { this.user_name = user_name; }

    public String getUser_name() { return user_name; }

    public void setUser_name(String user_name) { this.user_name = user_name; }

}
```

Рисунок 4.2 – Клас User

```
public class Item {

    private String itemName;

    public Item(String item) { itemName = item; }

    public String getItemName() { return itemName; }

    public void setItemName(String itemName) {
        this.itemName = itemName;
    }

}
```

Рисунок 4.3 – Клас Item

Створюємо клас вхідних даних, який буде використаний для ініціалізації даних. В класі InputData – створюємо різні продукти, на рис 4.4 приклад.

```
public static List<Item> items = Arrays.asList(
        new Item("Программист"),
        new Item("Дизайнер"),
        new Item("Повар"),
        new Item("Художник"),
        new Item("Переводчик")
```

Рисунок 4.4 – Клас InputData – продукти

Створимо трьох користувачів, які випадковим чином оцінили деякі з вищезазначених продуктів, використовуючи шкалу від 0,0 до 1,0. Де 0 означає відсутність інтересу, 0,5 – якось зацікавлене, а 1,0 - як повністю зацікавлені. В результаті ініціалізації даних ми отримаємо карту з даними призначеного для користувача рейтингу

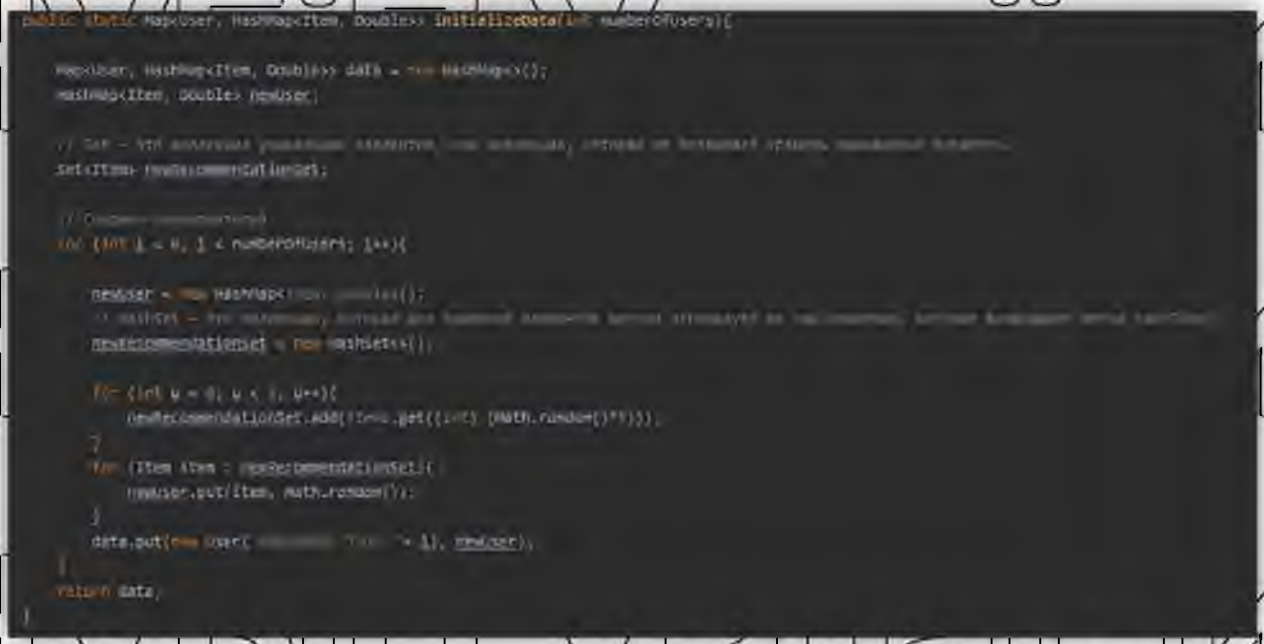

Рисунок 4.5 – метод створення користувача та отримання карти з даними призначеного для користувача рейтингу .

На основі наявних даних розрахуємо відносини між елементами, а також кількість прихильників елементів. Для кожного користувача ми перевіряємо його її рейтинг продуктів, рис 4.6.

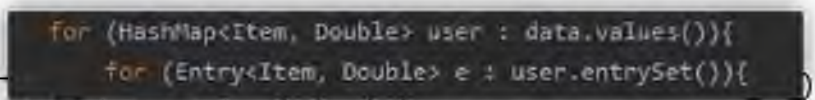

Рисунок 4.6 Перевірка для кожного користувача рейтингу продукту

На наступному кроці перевіряємо, чи існує елемент в наших матрицях. Якщо це перше входження, ми створюємо новий запис у «Map» на рис 4.7

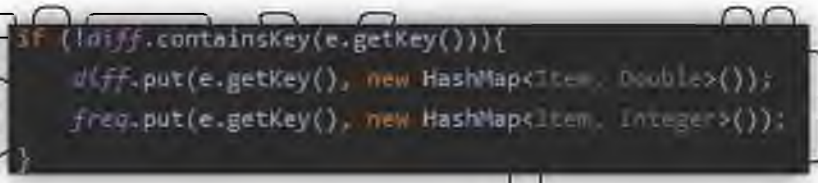

Рисунок 4.7

Перша матриця використовується для розрахунку відмінностей між призначеними для користувача рейтингами. Його значення можуть бути позитивними або негативними (оскільки різниця між рейтингами може бути негативною) і зберігаються як Double. З іншого боку, частоти зберігаються як значення Integer.

На наступному кроці збираємось порівняти рейтинги всіх продуктів. На рис 4.8

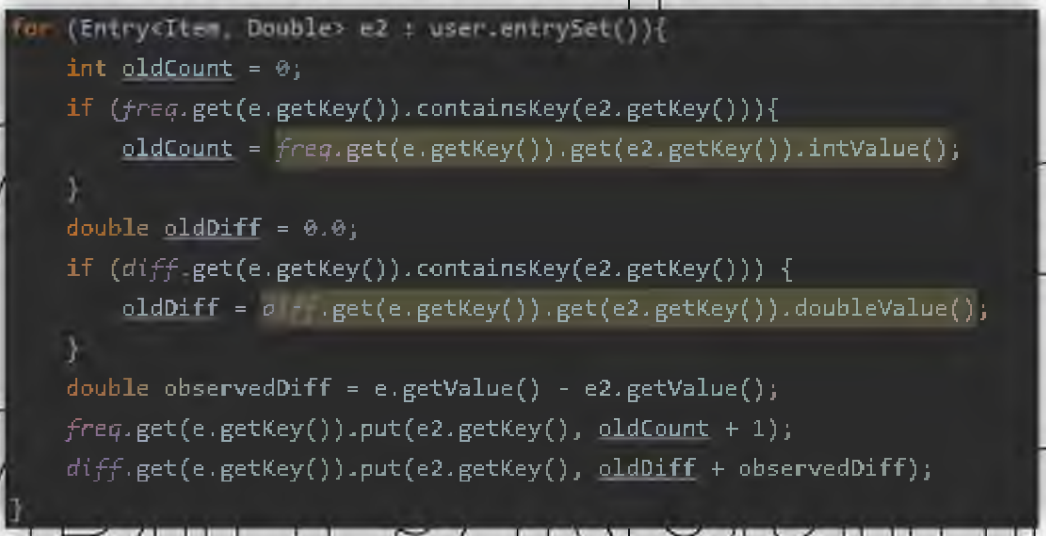

Рисунок 4.8 Порівняння рейтингів всіх продуктів

Якщо хтось оцінив продукт раніше, збільшуємо частоту на одиницю. Крім того, перевіряємо середню різницю між рейтингами товару і обчислюємо новий observedDiff. Додаємо суму oldDiff і observedDiff в якості нового значення елемента. Розраховуємо оцінки подібності всередині матриць на рис 4.9.

```
for (Item j : diff.keySet()) {
    for (Item i : diff.get(j).keySet()) {
        double oldValue = diff.get(j).get(i).doubleValue();
        int count = freq.get(j).get(i).intValue();
        diff.get(j).put(i, oldValue / count);
    }
}
```

Рисунок 4.9 – Розрахунок оцінки подібності.

Основна логіка знаходиться в тому, щоб розділити викладену різницю рейтингових продуктів на кількість їх входжень. Після цього кроку можемо розповсюдити нашу остаточну матрицю різних користувачів.

Збираємося прогнозувати всі відсутні рейтинги на основі існуючих даних(Рис 4.10). Для цього нам потрібно порівняти, рейтинги призначених для користувача, елементи з матрицею різниці, розкритою на попередньому кроці. Після цього нам потрібно підготувати «чисті» прогнози на рис 4.11.

```
for (Entry<User, HashMap<Item, Double>> e : data.entrySet()) {
    for (Item j : e.getValue().keySet()) {
        for (Item k : diff.keySet()) {
            try {
                double predictedValue = diff.get(k).get(j).doubleValue() + e.getValue().get(j).doubleValue();
                double finalValue = predictedValue * freq.get(k).get(j).intValue();
                uPred.put(k, uPred.get(k) + finalValue);
                uFreq.put(k, uFreq.get(k) + freq.get(k).get(j).intValue());
            } catch (NullPointerException e1) {
            }
        }
    }
```

Рисунок 4.10 Прогнозування відсутній даних

Рисунок 4.11 Підготування прогнозів

Хитрість, яку слід розглянути з великим набором даних, полягає в тому, щоб використовувати тільки записи елементів, які мають велике значення частоти (наприклад > 1). Якщо прогноз неможливий, його значення дорівнюватиме -1.

4.3 Компіляція алгоритму Slope One – на Java

Якщо зібрати алгоритм Slope One і про компілювати ми отримуємо такий результат на рис 4.12-14.

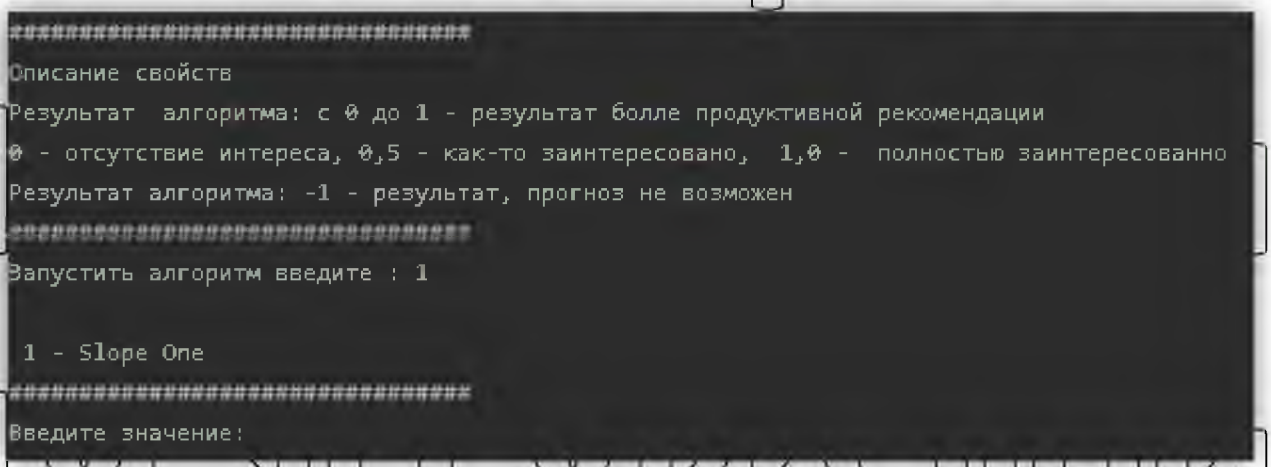

Рисунок 4.12 – Старт алгоритму

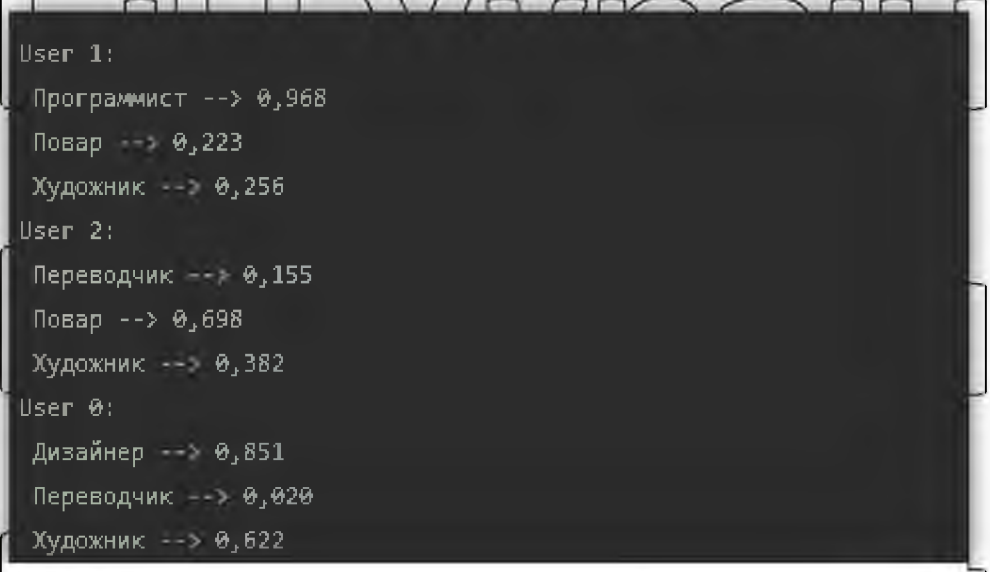

Рисунок 4.13 – Перевірка перед початком рекомендації

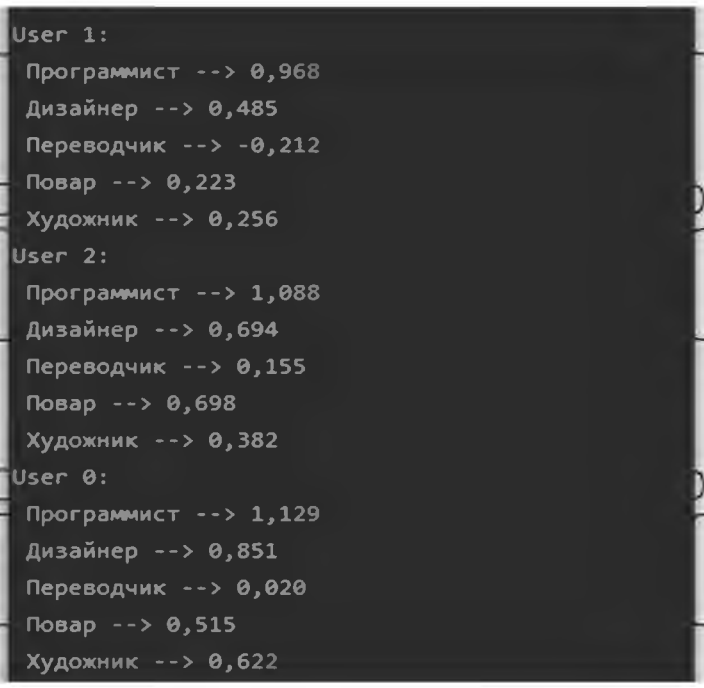

Рисунок 4.14 – Рекомендаційний прогноз

Як можливо побачити алгоритм Slope One – спочатку перевіряє те що можливо сподобалось вже користувачам, і на цій рекомендації по характеристикам що іншим користувачам також сподобалось формує рекомендацію, якщо вони навіть ще й не цікавились продуктом в даному прикладі продукт це професія.

5. ПРОТОТИП РОЗРОБЛЕНОГО МОБІЛЬНОГО ДОДАТКУ

Основною метою додатку є надання інформації користувачам, які хочуть отримувати найактуальніші новини та почати діяти з ними. Новий користувач спершу повинен зрозуміти для чого цей додаток створений, та як з ним працювати.

Після того як користувач ознайомиться з вступом додатку він зможе зробити перші кроки вже в самому додатку - перейти до авторизації, реєстрації, та перегляду подальшої інформації. Але якщо користувач закриє додаток і знов-таки відкриє його, то йому відкриється вступ з якого він почав роботу з додатком. Для вирішення даної проблеми потрібно буде використати декілька методів, для зберігання даних та також для перевірки даних, щоб виправити дану проблему.

Для того щоб в наступний запуск додатку, система не показувала «Вступ додатку» потрібно зберегти данні о тому що даний користувач вже був ознайомлений. Для цього використовуємо метод «SharedPreferences» - API для зберігання та вилучення простих значень. На рис 5.1 ми побачимо фрагмент коду цього методу, для зберігання даних. Дані цього методу

```
// Метод для проверки, Восстоновление данных при повторном запуске приложение
private boolean restorePrefData() {

    SharedPreferences pref = getApplicationContext().getSharedPreferences( name: "myPrefs", MODE_PRIVATE);
    Boolean IsIntroActivityOpenedBefore = pref.getBoolean( key: "isIntroOpened" ,  defValue: false);
    return  IsIntroActivityOpenedBefore;

}

//Метод для Сохраннение данных при нажатии кнопкпи "Начать"
private void savePrefsData() {

    SharedPreferences pref = getApplicationContext().getSharedPreferences( name: "myPrefs", MODE_PRIVATE);
    SharedPreferences.Editor editor = pref.edit();
    editor.putBoolean("isIntroOpened", true);
    editor.commit();
}
```

Рисунок 5.1 – метод зберігання простих значень

зберігаються в кешу додатку, якщо користувач захоче знов-таки ознайомитись з цими даними він просто видалить дані кешу в додатку.

Рисунок 5.2 IDEF0 діаграма роботи системи

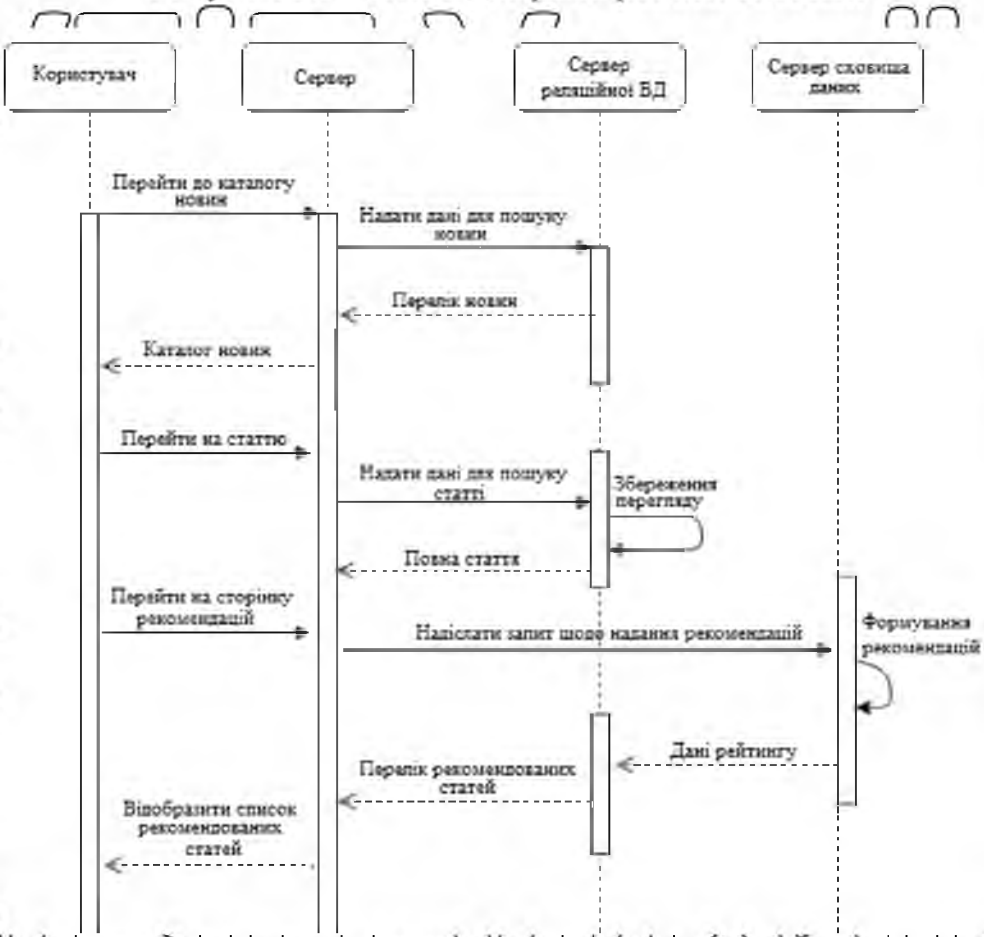

Рисунок 5.3 Діаграма послідовності

5.1 Аутентифікація

В додатку повинна бути система аутентифікації користувачів. Для того щоб користувач міг зберігати для себе яку небудь інформацію йому потрібно буде створити акаунт в якому буде водитись його дані, такі як: Ім'я, електрона адреса, пароль для захисту своїх даних. .

З'являється проблема з тим куди дані можливо зберігати, потрібно використовувати базу-даних для користувачів. В даному випадку будемо використовувати систему «Firebase» її Api - аутентифікації з Firebase за допомогою облікових записів на основі пароля на Android, та бази-даних. Версії Api використовуються на рис 5.4.[10]

```
//Firebase Auth - Authenticate with Firebase using Password-Based Accounts on Android
implementation 'com.google.firebase:firebase-auth:19.3.0'
// Firebase Storage
implementation 'com.google.firebase:firebase-storage:19.1.1'
```

Рисунок 5.4 – Firebase Api – версія.

На рис 5.5 перший метод, дані які користувач вів в пункти реєстрації. Отримуємо їх та повідомляємо користувача що його акаунт створений, якщо користувач все вірно вів. Якщо ні, то повідомляємо користувачу ту

```
private void CreateUserAccount(final String nickname, final String name, final String surname, String email, final String password) {
    // этот метод создает учетную запись пользователя с указанием адреса электронной почты и пароля
    mAuth.createUserWithEmailAndPassword(email,password)
            .addOnCompleteListener( activity: this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        // учетная запись пользователя успешно создана
                        showMessage("Account created");
                        // после того, как мы создали учетную запись пользователя, нам нужно обновить его фотографию и имя
                        // Теперь мы проверяем, является ли выбранное изображение нулевым или нет
                        if(pickedImgUri != null){
                            updateUserInfo( nickname ,name, surname, pickedImgUri,mAuth.getCurrentUser());
                        }else {
                            updateUserInfoWithoutPhoto(nickname,name,surname,mAuth.getCurrentUser());
                        }
                    }
                    else {
                        // не удалось создать аккаунт
                        // ERROR : Если адрес электронной почты неверен, ошибка отображается так же, как и метка строки
                        // TEST : " string method " + task.getException().getMessage()
                        message : Не удалось создать аккаунт - пароль должен содержать не менее 6 символов
                        showMessage(R.string.user_reg_error_new_account);
                        .setVisibility(View.INVISIBLE);
                        .setVisibility(View.VISIBLE);
                    }
```

Рисунок 5.5 Створення акаунту

помилку яка в нього склалась: не до кінця вів всі поля, або вів маленький пароль.

Firebase – mAuth – ми імпортуємо метод, для створення акаунту користувача для оновлення його даних.

На рис 5.6 в другому методі, дані які ми отримали в першому методі, оновлюємо та відправляємо системі Firebase. Користувача в цей час додаток переводить на головне вікно де вже розпочинає ознайомлюватись з інформацією в додатку.

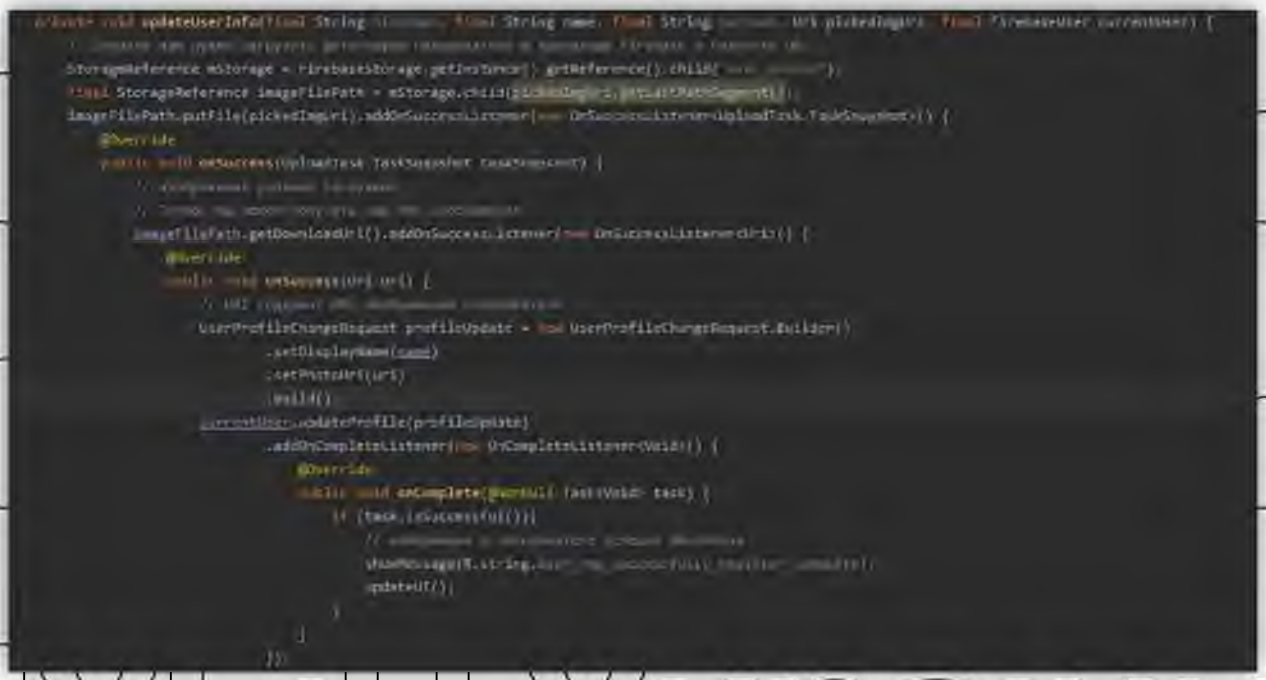

Рисунок 5.6 – метод оновлення та відправлення даних системі Firebase

4.2 Firebase - сервіс

В даному сервісі зберігаємо дані таки як : аутентифікації користувачів, сховище, та базу-даних. На рис 5.5 зберігаються всі користувачі які створили акаунт, також умовні позначення : ідентифікатор користувача, дата створення акаунту, метод через який користувач створив акаунт, та останній вхід у додаток.

В сервісі Firebase можливо слідкувати за коректністю вхідних даних, або видалення даних. Так як Firebase використовує хмарне сховище з ним простіше працювати.

Рисунок 5.7 – Firebase – користувачі які створили акаунт

4.3 Інтерфейс користувача

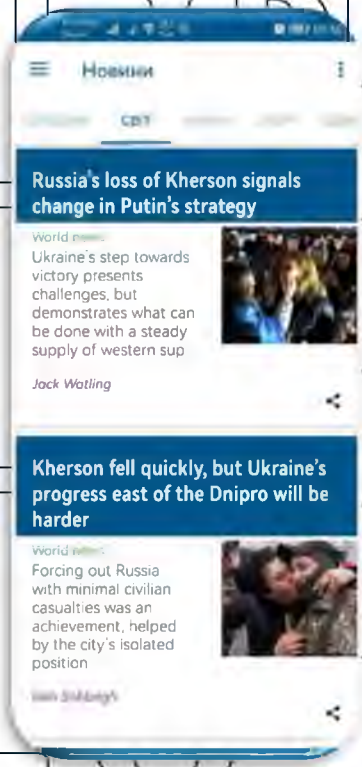

Рисунок 5.8 – Інтерфейс додатку з найактуальнішими світовими подіями

Коли користувач проходить аутентифікацію його зустрічає головна сторінка додатку, де користувач ознайомлюється з інформацією яку підносить додаток. На рис 5.8 користувач може побачити новини зі всього світу, передивитись доступні категорії.

На рис 5.8 зображено інтерфейс меню користувача, в якому користувач може мандрувати по категоріям, переходити до різних параметрів, як : налаштування новин, зміна користувацького інтерфейсу, сортування по даті, типу випуска та інші.

Стрічка новин формується за допомогою API сайтів новинних порталів, у форматі json, а потім інтегрується в стрічку. [12]

```
public News(String title, String section, String author, String url, String thumbnail, String trailText) {
    mTitle = title;
    mSection = section;
    mAuthor = author;
    mUrl = url;
    mThumbnail = thumbnail;
    mTrailTextHtml = trailText;
}
```

Рисунок 5.9 Конструктор класу новин

Після того як користувач ознайомиться з інтерфейсом, він в будь якому випадку перейде до перегляду новин. Завдяки налаштуванням користувач може вибрати модель того як дивитися новини: від найстаріших новин до нових або навпаки, або на основі актуальності, тобто те що користувачу буде найцікавіше в першу чергу.

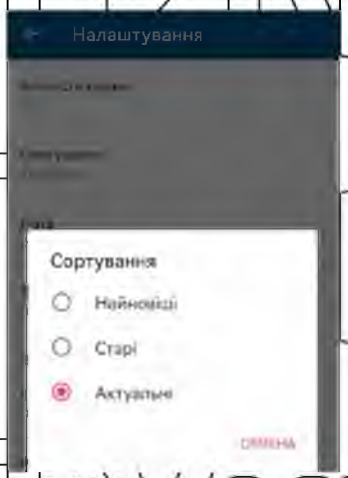

Рисунок 5.10 Налаштування новин

ВИСНОВОК

У результаті виконання дипломної роботи були досліджені алгоритми колаборативної фільтрації та створено рекомендаційну систему на їх основі.

Зроблено огляд основних типів рекомендаційних систем. Також проведений огляд використання даних алгоритмів великими компаніями.

Розроблено та покращено систему рекомендацій за допомогою колаборативної фільтрації. Розглянуто особливості розробки додатків мовою програмування Java.

Проаналізовано методи прогнозування рекомендацій, описано їх характеристики, переваги та недоліки, а також проведено порівняльний аналіз існуючих методів.

Розроблено програму, яка спираючись на досвід інших користувачів, пропонує найвірогідніші рекомендації. Також, практичною значимістю розробленої системи, є її відносна простота та можливість доповнення системи іншими модулями рекомендацій та реалізації додаткового функціоналу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Recommender system [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/Recommender_system (Дата звернення 20.10.2022)

2. Introduction to recommender systems [Електронний ресурс] / Режим доступу: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada (Дата звернення 20.10.2022)

3. Introduction to recommender systems [Електронний ресурс] / Режим доступу: https://thingsolver.com/introduction-to-recommender-systems/ (Дата звернення 20.10.2022)

4. Recommendation systems: Principles, methods and evaluation [Електронний ресурс] / Режим доступу: https://www.sciencedirect.com/science/article/pii/S1110866515000341(Дата звернення 20.10.2022)

5. A Collaborative Filtering Recommendation System in Java [Електронний ресурс] / Режим доступу: https://www.baeldung.com/java-collaborative-filtering-recommendations (Дата звернення 30.10.2022)

6. Recommendation System [Електронний ресурс] / Режим доступу: https://developers.google.com/machine-learning/recommendation/collaborative/basics (Дата звернення 30.10.2022)

7. Types of Recommendation Systems & Their Use Cases [Електронний ресурс] / Режим доступу: https://medium.com/mlearning-ai/what-are-the-types-of-recommendation-systems-3487cbafa7c9 (Дата звернення 30.10.2022)

8. P.Melville, R.J. Mooney, R.Nagarajan Content-Boosted Collaborative Filtering for Improved Recommendations [Електронний ресурс]/ Режим доступу: http://www.aaai.org/Papers/AAAI/2002/AAAI02-029.pdf (Дата звернення 30.10.2022)

9. Developers Android [Електронний ресурс] / Режим доступу: https://developer.android.com/docs (дата звернення 30.10.2022)
10. Firebase [Електронний ресурс]/Режим доступу: https://firebase.google.com/docs/auth/android/password-auth (дата звернення 10.11.2022)
11. Івченко О.В, Ільїн О.О Збірник матеріалів XIII Міжнародної науково-практичної конференції молодих вчених «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ: ЕКОНОМІКА, ТЕХНІКА, ОСВІТА ‘2022», 26–27 жовтня 2022 року, НУБіП України, Київ. – 77 с. (Електронне видання)]/Режим доступу: https://drive.google.com/file/d/1f2AWFKVqk3WOs2Eh0q6fSuA2qpqQeD97/view
12. Java API for JSON Processing: An Introduction to JSON [Електронний ресурс] / Режим доступу: https://www.oracle.com/technical-resources/articles/java/json.html (дата звернення 10.11.2022)

ДОДАТОК А

Лістинг програмного модулю

Клас News:

```
package com.example.android.newsfeed;

public class News {

    private String mTitle;
    private String mSection;

    private String mAuthor;

    private String mUrl;

    private String mThumbnail;

    private String mTrailTextHtml;

    public News(String title, String section, String author, String url, String thumbnail, String
trailText) {
        mTitle = title;
        mSection = section;
        mAuthor = author;
        mUrl = url;
        mThumbnail = thumbnail;
        mTrailTextHtml = trailText;
    }

    public String getTitle() {
        return mTitle;
    }

    public String getSection() {
        return mSection;
    }

    public String getAuthor() {
        return mAuthor;
    }

    public String getUrl() {
        return mUrl;
    }
```

```
    public String getThumbnail() {
        return mThumbnail;
    }

    public String getTrailTextHtml() {
        return mTrailTextHtml;
    }
}
```

Клас NewsLoader:

```
package com.example.android.newsfeed;

import androidx.loader.content.AsyncTaskLoader;
import android.content.Context;

import com.example.android.newsfeed.utils.QueryUtils;

import java.util.List;


public class NewsLoader extends AsyncTaskLoader<List<News>> {

    private static final String LOG_TAG = NewsLoader.class.getName();

    private String mUrl;

    public NewsLoader(Context context, String url) {
        super(context);
        mUrl = url;
    }

    @Override
    protected void onStartLoading() {
        // Trigger the loadInBackground() method to execute.
        forceLoad();
    }

    @Override
    public List<News> loadInBackground() {
        if (mUrl == null) {
            return null;
        }

        List<News> newsData = QueryUtils.fetchNewsData(mUrl);
        return newsData;
    }
```

}
Клас NewsAdapter:
```
package com.example.android.newsfeed.adapter;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.net.Uri;
import android.preference.PreferenceManager;
import androidx.cardview.widget.CardView;
import androidx.recyclerview.widget.RecyclerView;
import android.text.Html;
import android.text.format.DateUtils;
import android.util.Log;
import android.util.TypedValue;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import com.bumptech.glide.Glide;
import com.example.android.newsfeed.News;
import com.example.android.newsfeed.R;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.TimeZone;

public class NewsAdapter extends RecyclerView.Adapter<NewsAdapter.ViewHolder> {
    private Context mContext;
    private List<News> mNewsList;
    private SharedPreferences sharedPrefs;

    public NewsAdapter(Context context, List<News> newsList) {
        mContext = context;
        mNewsList = newsList;
    }

    @Override
    public NewsAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.news_card_item,
parent, false);
```

```
        return new ViewHolder(v);
    }

    @Override
    public int getItemCount() {
        return mNewsList.size();
    }

    class ViewHolder extends RecyclerView.ViewHolder {
        private TextView titleTextView;
        private TextView sectionTextView;
        private TextView authorTextView;
        private TextView dateTextView;
        private ImageView thumbnailImageView;
        private ImageView shareImageView;
        private TextView trailTextView;
        private CardView cardView;

        ViewHolder(View itemView) {
            super(itemView);
            titleTextView = itemView.findViewById(R.id.title_card);
            sectionTextView = itemView.findViewById(R.id.section_card);
            authorTextView = itemView.findViewById(R.id.author_card);
            dateTextView = itemView.findViewById(R.id.date_card);
            thumbnailImageView = itemView.findViewById(R.id.thumbnail_image_card);
            shareImageView = itemView.findViewById(R.id.share_image_card);
            trailTextView = itemView.findViewById(R.id.trail_text_card);
            cardView = itemView.findViewById(R.id.card_view);
        }
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        sharedPrefs = PreferenceManager.getDefaultSharedPreferences(mContext);

        setColorTheme(holder);

        setTextSize(holder);

        // Find the current news that was clicked on
        final News currentNews = mNewsList.get(position);

        holder.titleTextView.setText(currentNews.getTitle());
        holder.sectionTextView.setText(currentNews.getSection());

        if (currentNews.getAuthor() == null) {
            holder.authorTextView.setVisibility(View.GONE);
        } else {
```

```
            holder.authorTextView.setVisibility(View.VISIBLE);
            holder.authorTextView.setText(currentNews.getAuthor());
        }

        String trailTextHTML = currentNews.getTrailTextHtml();
        holder.trailTextView.setText(Html.fromHtml(Html.fromHtml(trailTextHTML).toString()));


        holder.cardView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Uri newsUri = Uri.parse(currentNews.getUrl());

                // Create a new intent to view the news URI
                Intent websiteIntent = new Intent(Intent.ACTION_VIEW, newsUri);

                mContext.startActivity(websiteIntent);
            }
        });

        if (currentNews.getThumbnail() == null) {
            holder.thumbnailImageView.setVisibility(View.GONE);
        } else {
            holder.thumbnailImageView.setVisibility(View.VISIBLE);
            // Load thumbnail with glide
            Glide.with(mContext.getApplicationContext())
                    .load(currentNews.getThumbnail())
                    .into(holder.thumbnailImageView);
        }

        holder.shareImageView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                shareData(currentNews);
            }
        });
    }

    /**
     * Set the user preferred color theme
     */
    private void setColorTheme(ViewHolder holder) {

        String colorTheme = sharedPrefs.getString(
                mContext.getString(R.string.settings_color_key),
                mContext.getString(R.string.settings_color_default));
```

```
        if (colorTheme.equals(mContext.getString(R.string.settings_color_white_value))) {
            holder.titleTextView.setBackgroundResource(R.color.white);
            holder.titleTextView.setTextColor(Color.BLACK);
        }else if
    (colorTheme.equals(mContext.getString(R.string.settings_color_sky_blue_value))) {
            holder.titleTextView.setBackgroundResource(R.color.nav_bar_start);
            holder.titleTextView.setTextColor(Color.WHITE);
        } else if
    (colorTheme.equals(mContext.getString(R.string.settings_color_dark_blue_value))) {
            holder.titleTextView.setBackgroundResource(R.color.color_app_bar_text);
            holder.titleTextView.setTextColor(Color.WHITE);
        } else if (colorTheme.equals(mContext.getString(R.string.settings_color_violet_value)))
    {
            holder.titleTextView.setBackgroundResource(R.color.violet);
            holder.titleTextView.setTextColor(Color.WHITE);
        } else if
    (colorTheme.equals(mContext.getString(R.string.settings_color_light_green_value))) {
            holder.titleTextView.setBackgroundResource(R.color.light_green);
            holder.titleTextView.setTextColor(Color.WHITE);
        } else if (colorTheme.equals(mContext.getString(R.string.settings_color_green_value)))
    {
            holder.titleTextView.setBackgroundResource(R.color.color_section);
            holder.titleTextView.setTextColor(Color.WHITE);
        }
    }

    private void setTextSize(ViewHolder holder) {

        String textSize = sharedPrefs.getString(
                mContext.getString(R.string.settings_text_size_key),
                mContext.getString(R.string.settings_text_size_default));

        if(textSize.equals(mContext.getString(R.string.settings_text_size_medium_value))) {
            holder.titleTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp22));
            holder.sectionTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp14));
            holder.trailTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp16));
            holder.authorTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp14));
            holder.dateTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp14));
        } else if(textSize.equals(mContext.getString(R.string.settings_text_size_small_value))) {
            holder.titleTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp20));
```

```
            holder.sectionTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp12));
            holder.trailTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp14));
            holder.authorTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp12));
            holder.dateTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp12));
        } else if(textSize.equals(mContext.getString(R.string.settings_text_size_large_value))) {
            holder.titleTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp24));
            holder.sectionTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp16));
            holder.trailTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp18));
            holder.authorTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp16));
            holder.dateTextView.setTextSize(TypedValue.COMPLEX_UNIT_PX,
                    mContext.getResources().getDimension(R.dimen.sp16));
        }
    }

    private void shareData(News news) {
        Intent sharingIntent = new Intent(Intent.ACTION_SEND);
        sharingIntent.setType("text/plain");
        sharingIntent.putExtra(android.content.Intent.EXTRA_TEXT,
                news.getTitle() + " : " + news.getUrl());
        mContext.startActivity(Intent.createChooser(sharingIntent,
                mContext.getString(R.string.share_article)));
    }

    public void clearAll() {
        mNewsList.clear();
        notifyDataSetChanged();
    }

    public void addAll(List<News> newsList) {
        mNewsList.clear();
        mNewsList.addAll(newsList);
        notifyDataSetChanged();
    }

    private String formatDate(String dateStringUTC) {
        // Parse the dateString into a Date object
```

```
        SimpleDateFormat simpleDateFormat =
                new SimpleDateFormat("yyyy-MM-dd'T'kk:mm:ss'Z'");
        Date dateObject = null;
        try {
            dateObject = simpleDateFormat.parse(dateStringUTC);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        // Initialize a SimpleDateFormat instance and configure it to provide a more readable
        // representation according to the given format, but still in UTC
        SimpleDateFormat df = new SimpleDateFormat("MMM d, yyyy  h:mm a",
Locale.ENGLISH);
        String formattedDateUTC = df.format(dateObject);
        // Convert UTC into Local time
        df.setTimeZone(TimeZone.getTimeZone("UTC"));
        Date date = null;
        try {
            date = df.parse(formattedDateUTC);
            df.setTimeZone(TimeZone.getDefault());
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return df.format(date);
    }

    private static long getDateInMillis(String formattedDate) {
        SimpleDateFormat simpleDateFormat =
                new SimpleDateFormat("MMM d, yyyy  h:mm a");
        long dateInMillis;
        Date dateObject;
        try {
            dateObject = simpleDateFormat.parse(formattedDate);
            dateInMillis = dateObject.getTime();
            return dateInMillis;
        } catch (ParseException e) {
            Log.e("Problem parsing date", e.getMessage());
            e.printStackTrace();
        }
        return 0;
    }

    private CharSequence getTimeDifference(String formattedDate) {
        long currentTime = System.currentTimeMillis();
        long publicationTime = getDateInMillis(formattedDate);
        return DateUtils.getRelativeTimeSpanString(publicationTime, currentTime,
                DateUtils.SECOND_IN_MILLIS);
    }
```

}

Клас CategoryFragmentPagerAdapter:

```
package com.example.android.newsfeed.adapter;

import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import android.content.Context;
import androidx.fragment.app.FragmentPagerAdapter;

import com.example.android.newsfeed.R;
import com.example.android.newsfeed.fragment.BusinessFragment;
import com.example.android.newsfeed.fragment.CultureFragment;
import com.example.android.newsfeed.fragment.EnvironmentFragment;
import com.example.android.newsfeed.fragment.FashionFragment;
import com.example.android.newsfeed.fragment.HomeFragment;
import com.example.android.newsfeed.fragment.ScienceFragment;
import com.example.android.newsfeed.fragment.SocietyFragment;
import com.example.android.newsfeed.fragment.SportFragment;
import com.example.android.newsfeed.fragment.WorldFragment;
import com.example.android.newsfeed.utils.Constants;


public class CategoryFragmentPagerAdapter extends FragmentPagerAdapter {

    private Context mContext;

    public CategoryFragmentPagerAdapter(Context context, FragmentManager fm) {
        super(fm);
        mContext = context;
    }

    @Override
    public Fragment getItem(int position) {
        switch (position) {
            case Constants.HOME:
                return new HomeFragment();
            case Constants.WORLD:
                return new WorldFragment();
            case Constants.SCIENCE:
                return new ScienceFragment();
            case Constants.SPORT:
                return new SportFragment();
            case Constants.ENVIRONMENT:
                return new EnvironmentFragment();
            case Constants.SOCIETY:
```

```
            return new SocietyFragment();
        case Constants.FASHION:
            return new FashionFragment();
        case Constants.BUSINESS:
            return new BusinessFragment();
        case Constants.CULTURE:
            return new CultureFragment();
        default:
            return null;
    }
}

@Override
public int getCount() {
    return 9;
}

@Override
public CharSequence getPageTitle(int position) {
    int titleResId;
    switch (position) {
        case Constants.HOME:
            titleResId = R.string.ic_title_home;
            break;
        case Constants.WORLD:
            titleResId = R.string.ic_title_world;
            break;
        case Constants.SCIENCE:
            titleResId = R.string.ic_title_science;
            break;
        case Constants.SPORT:
            titleResId = R.string.ic_title_sport;
            break;
        case Constants.ENVIRONMENT:
            titleResId = R.string.ic_title_environment;
            break;
        case Constants.SOCIETY:
            titleResId = R.string.ic_title_society;
            break;
        case Constants.FASHION:
            titleResId = R.string.ic_title_fashion;
            break;
        case Constants.BUSINESS:
            titleResId = R.string.ic_title_business;
            break;
        default:
```

```
                titleResId = R.string.ic_title_culture;
                break;
            }
        return mContext.getString(titleResId);
    }
}
```