

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ДЛЯ ПРОГРАМ ЛОЯЛЬНОСТІ
RESEARCH AND SOFTWARE IMPLEMENTATION OF THE
RECOMMENDATION SYSTEM FOR LOYALTY PROGRAMS

Виконав: студент(ка) 2 курсу, групи КНТ-121м

Спеціальності 121 Інженерія програмного

забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Марічев Д.О.

(прізвище та ініціали)

Керівник Колпакова Т.О.

(прізвище та ініціали)

Рецензент Коцур М.І.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Марічева Дениса Олександровича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація рекомендаційної системи для програм лояльності. Research and Software Implementation of the Recommendation System for Loyalty Programs

керівник проєкту (роботи) Колпакова Тетяна Олексіївна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “04” листопада 2022 року №363

2. Строк подання студентом проєкту (роботи) 1 грудня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Розробка архітектури програми. 4. Основні рішення щодо реалізації рекомендаційної системи. 5. Керівництво системного програміста. 6. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Колпакова Т.О., доцент		
Нормоконтролер	Дейнега Л.Ю., асистент		

7. Дата видачі завдання “1” вересня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4–5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7–8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10–13 тижні	Додатки
8	Нормоконтроль та рецензування.	14–15 тижні	
9	Захист роботи.	16 тиждень	

Студент(ка)

(підпис)

Марічев Д.О.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Колпакова Т.О.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
109 с., 46 рис., 13 табл., 2 дод., 21 джерело.

РЕКОМЕНДАЦІЙНА СИСТЕМА, РЕКОМЕНДАЦІЇ,
ПРОГНОЗУВАННЯ, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, ОБРОБКА
ВЕЛИКИХ ДАНИХ, СИСТЕМА ЛОЯЛЬНОСТІ.

Об'єкт дослідження – інформаційна технологія генерації
рекомендацій.

Предмет дослідження – методи та алгоритми формування
рекомендацій.

Мета роботи – розробка рекомендаційної системи нагород для
програм лояльності.

Матеріали, методи та технічні засоби: мова програмування Python,
бібліотеки Pandas та NumPy, різноманітні методи та підходи реалізації
рекомендаційних систем.

Результати. Реалізовано рекомендаційну систему, яка здатна
оброблювати великі обсяги даних та видавати релевантні рекомендації
користувачам.

Висновки. Досліджено та програмно реалізовано найпопулярніші
методи рекомендаційних методів та систем. Розроблено рекомендаційну
систему нагород для програм лояльності.

Галузь використання – програми лояльності, в яких передбачено
отримання нагород користувачами.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 109 pages, 46 figures, 13 tables, 2 appendixes, 21 sources.

RECOMMENDATION SYSTEMS, RECOMMENDATIONS, PROGNOSTICATION, COLLABORATIVE FILTERING, BIG DATA PROCESSING, LOYALTY SYSTEMS.

Object of study is an information technology for generating recommendations.

Subject of study is the methods and algorithms for forming recommendations.

The purpose of the work is to develop a reward recommendation system for loyalty programs.

Materials, methods, and tools: Python programming language, Pandas and NumPy libraries, different methods and approaches for the implementation of recommendation systems.

Results. A recommendation system has been implemented that is capable of processing large volumes of data and providing relevant recommendations for users.

Conclusions. The most popular methods of recommendation methods and systems have been researched and programmatically implemented. A reward recommendation system for loyalty programs has been developed.

The field of use is loyalty programs in which users can receive rewards.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань.....	10
дослідження	10
1.1 Визначення рекомендаційної системи.....	10
1.2 Програми лояльності	13
1.2.1 Бонусна програма.....	13
1.2.2 Фіксована знижка.....	15
1.2.3 Програма знижок.....	16
1.2.4 Товар в подарунок.....	17
1.2.5 Партнерська програма лояльності.....	18
1.3 Приклади існуючих програм лояльності.....	19
1.3.1 Програма лояльності Starbucks.....	19
1.3.2 Програма лояльності Espresso Point (inCust).....	22
1.4 Висновки та постановка задачі	24
2 Матеріали і методи.....	25
2.1 Підходи реалізації рекомендаційних систем.....	25
2.2 Кореляції та її методи	29
2.2.1 Метод кореляції Пірсона	31
2.2.2 Метод кореляції Спірмена	32
2.2.3 Метод кореляції Жаккара.....	33
2.2.4 Метод косинусної міри подібності.....	34
2.3 Оцінювання рекомендаційних систем	35
2.4 Типи даних	37
2.5 Формат збереження даних.....	39
2.6 Мова програмування та допоміжні бібліотеки	41
3 Проєктування архітектури системи	47
3.1 Онлайн частина	49

3.2 Офлайн частина	50
4 Основні рішення щодо реалізації рекомендаційної системи	52
4.1 Вхідні дані.....	52
4.2 Оптимізація зберігання даних.....	54
4.3 Розробка офлайн частини.....	57
4.3.1 Рекомендації на основі популярності нагород.....	59
4.3.2 Рекомендації на основі методу кореляції Пірсона	60
4.3.3 Рекомендації на основі методу кореляції Спірмена.....	61
4.3.4 Рекомендації на основі методу кореляції Жаккара	63
4.3.5 Рекомендації на основі методу косинусу подібності	65
4.3.6 Рекомендації на основі методу косинусу подібності та на основі популярності нагород	67
4.3.7 Порівняння результатів	69
4.4 Розробка онлайн частини	70
4.8 Висновки за розділом	71
5 Керівництво системного програміста	73
5.1 Призначення програми та системні вимоги	73
5.4 Підготовка до запуску системи	73
5.3 Вхідні дані.....	74
5.4 Запуск рекомендаційного модулю (офлайн частини)	74
5.5 Запуск інтерфейсу взаємодії (онлайн частини)	75
Висновки	76
Перелік джерел посилання	77
Додаток А Текст програми.....	79
Додаток Б Слайди презентації	99

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- AP – Average Precision;
- CSV – Comma-Separated Values;
- HTTP – Hyper Text Transport Protocol;
- ID – Identity Document;
- MAP – Mean Average Precision;
- P – Precision;
- PEP – Python Enhancement Proposals;
- БД – база даних;
- КФ – колаборативна фільтрація;
- РС – рекомендаційна система.

ВСТУП

На сьогоднішній день, коли на ринку існує безліч однотипних продуктів, послуг та різних сервісів приблизно однакової якості та ціни, наявність програми лояльності є дуже важливою перевагою у конкурентній боротьбі. Незалежно від обсягів продажу та сфер діяльності, впровадження програм лояльності є актуальним питанням для пришвидшення розвитку бізнесу.

Якщо бренд не думає та не піклується про свого споживача, у нього немає шансів завоювати повсюдну впізнаваність, побудувати свою спільноту відданих клієнтів та стати їх улюбленим брендом. А програма лояльності – один з важливих інструментів, які допомагають підвищити привабливість компанії для споживача, чи то великий інтернет-магазин, чи малий бізнес.

Програма лояльності допомагає зробити випадкових покупців постійними. А потім мотивувати їх на нові покупки, допомагає налагоджувати постійну комунікацію та розвивати емоційну прив'язаність до бренду. Фактично, головне завдання таких програм – побудувати довготривалі стосунки зі своїми клієнтами. Працювати з лояльними клієнтами простіше та ефективніше, ніж з новими споживачами, які ще не знають ані компанію, ані її продукт.

Найбільш успішні програми лояльності для своєї роботи використовують рекомендаційні системи, які підлаштовуються під користувачів і пропонують їм найбільш релевантні пропозиції. Різноманітні компанії та бренди витрачають багато часу і коштів для вдосконалення і поліпшення своїх рекомендаційних систем

Мало розробити та впровадити програму лояльності. Необхідно також правильно розробити і налаштувати рекомендаційну систему. Якщо допомагати своїм клієнтам досягати цілі, вони будуть асоціювати свої успіхи цим брендом, і тому повертатися за новою порцією «успіху» [1].

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Визначення рекомендаційної системи

Для початку необхідно дати визначення рекомендаційної системи.

Рекомендаційна система (РС) – це програмне забезпечення, яке, аналізуючи різноманітні дані користувачів, пропонує їм персональний продукт, який, ймовірно, покриває їх потреби. Основною ціллю РС є фільтрація певної підмножини інформації.

Останнім часом, значно зросла кількість інформації, з якою щодня стикаються люди. Наслідком цього є те, що людина повинна орієнтуватися у надзвичайно великій кількості різноманітних альтернативних пропозицій, коли вона хоче щось знайти. В той же час, власники інтернет-магазинів, бізнесів і сервісів зацікавлені у тому, щоб кожному конкретному користувачеві були персональні рекламні пропозиції та рекомендації. Персоніфіковані рекламні пропозиції та рекомендації можуть значно збільшити прибутки компанії.

Сьогодні різноманітні компанії почали конкурувати між собою не тільки якістю і ціною пропонованих ними продуктів чи послуг, але й в тому, наскільки релевантні рекомендаційні системи вони використовують.

Поява та зростання інтернет магазинів має значний вплив на поведінку покупців, надаючи їм доступ до значної різноманітності товарів та інформації. Через це покупцям ускладнився вибір товарів, які найбільше відповідають їхнім потребам. Одним із основних методів вирішення проблеми інформаційного перевантаження є рекомендаційні системи, які забезпечують автоматизоване та персоналізоване пропонування товарів для споживачів [2].

При розробці РС також варто враховувати область її застосування, оскільки від вибору області застосування будуть залежати рішення щодо конфігурації та задання певних правил для правильної роботи РС.

Монтанер в [3] класифікує існуючі додатки РС до певних доменів додатків. Найпоширеніші додатки рекомендаційних систем було узагальнені у класи доменів:

- розваги – рекомендації для фільмів, музики, ігор, тощо;
- контент – персоналізовані газети, новини, рекомендації документів, вебсторінок в пошукових запитах, рекомендації онлайн курсів, а також фільтрація по категоріям для електронної пошти;
- електронна комерція – рекомендації товарів для покупок, таких як книжки, телефони, ноутбуки, аксесуари, тощо;
- послуги – рекомендації туристичних послуг, рекомендації експертів для консультації, рекомендації будинків для оренди або послуги знайомств;
- соціальні мережі – рекомендація людей у соціальних мережах та рекомендації контенту в соціальних медіа.

Найчастіше рекомендаційні системи використовують в галузі інтернет-комерції, для того, щоб допомогти своїм покупцям обрати необхідні товари. Щоб генерувати рекомендації, такі компанії збирають інформацію про переваги своїх користувачів і за допомогою цих даних намагаються запропонувати корисні товари. В деяких випадках інтернет-магазини можуть маніпулювати результатами рекомендацій та спеціально додавати до них ті товари, які не користуються великим попитом.

На сьогоднішній день існує багато різних методів для генерування рекомендацій, при тому, що кожен з них має як свої переваги так і недоліки. Саме через це дослідження в цій області актуальні сьогодні і будуть актуальними ще довгий проміжок часу.

Зазвичай, поліпшення рекомендаційних систем полягає у поліпшенні алгоритмів рекомендації, які намагаються видавати покупцям найбільш точні рекомендації, які є релевантними у певний момент часу. Для того, щоб покращувати рекомендації, необхідно аналізувати і вдосконалювати математичні алгоритми, які лежать в основі рекомендаційних сервісів.

Системи рекомендацій є одним із потужних засобів, за допомогою яких бізнеси в галузі інтернет-комерції отримують постійних клієнтів.

Повторна покупка – це один з ключових показників для бізнесів, через те, що повторний клієнт означає меншу витрату грошей на те, щоб спонукати його до покупки.

Насправді, рекомендаційні системи не є новою технологією, бо вони з'явилися ще тоді, коли тільки почали розвиватися різноманітні інтернет-сервіси. Сфери застосування рекомендаційних систем дуже різноманітні: пошук товарів, фільмів, музики, статей, зображень та навіть нових знайомств.

Невід'ємною частиною рекомендаційної системи є так званий Data Mining – це термін, що застосовується для опису здобуття знань у базах даних, дослідження даних, обробки зразків даних, очищення і збору даних. Це процес виявлення кореляції, тенденцій, шаблонів, зав'язків і категорій. Сутність і мету технології Data Mining можна описати так: це технологія, призначена для пошуку у великих інформаційних масивах даних неочевидних, об'єктивних, корисних на практиці закономірностей. Здійснюється за допомогою використання технологій розпізнавання шаблонів, а також статистичних і математичних методів [4].

Схематично цей процес можна описати такими кроками (рис. 1.1):

- користувач «спілкується» з системою через інтерфейс взаємодії;
- система збирає та зберігає данні користувача, наприклад, інформацію про куплені чи переглянуті товари;
- при необхідності користувачу відобразити його персональні рекомендації, спеціальний рекомендаційний алгоритм генерує рекомендації на основі раніше збережених даних.

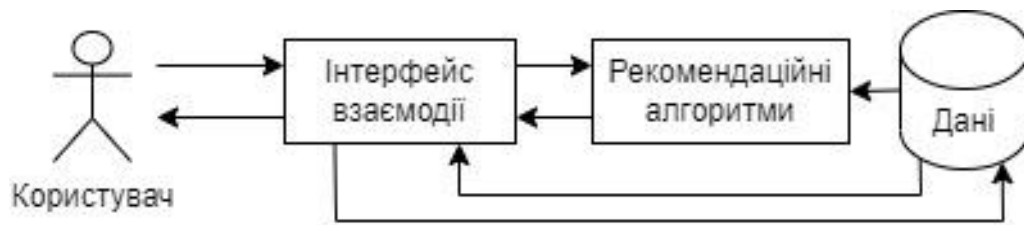


Рисунок 1.1 – Схема принципу роботи рекомендаційної системи

1.2 Програми лояльності

Залучення нових клієнтів – це, безсумнівно, важливий процес, без якого не проходить просування будь-якого магазину. Реклама в більшості випадків успішно працює, і бізнес отримує клієнта. Для того, щоб зберегти вже існуючого клієнта, використовують різноманітні програми лояльності.

Програма лояльності – це врегулювання взаємовідносин з клієнтом шляхом впровадження різного роду маркетингових заходів для збільшення повторних продажів. Простими словами, це метод збереження постійного покупця.

Багато власників магазинів забувають про впровадження програм лояльності, помилково думаючи, що це під силу тільки великим мережним бізнесам. Насправді програми лояльності – це універсальний метод для зростання продажів і залучення і втримання клієнтів.

Існують різні типи програм лояльності зі своїми плюсами і мінусами.

1.2.1 Бонусна програма

При використанні бонусної програми покупець отримує власну бонусну картку. Відповідно до шкали конвертації балів за кожну зроблену покупку клієнт буде отримувати певну кількість балів. Курс конвертації задають самі власники бізнесу. Наприклад, за кожні витрачені 10 грн покупцеві нараховується 1 бал.

Балами клієнт може надалі розплатитися. У цьому випадку використовують іншу шкалу конвертації. Наприклад, кожен накопичений 100 балів дорівнюють 1 грн. Таким чином, чим більше покупець витрачає, тим більше балів, а відповідно грошей він накопичує.

Також за свої накопичені бонусні бали покупці можуть отримувати різного роду нагороди, наприклад, за 100 бонусів можна отримати безкоштовну чашку кофе або якийсь інший фізичний товар чи продукт. Замість подарункового товару можна обрати, наприклад, 30% знижку на наступну купівлю в цьому ж магазині. Схема роботи бонусної програми з отриманням нагород за накопичені бонуси наведена на рисунку 1.2.

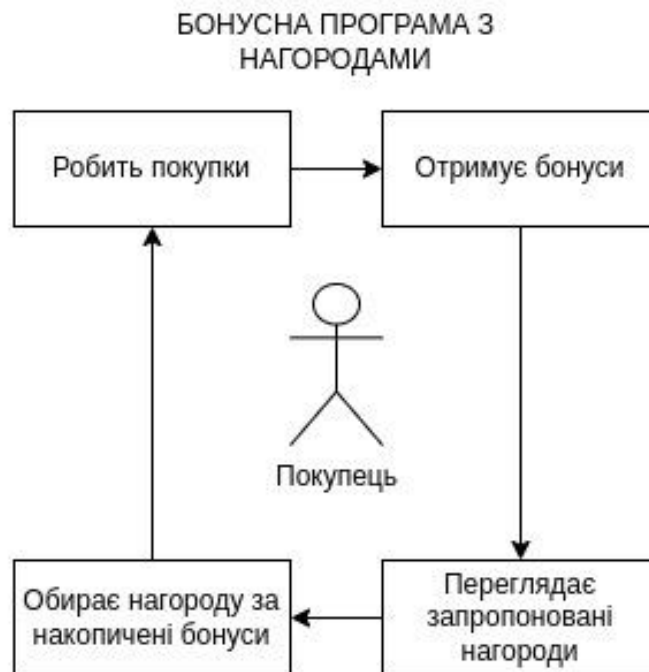


Рисунок 1.2 – Схема роботи бонусної програми з отриманням нагород за накопичені бонуси

Переваги бонусної програми:

- клієнти знайомі з такою системою нарахування балів;
- мотивація до здійснення покупки саме в вашому магазині;

- можна встановити «термін придатності» накопичених бонусів, щоб стимулювати клієнта до чергової покупки
- не всі клієнти користуються бонусами, навіть накопичивши їх, тому часто така програма лояльності працює тільки в плюс;
- фінансово найвигідніший варіант програми лояльності для бізнесу.

Недолік бонусної програми полягає в тому, що в ній немає миттєвого результату для клієнта, через тривале накопичення бонусів.

Бонусна програма є дуже поширеною в Україні. Найчастіше нею користуються магазини широкого попиту.

Наприклад, бонуси будуть вигідні супермаркетам, продуктовими магазинами, заправкам, магазинах побутової хімії, магазинам техніки, електроніки, комп'ютерів, мобільних телефонів.

Наприклад, для магазинів техніки, при першій покупці клієнтові нараховують бонуси, якими він зможе погасити частину суми наступної покупки. Таким чином, бізнес перетворює нового покупця у постійного, мотивуючи повернутися у магазин через нараховані бонуси.

1.2.2 Фіксована знижка

Фіксована знижка – це один з типів програм лояльності, при якому покупець купує товар на певну суму, яку власник бізнесу може вказати в залежності від цінової політики магазину. Наприклад, при покупці товару на 1000 грн клієнт отримує знижку в розмірі 3%. В такому випадку накопичувальної системи немає і розмір знижки не буде збільшуватися. Тобто немає значення придбав покупець на 1100 грн чи на 15000 грн, знижка буде фіксована.

Переваги фіксованої знижки:

- легко організувати, не потрібно витрачати багато сил і ресурсів на впровадження;
- зрозумілі умови для клієнтів;

- покупець завжди в курсі відсотка знижки;
- можливість отримати результат «тут і зараз»;
- знижка може бути як одноразовою, так і постійною.

Недоліки фіксованої знижки:

- через велику кількість конкурентів з такими ж умовами знижки, клієнт може вибрати інший магазин;
- немає постійного інтересу і мотивації до отримання вищого відсотка знижки або бонусів, як було описано раніше;
- немає можливостей для розширення програми лояльності через незмінність відсотка знижки;
- при відсутності клієнтоорієнтованості, така знижка не врятує.

Фіксована знижка є універсальною, тому нею можуть користуватися будь-які магазини. Проте вона буде не дуже вигідна і без того часто відвідуваним місцям, як то магазини продуктів або супермаркети.

Система вважається застарілою і малоефективною, але все ще часто зустрічається в магазинах одягу, взуття, магазинах квітів. Тобто, найчастіше це місця, де покупки відбуваються не щодня, а раз на кілька місяців.

1.2.3 Програма знижок

Програма знижок, також відома як багаторівнева, – складна програма лояльності, яка працює на збереження постійних покупців шляхом постійної мотивації і підвищення їх статусу. Багатоярусну програму лояльності можна впроваджувати по-різному, обігруючи і придумуючи рівні, ступені, звання клієнтам, але суть залишається все та ж.

Головне правило – чим більше купують, тим більше отримують. Як це працює? Бізнес встановлює спеціальну шкалу знижок, яка буде актуальна для магазину. Наприклад, щоб отримати знижку в 5% потрібно придбати товару на 5000 грн, 10% – на 10000 грн. Таким чином, щоб отримати вище знижку, потрібно більше витратити.

Це лише один із прикладів роботи багаторівневої програми лояльності. Замість знижки можуть бути різні подарунки, бонуси або додаткові плюси у вигляді можливості першому отримувати інформацію про нові колекції, знижки тощо.

Переваги програм знижок:

- мотивування клієнта до чергової покупки;
- «добирання» товару до потрібної суми (особливо добре працює, якщо про це нагадує продавець);
- можливість отримання знижок, що дуже подобається покупцям;
- вигідна для власника, тому що знижку потрібно «заробити».

Недоліки програм знижок:

- проблема в складному і тривалому впровадженні;
- дорога програма лояльності;
- відсутність миттєвого результату для покупця.

Як вже було сказано, програма знижок більше підходить магазинам з дорогим товаром, авіакомпаніям, готелям. Найчастіше її використовують відомі бренди, чиє ім'я на слуху.

Для бізнесів-початківців це не найкращий варіант, але її можуть спробувати магазини косметики, парфумерії та одягу.

1.2.4 Товар в подарунок

Товар в подарунок – це ще один із варіантів реалізації системи лояльності, в якому при купівлі товару в потрібній кількості клієнт отримує подарунок. Це може бути знижка в 50% на третій товар або безпосередньо третій товар безкоштовно. Так працюють акції по типу «1 + 1 = 3».

Ще один варіант – коли при покупці на певну суму покупець отримує подарунок на вибір магазину.

Переваги товару в подарунок:

- покупець отримує ефект на місці;

- зручна і проста програма лояльності і для розуміння клієнта, і для реалізації в магазині;

- карта лояльності не є обов'язковою при таких акціях;

- присутні мотивація до придбання більшої кількості одиниць товару.

Недоліки товару в подарунок:

- потреба в завищенні ціни, щоб не отримати зниження чистого прибутку;

- немає взаємодій з клієнтом і цілеспрямованих пропозицій.

Акцію по типу «1 + 1 = 3» часто можна зустріти в продуктових магазинах або магазинах побутової хімії, косметики. Люблять використовувати такі ходи заправки, але при цьому ціни там вже попередньо вище звичайних.

1.2.5 Партнерська програма лояльності

Створення партнерських груп з кількох магазинів можна вважати свого роду взаємною рекламою. Кілька магазинів або мереж між собою домовляються і укладають угоду на спільне співробітництво. Для клієнта створюється одна карта, яка буде діяти в усіх партнерських магазинах. Розмір накопичення бонусів або знижки може бути однаковим скрізь або ж залежати від певної торгової точки.

В іншому програма лояльності буде працювати так, як описано в варіантах вище. Все залежить від обраного типу.

Переваги партнерських програм лояльності:

- вигідна для всіх партнерів реклама та збільшення клієнтської бази;

- привабливість пропозиції для клієнта, адже він одним махом отримує карту в кілька місць;

- найчастіше безкоштовна участь в програмі лояльності, без попередніх покупок на велику суму;

- можливість надання індивідуальних пропозицій для покупця;

- розширення географії покупців не тільки в одному місті, а й по всій країні.

Недоліки партнерських програм лояльності:

- дороге впровадження і запуск таких програм лояльності;
- складність в пошуку партнерів з відповідною клієнтською базою.

Партнерська програма лояльності зазвичай укладається між великими компаніями або мережами магазинів. На початковому етапі краще організовувати таку систему в межах одного міста.

Такий варіант буде добре працювати з магазинами одягу, продуктовими магазинами, супермаркетами і гіпермаркетами [5].

1.3 Приклади існуючих програм лояльності

Де би ми не були, нас завжди будуть оточувати компанії зі своїми програмами лояльності. Кожна з цих програм лояльності не перестає розвивати і адаптуватися під нові реалії бізнесу. Велику роль в цих програмах лояльності займають рекомендаційні системи.

1.3.1 Програма лояльності Starbucks

Starbucks – навряд чи знайдеш людину, яка не чула такої назви. Це успішний бізнес (понад 22 тис. кав'ярень у 67 країнах) із чималим доходом і незліченною кількістю клієнтів.

У 2007 році мережа переживала спад продажів, масове звільнення персоналу й закриття великої кількості своїх кав'ярень. У боротьбі з кризою рішенням стала повна зміна меню, створення нових рецептів кави й унікальної в своєму роді програми лояльності.

До кризи Starbucks уже впроваджував карткову програму лояльності за принципом: «Ви купуєте у нас картку «Gold» вартістю 25\$, і вона дає вам постійну знижку в 10% в чеку». Як показала практика, клієнти не були готові

витрачати гроші на таку дисконтну карту й більшість відмовлялася від її покупки.

Із 2008 року, My Starbucks Reward – нова програма лояльності Starbucks. Це безкоштовна карта, яка:

- може бути подарунковим сертифікатом для друзів клієнта;
- є бонусною карткою для накопичення зірок;
- виконує роль карти, якою можна розрахуватися в мережі Starbucks, попередньо поклавши на неї гроші.

Правила участі в My Starbucks Rewards: три рівні – Базовий (Welcome), Зелений (Green) і Золотий (Gold).

Клієнти Welcome-рівня (не залежить від кількості накопичених бонусів) отримували подарунок на День народження – безкоштовний напій на вибір.

Green рівень – 5 накопичених зірок (1 замовлення = 1 зірка) – безкоштовний напій і доступ до платного Wi-Fi. Виходило, що після п'яти відвідувань кав'ярні, клієнту дарували одну каву безкоштовно.

Gold-рівень (30 накопичених зірок) – видача клієнту іменної карти, звання VIP-клієнта й безкоштовної кави після кожних 12 зірок.

З більш детальними та актуальними умовами системи лояльності мережі Starbucks можна ознайомитися на спеціальній сторінці Starbucks Rewards (рис 1.3).

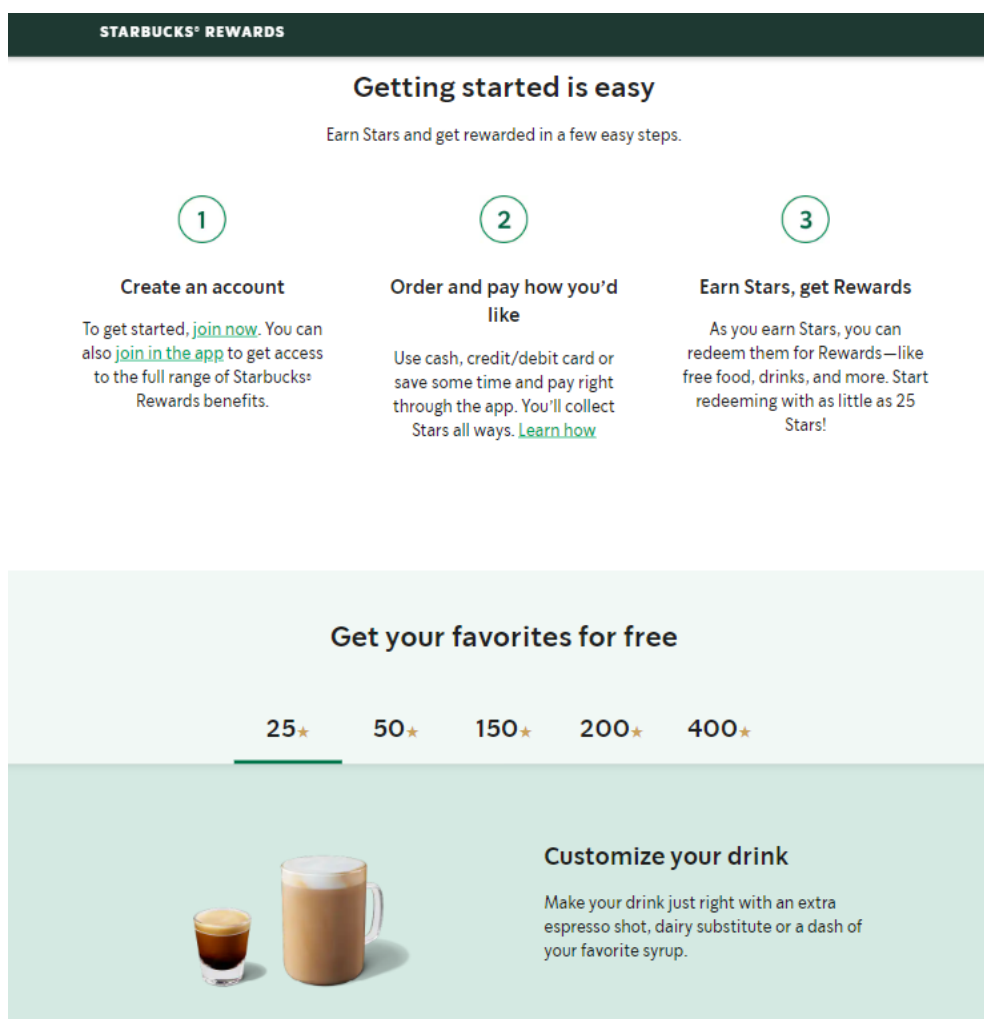


Рисунок 1.3 – Скріншот системи лояльності з сайту Starbucks

Програма має колосальний успіх і налічує більш ніж 12 мільйонів активних учасників. А тепер про прибуток, який фірма отримує за кожного такого активного клієнта: по суті, вони не дають йому нічого – лише одну 13-ту чашку кави безкоштовно. І тут можна згадати про ціни на каву в цьому закладі й припустити суму середнього чека. Якщо помножити на 12 відвідувань і вийде підвищення продажів, стрімке збільшення відсотка постійних клієнтів, успіх і популярність.

У 2016 році Starbucks знову змінює правила гри: тепер є тільки два рівні (Green і Gold), і якщо раніше нарахування зірок не мало стосунку до суми чека, а головним був факт замовлення, то тепер до уваги беруть і те, скільки витратив клієнт: раніше він отримував 1 зірку за 1 відвідування, а тепер її видають мінімум за \$2. Умови переходу на рівень Gold теж

змінилися – тепер це не 30 зірок, а 300, і безкоштовну чашку кави отримуєш не після 12 зірок, а після 125. Ще одна новинка – є один день на місяць, коли клієнт отримує подвійну кількість зірок.

Таким чином, Starbucks вдалося врятувати справу, яка мало не терпіла фіаско, зупинити чисельне закриття закладів і повернутися до лав найуспішніших бізнесів у світі з прибутком в \$626,7 млн за 2015 рік [6].

1.3.2 Програма лояльності Espresso Point (inCust)

Відкриття малого бізнесу – ризик: ризик втратити вкладення, зазнати невдачу або потонути в морі конкуренції. З конкуренцією довелося боротися і власнику Espresso Point, коли він відкривав мережу кавових машин у 2010 році.

Власник Espresso Point, придумував схеми з повернення клієнтів: спеціальні тимчасові акції у вигляді знижок на деякі позиції в меню або ж сьома чашка в подарунок із випуском спеціальних карт, на яких клієнтам позначали кожну куплену чашку кави. Результати були незадовільні – великі відсотки клієнтів так і не поверталися, інші – втрачали або забували свої карти. Тому витрати на залучення не поверталися прибутком.

Після цього вирішили, що потрібно під'єднати програму лояльності, яка буде стабільна й принесе результати. Серед безлічі пропозицій із випуском карт і мобільними додатками, власник обрав підключення безкарткової програми лояльності inCust (рис. 1.4). В результаті це дозволило зібрати базу клієнтів, скоротити витрати й підвищити продажі – прийнятне рішення для малого бізнесу.

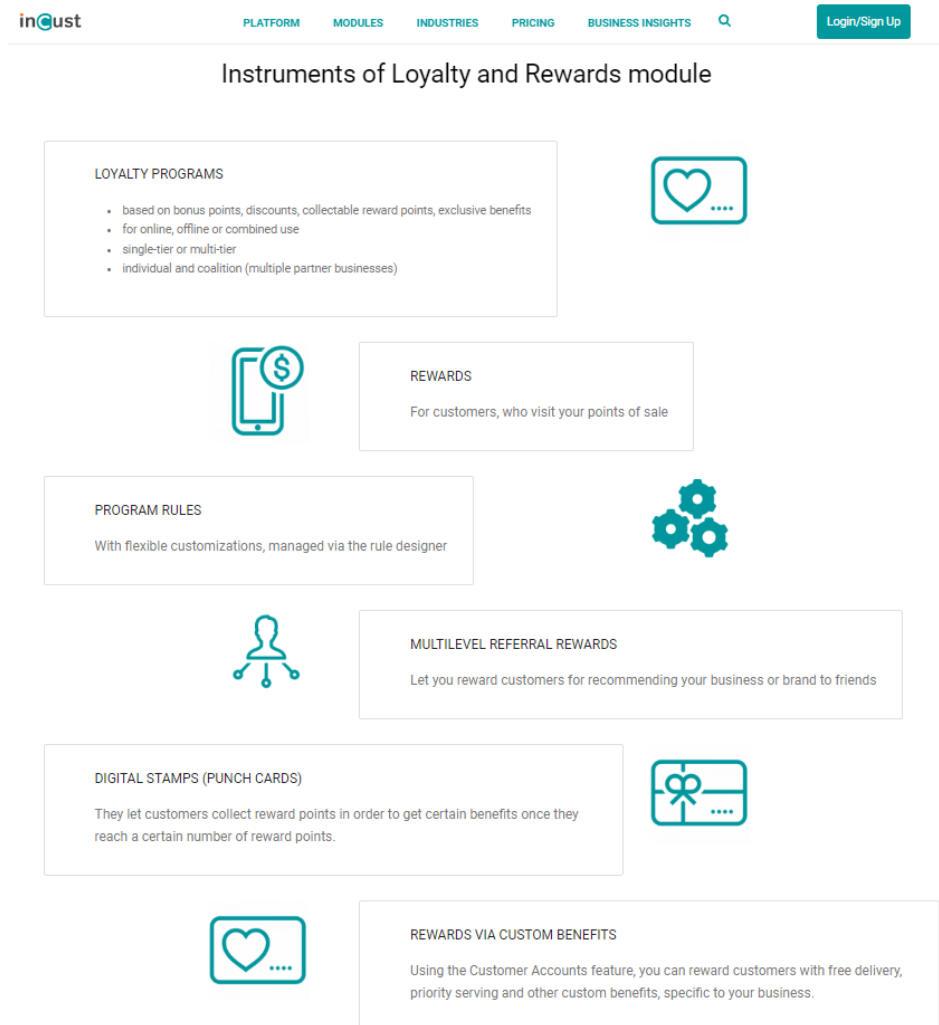


Рисунок 1.4 – Скріншот системи лояльності з сайту inCust

Перший функціонал, який застосували у випадку Espresso Point, – нарахування бонусів від суми чека. Таким чином, клієнт отримував мотивацію, яка змушувала повернутися назад і купити ще чашечку.

Далі звіти дозволили визначити дні з найменшими продажами й запуснути тимчасову акцію по 100%-му нарахуванню бонусів.

Крім цього, клієнти отримують сповіщення про кількість накопичених бонусів і акції, а також безкоштовний додаток, де вони можуть відстежувати свої покупки й останні новини мережі.

Наступним кроком було додання можливості клієнтам обирати щотижневу нагороду за накопичені бонусні бали. Для кожного окремого клієнта формувалися персональний перелік запропонованих нагород.

За більш ніж 6 місяців роботи продажі зросли завдяки тому, що відсоток постійних клієнтів збільшився з 28% до 60% [6].

1.4 Висновки та постановка задачі

Підбиваючи висновки для поточного розділу можна виділити наступне: у сучасному світі бізнеси повинні адаптуватися під потреби та вимоги своїх клієнтів; намагатися усілякими способами втримати постійних покупців, так як саме вони приносять найбільше доходу бізнесу; впроваджувати різноманітні програми лояльності для знаходження «спільної мови» зі своїми клієнтами.

Також було виявлено, що впровадження та покращення вже існуючих рекомендаційних систем великою мірою впливає на зацікавленість і утримання постійних користувачів (покупців).

Аналізуючи пройдений шлях, можна сказати, що правильний підхід до клієнтів дозволяє не тільки вижити бізнесу, але й домогтися стабільності та значного збільшення доходів.

Задача даної роботи полягає в тому щоб дослідити найпопулярніші методи і підходи створення рекомендаційних систем. Розробити прототип рекомендаційної системи, яка буде рекомендувати клієнтами програмам лояльності найбільш релевантні для них нагороди.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Підходи реалізації рекомендаційних систем

Бувають два типи рекомендацій: персоналізовані та неперсоналізовані.

Неперсоналізованими рекомендаціями вважається ті рекомендації, які генеруються без урахування попередньої активності користувача. Найчастішим прикладом неперсоналізованих рекомендацій вважається показ найпопулярніших продуктів, так званих бестселерів (рис. 2.1) або нових товарів (рис. 2.2).

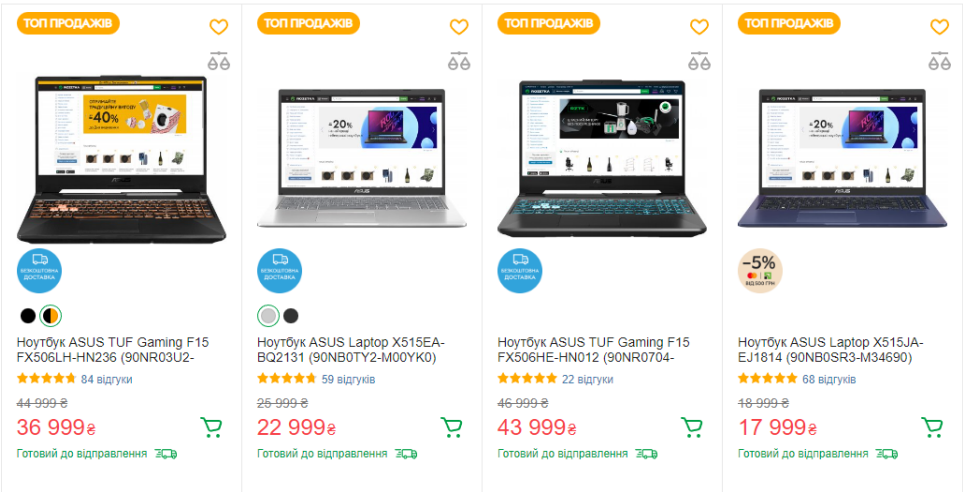


Рисунок 2.1 – Приклад неперсоналізованих рекомендацій (за популярністю)

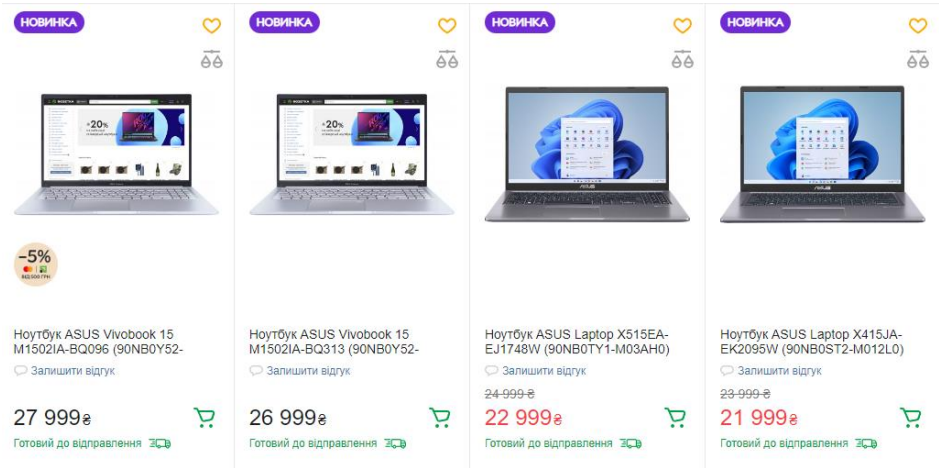


Рисунок 2.2 – Приклад неперсоналізованих рекомендацій (за новизною)

Персоналізовані рекомендації базуються на даних про поточного користувача, для якого генеруються рекомендації. Серед підходів персоналізованої рекомендації розрізняють:

- базовані на контенті;
- колаборативну фільтрацію (КФ);
- гібридні.

Принцип рекомендацій, базованих на контенті, – це порівнювати між собою властивості товарів, і тим самим знаходити спільні між ними риси. Зазвичай, спочатку запам'ятовують характеристики тих товарів, з якими вже взаємодіяв користувач, а після цього вже шукають товари, які схожі за характеристиками на ті товари, з якими вже взаємодіяв користувач до цього.

У принципі генерації рекомендацій, базованих на контенті, є дуже значні мінуси. По-перше, дуже важко написати алгоритм, який буде аналізувати контент, бо кожний випадок буде майже унікальним, також важко аналізувати контент, який, наприклад, являється зображеннями, музикою чи відео. По-друге, проблемою є те, що контент товару зазвичай є недостатнім для аналізу. По-третє, системи, які базовані на контенті будуть створювати «ефект бульбашки», іншими словами, будуть завжди рекомендувати користувачу тільки продукти, схожі на ті, з якими він вже взаємодіяв. Наприклад, в додатку для рекомендацій фільмів, система може рекомендувати користувачу фільми такого ж жанру, або ті які мають однакових акторів з тими фільмами, які користувач вже подивився. Однак система не спроможна порекомендувати фільми, які відрізняються від попередніх, і які також були б цікаві користувачу.

На відміну від рекомендацій, заснованих на контенті, методи колаборативної фільтрації використовують дані про взаємодію з товарами від інших користувачів системи. Головною ідеєю є те, що оцінка певного користувача для товару буде схожою на ту, яку поставив схожий користувач. Схожість користувачів вираховується різними методами.

Колаборативна фільтрація має декілька переваг на рекомендаціях основаних на контенті. Наприклад, товари, у яких контент відсутній або є важкодоступним, можуть бути рекомендовані, базуючись на оцінках інших користувачів системи. Також, колаборативна фільтрація базується на якості товарів, оцінених іншими користувачами, а не на контенті, який може бути поганим показником якості товару. Ще однією перевагою колаборативної фільтрації є те, що вона може рекомендувати різні товари з різним контентом, через те, що інші користувачі системи виявили інтерес до цих товарів.

Гібридними методами називають методи генерації рекомендацій, які поєднують у собі підходи двох попередніх методів. Прикладом гібридних рекомендаційних систем є генерація рекомендацій у мобільному застосунку Tiktok (рис. 2.3).

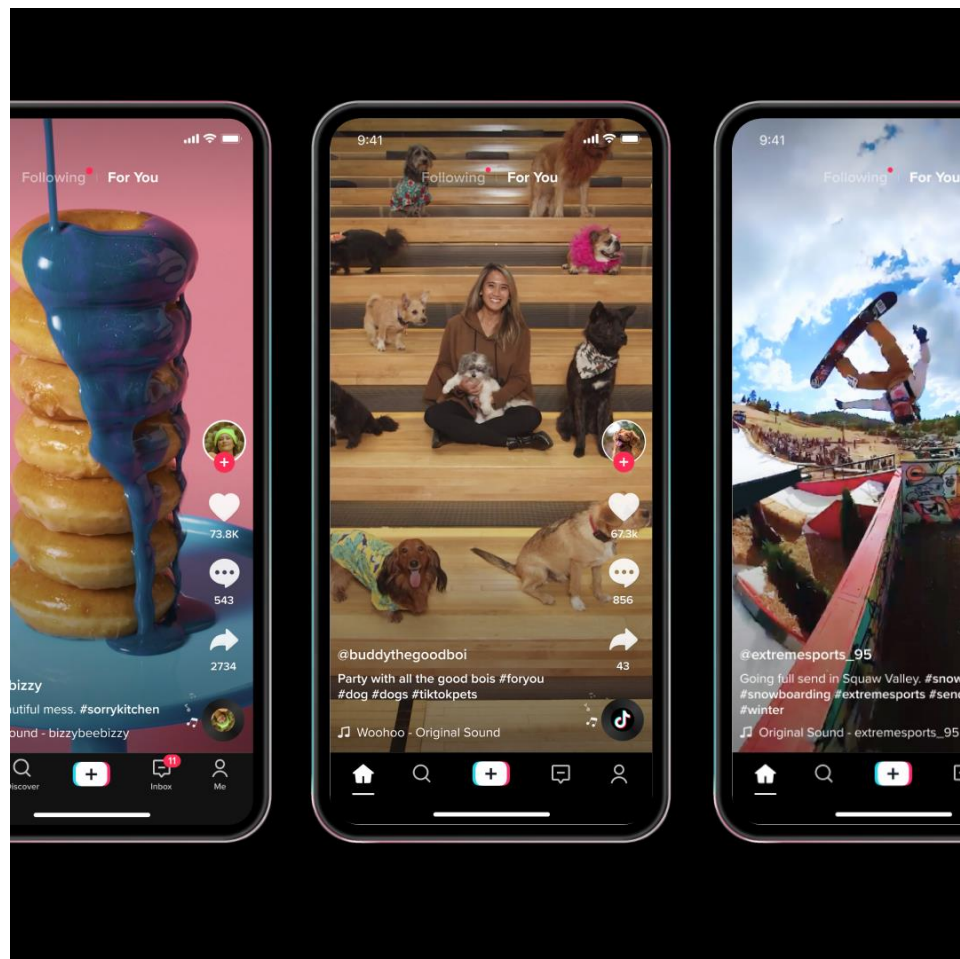


Рисунок 2.3 – Приклад гібридної рекомендаційної системи (Tiktok)

Методи колаборативної фільтрації, які працюють на основі сусідства (близькості) поділяються на:

- аналіз подібності користувачів (user-based);
- аналіз подібності продуктів (item-based).

User-based методи використовують схожість користувачів між собою, для цього ці алгоритми знаходять інших користувачів, чії попередні оцінки схожі на ті, що й у обраного користувача, і використовують їх оцінки інших продуктів для прогнозування оцінки обраного користувача (рис. 2.4).

Item-based методи використовують схожість між продуктами замість схожості користувачів. Якщо два продукти мають дві однакові оцінки користувачів, то вони схожі, і користувачі повинні мати аналогічні рекомендації для подібних продуктів (рис. 2.5).

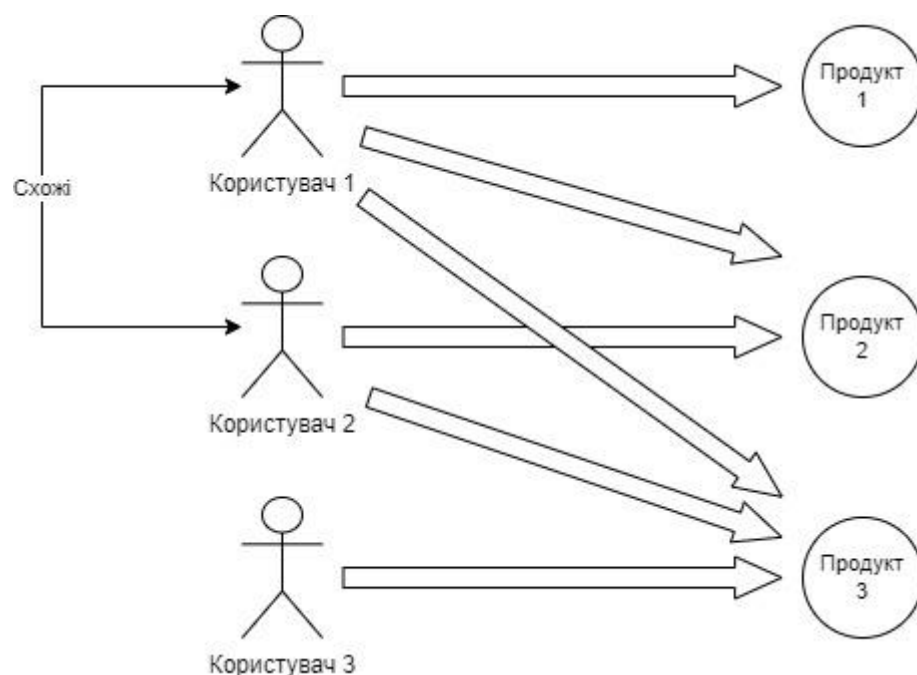


Рисунок 2.4 – Схема роботи user-based рекомендаційних систем

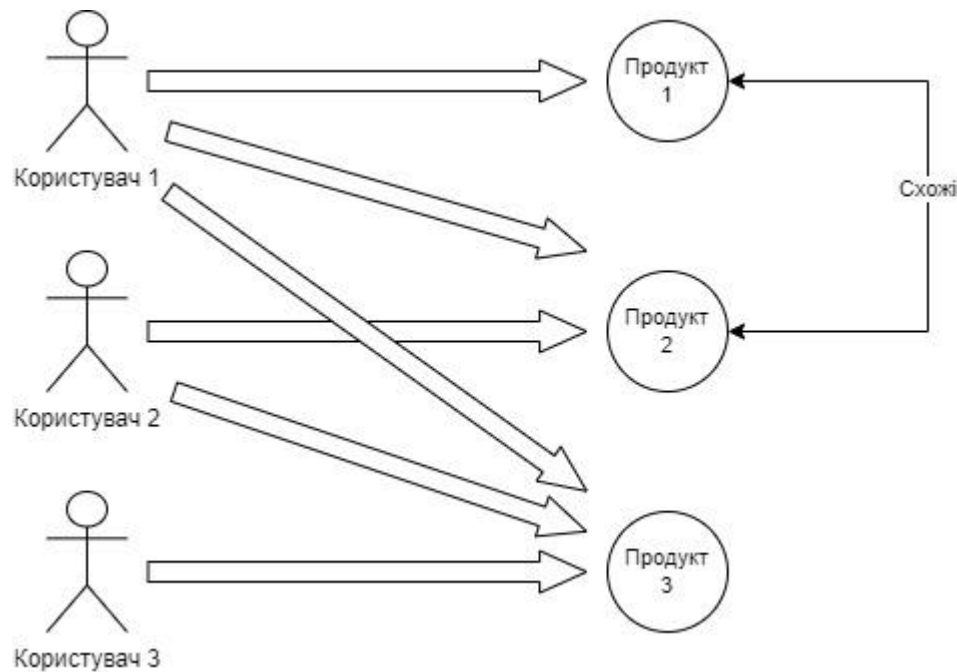


Рисунок 2.5 – Схема роботи item-based рекомендаційних систем

Колаборативна фільтрація на основі подібності користувачів (User-based) зазвичай має вищу точність в порівнянні з колаборативною фільтрацією на основі продуктів (Item-based). Однак, недоліком є ресурсомісткість (вимога до пам'яті) і складність (кількість обчислень, необхідну для отримання рекомендацій).

2.2 Кореляції та її методи

Кореляційний аналіз (від англ. Correlation analysis) – це статистичний метод вивчення взаємозв'язку між двома та більш випадковими величинами. Як випадкові величини в емпіричних дослідженнях виступають значення змінних, вимірювані властивості досліджуваних об'єктів спостереження. Суть кореляційного аналізу полягає у розрахунку коефіцієнтів кореляції.

Коефіцієнти кореляції можуть приймати, як правило, позитивні та негативні значення. Знак коефіцієнта кореляції дозволяє інтерпретувати напрямок зв'язку, а абсолютне значення – силу зв'язку.

Спосіб розрахунку коефіцієнтів кореляції залежить від шкал вимірювання змінних, між якими досліджується взаємозв'язок. Для змінних, що вимірюються в кількісній шкалі (інтервальній шкалі або шкалі відносин), розраховують коваріацію або кореляційний момент, а на його основі лінійний коефіцієнт кореляції.

Для оцінки сили та напрямки зв'язку між змінними, виміряними у порядковій шкалі, використовуються непараметричні рангові коефіцієнти кореляції. Існують заходи оцінки зв'язку та між дихотомічними змінними.

Кореляційний аналіз використовується в економіці, соціології та психології, в рекомендаційних системах, медицині, управлінні якістю, біометрії та інших сферах. Популярність кореляційного аналізу пояснюється тим, що коефіцієнти кореляції відносно прості у розрахунку, та їх застосування не потребує спеціальної математичної підготовки. З іншого боку – коефіцієнти кореляції легко інтерпретувати [7].

Зазвичай, кореляція використовується у рекомендаційних системах для того, щоб порівняти наскільки схожі між собою об'єкти, наприклад користувачі чи якісь продукти. Результати порівнянь зберігаються у так званих матрицях кореляції, які представляють собою звичайну таблицю, в якій у колонках і рядках зберігається інформація про об'єкти, а на перетині величина схожості цих двох об'єктів. Приклад матриці кореляції наведений у таблиці 2.1.

Таблиця 2.1 – Приклад матриці кореляції

Користувач/користувач	Користувач 1	Користувач 2	Користувач 3
Користувач 1	1	0.548193	0.018373
Користувач 2	0.891883	1	-0.493831
Користувач 3	-0.351102	0.103381	1

2.2.1 Метод кореляції Пірсона

Коефіцієнт кореляції Пірсона (позначають «r») – це показник кореляції (лінійної залежності) між двома змінними X та Y , який набуває значень від -1 до $+1$ включно. Він широко використовується в науці для вимірювання ступеня лінійної залежності між двома змінними. Показник був розроблений Карлом Пірсоном зі схожої ідеї, представленої Френсісом Гальтоном в 1880-х рр.

Формулу розрахунку кореляції на основі метода Пірсона зазначено нижче (2.1).

$$r_{XY} = \frac{\sum_{i=1}^N (X_i - \bar{X}) \times (Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^N (X_i - \bar{X})^2 \times \sum_{i=1}^N (Y_i - \bar{Y})^2}}, \quad (2.1)$$

де N – кількість елементів;

X_i – елемент з вибірки X з певним індексом;

Y_i – елемент з вибірки Y з певним індексом;

$\bar{X}$ – середнє значення вибірки X ;

$\bar{Y}$ – середнє значення вибірки Y .

Коефіцієнт кореляції Пірсона є симетричним показником, так як кореляція між X та Y така ж, як між Y та X . Якщо є причинно-наслідковий зв'язок, то має бути і кореляція; але, якщо є кореляція, то не обов'язково є причинно-наслідковий зв'язок [8].

2.2.2 Метод кореляції Спірмена

Для встановлення тісноти зв'язку як між кількісними, так і між атрибутивними ознаками, за умови, що значення цих ознак можуть бути проранжовані за ступенем зниження або зростання, використовують коефіцієнт кореляції рангів. Його основна відмінність полягає в тому, що він обчислюється не на основі первинних даних, а на основі рангів, які присвоюються усім значенням досліджуваних ознак, що розміщені у порядку зростання або зниження. Якщо у кількох одиниць значення ознаки співпадають, то ранг кожної з них дорівнює середньому значенню з рангів, які припадають на ці одиниці. Приклад рангів наведено у таблиці 2.2.

Таблиця 2.2 – Приклад рангів

X	Y	Ранг X, d_x	Ранг Y, d_y
28	21	1	1
30	25	2	2
36	29	4	3
40	31	5	4
30	32	3	5
46	34	6	6
56	35	8	7
54	38	7	8
60	39	10	9
56	41	9	10
60	42	11	11

Отже, до застосування цього коефіцієнта вдаються, коли розмір варіюючої ознаки має значення не сам по собі, а лише з тієї точки зору, чи більший він, чи менший від інших величин [9].

Формулу розрахунку кореляції на основі метода Спірмена зазначено нижче (2.2).

$$r_s = 1 - \frac{6 \times \left(\sum_{i=1}^N d_i^2 \right)}{N \times (N^2 - 1)}, \quad (2.2)$$

де N – кількість елементів;

d_i – різниця між рангами елементу з певним індексом.

2.2.3 Метод кореляції Жаккара

Коефіцієнт подібності Жаккара (спочатку присвоєний французькій назві «коефіцієнт комунауті» від Пола Жаккара) – це статистика, яка використовується для оцінки подібності та різноманітності з вибірок даних.

Коефіцієнт Жаккара вимірює подібність між кінцевими вибірковими наборами і визначається як розмір перетину, поділений на розмір об'єднання наборів вибірок [10]. Візуалізація операція перетину та об'єднання множин зображено на рисунку 2.6.

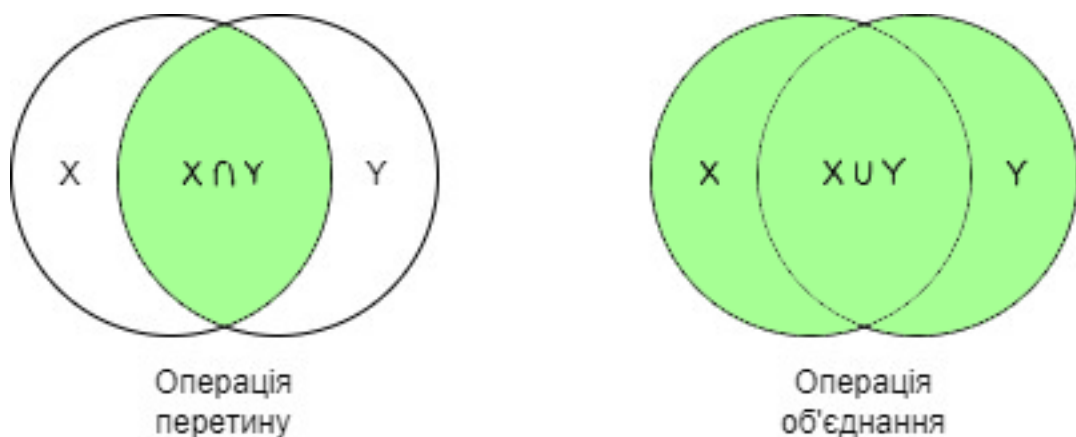


Рисунок 2.6 – Візуалізація операцій перетину та об'єднання множин

Формулу розрахунку кореляції на основі метода Жаккара зазначено нижче (2.3).

$$J_{XY} = \frac{|X \cap Y|}{|X \cup Y|} \quad (2.3)$$

Індекс Жаккара корисний для визначення подібності між об'єктами, які з складаються з двійкових атрибутів. Тобто дві послідовності, кожна з яких складається з двійкових атрибутів. Кожен атрибут може бути 0 чи 1.

2.2.4 Метод косинусної міри подібності

Косинусна міра подібності – це міра кута між двома ненульовими n -вимірними векторами, які представляють об'єкти у n -вимірному просторі.

Два однаково напрямлені вектори мають косинусну міру подібності, рівну 1, два ортогональні вектори мають міру подібності 0, а два вектори, що мають діаметрально протилежні напрями, мають міру подібності -1. Слід зазначити, що в основному в задачах використовується косинусна міра подібності в діапазоні від 0 до 1.

Одиничні вектори вважаються максимально «подібними», якщо вони паралельні, а у випадку ортогональності вектори вважаються максимально «розбіжними».

Формулу розрахунку кореляції на основі методу косинусної подібності зазначено нижче (2.4).

$$\cos_{XY} = \frac{\sum_{i=1}^N X_i \times Y_i}{\sqrt{\sum_{i=1}^N X_i^2} \times \sqrt{\sum_{i=1}^N Y_i^2}}, \quad (2.4)$$

де N – кількість елементів;

X_i – елемент з вибірки X з певним індексом;

Y_i – елемент з вибірки Y з певним індексом.

2.3 Оцінювання рекомендаційних систем

Зазвичай, для того, щоб оцінити розроблену рекомендаційну систему вхідні дані ділять на дві частини: для тренування та для тестування приблизно у співвідношенні 80/30 відповідно.

Дані для тренування – це та частина даних, яку рекомендаційна система буде використовувати для того, щоб навчитися генерувати рекомендації.

Дані для тестування – це та частина даних, про яку рекомендаційна система не буде знати. Ці дані будуть вважатися за справжній вибір користувачів у майбутньому.

Процес тестування розробленої рекомендаційної системи відбувається наступним чином:

- РС навчається на даних для тренування;
- РС генерує список рекомендацій для користувача;
- отриманий список рекомендацій порівнюється зі списком «майбутніх» даних користувача з тестової вибірки.

Основна перешкода для розробки та покращення рекомендаційних систем – це вибір метрик для оцінки точності та релевантності рекомендацій. Це може бути складно, тому що в багатьох випадках метою не є рекомендувати ті самі продукти, які користувач купив раніше.

Усі метрики якості можна умовно поділити на три категорії:

- Prediction Accuracy, оцінюють точність прогнозованого рейтингу;
- Decision support, оцінюють релевантність рекомендацій;
- Rank Accuracy, оцінюють якість ранжування рекомендацій, що видаються.

Одним з найкращих методів оцінки (Decision support категорії) релевантності рекомендаційної системи являється метод оцінки точності (від англ. Precision, P), який розраховується за формулою 2.5.

Нехай $Relevance(i)$ – це індикатор релевантності рекомендованого товару на i -тій позиції, який може набувати значення 0 або 1.

$$P@N = \frac{1}{N} \times \sum_{i=1}^N Relevance(i), \quad (2.5)$$

де N – кількість елементів.

Але Precision не дбає про порядок. Тому краще використовувати точність при відсіканні i елементів. Припустимо, що генерується N рекомендацій і розглядаємо лише перший елемент, потім лише перші два, потім лише перші три тощо. ці підмножини можуть бути проіндексовані за допомогою i .

Precision у точці відсікання i , $P@i$ – це просто Precision, розрахований з урахуванням лише підмножини рекомендацій від рангу 1 до i . Оцінка рекомендацій визначається прогнозованою цінністю. Наприклад, продукт із найвищим прогнозованим значенням ранжується 1, продукт з i -м найвищим прогнозованим значенням ранжується i (рис. 2.7).

Ранг	Рекомендовані нагороди (ID)	Вгадано нагороду
1	34	Ні
2	56	Так
3	6	Ні
4	12	Так
5	91	Ні

$P@3 = 1/3$

Ранг	Рекомендовані нагороди (ID)	Вгадано нагороду
1	34	Ні
2	56	Так
3	6	Ні
4	12	Так
5	91	Ні

$P@4 = 2/4$

Рисунок 2.7 – Приклад розрахунку Precision@K

Метрика вказує, скільки серед рекомендованих товарів було релевантних для одного користувача. Недоліком є те, що вона не враховує порядок розташування релевантних товарів. Тому використовується середня точність (від англ. Average Precision, AP), яка розраховується за формулою 2.6.

$$AP@N = \frac{1}{N} \times \sum_{i=1}^N \text{Relevance}(i) \times P@i, \quad (2.6)$$

де N – кількість елементів.

Для того, щоб розрахувати $AP@K$ для кожного з користувачів, які наявні у вибірці для проведення тестування, необхідно використати середню точність для кожної середньої точності (від англ. Mean Average Precision, MAP), яка розраховується за формулою 2.7.

$$MAP@N = \frac{1}{U} \sum_{i=1}^U AP@N_i, \quad (2.7)$$

де N – кількість елементів;

U – кількість користувачів.

2.4 Типи даних

Для роботи РС необхідні дані, з якими вона буде взаємодіяти. Існує два основних методи збору цих даних: явний та неявний. Дані, отримані цими двома методами так і називаються: явні дані та неявні дані відповідно.

Явні дані – це такі дані, які отримуються від користувача в якості відповіді на певні «запитання» системи. Зазвичай, явні дані отримують за ініціативи користувача. Приклади явних даних:

– запит у користувача оцінки певного товару;

- запит у користувача ранжування певної групи товарів від найкращого до найгіршого;

- пропонування користувачу двох продуктів з прохання обрати кращий з них;

- запит у користувача на створення списку улюблених продуктів.

Неявні – це такі дані, які збираються системою без потреби посередніх дій користувачем. Такі дані формуються поведінкою користувача на сервісі. Приклади неявних даних:

- фіксування часу, який витратив користувач на прочитання певної статті;

- кількість кліків користувача на певних сторінки сайту;

- відстежування «зон уваги» користувача на певних сторінках;

- географічні дані користувача;

- технічні характеристики пристрою, з якого відвідую сайт користувач.

Наприклад, фіксується, скільки часу користувач читав ту чи іншу статтю, скільки користувач робив кліків мишкою на різних сторінках сайту.

Довіра до рекомендації будується на тому, що користувач розуміє, чому саме такий контент (реklamний банер) йому був рекомендований. Користувачі набагато краще сприймають рекомендації, якщо надавати їх якимись поясненнями. Наприклад, у кожному рекламному банері на відеохостингу YouTube можна дізнатися про причини показу такої реклами.

Наприклад, застосунок Google Chrome збирає наступні дані про користувачів:

- геолокація;

- аудіо дані;

- історія переглядів;

- унікальні характеристики пристрою;

- дані про фінансові операції;

– статистичні дані про кліки користувача.

Можна побачити цікаву закономірність, чим більші компанії – тим більше вони збирають персональних даних своїх користувачів для подальшого власного оброблення чи для продажу інших зацікавленим компаніям.

2.5 Формат збереження даних

Одним з найпопулярніших форматів файлів для збереження даних є CSV (англ. Comma-Separated Values). Формат CSV використовують для зберігання таблиць у текстових файлах. Дані часто упаковують саме в таблиці, тому файли у CSV форматі дуже популярні.

У CSV є купа плюсів перед іншими форматами, бо це все ж таки звичайний текстовий файл, який відкривається і зчитується дуже швидко на будь-якому пристрої та у будь-якому середовищі без додаткових інструментів.

Через свої переваги CSV став найпопулярнішим форматом обміну даними, хоча йому вже багато років.

Рядки файлу CSV формату – це окремі рядки таблиці. Значення кожної окремої колонки – це значення, розділені комами. Якщо значення містять зарезервовані символи, то як роздільник застосовуються подвійні лапки.

Варто відзначити, що подібний формат даних, що використовується великою кількістю програмних комплексів, деякі з утиліт застосовують у ролі роздільників такий вид, як крапка з комою.

Тип файлу, що має розширення CSV, містить інформацію, яку можуть імпортувати в базу даних або експортувати з неї, так само подібний текстовий файл може мати дані, необхідні для організації таблиць.

CSV файл може прочитати текстовий редактор, причому до списку таких редакторів входить більшість програм, які застосовуються для роботи з текстом та таблицями.

Зазвичай, у першому рядку CSV файлу зазначають назви стовпців.

Припустимо, що у нас є таблиця (табл. 2.3), в якій зберігаються дані про деяких користувачів.

Таблиця 2.3 – Дані про користувачів

Ім'я користувача (Name)	Електронна адреса (Email)	Країна (Country)
Denys	test@gmail.com	Ukraine
Oleg	olg63@gmail.com	Poland
Jose	rater@gmail.com	USA
Bogdan	bogdan.oil@ukr.net	Ukraine
Matt	matt.tepper@gmail.com	USA
Maxim	max818@ukr.net	Ukraine
Olga	olg7a_test@gmail.com	Poland

Якщо перевести ці дані в CSV формат, то вміст файлу буде виглядати наступним чином:

```
Name,Email,Country
Denys,test@gmail.com,Ukraine
Oleg,olg63@gmail.com,Poland
Jose,rater@gmail.com,USA
Bogdan,bogdan.oil@ukr.net,Ukraine
Matt,matt.tepper@gmail.com,USA
Maxim,max818@ukr.net,Ukraine
Olga,olg7a_test@gmail.com,Poland
```

У такому вигляді необхідні дані можна легко транспортувати з однієї системи в іншу.

2.6 Мова програмування та допоміжні бібліотеки

Для розробки рекомендаційної системи було обрано мову програмування Python через ряд її переваг:

- гнучкість, яка надає можливість змінювати певні аспекти поведінки мови програмування під конкретні задачі;
- можливість розширення, за допомогою якої відпадає необхідність дублювати код, достатньо просто імпортувати вже готовий код чи чийсь бібліотеку;
- простий синтаксис, з якого було прибрано все зайве, через що код чистий і зрозумілий без зайвих дужок і виразів.;
- інтерпретованість, за допомогою якої Python існує на всіх популярних платформах і за замовчуванням входить в більшість дистрибутивів Linux, а значить є на більшості серверів;
- величезна кількість вже готових бібліотек, які допоможуть вирішити майже будь-яку задачу;
- PEP (Python Enhancement Proposals), це єдиний стандарт для написання коду, що робить код підтримуваним і читабельним навіть при переході від одного програміста до іншого;
- відкритість коду, що дозволяє будь-кому, хто зацікавлений у розвитку мови, взяти участь в його розробці і поліпшити його;
- ком'юніті, яке утворилося навколо цієї мови програмування, яке готове прийти на допомогу будь-якому початківцю або вже вмілому розробнику і розібратися в його проблемі.

Однією з найкращих бібліотек для мови програмування Python, яка призначена для роботи зі списками даних являється бібліотека NumPy [11].

NumPy додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою математичних функцій для операцій з цими масивами. Попередник NumPy, Numeric, був спочатку створений Jim Hugunin.

Оскільки Python – це інтерпретована мова, математичні алгоритми, часто працюють в ньому набагато повільніше ніж у компільованих мовах, таких як C або навіть Java. NumPy намагається вирішити цю проблему для великої кількості обчислювальних алгоритмів забезпечуючи підтримку багатовимірних масивів і безліч функцій і операторів для роботи з ними. Таким чином будь-який алгоритм, який може бути виражений в основному як послідовність операцій над масивами і матрицями, працює так само швидко, як еквівалентний код, написаний на C.

NumPy можна розглядати як гарну вільну альтернативу MATLAB, оскільки мова програмування MATLAB зовні нагадує NumPy: обидві вони інтерпретовані, і обидві дозволяють користувачам писати швидкі програми, поки більшість операцій проводяться над масивами або матрицями, а не над скалярами.

NumPy додає об'єкт масиву, який приблизно у 50 разів швидший за традиційні списки у мові програмування Python. Об'єкт масиву в NumPy називається `ndarray`, він надає багато допоміжних функцій, які роблять роботу з `ndarray` дуже легкою. Масиви дуже часто використовуються в науці про дані, де швидкість і ресурси дуже важливі [12].

Порівняння швидкості роботи операції додавання нового елементу до масиву з бібліотеки NumPy з тією ж самою операцією, але з використанням вбудованим списком мови програмування Python зображено на рисунку 2.8.

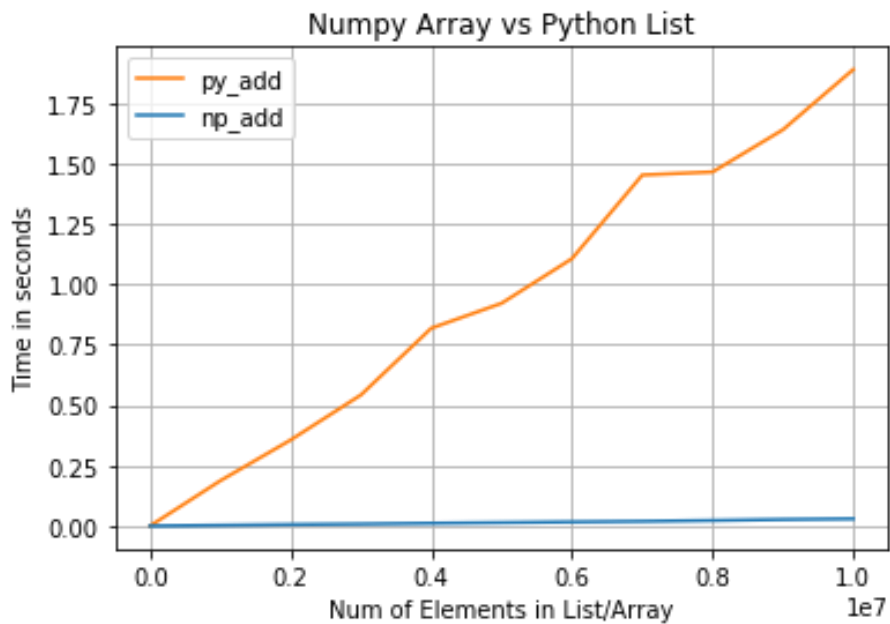


Рисунок 2.8 – Порівняння швидкості роботи масиву з бібліотеки NumPy з вбудованим списком мови програмування Python

Ще однією потужною бібліотекою мови програмування Python можна назвати бібліотеку Pandas [13].

Pandas – це програмна бібліотека, написана для мови програмування Python для маніпулювання даними та їхнього аналізу. Вона, зокрема, пропонує структури даних та операції для маніпулювання чисельними таблицями та часовими рядами.

Pandas є вільним програмним забезпеченням, що випускається за ліцензією BSD. Ця назва походить від терміну «панельні дані» (від англ. panel data), який в економетрії позначає багатовимірні структуровані набори даних [14].

В основі Pandas лежить клас DataFrame, що надає можливості роботи із двовимірними масивами неоднотипних даних. Об'єкт DataFrame складається з об'єктів Series – одновимірних масивів NumPy ndarray, об'єднаних під однією назвою. Тому в DataFrame кожен стовпець може мати свій тип даних. На рисунку 2.9 схематично показано архітектуру DataFrame.

pandas.DataFrame		
pandas.Series	pandas.Series	pandas.Series
Ім'я	Рік народження	Зріст
Олег	1998	183
Артур	2000	179
Олександр	1997	182

Рисунок 2.9 – Приклад об'єкту DataFrame з бібліотеки Pandas

На відміну від NumPy, бібліотека Pandas не вимагає знань лінійної алгебра або умінь працювати з багатовимірними масивами. DataFrame можна представити як таблицю, де кожен рядок представляє масив. А з рядками та стовпцями зручно працювати. Бібліотека Pandas підтримує такі самі операції, як і бібліотека NumPy:

- індексація масивів;
- статистичні дослідження;
- робота з порожніми (Nan) значеннями
- зміна форми масивів;
- вилучення окремих елементів;
- конкатенація.

Крім цього, Pandas також дозволяє: читання та запис даних, наприклад, CSV або Excel таблиця; угруповання даних; засоби візуалізації даних; роботу із датами; роботу з рядковими значеннями тощо [15].

Зважаючи на усі існуючі можливості бібліотеки NumPy та Pandas для мови програмування Python можна зробити висновок, що необхідно

використовувати ці дві бібліотеки в парі, бо без цих бібліотек важко уявити обробку великого обсягу даних і реалізацію важких алгоритмів, які потребують багато обчислюючих ресурсів.

Для реалізації API сервера було обрано найкращий веб-фреймворк за результатами рейтингу HotFrameworks [16] – Django. Скріншот з результатами рейтингу наведений на рисунку 2.10.

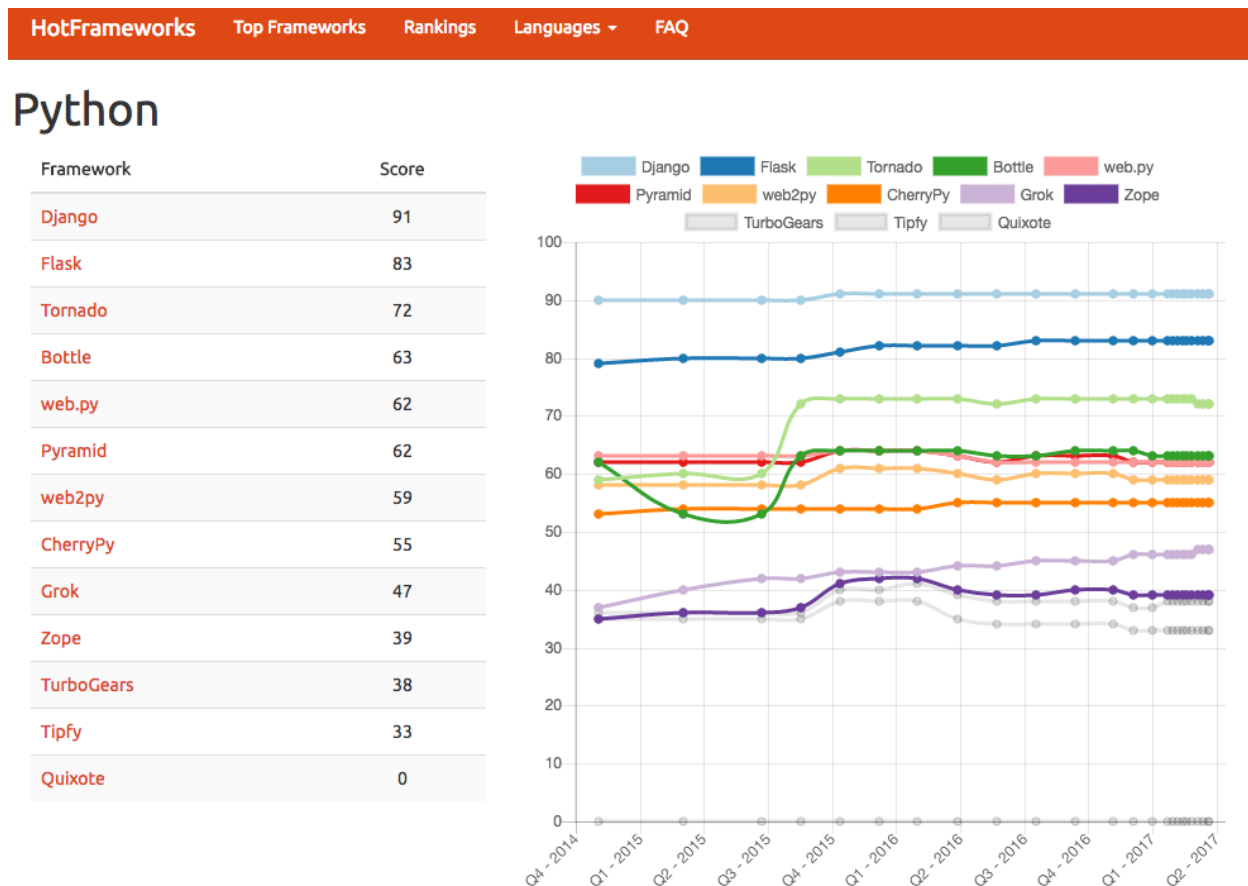


Рисунок 2.10 – Скріншот з результатами рейтингу HotFrameworks

Django – це безкоштовний та вільний фреймворк для веб-додатків, написаний на мові програмування Python. Фреймворк включає в себе набір компонентів, які допомагають розробляти веб-застосунки швидко і просто.

Щоразу під час розробки веб-застосунків потрібні схожі компоненти: спосіб аутентифікувати користувачів (вхід, вихід, реєстрація), панель управління системою, форми, інструменти для завантаження файлів тощо.

Фреймворки існують, щоб полегшити процес розробки та дозволити розробникам не «винаходити колесо» кожного разу.

З Django, можна створювати веб-застосунки від концепції до готового рішення за лічені години. Django дозволяє розробникам зосереджуватися саме на написанні логіки веб-сервісів, а не на написанні тих частин, які необхідно писати з нуля раз за разом, від одного проєкту до іншого.

Django надає наступні переваги:

- дуже швидко. Django був розроблений для того, щоб допомогти розробникам максимально швидко реалізовувати застосунки від концепції до завершення;

- усе включено. Django включає в себе десятки різних доповнень, які можна використовувати для вирішення майже будь-яких задач веб-розробки;

- безпечність. Django дуже прискіпливо відноситься до питання безпеки, надає можливість уникнути багатьох поширених помилок пов'язаних з безпекою, таких як ін'єкції коду, міжсайтовий скриптинг, підробка міжсайтових запитів;

- масштабованість. Деякі з найбільш навантажених сайтів на планеті використовують властивості цього фреймворку задля можливості гнучкого масштабування;

- універсальність. Компанії, організації та уряди використовують Django для створення різноманітних речей, починаючи з систем управління контентом та закінчуючи соціальними мережами та наукових платформ [17].

З ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

Сучасні онлайн сервіси працюють з використанням клієнт-серверної архітектури. В основі клієнт-серверної архітектури лежать два компоненти: клієнт і сервер.

Клієнт – це комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій.

Сервер – це більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних.

Зазвичай, під терміном «клієнт» також мають на увазі програмний застосунок, який виконує функцію інтерфейсу між користувачем та сервером. Клієнтська частина (від. англ. Front-end) найчастіше представлена веб-сторінкою або мобільним застосунком.

Серверна частина (від. англ. Back-end) може бути представлена застосунком, який приймає вхідні запити, оброблює їх та відправляє відповідь на них. Також, серверна частина відповідає за безпечну комунікацію з базою даних (БД).

Найпопулярнішим методом взаємодії клієнтської та серверної частин є використання HTTP (англ. Hyper Text Transport Protocol) протоколу.

HTTP – це протокол прикладного рівня, в якому обмін повідомленнями йде за звичайною схемою «запит-відповідь». Для ідентифікації ресурсів HTTP використовує глобальні веб-адреси. На відміну від багатьох інших протоколів, HTTP не зберігає свого стану. Це означає відсутність збереження проміжного стану між парами «запит-відповідь» [18].

На даний момент завдяки протоколу HTTP забезпечується робота всесвітньої павутини.

Компоненти, що використовують HTTP, можуть самостійно здійснювати збереження інформації про стан, пов'язаний з останніми запитами та відповідями.

Взаємодія користувача з сервісом відбувається наступним чином:

- користувач використовуючи клієнтську частину виконує певну дію, наприклад натискає певну кнопку;
- клієнтська частина формує та відправляє HTTP запит до серверної частини;
- серверна частина приймає запит, оброблює його, при необхідності взаємодіє з даними використовуючи базу даних, формує та відправляє відповідь;
- клієнтська частина приймає відповідь від серверної частини, та візуалізує результат користувачу за допомогою візуального інтерфейсу.

Схема роботи клієнт-серверної архітектури зображена на рисунку 3.1.

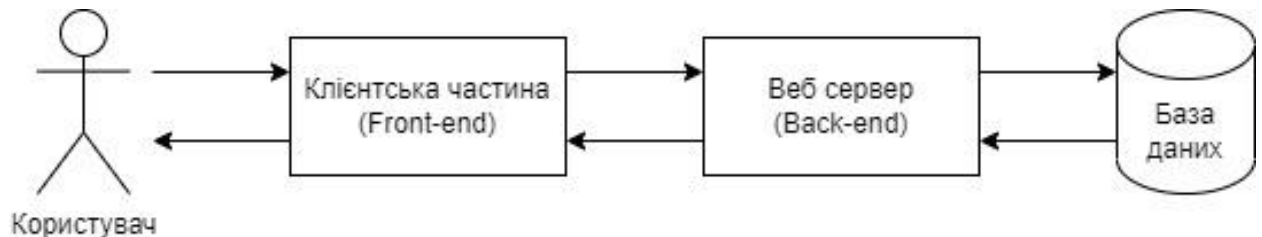


Рисунок 3.1 – Схема роботи клієнт-серверної архітектури

Оскільки зі збільшенням даних збільшується і час розрахунку подібності користувачів і генерації рекомендацій, то перераховувати усі ці дані після кожної нової події використання користувачем нової нагороди стає неможливим через величезне навантаження. Також, стає неможливою генерація рекомендацій для користувача кожного разу, як приходить запит від клієнтської частини на відображення, так як на розрахунок рекомендацій уходить достатньо багато часу.

Було вирішено розробити таку архітектуру, при якій генерація рекомендацій для усіх існуючих користувачів буде виконуватися у фоновому

режимі і з певною регулярністю для підтримання актуальності даних, наприклад раз на тиждень. Для цього рекомендаційний модуль було відокремлено від основної серверної частини. Схема розробленої архітектури зображена на рисунку 3.2.

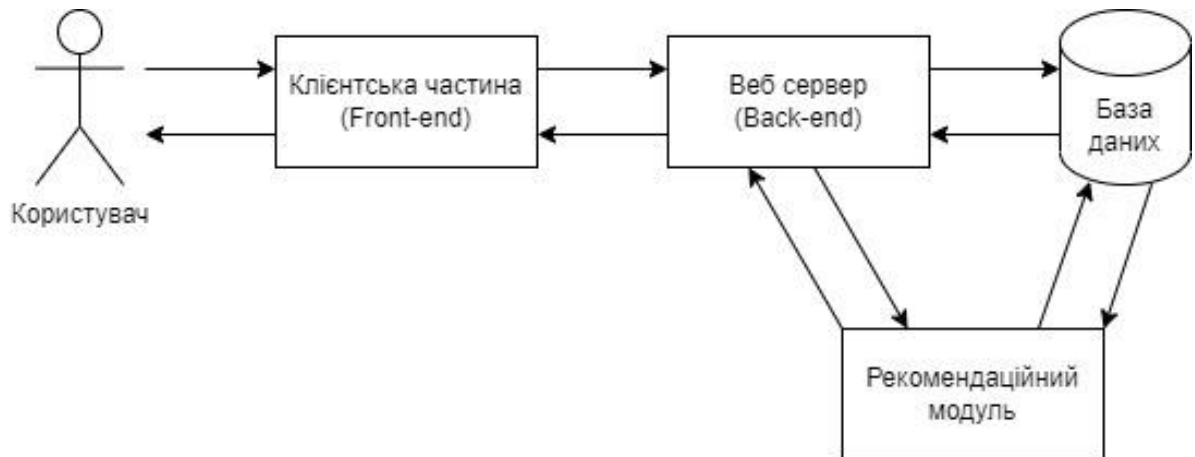


Рисунок 3.2 – Схема розробленої архітектури

В результаті, система буде поділена на онлайн (від англ. online) та офлайн (від англ. offline) умовні частини.

Онлайн частина буде відповідати за отримання HTTP запити, діставання заздалегідь згенерованого списку нагород, формування та відправку HTTP відповіді.

Офлайн частина, в свою чергу, буде відповідати за щотижневе генерування рекомендованих користувачам нагород.

3.1 Онлайн частина

Онлайн частина буде представлена веб-застосунком, який буде отримувати HTTP запити, оброблювати їх та повертати сформовані відповіді.

Буде реалізований спеціальний ендпоінт (від англ. endpoint), який буде відповідати за отримання списку рекомендацій для певного користувача.

Ендпоінт – це по факту звичайна URL адреса, яка використовується для отримання, створення, модифікації чи видалення певного ресурсу.

Сервер прийме цей HTTP запит, обробить його, та поверне у результаті список з унікальними ідентифікаторами (ID) рекомендованих нагород, в якому нагороди будуть відсортовані за релевантністю рекомендації.

Цей ендпоінт зможуть використовувати будь-які клієнтські застосунки, наприклад веб-браузери чи мобільні застосунки.

3.2 Офлайн частина

Для запуску процесу генерації рекомендацій, який буде окремо запускатися у фоновому режимі, буде використовуватися програма Cron.

Cron – це утиліта в операційних системах Unix і Linux, яка дозволяє користувачам виконувати команди або скрипти (групи команд) автоматично в заданий час.

Розпорядок запуску скриптів зберігається у спеціальному файлі під назвою crontab. Кожний рядок у цьому файлі складається з п'яти колонок, що відокремлюються пробілами чи табуляціями. Столпчики задають час виконання (Хвилина, Година, День, Місяць, День тижня), в них може міститися число, список чисел, або символ «*». Решта символів в рядку інтерпретуються як виконувана команда та її параметри.

Для того, щоб налаштувати запуск процесу генерації рекомендацій кожного тижня, необхідно використати таке значення «0 0 * * 0», або спеціальний макрос «@weekly».

В результаті щотижневого перерахунку рекомендацій, буде створюватися Python об'єкт, в якому буде зберігатися велика таблиця з рекомендованими нагородами для кожного з існуючих користувачів. Приклад такої таблиці наведений у таблиці 3.1.

Таблиця 3.1 – Приклад згенерованої таблиці рекомендацій нагород для кожного користувача

ID користувача	ID нагороди	ID нагороди	ID нагороди	ID нагороди
1	67	55	43	87
2	12	43	88	15
3	67	43	57	44
4	90	12	15	7
5	89	5	66	13

Після чого, цей об'єкт, в якому буде зберігатися таблиця рекомендацій та спеціальні допоміжні методи, буде збережено у спеціальний файл за допомогою утиліти pickle.

Pickle – це спеціальний інструмент мови програмування Python, який надає можливість в будь який момент зберегти об'єкт з усіма його даними у спеціальний файл. При необхідності цей файл можна знову імпортувати та отримати початковий об'єкт саме в тому стані, в якому він перебував на початку.

Наступним кроком збережений об'єкт буде переноситися на веб-сервер та імпортуватися у систему. Після чого, при надходженні запиту на отримання списку рекомендованих нагород для певного користувача система буде дуже швидко діставати заздалегідь згенерований список рекомендованих нагород та відправляти його у відповідь.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

4.1 Вхідні дані

Через те, що наявні програми лояльності не публікують реальні дані про дії своїх користувачів, було вирішено згенерувати такі дані власноруч.

Для генерації таких даних було написано спеціальну програму на мові програмування Python. Ця програма після свого виконання генерує два файли формату CSV з однаковою структурою але з різними даними та різним розміром.

Перший файл – це файл, призначений для тренування розробленої рекомендаційної системи.

Другий файл – це файл з даними, за допомогою яких буде проводитися оцінка роботи того чи іншого реалізованого рекомендаційного алгоритму.

У кожному з цих файлів знаходяться дані про те, який користувач яку саме нагороду використав і скільки разів. Приклад структури згенерованих CSV файлів наведений нижче:

```
user_id,reward_id,uses
1,5,4
2,83,3
2,17,11
3,17,2
4,6,5
4,11,3
```

Перед безпосереднім запуском цієї програми є можливість налаштувати деякі параметри генерування даних:

- `unique_users` – це кількість унікальних користувачів;
- `unique_rewards` – це кількість унікальних нагород;

– `max_rewards_per_user` – це значення, за допомогою якого є можливість регулювати скільки один користувач зможе використати різних нагород;

– `max_reward_uses_per_user` – це значення, яке надає можливість вказувати скільки один користувач зможе максимально використати одну й ту ж саму нагороду;

– `test_data_size` – це значення, яка надає можливість корегувати об’єм даних у тестовому файлі.

Налаштування програми для генерації даних про користувачів та їх використані нагороди для цього експерименту зазначені у таблиці 4.1.

Таблиця 4.1 – Налаштування програми генерації даних про користувачів та їх використані нагороди

Назва характеристики	Значення
<code>unique_users</code>	10 000
<code>unique_rewards</code>	2 000
<code>max_rewards_per_user</code>	10
<code>max_reward_uses_per_user</code>	15
<code>test_data_size</code>	0.07

За результатами виконання цієї програми було згенеровано два файли:

– `train_data.csv` – це файл з даними для тренування рекомендаційної системи. Кількість записів в якому становить 46907;

– `test_data.csv` – це файл з даними для проведення оцінки розробленої рекомендаційної системи. Кількість записів в якому становить 3530.

В подальшому ці файли будуть використовуватися при навчанні та тестуванні розробленої рекомендаційної системи.

4.2 Оптимізація зберігання даних

При обробці великих об'ємів даних виникає проблема недостатності розміру пам'яті, як оперативної (під час проведення розрахунків), так і постійної при збереженні результатів роботи.

Одним із варіантів зменшення об'єму використання пам'яті є правильне налаштування типів даних у бібліотеці NumPy. Бібліотека NumPy вміє дуже добре керувати виділенням пам'яті при створенні масивів чи матриць.

Для того щоб зменшити витрати пам'яті для якогось конкретного масиву, йому необхідно вказати один з існуючих типів даних (табл. 4.2) [19].

Таблиця 4.2 – Основні типи даних у бібліотеці NumPy

Назва	Значення
1	2
int8	Цілі числа в діапазоні від -128 до 127 (числа розміром 1 байт).
int16	Цілі числа в діапазоні від -32768 до 32767, (числа розміром 2 байти).
int32	Цілі числа в діапазоні від -2147483648 по 2147483647 (числа розміром 4 байти).
int64	Цілі числа в діапазоні від -9223372036854775808 по 9223372036854775807 (числа розміром 8 байт).
uint8	Цілі числа в діапазоні від 0 до 255 (числа розміром 1 байт).

Продовження таблиці 4.2

1	2
uint16	Цілі числа в діапазоні від 0 до 65535 (числа розміром 2 байти).
uint32	Цілі числа в діапазоні від 0 до 4294967295 (числа розміром 4 байти).
uint64	Цілі числа в діапазоні від 0 до 18446744073709551615 (числа розміром 8 байт).
float16	Речові числа половинної точності: 1 біт знака, 5 біт експоненти, 10 біт мантиси (числа розміром 2 байти).
float32	Речові числа одинарної точності: 1 біт знака, 8 біт експоненти, 23 біти мантиси (числа розміром 4 байти).
float64	Речові числа подвійної точності: 1 біт знака, 11 біт експоненти, 52 біта мантиси (числа розміром 8 байт).

Наприклад, якщо ми знаємо, що унікальні ідентифікатори (ID) користувачів та нагород завжди будуть додатними і скоріше за все будуть менше ніж 4294967295, то є сенс вказати тип uint32, який у порівнянні з стандартним типом int64 зекономить на кожному числі по 4 байти.

Для масивів, у яких зберігається кількість використання певної нагороди певним користувачем є сенс використовувати тип даних uint16.

Для економії об'єму пам'яті при зберіганні матриць, в яких більша частина значень дорівнює 0 (розріджені матриці) було вирішено використовувати підхід списку координат (від англ. Coordinate List).

Матриця зазвичай зберігається як двовимірний масив. Кожний запис у масиві є елемент матриці, і до нього звертаються за індексами i та j . Зазвичай

i – це індекс рядка, пронумерований зверху вниз, а j – індекс стовпця, пронумерований зліва направо. Для матриці розміру $m \times n$ обсяг пам'яті, необхідний для зберігання матриці в цьому форматі, пропорційний $m \times n$ (без урахування того факту, що розміри матриці також повинні бути збережені).

У разі розрідженої матриці суттєве скорочення вимог до пам'яті може бути реалізовано шляхом збереження лише ненульових елементів. Залежно кількості і розподілу ненульових записів можна використовувати різні структури даних, що дає величезну економію пам'яті. Компроміс полягає в тому, що доступ до окремих елементів стає складнішим і необхідні додаткові структури, щоб мати можливість однозначно відновити початкову матрицю [20].

Можливо, найбільш очевидним форматом зберігання розріджених матриць є список координат. Такий формат є триплетом (рядок, стовець, значення). При цьому зберігаються лише ненульові значення.

Приклад перетворення розрідженої матриці у список координат наведений на рисунку 4.1. На цьому прикладі вдалося зменшити розмір матриці з 5×7 до 3×5 .



Рисунок 4.1 – Приклад перетворення розрідженої матриці на список координат

4.3 Розробка офлайн частини

Офлайн частина представляє із себе програму, яка повинна на основі даних про використання користувачами нагород розрахувати для кожного із користувачів список рекомендованих нагород. Після чого ця програма повинна зберегти результати (включаючи матрицю рекомендацій) у спеціальний файл (за допомогою утиліти pickle) та передати його до онлайн частини.

Для розробки рекомендаційного модулю було обрано метод неперсоналізованих рекомендацій на основі популярності та персоналізований User-based метод колаборативної фільтрації (рис. 4.2.).

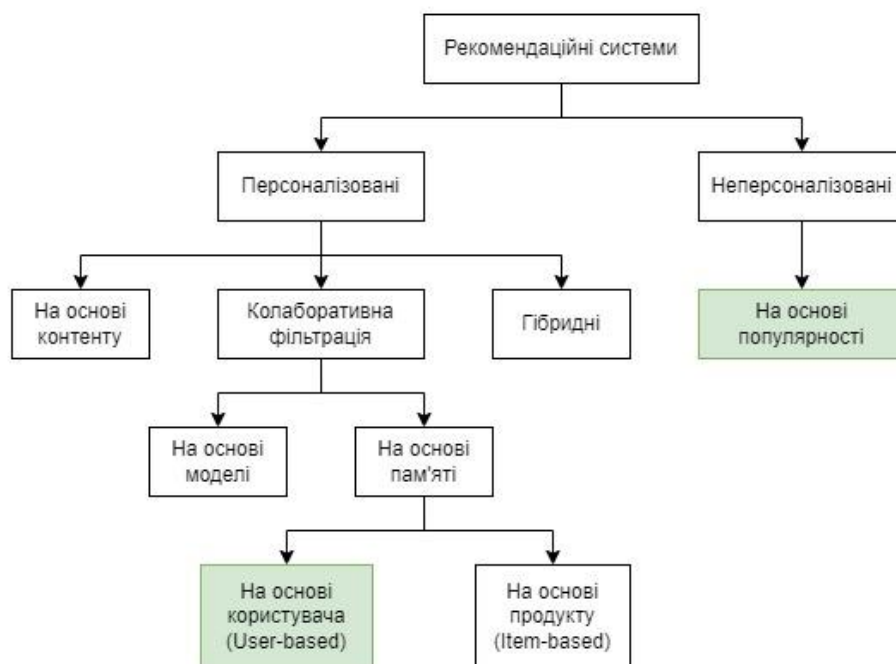


Рисунок 4.2 – Обрані методи генерації рекомендацій

У рекомендаційному модулі за допомогою мови програмування Python реалізований спеціальний клас – Recommender, який включає у себе методи тренування та тестування певного алгоритму.

Було реалізовано шість таких класів:

– TopRecommender – на основі найпопулярніших нагород;

- PearsonRecommender – на основі методу КФ з кореляцією Пірсона;
- SpearmanRecommender – на основі методу КФ з кореляцією Спірмена;
- JaccardRecommender – на основі методу КФ з кореляцією Жаккара;
- CosineSimilarityRecommender – на основі методу КФ з методом косинусу подібності;
- CosineSimilarityWithTopRecommender – на основі поєднання методу КФ з косинусом подібності та найпопулярніших нагород.

Після завантаження даних у рекомендаційну систему виконується побудова матриці кореляції між усіма користувачами. В цій матриці зберігаються значення, які демонструють на скільки один користувач схожий на іншого. Одним з найкращим методів візуального відображення матриць кореляції є теплова карта.

Рекомендаційний скрипт запускається кожен тиждень у фоновому режимі за допомогою утиліти Cron (рис. 4.3).

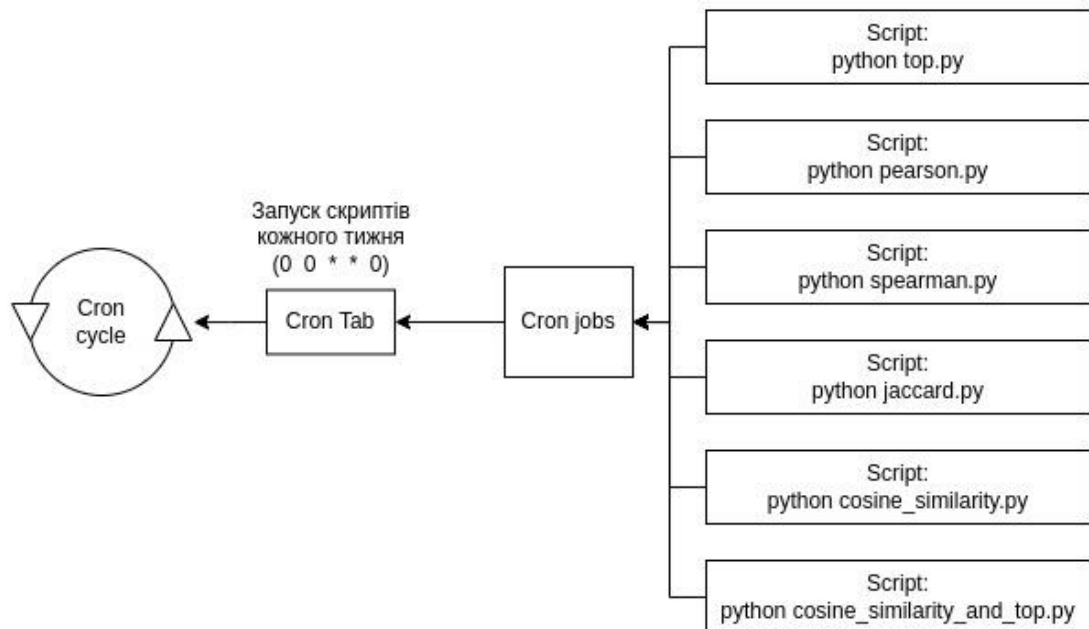


Рисунок 4.3 – Схема роботи утиліти Cron з скриптами рекомендаційного модулю

Теплова карта (від англ. heatmap) – це матричне представлення даних, в якому кожне значення відображається за допомогою певного кольору. Кожній величині відповідає свій колір, а матриця індексів зіставляє два елементи або їх характеристики. Теплові карти показують зв'язки кількох змінних між собою, відображаючи їх величини у вигляді певних кольорів. Також, подивившись інші точки на теплової карті, можна побачити, як у наборі даних один зв'язок порівнюється з усіма іншими. Саме кольори дозволяють легко та швидко проаналізувати дані, оскільки таке позначення є інтуїтивно-зрозумілим [21].

4.3.1 Рекомендації на основі популярності нагород

Найпростішим рішенням генерації рекомендацій можна назвати такий підхід, при якому ми пропонуємо користувачу найпопулярніші нагороди у системі. Для цього нам достатньо лише підрахувати кількість використань для кожної з існуючих нагород.

При такому підході ми ніяк не враховуємо вподобань користувача, іншими словами це є один із варіантів реалізації неперсоналізованих рекомендаційних систем.

У цього рішення є декілька значущих переваг:

- простота реалізації;
- велика швидкість роботи;
- відсутність проблеми «холодного старту».

Результати реалізованої рекомендаційної системи на основі популярності нагород наведені у таблиці 4.3.

Таблиця 4.3 – Результати реалізованої рекомендаційної системи на основі популярності нагород

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0011042790303970572
Час проведення тестування	14.0352 с.

4.3.2 Рекомендації на основі методу кореляції Пірсона

Наступним кроком було реалізовано User-based колаборативну фільтрацію, в якій порівняння схожості користувачів розраховувалося на основі методу кореляції Пірсона.

Значення кореляції, яка розраховується методом Пірсона, може набувати значень від -1 до 1. Чим ближче значення до 1, тим більше один об'єкт схожий на інший. Якщо ж значення дорівнює 1, то це значить, що об'єкти повністю однакові.

Результати генерації рекомендацій та тестування цього підходу наведені у таблиці 4.4.

Таблиця 4.4 – Результати реалізованої рекомендаційної системи на основі методу кореляції Пірсона

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0010544096589509221
Час проведення тестування	7.2215 с.
Об'єм матриці кореляції користувачів	1726,2 МБ
Час генерації матриці кореляції користувачів	8.1992 с.
Об'єм матриці з рекомендованими нагородами	14,1 МБ
Час генерації матриці з рекомендованими нагородами	104.5417 с.

Аналізуючи отриману матрицю кореляції користувачів (рис. 4.4), чітко видно, що діагональ матриці представляє собою суцільну жовту лінію, що означає те, що користувач повністю схожий сам на себе (значення кореляції дорівнює 1).

Також можна зауважити, що на тепловій карті переважають кольори, які відповідають значенням 0.5 та вище. Це свідчить про те, що метод Пірсона у даному випадку схильний вважати більшість користувачів більш схожими на інших.

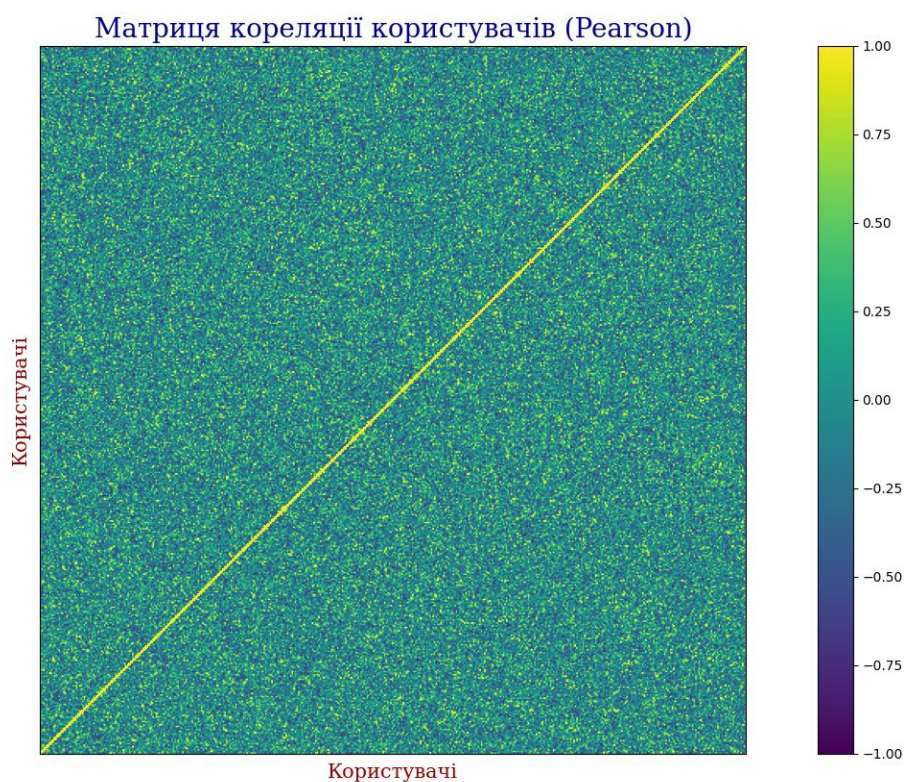


Рисунок 4.4 – Теплова карта матриці кореляції користувачів на основі методу кореляції Пірсона

4.3.3 Рекомендації на основі методу кореляції Спірмена

Наступною було реалізовано User-based колаборативну фільтрацію, в якій порівняння схожості користувачів розраховувалося на основі методу кореляції Спірмена.

Значення кореляції, яка розраховується методом Спірмена, може набувати значень від -1 до 1. Чим ближче значення до 1, тим більше один об'єкт схожий на інший. Якщо ж значення дорівнює 1, то це значить, що об'єкти повністю однакові.

Отримана візуалізація матриці кореляції користувачів на основі методу Спірмена (рис. 4.5) візуально досить сильно нагадує попередню візуалізацію матриці кореляції на основі методу Пірсона.

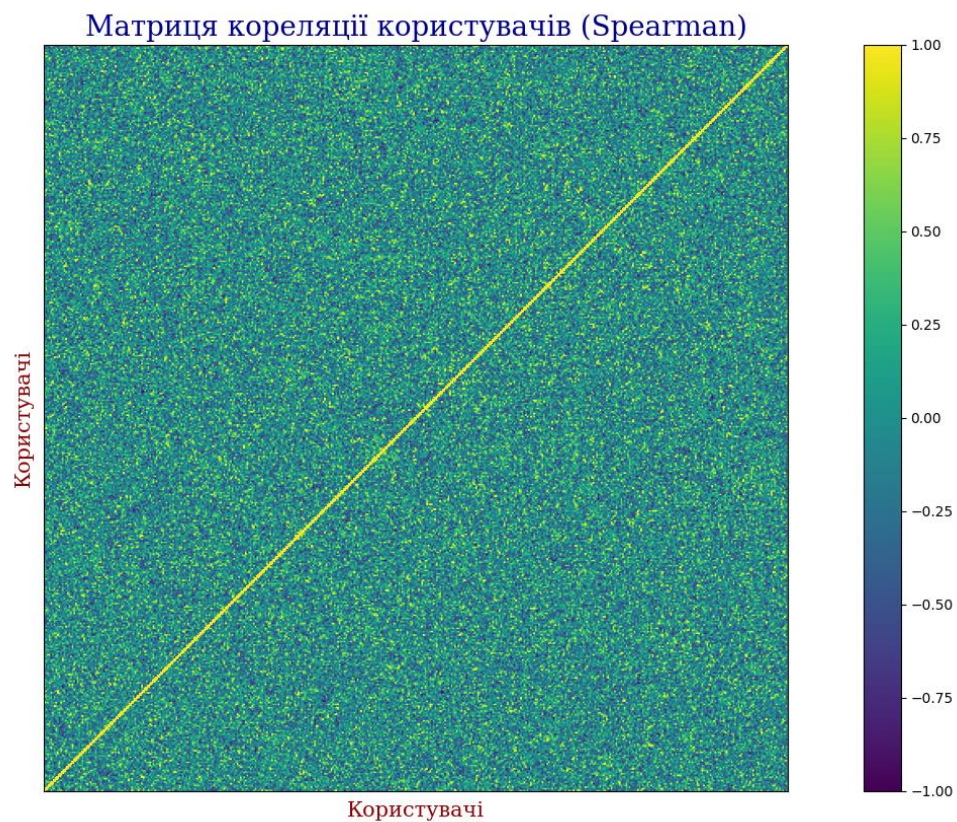


Рисунок 4.5 – Теплова карта матриці кореляції користувачів на основі методу кореляції Спірмена

Результати генерації рекомендацій та тестування цього підходу наведені у таблиці 4.5.

Таблиця 4.5 – Результати реалізованої рекомендаційної системи на основі методу кореляції Спірмена

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0009415440026735555
Час проведення тестування	5.4610 с.
Об'єм матриці кореляції користувачів	1727,1 МБ
Час генерації матриці кореляції користувачів	55.3027 с.
Об'єм матриці з рекомендованими нагородами	10 МБ
Час генерації матриці з рекомендованими нагородами	108.7511 с.

4.3.4 Рекомендації на основі методу кореляції Жаккара

Наступною було реалізовано User-based колаборативну фільтрацію, в якій порівняння схожості користувачів розраховувалося на основі методу кореляції Жаккара.

Значення кореляції, яка розраховується методом Жаккара, може набувати значень від 0 до 1. Чим ближче значення до 1, тим більше один об'єкт схожий на інший. Якщо ж значення дорівнює 1, то це значить, що об'єкти повністю однакові.

Отримана візуалізація матриці кореляції користувачів методом Жаккару (рис. 4.6) дуже сильно відрізняється від попередніх тим, що дивлячись на теплову карту здається, що усі значення дорівнюють 1, тим самим вказуючи що усі користувачі муж собою однакові, але це не так. Насправді ж, у цій матриці зберігаються значення дуже наближені до 1, наприклад, 0.9991873.

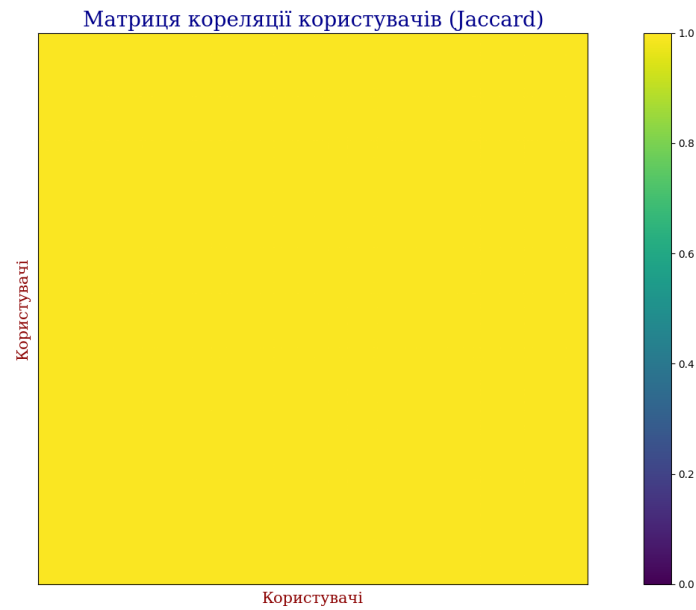


Рисунок 4.6 – Теплова карта матриці кореляції користувачів на основі методу кореляції Жаккара (на основі усього об’єму даних)

Після чого було вирішено спробувати запустити цей алгоритм не на усьому об’ємі даних (приблизно 10 000 користувачів), а лише з даними про 5 користувачів. Згенерована теплова карта з малим об’ємом даних зображена на рисунку 4.7.

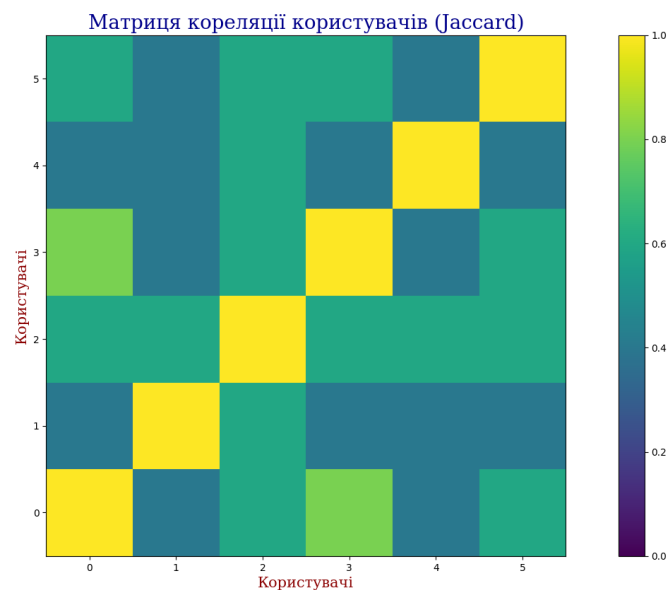


Рисунок 4.7 – Теплова карта матриці кореляції користувачів на основі методу кореляції Жаккара (на основі малого об’єму даних)

Порівнюючи ці дві теплові карти можна зробити висновок, що при генерації матриці кореляції за допомогою методу Жаккара зі збільшенням об'єму даних значення кореляції наближається до 1.

При використанні усього об'єму даних все ж таки зберігається порядок схожості користувачів навіть зі значеннями наближеними до 1.

Результати генерації рекомендацій та тестування цього підходу з використанням усього об'єму даних наведені у таблиці 4.6.

Таблиця 4.6 – Результати реалізованої рекомендаційної системи на основі методу кореляції Жаккара

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0009855101099914378
Час проведення тестування	3.7068 с.
Об'єм матриці кореляції користувачів	1731,4 МБ
Час генерації матриці кореляції користувачів	169.3373 с.
Об'єм матриці з рекомендованими нагородами	5 МБ
Час генерації матриці з рекомендованими нагородами	111.6493 с.

4.3.5 Рекомендації на основі методу косинусу подібності

Наступною було реалізовано User-based колаборативну фільтрацію, в якій порівняння схожості користувачів розраховувалося на основі методу косинусу подібності.

Значення кореляції, яка розраховується методом косинусу подібності, може набувати значень від 0 до 1. Чим ближче значення до 1, тим більше один об'єкт схожий на інший. Якщо ж значення дорівнює 1, то це значить, що об'єкти повністю однакові.

Аналізуючи отриману матрицю кореляції користувачів (рис. 4.8), чітко видно, що діагональ матриці представляє собою суцільну жовту лінію,

що означає те, що користувач повністю схожий сам на себе (значення кореляції дорівнює 1).

Також, можна зауважити, що на тепловій карті переважають кольори, які наближені до значення 0.2 та нижче. Це свідчить про те, що метод косинусу подібності у даному випадку схильний вважати більшість користувачів не схожими один на іншого.

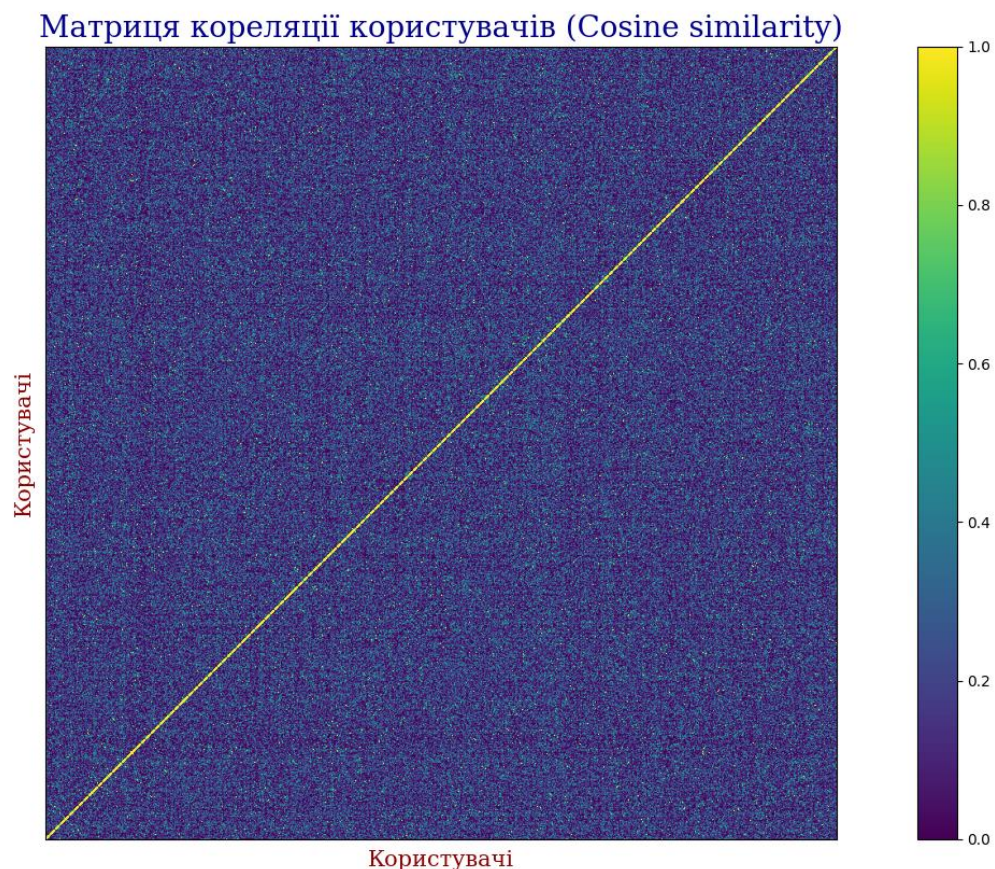


Рисунок 4.8 – Теплова карта матриці кореляції користувачів, яка була побудована за допомогою методу косинусу подібності

Результати генерації рекомендацій та тестування цього підходу наведені у таблиці 4.7.

Таблиця 4.7 – Результати реалізованої рекомендаційної системи на основі методу косинусу подібності

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0012376731041784062
Час проведення тестування	7.5807 с.
Об'єм матриці кореляції користувачів	1729,3 МБ
Час генерації матриці кореляції користувачів	7.7886 с.
Об'єм матриці з рекомендованими нагородами	15 МБ
Час генерації матриці з рекомендованими нагородами	83.5890 с.

4.3.6 Рекомендації на основі методу косинусу подібності та на основі популярності нагород

Частково проаналізувавши результати роботи розроблених рекомендаційних систем було виділено дві реалізації рекомендаційних систем, що показали найкращі результати:

- User-based колаборативна фільтрація на основі методу косинусу подібності з оцінкою точності (MAP) 0.0012376731041784062;
- на основі найпопулярніших нагород з оцінкою точності (MAP) 0.0011042790303970572.

Було вирішено об'єднати ці два підходи у одну систему, яка буде вирішувати проблему «холодного старту», а саме відсутності даних про певних користувачів (рис. 4.9).

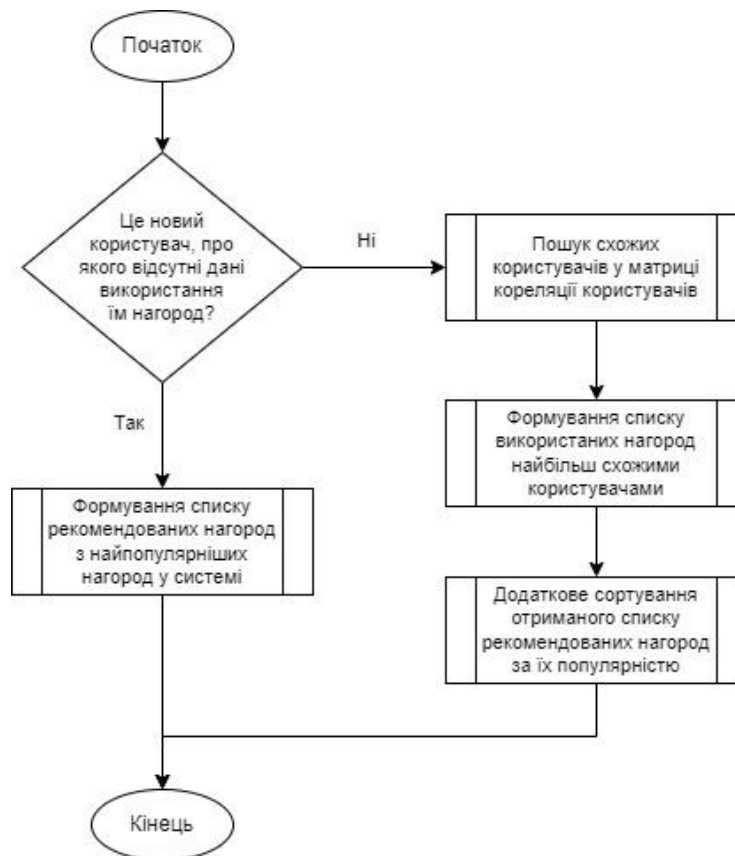


Рисунок 4.9 – Блок схема роботи рекомендаційної системи на основі поєднання двох підходів

Якщо виникне необхідність згенерувати рекомендації для користувача, про якого у системі ще не буде даних щодо використання їм нагород, то в такому випадку будуть згенеровані неперсональні рекомендації на основі популярності нагород.

Якщо ж у системі будуть дані про використання користувачем певних нагород, то для нього будуть знаходитися схожі користувачі за допомогою згенерованої матриці кореляції користувачів (на основі методу косинусу подібності). Після чого буде сформовано список нагород, які використовували найбільш схожі користувачі. Додатковим кроком буде сортування цих нагород за популярністю.

Таким чином, поєднується ефективність двох різних рекомендаційних підходів, а саме User-based колаборативної фільтрації та неперсоналізованих рекомендацій на основі популярності.

Результати генерації рекомендацій та тестування цього підходу наведені у таблиці 4.8.

Таблиця 4.8 – Результати реалізованої рекомендаційної системи на основі методу косинусу подібності та на основі популярності нагород

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0014293314287485821
Час проведення тестування	10.0469 с.
Об'єм матриці кореляції користувачів	1729,3 МБ
Час генерації матриці кореляції користувачів	7.8431 с.
Об'єм матриці з рекомендованими нагородами	16 МБ
Час генерації матриці з рекомендованими нагородами	95.1706 с.

4.3.7 Порівняння результатів

Протестувавши усі зазначені вище рекомендаційні системи було отримано оцінки за формулою MAP, на основі яких було побудовано порівняльний графік зображений на рисунку 4.10.

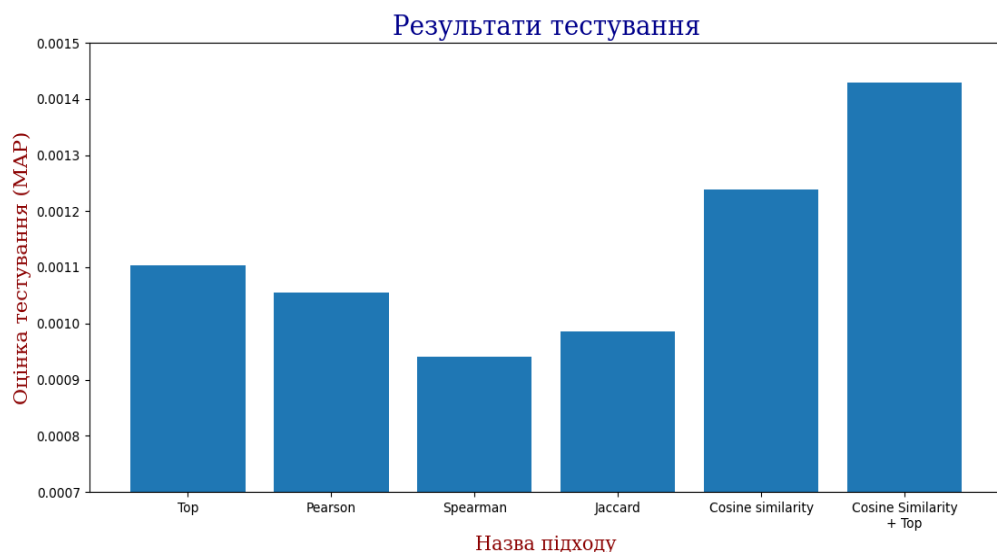


Рисунок 4.10 – Результати тестування розроблених рекомендаційних систем

Виходячи з результатів, можна зробити висновок, що User-based КФ на основі методу Спірмена показала найгірші результати. При цьому неперсоналізовані рекомендації згенеровані на основі популярності нагород показали досить непоганий результат.

Найкращим рішенням виявився підхід, в якому поєднується відразу декілька різних методів, а саме User-based КФ на основі косинусу подібності та неперсоналізовані рекомендації на основі популярності.

4.4 Розробка онлайн частини

Для реалізації онлайн частини було використано веб-фреймворк Django написаний на мові програмування Python. За допомогою цього фреймворку було реалізовано застосунок, який надає можливість відправляти HTTP запити для отримання списку рекомендованих нагород для певного користувача. URL цього ендпоінту має такий вигляд:

`http://localhost:8000/api/user/user_id/recommendations`

Замість `user_id` необхідно вказати унікальний ідентифікатор (ID) певного користувача, для його є необхідність отримати список рекомендованих нагород.

Для того, щоб отримати список рекомендованих нагород для користувача з унікальним ідентифікатором (ID), наприклад 10, необхідно відправити GET запит на наступний URL:

`http://localhost:8000/api/user/10/recommendations`

Сервер прийме цей HTTP запит, обробить його, та поверне у результаті список к унікальними ідентифікаторами (ID) рекомендованих нагород, в якому нагороди будуть відсортовані за релевантністю рекомендації, наприклад такий:

```
{
  "recommended_rewards": [
    55,
```

```

        16,
        22,
        17,
        37
    ]
}

```

Для тестування роботи розробленого ендпоінту було використано програму Postman, яка надає можливість легко конфігурувати та відправляти HTTP запити. Приклад отримання списку з рекомендованими нагородами для конкретного користувача за допомогою розробленого ендпоінту зображений на рисунку 4.11.

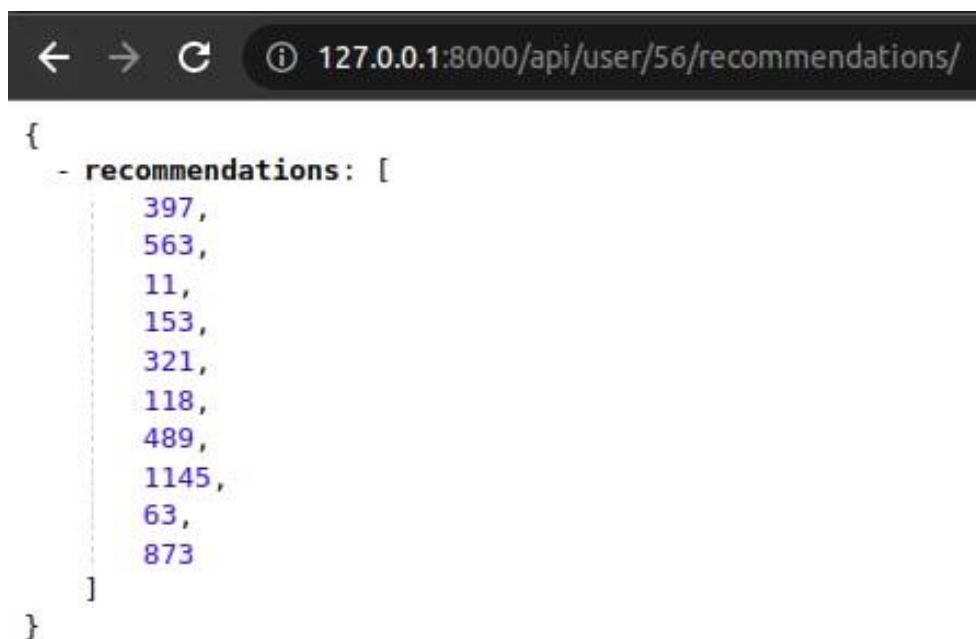


Рисунок 4.11 – Приклад отриманого списку з рекомендованими нагородами для конкретного користувача

4.8 Висновки за розділом

В даному розділі було описано основні рішення, які були прийняті в результаті процесу розробки рекомендаційної системи.

Було описано джерело походження даних, а саме спеціально розроблена програма для генерування даних необхідних для тренування та тестування РС. Додатково було описано підходи, які призначені зменшити обсяг даних та пришвидшення їх обробки.

Наступним кроком було описано процес розробки рекомендаційного модулю, який був реалізований за допомогою різних методів. Було проведено оцінку точності кожного із варіантів, після чого було обрано два найкращих із них та об'єднано задля покращення характеристик.

Також було описано процес розробки примітивного веб-застосунку, який повинен надавати можливість отримувати рекомендовані нагороди за допомогою HTTP запитів.

5 КЕРІВНИЦТВО СИСТЕМНОГО ПРОГРАМІСТА

5.1 Призначення програми та системні вимоги

Призначенням програмного продукту є генерація списку рекомендованих нагород для користувачів програм лояльності. Цей програмний продукт аналізує великий об'єм даних, пов'язаний з використанням користувачами нагород у системі. Надає спеціальний інтерфейс, який дозволяє за допомогою HTTP запиту взаємодіяти з рекомендаційним модулем та отримувати списки рекомендованих нагород.

Характеристики комп'ютера, необхідні для стабільної та швидкої роботи програми, повинні задовольняти вимогам, зазначеним в таблиці 5.1.

Таблиця 5.1 – Мінімальні системні вимоги

Характеристика	Значення
Операційна система	Unix-based (Linux, MacOS)
CPU	Intel або AMD 2-core, 2.6GHz
RAM	6GB
Простір на диску	300MB
Додаткове ПЗ	Інтерпретатор мови програмування Python

5.4 Підготовка до запуску системи

Для запуску усіх компонентів рекомендаційної системи необхідно використовувати інтерпретатор мови програмування Python третьої версії.

Для початку необхідно створити віртуальне оточення, встановити у нього усі необхідні бібліотеки із файлу `pyproject.toml` та активувати його за допомогою трьох команд:

```

pip install poetry
poetry install
poetry shell

```

Після створення та активування віртуального оточення з'являється можливість виконувати усі інші команди, які відносяться до цього проекту.

5.3 Вхідні дані

У якості вхідних даних виступають відомості про використані користувачами нагороди у форматі CSV.

При відсутності справжніх даних є можливість згенерувати «фейкові» дані за допомогою спеціального скрипту. Цей скрипт генерує два файли у тій ж самій теці звідки був запущений скрипт:

- train_data.csv – з даними для тренування рекомендаційного модулю;
- test_data.csv – з даними для тестування рекомендаційного модулю.

Для запуску цього скрипту необхідно виконати наступну команду у терміналі віртуального оточення:

```
python data_generator.py
```

5.4 Запуск рекомендаційного модулю (офлайн частини)

Рекомендаційний модуль представлений шістьма різними файлами, в кожному з яких реалізовано один із методів генерації рекомендацій:

- top.py – на основі популярності нагород;
- pearson.py – на основі User-based КФ з методом кореляції Пірсона;
- spearman.py – на основі User-based КФ з методом кореляції Спірмена;
- jaccard.py – на основі User-based КФ з методом кореляції Жаккара;
- cosine_similarity.py – на основі User-based КФ з методом косинусу подібності;
- cosine_similarity_and_top.py – на основі User-based КФ з методом косинусу подібності та на основі популярності нагород.

Для запуску будь-якого із цих варіантів (файлів) необхідна наявність файлів `train_data.csv` та `test_data.csv` поруч із обраним варіантом (файлом). Команда запуску тренування та тестування рекомендаційного модулю:

```
python cosine_similarity_and_top.py
```

В результаті тренування та тестування буде згенеровано спеціальний pickle-файл з назвою `recommender.obj`, в якому буде міститися усі згенеровані рекомендації.

5.5 Запуск інтерфейсу взаємодії (онлайн частини)

Інтерфейс взаємодії представляє із себе Django веб-застосунок, який приймає та оброблює HTTP запити, після чого генерує та відправляє відповіді на ці запити.

Для запуску веб-застосунку необхідно виконати у терміналі віртуального оточення наступну команду:

```
python manage.py runserver
```

В результаті виконання команди запуснений веб-додаток стане доступний за допомогою наступної адреси:

```
http://localhost:8000
```

Для отримання згенерованого списку рекомендованих нагород для певного користувача необхідно буде відправити HTTP запит будь-яким підходящим інструментом, замінивши `user_id` на ID необхідного користувача:

```
http://localhost:8000/api/user/user_id/recommendations
```

ВИСНОВКИ

В результаті виконання роботи було досліджено та реалізовано найпопулярніші підходи та методи реалізації рекомендаційних систем.

У першому розділі було розглянуто основні поняття теорії рекомендаційних систем та функції, які вони виконують. Було проаналізовано предметну область для якої необхідно було розробити рекомендаційну систему, було розглянуто різноманітні підходи роботи програм лояльності, розглянуто їх переваги та недоліки.

Було поглиблено розглянуто підходи до побудови рекомендаційних систем та найпопулярніші методи розрахунку кореляції, а саме метод Пірсона, Спірмена, Жаккара та косинусу подібності. Було розглянуто метод оцінки точності (Precision) для можливості порівняння релевантності згенерованих рекомендацій різними підходами. Розглянуто різні типи даних, які збирають та використовують рекомендаційні системи. Обрано формат файлу, у якому краще тимчасово зберігати дані, а саме CSV. Розглянуто особливості мови програмування Python та її бібліотек NumPy та Pandas.

Наступним кроком було розроблено архітектуру рекомендаційного застосунку, яка складається з офлайн та онлайн частин.

Розроблено спеціальну програму для генерування даних про використання користувачами нагород через відсутність справжніх даних існуючих програм лояльності.

Було реалізовано та оцінено результати роботи усіх розглянутих методів рекомендаційних системи. За результатами оцінювання було обрано два найкращих метода та поєднано їх у один задля покращення результатів та усунення недоліків цих підходів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке програма лояльності та як її використовувати [Електронний ресурс]. – Режим доступу: <https://tranzzo.ua/blog/chto-takoe-programma-loyalnosti-i-kak-ee-ispolzovat>
2. Ricci, F. Recommender Systems Handbook / F. Ricci, L. Rokach, S. Bracha. – New York: Springer, 2015. – 1003 p.
3. Montaner, M. A taxonomy of recommender agents on the internet / M. Montaner, B. Lopez, J.D.L.R. Esteva. – Netherlands: Artificial Intelligence Review, 2003. – 330 p.
4. Інтелектуальні технології Data Mining і Text Mining [Електронний ресурс]. – Режим доступу: https://pidru4niki.com/1623021247786/informatika/intelektualni_tehnologiyi_data_mining_text_mining
5. Як правильно підібрати програму лояльності [Електронний ресурс]. – Режим доступу: <https://skynum.com/ua/blog/kak-pravilno-podobrat-programmu-loyalnosti-dlya-vashego-magazina>
6. Приклади програм лояльності [Електронний ресурс]. – Режим доступу: <https://incust.com/ua/examples-of-loyalty-programs/>
7. Кореляція [Електронний ресурс]. – Режим доступу: https://economic-definition.com/Exchange_Terminology/Korrelyaciya_Correlation__eto.html
8. Коефіцієнт кореляції Пірсона [Електронний ресурс]. – Режим доступу: <https://studfile.net/preview/9290218/page:7/>
9. Коефіцієнт кореляції Спірмена [Електронний ресурс]. – Режим доступу: <https://studfile.net/preview/5921911/page:14/>
10. Коефіцієнт Жаккара [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Коефіцієнт_Жаккара
11. NumPy documentation [Electronic resource]. – Access mode: <https://numpy.org/doc/stable/>

12. NumPy Introduction [Electronic resource]. – Access mode: https://www.w3schools.com/python/numpy/numpy_intro.asp
13. Pandas documentation [Electronic resource]. – Access mode: <https://pandas.pydata.org/docs/>
14. Pandas [Electronic resource]. – Access mode: <https://uk.wikipedia.org/wiki/Pandas>
15. NumPy vs Pandas [Електронний ресурс]. – Режим доступу: <https://medium.com/@bigdataschool/numpy-vs-pandas-в-чем-разница-между-двумя-библиотеками-python-b85b056e100a>
16. Comparing Python web frameworks [Electronic resource]. – Access mode: <https://www.oreilly.com/library/view/mastering-python-networking/9781789135992/371c6df3-c6ce-4c7e-914d-f6c296d6cbcd.xhtml>
17. Why Django? [Electronic resource]. – Access mode: <https://www.djangoproject.com/start/overview/>
18. Що таке HTTP [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/HTTP>
19. NumPy типи даних [Електронний ресурс]. – Режим доступу: <https://coderlessons.com/tutorials/python-technologies/uchitsia-numpy/numpy-tipy-dannykh>
20. Розріджені матриці [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Розріджена_матриця
21. Візуалізація даних у Python [Електронний ресурс]. – Режим доступу: <https://medium.com/nuances-of-programming/4-простые-визуализации-данных-в-python-с-помощью-кода-ca58253fa1a3>

ДОДАТОК А
Текст програми

A.1 Код файла pyproject.toml

```
[tool.poetry]
name = "Loyalty Recommendation System"
version = "0.1.0"
description = ""
authors = ["denys-marichev <marichevdenysmail@gmail.com>"]

[tool.poetry.dependencies]
python = "^3.9"
pandas = "^1.5.0"
numpy = "^1.23.4"
seaborn = "^0.12.1"
matplotlib = "3.6.0"
PyQt5 = "^5.15.7"
scikit-learn = "^1.1.2"
Jinja2 = "^3.1.2"
scipy = "^1.9.3"
Django = "^4.1.3"

[tool.poetry.dev-dependencies]

[build-system]
requires = ["poetry-core>=1.0.0"]
build-backend = "poetry.core.masonry.api"
```

A.2 Код файла utils.py

```
import numpy as np
from scipy import sparse as sp
import matplotlib.pyplot as plt

def get_sorted_dict_by_values(original_dict: dict) -> dict:
    return {
        k: v for k, v in sorted(original_dict.items(), key=lambda item: item[1], reverse=True)
    }

def get_list_without_duplicates(original_list: list) -> list:
    return list(dict.fromkeys(original_list))

def average_precision(actual, recommended, k=30):
    ap_sum = 0
    hits = 0
    for i in range(k):
        product_id = recommended[i] if i < len(recommended) else None
        if product_id is not None and product_id in actual:
            hits += 1
        ap_sum += hits / (i + 1)
    return ap_sum / k

def normalized_average_precision(actual, recommended, k=30):
    actual = set(actual)
    recommended = list(recommended)
```

```

if len(actual) == 0:
    return 0.0

ap = average_precision(actual, recommended, k=k)
ap_ideal = average_precision(actual, list(actual)[:k], k=k)
return ap / ap_ideal

def draw_correlation_matrix(correlation_matrix, method: str):
    tmp_correlation_matrix = correlation_matrix
    N, M = tmp_correlation_matrix.shape
    s, t = 20, 20 # decimation factors for y and x directions
    T = sp.csc_matrix(
        (np.ones((M,)), np.arange(M), np.r_[np.arange(0, M, t), M]), (M, (M - 1) // t + 1)
    )
    S = sp.csr_matrix(
        (np.ones((N,)), np.arange(N), np.r_[np.arange(0, N, s), N]), ((N - 1) // s + 1, N)
    )
    result = S @ tmp_correlation_matrix @ T # downsample by binning into s x t rectangles

    plt.figure(figsize=(20, 20))

    font1 = {'family': 'serif', 'color': 'darkblue', 'size': 20}
    font2 = {'family': 'serif', 'color': 'darkred', 'size': 15}
    plt.title(f"Матриця кореляції користувачів ({method})", fontdict=font1)
    plt.xlabel("Користувачі", fontdict=font2)
    plt.ylabel("Користувачі", fontdict=font2)

    plt.gca().xaxis.set_major_locator(plt.NullLocator())
    plt.gca().yaxis.set_major_locator(plt.NullLocator())

    plt.imshow(result, cmap='viridis', vmin=-1, vmax=1, origin='lower')
    plt.colorbar()
    plt.show()

```

A.3 Код файлу `timing.py`

```

from functools import wraps
from time import time

def timing(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        ts = time()
        result = f(*args, **kwargs)
        te = time()
        print(f"func: {f.__name__}, time: {te-ts:2.4f}")
        return result
    return wrap

```

A.4 Код файлу `data_generator.py`

```

import random
from pathlib import Path

```

```

unique_users = 10_000
unique_rewards = 2000
max_rewards_per_user = 10
max_reward_uses_per_user = 15

# unique_users = 10_000
# unique_rewards = 3000
# max_rewards_per_user = 50
# max_reward_uses_per_user = 20

# unique_users = 1_000
# unique_rewards = 200
# max_rewards_per_user = 10
# max_reward_uses_per_user = 15

# unique_users = 10
# unique_rewards = 7
# max_rewards_per_user = 4
# max_reward_uses_per_user = 10

test_data_size = 0.07

def main():
    train_data_file = Path.cwd().joinpath("datasets").joinpath("train_data.csv")
    test_data_file = Path.cwd().joinpath("datasets").joinpath("test_data.csv")

    users_and_rewards = {}

    all_rewards = list(range(1, unique_rewards))
    all_users = list(range(1, unique_users))

    for user in all_users:
        rewards_for_this_user = random.randint(0, max_rewards_per_user)
        rewards = random.sample(all_rewards, rewards_for_this_user)
        users_and_rewards[user] = {
            r: random.randint(1, max_reward_uses_per_user)
            for r in rewards
        }

    all_lines = []

    for k, v in users_and_rewards.items():
        for reward, uses in v.items():
            line = f"{k},{reward},{uses}\n"
            all_lines.append(line)

    random.shuffle(all_lines)

    delimiter = int(len(all_lines) * test_data_size)

    with open(train_data_file, "w") as f1:
        f1.write("user_id,reward_id,uses\n")

        for l1 in all_lines[:delimiter]:
            f1.write(l1)

    with open(test_data_file, "w") as f2:
        f2.write("user_id,reward_id,uses\n")

        for l2 in all_lines[delimiter:]:
            f2.write(l2)

```

```

print("Train data len:", len(all_lines) - delimiter)
print("Test data len:", delimiter)
print("Total len:", len(all_lines))

```

```

if __name__ == "__main__":
    main()

```

A.5 Код файла top.py

```

import pickle
from pathlib import Path
from typing import Optional

import pandas as pd
import numpy as np
from timing import timing

from utils import normalized_average_precision

class TopRecommender:
    def __init__(
        self,
        data: pd.DataFrame,
        top_rewards_limit: int = 100,
        remove_already_used_rewards: bool = True
    ):
        self.data = data
        self.remove_already_used_rewards = remove_already_used_rewards
        self.top_rewards_limit = top_rewards_limit
        self.top_rewards = self._get_top_rewards()

    def _get_top_rewards(self) -> pd.DataFrame:
        top_rewards = self.data.groupby('reward_id').count().reset_index()
        top_rewards.columns = ['reward_id', 'count', 'uses']
        top_rewards = top_rewards.sort_values(['count'], ascending=False)
        top_rewards = top_rewards[:self.top_rewards_limit]
        return top_rewards

    def get_user_recommended_rewards(
        self,
        user: int,
        limit: Optional[int] = None
    ) -> pd.Series:
        recommended_rewards = self.top_rewards["reward_id"].reset_index().drop(
            columns="index"
        ).squeeze()

        user_rewards = self.data[self.data["user_id"] == user]["reward_id"]

        if self.remove_already_used_rewards:
            recommended_rewards = np.setdiff1d(recommended_rewards, user_rewards)

        if limit is not None:
            recommended_rewards = recommended_rewards[:limit]

        return pd.Series(recommended_rewards, dtype="uint32")

```

```

@timing
def test(self, test_data: pd.DataFrame, recommendation_per_user_limit: int) -> float:
    users = test_data["user_id"].unique()
    results = []

    for user in users:
        user_rewards = test_data[test_data["user_id"] == user]["reward_id"]
        recommended_rewards = self.get_user_recommended_rewards(
            user=user,
            limit=recommendation_per_user_limit
        )

        results.append(
            normalized_average_precision(user_rewards, recommended_rewards)
        )

    return float(np.mean(results))

def main():
    print("Method: Top")
    print("Loading train data...")
    train_data = pd.read_csv(
        filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("train_data.csv"),
        dtype="uint32"
    )
    print("Train data len:", len(train_data))

    recommender = TopRecommender(data=train_data)

    print("Loading test data...")
    test_data = pd.read_csv(
        filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("test_data.csv"),
        dtype="uint32"
    )
    print("Test data len:", len(test_data))

    print("Testing...")
    result = recommender.test(
        test_data=test_data,
        recommendation_per_user_limit=10
    )
    print("Test result:", result)

    print("Saving pickle file...")
    file_path = Path.cwd().joinpath("datasets").joinpath("recommender.obj")
    with open(Path.cwd().joinpath("datasets").joinpath("model.obj"), "wb") as f:
        pickle.dump(recommender, f, pickle.HIGHEST_PROTOCOL)
    print(f"Pickle file saved ({file_path})")

if __name__ == "__main__":
    main()

```

A.6 Код файла pearson.py

```

import pickle
from pathlib import Path
from typing import Optional

```

```

import pandas as pd
import numpy as np
from scipy import sparse as sp
import sys
from timing import timing

from utils import normalized_average_precision

class PearsonRecommender:
    def __init__(
        self,
        data: pd.DataFrame,
        top_similar_users: int = 5,
        remove_already_used_rewards: bool = True
    ):
        self.top_similar_users = top_similar_users
        self.remove_already_used_rewards = remove_already_used_rewards

        correlation_matrix = self._generate_correlation_matrix(data=data)
        self.recommendations = self._generate_recommendations(
            data=data,
            correlation_matrix=correlation_matrix
        )

    def get_user_recommended_rewards(
        self,
        user: int,
        limit: Optional[int] = None
    ) -> pd.Series:
        recommended_rewards = []

        for reward_index in np.where(self.recommendations.row == user):
            recommended_rewards.extend(self.recommendations.col[reward_index])

        if limit is not None:
            recommended_rewards = recommended_rewards[:limit]

        return pd.Series(recommended_rewards, dtype="uint32")

    @timing
    def _generate_correlation_matrix(self, data: pd.DataFrame) -> pd.DataFrame:
        user_reward_matrix = data.pivot_table(
            index="user_id",
            columns=["reward_id"],
            values="uses",
            fill_value=0
        ).astype("uint8")

        correlation_matrix = pd.DataFrame(
            data=np.corrcoef(user_reward_matrix.values),
            columns=user_reward_matrix.index,
            index=user_reward_matrix.index
        )
        print("Memory usage (correlation_matrix):", sys.getsizeof(correlation_matrix))

        return correlation_matrix

    @timing
    def _generate_recommendations(
        self,
        data: pd.DataFrame,
        correlation_matrix: pd.DataFrame

```

```

) -> sp.coo_matrix:
    all_users = data["user_id"].unique()
    all_rewards = data["reward_id"].unique()

    x_indexes = []
    y_indexes = []

    for user in all_users:
        similar_users = correlation_matrix[user].sort_values(ascending=False)
        similar_users = similar_users.iloc[1:] # Remove the same user
        similar_users = similar_users[similar_users > 0.0] # We leave only similar users
        similar_users = similar_users.iloc[:self.top_similar_users] # We leave the top similar users

        recommended_rewards = data[data["user_id"].isin(
            similar_users.index
        )]["reward_id"].unique()

        user_rewards = data[data["user_id"] == user]["reward_id"].unique()

        if self.remove_already_used_rewards:
            recommended_rewards = np.setdiff1d(recommended_rewards, user_rewards)

        for rw in recommended_rewards:
            x_indexes.append(user)
            y_indexes.append(rw)

    values = [1] * len(x_indexes)

    recommendation_matrix = sp.coo_matrix(
        arg1=(values, (x_indexes, y_indexes)),
        shape=(max(all_users) + 1, max(all_rewards) + 1)
    )

    print("Memory usage (recommendation_matrix):", recommendation_matrix.col.nbytes +
recommendation_matrix.row.nbytes + recommendation_matrix.data.nbytes)

    return recommendation_matrix

@timing
def test(self, test_data: pd.DataFrame, recommendation_per_user_limit: int) -> float:
    users = test_data["user_id"].unique()
    results = []

    for user in users:
        user_rewards = test_data[test_data["user_id"] == user]["reward_id"]
        recommended_rewards = self.get_user_recommended_rewards(
            user=user,
            limit=recommendation_per_user_limit
        )

        results.append(
            normalized_average_precision(user_rewards, recommended_rewards)
        )

    return float(np.mean(results))

def main():
    print("Method: Pearson")
    print("Loading train data...")
    train_data = pd.read_csv(
        filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("train_data.csv"),
        dtype="uint32"
    )

```

```

    )
    print("Train data len:", len(train_data))

    recommender = PearsonRecommender(
        data=train_data,
        top_similar_users=10
    )

    print("Loading test data...")
    test_data = pd.read_csv(
        filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("test_data.csv"),
        dtype="uint32"
    )
    print("Test data len:", len(test_data))

    print("Testing...")
    result = recommender.test(
        test_data=test_data,
        recommendation_per_user_limit=10
    )
    print("Test result:", result)

    print("Saving pickle file...")
    file_path = Path.cwd().joinpath("datasets").joinpath("recommender.obj")
    with open(Path.cwd().joinpath("datasets").joinpath("model.obj"), "wb") as f:
        pickle.dump(recommender, f, pickle.HIGHEST_PROTOCOL)
    print(f"Pickle file saved ({file_path})")

if __name__ == "__main__":
    main()

```

A.7 Код файла spearman.py

```

import pickle
from pathlib import Path
from typing import Optional

import pandas as pd
import numpy as np
from scipy import stats
from scipy import sparse as sp
import sys
from timing import timing

from utils import normalized_average_precision

class SpearmanRecommender:
    def __init__(
        self,
        data: pd.DataFrame,
        top_similar_users: int = 5,
        remove_already_used_rewards: bool = True
    ):
        self.top_similar_users = top_similar_users
        self.remove_already_used_rewards = remove_already_used_rewards

        correlation_matrix = self._generate_correlation_matrix(data=data)
        self.recommendations = self._generate_recommendations(

```

```

        data=data,
        correlation_matrix=correlation_matrix
    )

def get_user_recommended_rewards(
    self,
    user: int,
    limit: Optional[int] = None
) -> pd.Series:
    recommended_rewards = []

    for reward_index in np.where(self.recommendations.row == user):
        recommended_rewards.extend(self.recommendations.col[reward_index])

    if limit is not None:
        recommended_rewards = recommended_rewards[:limit]

    return pd.Series(recommended_rewards, dtype="uint32")

@timing
def _generate_correlation_matrix(self, data: pd.DataFrame) -> pd.DataFrame:
    user_reward_matrix = data.pivot_table(
        index="user_id",
        columns=["reward_id"],
        values="uses",
        fill_value=0
    ).astype("uint8")

    correlation_matrix = pd.DataFrame(
        data=stats.spearmanr(user_reward_matrix.T)[0],
        columns=user_reward_matrix.index,
        index=user_reward_matrix.index
    )
    print("Memory usage (correlation_matrix):", sys.getsizeof(correlation_matrix))

    return correlation_matrix

@timing
def _generate_recommendations(
    self,
    data: pd.DataFrame,
    correlation_matrix: pd.DataFrame
) -> sp.coo_matrix:
    all_users = data["user_id"].unique()
    all_rewards = data["reward_id"].unique()

    x_indexes = []
    y_indexes = []

    for user in all_users:
        similar_users = correlation_matrix[user].sort_values(ascending=False)
        similar_users = similar_users.iloc[1:] # Remove the same user
        similar_users = similar_users[similar_users > 0.0] # We leave only similar users
        similar_users = similar_users.iloc[:self.top_similar_users] # We leave the top similar users

        recommended_rewards = data[data["user_id"].isin(
            similar_users.index
        )]["reward_id"].unique()

        user_rewards = data[data["user_id"] == user]["reward_id"].unique()

        if self.remove_already_used_rewards:
            recommended_rewards = np.setdiff1d(recommended_rewards, user_rewards)

```

```

        for rw in recommended_rewards:
            x_indexes.append(user)
            y_indexes.append(rw)

    values = [1] * len(x_indexes)

    recommendation_matrix = sp.coo_matrix(
        arg1=(values, (x_indexes, y_indexes)),
        shape=(max(all_users) + 1, max(all_rewards) + 1)
    )

    print("Memory usage (recommendation_matrix):", recommendation_matrix.col.nbytes +
recommendation_matrix.row.nbytes + recommendation_matrix.data.nbytes)

    return recommendation_matrix

@timing
def test(self, test_data: pd.DataFrame, recommendation_per_user_limit: int) -> float:
    users = test_data["user_id"].unique()
    results = []

    for user in users:
        user_rewards = test_data[test_data["user_id"] == user]["reward_id"]
        recommended_rewards = self.get_user_recommended_rewards(
            user=user,
            limit=recommendation_per_user_limit
        )

        results.append(
            normalized_average_precision(user_rewards, recommended_rewards)
        )

    return float(np.mean(results))

def main():
    print("Method: Spearman")
    print("Loading train data...")
    train_data = pd.read_csv(
        filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("train_data.csv"),
        dtype="uint32"
    )
    print("Train data len:", len(train_data))

    recommender = SpearmanRecommender(
        data=train_data,
        top_similar_users=10
    )

    print("Loading test data...")
    test_data = pd.read_csv(
        filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("test_data.csv"),
        dtype="uint32"
    )
    print("Test data len:", len(test_data))

    print("Testing...")
    result = recommender.test(
        test_data=test_data,
        recommendation_per_user_limit=10
    )
    print("Test result:", result)

```

```

print("Saving pickle file...")
file_path = Path.cwd().joinpath("datasets").joinpath("recommender.obj")
with open(Path.cwd().joinpath("datasets").joinpath("model.obj"), "wb") as f:
    pickle.dump(recommender, f, pickle.HIGHEST_PROTOCOL)
print(f"Pickle file saved ({file_path})")

if __name__ == "__main__":
    main()

```

A.8 Код файла jaccard.py

```

import pickle
from pathlib import Path
from typing import Optional

import pandas as pd
import numpy as np
from scipy import sparse as sp
import sys

from scipy.spatial.distance import pdist, squareform
from timing import timing

from utils import normalized_average_precision

class JaccardRecommender:
    def __init__(
        self,
        data: pd.DataFrame,
        top_similar_users: int = 5,
        remove_already_used_rewards: bool = True
    ):
        self.top_similar_users = top_similar_users
        self.remove_already_used_rewards = remove_already_used_rewards

        correlation_matrix = self._generate_correlation_matrix(data=data)
        self.recommendations = self._generate_recommendations(
            data=data,
            correlation_matrix=correlation_matrix
        )

    def get_user_recommended_rewards(
        self,
        user: int,
        limit: Optional[int] = None
    ) -> pd.Series:
        recommended_rewards = []

        for reward_index in np.where(self.recommendations.row == user):
            recommended_rewards.extend(self.recommendations.col[reward_index])

        if limit is not None:
            recommended_rewards = recommended_rewards[:limit]

        return pd.Series(recommended_rewards, dtype="uint32")

@timing

```

```

def _generate_correlation_matrix(self, data: pd.DataFrame) -> pd.DataFrame:
    user_reward_matrix = data.pivot_table(
        index="user_id",
        columns=["reward_id"],
        values="uses",
        fill_value=0
    ).astype("uint8")

    correlation_matrix = pd.DataFrame(
        data=1 - squareform(pdist(user_reward_matrix, metric='hamming')),
        columns=user_reward_matrix.index,
        index=user_reward_matrix.index
    )
    print("Memory usage (correlation_matrix):", sys.getsizeof(correlation_matrix))

    return correlation_matrix

@timing
def _generate_recommendations(
    self,
    data: pd.DataFrame,
    correlation_matrix: pd.DataFrame
) -> sp.coo_matrix:
    all_users = data["user_id"].unique()
    all_rewards = data["reward_id"].unique()

    x_indexes = []
    y_indexes = []

    for user in all_users:
        similar_users = correlation_matrix[user].sort_values(ascending=False)
        similar_users = similar_users.iloc[1:] # Remove the same user
        similar_users = similar_users[similar_users > 0.0] # We leave only similar users
        similar_users = similar_users.iloc[:self.top_similar_users] # We leave the top similar users

        recommended_rewards = data[data["user_id"].isin(
            similar_users.index
        )]["reward_id"].unique()

        user_rewards = data[data["user_id"] == user]["reward_id"].unique()

        if self.remove_already_used_rewards:
            recommended_rewards = np.setdiff1d(recommended_rewards, user_rewards)

        for rw in recommended_rewards:
            x_indexes.append(user)
            y_indexes.append(rw)

    values = [1] * len(x_indexes)

    recommendation_matrix = sp.coo_matrix(
        arg1=(values, (x_indexes, y_indexes)),
        shape=(max(all_users) + 1, max(all_rewards) + 1)
    )

    print("Memory usage (recommendation_matrix):", recommendation_matrix.col.nbytes +
recommendation_matrix.row.nbytes + recommendation_matrix.data.nbytes)

    return recommendation_matrix

@timing
def test(self, test_data: pd.DataFrame, recommendation_per_user_limit: int) -> float:
    users = test_data["user_id"].unique()

```

```

results = []

for user in users:
    user_rewards = test_data[test_data["user_id"] == user]["reward_id"]
    recommended_rewards = self.get_user_recommended_rewards(
        user=user,
        limit=recommendation_per_user_limit
    )

    results.append(
        normalized_average_precision(user_rewards, recommended_rewards)
    )

return float(np.mean(results))

def main():
    print("Method: Jaccard")
    print("Loading train data...")
    train_data = pd.read_csv(
        filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("train_data.csv"),
        dtype="uint32"
    )
    print("Train data len:", len(train_data))

    recommender = JaccardRecommender(
        data=train_data,
        top_similar_users=10
    )

    print("Loading test data...")
    test_data = pd.read_csv(
        filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("test_data.csv"),
        dtype="uint32"
    )
    print("Test data len:", len(test_data))

    print("Testing...")
    result = recommender.test(
        test_data=test_data,
        recommendation_per_user_limit=10
    )
    print("Test result:", result)

    print("Saving pickle file...")
    file_path = Path.cwd().joinpath("datasets").joinpath("recommender.obj")
    with open(Path.cwd().joinpath("datasets").joinpath("model.obj"), "wb") as f:
        pickle.dump(recommender, f, pickle.HIGHEST_PROTOCOL)
    print(f"Pickle file saved ({file_path})")

if __name__ == "__main__":
    main()

```

A.9 Код файла cosine_similarity.py

```

import pickle
from pathlib import Path
from typing import Optional

```

```

import pandas as pd
import numpy as np
from scipy import sparse as sp
from sklearn.metrics.pairwise import cosine_similarity
import sys
from timing import timing

from utils import normalized_average_precision

class CosineSimilarityRecommender:
    def __init__(
        self,
        data: pd.DataFrame,
        top_similar_users: int = 5,
        remove_already_used_rewards: bool = True
    ):
        self.top_similar_users = top_similar_users
        self.remove_already_used_rewards = remove_already_used_rewards

        correlation_matrix = self._generate_correlation_matrix(data=data)
        self.recommendations = self._generate_recommendations(
            data=data,
            correlation_matrix=correlation_matrix
        )

    def get_user_recommended_rewards(
        self,
        user: int,
        limit: Optional[int] = None
    ) -> pd.Series:
        recommended_rewards = []

        for reward_index in np.where(self.recommendations.row == user):
            recommended_rewards.extend(self.recommendations.col[reward_index])

        if limit is not None:
            recommended_rewards = recommended_rewards[:limit]

        return pd.Series(recommended_rewards, dtype="uint32")

    @timing
    def _generate_correlation_matrix(self, data: pd.DataFrame) -> pd.DataFrame:
        # data["value"] = np.array(
        #     list(repeat(1, data.shape[0])),
        #     dtype="uint8"
        # )

        user_reward_matrix = data.pivot_table(
            index="user_id",
            columns=["reward_id"],
            values="uses",
            fill_value=0
        ).astype("uint8")

        correlation_matrix = pd.DataFrame(
            data=cosine_similarity(user_reward_matrix),
            columns=user_reward_matrix.index,
            index=user_reward_matrix.index
        )
        print("Memory usage (correlation_matrix):", sys.getsizeof(correlation_matrix))

        return correlation_matrix

```

```

@timing
def _generate_recommendations(
    self,
    data: pd.DataFrame,
    correlation_matrix: pd.DataFrame
) -> sp.coo_matrix:
    all_users = data["user_id"].unique()
    all_rewards = data["reward_id"].unique()

    x_indexes = []
    y_indexes = []

    for user in all_users:
        similar_users = correlation_matrix[user].sort_values(ascending=False)
        similar_users = similar_users.iloc[1:] # Remove the same user
        similar_users = similar_users[similar_users > 0.0] # We leave only similar users
        similar_users = similar_users.iloc[:self.top_similar_users] # We leave the top similar users

        recommended_rewards = data[data["user_id"].isin(
            similar_users.index
        )]["reward_id"].unique()

        user_rewards = data[data["user_id"] == user]["reward_id"].unique()

        if self.remove_already_used_rewards:
            recommended_rewards = np.setdiff1d(recommended_rewards, user_rewards)

        for rw in recommended_rewards:
            x_indexes.append(user)
            y_indexes.append(rw)

    values = [1] * len(x_indexes)

    recommendation_matrix = sp.coo_matrix(
        arg1=(values, (x_indexes, y_indexes)),
        shape=(max(all_users) + 1, max(all_rewards) + 1)
    )

    print("Memory usage (recommendation_matrix):", recommendation_matrix.col.nbytes +
recommendation_matrix.row.nbytes + recommendation_matrix.data.nbytes)

    return recommendation_matrix

@timing
def test(self, test_data: pd.DataFrame, recommendation_per_user_limit: int) -> float:
    users = test_data["user_id"].unique()
    results = []

    for user in users:
        user_rewards = test_data[test_data["user_id"] == user]["reward_id"]
        recommended_rewards = self.get_user_recommended_rewards(
            user=user,
            limit=recommendation_per_user_limit
        )

        results.append(
            normalized_average_precision(user_rewards, recommended_rewards)
        )

    return float(np.mean(results))

```

```

def main():
    print("Method: Cosine similarity")
    print("Loading train data...")
    train_data = pd.read_csv(
        filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("train_data.csv"),
        dtype="uint32"
    )
    print("Train data len:", len(train_data))

    recommender = CosineSimilarityRecommender(
        data=train_data,
        top_similar_users=10
    )

    print("Loading test data...")
    test_data = pd.read_csv(
        filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("test_data.csv"),
        dtype="uint32"
    )
    print("Test data len:", len(test_data))

    print("Testing...")
    result = recommender.test(
        test_data=test_data,
        recommendation_per_user_limit=10
    )
    print("Test result:", result)

    print("Saving pickle file...")
    file_path = Path.cwd().joinpath("datasets").joinpath("recommender.obj")
    with open(Path.cwd().joinpath("datasets").joinpath("model.obj"), "wb") as f:
        pickle.dump(recommender, f, pickle.HIGHEST_PROTOCOL)
    print(f"Pickle file saved ({file_path})")

if __name__ == "__main__":
    main()

```

A.10 Код файла cosine_similarity_and_top.py

```

import pickle
from pathlib import Path
from typing import Optional

import pandas as pd
import numpy as np
from scipy import sparse as sp
from sklearn.metrics.pairwise import cosine_similarity
import sys
from timing import timing

from utils import normalized_average_precision

class CosineSimilarityWithTopRecommender:
    def __init__(
        self,
        data: pd.DataFrame,
        top_similar_users: int = 5,
        remove_already_used_rewards: bool = True
    )

```

```

):
    self.data = data
    self.top_rewards = self._get_top_rewards(data=data)
    self.top_similar_users = top_similar_users
    self.remove_already_used_rewards = remove_already_used_rewards

    correlation_matrix = self._generate_correlation_matrix(data=data)
    self.recommendations = self._generate_recommendations(
        data=data,
        correlation_matrix=correlation_matrix
    )

def _get_top_rewards(self, data: pd.DataFrame) -> np.ndarray:
    return data.groupby("reward_id").count().sort_values(
        'user_id', ascending=False
    ).index.to_numpy()

def get_user_recommended_rewards(
    self,
    user: int,
    limit: Optional[int] = None
) -> pd.Series:
    recommended_rewards = []

    for reward_index in np.where(self.recommendations.row == user):
        recommended_rewards.extend(self.recommendations.col[reward_index])

    recommended_rewards = self.top_rewards[np.isin(self.top_rewards, recommended_rewards)]

    if limit is not None:
        recommended_rewards = recommended_rewards[:limit]

    return pd.Series(recommended_rewards, dtype="uint32")

@timing
def _generate_correlation_matrix(self, data: pd.DataFrame) -> pd.DataFrame:
    # data["value"] = np.array(
    #     list(repeat(1, data.shape[0])),
    #     dtype="uint8"
    # )

    user_reward_matrix = data.pivot_table(
        index="user_id",
        columns=["reward_id"],
        values="uses",
        fill_value=0
    ).astype("uint8")

    correlation_matrix = pd.DataFrame(
        data=cosine_similarity(user_reward_matrix),
        columns=user_reward_matrix.index,
        index=user_reward_matrix.index
    )
    print("Memory usage (correlation_matrix):", sys.getsizeof(correlation_matrix))

    return correlation_matrix

@timing
def _generate_recommendations(
    self,
    data: pd.DataFrame,
    correlation_matrix: pd.DataFrame
) -> sp.coo_matrix:

```

```

all_users = data["user_id"].unique()
all_rewards = data["reward_id"].unique()

x_indexes = []
y_indexes = []

for user in all_users:
    similar_users = correlation_matrix[user].sort_values(ascending=False)
    similar_users = similar_users.iloc[1:] # Remove the same user
    similar_users = similar_users[similar_users > 0.0] # We leave only similar users
    similar_users = similar_users.iloc[:self.top_similar_users] # We leave the top similar users

    recommended_rewards = data[data["user_id"].isin(
        similar_users.index
    )]["reward_id"].unique()

    user_rewards = data[data["user_id"] == user]["reward_id"].unique()

    if self.remove_already_used_rewards:
        recommended_rewards = np.setdiff1d(recommended_rewards, user_rewards)

    for rw in recommended_rewards:
        x_indexes.append(user)
        y_indexes.append(rw)

values = [1] * len(x_indexes)

recommendation_matrix = sp.coo_matrix(
    arg1=(values, (x_indexes, y_indexes)),
    shape=(max(all_users) + 1, max(all_rewards) + 1)
)

print("Memory usage (recommendation_matrix):", recommendation_matrix.col.nbytes +
recommendation_matrix.row.nbytes + recommendation_matrix.data.nbytes)

return recommendation_matrix

@timing
def test(self, test_data: pd.DataFrame, recommendation_per_user_limit: int) -> float:
    users = test_data["user_id"].unique()
    results = []

    for user in users:
        user_rewards = test_data[test_data["user_id"] == user]["reward_id"]
        recommended_rewards = self.get_user_recommended_rewards(
            user=user,
            limit=recommendation_per_user_limit
        )

        results.append(
            normalized_average_precision(user_rewards, recommended_rewards)
        )

    return float(np.mean(results))

def main():
    print("Method: Cosine similarity + Top")
    print("Loading train data...")
    train_data = pd.read_csv(
        filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("train_data.csv"),
        dtype="uint32"
    )

```

```

print("Train data len:", len(train_data))

recommender = CosineSimilarityWithTopRecommender(
    data=train_data,
    top_similar_users=10
)

print("Loading test data...")
test_data = pd.read_csv(
    filepath_or_buffer=Path.cwd().joinpath("datasets").joinpath("test_data.csv"),
    dtype="uint32"
)
print("Test data len:", len(test_data))

print("Testing...")
result = recommender.test(
    test_data=test_data,
    recommendation_per_user_limit=10
)
print("Test result:", result)

print("Saving pickle file...")
file_path = Path.cwd().joinpath("datasets").joinpath("recommender.obj")
with open(Path.cwd().joinpath("datasets").joinpath("model.obj"), "wb") as f:
    pickle.dump(recommender, f, pickle.HIGHEST_PROTOCOL)
print(f"Pickle file saved ({file_path})")

if __name__ == "__main__":
    main()

```

ДОДАТОК Б
Слайди презентації

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ

Дипломна кваліфікаційна робота магістра

**ДОСЛІДЖЕННЯ ТА ПРОГРАМНА
РЕАЛІЗАЦІЯ РЕКОМЕНДАЦІЙНОЇ
СИСТЕМИ ДЛЯ ПРОГРАМ ЛОЯЛЬНОСТІ**

Підготував:
ст. гр. КНТ-121м

Марічев Д.О.

Керівник:
к. т. н., доцент

Колпакова Т.О.

1

Рисунок Б.1 – Слайд №1

Об'єкт дослідження – інформаційна технологія генерації рекомендацій.

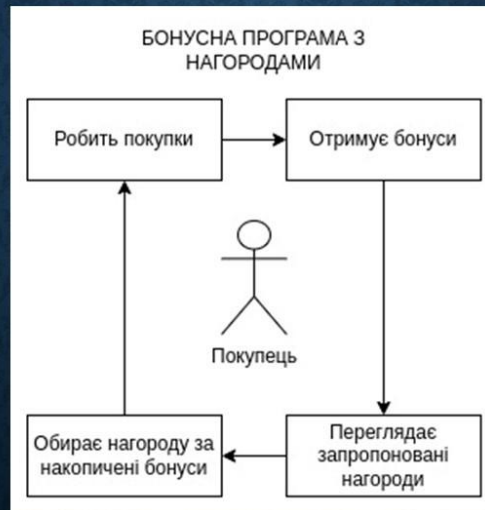
Предмет дослідження – методи та алгоритми формування рекомендацій.

Мета роботи – розробка рекомендаційної системи нагород для програм лояльності.

2

Рисунок Б.2 – Слайд №2

ПРОГРАМИ ЛОЯЛЬНОСТІ



3

Рисунок Б.3 – Слайд №3

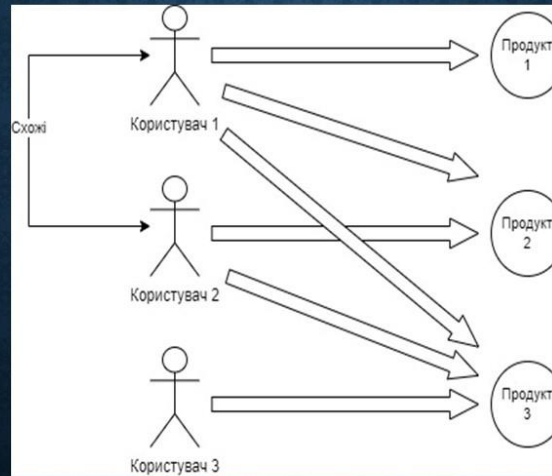
РЕКОМЕНДАЦІЙНА СИСТЕМА



4

Рисунок Б.4 – Слайд №4

USER-BASED КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ



5

Рисунок Б.5 – Слайд №5

ОЦІНКА РОБОТИ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

Precision: $P@N = \frac{1}{N} \times \sum_{i=1}^N \text{Relevance}(i)$

Average Precision: $AP@N = \frac{1}{N} \times \sum_{i=1}^N \text{Relevance}(i) \times P@i$

Mean Average Precision: $MAP@N = \frac{1}{U} \sum_{i=1}^U AP@N_i$

Ранг	Рекомендовані нагороди (ID)	Вгадано нагороду
1	34	Ні
2	56	Так
3	6	Ні
4	12	Так
5	91	Ні

P@3 = 1/3

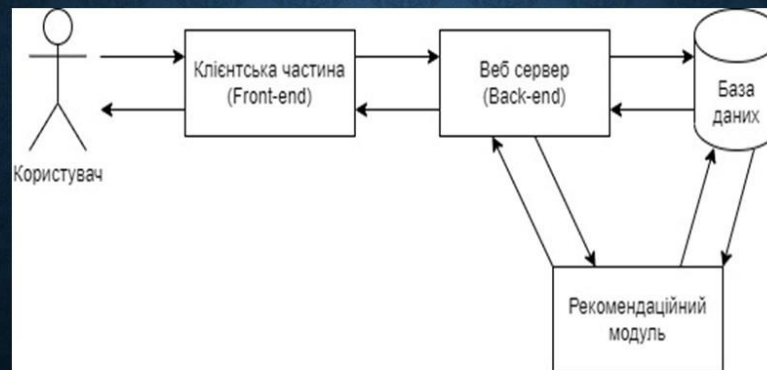
Ранг	Рекомендовані нагороди (ID)	Вгадано нагороду
1	34	Ні
2	56	Так
3	6	Ні
4	12	Так
5	91	Ні

P@4 = 2/4

6

Рисунок Б.6 – Слайд №6

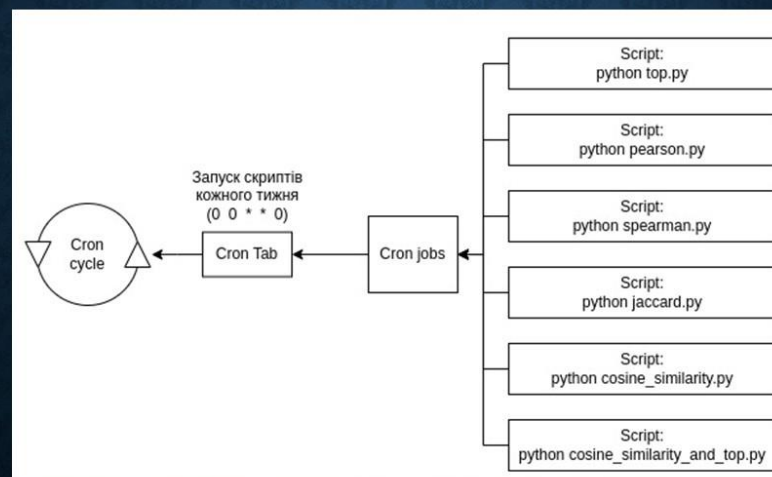
АРХІТЕКТУРА СИСТЕМИ



7

Рисунок Б.7 – Слайд №7

ФУНКЦІОНУВАННЯ РЕКОМЕНДАЦІЙНОГО МОДУЛЮ



8

Рисунок Б.8 – Слайд №8

ДАНІ ДЛЯ РЕКОМЕНДАЦІЙНОГО МОДУЛЮ

user_id,reward_id,uses

1,5,4
2,83,3
2,17,11
3,17,2
4,6,5
4,11,3

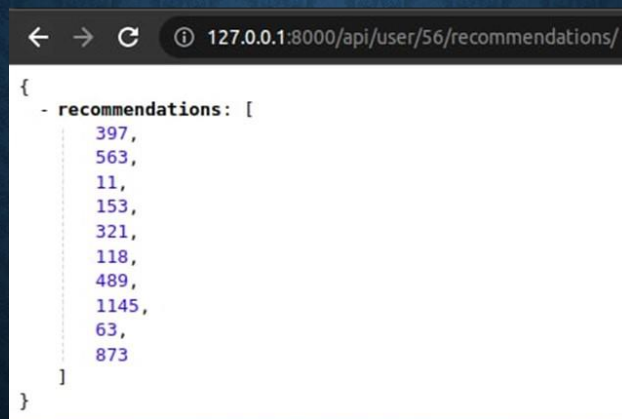
Пояснення колонок з даними:

- **user-id** – унікальний ідентифікатор користувача;
- **reward_id** – унікальний ідентифікатор нагороди;
- **uses** – кількість використань.

9

Рисунок Б.9 – Слайд №9

ФУНКЦІОНУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ



```
{
  - recommendations: [
    397,
    563,
    11,
    153,
    321,
    118,
    489,
    1145,
    63,
    873
  ]
}
```

http://localhost:8000/api/user/<user_id>/recommendations/

10

Рисунок Б.10 – Слайд №10

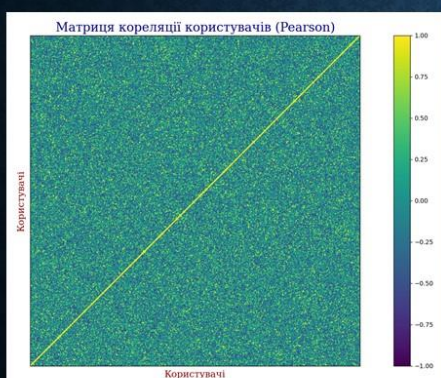
РЕКОМЕНДАЦІЇ НА ОСНОВІ ПОПУЛЯРНOSTІ НАГОРОД

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0011042790303970572
Час проведення тестування	14.0352 с.

11

Рисунок Б.11 – Слайд №11

USER-BASED РЕКОМЕНДАЦІЇ НА ОСНОВІ МЕТОДУ КОРЕЛЯЦІЇ ПІРСОНА



$$r_{ij} = \frac{\sum_{k=1}^N (X_k - \bar{X})(Y_k - \bar{Y})}{\sqrt{\sum_{k=1}^N (X_k - \bar{X})^2 \times \sum_{k=1}^N (Y_k - \bar{Y})^2}}$$

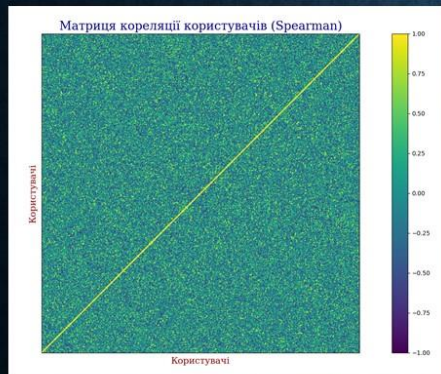
де N – кількість елементів; X_i – елемент з вибірки X з певним індексом; Y_i – елемент з вибірки Y з певним індексом; $\bar{X}$ – середнє значення вибірки X ; $\bar{Y}$ – середнє значення вибірки Y .

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0010544096889509221
Час проведення тестування	7.2215 с.
Об'єм матриці кореляції користувачів	1726,2 МБ
Час генерації матриці кореляції користувачів	8.1992 с.
Об'єм матриці з рекомендованими нагородами	14,1 МБ
Час генерації матриці з рекомендованими нагородами	104.5417 с.

12

Рисунок Б.12 – Слайд №12

USER-BASED РЕКОМЕНДАЦІЇ НА ОСНОВІ МЕТОДУ КОРЕЛЯЦІЇ СПІРМЕНА



$$r_s = 1 - \frac{6 \times \left(\sum_{i=1}^N d_i^2 \right)}{N \times (N^2 - 1)},$$

де N – кількість елементів;

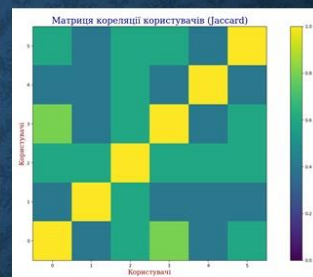
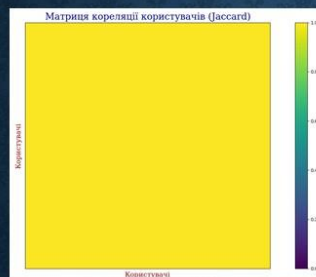
d_i – різниця між рангами елементу з певним індексом.

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0009415440026735555
Час проведення тестування	5.4610 с.
Об'єм матриці кореляції користувачів	1727,1 МБ
Час генерації матриці кореляції користувачів	55.3027 с.
Об'єм матриці з рекомендованими нагородами	10 МБ
Час генерації матриці з рекомендованими нагородами	108.7511 с.

13

Рисунок Б.13 – Слайд №13

USER-BASED РЕКОМЕНДАЦІЇ НА ОСНОВІ МЕТОДУ КОРЕЛЯЦІЇ ЖАККАРА



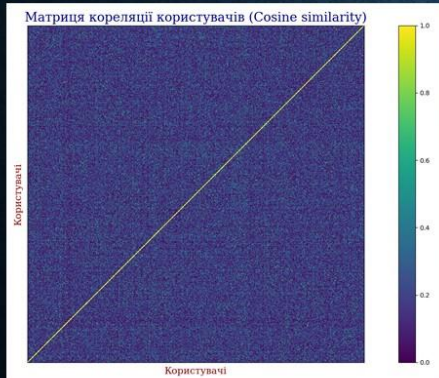
$$J_{XY} = \frac{|X \cap Y|}{|X \cup Y|}$$

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0009855101099914378
Час проведення тестування	3.7068 с.
Об'єм матриці кореляції користувачів	1731,4 МБ
Час генерації матриці кореляції користувачів	169.3373 с.
Об'єм матриці з рекомендованими нагородами	5 МБ
Час генерації матриці з рекомендованими нагородами	111.6493 с.

14

Рисунок Б.14 – Слайд №14

USER-BASED РЕКОМЕНДАЦІЇ НА ОСНОВІ МЕТОДУ КОСИНУСУ ПОДІБНОСТІ



$$\cos_{ij} = \frac{\sum_{k=1}^N X_{ik} \times Y_{jk}}{\sqrt{\sum_{k=1}^N X_{ik}^2} \times \sqrt{\sum_{k=1}^N Y_{jk}^2}},$$

де N – кількість елементів;

X_i – елемент з вибірки X з певним індексом;

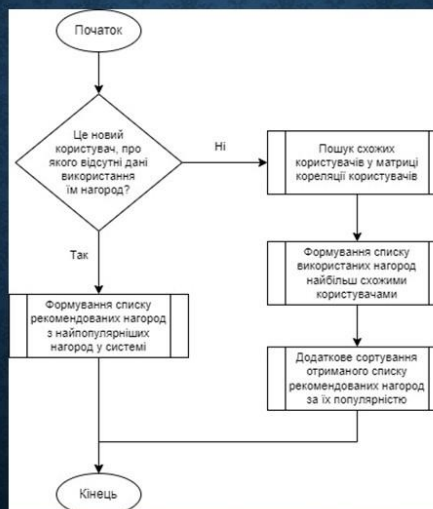
Y_j – елемент з вибірки Y з певним індексом.

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0012376731041784062
Час проведення тестування	7.5807 с.
Об'єм матриці кореляції користувачів	1729,3 МБ
Час генерації матриці кореляції користувачів	7.7886 с.
Об'єм матриці з рекомендованими нагородами	15 МБ
Час генерації матриці з рекомендованими нагородами	83.6890 с.

15

Рисунок Б.15 – Слайд №15

КОМБІНОВАНА РЕКОМЕНДАЦІЙНА СИСТЕМА



16

Рисунок Б.16 – Слайд №16

КОМБІНОВАНА РЕКОМЕНДАЦІЙНА СИСТЕМА

Поєднання двох найкращих методів:

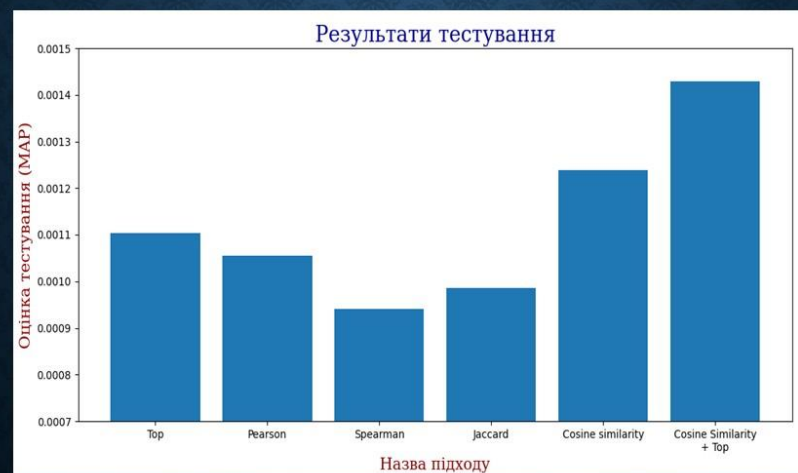
- User-based рекомендацій на основі методу косинусу подібності;
- Неперсоналізованих рекомендацій на основі популярності нагород.

Критерій	Значення
Оцінка точності алгоритму (MAP)	0.0014293314287485821
Час проведення тестування	10.0469 с.
Об'єм матриці кореляції користувачів	1729,3 МБ
Час генерації матриці кореляції користувачів	7.8431 с.
Об'єм матриці з рекомендованими нагородами	16 МБ
Час генерації матриці з рекомендованими нагородами	95.1706 с.

17

Рисунок Б.17 – Слайд №17

ПОРІВНЯННЯ РЕЗУЛЬТАТІВ



18

Рисунок Б.18 – Слайд №18

ВИСНОВКИ

В результаті виконання роботи було досліджено та реалізовано найпопулярніші підходи та методи реалізації рекомендаційних систем.

Було проаналізовано предметну область для якої необхідно було розробити рекомендаційну систему. Розглянуто різноманітні підходи роботи програм лояльності, їх переваги та недоліки.

Розроблено спеціальну програму, для генерування даних про використання користувачами нагород через відсутність справжніх даних існуючих програм лояльності.

Було реалізовано та оцінено результати роботи усіх розглянутих методів рекомендаційних системи. За результатами оцінювання було обрано два найкращих метода та поєднано їх у один задля покращення результатів та усунення недоліків цих підходів.

19

Рисунок Б.19 – Слайд №19