

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«__» _____ 20__ р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Рекомендаційна система з підбору літератури з використанням
методів штучного інтелекту»

Виконала:

студентка IV курсу, групи КІ-02

Ліхачова Анна Андріївна _____

Керівник:

доцент кафедри ШІ, доктор філософії,

Гуськова Віра Геннадіївна _____

Консультант з економічного розділу:

професор, д.е.н., проф. кафедри ЕК ФММ,

Шевчук Олена Анатоліївна _____

Консультант з нормоконтролю:

фахівець першої категорії кафедри ШІ,

Кравець П.В. _____

Рецензент:

д.т.н., професор,

Дмитрієва Ольга Анатоліївна _____

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

Студентка _____

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«31» січня 2024 р.

ЗАВДАННЯ
на дипломну роботу студенту
Ліхачовій Анні Андріївні

1. Тема роботи «Рекомендаційна система з підбору літератури з використанням методів штучного інтелекту», керівник роботи Гуськова Віра Геннадіївна, доцент кафедри ШІ, доктор філософії, затверджені наказом по університету від «31» травня 2024 р. № 2240-С
2. Термін подання студентом роботи «11» червня 2024 року.
3. Вихідні дані до роботи: рекомендації літератури на основі вподобань та попередніх оцінок користувача.
4. Зміст роботи: аналіз предметної області дослідження, огляд та порівняльний аналіз існуючих методів вирішення задачі, вибір методу для реалізації, реалізація системи, аналіз роботи алгоритмів, що реалізовані в системі.
5. Перелік ілюстративного матеріалу: презентація для захисту дипломної роботи.
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук О. А., професор		

7. Дата видачі завдання «05» лютого 2024 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Дослідження предметної області	22.04.2024	Виконано
2	Написання першого розділу роботи	29.04.2024	Виконано
3	Написання другого розділу роботи	06.05.2024	Виконано
4	Розроблення програмного продукту	20.05.2024	Виконано
5	Написання третього розділу роботи	27.05.2024	Виконано
6	Написання четвертого розділу роботи	23.05.2024	Виконано
7	Оформлення записки згідно з положеннями нормоконтролю	10.06.2024	Виконано
8	Підготовка презентації та доповіді	13.06.2024	Виконано

Студент

Анна ЛІХАЧОВА

Керівник

Віра ГУСЬКОВА

РЕФЕРАТ

Дипломна робота: 111 с., 8 рис., 6 табл., 31 посилання, 1 додаток.

РЕКОМЕНДАЦІЙНІ СИСТЕМИ, ГЕНЕТИЧНИЙ АЛГОРИТМ, МАТРИЧНІ АЛГОРИТМИ, ФІЛЬТРАЦІЯ ЗА КОНТЕНТОМ, ПОШУК ЗА НАЙБЛИЖЧИМИ СУСІДАМИ, API.

Об'єкт дослідження – рекомендації літератури на основі вподобань користувачів.

Предмет дослідження – методи штучного інтелекту для створення рекомендаційної системи для літератури.

Мета роботи – програмна реалізація системи, що за допомогою методів штучного інтелекту рекомендує літературу на основі вподобань користувачів.

Результати: реалізовано систему, що складається з таких мікросервісів: сервіс зі збереження даних, сервіс для роботи з зовнішніми джерелами та сервіси з методами штучного інтелекту для вирахування допасованості, проведено аналіз результатів роботи методів штучного інтелекту.

ABSTRACT

Bachelor's thesis: 111 p., 8 figures, 6 tables, 31 references, 1 appendix.

RECOMMENDATION SYSTEMS, GENETIC ALGORITHM, MATRIX ALGORITHMS, CONTENT FILTERING, NEAREST NEIGHBOR SEARCH, API.

The object of the study is literature recommendations based on user preferences.

The subject of research is artificial intelligence methods for creating a literature recommendation system.

The purpose of the work is a software implementation of a system that uses artificial intelligence methods to recommend literature based on user preferences.

The results are a system consisting of the following microservices was implemented: a data storage service, a service for working with external sources, and services with artificial intelligence methods for calculating the fit, and the results of the artificial intelligence methods were analyzed.

ЗМІСТ

ВСТУП	9
РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ ДОДАТКІВ ТА ПЛАТФОРМ.....	10
1.1 Goodreads	10
1.2 StoryGraph	12
1.3 LibraryThing	14
1.4 BookLikes	16
1.5 Litsy.....	18
1.6 BookBub	20
1.7 Libib	22
1.8 Постановка задачі	24
Висновок до розділу 1	25
РОЗДІЛ 2 ПІДХОДИ ДО РЕАЛІЗАЦІЇ.....	26
2.1 Методи збору інформації із зовнішніх джерел.....	26
2.1.1 Публічне API для отримання даних.....	26
2.1.2 Data Scraping.....	27
2.1.3 Data Providers.....	28
2.1.4 Створення власної бази даних	29
2.2 Методи збереження даних	30
2.2.1 Реляційні бази даних	30
2.2.2 Нереляційні бази даних.....	31
2.2.3 Файлове збереження	32
2.2.4 Хмарні сховища.....	32
2.3 Методи штучного інтелекту для вирахування допасованості.....	33
2.3.1 Генетичні алгоритми.....	33
2.3.2 Фільтрація на основі контенту.....	34
2.3.3 Алгоритм k-найближчих сусідів.....	35
2.3.4 Моделі на основі матричного розкладу	37

	7
2.3.4.1 Сингулярний розклад матриці.....	37
2.3.4.2 Метод найменших квадратів	39
Висновок до розділу 2	40
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ВИМОГИ ДО СИСТЕМИ.....	42
3.1 Функціональні вимоги до реалізації	42
3.2 Інструментарій розробки.....	43
3.2.1 Python	43
3.2.2 PyCharm	45
3.2.3 Flask.....	46
3.2.4 Scipy	47
3.2.5 Sklearn	48
3.2.6 Numpy.....	49
3.3.7 Docker.....	50
3.3 Реалізація	51
3.3.1 Сервіс збереження даних	51
3.3.2 Сервіс роботи із зовнішніми джерелами	53
3.3.3 Сервіси з вирахування допасованості.....	55
3.3.3.1 Сервіси на основі матричного розкладу.....	56
3.3.3.1.1 Сервіс на основі SVD-розкладу	57
3.3.3.1.2 Сервіс на основі NMF-розкладу.....	58
3.3.3.1.3 Сервіс на основі PCA-розкладу	60
3.3.3.1.4 Сервіс на основі ALS-розкладу.....	62
3.3.3.2 Сервіс на основі пошуку за найближчими сусідами ..	63
3.3.3.3 Сервіс на основі фільтрації за контентом	65
3.3.3.4 Сервіс на основі генетичних алгоритмів.....	66
3.3.4 Аналіз результатів	67
Висновок до розділу 3	70
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ	72
4.1 Постановка задачі проектування.....	73

	8
4.2 Обґрунтування функцій програмного продукту.....	73
4.3 Обґрунтування системи параметрів програмного продукту	76
4.4 Аналіз експертного оцінювання параметрів	78
4.5 Аналіз рівня якості варіантів реалізації функцій.....	82
4.6 Економічний аналіз варіантів розробки ПП.....	83
4.7 Вибір кращого варіанту ПП техніко-економічного рівня.....	89
Висновок до розділу 4	90
ВИСНОВКИ.....	91
ПЕРЕЛІК ПОСИЛАНЬ.....	93
ДОДАТОК А ЛІСТИНГ ПРОГРАМИ	96

ВСТУП

За даними Google за 2010 рік, у світі існує майже 130 мільйонів книжок, і це число продовжує невпинно зростати. Важливо зазначити, що книжки, видані до середини 1960-х років, не враховані у цьому числі. Перша друкована книга була створена у 1454 році, тобто приблизно 500 років світової літератури залишаються поза статистикою.

Незважаючи на масову цифровізацію, люди все ще активно читають книги. Електронні книги, аудіокниги та класичні паперові варіанти займають місце на наших полицях та в електронних бібліотеках. Але як серед такого розмаїття літератури обрати «ту саму» книгу, що відповідатиме настрою? Раніше для цього ми шукали статті авторів, щоб зрозуміти їхнє натхнення, спілкувалися на форумах, щоб отримати рекомендації від друзів, або просто переглядали бібліотеки у пошуках чогось цікавого. Сьогодні ж ми можемо звернутися до методів штучного інтелекту, які здатні знайти закономірності у, здавалося б, непокєднуваних поняттях. Завдяки цим закономірностям ми можемо легко знайти потрібну книгу.

Наразі існує небагато застосунків, які пропонують рекомендаційні системи для книголюбів. Кожен з них має свої переваги та недоліки. Тому було вирішено створити застосунок, який максимально точно задовольнить потреби користувачів у новинках літератури. У цьому проєкті розглянуто різні методи збору даних, їх збереження, алгоритми підбору та програмні реалізації, щоб створити ефективну рекомендаційну систему для любителів книг.

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ ДОДАТКІВ ТА ПЛАТФОРМ

У світі не так багато застосунків, які можуть нам підказати рекомендації по літературі. Можливості цих застосунків полягають в створенні персональних пропозицій на основі різних факторів та чинників. В залежності від застосунку, дані, на яких базуються ці пропозиції можуть відрізнятися [1].

1.1 Goodreads

Goodreads – це потужна платформа для любителів літератури, що надає можливість організувати своє читання, отримувати рекомендації та взаємодіяти з іншими читачами. Вона функціонує як база даних книг і соціальна мережа одночасно, пропонуючи широкий спектр можливостей для користувачів [2].

Нижче наведено основні можливості Goodreads.

1. Пошук книг. Goodreads має велику базу даних книг, що дозволяє користувачам знаходити книги за назвою, автором або жанром. Ви також можете переглядати списки книг, відсортовані за популярністю або рейтингом.
2. Рекомендації. На основі книг, які ви вже прочитали або додали до свого списку, Goodreads надає персоналізовані рекомендації, які допоможуть вам знайти нові книги, що можуть вам сподобатися.
3. Відгуки та рейтинги. Користувачі можуть залишати відгуки про прочитані книги, а також оцінювати їх за п'ятибальною шкалою. Це допомагає іншим користувачам приймати обґрунтовані рішення щодо читання.

4. Списки літератури. Goodreads дозволяє користувачам створювати та впорядковувати списки книг, які вони хочуть прочитати або вже прочитали. Крім того, платформа допомагає відстежувати прогрес у читанні.
5. Можливості спільноти. Goodreads має активну спільноту, де користувачі можуть обговорювати книги, ділитися відгуками та пропонувати рекомендації. Ви можете приєднатися до груп за інтересами, щоб обговорювати книги з іншими читачами і навіть взаємодіяти з авторами.

Додаток Goodreads інтуїтивно зрозумілий і зручний у користуванні, він складається з наступних розділів:

- домашня сторінка: відображає останні оновлення від друзів, рекомендовані книги, популярні книги, останні дії та статистику читання;
- моя книжкова полиця: персональний каталог книг, які є прочитаними, наявні в списку побажань або читаються зараз, також можна створювати власні полиці для кращої організації;
- спільнота: місце для спілкування з іншими читачами, обговорення книг та участі в групах;
- пошук: дозволяє користувачам шукати книги, авторів, цитати та інший контент за допомогою фільтрів;
- профіль: показує активність користувача, включаючи прочитані книги, написані рецензії, друзів і групи, до яких він приєднався;
- відгуки: користувачі можуть залишати відгуки та оцінювати книги, які відображаються в їхньому профілі та на сторінках книг;
- рекомендації: сервіс надає персоналізовані книжкові рекомендації на основі історії читання.

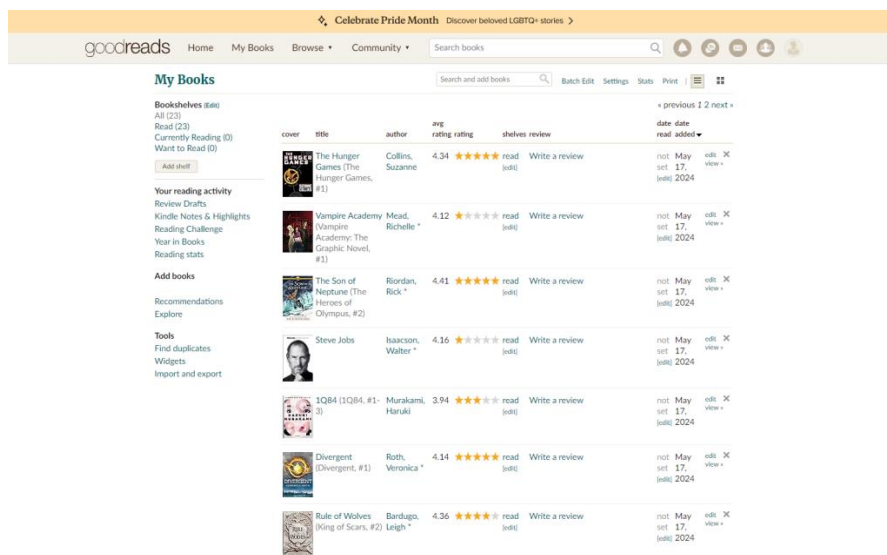


Рисунок 1.1 – Зовнішній вигляд сервісу Goodreads

Основна перевага Goodreads полягає у поєднанні даних про оцінки та характеристики книжок з системою рекомендацій. Хоча оцінка книг є суб'єктивною, часто люди з подібними смаками знаходять корисні рекомендації. Це створює ефективну екосистему-спільноту для книголюбів, де кожен може знайти щось для себе.

1.2 StoryGraph

StoryGraph є відносно новим застосунком для любителів читання, який надає більш персоналізовані рекомендації та іншу систему відслідковування читання, порівняно з Goodreads. Далі наведено основні функції та можливості StoryGraph [3].

1. Пошук книг. StoryGraph дозволяє користувачам шукати книги за допомогою різноманітних фільтрів, таких як настрої, тема, жанр тощо. Ця детальна функція пошуку допоможе вам знайти книги, які ідеально відповідають вашим поточним читацьким уподобанням.

2. Персоналізовані рекомендації. Використовуючи алгоритми машинного навчання, StoryGraph аналізує ваші попередні читацькі звички та відгуки, щоб надати високоточні книжкові рекомендації, адаптовані до ваших смаків.
3. Відстеження читання. Користувачі можуть відстежувати книги, які вони прочитали, хочуть прочитати або читають зараз. Платформа також надає детальну статистику про ваше читання, включаючи кількість прочитаних книг і сторінок, а також жанри і читацькі настрої.
4. Відгуки та рейтинги. Після прочитання книги користувачі можуть залишити свої думки та оцінити книгу. Ці відгуки та оцінки допомагають покращити персоналізовані рекомендації, які надає StoryGraph.
5. Створюйте списки для читання. StoryGraph дозволяє створювати власні списки для читання на основі різноманітних критеріїв і тем, допомагаючи вам ефективно організувати свій план читання.
6. Можливості спільноти. Подібно до Goodreads, StoryGraph дозволяє користувачам взаємодіяти з іншими читачами, приєднуватися до груп, ділитися відгуками та рекомендаціями.

Додаток розроблений так, щоб бути зручним та інтуїтивно зрозумілим, і складається з наступних основних розділів:

- головна сторінка: містить персоналізовану добірку книжкових рекомендацій, статистику читачів та останні оновлення;
- моя книжкова полиця: містить всі додані користувачем книги, включаючи ті, що знаходяться в процесі, прочитані та заплановані до прочитання;
- відгуки читачів: користувачі можуть переглядати власні рецензії та рейтинги книг, а також відгуки інших читачів;
- пошук: дозволяє знаходити книги за назвою, автором, жанром, настроєм тощо;

- рекомендації: надає персоналізовані книжкові пропозиції на основі читацьких вподобань та історії читання.

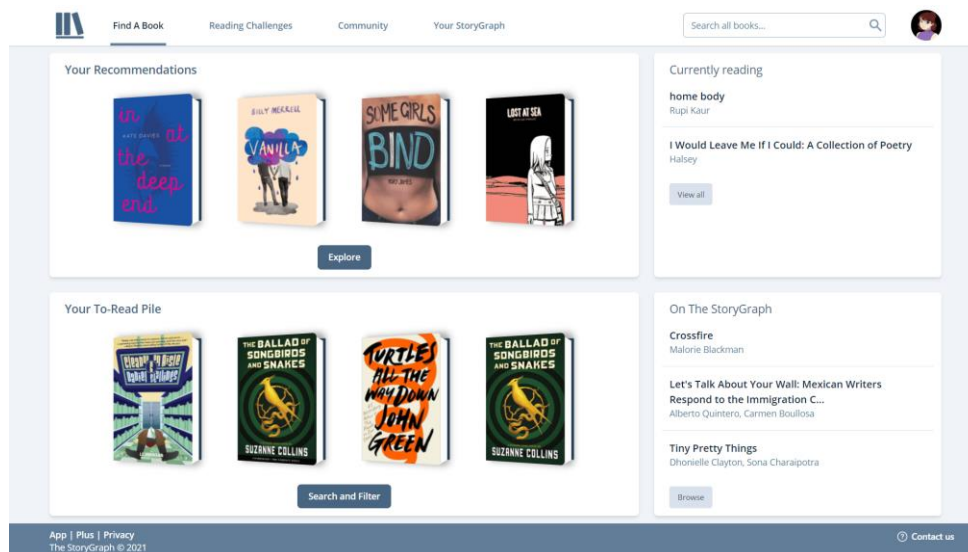


Рисунок 1.2 – Зовнішній вигляд сервісу StoryGraph

StoryGraph створений для читачів, які віддають перевагу більш детальному та особистісному підходу до вибору книг, та тих, хто хоче більш глибокого аналізу своїх читацьких звичок. Його система обліку та відстеження може надати краще розуміння власного читацького досвіду, що робить StoryGraph цікавим інструментом.

1.3 LibraryThing

LibraryThing є соціальним каталогізаційним вебсайтом, призначеним для любителів книг, який дозволяє користувачам зберігати дані про книги, які вони володіють або хочуть прочитати, та вести записи своїх книжкових колекцій. Розглянуто основні можливості LibraryThing [4].

1. Реєстрація книг. Користувачі можуть легко додавати книги до своєї бібліотеки, використовуючи ISBN, назви книг, авторів або навіть скануючи штрих-коди.

2. Тегування. LibraryThing дозволяє користувачам позначати книги тегами, що дає їм змогу класифікувати та організовувати свої колекції за допомогою власних міток.
3. Соціальні елементи. Сайт має потужний соціальний компонент, що включає такі опції, як «Читачі, як ви» та можливість спілкуватися з іншими користувачами через спільноти та групи.
4. Рекомендації. LibraryThing використовує дані своїх користувачів для створення персоналізованих книжкових рекомендацій на основі їхніх інтересів і вже прочитаних книг.
5. Відгуки та рейтинги. Користувачі можуть залишати оцінки та відгуки про книги, допомагаючи іншим вирішити, що читати далі.
6. Діяльність та події. LibraryThing час від часу організовує події, такі як обмін книгами між користувачами, сприяючи розвитку спільноти читачів.

Структура LibraryThing. Платформа складається з наступних розділів:

- головна сторінка: цей розділ містить головні анонси, останні оновлення від друзів та пропозиції щодо читання;
- моя бібліотека: основний розділ, де користувачі зберігають свій каталог книг, який можна впорядкувати за допомогою тегів;
- обговорення: розділ для спілкування з іншими користувачами та участі в дискусіях про книги, жанри чи літературні теми;
- місцеві події: розділ, в якому представлені літературні події, що відбуваються у регіоні, де знаходиться користувач;
- рекомендації: сторінка, на якій користувачі отримують пропозиції щодо книг на основі їхніх інтересів та читацьких уподобань;
- статистика: надає різноманітну статистику про бібліотеку користувача, наприклад, кількість книг за жанрами чи авторами.

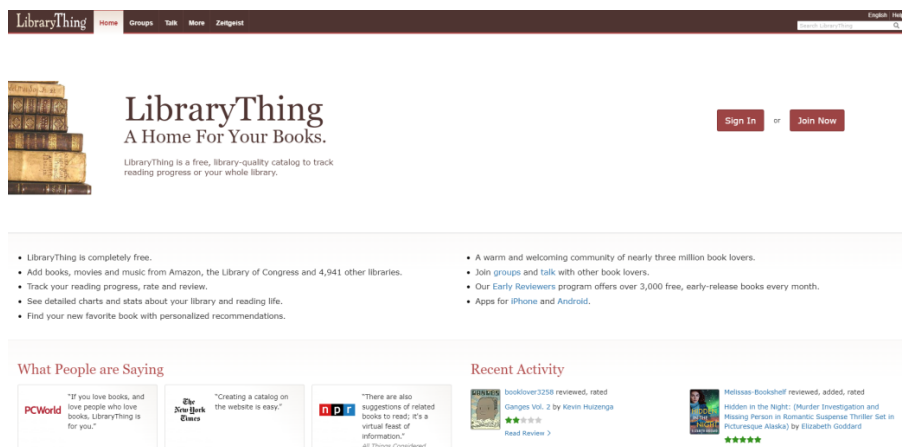


Рисунок 1.3 – Зовнішній вигляд сервісу LibraryThing

LibraryThing створений, щоб покращити досвід читання, допомагаючи організувати книжкову колекцію, відкривати для себе нові книги та взаємодіяти зі спільнотою інших читачів.

1.4 BookLikes

BookLikes – це соціальна мережа для книголюбів, яка поєднує в собі елементи блогу і платформи для ведення каталогу особистої бібліотеки. Це місце, де можна одночасно зберігати та ділитися своїми книжковими перевагами, відгуками і відкривати нові книги через рекомендації та рецензії інших користувачів. BookLikes - це динамічна платформа для книголюбів, що пропонує різноманітні функції для покращення досвіду читання та обміну книжками. Нижче наведено ключові особливості BookLikes [5].

1. Вedenня блогу. Кожен користувач має власний блог на BookLikes, де він може публікувати рецензії, цитати, книжкові акції та будь-що інше, пов'язане з книгами.
2. Книжковий каталог. BookLikes дозволяє користувачам вести детальний каталог книг, включаючи ті, які вони прочитали, хочуть

прочитати або читають зараз. Каталог може бути організований за жанром, автором, рейтингом тощо.

3. Соціальна взаємодія. Користувачі можуть стежити один за одним, відповідати на дописи, репостити контент і коментувати контент, опублікований іншими, сприяючи створенню жвавої спільноти книголюбів.
4. Пошук та рекомендації. BookLikes пропонує систему рекомендацій, засновану на прочитаних вами книгах, ваших інтересах та відгуках з інших джерел, допомагаючи вам знаходити нові книги, які вам сподобаються.
5. Відгуки та оцінки. Користувачі можуть залишати оцінки та відгуки про книги, які впливають на загальний рейтинг книги на платформі.

Структура BookLikes. Платформа складається з наступних розділів:

- головна сторінка: відображає стрічку новин з останніми дописами від користувачів, на яких є підписка, та новинами від BookLikes;
- книжкова полиця: у цьому розділі зберігається інформація про книги, які користувач вже прочитав, читає або планує прочитати, з різними опціями фільтрації та сортування;
- блог: особистий блог, де можна публікувати рецензії, думки та інший контент, пов'язаний з книгами;
- рекомендації: надає автоматичні пропозиції щодо книг, які користувач міг би додати до своєї книжкової полиці, на основі його історії читання та оцінок;
- спільнота: місце, де можна обговорювати книги, спілкуватися з іншими користувачами.

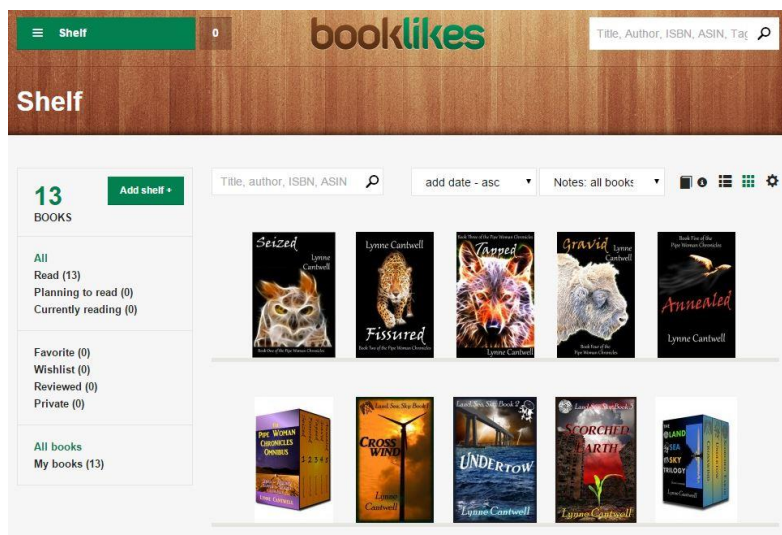


Рисунок 1.4 – Зовнішній вигляд сервісу BookLikes

BookLikes може привабити тих, хто шукає більш персоналізований підхід до спілкування з іншими читачами та хто любить писати власні рецензії та вести блог про книги. Це платформа, яка надає велику свободу висловлювання і в той же час допомагає знаходити нові книги та однодумців.

1.5 Litsy

Litsy – це мобільний застосунок, який поєднує елементи соціальної мережі з функціями каталогізації та обміну інформацією про книги. Ідея Litsy полягає в створенні спільноти, де кожен може поділитися своїми думками про прочитані книги, знайти однодумців, отримати рекомендації та знайомитися з новими творами. Нижче описано основні можливості платформи [6].

1. Реєстрація книг. Користувачі можуть «zareєструватися» в книгах, які вони зараз читають, подібно до реєстрації в реальних місцях на інших соціальних платформах.

2. Дописи та огляди. Litsy дозволяє користувачам ділитися зображеннями, цитатами, рецензіями та іншим контентом, пов'язаним з книгами, безпосередньо зі своїх мобільних пристроїв.
3. Хештеги та теги. Користувачі можуть використовувати хештеги для категоризації своїх дописів, що полегшує навігацію та пошук потрібного контенту.
4. Соціальна взаємодія. Litsy дозволяє користувачам відстежувати інших, вподобати дописи, коментувати та репостити контент, сприяючи формуванню динамічної спільноти читачів.
5. Рекомендації. Платформа пропонує книжкові рекомендації на основі активності та вподобань користувачів у додатку.

Додаток складається з наступних розділів:

- головна сторінка (стрічка): головна стрічка, де користувачі можуть бачити останні пости від людей, на яких вони підписані, а також вибраний контент, класифікований за хештегами;
- профілі книг: кожна книга має власну сторінку профілю, де зібрані всі відгуки, цитати та зображення, якими діляться користувачі;
- пошук: дозволяє користувачам знаходити певні книги, користувачів, хештеги та дописи;
- профіль користувача: відображає особисту інформацію, дописи та список користувачів, на яких підписується користувач і які підписуються на нього.

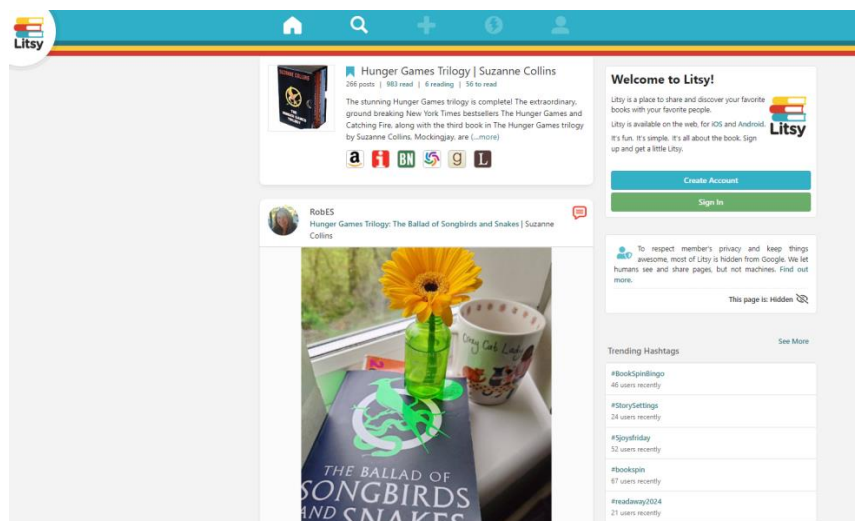


Рисунок 1.5 – Зовнішній вигляд сервісу Litsy

Litsy – платформа, що створена для користувачів, які вважають за краще лаконічні та візуальні способи спілкування, і хочуть поєднувати читачський досвід з елементами соціальної мережі.

1.6 BookBub

BookBub – це сервіс, який пропонує користувачам персоналізовані повідомлення про знижки та пропозиції на електронні книги. Він спрямований на любителів читання, які прагнуть знайти відмінні угоди для книг, що їх цікавлять, та ознайомитися з новими авторами та жанрами. BookBub також працює як маркетинговий інструмент для авторів та видавництва для просування їх книжок. Далі наведено основні можливості та особливості BookBub [7].

1. Сповіщення про знижки на книги. Користувачі можуть підписатися на щоденні електронні розсилки на основі своїх улюблених категорій книг і отримувати сповіщення про розпродажі та акції.

2. Персоналізація. Під час реєстрації користувачі обирають свої улюблені жанри та авторів, що дозволяє їм отримувати більш релевантні пропозиції, адаптовані до їхніх інтересів.
3. Знайомство з новими книгами. BookVub надає рекомендації щодо пошуку нових книг у жанрах та авторів, які вам подобаються.
4. Послуги для авторів та видавців. BookVub пропонує рекламні послуги для авторів та видавців, допомагаючи збільшити продажі книг та впізнаваність серед цільової аудиторії.
5. Безкоштовні книги. Окрім знижок, BookVub іноді надає інформацію про книги, які доступні для безкоштовного завантаження протягом обмеженого часу.

Платформа складається з наступних розділів:

- головна сторінка: користувачі можуть переглянути актуальні пропозиції, нещодавні знижки на популярні книги та запропоновані теми;
- персональні рекомендації: у цьому розділі відображаються книжкові пропозиції, які відповідають уподобанням, обраним користувачами;
- пошук: користувачі можуть шукати книги за назвою або автором, а також за жанром і категорією;
- профіль: дозволяє користувачам створити особистий профіль, де вони можуть змінювати свої вподобання та відстежувати свої завантаження і покупки.

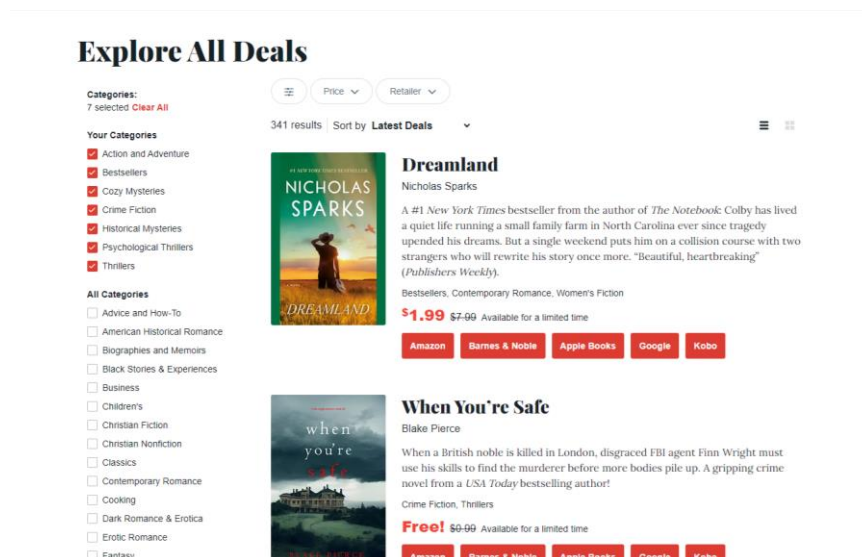


Рисунок 1.6 – Зовнішній вигляд сервісу BookBub

BookBub створений, щоб допомогти читачам знаходити вигідні пропозиції на книги, відкривати для себе нові видання та персоналізувати свій читацький досвід, а також пропонувати авторам і видавцям ефективні інструменти просування.

1.7 Libib

Libib є веб-платформою та мобільною додатком, що надає можливості для каталогізації приватних колекцій книг, фільмів, музики та відеоігор. Цей сервіс орієнтується на індивідуальних користувачів, а також на малі бібліотеки, аматорські збірники та колекціонерів. Далі розглянуто основні можливості Libib [8].

1. Каталогізація. Libib дозволяє користувачам створювати каталоги для книг, фільмів, музики та відеоігор за допомогою смартфона або веб-сайту.

2. Сканування штрих-кодів. Користувачі можуть швидко додавати елементи до своїх колекцій, скануючи штрих-коди, що робить процес каталогізації ефективним і простим.
3. Персональні бібліотеки. Створюйте кілька бібліотек і класифікуйте матеріали за назвою, автором, жанром та іншими критеріями, щоб упорядкувати свої колекції.
4. Соціальна взаємодія. Ви можете зробити вибрані бібліотеки відкритими для друзів або загальнодоступними для спільноти Libib, що полегшить соціальну взаємодію та обмін інформацією.
5. Відгуки та оцінки. Користувачі можуть залишати відгуки і ставити оцінки об'єктам у своїх колекціях, додаючи особистий підхід і допомагаючи іншим відкривати для себе новий контент.
6. Створення списків. Libib дозволяє користувачам створювати різноманітні списки, такі як списки прочитаних книг чи покупок, для подальшої організації та планування своїх колекцій.

Платформа складається з наступних розділів:

- головна сторінка: основний інтерфейс для управління колекціями та пошуку власних рецензій і бібліотек;
- каталогізація: сторінка для додавання нових елементів до бібліотеки з можливістю сканування штрих-кодів та ручного введення;
- мої бібліотеки: розділ для перегляду та керування існуючими бібліотеками;
- списки: розділ, де користувачі можуть створювати списки та керувати ними;
- соціальна взаємодія: сторінка для перегляду бібліотек інших користувачів і взаємодії з ними за допомогою відгуків і оцінок.

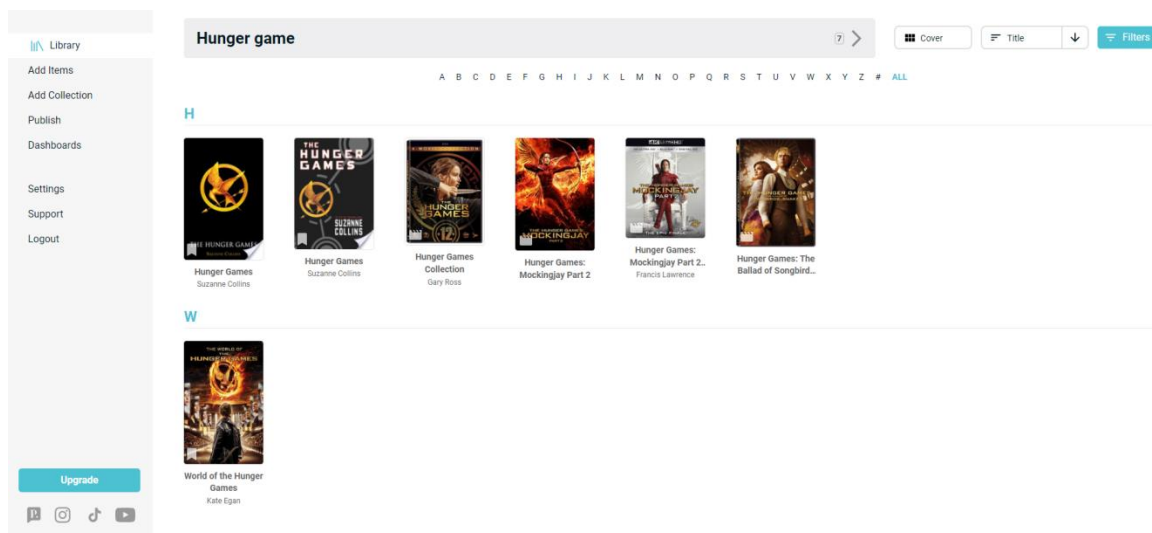


Рисунок 1.7 – Зовнішній вигляд сервісу Libib

Libib ідеально підходить для власників великих колекцій, які хочуть каталогізувати свої книжки та поділитися ними з іншими. Також це зручний інструмент для малих бібліотек чи клубів, що прагнуть легко управляти своїми ресурсами. Libib пропонує базовий безкоштовний тариф, а також платні послуги з додатковими функціями для тих, хто має потребу у розширених можливостях каталогізації та управління.

1.8 Постановка задачі

Вхідними даними для задачі рекомендації літератури є набір оцінок книг, що вже були прочитані користувачем. У рамках роботи необхідно дослідити методи штучного інтелекту, які застосовуються у створенні рекомендацій контенту. Необхідно провести аналіз роботи кожного з методів. Також необхідно провести аналіз інструментів, що застосовуються для створення сервісу зберігання даних та сервісу роботи з даними із зовнішніх джерел.

Висновок до розділу 1

Опрацювавши існуючі платформи та застосунки, такі як Goodreads, StoryGraph, LibraryThing, BookLikes, Litsy, BookBub та Libib, можна зробити висновок, що вони надають значний спектр інструментів для любителів книг. Хоча кожен з додатків має свої унікальні особливості та функціональні можливості, вони всі покликані зробити процес вибору та читання книг більш цікавим та зручним.

Рекомендаційні системи цих платформ базуються на різноманітних даних, включаючи відгуки користувачів, їх читацькі переваги, популярність книг, і навіть сканування штрих-кодів особистих колекцій. Для тих, хто шукає нові книжкові відкриття, пропонується широкий вибір рекомендацій, які спираються на складні алгоритми аналізу даних та машинне навчання.

Додатки і сайти також включають соціальні елементи, що дозволяють користувачам знаходити однодумців і ділитися рецензіями та думками про прочитані твори. Це важливо для формування спільноти книголюбів, де можна обмінюватися думками і знаходити книги, які часто залишаються поза увагою широкої публіки.

Платформи для читачів активно розвиваються, пропонуючи все більш інноваційні технології для задоволення книжкових потреб своїх користувачів. У підсумку можна сказати, що сучасні цифрові інструменти значно розширюють горизонти читання, роблячи процес пошуку та вибору літератури більш особистісним та динамічним.

З урахуванням аналізу існуючих платформ і застосунків, можна зробити висновок, що вони пропонують різноманітні інструменти для любителів книг, але існує необхідність у створенні більш точної та ефективної системи рекомендацій. Тому зосередимося на якості підбору літератури, яка відповідає індивідуальним перевагам користувача.

РОЗДІЛ 2 ПІДХОДИ ДО РЕАЛІЗАЦІЇ

Застосунок для підбору рекомендацій можна розділити на кілька функціональних частин: отримання і оновлення даних про оцінки, збереження даних для подальшої роботи з ними, процес вирахування рекомендацій для користувача

2.1 Методи збору інформації із зовнішніх джерел

Часто перший пункт може бути реалізований по-різному в залежності від платформи. Великі та відомі застосунки вже мають постійних користувачів, тому для них важливо розробити інтерфейс для введення оцінок одразу на сайті. Якщо ж застосунок новий, то необхідно передбачити можливості отримання даних. Тут теж варто розділити на кілька етапів: отримання загальної бази об'єктів, якими ми будемо оперувати та перенесення даних користувача з популярних платформ.

2.1.1 Публічне API для отримання даних

Платформа Goodreads зупинила видавання своїх ключів з 2020 року. Проте залишила можливість переносити свої дані за допомогою csv (comma-separated values) файлів [9]. Попри це, є Google Books API [11] та Book Search API (від openlibrary) [10]. Відповідно ми можемо не створювати свою базу даних, а використовувати get запити для отримання наших даних. Серед переваг такого підходу:

- легкий доступ: зручний спосіб отримати великий обсяг інформації швидко і легко;
- актуальність даних: дані часто оновлюються, забезпечуючи актуальну інформацію;
- широкий спектр даних: доступ до детальної інформації про книги, включаючи обкладинки, рецензії, оцінки тощо.

Серед недоліків:

- обмеження доступу: деякі API мають обмежений доступ або вимагають реєстрацію і затвердження (наприклад, Goodreads);
- залежність від сервісу: зміни в API або його закриття можуть суттєво вплинути на вашу базу даних;
- ліміти запитів: багато API мають обмеження на кількість запитів, що може стати проблемою для масштабованості.

2.1.2 Data Scraping

Data Scraping (або Web Scraping) - це процес автоматичного збору інформації з веб-сайтів. За допомогою спеціальних програм або скриптів, дані витягуються з HTML-коду веб-сторінок і перетворюються у структурований формат, який можна використовувати для різних цілей, таких як аналіз, збереження в базі даних або подальша обробка [13]. Нижче наведено етапи збору інформації.

1. Збір HTML-коду: скрипт або програма запитує веб-сторінку і отримує її HTML-код.
2. Парсинг HTML: HTML-код аналізується і потрібні дані витягуються за допомогою різних методів, таких як регулярні вирази або спеціалізовані бібліотеки.

3. Збереження даних: витягнуті дані зберігаються у структурованому форматі, наприклад, у файлі CSV, базі даних або в іншому зручному форматі.

Переваги:

- гнучкість: можна отримати дані з будь-якого доступного веб-сайту;
- повний контроль над даними: можливість витягувати саме ті дані, які потрібні.

Недоліки:

- юридичні ризики: порушення умов використання веб-сайтів або авторських прав може мати серйозні наслідки;
- технічна складність: вимагає написання і підтримки скриптів для парсингу, що може бути складним і затратним.

2.1.3 Data Providers

Існують організації або компанії, які збирають, зберігають та надають доступ до великих обсягів даних. Ці дані можуть охоплювати широкий спектр інформації, включаючи метадані книг, фінансові дані, аналітику ринку, демографічну інформацію тощо. Постачальники даних часто пропонують свої послуги на платній основі, забезпечуючи надійність і точність наданої інформації [12].

Переваги:

- надійність і якість даних: професійні провайдери гарантують високу якість і точність даних;
- підтримка і документація: зазвичай постачальники даних надають хорошу підтримку та документацію.

Недоліки:

- вартість: доступ до даних часто є платним, що може бути дорого для невеликих проєктів;
- залежність: проєкт буде залежати від постачальника даних.

2.1.4 Створення власної бази даних

Не варто забувати про варіант з створенням власної бази [14].

Переваги:

- повний контроль над структурою і наповненням бази даних;
- можливість створювати унікальну базу даних, що може бути конкурентною перевагою.

Недоліки:

- створення і підтримка власної бази даних потребує значних ресурсів;
- збирання даних з нуля може бути довгим процесом, що уповільнить процес запуску продукту.

Для прискорення процесу можна використати краудсорсинг, але тоді дані багато втрачають на точності, хоч і заповнення бібліотеки прискорюється в рази, порівняно з заповненням вручну.

2.2 Методи збереження даних

Важливим аспектом також є методи збереження даних. Цей крок дуже відповідальний з точки зору розробки, адже подальша архітектура напряду впливатиме на швидкодію застосунку та потенційну вартість використання.

2.2.1 Реляційні бази даних

Реляційні бази даних використовують таблиці для збереження даних. Відносини між таблицями встановлюються за допомогою ключів. Найпопулярніші системи управління реляційними базами даних (СУРБД) включають MySQL, PostgreSQL, SQLite. Наприклад MySQL має відкритий вихідний код, широко використовуваний в веб-розробці. PostgreSQL – потужна, об'єктно-реляційна СУРБД з розширеними можливостями. А SQLite – це легка СУРБД, яка зберігає дані у одному файлі, підходить для невеликих проєктів або локального використання.

Переваги:

- структурованість: чітка схема даних з чіткими зв'язками між таблицями;
- мова запитів SQL: стандартна мова для роботи з даними;
- цілісність даних: забезпечується за допомогою ключів та обмежень.

Недоліки:

- складність масштабування: вертикальне масштабування є складним і дорогим;
- жорстка схема: зміни в схемі можуть вимагати значних зусиль.

2.2.2 Нереляційні бази даних

NoSQL бази даних пропонують гнучкіші підходи до зберігання даних без жорсткої схеми. Вони підходять для обробки великих обсягів даних і часто використовуються для розподілених систем.

Серед найбільш використовуваних:

- MongoDB: документоорієнтована база даних, яка зберігає дані в форматі JSON-подібних документів [15];
- Cassandra: колонкоорієнтована база даних, розроблена для обробки великих обсягів даних у розподіленій системі [16];
- Redis: база даних ключ-значення, яка використовується для швидкого доступу до даних [17].

Переваги використання нереляційних баз даних:

- масштабованість: легше горизонтально масштабувати;
- гнучкість: можливість зберігати дані без жорсткої схеми;
- висока продуктивність: підходять для швидкого зчитування і запису даних.

Недоліки:

- відсутність стандарту: кожен тип NoSQL бази даних має свої специфічні особливості;
- цілісність даних: відсутність жорстких обмежень може призвести до проблем з цілісністю даних.

2.2.3 Файлове збереження

Дані зберігаються у файлах різних форматів, таких як CSV, JSON, XML. Це простий метод збереження даних, який підходить для невеликих проєктів або для обміну даними між системами.

Переваги:

- простота використання: легко створювати, читати та обробляти;
- сумісність: широко підтримуються різними мовами програмування і додатками;
- гнучкість: можна легко змінювати структуру даних.

Недоліки:

- обмежена масштабованість: підходить для невеликих обсягів даних;
- цілісність даних: немає вбудованих механізмів забезпечення цілісності даних;
- продуктивність: повільніше при роботі з великими обсягами даних.

2.2.4 Хмарні сховища

Використання хмарних сервісів для зберігання даних. Хмарні сховища надають доступ до даних через інтернет і часто мають вбудовані інструменти для обробки та аналізу даних. Приклади:

- Amazon RDS: керована реляційна база даних на платформі AWS [18];
- Google Cloud Firestore: NoSQL база даних з автоматичною синхронізацією і масштабуванням [19];

- Microsoft Azure Cosmos DB: глобально розподілена база даних з підтримкою кількох моделей даних [20].

Переваги:

- масштабованість: легко масштабуються в залежності від потреб;
- доступність: доступ з будь-якої точки світу через інтернет;
- безпека: вбудовані засоби безпеки і резервного копіювання.

Недоліки:

- вартість: може бути дорогою для великих обсягів даних і тривалого використання;
- залежність від провайдера: дані зберігаються на серверах провайдера, що може створювати ризики;
- затримка: може бути затримка доступу до даних через мережу.

2.3 Методи штучного інтелекту для вирахування допасованості

Головною частиною подібної системи є безпосередньо алгоритм підбору. В рамках даної роботи розглядаємо алгоритми, які не передбачають великої кількості даних і орієнтуються на рекомендації виключно в рамках уподобань одного користувача. Розглянемо кілька з них.

2.3.1 Генетичні алгоритми

Ідея використання генетичних алгоритмів полягає в тому, щоб імітувати процес еволюції, де кожна книга або набір книг вважається "індивідом", а їх характеристики - "генами" [21].

Нижче показано як цей процес виглядає у рамках системи рекомендацій книжок.

1. Ініціалізація популяції: Спочатку створюється популяція випадково вибраних книг або списки книг.
2. Функція пристосування: Пристосування (фітнес) кожної книги оцінюється на основі попередніх переваг користувача - наприклад, жанри, які вони зазвичай читають, автори, які їм подобаються, рейтинги книг тощо.
3. Відбір: Вибираються книги з найвищим рівнем пристосування, які потім служитимуть "батьками" для наступного покоління рекомендацій.
4. Кросовер (схрещування): "Гени" від декількох "батьків" комбінуються для створення "дітей" - це можуть бути нові списки книг, які включають атрибути обох "батьків".
5. Мутації: Щоб запобігти локальній оптимальності та збільшити різноманітність, обрані книги можуть піддаватися мутаціям - змінам, що базуються на випадковому виборі.
6. Нове покоління: Популяція оновлюється шляхом заміни старого покоління новими "дітьми".
7. Фіналізація: Процес повторюється протягом багатьох поколінь до тих пір, поки не буде знайдено оптимальний набір рекомендацій або не буде досягнуто призначену кількість ітерацій.

2.3.2 Фільтрація на основі контенту

Фільтрація на основі контенту (content-based filtering) - це техніка будування системи рекомендацій, яка використовує детальні інформації про елемент, щоб рекомендувати додаткові елементи зі схожими властивостями. У

контексті рекомендацій книжок, ця методика враховує характеристики самої книги, такі як жанр, автор, тема, опис, та схожі фактори [22].

Процес фільтрації на основі контенту для системи рекомендації книжок включатиме кроки, що наведені нижче.

1. Екстракція характеристик: Збір властивостей книг, які можна оцифрувати, таких як текст опису, метадані (наприклад, автор, жанр, видалення), рецензії тощо.
2. Створення профілю елемента: Кожній книзі присвоюються певні ключові слова чи концепції, що утворюють її "профіль".
3. Створення профілю користувача: Аналіз переваг користувача, історії прочитаних книг, оцінок та відгуків для побудови уявлення про його схильності.
4. Вимір схожості: Використання методів обрахунку схожості (наприклад, косинусної схожості) для порівняння профілю користувача з профілем книги.
5. Рекомендація: Рекомендування книг, чиї профілі найбільш схожі на профіль користувача.
6. Повторне навчання: Система продовжує навчатися і вдосконалюватися з кожною новою взаємодією користувача, щоб підтримувати актуальність та точність рекомендацій.

2.3.3 Алгоритм k -найближчих сусідів

Алгоритм k -найближчих сусідів (k -nearest neighbors, k -NN) – це тип ленивого навчання, який класифікує об'єкти на основі найближчих образів у просторі ознак. У контексті рекомендаційної системи для книг, k -NN можна використовувати для знаходження книг, які є схожими на ті, що користувач уже оцінив високо [23].

Алгоритм k-NN.

1. Визначення простору ознак: Кожна книга в датасеті представлена набором характеристик, таких як жанр, автор, рік випуску тощо. Ці характеристики використовуються для визначення 'позиції' книги у багатовимірному просторі ознак.
2. Нормалізація даних: Оскільки різні ознаки можуть мати різні масштаби, всі дані, як правило, нормалізуються для забезпечення рівних відстаней.
3. Розрахунок відстані: Використовуючи метрику відстані (наприклад, Евклідову або Манхеттенську), алгоритм вимірює дистанцію між книгами в просторі ознак.
4. Знаходження k-найближчих сусідів: Для даної книги або профілю користувача алгоритм ідентифікує k найближчих сусідів, тобто книг, які є найближчими за відстанню в просторі ознак.
5. Агрегація результатів: Опираючись на оцінки чи інші характеристики k найближчих сусідів, вираховується агреговане значення, яке може служити оцінкою або рекомендацією.
6. Рекомендація: Система надає користувачу рекомендації на базі виявлених k сусідів, зазвичай тих книжок, які найближчі до переваг користувача.

При впровадженні k-NN у системи рекомендації книг важливо визначити оптимальне значення k, яке б забезпечило найкращі результати. Це може залежати від розміру датасету, розподілу даних та конкретних цілей рекомендацій.

2.3.4 Моделі на основі матричного розкладу

Моделі на основі матричного розкладу (Matrix Factorization Models) є однією з найбільш популярних і ефективних методик у створенні рекомендаційних систем, особливо для проблеми з багатьма користувачами і елементами, таких як користувачі та фільми в Netflix або користувачі та книги в системах рекомендації книжок.

Матричний розклад працює шляхом апроксимації початкової великої матриці (наприклад, матриці "користувачі * елементи", де кожна комірка відображає оцінку, дану користувачем елементу) за допомогою добутку двох або більше менших матриць. Це виявляє латентні (приховані) фактори, які впливають на вибір користувача, такі як інтереси, жанрові переваги, контекст тощо [24].

2.3.4.1 Сингулярний розклад матриці

Сингулярний розклад матриці (Singular Value Decomposition, SVD) є часто застосовуваною технікою у математиці та обчислювальній науці, зокрема в моделях на основі матричного розкладу для рекомендаційних систем. SVD дозволяє апроксимувати складну матрицю через три простіші матриці, що мають специфічне математичне значення [24].

Для матриці рейтингів R (де рядки представляють користувачів, а стовпці - елементи, такі як книги), SVD розкладає R на три матриці:

- U (матриця лівих сингулярних векторів): відображає користувачів у латентний простір ознак;

- Σ (діагональна матриця сингулярних значень): містить сингулярні значення, що представляють вагу кожного з латентних ознак у відношенні до апроксимації даних;
- V^T (матриця правих сингулярних векторів, транспонована): проєкція елементів (книг) на латентний простір ознак.

Матриця R апроксимується як:

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

Застосування SVD у рекомендаційних системах:

- прогнозування рейтингів: SVD може допомогти у прогнозуванні відсутніх рейтингів у матриці користувачів та книг, використовуваних у рекомендаційних системах;
- зниження розмірності: SVD ефективно зменшує розмірність даних, видаляючи менш важливі ознаки, що може поліпшити час виконання та зменшити перенавчання.

Проблеми при використанні SVD у рекомендаційних системах:

- відсутні дані: у реальних сценаріях матриця рейтингів часто є розрідженою, що означає, що в ній багато незаповнених комірок, традиційний SVD не може бути застосований безпосередньо до розріджених матриць;
- часова складність: безпосередній розклад великих матриць може бути часомістким процесом.

Адаптації SVD для рекомендаційних систем:

- FunkSVD або SVD++: модифікації SVD, які адаптовані для роботи з розрідженими матрицями і використовуються в компаніях, таких як Netflix;
- альтернативний найменших квадратів (ALS): ще одна стратегія, яка часто застосовується для розкладу розріджених матриць.

2.3.4.2 Метод найменших квадратів

Метод найменших квадратів (Least Squares Method) є стандартним підходом у регресійному аналізі, в якому сума квадратів розбіжностей між спостережуваними та прогнозованими даними мінімізується. У контексті моделей на основі матричного розкладу, особливо у рекомендаційних системах, метод найменших квадратів використовується для знаходження латентних факторів, які найкраще відповідають спостережуваним рейтингам користувачів [24].

Розглянемо розріджену матрицю рейтингів R , де користувачі представлені рядками, а елементи (наприклад, книги) - стовпцями. Матричний розклад прагне знайти дві невідомі матриці нижчої рангу - матрицю користувачів U та матрицю елементів V , таким чином, що їхній добуток апроксимує R :

$$R \approx UV^T.$$

Функція втрати, яку мінімізує метод найменших квадратів, має такий вигляд:

$$L(U, V) = ||R - UV^T||^2,$$

де $|| \cdot ||$ вказує на норму Фробеніуса, яка вимірює розмір відмінностей між спостережуваною та апроксимованою матрицею.

Таким чином, мінімізація $L(U, V)$ веде до пошуку U та V , що найкраще "підходять" до вихідної матриці R .

Один з підходів до мінімізації $L(U, V)$ - це алгоритм альтернативних найменших квадратів (ALS). Робота ALS описана нижче.

1. Начальна ініціалізація матриць U та V з випадковими значеннями.
2. Фіксація однієї з матриць (наприклад, U) і оптимізація другої (V) за методом найменших квадратів.
3. Фіксація V і оптимізація U наступним кроком за методом найменших квадратів.
4. Повторення кроків 2 і 3 до збіжності алгоритму.

ALS має перевагу в обробці великих та розріджених матриць, що робить його підходящим для рекомендаційних систем.

Висновок до розділу 2

У розділі було розглянуто множину методологій, які застосовуються у створенні рекомендаційних систем для ефективного вибору відповідних рішень. Приділено увагу методам збору інформації, зокрема через публічні API, Data Scraping, використання послуг Data Providers та створення власної бази даних. Важливість аналізу кожного методу у контексті конкретного застосування підкреслюється їхніми сильними та слабкими сторонами, на які треба зважати, вибираючи оптимальний підхід для конкретної системи.

Також у розділі обговорювалися стратегії щодо збереження та організації даних, що включають використання реляційних та нереляційних баз даних, файлове збереження та хмарні сховища. Кожен із цих підходів має свої переваги, такі як структурованість та цілісність даних для реляційних баз, а також масштабованість та гнучкість для нереляційних баз і хмарних рішень. Ретельний вибір та правильна імплементація відповідної системи є ключовим для підтримки ефективності та швидкодії майбутньої рекомендаційної системи.

Крім того, велика увага приділялася методам штучного інтелекту для вирахування рекомендацій, зокрема, були розглянуті генетичні алгоритми, фільтрація на основі контенту, алгоритми k-найближчих сусідів та моделі на основі матричного розкладу. Методи вибору рекомендацій є вирішальними для персоналізації користувацького досвіду та підвищення задоволення користувачів, і при їхньому виборі слід враховувати специфіку даних, якими оперує система.

Моделі на основі матричного розкладу виявились одними з найбільш ефективних методів для роботи з великими обсягами даних. Такі моделі, як SVD і ALS, уможливлюють виявлення латентних факторів, які можуть впливати на вибір та вподобання користувачів.

Відповідно вибір правильних методологій для збору даних, їхнього збереження, обробки та аналізу є вирішальним для створення ефективної рекомендаційної системи. Потрібно взяти до уваги всі передумови та можливі обмеження, щоб створити систему, яка буде не лише функціональною та ефективною, а й максимально відповідає потребам користувачів.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ВИМОГИ ДО СИСТЕМИ

Сервіси рекомендацій книжок покликані допомогти користувачам знаходити літературу, що відповідає їхнім інтересам та потребам. Вони не тільки спрощують процес вибору, але й сприяють відкриттю нових авторів та жанрів, які могли залишитися непоміченими без таких рекомендацій.

Створення ефективного сервісу рекомендацій книжок вимагає ретельного підходу до реалізації функціональних вимог та використання потужного інструментарію для процесу розробки, починаючи від визначення вимог та закінчуючи аналізом результатів.

3.1 Функціональні вимоги до реалізації

Функціональні вимоги визначають основні можливості та характеристики сервісу рекомендацій книжок. Їх можна поділити на мікросервіси. Мікросервіси ж можна категоризувати за галузями використання:

- сервіси для роботи з даними;
- сервіси для роботи із запитами до сторонніх сервісів;
- сервіси для аналізу допасованості.

Відповідно потрібно описати основні вимоги до таких сервісів, для більш чіткого розуміння подальшої взаємодії та необхідних інструментів реалізації.

Сервіси для роботи з даними. Сервіс повинен працювати з наданими даними від користувача. Серед таких наданих даних можуть бути: власноруч введені книжки, яких можливо немає в бібліотеці за використаним API; назви та автори книжок, які користувач хоче додати, відповідно сервіс має надати необхідні дані на інші сервіси; парсинг даних від інших популярних застосунків

для перевикористання вже введеного користувачем функціоналу; передача наявних даних в сервіси для аналізу допасованості. Сервіс для роботи із запитами до сторонніх сервісів. Такий сервіс має реалізовувати наступні потреби користувача: запити до бази даних з API щоб знайти книги, введені користувачем і зібрати необхідну інформацію про них; запити до бази даних з API для пошуку відповідних рекомендацій за тегами, жанрами, авторами або будь-якою іншою релевантною інформацією для пошуку, щоб передати рекомендації в сервіс для роботи з даними, а звідти відповідно в сервіси для аналізу допасованості. Сервіси для аналізу допасованості. В ідеалі, мати один такий сервіс, який повністю задовільняє потреби користувача і надає «ідеальні» рекомендації. Проте це неможливо. Тому необхідно надати комплексну інформацію з аналізу допасованості. Відповідно для цього створити сервіси, які аналізують дані за різними критеріями, алгоритмами та підходами і відповідно дати можливість користувачу обирати серед найбільш релевантних підходів на його думку.

3.2 Інструментарій розробки

Для успішної розробки системи важливо обрати ефективні та перевірені інструменти. Низка інструментів будуть використані для створення та підтримки якісної роботи системи.

3.2.1 Python

Python - це інтерпретована об'єктно-орієнтована мова програмування загального призначення. Розроблена Гвідо ван Россумом у 1991 році, вона відзначається простим і легкозрозумілим синтаксисом, сприяючи швидкому

створенню програм. Популярність Python зумовлена його простотою, ефективністю та широким спектром використання [25].

Основні риси Python:

- простий синтаксис: має синтаксис, подібний до англійської мови, що полегшує розуміння коду, також відступи для визначення блоків коду роблять код більш легким для читання;
- інтерпретованість: Python використовується як інтерпретована мова, що дозволяє швидко експериментувати та розробляти програми без додаткових кроків компіляції;
- розширюваність: Python забезпечує широкий спектр модулів і бібліотек, що розширюють його функціональність, що дозволяє повторно використовувати код, що в свою чергу пришвидшує розробку;
- багатofункціональність: Python має ряд вбудованих бібліотек, які пропонують готові рішення для різних завдань, спрощуючи процес написання програмного забезпечення;
- крос-платформеність: Python підтримується на різних операційних системах, що робить його універсальним для використання на різних платформах;
- активна спільнота: Python має велику кількість учасників спільноти розробників, яка надає підтримку та розвиває мову, через що мова поширена в різних галузях та областях.

Python широко використовується в різних сферах, включаючи розробку веб-додатків, науково-дослідну діяльність, розробку штучного інтелекту, аналіз та автоматизацію задач, розробку ігор тощо. Простота, ефективність та широкі функціональні можливості роблять Python однією з найпопулярніших мов програмування.

3.2.2 PyCharm

PyCharm – це інтегроване середовище розробки (IDE) для Python, створене компанією JetBrains. Воно надає широкий спектр інструментів та функцій, спеціально розроблених для підвищення продуктивності у розробці програм на Python [26].

Ключовими функціями є:

- редагування коду: інтелектуальний редактор коду PyCharm спрощує написання коду завдяки таким функціям, як автозавершення, підсвічування синтаксису та безпечний рефакторинг коду;
- налагодження: спрощуючи налагодження коду Python, PyCharm дозволяє встановлювати точки зупинки, перевіряти змінні, крокувати по коду та розуміти ієрархію викликів функцій;
- управління проєктами: PyCharm полегшує завдання управління проєктами, такі як створення, додавання файлів, керування залежностями та налаштування середовища виконання;
- інтеграція з керуванням версіями: інтегрований з Git, SVN та Mercurial, PyCharm дозволяє безперешкодно використовувати системи контролю версій безпосередньо у своєму інтерфейсі;
- підтримка розширень: завдяки широкому спектру плагінів і розширень PyCharm дозволяє збільшити можливості для роботи з різними фреймворками, бібліотеками та інструментами;
- інструменти тестування: PyCharm має вбудовану підтримку тестування коду, що спрощує процес створення, виконання та аналізу результатів тестування прямо в IDE.

JetBrains надає безкоштовну підписку для студентів, щоб допомогти їм вивчити мову програмування. Завдяки своїм потужним можливостям, зручному

інтерфейсу та підтримці великої спільноти розробників, PyCharm є одним з найпопулярніших середовищ розробки для Python.

3.2.3 Flask

Flask – це легкий веб-фреймворк для Python, який дозволяє швидко та ефективно розробляти веб-додатки. Він відомий своєю простотою та гнучкістю, що робить його одним з найпопулярніших інструментів для веб-розробки на Python [27].

Основні особливості Flask:

- простота у використанні: Flask може похвалитися простим синтаксисом, що спрощує розробку веб-додатків, він пропонує фундаментальний набір функцій, дозволяючи при цьому легко розширювати їх у разі потреби;
- спрощена структура: на відміну від фреймворків з жорсткою структурою проекту, Flask дозволяє розробникам організовувати свій код відповідно до особистих уподобань;
- розширюваність: завдяки великій кількості розширень, Flask полегшує інтеграцію додаткової функціональності у фреймворк, включаючи підтримку баз даних, автентифікації, форм, API тощо;
- вбудований сервер розробки: Flask має вбудований сервер розробки для швидкого тестування веб-додатків, що усуває необхідність у налаштуванні окремого сервера;
- підтримка шаблонів: Flask підтримує шаблонні движки, такі як Jinja2, що полегшує розробку веб-сторінок, відокремлюючи логіку від представлення;

- сумісність з RESTful: ідеально підходить для розробки RESTful API, адже Flask спрощує створення маршрутів, обробку HTTP-запитів і передачу даних у таких форматах, як JSON.

Фреймворк Flask є надзвичайно гнучким та простим для використання під час створення проєктів, який надає розробникам широкий спектр можливостей для реалізації веб-додатків на Python.

3.2.4 Scipy

SciPy – це відкрите програмне забезпечення для наукових обчислень та технічного розрахунку у Python, яке надає широкий набір функцій та інструментів для розв'язання різноманітних завдань у галузі науки, інженерії та аналізу даних [28].

Основні концепції та можливості SciPy включають:

- наукові інструменти: SciPy вміщує в себе різноманітні функції та модулі для виконання наукових обчислень, включаючи оптимізацію, чисельні методи, обробку сигналів, обробку зображень, статистику та інші;
- оптимізація: SciPy надає інструменти для пошуку мінімумів та максимумів функцій, оптимізації параметрів та розв'язання різних задач оптимізації;
- чисельні методи: Бібліотека містить численні методи для розв'язання диференціальних рівнянь, інтегрування функцій, роботи з матрицями та інші алгоритми чисельного аналізу;
- обробка сигналів та зображень: SciPy пропонує інструменти для обробки сигналів, включаючи фільтрацію, кореляцію та

спектральний аналіз, а також для роботи із зображеннями, включаючи обробку, аналіз та візуалізацію;

- статистика: бібліотека містить функції для виконання різних статистичних обчислень, таких як обчислення середнього, медіани, дисперсії, кореляції та інших.

SciPy дозволяє вченим, інженерам та аналітикам використовувати потужні інструменти для вирішення складних завдань у своїх дослідженнях та розробках, роблячи процес наукових обчислень в Python більш доступним і ефективним.

3.2.5 Sklearn

Scikit-learn, часто називаний Sklearn, є відкритою бібліотекою машинного навчання для мови програмування Python, яка надає широкий спектр алгоритмів та інструментів для класифікації, регресії, кластеризації, вимірювання важливості ознак та багато іншого [29].

Основні можливості scikit-learn включають:

- реалізація алгоритмів машинного навчання: scikit-learn містить реалізації різних алгоритмів машинного навчання, включаючи методи найближчих сусідів, дерева рішень, випадкові ліси, наївний байєсівський класифікатор, метод опорних векторів та багато інших;
- інструменти попередньої обробки даних: scikit-learn надає інструменти для попередньої обробки даних, такі як масштабування ознак, видалення відсутніх значень, кодування категоріальних змінних та витягування ознак;
- вбудовані набори даних: бібліотека містить деякі стандартні набори даних для навчання та тестування моделей машинного навчання;

- оцінка та валідація моделей: `scikit-learn` надає інструменти для оцінки та валідації моделей, включаючи розрахунок метрик ефективності, хрестову перевірку та підбір параметрів моделей;
- інтеграція з іншими бібліотеками: `scikit-learn` легко інтегрується з іншими бібліотеками Python, такими як `NumPy`, `SciPy` та `pandas`, що дозволяє легко використовувати їх разом для аналізу даних та побудови моделей машинного навчання.

`Scikit-learn` є потужним інструментом для вивчення та застосування методів машинного навчання в Python, допомагаючи вирішувати різноманітні завдання у сферах науки, бізнесу та досліджень.

3.2.6 *Numpy*

`NumPy` – це відкрита бібліотека для мови програмування Python, яка додає підтримку для великих масивів та математичних функцій високого рівня для роботи з цими масивами. Вона є однією з основних бібліотек у екосистемі Python для наукових обчислень та обробки даних [30].

Основні можливості `NumPy` включають:

- масиви: об'єкт `NumPy ndarray` - це потужний N-вимірний масив для ефективних операцій з однорідними даними, що виключає необхідність використання циклів;
- векторизовані операції: `NumPy` дозволяє виконувати математичні операції на масивах без необхідності використання циклів, що робить код більш ефективним та зрозумілим;
- математичні функції: Бібліотека містить великий набір математичних функцій, таких як обчислення синусу, косинусу,

експоненти, логарифму та інших, які можна застосовувати до масивів;

- індексація та зрізи: NumPy дозволяє виконувати операції індексації та зрізів над масивами, що дозволяє вибирати та маніпулювати даними у масивах швидко та ефективно;
- інтеграція з іншими бібліотеками: NumPy легко інтегрується з іншими бібліотеками Python, такими як SciPy, pandas та matplotlib, що дозволяє використовувати їх разом для вирішення різноманітних завдань у наукових обчисленнях та візуалізації даних.

NumPy є потужним інструментом для виконання операцій з масивами та виконання математичних обчислень у Python, що робить його важливою складовою для наукових, інженерних та аналітичних завдань.

3.3.7 Docker

Docker – це платформа з відкритим вихідним кодом, яка спрощує пакування, розгортання та керування додатками. Це досягається завдяки використанню віртуальних контейнерів [31].

Ось принцип роботи Docker.

1. Контейнери: автономні "пакунки", що містять програму та всі її залежності. Ці контейнери легкі та ізольовані, що дозволяє їм працювати на різних операційних системах без конфліктів. Це забезпечує узгодженість середовища для ваших додатків, незалежно від конфігурації базового сервера.
2. Docker-файл: текстовий файл, який визначає кроки, включаючи вибір базового образу, встановлення залежностей, копіювання

необхідних файлів і налаштування середовища для створення контейнера. Докер-файли автоматизують процес створення контейнера, гарантуючи узгодженість і відтворюваність.

3. **Образи:** виконувані пакети, що містять все необхідне для запуску програми, включаючи операційну систему, залежності та саму програму. Можна завантажити образи із загальнодоступних репозиторіїв, таких як Docker Hub, або створити свій власний за допомогою Docker-файлу.

Для зручності користувачів Docker Desktop пропонує офіційну версію для Windows і macOS. Вона надає зручний інтерфейс для встановлення та керування Docker на вашому комп'ютері. Він включає в себе ядро Docker Engine, інтерфейс командного рядка (CLI) та інші необхідні компоненти. Docker Desktop дозволяє створювати, запускати та керувати контейнерами безпосередньо на робочому столі за допомогою CLI або графічного інтерфейсу користувача (GUI).

3.3 Реалізація

Програма працює за ендпоінтами, кожен з яких є реалізацією частини алгоритму чи інтерфейсу відповідно.

3.3.1 Сервіс збереження даних

Цей сервіс надає можливість користувачам зберігати та керувати своєю персональною бібліотекою книг. Користувачі можуть додавати нові книги,

вносити зміни до існуючих записів, видаляти книги, а також отримувати рекомендації на основі їх вибору та попередніх оцінок.

Кроки реалізації.

1. Створення бази даних для збереження інформації про книги користувачів.
2. Реалізація API для взаємодії з даними:
3. додавання нових книг;
4. редагування існуючих записів;
5. видалення книг;
6. отримання списку книг користувача.

Підтримка інтеграції з іншими сервісами:

- парсинг даних з інших популярних джерел для автоматичного додавання книг;
- можливість передачі даних в сервіси аналізу для отримання рекомендацій.

Автентифікація користувачів та забезпечення конфіденційності даних.

Вхідні дані:

- дані про книги користувачів, такі як назва, автор, жанр, рейтинг і т.д.;
- запити користувачів на додавання, редагування або видалення книг;
- дані, отримані в результаті парсингу або інтеграції з іншими джерелами.

Вихідні дані:

- підтвердження успішної операції збереження, редагування або видалення книги;
- список книг користувача, включаючи інформацію про кожну книгу;
- рекомендації книг на основі вибору та оцінок користувача.

Опис функцій API:

- `add_book(book_data)`: додає нову книгу до бібліотеки користувача на основі наданих даних;
- `edit_book(book_id, new_data)`: редагує існуючий запис про книгу за її ідентифікатором з новими даними;
- `delete_book(book_id)`: видаляє книгу з бібліотеки користувача за її ідентифікатором;
- `get_user_books(user_id)`: повертає список книг користувача за його ідентифікатором;
- `get_recommendations(user_id)`: повертає рекомендації книг для користувача на основі його вибору та попередніх оцінок.

Нижче наведено опис роботи.

1. Аутентифікація та авторизація користувача.
2. Обробка запитів користувачів для збереження, редагування або видалення книг.
3. Інтеграція з іншими сервісами для автоматичного додавання книг або отримання рекомендацій.
4. Забезпечення безпеки та конфіденційності даних користувачів.

3.3.2 Сервіс роботи із зовнішніми джерелами

Цей сервіс надає можливість конвертації даних з CSV-файлу у словник та отримання додаткової інформації про книги з використанням API від Open Library.

Кроки реалізації.

1. Читання даних з CSV-файлу та конвертація їх у словник.
2. Перетворення словника в новий формат з обраними ключами.

3. Використання отриманих даних (назва та автор книги) для отримання додаткової інформації з API.
4. Оновлення словника з отриманою додатковою інформацією.

Вхідні дані:

- шлях до CSV-файлу з даними про книги.

Вихідні дані:

- словник з додатковою інформацією про книги, отриманою з API.

Опис функцій:

- `goodreads_csv_to_dict(csv_file_path)`: ця функція приймає шлях до CSV-файлу з даними про книги та повертає список словників з цими даними;
- `dict_to_storage(book_list)`: ця функція перетворює список словників з даними про книги в новий формат з обраними ключами та повертає оновлений список словників;
- `get_info_api(book_element)`: ця функція виконує запит до API Open Library за даними про книгу та оновлює словник з додатковою інформацією.

3.3.3 Сервіси з вираховання допасованості

Вираховання допасованості є ключовим елементом сучасних технологічних екосистем. Вони надають основу для вираховання адаптивних, ефективних та еластичних послуг, які відповідають специфічним потребам користувачів та бізнес-вимогам. Застосовуються в різних галузях, від хмарних обчислень до розумних міст та інтернету речей (IoT).

Основні характеристики:

- адаптивність: сервіси з вираховання допасованості реагують на змінність потреб користувачів та вимоги ринку, можуть швидко змінювати свою функціональність, розширювати або зменшувати обсяги обробки даних та ресурсів;
- ефективність: ці сервіси оптимізовані для виконання конкретних завдань, максимально використовуючи наявні ресурси та забезпечуючи високу продуктивність;
- еластичність: сервіси можуть масштабуватися вгору або вниз в залежності від поточного навантаження або вимог щодо обробки даних, що дозволяє забезпечувати стабільну продуктивність навіть при змінній величині робочого навантаження;
- розподіленість: сервіси можуть бути розгорнуті в різних середовищах, включаючи хмарні, краєві та локальні обчислення, щоб надати максимальну гнучкість та доступність.

3.3.3.1 Сервіси на основі матричного розкладу

Альтернативні мінімізація суми квадратів (ALS) – це ітеративний метод оптимізації для рекомендаційних систем та факторизації матриць. У факторизації матриць ALS розбиває вихідну матрицю на декілька менших матриць, які потім оптимізуються шляхом змінних однієї з них та фіксації інших. Цей процес повторюється до зближення до мінімуму функції втрат. В рекомендаційних системах ALS може використовуватися для прогнозування оцінок користувачів для товарів, виходячи з їх попередніх оцінок.

Нестрогий факторний розклад (NMF) – це метод розкладання матриць на добуток двох або більше менших матриць з нестриктурними умовами. У контексті рекомендаційних систем, NMF використовується для отримання латентних факторів, які описують спільні характеристики між користувачами та предметами. Ці латентні фактори можуть бути використані для рекомендацій товарів, які можуть сподобатися користувачам, або для стиснення та відновлення даних.

Сингулярний розклад матриць (SVD) – це метод факторизації матриць, який розкладає вихідну матрицю на добуток трьох матриць: U , S , та V^t . У контексті рекомендаційних систем, SVD може використовуватися для зменшення розмірності даних та отримання латентних факторів, які описують взаємодію між користувачами та предметами. Ці латентні фактори дозволяють здійснювати рекомендації на основі схожості між користувачами та предметами.

Аналіз головних компонентів (PCA) – це метод зменшення розмірності даних, який використовується для виявлення головних компонентів або напрямків, що мають найбільшу дисперсію у вихідних даних. У рекомендаційних системах, PCA може використовуватися для зменшення розмірності матриці оцінок користувачів та предметів, з метою виявлення латентних факторів, що визначають взаємодію між ними. Ці латентні фактори

можуть бути використані для рекомендацій товарів на основі схожості між користувачами або предметами.

3.3.3.1.1 Сервіс на основі SVD-розкладу

Цей Flask – сервіс використовує метод SVD (Сингулярний Розклад) для створення рекомендацій книг на основі попередніх оцінок користувачів.

Кроки реалізації.

1. Створення Flask-додатку та визначення маршруту `/svd_predict_ratings` для POST-запитів.
2. У POST-запиті виконується отримання JSON-об'єкта з даними, які включають дані про книги (`data`), оцінки користувачів (`user_ratings`).
3. Кодування авторів та предметів за допомогою `OneHotEncoder`.
4. Формування матриці даних для подальшого аналізу SVD.
5. Застосування SVD для отримання матриць U , S та V_t .
6. Відновлення вихідної матриці за допомогою SVD.
7. Прогнозування оцінок для книг, які користувач ще не оцінив.
8. Нормалізація прогнозованих оцінок до діапазону від 0 до 5.
9. Повернення JSON-об'єкта з прогнозованими оцінками для нерейтингованих книг.

Вхідні дані:

- POST-запит на маршрут `/svd_predict_ratings`;
- JSON-об'єкт у тілі запиту з полями:
 - `data`: масив з даними про книги, де кожен елемент масиву містить ім'я автора, оцінку, кількість сторінок, список жанрів;
 - `user_ratings`: масив з оцінками користувача для відповідних книг. 0 означає, що книга не була оцінена користувачем.

Вихідні дані:

- JSON-об'єкт з полем `predicted_ratings`, що містить прогнозовані оцінки для книг, які користувач ще не оцінив.

Нижче наведено покроковий опис роботи.

1. Запуск Flask-сервісу.
2. Прийом POST-запитів на маршрут `/svd_predict_ratings`.
3. Обробка вхідних даних: кодування авторів та предметів, створення матриці даних.
4. Використання SVD для зменшення розмірності матриці та отримання головних компонент.
5. Відновлення вихідної матриці за допомогою сингулярного розкладу.
6. Прогнозування оцінок для книг, які ще не оцінені користувачем.
7. Нормалізація прогнозованих оцінок та їх повернення у JSON-форматі.

Цей сервіс дозволяє легко отримувати рекомендації для книг на основі існуючих даних про оцінки та характеристики книг, використовуючи метод SVD для зменшення розмірності та факторизації матриць.

3.3.3.1.2 Сервіс на основі NMF-розкладу

Цей Flask-сервіс використовує метод неконтрольованого навчання NMF (Негативна Матриця Факторизації) для створення рекомендацій книг на основі попередніх оцінок користувачів.

Кроки реалізації.

1. Створення Flask-додатку та визначення маршруту `/nmf_predict_ratings` для POST-запитів.

2. У POST-запиті виконується отримання JSON-об'єкта з даними, які включають дані про книги (`data`), оцінки користувачів (`user_ratings`).
3. Кодування авторів та предметів за допомогою `OneHotEncoder`.
4. Формування матриці даних для подальшого аналізу NMF.
5. Застосування NMF для отримання матриць W та H .
6. Відновлення вихідної матриці з отриманих W та H .
7. Виділення частини матриці, що відповідає оцінкам.
8. Прогнозування оцінок для книг, які користувач ще не оцінив.
9. Нормалізація прогнозованих оцінок до діапазону від 0 до 5.
10. Повернення JSON-об'єкта з прогнозованими оцінками для нерейтингованих книг.

Вхідні дані:

- POST-запит на маршрут `/predict_ratings`;
- JSON-об'єкт у тілі запиту з полями:
 - `data`: масив з даними про книги, де кожен елемент масиву містить ім'я автора, оцінку, кількість сторінок та список тем;
 - `user_ratings`: масив з оцінками користувача для відповідних книг. 0 означає, що книга не була оцінена користувачем.

Вихідні дані:

- JSON-об'єкт з полем `predicted_ratings`, що містить прогнозовані оцінки для книг, які користувач ще не оцінив.

Опис роботи.

1. Запуск `Flask-сервісу`.
2. Прийом POST-запитів на маршрут `/predict_ratings`.
3. Обробка вхідних даних: кодування авторів та предметів, створення матриці даних.
4. Використання NMF для розбиття матриці на дві менші матриці (W та H).

5. Відновлення вихідної матриці та прогнозування оцінок для книг, які ще не оцінені користувачем.
6. Нормалізація прогнозованих оцінок та їх повернення у JSON-форматі.

Цей сервіс дозволяє легко отримувати рекомендації для книг на основі існуючих даних про оцінки та характеристики книг, використовуючи метод NMF для факторизації матриць.

3.3.3.1.3 Сервіс на основі PCA-розкладу

Цей Flask-сервіс використовує метод зменшення розмірності PCA (Головні Компоненти Аналізу) для створення рекомендацій книг на основі попередніх оцінок користувачів.

Кроки реалізації.

1. Створення Flask-додатку та визначення маршруту `/pca_predict_ratings` для POST-запитів.
2. У POST-запиті виконується отримання JSON-об'єкта з даними, які включають дані про книги (`data`), оцінки користувачів (`user_ratings`).
3. Кодування авторів та предметів за допомогою `OneHotEncoder`.
4. Формування матриці даних для подальшого аналізу PCA.
5. Застосування PCA для отримання головних компонент.
6. Відновлення вихідної матриці за допомогою оберненого перетворення PCA.
7. Виділення частини матриці, що відповідає оцінкам.
8. Прогнозування оцінок для книг, які користувач ще не оцінив.
9. Нормалізація прогнозованих оцінок до діапазону від 0 до 5.

10. Повернення JSON-об'єкта з прогнозованими оцінками для нерейтингованих книг.

Вхідні дані:

- POST-запит на маршрут `/pca_predict_ratings`;
- JSON-об'єкт у тілі запиту з полями:
 - `data`: масив з даними про книги, де кожен елемент масиву містить, ім'я автора, оцінку, кількість сторінок та список тем;
 - `user_ratings`: масив з оцінками користувача для відповідних книг. 0 означає, що книга не була оцінена користувачем.

Вихідні дані:

- JSON-об'єкт з полем `predicted_ratings`, що містить прогнозовані оцінки для книг, які користувач ще не оцінив.

Опис роботи.

1. Запуск Flask-сервісу.
2. Прийом POST-запитів на маршрут `/pca_predict_ratings`.
3. Обробка вхідних даних: кодування авторів та предметів, створення матриці даних.
4. Використання PCA для зменшення розмірності матриці та отримання головних компонент.
5. Відновлення вихідної матриці за допомогою оберненого перетворення PCA.
6. Прогнозування оцінок для книг, які ще не оцінені користувачем.
7. Нормалізація прогнозованих оцінок та їх повернення у JSON-форматі.

Цей сервіс дозволяє легко отримувати рекомендації для книг на основі існуючих даних про оцінки та характеристики книг, використовуючи метод PCA для зменшення розмірності та факторизації матриць.

3.3.3.1.4 Сервіс на основі ALS-розкладу

Цей Flask-сервіс використовує метод ALS (Alternating Least Squares) для створення рекомендацій книг на основі попередніх оцінок користувачів.

Кроки реалізації

1. Створення Flask-додатку та визначення маршруту `/als_predict_ratings` для POST-запитів.
2. У POST-запиті виконується отримання JSON-об'єкта з даними, які включають дані про книги (`data`), оцінки користувачів (`user_ratings`).
3. Кодування авторів та предметів за допомогою `OneHotEncoder`.
4. Формування матриці даних для подальшого аналізу ALS.
5. Застосування ALS для обчислення факторів користувачів та книг.
6. Прогнозування оцінок для книг, які користувач ще не оцінив.
7. Нормалізація прогнозованих оцінок до діапазону від 0 до 5.
8. Повернення JSON-об'єкта з прогнозованими оцінками для нерейтингованих книг.

Вхідні дані:

- POST-запит на маршрут `/als_predict_ratings`;
- JSON-об'єкт у тілі запиту з полями:
 - `data`: масив з даними про книги, де кожен елемент масиву містить ім'я автора, оцінку, кількість сторінок та список тем;
 - `user_ratings`: масив з оцінками користувача для відповідних книг. 0 означає, що книга не була оцінена користувачем.

Вихідні дані:

- JSON-об'єкт з полем `predicted_ratings`, що містить прогнозовані оцінки для книг, які користувач ще не оцінив.

Опис роботи.

1. Запуск Flask-сервісу.

2. Прийом POST-запитів на маршрут `/als_predict_ratings`.
3. Обробка вхідних даних: кодування авторів та предметів, створення матриці даних.
4. Використання ALS для обчислення факторів користувачів та книг.
5. Прогнозування оцінок для книг, які ще не оцінені користувачем.
6. Нормалізація прогнозованих оцінок та їх повернення у JSON-форматі.

Цей сервіс дозволяє легко отримувати рекомендації для книг на основі існуючих даних про оцінки та характеристики книг, використовуючи метод ALS для факторизації матриць.

3.3.3.2 Сервіс на основі пошуку за найближчими сусідами

Цей Flask-сервіс використовує метод лінійної регресії за найближчими сусідами для створення рекомендацій книг на основі попередніх оцінок користувачів.

Кроки реалізації.

1. Створення Flask-додатку та визначення маршруту `/lr_predict_ratings` для POST-запитів.
2. У POST-запиті виконується отримання JSON-об'єкта з даними, які включають дані про книги (`data`), оцінки користувачів (`user_ratings`).
3. Кодування авторів та предметів за допомогою `OneHotEncoder`.
4. Формування матриці даних для подальшого аналізу лінійної регресії.
5. Навчання моделі лінійної регресії на основі відомих оцінок.
6. Прогнозування оцінок для книг, які користувач ще не оцінив.

7. Повернення JSON-об'єкта з прогнозованими оцінками для нерейтингованих книг.

Вхідні дані

- POST-запит на маршрут `/lr_predict_ratings`;
- JSON-об'єкт у тілі запиту з полями:
 - `data`: масив з даними про книги, де кожен елемент масиву містить назву книги, ім'я автора, кількість сторінок та список тем;
 - `user_ratings`: масив з оцінками користувача для відповідних книг. 0 означає, що книга не була оцінена користувачем.

Вихідні дані:

- JSON-об'єкт з полем `predicted_ratings`, що містить прогнозовані оцінки для книг, які користувач ще не оцінив.

Опис роботи

1. Запуск Flask-сервісу.
2. Прийом POST-запитів на маршрут `/lr_predict_ratings`.
3. Обробка вхідних даних: кодування авторів та предметів, створення матриці даних.
4. Використання лінійної регресії для навчання моделі на основі відомих оцінок.
5. Прогнозування оцінок для книг, які ще не оцінені користувачем.
6. Повернення прогнозованих оцінок у форматі JSON.

Цей сервіс дозволяє легко отримувати рекомендації для книг на основі існуючих даних про оцінки та характеристики книг, використовуючи метод лінійної регресії для прогнозування оцінок.

3.3.3.3 Сервіс на основі фільтрації за контентом

Цей Flask-сервіс використовує метод косинусної подібності для створення фільтрації даних за контентом для рекомендацій книг на основі попередніх оцінок користувача.

Кроки реалізації

1. Створення Flask-додатку та визначення маршруту `/cosine_similarity_predict` для POST-запитів.
2. У POST-запиті виконується отримання JSON-об'єкта з даними, які включають дані про книги (`data`) та оцінки користувачів (`user_ratings`).
3. Кодування авторів та предметів за допомогою `OneHotEncoder`.
4. Формування та нормалізація матриці даних для подальшого аналізу.
5. Обчислення матриці подібності за допомогою косинусної подібності.
6. Прогнозування оцінок для книг, які користувач ще не оцінив, на основі подібності до оцінених книг.
7. Повернення JSON-об'єкта з прогнозованими оцінками для нерейтингованих книг.

Вхідні дані:

- POST-запит на маршрут `/cosine_similarity_predict`;
- JSON-об'єкт у тілі запиту з полями:
 - `data`: масив з даними про книги, де кожен елемент масиву містить назву книги, ім'я автора, кількість сторінок та список тем;
 - `user_ratings`: масив з оцінками користувача для відповідних книг. 0 означає, що книга не була оцінена користувачем.

Вихідні дані:

- JSON-об'єкт з полем `predicted_ratings`, що містить прогнозовані оцінки для книг, які користувач ще не оцінив.

3.3.3.4 Сервіс на основі генетичних алгоритмів

Цей Flask-сервіс використовує генетичний алгоритм для створення рекомендацій книг на основі попередніх оцінок користувачів.

Кроки реалізації

1. Створення Flask-додатку та визначення маршруту `/genetic_predict_ratings` для POST-запитів.
2. У POST-запиті виконується отримання JSON-об'єкта з даними, які включають дані про книги (`data`) та оцінки користувачів (`user_ratings`).
3. Ініціалізація популяції для генетичного алгоритму.
4. Запуск генетичного алгоритму з обчисленням функції пристосованості для кожного індивідуума.
5. Вибір батьків, кросовер та мутація для створення нового покоління.
6. Прогнозування оцінок для книг, які користувач ще не оцінив.
7. Нормалізація прогнозованих оцінок до діапазону від 0 до 5.
8. Повернення JSON-об'єкта з прогнозованими оцінками для нерейтингованих книг.

Вхідні дані:

- POST-запит на маршрут `/genetic_predict_ratings`.
- JSON-об'єкт у тілі запиту з полями:
 - `data`: масив з даними про книги, де кожен елемент масиву містить інформацію про книгу;

- `user_ratings`: масив з оцінками користувача для відповідних книг. 0 означає, що книга не була оцінена користувачем.

Вихідні дані:

- JSON-об'єкт з полем `predicted_ratings`, що містить прогнозовані оцінки для книг, які користувач ще не оцінив.

Опис роботи.

1. Запуск Flask-сервісу.
2. Прийом POST-запитів на маршрут `/genetic_predict_ratings`.
3. Обробка вхідних даних: ініціалізація популяції та параметрів генетичного алгоритму.
4. Виконання генетичного алгоритму для пошуку найкращих рекомендацій.
5. Прогнозування оцінок для книг, які ще не оцінені користувачем, на основі генетичного алгоритму.
6. Нормалізація прогнозованих оцінок та їх повернення у JSON-форматі.

Цей сервіс дозволяє легко отримувати рекомендації для книг на основі існуючих даних про оцінки та характеристики книг, використовуючи генетичний алгоритм для оптимізації вибору книг.

3.3.4 Аналіз результатів

Аналізуючи різні методи прогнозування книжкових рейтингів, стає зрозуміло, що кожен підхід має унікальні переваги та недоліки, що робить їх придатними для різних сценаріїв. Вибір найбільш підходящого методу залежить від характеристик даних, обсягу доступної інформації, обчислювальних ресурсів

та бажаної точності прогнозів. Нижче ми детально розглянемо кожен метод та його застосування.

Сервіс на основі SVD використовує метод сингулярної декомпозиції (SVD) для декомпозиції матриці характеристик книг і подальшого прогнозування рейтингів. Цей метод ефективно виявляє приховані зв'язки між книжками та користувачами і допомагає зменшити розмірність даних, що полегшує визначення основних компонентів, які впливають на рейтинги. Однак він не враховує індивідуальні особливості користувачів, зосереджуючись на загальних тенденціях. Крім того, він вимагає значного обсягу даних для точних прогнозів. Результати використання SVD включають прогнозовані рейтинги книжок на основі їхніх характеристик та загальної структури рейтингової матриці. Нормалізовані рейтинги дозволяють порівнювати книги незалежно від вихідної шкали.

Сервіс на основі NMF використовує метод негативної матричної факторизації (NMF) для декомпозиції матриці характеристик книг і прогнозування рейтингів. NMF добре працює з не-негативними даними і дозволяє отримати інтерпретовані фактори, що описують основні характеристики книг. Цей метод забезпечує більш зрозумілі результати порівняно з SVD, оскільки фактори мають чіткі інтерпретації. Проте NMF може бути менш точним, ніж інші методи, якщо дані не відповідають вимогам не-негативності. Також, як і SVD, він вимагає значного обсягу даних для забезпечення точних прогнозів. Нормалізовані рейтинги, отримані за допомогою NMF, дозволяють легко порівнювати книги незалежно від вихідної шкали.

Сервіс на основі PCA використовує метод головних компонент (PCA) для зменшення розмірності даних і прогнозування рейтингів. PCA допомагає виявити основні компоненти, які пояснюють найбільшу частку варіації в даних. Цей метод ефективний для обробки великих обсягів даних і зменшення розмірності без значних втрат інформації. Однак PCA може не завжди добре працювати з розрідженими матрицями і не враховує індивідуальні відмінності користувачів. Крім того, результати PCA можуть бути складнішими для

інтерпретації порівняно з NMF. Нормалізовані рейтинги, отримані за допомогою PCA, дозволяють легко порівнювати книги незалежно від вихідної шкали.

Сервіс на основі ALS використовує метод чергування найменших квадратів (ALS) для декомпозиції матриці характеристик книг і прогнозування рейтингів. ALS добре підходить для роботи з великими розрідженими матрицями, часто зустрічається у реальних даних. Цей метод дозволяє враховувати індивідуальні відмінності користувачів, що покращує точність прогнозів. ALS швидко сходиться і забезпечує гарні результати навіть з обмеженою кількістю даних. Однак цей метод може бути чутливим до вибору параметрів і вимагає налаштування для досягнення найкращих результатів. Нормалізовані рейтинги, отримані за допомогою ALS, дозволяють легко порівнювати книги незалежно від вихідної шкали.

Сервіс на основі косинусної подібності (фільтрація за контентом) вимірює схожість між книгами для прогнозування рейтингів. Цей метод враховує схожість між книгами, що дозволяє надавати більш персоналізовані рекомендації. Він також простий у реалізації та швидкий в обчисленнях. Однак він може бути менш точним у випадку багатовимірних і складних даних, оскільки фокусується на схожості книжок, не враховуючи різноманітність оцінок користувачів. Результати включають прогнозовані рейтинги книжок на основі їхньої схожості з уже оціненими книжками, а нормалізовані бали полегшують інтерпретацію результатів.

Сервіс на основі лінійної регресії використовує лінійну регресію для прогнозування рейтингів на основі характеристик книги. Він простий у застосуванні та інтерпретації і добре працює з даними, що демонструють лінійні зв'язки. Однак цей метод не враховує нелінійні зв'язки між характеристиками книги і є менш ефективним для складних даних, що включають численні взаємозв'язки. Результатом є прогнозовані рейтинги для книг на основі лінійних зв'язків між характеристиками, хоча оцінки є ненормованими, що може ускладнити порівняння.

Сервіс на основі генетичного алгоритму оптимізує вибір книг і прогнозує рейтинги, враховуючи індивідуальні вподобання користувачів. Цей метод здатен знаходити глобальні оптимуми у складних пошукових просторах. Однак він має високу обчислювальну складність і значні вимоги до ресурсів для обробки великих поколінь. Крім того, випадковість процесів мутації та кросинговеру може призвести до нестабільних результатів. Результатом є прогнозовані рейтинги книг на основі оптимізованого відбору, причому нормалізовані рейтинги забезпечують порівнянність результатів.

Висновок до розділу 3

У цьому розділі визначено функціональні вимоги для рекомендаційної системи, включаючи сервіси для роботи з даними, запитами до сторонніх сервісів та аналізу допасованості. Було обґрунтовано вибір усіх інструментів, які будуть використовуватись для розробки, і проведено аналіз їхніх можливостей та функціоналу. Після визначення функціональних вимог та вибору інструментів було створено власний сервіс, який дозволяє користувачам керувати власною книгозбірнею, додаючи, редагуючи та видаляючи книги, а також отримувати персоналізовані рекомендації на основі своїх читацьких вподобань та рейтингів. Реалізація включає створення бази даних для зберігання інформації про книги, розробку API для функцій управління книгами, підтримку інтеграції з іншими сервісами для автоматичного додавання книг і генерації рекомендацій, а також забезпечення надійної автентифікації користувачів і конфіденційності даних. Сервіс обробляє дані про книги та запити користувачів, надаючи підтвердження операцій, детальний список книг та індивідуальні книжкові рекомендації. Для досягнення оптимальних результатів може бути корисним комбінування методів.

Сервіс дозволяє користувачам керувати власною книгозбіркою, додаючи, редагуючи та видаляючи книги, а також отримувати персоналізовані рекомендації на основі своїх читацьких вподобань та рейтингів. Реалізація включає створення бази даних для зберігання інформації про книги, розробку API для функцій управління книгами, підтримку інтеграції з іншими сервісами для автоматичного додавання книг і генерації рекомендацій, а також забезпечення надійної автентифікації користувачів і конфіденційності даних. Сервіс обробляє дані про книги та запити користувачів, надаючи підтвердження операцій, детальний список книг та індивідуальні книжкові рекомендації. Для досягнення оптимальних результатів може бути корисним комбінування методів.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на рекомендації літератури.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

- F_1 – вибір самої програми;
- F_2 – якісний аналіз даних;
- F_3 – графічні показники.

Кожна з цих функцій має декілька варіантів реалізації:

- Функція F_1 :

- Python;
- C++.
- Функція F_2 :
 - застосування вбудованих функцій;
 - створення своїх обчислень значень.
- Функція F_3 :
 - використання шаблонних графіків;
 - створення своїх.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

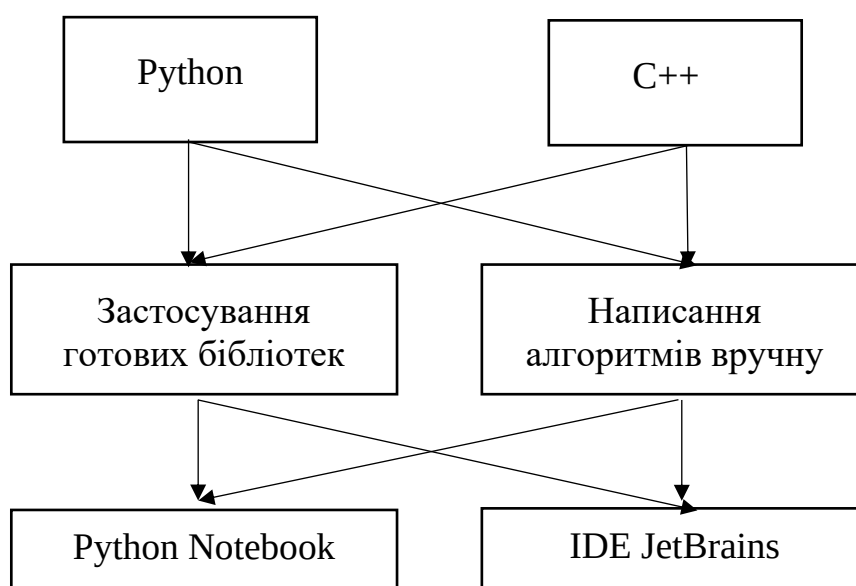


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
---------	---------------------	----------	----------

F_1	A	Багато готових рішень, доступність багатьох бібліотек, кросплатформеність	Повільне виконання, динамічна типізація
	B	Код швидко виконується	Мала кількість бібліотек та готових рішень
F_2	A	Доступність та легкість при написанні	Іноді не відповідає задачі яку треба розв'язати
	B	Ідеально описують усі необхідні характеристики	Достатньо затратно реалізовувати свої алгоритми для подальшої реалізації
F_3	A	Широкий вибір можливостей, легко запускається на будь-якому сервері	Відсутність відлагодження коду
	B	Доступна можливість відлагодження коду	Необхідна локальна інсталяція

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1 .

Перевагу даємо загальнодоступності. Для спрощення роботи по написанню коду варіант Б має бути відкинтий.

Функція F_2 .

Існуючі бібліотеки мови програмування Python є надійними і оптимізованими для різних задач, тому використовуємо їх для більшості функціоналу. Але наша робота вимагає написання деяких алгоритмів вручну, тому розглянемо варіант А та Б.

Функція F_3 .

Через можливість відлагодження коду обираємо другий варіант – варіант Б.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

$$F_1a - F_2a - F_3b$$

$$F_1a - F_2b - F_3b$$

Для оцінювання якості розглянутих функцій обрана система параметрів описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X_1 – швидкодія мови програмування;
- X_2 – об'єм пам'яті для обчислень та збереження даних;
- X_3 – потенційний об'єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі

Швидкодія мови програмування	X1	оп/мс	100	1000	10000
Об'єм пам'яті	X2	Мб	320	128	64
Потенційний об'єм програмного коду	X3	кількість рядків коду	1500	800	350

За даними таблиці 4.3 будуються графічні характеристики параметрів –
рис. 4.2 – рис. 4.4.

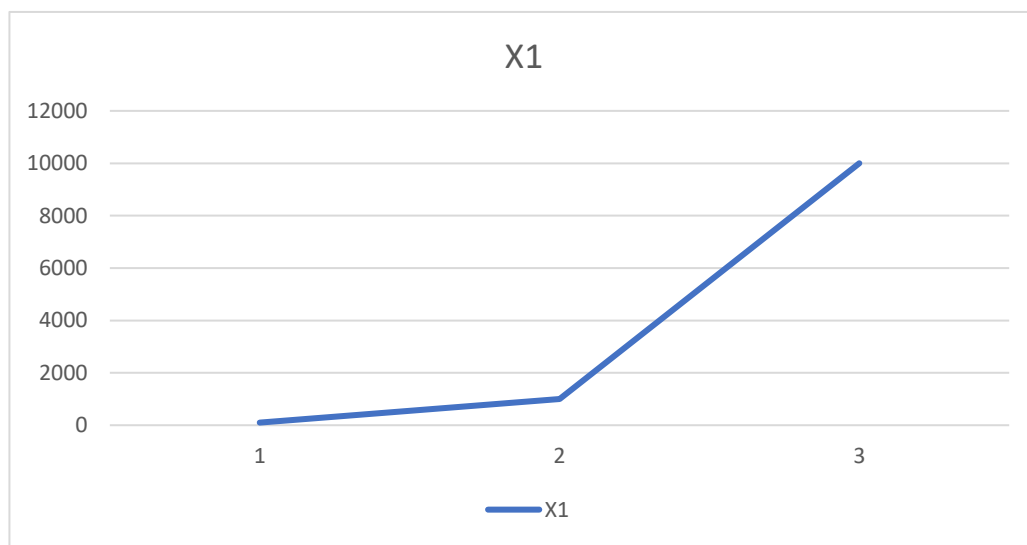


Рисунок 4.2 – X1, швидкодія мови програмування

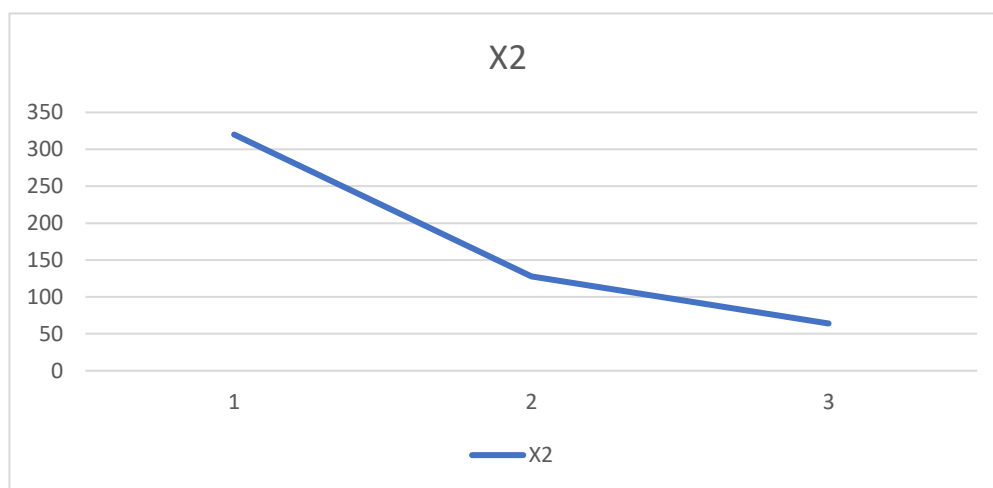


Рисунок 4.3 – X2, об'єм пам'яті

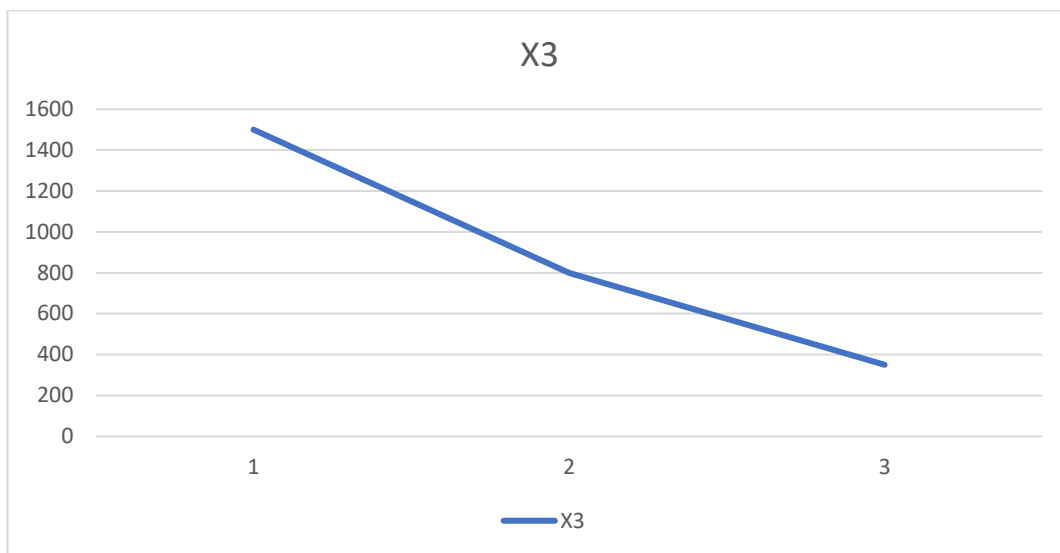


Рисунок 4.4 – X3, потенційний об'єм коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
$X1$	Швидкодія мови програмування	Оп/мс	1	2	1	1	1	1	1	8	-6	36
$X2$	Об'єм пам'яті	Мб	3	3	2	2	3	2	2	17	3	9
$X3$	Потенційний об'єм програмного коду	Кількість рядків коду	2	1	3	3	2	3	3	17	3	9
	Разом		6	6	6	6	6	6	6	42	0	54

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

- сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 42,$$

де N – число експертів,

n – кількість параметрів.

- середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 14.$$

- відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T.$$

Сума відхилень по всіх параметрах повинна дорівнювати 0;

- загальна сума квадратів відхилення:

$$S = \sum_{i=1}^N \Delta_i^2 = 54.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 54}{7^2(3^3-3)} = 2,20408 > W_k = 0,67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0,5
X1 і X3	<	>	<	<	<	<	<	<	0,5
X2 і X3	>	>	<	<	>	<	<	<	0,5

Числове значення, що визначає ступінь переваги i -го параметра над j -тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$.

Для кожного параметра зробимо розрахунок вагомості K_{bi} за наступними формулами:

$$K_{bi} = \frac{b_i}{\sum_{i=1}^n b_i},$$

$$b_i = \sum_{j=1}^N a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{bi} = \frac{b'_i}{\sum_{i=1}^n b'_i},$$

$$b'_i = \sum_{j=1}^N a_{ij} b_j.$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j			Перша ітер.		Друга ітер.	
	$X1$	$X2$	$X3$	b_i	K_{bi}	b_i^1	K_{bi}^1
$X1$	1	0,5	0,5	2	0,222	5,5	0,2156
$X2$	1,5	1	1	3,5	0,388	10	0,3921
$X3$	1,5	1	1	3,5	0,388	10	0,3921
Всього:				9	1	25,5	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X_2 та X_1 відповідають технічним вимогам умов функціонування даного ПП.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^n K_{ei,j} B_{i,j},$$

де n – кількість параметрів;

K_{ei} – коефіцієнт вагомості i -го параметра;

B_i – оцінка i -го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	A	X1	1000	9	0,22	1,98
F2	A	X2	128	7	0,38	2,66
F2	B	X2	320	6	0,38	2,28
F3	A	X3	800	5	0,38	1,9

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \dots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1,98 + 2,66 + 1,9 = 6,54;$$

$$K_{K2} = 1,98 + 2,28 + 1,9 = 6,16 .$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання.

1. Розробка проекту програмного продукту.
2. Розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_0 = T_P \cdot K_P \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_p – трудомісткість розробки ПП;

K_{Π} – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_p = 37$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_{\Pi} = 1.8$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ} = 0.9$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 37 \cdot 1.8 \cdot 0.9 = 59,94 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_p = 29$ людино-днів, $K_{\Pi} = 0.9$, $K_{СК} = 1$, $K_{СТ} = 0.8$:

$$T_2 = 29 \cdot 0.9 \cdot 0.8 = 20.88 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_1 = (59,94 + 20.88 + 4.8 + 20.88) \cdot 8 = 852 \text{ людино-годин.}$$

$$T_{II} = (59,94 + 20,88 + 6,91 + 20,88) \cdot 8 = 868,88 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь один програміст з окладом 30000 грн., один аналітик в області даних з окладом 25000. Визначимо середню зарплату за годину за формулою:

$$C_q = \frac{M}{T_m \cdot t} \text{ грн.,}$$

де M – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$C_q = \frac{30000 + 25000}{3 \cdot 21 \cdot 8} = 109,13 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{зП} = C_q \cdot T_i \cdot K_d,$$

де C_q – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

K_d – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I. } C_{зП} = 109,13 \cdot 852 \cdot 1,2 = 111\,574,51 \text{ грн.}$$

$$\text{II. } C_{зП} = 109,13 \cdot 868,88 \cdot 1,2 = 113\,785,05 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I. } C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 111\,574,51 \cdot 0.22 = 24\,546,39 \text{ грн.}$$

$$\text{II. } C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 113\,785,05 \cdot 0.22 = 25\,032,71 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. (C_M)

Так як одна ЕОМ обслуговує одного програміста з окладом 30000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_G = 12 \cdot M \cdot K_3 = 12 \cdot 30000 \cdot 0,2 = 72\,000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{\text{ЗП}} = C_G \cdot (1 + K_3) = 72000 \cdot (1 + 0.2) = 86\,400 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 86\,400 \cdot 0,22 = 19\,008 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$$C_A = K_{\text{ТМ}} \cdot K_A \cdot Ц_{\text{ПР}} = 1.4 \cdot 0.25 \cdot 10000 = 3500 \text{ грн.,}$$

де $K_{\text{ТМ}}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

K_A – річна норма амортизації;

$Ц_{\text{ПР}}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{TM} \cdot C_{ПР} \cdot K_P = 1.4 \cdot 10000 \cdot 0.08 = 1120 \text{ грн.},$$

де K_P – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{\text{ЕФ}} = (D_K - D_B - D_C - D_P) \cdot t_3 \cdot K_B = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.35 = \\ = 627,2 \text{ години},$$

де D_K – календарна кількість днів у році;

D_B, D_C – відповідно кількість вихідних та святкових днів;

D_P – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} \cdot N_C \cdot K_3 \cdot C_{\text{ЕН}} = 627,2 \cdot 0,2 \cdot 0,3 \cdot 5,06 = 190,41 \text{ грн.},$$

де N_C – середньо-споживча потужність приладу;

K_3 – коефіцієнтом зайнятості приладу;

$C_{\text{ЕН}}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_H = C_{\text{ПР}} \cdot 0.67 = 10000 \cdot 0.67 = 6700 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_A + C_P + C_{\text{ЕЛ}} + C_H,$$

$$C_{\text{ЕКС}} = 86\,400 + 19\,008 + 3500 + 1120 + 190,41 + 6700 = 116\,918,41 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = C_{\text{ЕКС}} / T_{\text{ЕФ}} = 116\,918,41 / 627,2 = 186,41 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_M = C_{\text{М-Г}} \cdot T,$$

$$\text{I. } C_M = 186,41 \cdot 852 = 158\,821,32 \text{ грн.}$$

$$\text{II. } C_M = 186,41 \cdot 868,88 = 161\,967,92 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{\text{ЗП}} \cdot 0.67,$$

$$\text{I. } C_H = 111\,574,51 \cdot 0.67 = 74\,754,92 \text{ грн.}$$

$$\text{II. } C_H = 113\,785,05 \cdot 0,67 = 76\,234,98 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{\text{ПП}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{М}} + C_{\text{Н}},$$

$$\text{I. } C_{\text{ПП}} = 111\,574,51 + 24\,546,39 + 158\,821,32 + 74\,754,92 = 369\,697,14 \text{ грн.}$$

$$\text{II. } C_{\text{ПП}} = 113\,785,05 + 25\,032,71 + 161\,967,92 + 76\,234,98 = 377\,019,66 \text{ грн.}$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = K_{\text{К}j} / C_{\text{Ф}j},$$

$$K_{\text{ТЕР}1} = 6,54 / 369\,697,14 = 1,775 \cdot 10^{-5},$$

$$K_{\text{ТЕР}2} = 6,16 / 377\,019,66 = 1,636 \cdot 10^{-5}$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{\text{ТЕР}1} = 1,775 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості – $K_{\text{ТЕР}} = 1,775 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- вибір програмного продукту – Python;
- використання готових бібліотек;
- середовище розробки IDE JetBrains.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

Висновок до розділу 4

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання ФВА програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують. На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У роботі було проведено огляд існуючих систем рекомендації літератури, таких як Goodreads, StoryGraph, LibraryThing, BookLikes, Litsy, BookBub та Libib та зроблено висновок, що сучасні цифрові інструменти значно розширюють горизонти читання, роблячи процес пошуку та вибору літератури більш особистісним та динамічним. Незважаючи на це, існує необхідність у створенні більш точної та ефективної системи рекомендацій, яка б відповідала індивідуальним перевагам користувачів. Розглянувши різні методології, що застосовуються для створення рекомендаційних систем, можна відзначити важливість збору інформації через публічні API, Data Scraping, використання послуг Data Providers та створення власної бази даних. Важливим аспектом є також вибір стратегії збереження та організації даних, включаючи реляційні та нереляційні бази даних, файлове збереження та хмарні сховища.

Методи штучного інтелекту, такі як генетичні алгоритми, фільтрація на основі контенту, алгоритми k-найближчих сусідів та моделі на основі матричного розкладу (SVD, ALS, PCA, NMF), мають вирішальне значення для персоналізації користувацького досвіду. Зокрема, моделі на основі матричного розкладу виявились одними з найбільш ефективних для роботи з великими обсягами даних, оскільки вони дозволяють виявити латентні фактори, що впливають на вибір та вподобання користувачів. Правильний вибір методологій для збору, збереження, обробки та аналізу даних є ключовим для створення ефективної рекомендаційної системи, що відповідатиме потребам користувачів.

Також виявлено, що кожен з розглянутих методів прогнозування книжкових рейтингів має свої унікальні переваги та недоліки, що робить їх придатними для різних сценаріїв залежно від характеристик даних, обсягу інформації, обчислювальних ресурсів та необхідної точності прогнозів. Методи, такі як SVD, NMF, PCA та ALS, добре підходять для роботи з великими обсягами даних і розрідженими матрицями, забезпечуючи точні результати та зменшуючи

розмірність даних, проте вимагають налаштування та значних ресурсів. Методи косинусної подібності та лінійної регресії прості в реалізації і швидкі в обчисленнях, але можуть бути менш точними у випадку складних даних. Генетичні алгоритми забезпечують високий рівень оптимізації, проте є обчислювально складними та ресурсомісткими. Вибір найбільш підходящого методу залежить від конкретних вимог і обмежень проєкту, що визначає ефективність і якість прогнозів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Y. Wadhwani, “The Best Free Sites for Book Lovers,” How-To Geek, Nov. 25, 2018. <https://www.howtogeek.com/395629/the-best-free-sites-for-book-lovers/>
2. GoodReads, “Goodreads,” Goodreads, 2022. <https://www.goodreads.com/>
3. “The StoryGraph,” thestorygraph.com. <https://thestorygraph.com/>
4. LibraryThing, “LibraryThing | Catalog your books online,” Librarything.com, 2011. <https://www.librarything.com/>
5. “BookLikes - Blog platform designed for book lovers,” booklikes.com. <http://booklikes.com/>
6. “Litsy,” Litsy. <https://www.litsy.com/>
7. “BookBub: Get ebook deals, handpicked recommendations, and author updates,” www.bookbub.com. <https://www.bookbub.com/>
8. “Libib.com - Home library management web app,” www.libib.com. <https://www.libib.com/>
9. “Goodreads Help,” help.goodreads.com. <https://help.goodreads.com/s/article/Does-Goodreads-support-the-use-of-APIs>
10. “Developer Center / APIs | Open Library,” openlibrary.org. <https://openlibrary.org/developers/api>
11. “Google Books APIs,” Google Developers. <https://developers.google.com/books> R. Mitchell, Web Scraping with Python. “O’Reilly Media, Inc.,” 2018.
12. “How to select a data provider in order to conduct a successful study,” www.netquest.com. <https://www.netquest.com/en/blog/how-to-select-a-data-provider-in-order-to-conduct-a-successful-study>
13. C. J. Date, An Introduction to Database Systems. Addison Wesley, 2000.
14. J. Celko, Joe Celko’s SQL for smarties : advanced SQL programming. San Francisco (Calif.) ; San Diego (Calif.) ; New York (N.Y): Morgan Kaufmann Publishers, 2000

15. “MongoDB Documentation,” <https://github.com/mongodb/docs-bi-connector/blob/DOCSP-3279/source/index.txt>.
<https://www.mongodb.com/docs/>
16. “Documentation,” [cassandra.apache.org](https://cassandra.apache.org/doc/latest/). <https://cassandra.apache.org/doc/latest/>
17. Docs, “Docs,” Redis.io, 2024. <https://redis.io/docs/latest/>
18. AWS, “Amazon Relational Database Service (RDS) – AWS,” Amazon Web Services, Inc., 2019. <https://aws.amazon.com/rds/>
19. “Cloud Firestore,” Google Cloud. <https://cloud.google.com/firestore>
20. “Azure Cosmos DB | Microsoft Azure,” [azure.microsoft.com](https://azure.microsoft.com/en-us/services/cosmos-db/).
<https://azure.microsoft.com/en-us/services/cosmos-db/>
21. L. Miralles-Pechuán, H. Ponce, and L. Martínez-Villaseñor, “A novel methodology for optimizing display advertising campaigns using genetic algorithms,” *Electronic Commerce Research and Applications*, vol. 27, pp. 39–51, Jan. 2018, doi: <https://doi.org/10.1016/j.elerap.2017.11.004>.
22. P. B.Thorat, R. M. Goudar, and S. Barve, “Survey on Collaborative Filtering, Content-based Filtering and Hybrid Recommendation System,” *International Journal of Computer Applications*, vol. 110, no. 4, pp. 31–36, Jan. 2015, doi: <https://doi.org/10.5120/19308-0760>.
23. D. T. Larose and C. D. Larose, *Discovering knowledge in data : an introduction to data mining*. Hoboken: Wiley, 2014, pp 149-164.
24. Lars Eldén, *Matrix methods in data mining and pattern recognition*. Philadelphia, Pa: Society For Industrial And Applied Mathematics, 2007.
25. M. Lutz, *Learning Python*. “O’Reilly Media, Inc.,” 2013.
26. Hu, Qiang, Lei Ma, and Jianjun Zhao. "DeepGraph: A PyCharm tool for visualizing and understanding deep learning models." 2018 25th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2018.
27. Copperwaite, Matt, and Charles Leifer. *Learning flask framework*. Packt Publishing Ltd, 2015.
28. “SciPy documentation — SciPy v1.8.1 Manual,” [docs.scipy.org](https://docs.scipy.org/doc/scipy/).
<https://docs.scipy.org/doc/scipy/>
29. Scikit-Learn, “User guide: contents — scikit-learn 0.22.1 documentation,” [Scikit-learn.org](https://scikit-learn.org/stable/user_guide.html), 2019. https://scikit-learn.org/stable/user_guide.html

30. NumPy, “Overview — NumPy v1.19 Manual,” [numpy.org](https://numpy.org/doc/stable/), 2022.
<https://numpy.org/doc/stable/>
31. Miell, Ian, and Aidan Sayers. Docker in practice. Simon and Schuster, 2019.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Лістинг data_store.py

```

from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///user_library.db'
db = SQLAlchemy(app)

class Book(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    author = db.Column(db.String(50), nullable=False)
    genre = db.Column(db.String(50))
    rating = db.Column(db.Float)

    def __repr__(self):
        return f"Book('{self.title}', '{self.author}', '{self.genre}', '{self.rating}')"

@app.route('/add_book', methods=['POST'])
def add_book():
    data = request.get_json()
    new_book = Book(title=data['title'], author=data['author'], genre=data.get('genre'),
rating=data.get('rating'))
    db.session.add(new_book)
    db.session.commit()
    return jsonify({'message': 'Book added successfully!'})

@app.route('/edit_book/<int:book_id>', methods=['PUT'])
def edit_book(book_id):
    book = Book.query.get_or_404(book_id)
    data = request.get_json()
    book.title = data.get('title', book.title)
    book.author = data.get('author', book.author)
    book.genre = data.get('genre', book.genre)
    book.rating = data.get('rating', book.rating)
    db.session.commit()
    return jsonify({'message': 'Book updated successfully!'})

```

```
@app.route('/delete_book/<int:book_id>', methods=['DELETE'])
def delete_book(book_id):
    book = Book.query.get_or_404(book_id)
    db.session.delete(book)
    db.session.commit()
    return jsonify({'message': 'Book deleted successfully!'})
```

```
@app.route('/get_user_books', methods=['GET'])
def get_user_books():
    books = Book.query.all()
    book_list = []
    for book in books:
        book_data = {
            'id': book.id,
            'title': book.title,
            'author': book.author,
            'genre': book.genre,
            'rating': book.rating
        }
        book_list.append(book_data)
    return jsonify(book_list)
```

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

Лістинг content.py

```
from flask import Flask, request, jsonify
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
```

```
app = Flask(__name__)
```

```
@app.route('/cosine_similarity_predict', methods=['POST'])
def cosine_similarity_predict():
    request_data = request.get_json()
    data = request_data['data']
    user_ratings = np.array(request_data['user_ratings'])
```

```

titles = [row[0] for row in data]
authors = [row[1] for row in data]
pages = np.array([row[2] for row in data])
subjects_list = [row[3] for row in data]

author_encoder = OneHotEncoder(sparse_output=False)
author_encoded = author_encoder.fit_transform(np.array(authors).reshape(-1, 1))

subject_encoder = OneHotEncoder(sparse_output=False)
flat_subjects = [subject for sublist in subjects_list for subject in sublist]
subject_encoder.fit(np.array(flat_subjects).reshape(-1, 1))
subject_encoded = np.array(
    [subject_encoder.transform(np.array(subj).reshape(-1, 1)).sum(axis=0) for subj
in subjects_list])

features = np.concatenate([author_encoded, pages.reshape(-1, 1),
subject_encoded], axis=1)

scaler = StandardScaler()
features_normalized = scaler.fit_transform(features)

rated_indices = np.where(user_ratings != 0)[0]
rated_features_normalized = features_normalized[rated_indices]
similarity_matrix_rated_normalized =
cosine_similarity(rated_features_normalized)

predicted_ratings_normalized = np.zeros(len(user_ratings))
for i, rating in enumerate(user_ratings):
    if rating == 0: # Book not rated
        unrated_feature_normalized = features_normalized[i].reshape(1, -1)
        similarity_with_rated_normalized =
cosine_similarity(unrated_feature_normalized,
                    rated_features_normalized).flatten()

        weighted_sum = np.sum(similarity_with_rated_normalized *
user_ratings[rated_indices])
        sum_of_weights = np.sum(similarity_with_rated_normalized)
        predicted_ratings_normalized[i] = weighted_sum / (
            sum_of_weights + 1e-6) # Add small epsilon to avoid division by
zero

```

```

    predictions = {'Book {titles[idx]}': rating for idx, rating in
enumerate(predicted_ratings_normalized) if
    user_ratings[idx] == 0}

```

```

return jsonify(predicted_ratings=predictions)

```

```

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

```

Лістинг als.py

```

from flask import Flask, request, jsonify
from sklearn.preprocessing import OneHotEncoder
from scipy.sparse import csr_matrix
import numpy as np

app = Flask(__name__)

@app.route('/als_predict_ratings', methods=['POST'])
def als_predict_ratings():
    request_data = request.get_json()
    data = request_data['data']
    user_ratings = np.array(request_data['user_ratings'])

    authors = [row[0] for row in data]
    subjects = [subject for row in data for subject in row[3]]

    author_encoder = OneHotEncoder(sparse_output=False)
    author_encoded = author_encoder.fit_transform(np.array(authors).reshape(-1, 1))

    subject_encoder = OneHotEncoder(sparse_output=False)
    subject_encoded = subject_encoder.fit_transform(np.array(subjects).reshape(-1, 1))

    matrix = []
    for i, row in enumerate(data):
        author = author_encoded[i]
        rating = row[1]
        pages = row[2]
        subject = subject_encoder.transform(np.array(row[3]).reshape(-1,
1)).sum(axis=0)
        matrix.append(np.concatenate([author, [rating, pages], subject]))

```

```

matrix = np.array(matrix)

# ALS expects a sparse matrix
sparse_matrix = csr_matrix(matrix)

# ALS parameters
num_users, num_items = sparse_matrix.shape
latent_factors = 2
num_iterations = 10
lambda_reg = 0.01 # Regularization parameter

# Initialize user and item factors randomly
user_factors = np.random.rand(num_users, latent_factors)
item_factors = np.random.rand(num_items, latent_factors)

# ALS algorithm
for _ in range(num_iterations):
    # Update user factors
    for i in range(num_users):
        sparse_item_factors = csr_matrix(item_factors)
        ratings = sparse_matrix.getrow(i).toarray().flatten()
        confidence = 1 + 10 * ratings # Example of implicit feedback
        A_i = sparse_item_factors.T.dot(sparse_item_factors.multiply(confidence[:,
np.newaxis])).toarray()
        v_i = sparse_item_factors.T.dot(ratings)
        user_factors[i] = np.linalg.solve(A_i + lambda_reg * np.eye(latent_factors),
v_i)

    # Update item factors
    for j in range(num_items):
        sparse_user_factors = csr_matrix(user_factors)
        ratings = sparse_matrix.getcol(j).toarray().flatten()
        confidence = 1 + 10 * ratings # Example of implicit feedback
        A_j = sparse_user_factors.T.dot(sparse_user_factors.multiply(confidence[:,
np.newaxis])).toarray()
        v_j = sparse_user_factors.T.dot(ratings)
        item_factors[j] = np.linalg.solve(A_j + lambda_reg * np.eye(latent_factors),
v_j)

# Predict ratings for all items
predicted_ratings = np.dot(user_factors, item_factors.T)

```

```

# Only predict for books not rated by the user
unrated_indices = np.where(user_ratings == 0)[0]

# Normalize predicted ratings
min_rating = np.min(predicted_ratings)
max_rating = np.max(predicted_ratings)
predicted_ratings_normalized = 5 * (predicted_ratings - min_rating) / (max_rating
- min_rating)

predictions = {f"Book {idx}": predicted_ratings_normalized[0, idx] for idx in
unrated_indices}

return jsonify(predicted_ratings=predictions)

```

```

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

```

Лістинг data_store.py

```

from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///user_library.db'
db = SQLAlchemy(app)

class Book(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    author = db.Column(db.String(50), nullable=False)
    genre = db.Column(db.String(50))
    rating = db.Column(db.Float)

    def __repr__(self):
        return f"Book('{self.title}', '{self.author}', '{self.genre}', '{self.rating}')"

@app.route('/add_book', methods=['POST'])
def add_book():
    data = request.get_json()
    new_book = Book(title=data['title'], author=data['author'], genre=data.get('genre'),
rating=data.get('rating'))

```

```

db.session.add(new_book)
db.session.commit()
return jsonify({'message': 'Book added successfully!'})

@app.route('/edit_book/<int:book_id>', methods=['PUT'])
def edit_book(book_id):
    book = Book.query.get_or_404(book_id)
    data = request.get_json()
    book.title = data.get('title', book.title)
    book.author = data.get('author', book.author)
    book.genre = data.get('genre', book.genre)
    book.rating = data.get('rating', book.rating)
    db.session.commit()
    return jsonify({'message': 'Book updated successfully!'})

@app.route('/delete_book/<int:book_id>', methods=['DELETE'])
def delete_book(book_id):
    book = Book.query.get_or_404(book_id)
    db.session.delete(book)
    db.session.commit()
    return jsonify({'message': 'Book deleted successfully!'})

@app.route('/get_user_books', methods=['GET'])
def get_user_books():
    books = Book.query.all()
    book_list = []
    for book in books:
        book_data = {
            'id': book.id,
            'title': book.title,
            'author': book.author,
            'genre': book.genre,
            'rating': book.rating
        }
        book_list.append(book_data)
    return jsonify(book_list)

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

```

Лістинг ga.py

```

from flask import Flask, request, jsonify
import numpy as np

app = Flask(__name__)

def fitness(individual, user_ratings):
    # Calculate fitness as the sum of ratings for recommended books, excluding
    unrated books
    rated_books_mask = user_ratings > 0
    total_rating = np.sum(np.array(individual) * user_ratings * rated_books_mask)
    return total_rating

@app.route('/genetic_predict_ratings', methods=['POST'])
def genetic_predict_ratings():
    request_data = request.get_json()
    data = request_data['data']
    user_ratings = np.array(request_data['user_ratings'])

    # Genetic algorithm parameters
    num_books = len(data)
    population_size = 50
    mutation_rate = 0.01
    num_generations = 100

    # Initialize population
    population = np.random.randint(2, size=(population_size, num_books))

    # Genetic algorithm
    for generation in range(num_generations):
        # Evaluate fitness for each individual
        fitness_scores = np.array([fitness(individual, user_ratings) for individual in
population])

        # Prevent division by zero in case of all zero fitness scores
        if np.sum(fitness_scores) == 0:
            fitness_scores += 1

        # Select individuals based on their fitness scores

```

```

    selected_indices = np.random.choice(range(population_size),
size=population_size, replace=True,
                                     p=fitness_scores / np.sum(fitness_scores))

    # Perform crossover
    offspring = np.empty_like(population)
    for i in range(population_size):
        parent1_idx = selected_indices[i]
        parent2_idx = selected_indices[np.random.randint(population_size)]
        crossover_point = np.random.randint(num_books)
        offspring[i, :] = np.concatenate(
            (population[parent1_idx, :crossover_point], population[parent2_idx,
crossover_point:]))

    # Perform mutation
    mutation_mask = np.random.rand(population_size, num_books) < mutation_rate
    offspring ^= mutation_mask

    # Replace the old population with the new generation of individuals
    population = offspring

    # Extract recommendations from the best individual
    best_individual = population[np.argmax([fitness(ind, user_ratings) for ind in
population])]
    recommendations = np.where(best_individual == 1)[0]

    # Predict ratings for books with 0 rating
    predicted_ratings = user_ratings.copy()
    unrated_indices = np.where(user_ratings == 0)[0]

    for idx in unrated_indices:
        # Temporarily assign a hypothetical rating to the unrated book
        hypothetical_ratings = user_ratings.copy()
        hypothetical_ratings[idx] = 5 # Let's assume the hypothetical rating is the
maximum rating (5)

        # Calculate the fitness with the hypothetical rating
        hypothetical_fitness = fitness(best_individual, hypothetical_ratings)

        # Predict the rating based on the change in fitness, capped at the maximum
rating (5)
        predicted_rating = hypothetical_fitness - fitness(best_individual, user_ratings)

```

```

    predicted_ratings[idx] = min(predicted_rating, 5)

# Normalize predicted ratings
min_rating = np.min(predicted_ratings)
max_rating = np.max(predicted_ratings)
predicted_ratings_normalized = 5 * (predicted_ratings - min_rating) / (max_rating
- min_rating)

    predictions = {f"Book {idx}": predicted_ratings_normalized[idx] for idx in
unrated_indices}

    return jsonify(predicted_ratings=predictions)

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

```

Лістинг k-nn.py

```

from flask import Flask, request, jsonify
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression
import numpy as np

app = Flask(__name__)

@app.route('/lr_predict_ratings', methods=['POST'])
def lr_predict_ratings():
    request_data = request.get_json()
    data = request_data['data']
    user_ratings = np.array(request_data['user_ratings'])

    titles = [row[0] for row in data]
    authors = [row[1] for row in data]
    pages = np.array([row[2] for row in data])
    subjects = [subject for row in data for subject in row[3]]

    author_encoder = OneHotEncoder(sparse_output=False)
    author_encoded = author_encoder.fit_transform(np.array(authors).reshape(-1, 1))

    subject_encoder = OneHotEncoder(sparse_output=False)
    unique_subjects = list(set(subjects))

```

```

subject_mapping = {subject: idx for idx, subject in enumerate(unique_subjects)}
subjects_grouped = [subject_mapping[subject] for subject in subjects]
subject_encoded =
subject_encoder.fit_transform(np.array(subjects_grouped).reshape(-1, 1))

min_rows = min(author_encoded.shape[0], subject_encoded.shape[0])
author_encoded = author_encoded[:min_rows]
subject_encoded = subject_encoded[:min_rows]

features = np.concatenate([author_encoded, pages.reshape(-1, 1),
subject_encoded], axis=1)

model = LinearRegression()
model.fit(features[user_ratings != 0], user_ratings[user_ratings != 0])

predicted_ratings = model.predict(features[user_ratings == 0])

predictions = {f"Book {titles[idx]}": rating for idx, rating in
enumerate(predicted_ratings)}

return jsonify(predicted_ratings=predictions)

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

```

Лістинг nmf.py

```

from flask import Flask, request, jsonify
from sklearn.decomposition import NMF
import numpy as np
from sklearn.preprocessing import OneHotEncoder

app = Flask(__name__)

@app.route('/predict_ratings', methods=['POST'])
def predict_ratings():
    request_data = request.get_json()
    data = request_data['data']
    user_ratings = np.array(request_data['user_ratings'])

    authors = [row[0] for row in data]

```

```

subjects = [subject for row in data for subject in row[3]]

author_encoder = OneHotEncoder(sparse_output=False)
author_encoded = author_encoder.fit_transform(np.array(authors).reshape(-1, 1))

subject_encoder = OneHotEncoder(sparse_output=False)
subject_encoded = subject_encoder.fit_transform(np.array(subjects).reshape(-1, 1))

matrix = []
for i, row in enumerate(data):
    author = author_encoded[i]
    rating = row[1]
    pages = row[2]
    subject = subject_encoder.transform(np.array(row[3]).reshape(-1,
1)).sum(axis=0)
    matrix.append(np.concatenate([author, [rating], [pages], subject]))

matrix = np.array(matrix)

# Apply NMF
nmf = NMF(n_components=2)
W = nmf.fit_transform(matrix)
H = nmf.components_

# Reconstruct the matrix
reconstructed_matrix = np.dot(W, H)

# Isolate the part of the matrix corresponding to ratings
rating_indices = author_encoded.shape[1]
predicted_ratings = reconstructed_matrix[:, rating_indices]

# Only predict for books not rated by the user
unrated_indices = np.where(user_ratings == 0)[0]

# Normalize predicted ratings
min_rating = np.min(predicted_ratings)
max_rating = np.max(predicted_ratings)
predicted_ratings_normalized = 5 * (predicted_ratings - min_rating) / (max_rating
- min_rating)

predictions = {f"Book {idx}": predicted_ratings_normalized[idx] for idx in
unrated_indices}

```

```
return jsonify(predicted_ratings=predictions)
```

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

Лістинг pca.py

```
from flask import Flask, request, jsonify
from sklearn.decomposition import PCA
import numpy as np
from sklearn.preprocessing import OneHotEncoder

app = Flask(__name__)

@app.route('/pca_predict_ratings', methods=['POST'])
def pca_predict_ratings():
    request_data = request.get_json()
    data = request_data['data']
    user_ratings = np.array(request_data['user_ratings'])

    authors = [row[0] for row in data]
    subjects = [subject for row in data for subject in row[3]]

    author_encoder = OneHotEncoder(sparse_output=False)
    author_encoded = author_encoder.fit_transform(np.array(authors).reshape(-1, 1))

    subject_encoder = OneHotEncoder(sparse_output=False)
    subject_encoded = subject_encoder.fit_transform(np.array(subjects).reshape(-1, 1))

    matrix = []
    for i, row in enumerate(data):
        author = author_encoded[i]
        rating = row[1]
        pages = row[2]
        subject = subject_encoder.transform(np.array(row[3]).reshape(-1, 1)).sum(axis=0)
        matrix.append(np.concatenate([author, [rating], [pages], subject]))

    matrix = np.array(matrix)
```

```

# Apply PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(matrix)

# Reconstruct the matrix (approximation)
reconstructed_matrix = pca.inverse_transform(principal_components)

# Isolate the part of the matrix corresponding to ratings
rating_indices = author_encoded.shape[1] + 1 # Assuming rating is the second
feature after one-hot encoded authors
predicted_ratings = reconstructed_matrix[:, rating_indices]

# Only predict for books not rated by the user
unrated_indices = np.where(user_ratings == 0)[0]

# Normalize predicted ratings
min_rating = np.min(predicted_ratings)
max_rating = np.max(predicted_ratings)
predicted_ratings_normalized = 5 * (predicted_ratings - min_rating) / (max_rating
- min_rating)

predictions = {f"Book {idx}": predicted_ratings_normalized[idx] for idx in
unrated_indices}

return jsonify(predicted_ratings=predictions)

```

```

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

```

Лістинг svd.py

```

from flask import Flask, request, jsonify
from sklearn.preprocessing import OneHotEncoder
import numpy as np

app = Flask(__name__)

@app.route('/svd_predict_ratings', methods=['POST'])
def svd_predict_ratings():
    request_data = request.get_json()
    data = request_data['data']

```

```

user_ratings = np.array(request_data['user_ratings'])

authors = [row[0] for row in data]
subjects = [subject for row in data for subject in row[3]]

author_encoder = OneHotEncoder(sparse_output=False)
author_encoded = author_encoder.fit_transform(np.array(authors).reshape(-1, 1))

subject_encoder = OneHotEncoder(sparse_output=False)
subject_encoded = subject_encoder.fit_transform(np.array(subjects).reshape(-1, 1))

matrix = []
for i, row in enumerate(data):
    author = author_encoded[i]
    rating = row[1]
    pages = row[2]
    subject = subject_encoder.transform(np.array(row[3]).reshape(-1,
1)).sum(axis=0)
    matrix.append(np.concatenate([author, [rating, pages], subject]))

matrix = np.array(matrix)

# Apply SVD
U, S, Vt = np.linalg.svd(matrix, full_matrices=False)

# Reconstruct the matrix
S_matrix = np.diag(S)
reconstructed_matrix = np.dot(np.dot(U, S_matrix), Vt)

# Predicted ratings for all books
predicted_ratings = np.dot(np.dot(U, S_matrix), Vt)

# Only predict for books not rated by the user
unrated_indices = np.where(user_ratings == 0)[0]

# Normalize predicted ratings
min_rating = np.min(predicted_ratings)
max_rating = np.max(predicted_ratings)
predicted_ratings_normalized = 5 * (predicted_ratings - min_rating) / (max_rating
- min_rating)

predictions = {f"Book {idx}": predicted_ratings_normalized[0, idx] for idx in

```

```
unrated_indices}  
  
    return jsonify(predicted_ratings=predictions)  
  
if __name__ == '__main__':  
    app.run(debug=True)
```