НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____2016 р.

Дипломна робота

___першого (бакалаврського)_____**рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності 7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код та назва спеціальності)

на тему: Дослідження технології побудови рекомендаційних систем для інтернет-маркетингу

Виконав (-ла): студент (-ка) _4__ курсу, групи _ДА-22__
(шифр групи)

Мазурік Олексій Юрійович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник _____ас. Гречко А.Е._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Економічний проф., д.е.н., Семенченко Н.В.__ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____доц., к.ф.-м.н., Подколзін Г.Б. _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль _____ст. викладач Бритов О.А_____ _____
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2016 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2016 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Мазуріку Олексію Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Дослідження технології побудови рекомендаційних систем для інтернет-маркетингу

Керівник проекту (роботи) Гречко Анна Едуардівна, ас.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання),

затверджені наказом по університету від 12 травня 2016 р. № 50-ст

2. Строк подання студентом проекту (роботи) 08.06.2016

3. Вихідні дані до проекту (роботи)

Операційна система OS X
Частота процесору 2.2 ГГц
Середовище розробки – RubyMine
Бібліотеки, що використовувалися: Ruby on Rails, Recommendable
Системи збірки, розгортання в мережі Інтернет та тестування: Assets Pipeline, Capistrano, RSpec

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Проаналізувати типи рекомендаційних алгоритмів.

2. Проаналізувати рекомендаційні технології відомих інтернет-сервісів.
3. Розробити веб-додаток із вбудованою системою рекомендацій
4. Проаналізувати результати роботи рекомендаційної системи.
5. Виконати економічний аналіз програмного продукту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Схема роботи системи з рекомендаціями – плакат.
2. Порівняльна таблиця базових алгоритмів для створення рекомендаційних систем – плакат.
3. Зведена таблиця РС Amazon та eBay – плакат.
4. Результати роботи РС - плакат.

6. Консультанти розділів проекту (роботи)*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Семенченко Н.В., професор		

7. Дата видачі завдання 01.02.2016

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Збір інформації	15.02.2016	
3	Вивчення базових рекомендаційних алгоритмів та вибір варіанту для розробки	28.02.2016	
4	Аналіз сервісів, що використовують РС	10.03.2016	
5	Розробка алгоритму та структури системи	15.03.2016	
6	Розробка програми	25.03.2016	
7	Тестування додатку та отримання результатів	25.04.2016	
8	Оформлення дипломної роботи	20.05.2016	
9	Отримання допуску до захисту та подача роботи в ДЕК	08.06.2016	

Студент _____ О.Ю. Мазурік
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ А.Е. Гречко
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломного проекту (роботи).

АНОТАЦІЯ

бакалаврської дипломної роботи Мазуріка Олексія Юрійовича на тему: “Дослідження технології побудови рекомендаційних систем для інтернет-маркетингу”

Дана дипломна робота присвячена дослідженню методів побудови рекомендаційних систем. Метою роботи є широкий аналіз існуючих підходів до створення систем, що прогнозують рекомендації, та реалізація додатку із вбудованою системою рекомендацій.

У роботі було розглянуто відомі базові підходи, що виористовуються при розробці системи, що надає рекомендації, та методи кореляції результатів роботи таких систем. Проаналізовано реалізацію цих підходів на прикладі відомих інтернет-сервісів з каталогом товарів. Розроблено веб-додаток з використанням Rails та імплементацією рекомендаційної системи на базі колаборативної фільтрації для більш ефективного залучення користувачів до роботи з реалізованим веб-додатком. Також було детально проаналізовано результати роботи створеної рекомендаційної системи. Отримані результати дослідження пропонується використовувати в якості методичного матеріалу розробниками ПЗ під час проектування та розробки рекоменаційних систем.

Загальний обсяг роботи: 113 сторінок, 1 додаток на 14 сторінок, 21 рисунок, 13 таблиць, 17 посилань.

Ключові слова: рекомендаційні системи, рекомендації, прогнозування, інтернет-маркетинг, колаборативна фільтрація, контентна фільтрація, веб-додаток, Rails.

АННОТАЦИЯ

бакалаврской дипломной работы Мазурика Алексея Юрьевича на тему: “Исследование технологии построения рекомендационных систем для интернет-маркетинга”

Данная дипломная работа посвящена исследованию методов построения рекомендательных систем. Целью работы является широкий анализ существующих подходов к созданию систем, которые прогнозируют рекомендации, и реализация приложения со встроенной системой рекомендаций.

В работе были рассмотрены базовые подходы, которые используются при разработке рекомендательной системы, и методы корреляции результатов работы таких систем. Была проанализирована реализация данных алгоритмов на примере известных интернет-сервисов с каталогом товаров. Разработано веб-приложение с использованием Rails и встроенной системой рекомендаций на базе коллаборативной фильтрации для более эффективного привлечения пользователей к работе с данным сервисом. Также были детально проанализированы результаты работы созданной рекомендательной системы. Полученные результаты исследования предлагается использовать в качестве методических материалов разработчикам ПО во время проэктирования и разработки рекомендательных систем.

Общий объем работы: 113 страниц, 1 приложение на 14 страниц, 21 рисунок, 13 таблиц, 17 ссылок.

Ключевые слова: рекомендационные системы, рекомендации, интернет-маркетинг, коллаборативная фильтрация, контентная фильтрация, веб-приложение на Rails.

ANNOTATION

for the bachelor thesis of Mazurik Oleksii Yurievich on “Internet marketing recommender system building technology research”

Thesis is devoted to the recommender system building methods research. The aim is to deeply analyse basic approaches for building systems that predict recommendations and create web-app with built-in recommender system.

In the course of the thesis basic methods to the building of recommender systems and approaches to their results' correlation were investigated. Also implementations of this systems were examined by the example of popular web-apps with products catalogs. Rails web-app with built-in recommender system that uses collaborative filtering to predict goods and attract users was developed. Results of this recommender system were analysed. The results of research are encouraged to use as teaching materials for software developers during designing and development of recommender systems.

Total volume of work: 113 pages, 1 appendix for 14 pages, 21 figures, 13 tables, 17 links.

Keywords: recommender systems, recommendations, prognostication, internet marketing collaborative filtering, content-based filtering, Rails app.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

РС – Рекомендаційна Система

КФ – Колаборативна Фільтрація

CF – Collaborative Filtering

CBF – Content-Based Filtering

SVD – Singular Value Decomposition

TF-IDF – Term Frequency - Inverse Document Frequency

ВСТУП

Протягом значного проміжку часу в усьому світі швидкими темпами збільшується кількість інформації. Люди кожного дня сприймають та фільтрують вхідний потік інформації, що надходить з різних джерел: робота, побутові проблеми, популярні джерела інформації тощо. Після винайдення мережі Інтернет кількість такої інформації стала стрімко зростати, з'явилася велика кількість сервісів для надання користувачам всього необхідного для комфортного життя.

За останню декаду набули значної популярності інтернет-сервіси, що пропонують товари всіх можливих видів (інтернет-магазини), інформацію на будь-який смак (інтернет-журнали, новини, книги, статті) тощо. Користувачу стало надзвичайно важко орієнтуватися в каталогах товарів та списках статей, навіть із вбудованим пошуком та фільтрацією, оскільки дуже важко зробити вибір при настільки великому об'ємі інформації.

Рекомендаційні системи з'явилися на сучасному ринку IT як механізм для заміни статичному списку рекомендацій при пошуку або покупках на веб-сайтах. Ці системи формують рейтинговий перелік об'єктів (товарів, фільмів, музичних композицій) на основі різних критеріїв: релевантність, популярність, історія оцінок тощо.

Такі системи почали широко використовувати існуючі інтернет-компанії в рамках інтернет-маркетингу. За допомогою прогнозування рекомендацій вони мають на меті збільшити залученість користувачів до конкретного сервісу. Також, при розробці рекомендаційної системи з релевантними рекомендаціями, що заслужили довіру користувачів, можна розміщувати серед цих рекомендацій інші товари, що рекламуються.

При розробці рекомендаційних систем зазвичай розробники стикаються з рядом проблем прогнозування:

- *Розрідженість даних* (більшість користувачів не ставить оцінки товарам, отже дані з попередніми оцінками являють собою розріджену матрицю).
- *Холодний старт* (робота з новими користувачами або товарами).
- *Синонімія* (проблема розпізнавання схожих товарів з різними назвами). [2]
- *Шахрайство* (цілеспрямоване завищення рейтингів певних товарів їх власниками).
- *Розмаїття* (при великій вибірці нові або маловідомі товари мають низькі позиції в рейтинговому списку).
- *Білі ворони* (унікальні користувачі, смаки яких дуже важко обробити, оскільки вони не співпадають зі смаками відокремлених типів). [3]

Отже, як можна помітити, проблем створення системи прогнозування для інтернет-сервісів чимало, тож дуже цікаво подивитися як саме різні підходи до створення рекомендаційних систем вирішують ці проблеми та покращують якість рекомендацій.

Метою даного дипломного проекту є аналіз існуючих рекомендаційних систем для інтернет-маркетингу, дослідження рекомендаційних алгоритмів та виявлення їх переваг та недоліків. Також, в якості прикладу застосування системи прогнозування рекомендацій буде розроблено веб-додаток за допомогою Rails, PostgreSQL та Redis, де буде протестовано якість рекомендацій.

Отримані результати дослідження пропонується використовувати в якості методичного матеріалу розробниками ПЗ під час проектування та розробки рекоменаційних систем.

1 АНАЛІЗ РЕКОМЕНДАЦІЙНИХ АЛГОРИТМІВ

Рекомендаційна система – це система, що рекомендує товари користувачам серед величезного потоку інформації, в залежності від їх потреб. Товари – елементи конкретного сервісу, до яких користувач має інтерес: фільми, ресторани, книги, статті і т.ін. Інтереси користувачів можуть бути представлені декількома способами: із застосуванням оцінок, які користувачі надають товарам або за допомогою ключових слів кожного товару. [1]

Щоб зберігати вподобання користувачів стосовно товарів, РС використовують профілі користувачів. У більшості РС профіль користувача містить набори оцінок та/або ключових слів (тегів). Оцінки, надані користувачами товарам, можуть належати різним проміжкам (0-1, 1-5, 1-10): чим вищий рейтинг, тим більше конкретний товар сподобався користувачу. Після кожного оцінювання все рейтинги користувача агрегуються через ряд обчислень, вимірюються схожість користувачів, а потім прогнозуються рекомендації для даного користувача. Ключові слова автоматично підвантажуються з текстів або товарів, які користувачі проглядали або оцінювали в минулому. Вони також можуть мати ваги в залежності від того, на скільки користувач оцінив конкретне слово, або більш значущі слова матимуть більшу вагу, ніж менш значущі (алгоритм TF-IDF). Після цього тексти (товари) зіставляються з профілем користувача та найбільш відповідающі йому – рекомендуються.

Рейтинги можуть бути явними та неявними. Явна оцінка – це оцінка, якою користувач показав зацікавленість даним товаром в межах своєї системи оцінювання. Неявні рейтинги вираховуються з історії покупок або поведінки користувачів. До їх переваг можна віднести зниження навантаження на користувача оцінюванням товарів. Джерелом неявних рейтингів можуть бути час, витрачений на читання статті, посилання на товар в інших джерелах

(наприклад алгоритм ранжування сторінок Google). Інші індикатори поведінки перегляду, як рух курсору, ввід клавіатури та швидкість прокрутки сторінки, також були досліджені в якості неявних показників інтересу та показали непогані результати.

Загальна архітектура рекомендаційної системи (рис. 1.1) може бути представлена наступним чином:

1) Довідкові дані (background data) – системна інформація про товари сервісу, що збирається, як правило, в фоновому режимі.
2) Вхідні дані (input data) – інформація, яку користувач вносить в систему, щоб отримати рекомендації.
3) Рекомендаційний алгоритм - алгоритм, що комбінує системну інформацію та вхідні дані для отримання рекомендацій.

Типові системи в якості довідкових даних використовують профілі користувачів, а в якості вхідних – дії користувача (оцінювання товарів, час, проведений на сторінці, тощо). [10]

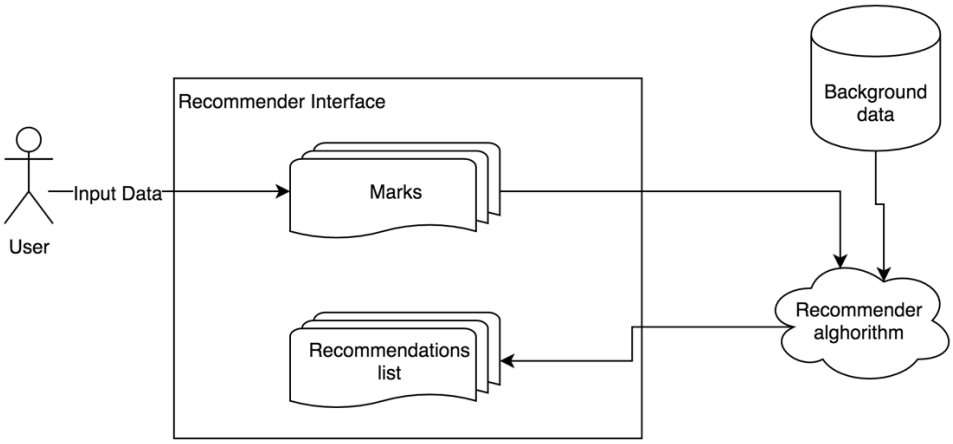

Рисунок 1.1 – Базова архітектура рекомендаційної системи

1.1 Базові підходи розробки рекомендаційних систем

У більшості рекомендаційних систем використовується один з двох базових підходів: Колаборативна фільтрація (collaborative filtering) та контентна фільтрація (content-based filtering). Також існує клас підходів, що базуються на поєднанні двох основних – гібридна фільтрація (hybrid filtering).

1.1.1 Колаборативна фільтрація

Метод прогнозу в рекомендаційних системах, який використовує відомі переваги (оцінки) групи користувачів для прогнозування невідомих переваг (оцінок) іншого користувача. За допомогою цього алгоритму будується певна таблиця користувачів, які групуються за схожістю, та прогнозуються результати для інших користувачів. [2]

Колаборативна фільтрація прогнозує рекомендації, засновані на моделі попередньої поведінки користувача. Ця модель може бути побудована виключно на основі поведінки цього користувача або - що більш ефективно - з урахуванням поведінки інших користувачів з подібними характеристиками. У тих випадках, коли колаборативна фільтрація бере до уваги реакцію інших користувачів, вона використовує знання про групу (group knowledge) для вироблення рекомендацій на основі схожості користувачів. По суті рекомендації базуються на автоматичній взаємодії множини користувачів і на виділені (методом фільтрації) тих користувачів, які демонструють схожі уподобання або шаблони поведінки.

Наприклад, при створенні веб-блогу, на якому необхідно запровадити рекомендаційну систему на основі інформації від багатьох користувачів, які підписуються на блоги і читають їх, можна згрупувати цих користувачів за їх інтересами. Можна об'єднати в одну групу користувачів, які читають кілька однакових блогів і за цією інформацією ідентифікувати найпопулярніші блоги серед тих, які читають учасники цієї групи. Потім конкретному користувачу з

цієї групи будуть рекомендуватися найпопулярніший блог з тих, на які він ще не підписаний. [6]

На рис. 1.2 рядки відповідають блогам, а стовпці - користувачам. Осередок на перетині рядка блогу і строки користувача містить кількість статей, прочитаних цим користувачем в цьому блозі. Кластеризація користувачів на основі читацьких вподобань (наприклад, за допомогою алгоритму найближчих сусідів) дозволяє виділити два кластери, кожен з яких містить по два користувача. Варто звернути увагу на схожість читацьких звичок у членів кожного кластера: Кластер 1 утворюють учасники з іменами Марк і Еліза, кожен з яких прочитав по кілька статей по тематиці "Linux" і за тематикою "Хмарні обчислення". Кластер 2 утворюють учасники з іменами Меган і Джилл, кожен з яких прочитав по кілька статей по тематиці "Java-технології" і за тематикою "Agile-технології".

Блоги	Марк	Меган	Элиза	Джилл
Linux	13	3	11	-
Продукты с открытым кодом	10	-	-	3
Облачные вычисления	6	1	9	-
Java-технологии	-	6	-	9
Agile-технологии	-	7	1	8
	Количество статей, прочитанных данным пользователем			
Кластер	1	2	1	2

Рисунок 1.2 – Приклад колаборативної фільтрації [6]

Тепер можна ідентифікувати певні відмінності в рамках кожного кластера і сформувати рекомендації. У кластері 1 Марк прочитав 10 статей з блогу “Продукти з відкритим вихідним кодом”, а Еліза не прочитали жодної такої статті; Еліза прочитала одну статтю в блозі з Agile-технологіями, а Марк не прочитав жодної такої статті. Таким чином, виходячи з рис. 1, Елізі можна порекомендувати блог “Продукти з відкритим вихідним кодом”. Для Марка неможливо сформувати ніяких рекомендації, оскільки невелика різниця між ним і Елізою за кількістю прочитань блогу по Agile-технологій з великою

ймовірністю буде відфільтрована. У кластері 2 Джилл прочитала три статті в блозі “Продукти з відкритим вихідним кодом”, а Меган не прочитали жодної такої статті; Меган прочитала 3 статті в блозі по Linux, а Джилл не прочитала жодної такої статті. Таким чином, для учасників кластера 2 можна сформувати наступні дві рекомендації: учаснику Джилл рекомендується блог по Linux, а учаснику Меган - блог “Продукти з відкритим вихідним кодом”. [6]

Інший спосіб розгляду цих відносини заснований на їх схожості та відмінності, як ілюструє діаграма Венна на рисунку 1.3. Схожість визначають, за якими ознаками (за допомогою відповідного алгоритму) слід згрупувати користувачів. Відмінності - це можливості, які можуть бути використані для вироблення рекомендацій - наприклад, за допомогою застосування фільтра популярності.

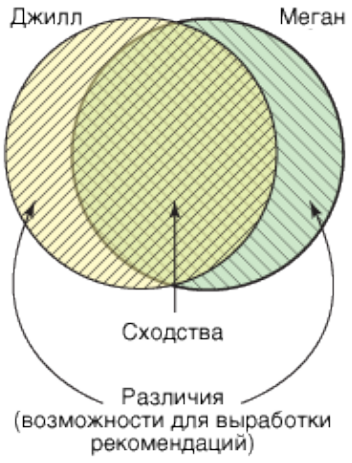

Рисунок 1.3 – Схожості та відмінності, що використовуються в алгоритмах колаборативної фільтрації [6]

Хоча на рисунку 1.3 показана спрощена картина (з впливом розрідженості даних через використання лише двох зразків), таке уявлення досить зручне.

Переваги: швидка робота алгоритмів (K-based та ін.) - мала кількість ітерацій; прості в реалізації.

Недоліки: не вирішені проблеми холодного старту, шахрайства, нема що рекомендувати новим або нетиповим користувачам; розріджені матриці оцінок (іноді неможливо зробити прогноз).

1.1.1.1 Алгоритм SVD

Майже всі алгоритми колаборативної фільтрації мають такі недоліки, як холодний старт, тривіальність результатів рекомендацій тощо. Одним з досить нових алгоритмів, який знижує вплив типових проблем колаборативної фільтрації, виявився SVD алгоритм, який було створено саме для покращення результатів звичайних алгоритмів. [3]

Постановка завдання

Маємо множину користувачів $u \in U$, множину об'єктів (фільми, треки, товари тощо) $i \in I$, та множину подій (дії, які користувачі виконують над об'єктами) $(r_{ui}, u, i, \dots) \in D$. Кожна подія задається користувачем u, об'єктом i, своїм результатом r_{ui}, а також, можливо, ще іншими характеристиками. Наша ціль:

- передбачити перевагу:

$$\hat{r}_{ui} = Predict(u, i, \dots) \approx r_{ui} \tag{1.1}$$

- персональні рекомендації:

$$u \to (i_1, \dots, i_K) = Recommend_K(u, \dots). \tag{1.2}$$

- схожі об'єкти:

$$u \to (i_1, \dots, i_M) = Similar_m(i, \dots). \tag{1.3}$$

Таблиця 1.1 – Оцінки користувачів

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
User 1	5	4	5			
User 2	4		5			
User 3		3	5		4	
User 4				3	4	
User 5			4	2	4	
User 6	3					5

Таблиця 1.2 - Завдання для прогнозування

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
User 1	5	4	5			
User 2	4	?	5			
User 3		3	5	?	4	
User 4			?	3	4	
User 5	?		4	2	4	?
User 6	3					5

Основна ідея колаборативної фільтрації: схожим користувачам зазвичай подаються схожі об’єкти. [2] Найпростіший метод вирішення даної задачі:

- Оберемо деяку умовну міру схожості користувачів за їх історією оцінок $sim(u, v)$.

- Поділимо користувачів на групи (кластери) таким чином, щоб схожі користувачі знаходилися в одному кластері: $u \to F(u)$.
- Оцінку користувача об'єкту будемо передбачувати як середню оцінку кластера для цього об'єкта за формулою:

$$\widehat{r_{ui}} = \frac{1}{|F(u)|} \cdot \sum_{v \in F(u)} r_{vi} \tag{1.4}$$

Проблеми алгоритму:

- Рекомендація новим користувачам. Для таких користувачів не знайдется відповідного кластера зі схожими на них користувачами.
- Рекомендація для нетипового користувача. Ми ділимо усіх користувачів на якісь класи (шаблони).
- Якщо в кластері ніхто не оцінив об'єкт, то зробити передбачення не вийде.

Два підходи, щодо покращення роботи алгоритму:

User-based

Підхід, що спирається на припущення, що вподобання користувача схожі на вподобання схожих користувачів. Замінемо жорстку кластеризацію на формулу:

$$\widehat{r_{ui}} = \underline{r_u} + \frac{\sum_{v \in U_i} sim(u,v)\left(r_{vi} - \underline{r_v}\right)}{\sum_{v \in U_i} sim(u,v)} \tag{1.5}$$

Проблеми підходу:

- Нові/нетипові користувачі.
- Проблема розмаїття. [1]

Item-based

Підхід, в якому вважається, що користувачу сподобаються схожі товари з тим, що він вже обрав. Жорстка кластеризація буде замінена наступним чином:

$$\widehat{r_{ui}} = \underline{r_i} + \frac{\sum_{j \in I_u} sim(i,j)(r_{uj} - \underline{r_j})}{\sum_{j \in I_u} sim(i,j)} \tag{1.6}$$

Недоліки:

- Холодний старт.
- Рекомендації часто тривіальні. [1]

Алгоритм SVD

SVD (Singular Value Decomposition) - сингулярне розкладання матриці. Теорема про сингулярний розклад стверджує, що у будь-якої матриці *A* розміру n на m існує розкладання в добуток трьох матриць: *U*, *Σ* та V^t:

$$\underset{n \times m}{A} = \underset{n \times n}{U} \times \underset{n \times m}{\Sigma} \times \underset{m \times m}{V^T} \tag{1.7}$$

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

$$UU^T = I_n, VV^T = I_m \tag{1.8}$$

$$\Sigma = diag(\lambda_1, \dots, \lambda_{\min(n,m)}), \lambda_1 \geq \cdots \geq \lambda_{\min(n,m)} \geq 0 \tag{1.9}$$

Лямбди в формулі розташовані за спаданням. Доведення теореми буде опущене.

Окрім звичайного розкладання існує ще усічене, коли з усіх лямбд залишаються лише перші ***d*** чисел, а інші вважються рівними нулю.

$$\lambda_1, \dots, \lambda_{\min(n,m):=0} \tag{1.10}$$

Це рівносильно тому, що в матрицях ***U*** і ***V*** залишати лише **d** стовпців, а матрицю *Σ* обрізати до квадратної розмірністю $\boldsymbol{d \times d}$.

$$\underset{n\times m}{A'} = \underset{n\times d}{U'} \times \underset{d\times d}{\Sigma'} \times \underset{d\times m}{V'^T} \tag{1.11}$$

Виявляється, що на практиці нова матриця A' дуже добре наближає вихідну матрицю ***A*** та, тим більше, є найкращим наближенням. [3]

SVD для рекомендацій

Отримана формула спрощується, вважаючи добуток перших двох матриць за одну:

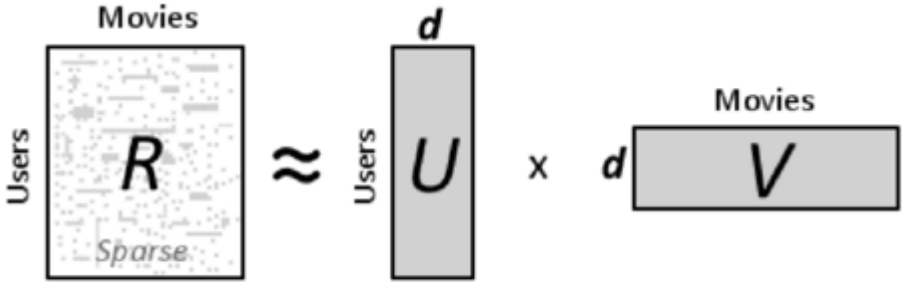

Рисунок 1.4 – Приклад застосування SVD для рекомендацій [5]

$$\widehat{r_{ui}} = f(p_u, q_i) \tag{1.12}$$

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

Наприклад, у векторі користувача на першому місці буде стояти параметр, який відповідає за стать користувача (хлопчик чи дівчинка), а на другому - його вік. У фільмів на аналогічному першому місці буде стояти параметр, який вказує на те, чи цей фільм подобається більше хлопчикам/дівчатам, а інший - для якої вікової категорії цей фільм більше підходить. З цього видно, що цей алгоритм дозволяє не тільки передбачувати оцінки, також можливо передбачувати скриті інтереси користувачів та параметри об’єктів. [5]

Підхід має ряд проблем. По-перше, не повністю відома матриця оцінок ***R***,

а, по-друге, розклад матриці на добуток не єдиний, тому не факт, що на першому місці вектора $\boldsymbol{p}$ буде стояти параметр, що відповідає за стать. Подолати ці проблеми можуть допомогти сучасні методи оптимізації та машинне навчання.

Матриця оцінок невідома, тому однозначний SVD розклад знайти неможливо. Але створити рекомендаційну систему, яка буде працювати схожим чином - можна. Отже, необхіднознайти деякий вектор користувача та вектор фільму з різними параметрами. Оскільки задані попередні оцінки користувачів, їх можна використати як навчальну вибірку. А для знаходження результатів, близьких до існуючих, необхідно обрати оптимальні параметри моделей фільмів та користувачів. Для покращення результатів машинного навчання часто використовують регуляризатори.

Пошук оптимальних параметрів можна прискорити використавши вже відомі алгоритми градієнтного спуску та метод найменших квадратів. [4]

1.1.2 Контентна фільтрація

Тематична фільтрація формує рекомендацію на базі поведінки користувача. Наприклад, цей підхід може використовувати ретроспективну інформацію про перегляди (які блоги читає користувач і характеристики цих блогів). Якщо який-небудь користувач зазвичай читає статті про Linux або регулярно залишає коментарі в блогах з проектування програмного забезпечення, то тематична фільтрація може використовувати цю ретроспективну інформацію для виявлення подібного контенту і пропозиції такого контенту як рекомендованого для цього користувача (статті в блогах по Linux або в інших блогах з проектувння програмного забезпечення). Цей контент може бути визначений в ручному режимі або завантажений автоматично на базі інших методів подібності.

Повертаючись до рис. 1.2 розглянеться користувач з ім'ям Еліза. Припустимо, вирішено застосувати ранжування блогів, яке вказує, що користувачі, котрі читали статті по Linux, можуть зацікавитися хмарними обчисленнями і продуктами з відкритим вихідним кодом. В цьому випадку можна запропонувати Елізі, на основі її поточних читацьких звичок, блог “Продукти з відкритим вихідним кодом”. Цей підхід, проілюстрований на рис. 1.5, спирається виключно на контент, до якого звертається один користувач, а не на реакцію інших користувачів в системі. [6]

Рисунок 1.5 – Ранжований список відмінностей, що використовується в контентній фільтрації [6]

Діаграму Венна (рис. 1.3) можна застосувати і в цьому випадку: якщо одна сторона - це користувач Еліза, а інша - це ранжований набір подібних блогів, то подібності ігноруються (оскільки Еліза вже прочитала відповідні блоги), а ранжування відмінностей створює можливості для вироблення рекомендацій.

Переваги: більш точний результат; немає проблеми холодного старту, оскільки рекомендації базуються на моделі об’єкта, а не на попередніх оцінках користувачів.

Недоліки: “затратне” створення моделі (її побудова досить складна), невисока швидкодія алгоритмів (багато обчислень) та втрата точності при скороченні параметрів моделі.

1.1.2.1 Алгоритм TF-IDF

Алгоритм TF-IDF розшифровується як “частота терміна” (*term frequency*) – “зворотня частота зустрічання терміна в документі” (*inverse document frequency*). TF-IDF – статистичний показник, що використовується для оцінки важливості слів у контексті документа, що є частиною колекції документів. Вага (значимість) слова пропорційна кількості вживань цього слова у документі, і обернено пропорційна частоті вживання цього слова у інших документах.

Наприклад, запит “The Civil War” буде розбито на слова: “the”, “civil” та “war”. Вага слова “the” буде наближуватися до нуля, оскільки в будь-якій статті англійською мовою воно зустрічається дуже часто. Два інших слова матимуть більшу вагу при пошуку документа у колекції.

Показник TF-IDF використовується в задачах аналізу текстів та інформаційного пошуку. Його можна застосовувати як один з критеріїв релевантності документа до пошукового запиту, а також при розрахунку міри спорідненості документів при кластеризації. [11]

Обернена частота для корпуса документів (idf) розраховується за формулою:

$$idf(t_i, D) = \log \frac{|D|}{|\{d \in D: t_i \in D\}|}, \tag{1.13}$$

де $\boldsymbol{D}$ – власне корпус (множина документів $\boldsymbol{d}$), а $\boldsymbol{t_i}$ – ключове слово. Брати логарифм від цього відношення необов’язково, найголовніше – знизити збиткову популярність слова. Логарифм має сенс використовувати у випадку, коли корпус документів надзвичайно великий.

Пряма частота ключового слова розраховується наступним чином:

$$tf(t_i, d) = \frac{n_i}{\sum_k n_k}, \tag{1.14}$$

де $\boldsymbol{n_i}$ – число входжень ключового слова в документ $\boldsymbol{d}$, а сума у знаменнику – це загальна кількість слів у документів.

Для ранжування слів за важливість ми просто беремо добуток від цих чисел:

$$tf - idf(t_i, d, D) = tf(t_i, d) \cdot idf(t_i, D). \quad (1.15)$$

Неважко помітити, що обернена частота обраховується за всім корпусом, а пряма частота ключового слова залежить від документа, який ми хочемо проаналізувати. Якщо всі слова в текстах всіх документів корпусу звести до словарних форм або хоча б лематизувати, це значно збільшить швидкодію наведеного алгоритму.

Якщо обрахувати увесь корпус документів та ранжувати слова в кожному документі за нисхідним значенням tf-idf, то перші декілька слів даватимуть досить адекватний огляд всього змісту документу. Такі слова можна навіть використовувати як теги. Але не слід цього робити на сайтах новин, адже там достатньо лише заголовку, щоб зрозуміти зміст документу.

У величезній колекції документів досить імовірно, що більші документи будуть мати більшу вагу, ніж менші, оскільки частота зустрічання терміну буде значно вищою. Щоб вирішити цю проблему алгоритм TF-IDF іноді включає в себе третій компонент: коефіцієнт нормалізації, який урівнює коефіцієнт TF-IDF для документів різних розмірів. [7]

1.1.3 Гібридні алгоритми фільтрації

Гібридні підходи, які поєднують колаборативну і контентну фільтрацію, також підвищують ефективність (і складність) рекомендаційних систем. Простий приклад гібридної системи міг би використовувати підходи, що показані на рис. 1.2 і на рис. 1.5. Об'єднання результатів колаборативної і контентної фільтрації потенційно дозволяє підвищити точність рекомендації.

Гібридний підхід може бути корисний, якщо застосування колаборативної фільтрації відбувається на сильно розріджених даних (приклад холодного старту). Гібридний підхід дозволяє спочатку зважувати результати згідно контентної фільтрації, а потім зміщувати ці ваги у напрямку до колаборативної фільтрації (в міру "визрівання" доступного набору даних для конкретного користувача). [11]

1.1.3.1 Моделі гібридних систем

Таблиця 1.3 – Моделі рекомендаційних систем на базі гібридних алгоритмів

Тип	Характеристика
Зважена	Оцінки, отримані методами колаборативної та контентної фільтрації, агрегуються для отримання єдиної рекомендації.
Комутуюча	Система використовує критерії, щоб перемикатися між методами контентної та колаборативної фільтрації.
Каскадна	Рекомендація такої моделі з’являється за рахунок удосконалення рекомендації інших систем.
Комбінаційна	Критерії з різних джерел рекомендацій вкидаються в єдиний рекомендаційний алгоритм.
Нарощувальна	Вихідні дані, отримані з однієї системи, використовуються як вхідні для іншої.
Змішана	Рекомендації з різних рекомендаційних систем показуються одночасно
Мета-рівень	Навчена модель з однієї рекомендаційної системи використовується в якості системних даних для іншої системи.

Переваги: велика швидкодія; кращі результати.

Недоліки: дуже дорога розробка рекомендаційної системи, оскільки реалізація цього типу алгоритмів дуже складна; важко підтримувати, оскільки навіть незначні зміни в роботі призводять до змін роботи алгоритму. [5]

1.1.4 Порівняльна характеристика базових рекомендаційних алгоритмів

Базові рекомендаційні алгоритми мають переваги та недоліки. Їх список наведено у таблиці 1.4:

Таблиця 1.4 – Переваги та недоліки базових рекомендаційних алгоритмів

	Переваги	***Недоліки***	***Застосування***
Колаборативна фільтрація	1. Рекомендації незалежні від контенту сервісу 2. Використовується якість оцінок та смаки користувачів 3. Інтуїтивно прозорий	1. Проблема холодного старту (першого користувача) 2. Розрідженість матриці оцінок	• Блоги • Інформаційні портали
Контентна фільтрація	1. Немає проблеми холодного старту 2. Немає проблеми розрідженості даних	1. Залежна від контенту 2. Ніяк спиратися на смаки та якість оцінок користувачів 3. Вузькоспрямована	• Інтернет-магазини • Інтернет-аукціони • Блоги
Гібридна фільтрація	1. Майже відсутні базові проблеми CF та CBF 2. Висока швидкодія	1. Висока складність розробки 2. Складна підтримка 3. Вузькоспрямована	• Великі інтернет-магазини • Складні соціальні мережі

1.2 Допоміжні алгоритми, що використовуються рекомендаційними системами

Відомий підхід, застосований в конкурсі Netflix prize [17], наочно демонструє, що в рекомендаційних механізмах може бути використані велике розмаїття алгоритмів. Отримані результати можуть відрізнятися в залежності від проблеми, для якої спроектований конкретний алгоритм, і від взаємозв’язків, які можна знайти між даними. Багато з цих алгоритмів прийшли з області машинного навчання (яка є підмножиною штучного інтелекту), яка займається алгоритмами для навчання, прогнозування і прийняття рішень.

1.2.1 Кореляція Пірсона

Схожість між двома користувачами та їх атрибутами (параметри статей в блогах) може бути точно обчислено за допомогою кореляції Пірсона. Цей алгоритм вимірює лінійну залежність між двома змінними (або користувачами) як функцію їх атрибутів. Однак він не обчислює це значення на всій множині користувачів. Цю множину необхідно попередньо розбити на групи (відфільтровані за схожістю) на базі високорівневих показників схожості. [6]

1.2.2 Алгоритми кластеризації

Алгоритми кластеризації - це різновид "Навчання без вчителя" (unsupervised learning), що дозволяє виявити структуру в рядах на перший погляд випадкових (або немаркованих) даних. У загальному випадку такий алгоритм базується на виявленні подібностей між елементами (наприклад, між читачами блогу) за допомогою обчислення їх відстані від інших елементів в просторі ознак (feature space) (ознакою в просторі ознак може, наприклад, бути кількість прочитаних статей в групі блогів). Кількість незалежних ознак визначає розмірність простору ознак. Якщо елементи "близькі" один до одного, то їх можна об'єднати в один кластер. [6]

Існує безліч алгоритмів кластеризації. Найпростішим з них є алгоритм k-середніх (k-means), який розділяє елементи на k кластерів. Спочатку елементи розподіляються за цими кластерам в довільному порядку. Потім для кожного кластера обчислюється центр мас (або просто центр) як функція від його членів. Після цього перевіряється відстань кожного члена кластера від центру цього кластера. Якщо за результатами цієї перевірки член виявляється ближче до іншого кластеру, то він переміщається в цей кластер. Після перевірки всіх відстаней для всіх членів центри кластерів обчислюються наново. При досягненні стабільного стану (в процесі чергової ітерації члени не переміщувалися) набір вважається кластеризованим належним чином, і алгоритм зупиняє роботу.

Відстань між двома об'єктами може бути важкою для візуалізації. Один з поширених методів вирішення цього завдання полягає в тому, щоб розглядати кожен член кластера як багатовимірний вектор і обчислювати для нього евклідову відстань.

Існує безліч інших різновидів кластеризації, в тому числі теорія адаптивного резонансу (Adaptive Resonance Theory), нечітка кластеризація методом С-середніх (Fuzzy C-means), імовірнісна кластеризація за допомогою EM-алгоритму (Expectation-Maximization).

1.2.3 Вимірювання якості рекомендацій

Для покращення якості рекомендацій необхідно навчитися вимірювати отримані результати. Зараз досить стандартною процедурою є знаходження RMSE. Тобто квадрат різниці даних, які були отримані на навчальній вибірці, з даними, що були в тестовій вибірці. [5]

$$RMSE = \sqrt{\frac{1}{|D|} \cdot \sum_{(u,i)\in D} (\hat{r}_{ui} - r_{ui})^2} \tag{1.16}$$

Але даний підхід має декілька недоліків:

- Кожен користувач має своє бачення шкали оцінок
- Можливо мати майже ідеальний показник RMSE, але мати погану якість ранжування, і навпаки.

Додаткові властивості рекомендацій

Виявляється, що на сприйняття рекомендацій досить часто впливає не лише якість ранжування результатів. Також дуже важливими є інші фактори: різноманітність, несподіваність, новизна та багато інших. Тож при розробці рекомендаційної системи необхідно обов'язково мати на увазі ці фактори.

1.2.4 Байєсові мережі довіри (Bayesian Belief Nets)

Байєсові мережі являють собою графові моделі імовірнісних і причинно-наслідкових відносин між змінними у статистичному інформаційному моделюванні. У байесових мережах можуть органічно поєднуватися емпіричні частоти появи різних значень змінних, суб'єктивні оцінки «очікувань» і теоретичні уявлення про математичні ймовірності тих чи інших наслідків з апріорної інформації. Це є важливою практичним перевагою і відрізняє їх від інших методів інформаційного моделювання.

Для рекомендацій вони можуть бути застосовані наступним чином. Наприклад, для великого інтернет-магазин товарів існують дані про всі покупки користувачів за великий проміжок часу. Можна встановити залежність між типом користувачів та типом товарів які вони купують або переглядають, щоб прогнозувати рекомендації "на льоту".

1.2.5 Ланцюги Маркова (Markov chains)

Ланцюг Маркова − це один з найпростіших випадків послідовності випадкових подій. Але, незважаючи на простоту, він може бути корисною

навіть під час прогнозування результатів обчислень складних систем. Ланцюгом Маркова називають таку послідовність випадкових подій, у якій ймовірність кожної наступної події залежить від попередніх подій.

Цей підхід з теорії ймовірностей може бути використаний для вироблення рекомендацій в аналогічному до байєсових мереж довіри контексті. Але в цьому випадку проблема прогнозування рекомендації буде вирішена як послідовна оптимізація якості рекомендацій, а не як просте прогнозування.

1.2.6 Класифікація за методом Роккіо (Rocchio classification)

Цей метод класифікації базується на векторній моделі. Дуже часто його використовують в рекомендаційних системах в парі з алгоритмом TF-IDF: після того як обчислені значення tf-idf для всіх документів корпусу використовується аналіз релевантності документів (класифікація) для підвищення якості та точності рекомендацій.

1.3 Оцінювання в рекомендаційних системах для збору вхідних даних РС

Важливим етапом створення рекомендаційної системи є збір вхідних даних. Для сбору рейтингів використовують явні та неявні способи: вимірювання кількісті часу, яку користувач проводить на конкретній сторінці, чи оцінювання типу товарів що користувач обирає найчастіше. Найбільш розповсюджений та перевірений спосіб – явний спосіб, через систему оцінювання товару чи послуги.

Для оцінювання товарів використовують різні типи систем оцінювання. Найпоширеніші з них:

- 5-бальна;
- 10-бальна;
- бінарна (сподобалось/не сподобалось).

Вибір способу оцінок товару впливає на подальший розвиток проекту та прибуток від нього. Також він суттєво впливає на роботу алгоритму рекомендаційної системи. Отже, необхідно проаналізувати усі типи систем оцінювання та обрати найкращий. [9]

Рекомендаційні системи та алгоритми у свій час започаткувала та активно розвивала компанія Netflix. На своєму сайті вони надали інформацію, як саме користуватися такою системою:

★☆☆☆☆ Зовсім не сподобався;

★★☆☆☆ Не сподобався;

★★★☆☆ Сподобався;

★★★★☆ Дійсно сподобався;

★★★★★ В захваті.

Такий підхід перейняли багато інших відомих компаній, які надають послуги: Amazon, eBay та ін. Але він дуже суб’єктивний. Що може означати “дійсно сподобався” фільм? Чому інтервали між різними операціями нерівні (немає опції “дійсно не сподобався”)? Тож, можна зазначити що навіть текст, який несе на своїй меті допомогти виставити рейтинг товару, містить в собі суб’єктивність.

Незважаючи на росповсюдженість п’ятибальної шкали користувачі несвідомо зводять свій вибір до оцінювання бінарним підходом. В 2009 році Youtube (відео-платформа від Google) поширила статистику оцінок користувачів, серед яких найпопулярнішими оцінками стали “5” (~70%) та “1” (~25%).

Бінарне оцінювання використане на практиці показало, що воно є менш суб’єктивним, менш багатозначним, адже користувачам значно зручніше та швидше визначити: чи сподобався їм товар, чи ні.

Бінарний підхід також містить декілька недоліків. Він залишає деяку невизначеність при оцінюванні. Якщо користувачу не сподобався товар чи послуга, то існує велика вірогідність, що він зовсім не буде ставити оцінку. Така система - це система типу “все або нічого”. Але користувачі, які все таки ставлять оцінки дуже серйозно відносяться до своїх рекомендацій та дуже зацікавлені в тому, щоб система пропонувала їм релевантні результати.

Деякі проекти (в основному - соціальні мережі) долають проблему відмови від голосування (такі як VK, Facebook). Якщо опустити деякі деталі, то якийсь товар буде оцінено в +1, якщо ви проголосували позитивно, та -1, якщо ви не звернули увагу на нього. Це є різновид бінарного голосування, який теж гарно працює в сучасних умовах (в основному його використовують СМІ).

Отож, незважаючи на велику кількість способів оцінити товар чи послугу в проекті, краще це робити за допомогою бінарного методу, оскільки він не містить недоліків та суттєво знижує суб’єктивність оцінки. Також завдяки бінарному підходу відпадає проблема “холодного старту” (рекомендації новим користувачам) та значно покращується робота алгоритму (менше даних для обробки - більша швидкість). [8]

1.4 Висновок

В розділі було детально проаналізовано базові підходи до створення рекомендаційних систем. В результаті дослідження було зроблено висновок, що не існує універсального алгоритму, який би підійшов до будь-якого інтернет сервісу. Кожен тип алгоритмів слід використовувати лише за призначенням.

Алгоритми колаборативної фільтрації прогнозують рекомендації на основі схожості користувачів. В загальному випадку, алгоритми цієї групи розбивають користувачів на кластери, базуючись на їх оцінках, та прогнозують нові товари в рамках конкретної групи (кластеру). Реалізація таких алгоритмів порівняно проста, але на великому обсязі даних їх швидкодія різко знижується. Такі алгоритми добре підходять для блогів з помірними навантаженнями, соціальних мереж з невеликим обсягом інформації, що прогнозується тощо.

Підходи контентної фільтрації підходять до вирішення проблеми з іншого боку. Вони надають рекомендації на базі інтересів користувача та атрибутів продуктів. Реалізація таких алгоритмів є доволі складною, оскільки необхідно будувати складні математичні моделі, а їх важко створювати та підтримувати. Але швидкодія таких алгоритмів знаходиться на високому рівні. Цей блок алгоритмів широко застосовується в інтернет-магазинах.

Гібридні алгоритми показують найкращі результати, оскільки вони мають на меті вирішити основні проблеми колаборативної та контентної фільтрації. Вони базуються на змішуванні результатів в різному вигляді алгоритмів цих двох груп. Але проблема в тому, що такі системи створювати та підтримувати в декілька разів складніше, ніж системи з контентною фільтрацією, так як вони мають обраховувати велику кількість даних одночасно. Тому їх використання буде виправданим лише у випадку використання у складі крупного та високонавантаженого інтернет-сервісу з мільйонами користувачів та десятками мільйонів одиниць всілякого контенту. Зазвичай використовуються у великих інтернет-магазинах, соціальних мережах тощо.

Для покращення точності результатів колаборативної та контентної фільтрації необов'язково використовувати гібридні системи. Існують деякі алгоритми, які покращують результати цих алгоритмів. Наприклад, SVD алгоритм значно збільшує швидкодію алгоритмів колаборативної фільтрації,

оскільки знижує кількість обчислень, а алгоритм TF-IDF дозволяє створювати більш тонкі моделі для контентної фільтрації.

Також під час дослідження було виявлено, що майже будь-який алгоритм має отримувати на вхід оцінки користувачів, щоб вимірювати їх зацікавленість в даному товарі. Тому необхідно ретельно вибирати систему оцінювання. Найкращою виявилася бінарна система оцінювання.

2 АНАЛІЗ РЕКОМЕНДАЦІЙНИХ ТЕХНОЛОГІЙ ВІДОМИХ ІНТЕРНЕТ-СЕРВІСІВ

2.1 Програмна платформа інтернет-магазину eBay

Інтернет-магазин eBay має досить розвинену програмну платформу, яка об’єднує ряд програмних інтерфейсів (API) та сервісів, які направлені на ефективнішу взаємодію кінцевих користувачів з каталогом товарів eBay. Використання цих переваг дає можливість значно покращити якість роботи рекомендаційної системи, явно чи неявно вплинувши на зростання релевантності пропозицій товар чи послуг:

- Виконання пошукових запитів з урахуванням семантики запиту, що дозволяє забезпечувати кращу відповідність результатів пошуку очікуванням користувачів. Для обліку семантики запитів кожен рівень таксономічного графу (каталог товарів та послуг) асоціюється з набором ключових слів, які забезпечують максимально точне відображення можливих термінів в пошуковому запитів на відповідні розділи каталогу. Підхід, заснований на використанні ключових слів, є досить обмеженим на відміну від використання онтологій, але в контексті інтернет-магазину виглядає цілком виправданим.
- Обробка відгуків покупців з метою формування рейтингу продавця (Feedback API). Товари, що пропонуються з «високим» рейтингом матимуть більший пріорітет при формуванні пропозицій для потенційних покупців.
- Формування груп взаємопов'язаних товарів, які можуть пропонуватися «пакетом» (Related Items Management API). Таким чином, продавець має можливість явно вказувати які ще товари можуть бути корисні покупцю.
- Створення зв'язків, які описують взаємозв'язки комплектуючих та аксесуарів з продуктами (Product Services). Так покупець при пошуку

ноутбука може, наприклад, отримати одразу ж список сумок, що підходять до його моделі ноутбука.

2.1.1 Метрики “ефективності” просування товарів в каталозі eBay

Окремої уваги в контексті задачі збільшення релевантності пропозицій товарів та послуг заслуговує сервіс eBay Listing Analytics, який дозволяє виконати аналіз “ефективності” пропозицій конкретних продавців. Для оцінки “ефективності” (в даному контексті, успішності поточної групи товарів або послуг для цільової аудиторії) використовуються наступні ключові метрики:

- ***Rank***. Дана метрика визначає положення конкретних товарів та послуг в загальному рейтингу покупців. Так, наприклад, товар з рейтингом “5” буде пропонуватися покупцю раніше, ніж товар з рейтингом “15”. При роботі з сервісом eBay Listing Analytics пошук товарів виконується із використанням заданих ключових слів з метою оцінки їх рейтингу. Таким чином, конкретне місце товару чи послуги в списку пропозицій залежить від рангу та ключових слів, які використовуються при формуванні пошукового запиту.
- ***Format***. Ця метрика описує формат, якому відповідають товар чи послуга, що рекомендуються користувачу (auction-style listing, fixed price listing).
- ***Impressions***. Метрика “Враження” характеризує кількість разів, що поточна група товарів та послуг рекомендувалася потенційним покупцям в процесі пошуку.
- ***Clicks***. Ця метрика оцінює кількість запитів користувачів до детальних характеристик товарів чи послуг після перегляду списку рекомендацій.
- ***Click through***. Дана метрика – це відношення метрики Clicks до метрики Impressions (Clicks count / Impressions count). Чим більше значення цієї метрики, тим краще, бо це означає, що покупці частіше обирають (тобто заходять на сторінку детального опису) рекомендовані товари та послуги, на відміну від аналогічних.

- ***Sold items***. Ця метрика представляє собою кількість разів, коли покупці придбали рекомендований товар чи послугу.
- ***Sell through***. Ця метрика представляє собою відношення кількості проданих товаров чи послуг до кількості їх переглядів при пошуку. Чим більше значення цієї метрики, тим краще, адже це означає, що покупці частіше після перегляду описів цих пропозицій приймають рішення про купівлю.
- ***Watchers***. Загальна кількість переглядів конкретних товарів чи послуг.
- ***Sales***. Загальна кількість проданих товарів чи послуг в грошовому еквіваленті (тобто загальна сума покупок кожного товару зі списку).

Використання аналітичних засобів, таких як eBay Listing Analytics, дозволяє проаналізувати успішність різних груп товарів та послуг у покупців, визначити найефективнішу стратегію маркетингу товарів та скорегувати асортимент, а також збільшити якість обслуговування покупців за рахунок пропозицій тих товарів та послуг, які користувач очікує побачити. Але варто нагадати, що метрики зазначені вище якраз і забезпечують зріз з величезних об’ємів даних, що збирає каталог. [10]

2.2 Рекомендаційна система інтернет-магазину Amazon

Інший досить крупний інтернет-магазин Amazon (amazon.com) розробив власне високопродуктивне сховище пар “ключ-значення” (Highly Available Key-Value Store) Dynamo, яке використовується рекомендаційною системою Amazon. Dynamo використовує синтез добре відомих технік для досягнення масштабування та високої доступності: дані кластеризуються (partioning) та реплікуються, використовуючи узгоджене хешування (consistent hashing), а корректність даних забезпечується за допомогою версій об’єктів. Для 99% запитів СУБД забезпечує час відгуку на запит не більше, ніж 300 мс. Для витягання інформації зі сховища достатньо знати значення ключа. В період

пікових навантажень система забезпечує обробку декількох мільйонів запитів на день.

Рекомендаційна система інтернет-магазину Amazon реалізує різні підходи до формування рекомендацій, основною метою яких є максимальний облік інтересів користувачів за допомогою їх залучення до процесу “оцінювання товарів”, а також неявного аналізу їх поведінки на сайті:

- ***Customers who Bought***. Даний підхід до формуванню списку рекомендацій використовує інформацію про популярність товарів у покупців зі схожими інтересами. Так при виборі конкретної книги, користувачу буде запропонований список книг, які користуються попитом у людей, що вже придбали цю книгу, а також список авторів книг, чиї роботи купують покупці книг, автором яких є саме автор обраної книги. Для реалізації даного механізму сервіс використовує спеціальний алгоритм “Item to item Correlation”, що був запатентований компанією Amazon. Основна ідея алгоритму – це зіставлення товарів, що придбав користувач, з аналогічними товарами (з використанням ключових параметрів, характеристик) та формування рекомендацій з урахуванням рейтингу цих товарів.
- ***Eyes***. Цей сервіс дозволяє користувачам отримувати листи на електрону пошту про додавання нових товарів до каталогу Amazon. Користувачі можуть повністю контролювати параметри товарів, на базі яких буде готуватися нова вибірка для сповіщення. Запити можна формувати неявно, за допомогою використання вже існуючої вибірки в якості шаблона для пошуку.
- ***Amazon.com Delivers***. Даний серів дуже близький за функціоналом до сервісу Eyes. Користувач має можливість задати категорії каталогу (наприклад, кулінарні книги або фантастика) та оформити підписку на отримання сповіщень про рекомендовані товари з обраних категорій.

- ***Book Matcher***. Цей сервіс дає можливість користувачам залишати відгуки безпосередньо про куплені чи прочитані книги. Для прочитаних книг покупець формує рейтинг за п'ятибальною шкалою (від "hated it" до "loved it"). На базі інтересів користувачів сервіс формує рекомендації для нього. При цьому рекомендовані книги в свою чергу можуть бути оцінені (за допомогою надання рейтингу) покупцем (функція "rate these books"), що дозволить наступного разу більш точно передбачити його побажань при формуванні рекомендацій.
- ***Customer Comments***. Даний сервіс дає можливість отримувати рекомендації, базуючись на думках інших користувачів. Так, наприклад, для кожної книги в каталозі закріплено читацький рейтинг у вигляді списку від однієї до п'яти зірочок, який супроводжується текстовими коментарями покупців. Це дає можливість більш точно отримувати досвід про книгу перед її купівлею.

Коли користувач обирає для купівлі який-небудь товар, Amazon на базі цього вихідного товару рекомендує користувачу інші товари, які було куплено іншими користувачами (за допомогою матриці купівлі наступного товара на базі його схожості з попередньою покупкою). [11]

2.2.1 Пошуковий сервіс компанії A9

Паралельно з Dynamo, Amazon використовує сервіс пошуку товарів A9, який також заслуговує окремої уваги. Він запустився в 2003 році та мав на меті допомагати людям шукати речі, які вони справді жадають знайти. Він складається з декількох складових, які, працюючи разом, допомагають користувачам знайти товари.

2.2.1.1 Пошук продуктів

Робота пошукового сервісу розпочинається ще до того, як користувач введе запит в пошукову строку. Пошуковий алгоритм аналізує попередню

поведінку користувача, переглядає якомога більше продуктів, які користувачу були хоча б якимось чином цікаві у минулому, запитів користувача, щоб краще зрозуміти його стиль “спілкування” з пошуковими двигунами, та майже одразу ж готовий запропонувати товари за допомогою ***Instant Search***.

Користувачі очікують знайти коректні продукти навіть у тому випадку, якщо пошуковий запит не зовсім точний. Наприклад, коли користувач вводить запит “Harry Potter in books”, система починає шукати “Harry Potter” в каталозі книг.

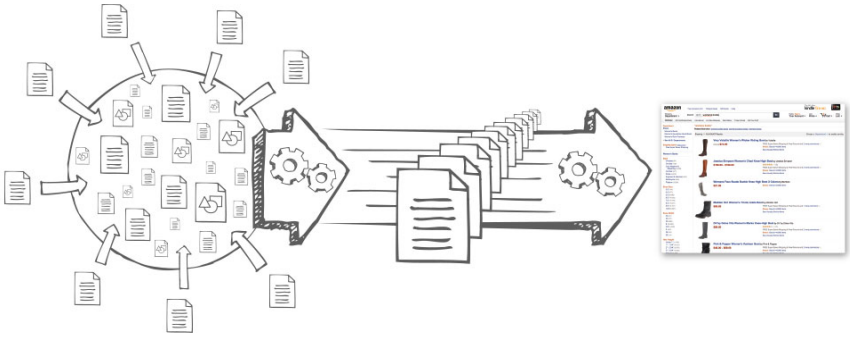

Рисунок 2.1 – A9 Product Search [12]

Одним з основних приципів команди A9 – це релевантність результатів пошуку. Алгоритми ранжування завжди навчаються на вибірках даних з поведінки реальних користувачів та намагаються також звертати увагу на широкий минулий досвід роботи з клієнтами.

2.2.1.2 Visual Search

Рисунок 2.2 – Приклад пошуку товарів за допомогою картинки [12]

За допомогою будь-якого смартфона з камерою користувачі можуть використовувати фото товару як пошуковий запит. Алгоритм, що використовує

Computer Vision для надання користувачам рекомендацій щодо товарів, називається Augmented Reality.

2.2.1.3 Advertising Technology

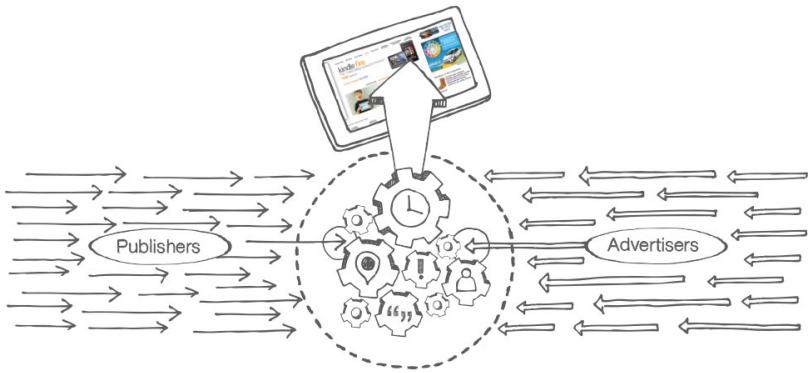

Рисунок 2.3 – Ілюстрація роботи рекламних сервісів A9 [12]

Було побудовано спеціальну рекламну систему, що на основі інтересів користувача рекомендує йому товари, що йому можуть сподобатися. Звернувши увагу на величину аудиторії Amazon, можна помітити, що ця система має величезну кількість запитів кожного дня (близько міліарду). Ця технологія включає в себе три сервіси, які направлені на надання найбільш релевантного контенту, використовуючі велику кількість рекомендаційних алгоритмів:

- ***Publisher Display Products***. Ця система надає відкритий API до каталогу продуктів Amazon. Тобто всілякі блогери та розробники мобільних додатків можуть використовувати цей сервіс для монетизації їх додатків. Про кожного користувача буде зібрана деяка інформація, на базі якої буде відображатися реклама з товарами, які є найбільш цікавими користувачу.
- ***Ad Tech Platform***. Створює рекламну базу для товарів та також надає відкритий API для розробників. Робота цієї системи розповсюджується не тільки на Amazon.
- ***Sponsored Links***. Просуває проспонсовану рекламу на інших ресурсах.

Також система A9 має інші сервіси, які націлені на швидкість пошукових алгоритмів з великої бази продуктів. [12]

2.3 Порівняльна характеристика сервісів Amazon та eBay

Нижче наведено перелік критично важливих рекомендаційних сервісів двох інтернет-гігантів та їх короткі характеристики.

Таблиця 2.1 – Зведена таблиця рекомендаційних сервісів

Інтернет-сервіс	**Вхідні дані**	**Рекомендаційна технологія**	**Вихідні рекомендації**
Amazon			
Customers who Bought	Схожий товар (база покупок)	Контентна фільтрація	Блок рекомендованих товарів
Eyes	Електронна пошта	Контентна фільтрація	Лист з рекомендаціями
Amazon.com Delivers	Електронна пошта	Контентна фільтраця	Лист з рекомендаціями, де наведені форми для корегування інтересів
Book Matcher	Найкращі товари	Колаборативна фільтрація	Список товарів
Customer Comments	Середній рейтинг товару. Коментарі користувачів	Гібридна фільтрація на базі рейтингів та ключових слів в коментарях	Кореляція рекомендацій інших сервісів Amazon
Product Search	Пошуковий запит (текст / картинка)	Гібридна фільтрація	Список товарів

Продовження таблиці 2.1

Інтернет-сервіс	**Вхідні дані**	**Рекомендаційна технологія**	**Вихідні рекомендації**
eBay			
Feedback API	Середній рейтинг товару. Коментарі користувачів	Гібридна фільтрація на базі рейтингів та ключових слів в коментарях	Кореляція рекомендацій інших сервісів eBay
Related Items Management API	Зібрані дані про інтереси споживача	Контентна фільтрація	Блок рекомендацій з комплектами товарів
Product Services	Дані інших користувачів про інтерес до аксесуарів для базового товару	Колаборативна фільтрація	Блок рекомендацій з аксесуарами
Product Search	Пошуковий запит	Гібридна фільтрація	Список рекомендацій
eBay Listing Analytics	Дані про поведінку всіх користувачів	Гібридна фільтрація	Кореляція рекомендацій інших сервісів

2.4 Висновок

В даному розділі було проаналізовано вбудовані модулі рекомендаційних систем діючих інтернет-ресурсів, що продають товари користувачам, Amazon та eBay. Звичайно, що комерційні сервіси намагаються тримати в таємниці способи збільшення прибутків (рекомендаційні системи – один із цих шляхів), тому характеристики для кожного з модулів були описані на базі інформації, що міститься у відкритих блогах цих компаній.

Як можна помітити, Amazon та eBay використовують схожі технології для надання користувачам рекомендацій, але якщо звернути увагу на статистику відгуків про якість рекомендаційної систем цих інтернет-сервісів, Amazon має більш розвинену інфраструктуру та якість систем, що корелюють дані користувачів для надання рекомендацій.

3 РОЗРОБКА ВЕБ-ДОДАТКУ ІЗ ВБУДОВАНОЮ СИСТЕМОЮ РЕКОМЕНДАЦІЙ

3.1 Особливості фреймворку Ruby on Rails

Ruby – динамічна мова програмування, яка має на меті збільшити простоту та продуктивність вихідного коду. Він має зручний синтаксис, який приємно читати та легко писати.

Ruby on Rails – фреймворк, написаний на мові програмування Ruby, тобто програмне забезпечення, що полегшує розробку та об’єднання декількох окремих складових проекту (наприклад, аутентифікація та авторизація користувачів або каталог статей в блозі).

Фреймворк, на відміну від CMS (система керування вмістом), яку може розгорнути та налаштувати навіть не-програміст, потребує проектування та розробки кваліфікованими спеціалістами. Але на ньому зручніше та швидше створювати проекти, які зовсім відрізняються функціоналом від типового сайту. А до веб-студій та агентств нечасто приходять за повністю типовими сайтами, оскільки замовники часто змінюють поведінку “на льоту”.

3.1.1 Переваги платформи Ruby on Rails

Основною перевагою мови програмування Ruby та фреймворку Ruby on Rails є швидкість розробки. На практиці швідкість розробки проектів на RoR вище на 30-40% по відношенню до інших мов програмування або фреймворків. Такий приріст швидкості розробки пояснюється широким набором готових до роботи «з коробки» інструментів RoR, можливістю використовувати готові бібліотеки інших розробників та, звичайно, зручністю програмування на Ruby.

Крім цього, на відміну від інших фреймворків, до складу RoR входять ефективні засоби для автоматизованого тестування, що прискорює перехід проекту від стадії “програму написано” до стадії “програма працює без

помилок”. Цей перехід майже завжди займає найбільшу кількість часу під час реалізації майже будь-яких проектів.

Також варто відмітити, що Ruby on Rails забезпечує кращу безпеку для додатків. Під час використання інструментів RoR виключені SQL-ін’єкції та XSS-атаки, всі вхідні параметри екрануються за замовчанням, вихідні змінні в шаблонах також екрануються. У розробника майже немає шансів допустити помилку безпеки.

Деякі розробники, що недостатньо добре знайомі з цією технологією, чомусь вважають, що інтернет-проекти на RoR погано масштабуються. Як приклад майже всі розробники наводять Twitter, який в свій час відмовився від Rails через якісь внутрішні причини. Але треба звернути увагу на більш відомі проекти, як Kickstarter, Groupon або Basecamp – всі ці проекти написані з використанням Rails без проблем з масштабуванням. В будь-якому випадку, проблеми продуктивності будь-якого проекту, - це не проблеми помилкового вибору платформи чи мови програмування. Найчастіше ці проблеми були викликані помилками під час проектування архітектури проекту, кешуванням даних або неоптимальним вибором СУБД.

3.1.2 Обмеження фреймворку

Розробників на Ruby on Rails менше, ніж розробників на PHP та його фреймворках, оскільки тут вищий поріг входження та, зазвичай, програміст приходить до Ruby вже після декількох років PHP. Але при цьому слід пам’ятати, що досвідчених розробників дуже мало в цих двох сферах.

Важливим обмеженням RoR вважається вбудована ORM ActiveRecord та модуль Active Support, які містять велику кількість допоміжних методів, які майже не використовуються в проектах, але ці модулі можна замінити чимось менш ресурсозатратним (Sequel).

3.1.3 Типові проекти на Ruby on Rails

Фреймворк RoR найкраще підходить для наступних типів проектів:

- Інтернет-магазини з складною системою фільтрації вмісту, модулями підбору та інтеграціями із зовнішніми сервісами.
- Купонні сервіси, веб-сервіси для колективних закупівель.
- Інформаційні портали, електронні видання.
- Біржи та торгівельні майданчики.
- Сайти повідомлень та знайомств.
- Соціальні мережі.
- Незвичайні/нестандартні, технічно важкі проекти.
- Сервіси та SaaS-проекти. [16]

3.2 Побудова memory-based алгоритму колаборативної фільтрації з бінорною системою оцінювання

Як було описано вище, існує величезна кількість алгоритмів для різних типів рекомендацій. Найшвидшими в реалізації та імплементації виявилися рекомендаційні алгоритми, що використовують колаборативну фільтрацію. В цьому розділі буде приділено увагу memory-based рішенню для колаборативної фільтрації.

Memory-based підхід досить розповсюджений під час реалізації рекомендаційної системи, що використовує колаборативну фільтрацію, на будь-якій серверній мові програмування, оскільки в якості “пам’яті” рекомендаційної системи він використовую ту саму ж базу даних, що і основний додаток.

Декілька з цих алгоритмів використовують широковідомі підході як Евклідова відстань, кореляція Пірсона, коефіцієнт схожості векторів Оцукі-Отіаі та алгоритм k-найближчих сусідів. Вони мають повну документацію,

багато прикладів реалізації та базуються на 5-бальній шкалі оцінювання товарів. Але, як було показано вище, зрозуміліше та краще спрацьовує підхід, що використовує лише бінарну систему рекомендацій (“сподобалося – не сподобалося”).

3.2.1 Коефіцієнт Жаккара

Міра Жаккара (коефіцієнт флористичної спільності, фр. *coefficient de communaute*) – бінарна міра подібності, що була запропонована Полем Жаккаром в 1901 році. Індекс Жаккара (як його ще називають) це коефіцієнт, що знаходиться шляхом обчислення подібності двох наборів даних. В інформатиці, коефіцієнт Жаккара для двох множин ***A*** та ***B*** дорівнює відношенню кількості елементів перетину множин до кількості елементів їхнього об’єднання:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

Наприклад, при порівняні двох користувачів u_1 та u_2. Щоб обчислити їх переріз, необхідно порахувати лише кількість тих товарів (послуг), що сподобалася двом користувачам (див. формулу 3.2). Об’єднання буде обраховуватися схожим чином – складаються сумарні кількості товарів, що сподобалися обом користувачам незалежно один від одного. Але такий спосіб не вирішує проблеми обрахунку подібності, коли користувачі використовують рейтинги (5-бальний, бінарний).

$$agreements(u_1, u_2) = \sum_{l \in L} \{\, l \mid l \in L_1 \cap L_2 \}, \tag{3.2}$$

де L – загальна множина множина позитивних оцінок користувачів u_1 та u_2;

L_1 – множина позитивних оцінок користувача u_1;

L_2 – множина позитивних оцінок користувача u_2;

l - позитивна оцінка користувача u_1 або u_2.

Можна видозмінити формулу для знаходження подібності користувачів за умови використання бінарної системи оцінювання товарів. Для цього, кількість спільних товарів, що сподобалися двом користувачам в перерізі, додається до кількості спільних товарів, що ці користувачі оцінили як “не сподобалось” (формула 3.3). Знайдений вираз ділиться на загальну кількість товарів (формула 3.4), що користувачі оцінювали будь-яким чином (“сподобалось” та “не сподобалось”).

$$disagreements(u_1, u_2) = \sum_{d \in D} \{ d \mid d \in D_1 \cap D_2 \}, \tag{3.3}$$

де $\boldsymbol{D}$ – загальна множина множина позитивних оцінок користувачів $\boldsymbol{u_1}$ та $\boldsymbol{u_2}$;

$\boldsymbol{D_1}$ – множина негативних оцінок користувача $\boldsymbol{u_1}$;

$\boldsymbol{D_2}$ – множина негативних оцінок користувача $\boldsymbol{u_2}$;

$\boldsymbol{d}$ - негативна оцінка користувача $\boldsymbol{u_1}$ або $\boldsymbol{u_2}$.

$$total(u_1, u_2) = \sum_{m \in L \cup D} \{ m \mid m \in L \cup D \}, \tag{3.4}$$

де $\boldsymbol{m}$ - позитивна оцінка користувача $\boldsymbol{u_1}$ або $\boldsymbol{u_2}$.

Але що робити у випадку, коли один товар сподобався користувачу $\boldsymbol{u_1}$, але не сподобався користувачу $\boldsymbol{u_2}$. Це ж також має якось впливати на подібність цих користувачів. Тому необхідно як додаток до міри узгодженості рейтингів цих користувачів обрахувати міру їх неузгодженості. Це можна зробити наступним чином: узгодженості між користувачами будуть обраховуватися як і раніше (складаються кількість товарів, що сподобалися обом користувачам), а щоб обрахувати неузгодженості береться кількість товарів які сподобалися одному користувачу, але не сподобалися іншому. Потім отримані результати віднімаються один від одного і результат ділиться на сумарну кількість товарів, що користувачі оцінювали впродовж користування продуктом (формула 3.5).

$$sim(u_1, u_2) = \frac{agreements(u_1, u_2) - disagreements(u_1, u_2)}{total(u_1, u_2)} \tag{3.5}$$

Отриманий коефіцієнт подібності буде лежати в межах [-1, 1]. Результат “-1” буде отримано у випадку, коли користувачу u_1 сподобалися товари, що не сподобалися u_2, або навпаки, а результат “1”, коли смаки двох користувачів повністю співпадуть (вони оцінювали однакові товари як “сподобалось”).

3.2.2 Прогнозування рекомендацій

Коефіцієнт подібності між користувачами використовується для прогнозування списку рекомендацій для користувачів. Для цього необхідно обрахувати для кожного товару *hive-mind* сумму (коефіцієнт колективної думки).

Для користувачів із заданої групи, які якимось чином оцінили товар (він їм сподобався або ні), необхідно виконати наступні обчислення: якщо користувачу сподобався товар, то до загальної суми додається коефіцієнт подібності поточного користувача з тим, що оцінював даний товар, в протилежному випадку – від суми віднімається коефіцієнт подібності між користувачами. Після цього отриманий результат ділиться на загальну кількість людей, що якимось чином оцінили даний товар.

Для кращого розуміння нижче наведений приклад коду, що обраховує прогнозовану оцінку товару для користувача:

```
class User
 def similarity_with(user)
 # Array#& is the set intersection operator.
 agreements = (self.likes & user.likes).size
 agreements += (self.dislikes & user.dislikes).size

 disagreements = (self.likes & user.dislikes).size
 disagreements += (self.dislikes & user.likes).size

 # Array#| is the set union operator
 total = (self.likes + self.dislikes) | (user.likes + user.dislikes)
```

```
    return (agreements - disagreements) / total.size.to_f
    end

    def prediction_for(item)
    hive_mind_sum = 0.0
    rated_by = item.liked_by.size + item.disliked_by.size

    item.liked_by.each { |u| hive_mind_sum += self.similarity_with(u) }
    item.disliked_by.each { |u| hive_mind_sum -= self.similarity_with(u) }

    return hive_mind_sum / rated_by.to_f
    end
  end
```

Таким чином такий спосіб обрахунку прогнозів дозволить користувачам, що ще невстигли оцінити багато товарів, отримати прогнози від користувачів, що оцінили велику кількість товарів. [13]

3.3 Реалізація блогу з системою рекомендацій контенту

3.3.1 Формулювання вимог до веб-додатку

В результаті аналізу існуючих рішень рекомендаційних систем, прикладів реалізації різних інтернет-сервісів було обрано платформу Ruby on Rails. Тому, зважаючи на вибір фреймворку, сформулюємо ряд характерних вимог до реалізації веб-додатку:

1. Створення базового функціоналу типового інтернет-блогу
    a. Авторизація та аутентифікація для користувачів
    b. Базові методи роботи зі статтями: створення, редагування, видалення
    c. Можливість динамічного створення секцій для впорядкування статей
    d. Додавання тегів для статей

2. Імлементація рекомендаційної системи на базі методів API бібліотеки Recommendable:
    a. Інтеграція з поточними моделями користувачів та статей
    b. Створення методів для оцінок статей користувачами
    c. Налаштування фонових процесів для прогнозування рекомендацій
3. Аналіз результатів тестування

3.3.2 Особливості середовища розробки

В даному підрозділі буде наведено покроковий процес установки та налаштування всіх необхідних додатків. Необхідно мати встановлену Unix-подібну операційну систему

3.3.2.1 Встановлення RVM та Rails

Ruby Version Manager (RVM) – це консольний додаток для керуванням версіями Ruby. Він дає можливість мати скільки завгодно версій Ruby, встановлених в одній системі, та керувати наборами бібліотек (gemsets lists).

Щоб встановити rvm в систему, необхідно ввести в командну строку (термінал) команду:

```
\curl -sSL https://get.rvm.io | bash -s stable
```

Після цього rvm стане доступним за допомогою команди `rvm`. Далі необхідно встановити ruby останньої версії за допомогою rvm:

```
rvm install 2.3.0
```

Необхідно завантажити бібліотеки bundler (менеджер бібліотек) та власне rails:

```
sudo gem install bundler
```

```
sudo gem install rails --no-ri --no-rdoc
```

3.3.2.2 Створення та налаштування проекту

Майже всі додатки Rails починаються однаково – з команди **`rails new`**. Ця команда створює скелет додатку в будь-якому каталозі (папці). Для початку треба обрати каталог для проектів, після цього запустити команду:

```
$ rails new blog_application
   create
   create README.rdoc
   create Rakefile
   create config.ru
   create .gitignore
   create Gemfile
   create app
   create app/assets/javascripts/application.js
   create app/assets/stylesheets/application.css
   create app/controllers/application_controller.
   …
   create test/test_helper.rb
   create tmp/cache
   create tmp/cache/assets
   create vendor/assets/javascripts
   create vendor/assets/javascripts/.keep
   create vendor/assets/stylesheets
   create vendor/assets/stylesheets/.keep
   run bundle install
  .
  Your bundle is complete! Use `bundle show [gemname]` to see where a
bundled
  gem is installed.
```

Як можна побачити з цього лістингу ця команда створює структуру проекту, генерує деякі основні файли та виконує команду **`bundle install`**, що встановлює всі залежності, що перераховані в ***Gemfile***. Після всіх команд буде побудовано додаток із наступною структурою папок:

Таблиця 3.1 – Опис структури додатку Rails

Файл/директорія	***Призначення***
app/	Основний код додатку (app), включає в себе контролери, моделі, представлення, допоміжні файли (хелпери) та асети (assets – файли JS, CSS, картинки, шрифти і т.ін.).
bin/	Бінарні файли основних частин – rails, rake …
config/	Файли конфігурації додатку – маршрути,
config.ru	Rack-конфігурації для Rack-орієнтованих серверів для старту додатку.
db/	Містить файли баз даних – схему та міграції.
Gemfile **Gemfile.lock**	Ці файли дозволяють керувати версіями бібліотек, що використовуються в додатку.
lib/	Розширені модулі вашого додатку та rake-задачі для виконання різного роду допоміжних завдань.
log/	Файли логів додатку.
public/	Точка входу в додаток в мережі інтернет. Містить всі статичні файли та скомільовані асети.
Rakefile	Містить перелік всіх rake-задач та залежностей, необхідних для їх виконання.
README.rdoc	Містить короткий опис додатку. Ви маєте додати пункти про те, що робить ваш додаток, як його налаштувати та сконфігурувати на іншій машині, і т.ін.
test/	Містить файли тестів та fixtures.
tmp/	Тут знаходяться тимчасові файли (кеш, pid, файли сесій тощо).
vendor/	Місце для коду сторонніх розробників. Тут можуть знаходитися коди Ruby Gems та вихідний код Rails (якщо вам необхідно змінити поведінку за замовчанням).

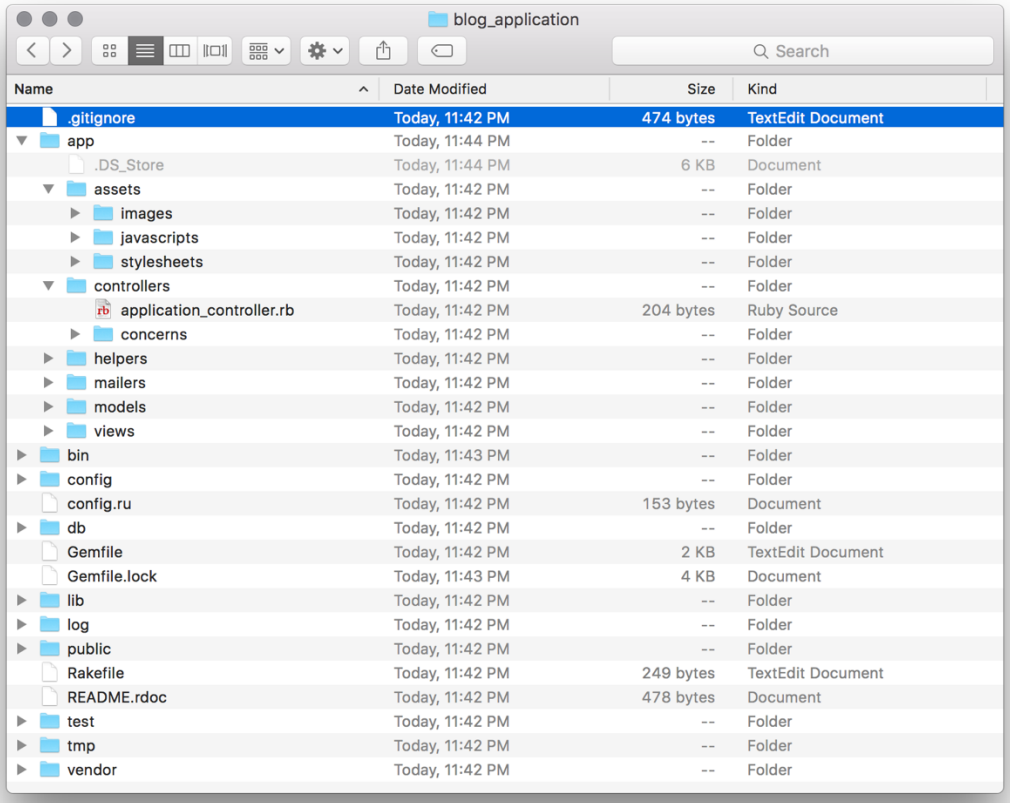

Рисунок 3.1 – Базова структура Rails-додатку

Після того, як проект було створено та згенеровано базову структуру необхідно додати усі бібліотеки для коректної роботи додатку. [14] Для цього необхідно додати усі залежності до Gemfile:

```
source 'https://rubygems.org'


gem 'rails', '4.2.6'
gem 'pg', '~> 0.18.4'
gem 'sidekiq'
gem 'redis'

# Layout
gem 'slim-rails'
gem 'sass-rails'
gem 'bootstrap-sass'
gem 'autoprefixer-rails'
```

```
gem 'therubyracer'

# Scripts
gem 'jquery-rails'
gem 'turbolinks'
gem 'uglifier'
gem 'coffee-rails'

# Utilities
gem 'puma', '~>3.1.0'
gem 'web-console', group: :development
gem 'simple_form'
gem 'devise'
gem 'nokogiri'
gem 'select2-rails'
gem 'acts-as-taggable-on', '~> 3.4'
gem 'ransack'
gem 'dotenv'

# Recommender system
gem 'recommendable'

group :development, :test do
 gem 'byebug'
 gem 'rubocop'
end

group :test do
 gem 'rspec'
end
```

Якщо не вказати для кожного гему корректної версії, то bundler автоматично підтягне найсвіжішу версію бібліотеки. На жаль, оновлення бібліотек інколи викликають незначні помилки. На вирішення цих проблем

інколи необхідно витратити велику кількість часу, оскільки stack-trace цих помилок інколи незрозумілий для програміста.

Після кожного редагування Gemfile необхідно завжди виконувати команду **`bunlde install`**, щоб встановити всі залежності для додатку.

Далі необхідно коректно налаштувати підключення до бази даних, адже під час цього пункту часто виникає велика кількість помилок. В якості СУБД було обрано PostgreSQL версії 9.5, адже серед open-source СУБД PostgreSQL є лідером через велику швидкодію, зручний інтерфейс доступу та купу нових функцій та модулів, які інколи значно спрощують роботу.

Для підключення бази даних необхідно мати бібліотеку **PG** та адаптер **postgresql** встановленими в системі. Далі налаштовується файл **`config/database.yml`** наступним чином:

```
default: &default
 adapter: postgresql
 encoding: unicode
 port: 5432
 pool: 5

development:
 <<: *default
 username: alekseymazurik
 database: mazurik_blog_development

test:
 <<: *default
 username: alekseymazurik
 database: mazurik_blog_test

production:
 <<: *default
 username: postgres
```

```
password: password
database: mazurik_blog_production
```

Зрозуміло, що цей лістинг має містити власні username та password для доступу до системного postgres (окрім production – більш детально про розгортання у хмарі в наступних підрозділах). Після підготовки всіх файлів необхідно створити базу даних у системі для коректної роботи додатку. Для цього необхідно запустити команду `$ rake db:create:all`, що створить усі необхідні бази даних для середовищ розробки та тестування. [15]

На цьому етапі закінчена початкова конфігурація додатку і можливе його запущення командою `bundle exec rails server`. Після виконання цієї команди на локальному сервері буде розгорнуто додаток, який стане доступним за адресою **localhost:3000** (стандартний для rails порт). За посиланням представлена веб-сторінка, що зображена на рисунку 3.2 [14].

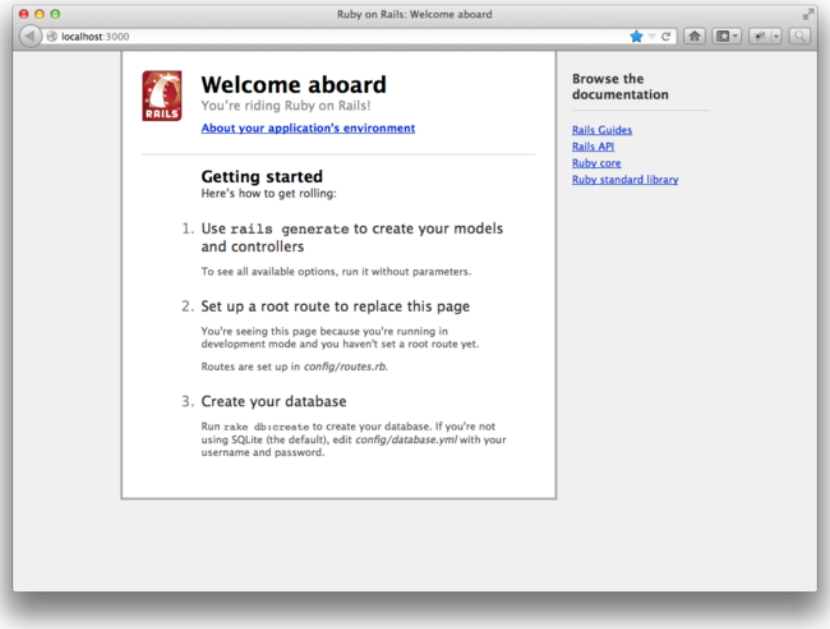

Рисунок 3.2 - Базова структура початкової сторінки Rails-додатку [14]

3.3.3 Особливості реалізації додатку на базі Rails

Додаток з базовою конфігурацією готовий, тепер необхідно спроектувати та створити додаток: веб-блог з вбудованою рекомендаційною системою.

3.3.3.1 Технології використані під час розробки

В якості основних були обрані наступні технології для роботи:

- **sidekiq** – бібліотека для простої та ефективної фонової обробки даних для Ruby (використовується для фонової роботи рекомендаційної системи);
- **recommendable** – бібліотека, що реалізує базовий функціонал рекомендаційної системи на базі колаборативної фільтрації;
- **redis** – адаптер для зв'язку з зовнішнім сховищем данних Redis;
- **devise** – бібліотека для аутентифікації користувачів в системі;
- **nokogiri** – бібліотека для парсингу зовнішніх сторінок;
- **puma** – application server для нашого додатку (за функціоналом та швидкістю значно швидше стандартного WEBrick);
- **slim-rails** – шаблонізатор для більш зручного написання views (представлень);
- **sass-rails** – препроцессор Sass для кращого написання CSS для нашого додатку.

Звичайно, Gemfile містить інші технології, що стабілізують та спрощують написання коду, але в цьому розділі було вирішено сконцентруватися лише на вище згаданих.

3.3.3.2 Архітектура додатку

Після вибору технологій необхідно спроектувати структуру додатку, щоб зробити написання коду більш доступним та зрозумілим. Структура додатку представлена на рис. 3.3.

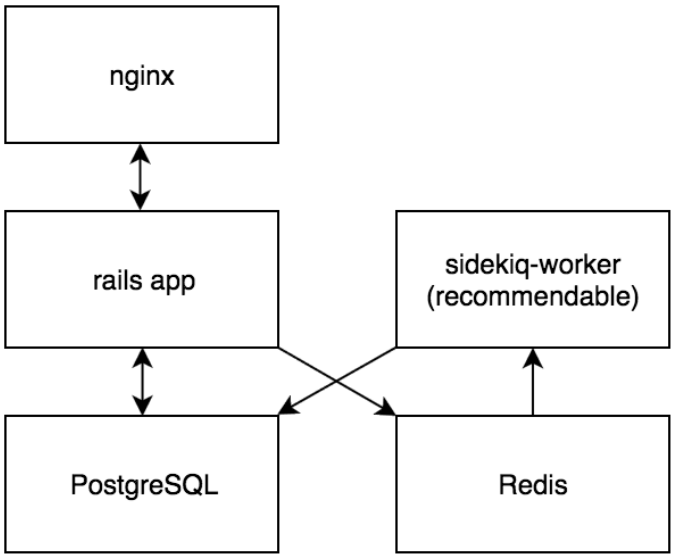

Рисунок 3.3 - Загальна архітектура веб-додатку

Задачею є створення додатоку, який зручно буде розмістити у хмарі із максимально простою архітектурою. В якості HTTP-серверу обрано nginx через його популярність та простоту налаштування. Сервер буде спілкуватися із rails додатком через HTTP-запити. В якості СУБД було обрано PostgreSQL.

Для коректної роботи та збільшення швидкодії додатку було вирішено виконувати обрахунки, що стосуються рейтингів в фоні, адже це зменшить навантаження на загальний проект. Для цього використовується технологія sidekiq, що дозволяє розгортати в фоні урізану версію додатку та виконувати обрахунки, використовуючи потоки, а не процеси (на відміну від rescue з його малою швидкодією).

Для того, щоб основний додаток міг спілкуватися з фоновим, використовується сховище пар ключ-значення. Найчастіше для таких дій використовують Redis через його простоту та високу швидкодію. Основний додаток буде додавати в чергу Redis запити на обрахунок потрібних рейтингів, а sidekiq після обробки поточних запитів буде витягувати та обробляти нові запити звідти.

3.3.3.3 Структура проекту

Моделі

Концепція моделі в Rails майже не відрізняється від базового означення цього компоненту в MVC. Модель – це складова патерну проектування MVC, що відповідає роботу з базою даних та надає зручний інтерфейс доступу до них. В базовому додатку rails кожна модель наслідується від **ActiveRecord::Base**. Після цього при роботі з моделями можна використовувати допоміжні методи, що значно спрощують подальшу розробку проекту (додаються методи для валідації, роботи з помилками, витягання з бази необхідної інформації тощо). Детальний код створення моделей представлений в Додатку А.

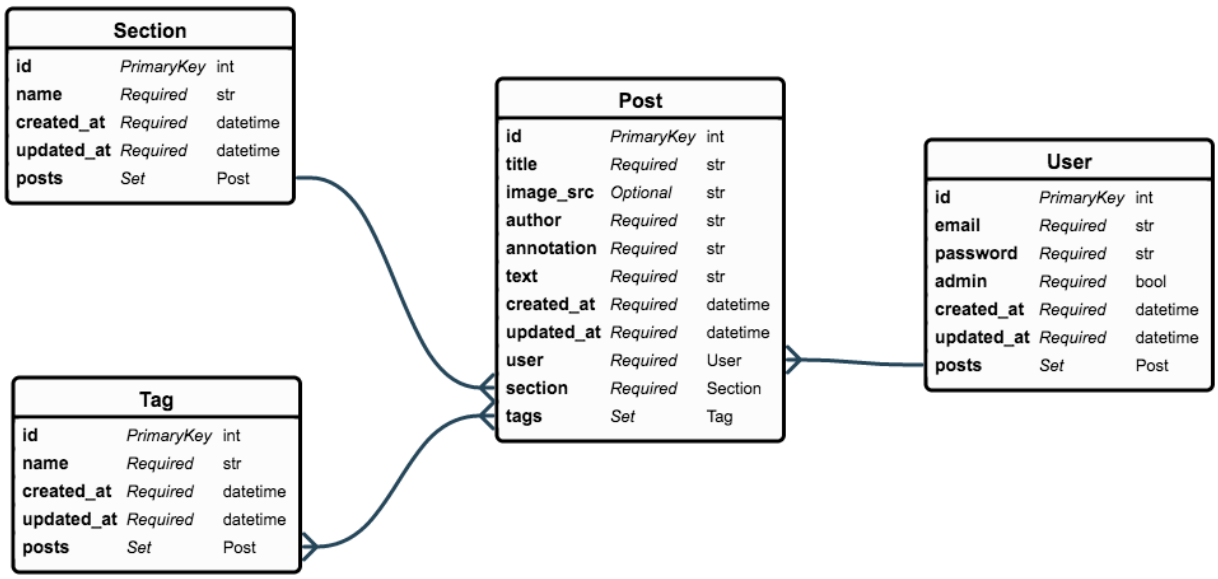

Рисунок 3.4 - Структура взаємозв’язків моделей блогу

Контролери

Контролери в Rails також беруть свій початок від базового означення. Контролер – це компонент патерну MVC, що забезпечує зв’язок між користувачами та системою: контролює введення даних користувачами та використовує моделі та представлення для реалізації необхідної реакції. Всі контролери наслідуються від базового контролера **ApplicationController**, що може містити методи, які мають бути доступними для всіх контролерів додатку.

Між контролерами та маршрутами існує прямий зв’язок. При розробці маршрутів в додатку на кожний маршрут має бути представлений метод контролера, який виконує ту чи іншу дію. Деякі методи можуть бути пустими: в цьому випадку як дія контролеру буде виконане лише надсилання представлення для цього маршруту. Увесь код контролерів представлений в Додатку А.

Представлення

Представлення в Rails – компонент, що відповідає за графічне відображення (візуалізацію) даних нашого додатку. В даному проекті для спрощеної та ефективнішої роботи з відображеннями (представленнями) було вирішено обрати шаблонізатор Slim, що дозволяє писати більш простий та зрозумілий код, ніж базовий шаблонізатор Erb. Шаблонізатори необхідні в першу чергу для того, щоб вставляти на веб-сторінку частини коду на будь-якій мові програмування (в даному випадку – Ruby).

Всі представлення (views) зберігаються в директорії **app/views**. На рисунках 3.5, 3.6 та 3.7 представлено основні сторінки додатку: сторінка входу, список статей, сторінка керування власними статтями користувача.

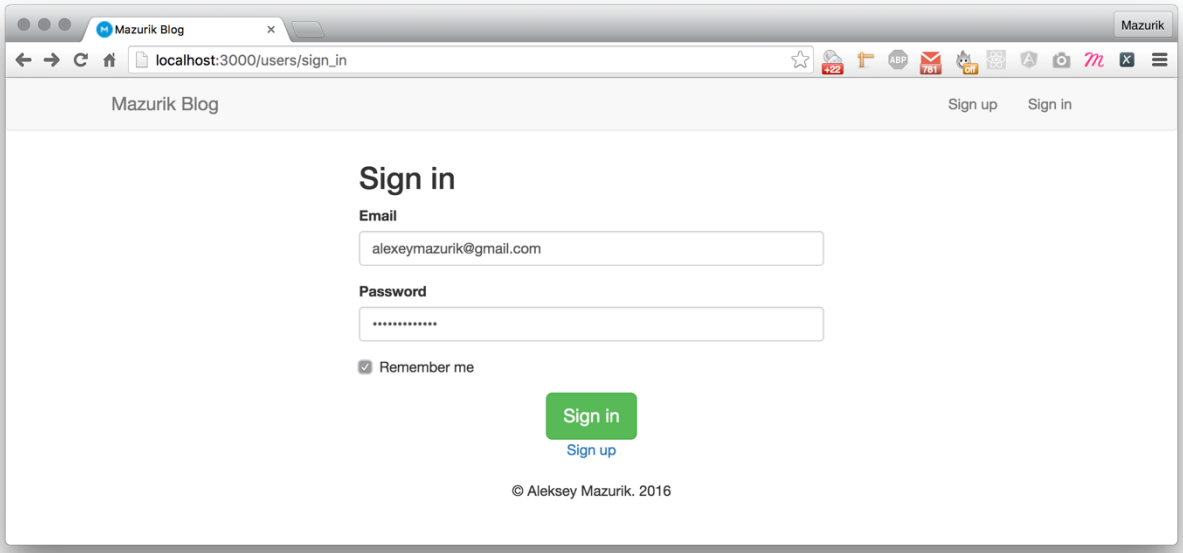

Рис. 3.5 - Представлення сторінки входу

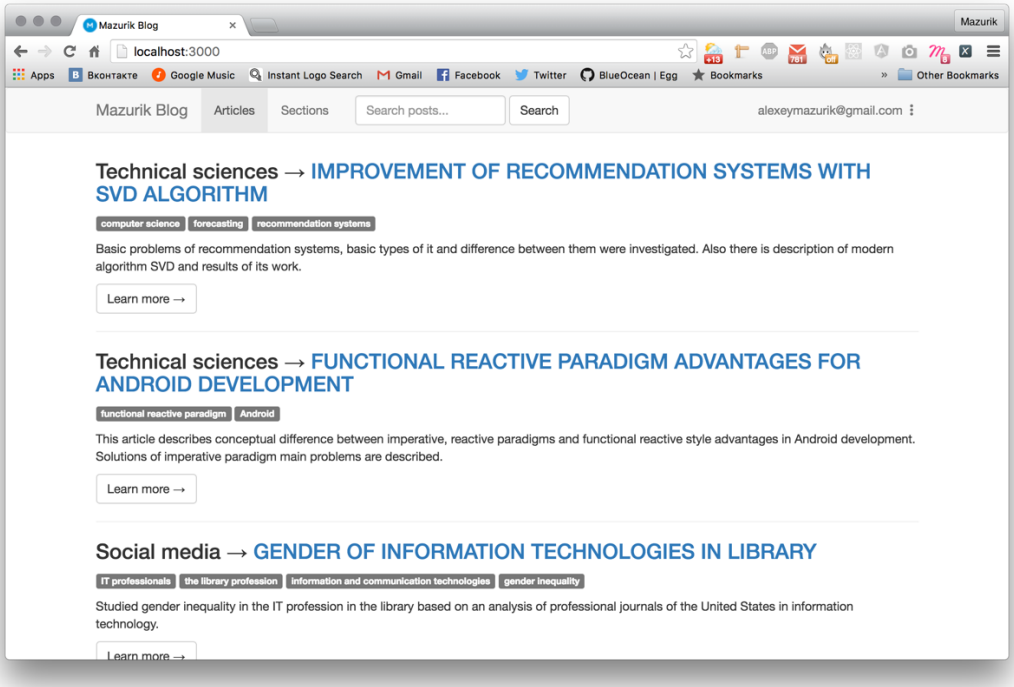

Рис. 3.6 - Представлення сторінки зі всіма статтями

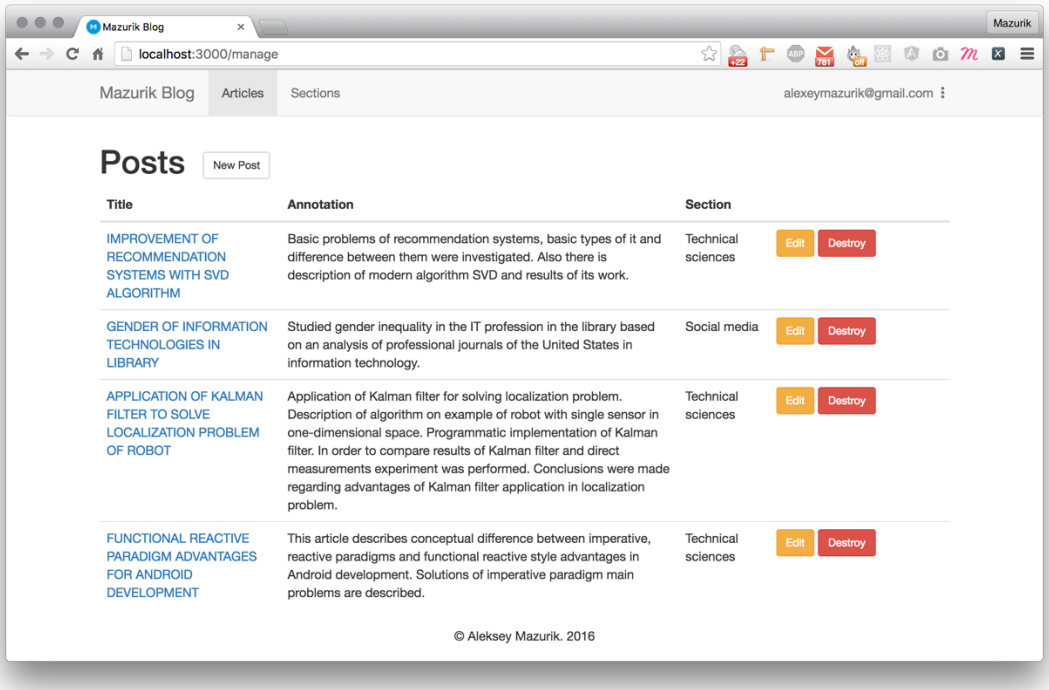

Рис. 3.7 - Представлення всіх статей користувача з можливістю адміністрування

Маршрутизація

Метою маршрутизації (роутинга) в будь-якому Rails проекті є зв’язування вхідних url (адрес сторінок) з конкретною дією контролера. Роутер в Rails розпізнає url, з’єднує запит користувача з необхідним контролером та віддає результат обробки запиту. Також маршрутизатор використовується для генерації шляхів (маршрутів) в додатку без необхідності явно прописувати кожен маршрут. Для цього використовуються спеціальні структури, які зрозуміло та ефективно описують блоки маршрутів. Нижче представлений приклад роутингу в нашому додатку з коментарями:

```
Rails.application.routes.draw do
 root to: 'posts#index' # переадресація головної сторінки
 get 'manage', to: 'posts#manage' # маршрут для керування постами
 resources :posts do # базові маршрути для статей
 member do
      # методи для оцінки постів (рекомендації)
 post :like
```

```
    delete :like, action: :unlike, as: :unlike
    post :dislike
    delete :dislike, action: :undislike, as: :undislike
    end
    end
    resources :sections, only: %w(index create update destroy) # маршрути для списку та керування секціями
    resources :tags, only: %w(index) # список тегів
    devise_for :users # роутинг для сторінок реєстрації та входу
  end
```

Міграції та автоматичне заповнення бази даних

Міграції – це одна з функцій Active Record, що довзволяє змінювати схему бази даних. Замість того, щоб описувати зміни схеми на чистому SQL, міграції в Rails дозволяються використовувати простий Ruby DSL для опису змін у ваших таблицях.

Кожну міграцію можна розглядати як нову версію бази даних. З самого початку схема нічого не містить, а кожна наступна міграція змінює її, додаючи чи видаляючи таблиці, стовпці або записи. Active Record за допомогою вбудованих кодових структур знає, як оновлювати вашу схему час від часу. Active Record також оновлює файл **db/schema.rb**, щоб він був синхронізований з поточною структурою вашої бази даних. Нижче наведений код міграції бази даних для створення таблиці статей:

```
class CreatePosts < ActiveRecord::Migration
  def change
  create_table :posts do |t|
  t.string :title
  t.text :text
  t.string :image_src
  t.string :author
```

```
    t.text :annotation
    t.references :user, index: true

    t.timestamps null: false
    end
    end
  end
```

Між міграціями можна зручно переключатися як вперед, так і назад. Файли міграцій необхідно завжди називати декларативно. Перша частина має містити випадково згенероване число (зазвичай – поточна дата і час, записані без розділових знаків), а друга – дію, що виконує ця міграція. Загальна структура - **db/migrate/YYYYMMDDHHMMSS_action_to_change_db.rb**.

Наприклад, цей файл називається **db/migrate/20160323000443_create_posts.rb**.

Після створення міграції необхідно промігрувати базу даних до найновішої версії за допомогою команди `rake db:migrate`.

Для початкового заповнення бази даних також інколи використовують міграції. Але більш коректний спосіб – сидування бази даних. Для цього необхідно написати код на Rails, який буде записувати в базу даних потрібну інформацію, використовуючи API Active Record.

Щоб наповнити базу текстами статей, було використано ресурс http://www.inter-nauka.com/ , де розміщені свіжі наукові публікації для різних секцій, тому можна доволі повно заповнити ресурс статтями різних розділів, щоб прослідкувати крок за кроком роботу рекомендаційного алгоритму, вбудованого в систему.

Для парсингу даних з обраного ресурсу вирішено було використовувати бібліотеку Nokogiri, оскільки вона має об’ємний функціонал для парсингу

різних типів документів, оформлених на мові розмітки (HTML, XML), та є на сьогодні швейцарським ножем для парсингу веб-сайтів будь-якої складності. Код для парсингу представлений нижче:

```
require 'open-uri'

SECTIONS = [ 'Technical sciences', ... ]

ARTICLE_URLS = [ 'http://www.inter-nauka.com/issues/2015/9/592/', ... ]

user = User.create!( email: 'admin@mazurik.me', password: 'password',
 password_confirmation: 'password', admin: true
)

SECTIONS.each do |section|
 Section.create!(name: section)
end

ARTICLE_URLS.each do |url|
 en_url = url.split('/').insert(3,'en').join('/')

 page = Nokogiri::HTML(open(en_url))
 title = page.css('div[record] .page-header a').text
 author = ''
 authors = page.css('div[record] .page-header + .row a')
 if authors.size == 1
 author = authors.text
 elsif authors.size > 1
 authors.each_with_index { |a, i| i == (authors.size - 1) ? author +=
a.text : author += a.text.concat(', ') }
 end
 basic_info = page.css('div[record] br+div p')
 basic_info.map do |p|
 p.css('strong').remove
 end
 summary, keywords = basic_info.map(&:text)
```

```
        keywords = keywords.chop.split(', ')
        section = page.css('b:contains("Branch of science:") + a').text

        # Parsing text of an article
        page = Nokogiri::HTML(open(url)).css('hr + div')
        page.search('div[style="text-align: center;"]').each { |div| div.remove }
        page.search('p').each do |p|
        p.remove
        break if p.text.downcase.include?('key words:')
        end

        article_text = page.children.to_html.strip

        Post.create!( title: title, text: article_text, annotation: summary,
            user_id: user.id, author: author, section_id: Section.where(name:
section).first.id, tag_list: keywords
        )
      end
```

Після написання коду для парсингу даних необхідно заповнити базу даних. Це можна зробити за допомогою команди `rake db:seed`. Вона просто запускає файл **db/seeds.rb** в контексті нашого додатку та наповнює потрібну базу.

3.4 Аналіз результатів роботи рекомендаційної системи

В результаті роботи над блогом із вбудованою рекомендаційною системою було створено додаток, що містить авторизацію користувачів, інструменти для додавання редагування статей (включаючи секції та теги) та модуль парсингу веб-сторінок для наповнення додатку тестовими даними.

Всього було стягнуто 14 статей з ресурсу http://www.inter-nauka.com/. Їх перелік та нумерація для подальшої обробки результатів представлена нижче:

1. Improvement of recommendation systems with SVD algorithm (технічні науки)

2. Application of kalman filter to solve localization problem of robot (технічні науки)
3. Functional reactive paradigm advantages for android development (технічні науки)
4. Research of eigenface algorithm for face recognition and its realization in matlab (технічні науки)
5. Comparative analysis of texture recognition methods (технічні науки)
6. Comparison of accuracy of sentiment analysis algorithm on twitter messages (технічні науки)
7. To the question of legal regulations providing of ecological safety in ukraine (право)
8. The responsibility of officials for violation of property rights intelektualno (право)
9. Gender of information technologies in library (соціальні медіа)
10. Theoretical understanding for formation's tendencies of modern society's communicative culture (соціальні медіа)
11. Tax system - present and past (економічні науки)
12. Statistical analysis criminal violation (економічні економічні науки)
13. Theory of consumer behavior in the internet (технічні науки)
14. Features component of human potential in the context of sustainable development (економічні науки)

Було відібрано тестову групу користувачів у складі 5 людей. Систему було протестовано на тестовій групі, яка користувалася блогом протягом деякого часу. В ході тестування системи було отримано результати, зображені в Таблиці 3.2. Користувачі виставляли оцінки статтям: 1 (сподобалось), -1 (не сподобалось). Є статті, які користувачі ніяк не оцінили.

Таблиця 3.2 – Зведені дані про оцінки користувачів

Розділ	**№ статті**	**Оцінки користувачів**				
		Іванна	Дмитро	Ігор	Олексій	Олександр
Технічні науки	1	-1	1		1	1
	2		1	-1	1	
	3	1	-1		1	
	4		1			
	5		-1			
	6	-1	1			
Право	7		-1	-1	1	
	8		-1	-1	1	
Соціальні медіа	9	1		-1		
	10	1		-1		
Економічні науки	11	1	1	-1		
	12	1	-1	-1		
	13		1	1		
	14			1		

Під час тестування користувач №5 (Олександр) оцінив лише одну статтю. Нижче будуть наведені результати роботи системи рекомендацій для нового користувача, адже доволі важливо побачити, як система буде працювати під час холодного старту.

Користувач №5 оцінив статтю №1 як сподобалось. Результат його оцінки та рекомендацій можна знайти на рисунку 3.8.

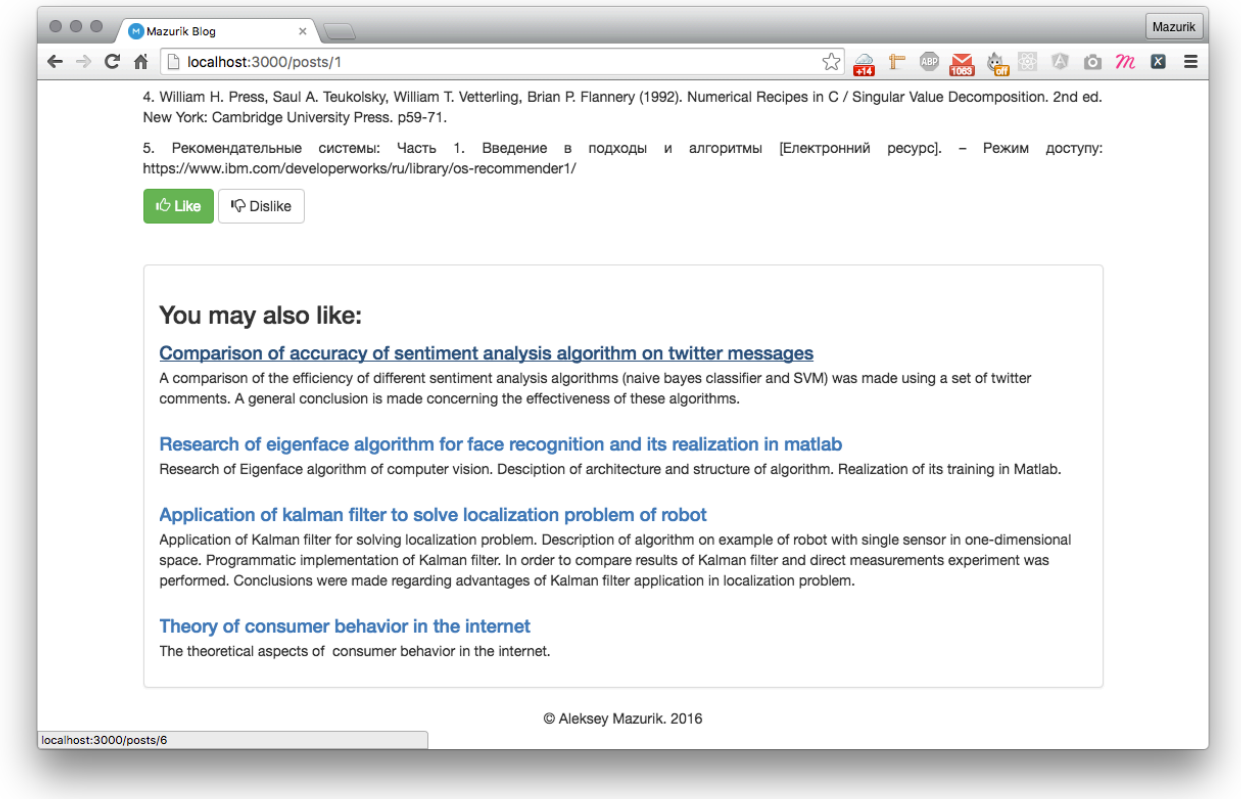

Рисунок 3.8 – Приклад рекомендацій користувачу

Система обрахувала схожих користувачів на основі статті, що користувач №5 оцінив. Схожі користувачі до цього - №2 (Дмитро) та №4 (Олексій), оскільки вони також оцінили статтю №1 як ту, що сподобалася. Система обрала для нього статті, які за статистикою вірогідніше всього сподобаються користувачу. Ними виявилися статті з номерами №6, №4, №2 та №13. При наданні рекомендацій статті сортуються за спаданням (від статті, що вірогідніше сподобається користувачу).

В будь-який момент для будь-якого користувача можна отримати рекомендації контенту системи (за наявності хоча б однієї оцінки), список схожих користувачів, найкращі товари та інші параметри.

Як можна побачити рекомендації, що були отримані під час роботи з сервісом, виявилися релевантними та актуальними для користувача, якщо оцінювати якість алгоритму з точки зору релевантності рекомендацій для методу колаборативної фільтрації. Отже, таким чином, таку рекомендаційну

систему можна імплементувати в будь-якому проекті, створеному за допомогою Rails, без глибокого занурення в рекомендаційні алгоритми. Результати роботи виявилися непоганими, контент, що рекомендувався – релевантним.

3.5 Висновок

Розробка веб-додатків за допомою Rails – ефективне рішення, якщо необхідно швидко розробити доволі складний, але типовий інтернет-сервіс, використовуючи патерн MVC. Це рішення широко застосовується для створення комерційних інтернет-сервісів (магазини, журнали тощо).

Використання рекомендаційних систем в додатках значно збільшує прибутковість сервісів, оскільки збільшується залученість користувачів до вашого сервісу. База Rails дозволяє створювати гнучкі рішення стосовно імлементації готових систем до вашого сервісу або написання їх з нуля. В будь-якому випадку це не буде великою проблемою, оскільки Rails використовує доволі зрозумілі методи для інтеграції зовнішніх або внутрішніх модулів до існуючої системи.

В даному розділі було детально описано процес реалізації веб-додатку з використанням Rails, проаналізовано обумовленість вибору фреймворку для розробки інтернет-блогу, особливості реалізації рекомендаційної системи. Звичайно, було протестовано та проаналізовано модуль веб-додатку, що відповідає за надання користувачам сервісу рекомендації.

В результаті дослідження роботи реалізованої рекомендаційної системи було зроблено висновок, що система поводить себе коректно, рекомендації – релевантні. Тому є сенс використовувати поточне рішення в контексті інтернет-маркетингу, щоб збільшувати процент конверсії, тобто залучати користувачів до необхідних дій, а саме – рекомендувати якомога більше релевантного контенту, щоб заслужити довіру користувачів.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для надання рекомендацій користувачам щодо релевантного контенту. Програмний продукт був розроблений за допомогою мови програмування Ruby.

Програмний продукт є кросплатформенним та рекомендується для використання на персональних комп'ютерах під управлінням операційних систем Windows, Linux чи Mac.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом: визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

Алгоритм методу:

- для кожної функції визначаються повні річні витрати й кількість робочих часів.
- для кожної функції наоснові оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
- після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.1 Постановка задачі

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозування рекомендацій контенту для користувачів. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для надання рекомендацій користувачам після оцінки контенту.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;

- забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;
- забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;
- передбачати мінімальні витрати на впровадження програмного продукту.

4.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який прогнозує рекомендації на базі оцінок користувачів. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

- F_1 – вибір мови програмування;
- F_2 – використання готових бібліотек;
- F_3 – вибір методу перевірки релевантності рекомендацій.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

a) мова програмування Ruby;
b) мова програмування Python;

Функція F_2:

a) написання алгоритму рекомендації вручну;
b) використання готової бібліотеки;

Функція F_3:

a) перевірка точності на наборі даних для тестування;
b) перевірка точності шляхом А/Б тестування.

4.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

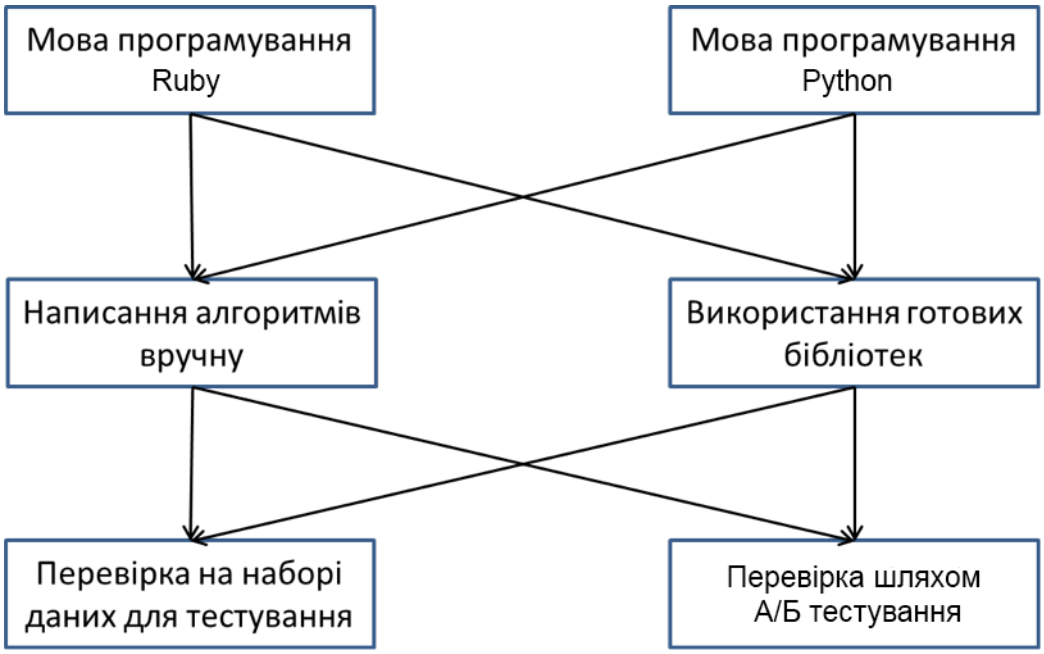

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

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.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Займає менше часу при написанні коду	Код повільніше виконується
	Б	Код швидше виконується	Необхідно більше часу для написання коду
F2	А	Більша гнучність у використанні	Необхідно більше часу для написання коду
	Б	Займає менше часу при написанні коду	Менша гнучкість у використанні
F3	*А*	Обчислювально легший	Оцінка точності не завжди відповідає реальній
	Б	Дає найбільш наближену до реальності оцінку	Займає велику кількість часу

Функція *F1:*

Оскільки проводиться оцінка великої кількості методів, потрібно швидко їх реалізовувати, тому варіант б) має бути відкинутий.

Функція *F2:*

Оскільки Ruby динамічна мова програмування, то готова бібліотека може бути трохи видозмінена (збільшено гнучкість рекомендаційної системи). Отож варіант а) має бути відкинутий.

Функція *F3:*

Оцінювання якості рекомендацій найважливіше в цьому продукті, тому мають бути проаналізованими обидва варіанти а) та б).

1. F1а – F2б – F3а
2. F1а – F2б – F3б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів ПП

4.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

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

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для збереження даних;
- *X3* – час обробки даних;
- *X4* – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті сервера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

4.2.2 Кількісна оцінка параметрів

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

Таблиця 4.2 – Основні параметри ПП

Назва параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	*середні*	*кращі*
Швидкодія мови програмування	X1	Оп/мс	2000	11000	19000
Об'єм пам'яті для обробки даних	X2	Мб	32	16	8
Час обробки даних алгоритмом	X3	мс	800	420	60
Потенційний об'єм програмного коду	X4	кількість строк коду	2000	1500	1000

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

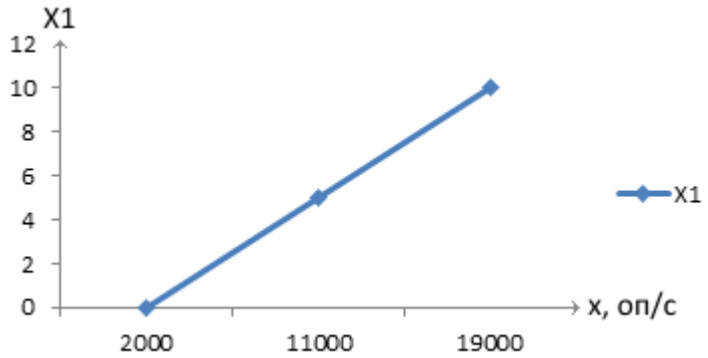

Рисунок 4.2 – X1, швидкодія мови програмування

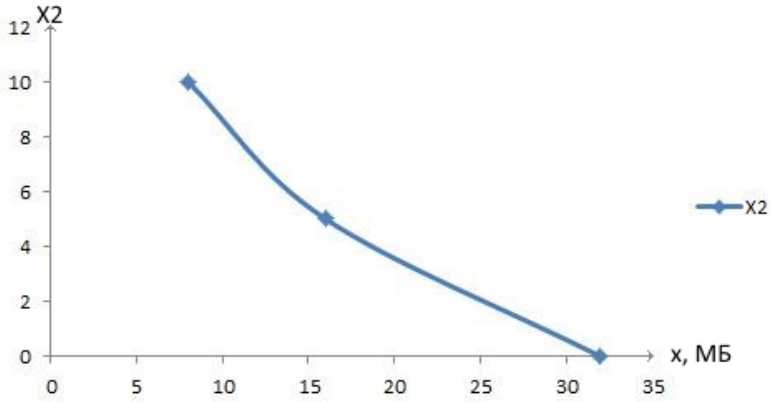

Рисунок 4.3 – X2, об’єм пам’яті для збереження даних

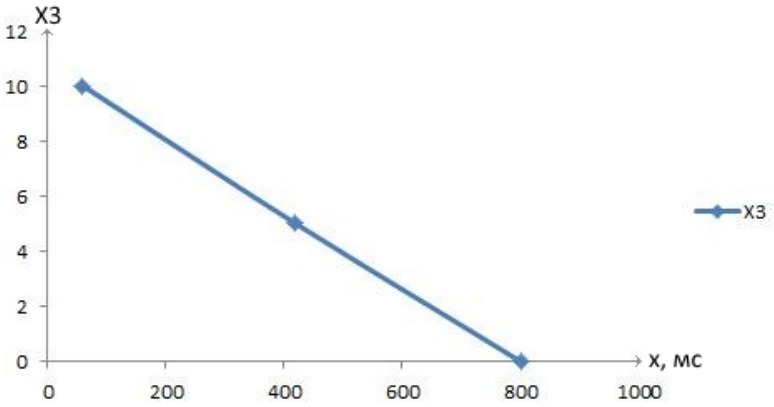

Рисунок 4.4 – X3, час обробки даних алгоритмом

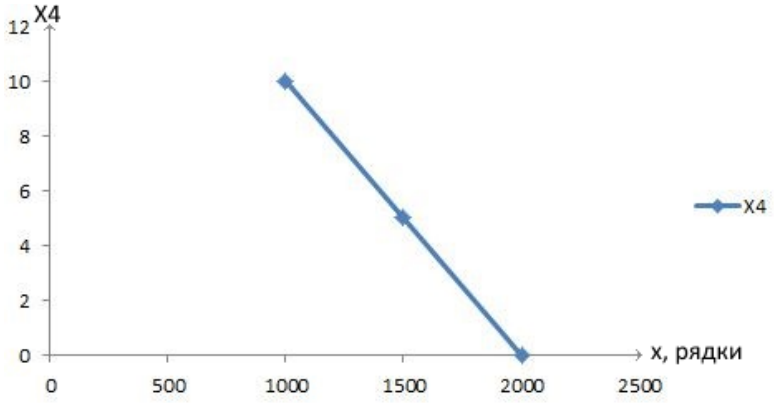

Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.2.3 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 прогнозуванні рекомендацій контенту для користувачів на базі відомих оцінок товарів.

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	4	3	4	4	4	4	4	27	0.75	0.56
X2	Об’єм пам’яті для збереження даних	Мб	4	4	4	3	4	3	3	25	-1.25	1.56
X3	Час обробки даних алгоритмом	Мс	2	2	1	2	1	2	2	12	-14.25	203.06
X4	Потенційний об’єм програмного коду	кількість строк коду	5	6	6	6	6	6	6	41	14.75	217.56
	Разом		15	15	15	15	15	15	15	105	0	420.75

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 105 \quad (4.1)$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 26.25 \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 420.75 \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 420.75}{7^2(5^3 - 5)} = 1{,}03 > W_k = 0.67 \quad (4.5)$$

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
X1 і X4	>	>	>	>	>	>	>	>	1.5
X2 і X3	<	<	<	<	<	<	<	<	0.5
X2 і X4	>	>	>	>	>	>	>	>	1.5
X3 і X4	>	>	>	>	>	>	>	>	1.5

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

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

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

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

$$K_{\text{Bi}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \quad (4.7)$$

де bi розраховується за наступною формулою:

$$b_i = \sum_{i=1}^{N} a_{ij}. \quad (4.8)$$

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:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (4.9)$$

де b`$_i$ розраховується за наступною формулою

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j. \quad (4.10)$$

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.**		**Третя ітер**	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1.0	0.5	0.5	1.5	3.5	0.219	22.25	0.216	100	0.215
X2	1.5	1.0	0.5	1.5	4.5	0.281	27.25	0.282	124.25	0.283
X3	1.5	1.5	1.0	1.5	5.5	0.344	34.25	0.347	156	0.348
X4	0.5	0.5	0.5	1.0	2.5	0.156	14.25	0.155	64.75	0.154
Всього:					16	1	98	1	445	1

4.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2*(об'єм пам'яті для збереження даних) *X1* (швидкодія мови програмування) та *X3* відповідають технічним вимогам умов функціонування даного ПП.1

Абсолютне значення параметра *X4 (*потенційний об'єм програмного коду) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 1800 або варіанту б) 1200.

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

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j}, \tag{4.11}$$

де n – кількість параметрів;$K_{\text{в}i}$– 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**	**Абсолютне значення параметра**	**Бальна оцінка параметра**	**Коефіцієнт вагомості параметра**	**Коефіцієнт рівня якості**
F1(X1)	А	11000	3.6	0.215	0.774
F2(X3)	А, Б	800	2.4	0.348	0.835
F2(X4)	А	1800	2	0.154	0.308
	Б	1200	8	0.154	1.232
F3(X2)	Б	16	3.4	0.283	0.962

За даними з таблиці 4.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$K_{K1} = 0.774 + 0.835 + 0.308 + 0.962 = 2.879$

$K_{K2} = 0.774 + 0.835 + 1.232 + 0.962 = 3.803$

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

4.4 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;

При цьому варіант 3 має додаткове завдання:

3. Реалізація методів аналізу;

А варіант 4 має інше додаткове завдання:

4. Обробка інтерфейсу готових бібліотек.

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 2.

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

$$T_О = T_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М}, \quad (4.13)$$

де $T_Р$ – трудомісткість розробки ПП;$К_П$ – поправочний коефіцієнт;$К_{СК}$ – коефіцієнт на складність вхідної інформації; $К_М$ – коефіцієнт рівня мови програмування;$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;$К_{СТ.М}$ – 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: $T_Р$ =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4$ людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм другої групи складності, степінь новизни Б), тобто $T_Р$ =27 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44$ людино-днів.

Для третього завдання (використовується алгоритм другої групи складності, ступінь новизни Г з використанням перемінної інформації):

Тр = 12 людино-днів;

Кп = 0.72; Кст = 0.8;

То = 12 • 0.72 • 0,8 = 6.91.

Для четвертого завдання (використовується алгоритм третьої групи складності, ступінь новизни Г):

Тр = 8 людино-днів;

Кп = 0.6; Кст = 1;

То = 8 • 0.6 • 1 = 4.8.

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

$Т_{I}$ = (122.4 + 19.44 + 6.91) · 8 = 1190 людино-годин;

$Т_{II}$ = (122.4 + 19.44 + 4.8) · 8 = 1173.12 людино-годин;

Найбільш високу трудомісткість має варіант І.

В розробці беруть участь два програмісти з окладом 6000 грн., один фінансовий аналітик з окладом 9000 грн. Визначимо зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \cdot t} \text{ грн.}, \quad (4.14)$$

де М – місячний оклад працівників; T_m – кількість робочих днів тиждень; t – кількість робочих годин в день.

$$СЧ = \frac{6000 + 6000 + 9000}{3 \cdot 21 \cdot 8} = 41.67 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$СЗП = С_{ч} \cdot Т_{i} \cdot КД, \quad (4.15)$$

де $С_ч$– величина погодинної оплати праці програміста; $Т_i$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 41.67 \cdot 1190 \cdot 1.2 = 59504.76$ грн.

II. $С_{ЗП} = 41.67 \cdot 1173.12 \cdot 1.2 = 58660.69$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 59504.76 \cdot 0.22 = 13091.05$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 58660.69 \cdot 0.22 = 12905.35$ грн.

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

Так як одна ЕОМ обслуговує одного програміста з окладом 6000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 6000 \cdot 0.2 = 14400$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 14400 \cdot (1 + 0.2) = 17280$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 17280 \cdot 0.22 = 3801.6$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 8000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 8000 = 2300$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 8000 \cdot 0.05 = 460$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4$ годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706.4 \cdot 0.156 \cdot 0.2 \cdot 2.0218 = 107.64$ грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 8000 \cdot 0.67 = 5360$ грн.

Тоді, річні експлуатаційні витрати будуть:

$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$

$С_{ЕКС} = 17280 + 3801.6 + 2300 + 460 + 107.64 + 5360 = 29309.24$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ} = 29309.24 / 1706.4 = 17.18$ грн/час.

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:

$С_М = С_{М-Г} \cdot Т$

I. $С_М = 17.18 \cdot 1190 = 20444.2$ грн.;

II. $С_М = 17.18 \cdot 1173.12 = 20154.2$ грн.;

Накладні витрати складають 67% від заробітної плати:

$С_Н = С_{ЗП} \cdot 0.67$

I. $С_Н = 59504.76 \cdot 0.67 = 39868.19$ грн.;

II. $С_Н = 58660.69 \cdot 0.67 = 39302.66$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$

I. $С_{ПП} = 59504.76 + 13091.05 + 20444.2 + 39868.19 = 132908.2$ грн.;

II. $С_{ПП} = 58660.69 + 12905.35 + 20154.2 + 39302.66 = 131022.9$ грн.;

4.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj}, \quad (4.16)$$

$К_{ТЕР1} = 2.879 / 132908.2 = 0.22 \cdot 10^{-4}$;

$К_{ТЕР2} = 3.803 / 131022.9 = 0.29 \cdot 10^{-4}$;

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 0.29 \cdot 10^{-4}$.

4.6 Висновок

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

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 $К_{ТЕР} = 0.29 \cdot 10^{-4}$. Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Ruby;
- використання готової бібліотеки;
- перевірка точності на наборі тестових даних.

ВИСНОВКИ

В результаті виконання роботи було проаналізовано основні алгоритми та базові підходи для прогнозування рекомендацій. Результати аналізу представлені на таблиці 1.4. Було описано особливості, переваги та недоліки кожного з них та виконано порівняльний аналіз існуючих підходів. Також було зроблено огляд допоміжних засобів для кореляції результатів, отриманих після застосування основних алгоритмів.

Алгоритми колаборативної фільтрації дозволяють швидко створювати та інтегрувати рекомендаційні системи, адже вони засновані лише на пошуку схожих користувачів. Тому колаборативна фільтрація найпоширеніша в інтернет-сервісах, що використовують рекомендаційні системи. Такі системи зазвичай підходять для невеликих блогів, інформаційних порталів тощо, бо в цих випадках характерні проблеми для рекомендаційних систем не матимуть широкого впливу.

Алгоритми контентної фільтрації прогнозують рекомендації завдяки створенню моделей користувачів та товарів. Рекомендаційні системи засновані на цьому підході потребують значно більше ресурсів для створення, ніж ті, що використовують колаборивну фільтрацію. Але вони не мають проблем холодного старту, унікальних користувачів, шахрайства тощо. Такі підходи зазвичай використовуються для створення інтернет-магазинів, де рекомендації товарів важливо підбирати під конкретного користувача.

Гібридні алгоритми мають на меті знизити вплив звичних для колаборативної та контентної фільтрації проблем. Але такі системи мають використовуватися виправдано, оскільки вони громіздкі, на їх побудову необхідно витратити велику кількість часу, коштів та людских ресурсів. Вони

зазвичай використовуються в крупних інтернет-магазинах та соціальних мережах.

Були розглянуті шляхи до подолання типових недоліків базових алгоритмів. Для вирішення недоліків колаборативної фільтрації використовується алгоритм SVD. Адже з ростом кількості користувачів та їх оцінок в будь-якому інтернет-сервісі буде зростати розмірність матриці оцінок. Цей алгоритм дозволяє зменшити розмірність цієї матриці з невеликою втратою точності. Щоб пришвидшити роботу алогоритмів контентної фільтрації зазвичай використовують алгоритм TF-IDF, який дозволяє будувати рейтингові списки документів великого корпусу документів.

Також було виконано порівняльний аналіз існуючих рішень імлементації рекомендаційних систем у відомі інтернет-компанії, які збільшили свої прибутки саме через інтеграцію рекомендаційних систем до каталогу своїх продуктів та сервісів. Було проаналізовано рекомендаційні системи та модулі широковідомих компаній Amazon та eBay. Amazon використовує більшу кількість рекомендаційних сервісів на базі гібридної фільтрації, що дає їм більше можливостей для надання релевантніших рекомендацій. Як можна побачити з таблиці 2.1, Amazon під кожну задачу використовує конкретний алгоритм, тому важливо проводити дослідження з вибору алгоритму для рекомендаційної системи на її налаштування під конкретну задачу, оскільки в майбутньому це зможе значно покращити результати прогнозуванн рекомендацій.

В якості демонстрації проведеного аналізу базових алгоритмів було реалізовано веб-додаток із інтегрованою рекомендаційною системою на базі колаборативної фільтрації. Платформою для створення інтернет-сервісу було обрано фреймворк Ruby on Rails, який дозволяє ефективно створювати проекти високої складності з використанням різних модулів (внутрішніх та зовнішніх).

Rails – сучасний стандарт майже усіх веб-додатків для бізнесу, який дозволяє швидко створювати прототипи веб-додатків. Вбудована рекомендаційна система на основі колаборативної фільтрації має на меті збільшити залученість користувачів до сервісу та збільшити його конверсію. Отримані результати від створеної рекомендаційної системи виявилися задовільними: прогнозовані рекомендації виявилися релевантними, а алгоритм пошуку схожих користувачів досить точно виявляв групу схожих користувачів.

Так як в даному проекті було реалізовано веб-додаток з рекомендаційною системою на базі колаборативної фільтрації, в подальшому можна зробити більш глибоке та детальне дослідження більш ефективних способів реалізації, а саме – гібридних алгоритмів. Розвиток інтернет-сервісів рухається в сторону логічного розділення модулів продукту, тому можна розробити незалежний сервіс для рекомендацій будь-якого контенту із використанням відкритого API.

ПЕРЕЛІК ПОСИЛАНЬ

1. Melville P. Content-Boosted Collaborative Filtering for Improved Recommendations / Melville P.,Mooney R., Nagarajan R. // National Conference on Artificial Intelligence : «AAAI-2002», 20-25 July 2016, Edmonton, Canada : materials. – Edmonton, Canada : AAAI, 2002. – C. 187-192.
2. Sarwar B. Item-Based Collaborative Filtering Recommendation Algorithms / Sarwar B., Karypis G., Konstan J., and Riedl J. // Hong Kong WWW10 Conference : «WWW10», 20-25 January 2001, Hong Kong : materials. – Hong Kong : WWW10 Press, 2001. – C.285-289.
3. William H. Press. Numerical Recipes in C 2nd ed./ William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. – New York, USA : Cambridge University Press, 1992. – C. 59-71.
4. Linden G. Item-to-Item Collaborative Filtering. / Linden G., Smith B., and York J. – Los Alamitos, CA, USA: IEEE Internet Computing, 2003. – C. 76-80.
5. Мазурік О.Ю. Покращення результатів роботи рекомендаційних алгоритмів за допомогою алгоритму SVD / Мазурік О.Ю. // International Scientific Journal. – 2015. – #9. – С. 61-65.
6. Офіційний блог для розробників компанії IBM. – Режим доступа : https://www.ibm.com/developerworks/ru/library/os-recommender1/. − Дата доступа : 20.04.2016.
7. Офіційний блог компанії Surfingbird. – Режим доступа : https://habrahabr.ru/company/surfingbird/. − Дата доступа : 30.05.2016.
8. Мазурік О.Ю. Порівняльний аналіз моделей оцінювання в рекомендаційних системах / Мазурік О.Ю. // Системний аналіз та інформаційні технології : «САІТ-2016», 30 травня –2 червня 2016, Київ, Україна : матеріали. – К. : НТУУ «КПІ», 2016. – С. 113

9. David Celis Blog: Why I Hate Five-Star Ratings. – Режим доступа: https://davidcel.is/posts/why-i-hate-five-star-ratings/. − Дата доступа: 15.03.2016.
10. Филиппов С.А. Организация больших объемов данных в рекомендательных системах поддержки жизнеобеспечения, входящих в состав глобальных платформ электронной коммерции / Филиппов С.А., В.Н. Захаров, С.А. Ступников, Д.Ю. Ковалев // Data Analytics and Management in Data Intensive Domains : «DAMDID/RCDL-2015», 13 – 16 октября 2015, Обнинск, Россия : материалы. – М. : Институт проблем информатики ФИЦ ИУ РАН, 2015. – С. 119-124
11. J. Ben Schafer. Recommender Systems in E-Commerce / J. Ben Schafer, Joseph Konstan, John Riedl // GroupLens Research Project Book. – Minneapolis, Minnesota, USA: University of Minnesota, 2011. – С. 43 – 55.
12. Офіційна сторінка компанії A9. – Режим доступа: https://www.a9.com/. − Дата доступа: 30.05.2016.
13. David Celis Blog: Collaborative Filtering With Likes and Dislikes. – Режим доступа: https://davidcel.is/posts/collaborative-filtering-with-likes-and-dislikes/. − Дата доступа: 25.04.2016.
14. Hartl M. Ruby on Rails Tutorial: Learn Web Development with Rails (3rd Edition) / Hartl M. –New Jersey, USA : Addison-Wesley, 2015. – С. 100-250.
15. Офіційний блог для розробників компанії DigitalOcean. – Режим доступа: https://www.digitalocean.com/community/tutorials/how-to-automate-ruby-on-rails-application-deployments-using-capistrano. − Дата доступа : 25.05.2016.
16. Офіційний блог компанії Make Agency. – Режим доступа: http://makeagency.ru/blog/item/pochemu-ruby-on-rails. − Дата доступа : 20.05.2016.
17. Офіціний сайт змагання Netflix Prize. – Режим доступа: http://www.netflixprize.com/. – Дата доступа : 20.01.2016

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Контролери

Лістинг файлу app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
 # Prevent CSRF attacks by raising an exception.
 # For APIs, you may want to use :null_session instead.
 protect_from_forgery with: :exception
 before_action :authenticate_user!
end
```

Лістинг файлу app/controllers/posts_controller.rb

```
class PostsController < ApplicationController
 RECOMMENDABLE_METHODS = %w(like unlike dislike undislike).freeze

 before_action :set_post, only: %w(show edit update
destroy).concat(RECOMMENDABLE_METHODS)
 before_action :fetch_sections, only: %w(index manage show edit update new
create)
 before_action :post_owner, only: %w(edit update destroy)
 respond_to :json, only: RECOMMENDABLE_METHODS

 # GET /posts
 def index
 @q = Post.ransack(params[:q])
 @q.sorts = 'section_id asc' if @q.sorts.empty?
 @posts = @q.result(distinct: true)
 end

 # GEt /posts/manage
 def manage
 if current_user.admin?
```

```
@posts = Post.all
else
@posts = current_user.posts
end
end

# GET /posts/1
def show
end

# GET /posts/new
def new
@post = Post.new
end

# GET /posts/1/edit
def edit
end

# POST /posts
def create
@post = Post.new(post_params)

if @post.save
redirect_to @post, notice: 'Post was successfully created.'
else
render :new
end
end

# PATCH/PUT /posts/1
def update

if @post.update(post_params)
redirect_to posts_path, notice: 'Post was successfully updated.'
else
```

```
render :edit
end
end

# DELETE /posts/1
def destroy
@post.destroy
redirect_to posts_url, notice: 'Post was successfully destroyed.'
end

RECOMMENDABLE_METHODS.each do |method|
define_method(method) do
@post = Post.find(params[:id])

if current_user.send(method, @post)
head :ok
else
head :unprocessable_entity
end
end
end

private

# Use callbacks to share common setup or constraints between actions.
def set_post
@post = Post.find(params[:id])
end

def fetch_sections
@sections = Section.all
end

# Only allow a trusted parameter 'white list' through.
def post_params
params.require(:post).permit(:title, :text, :annotation, :image_src,
:section_id, :user_id, :author, tag_list: [])
```

```
 end

 def post_owner
 unless @post.user_id == current_user.id || current_user.admin?
 flash[:notice] = 'Access denied as you are not owner of this job'
 redirect_to posts_path
 end
 end
end
```

Лістинг файлу app/controllers/sections_controller.rb

```
class SectionsController < ApplicationController
 before_action :set_section, only: %w(update destroy)

 def index
 @section = Section.new
 @sections = Section.all.order(:id)
 end

 def create
 @section = Section.new(section_params)
 if @section.save
 redirect_to sections_path, notice: 'Section was successfully created.'
 else
 redirect_to sections_path, notice: 'Some errors occurred.'
 end
 end

 def update
 if @section.update(section_params)
 redirect_to sections_path, notice: 'Section was successfully updated.'
 else
 redirect_to sections_path, notice: 'Some errors occurred.'
 end
 end

 # DELETE /sections/1
```

```
 def destroy
 @section.destroy
 redirect_to sections_path, notice: 'Section was successfully destroyed.'
 end

 private

 def set_section
 @section = Section.find(params[:id])
 end

 def section_params
 params.require(:section).permit(:name)
 end
end
```

Лістинг файлу app/controllers/tags_controller.rb

```
class TagsController < ApplicationController
 def index
 @tags = ActsAsTaggableOn::Tag.all.order(:id)
 end
end
```

Моделі

Лістинг файлу app/models/post.rb

```
class Post < ActiveRecord::Base
 acts_as_taggable_on :tags
 belongs_to :user
 belongs_to :section

 validates :title, presence: true, length: 1..255
end
```

Лістинг файлу app/models/section.rb

```
class Section < ActiveRecord::Base
 has_many :posts, dependent: :destroy
```

```
 validates :name, presence: true, length: 1..255
end
```

Лістинг файлу app/models/user.rb

```
class User < ActiveRecord::Base
 # Include default devise modules. Others available are:
 # :confirmable, :lockable, :timeoutable and :omniauthable
 devise :database_authenticatable, :registerable,
 :rememberable, :trackable, :validatable

 recommends :posts

 has_many :posts, dependent: :destroy
end
```

Представлення

Лістинг файлу app/views/layouts/application.html.slim

```
doctype html
html
 head
 title
 | Mazurik Blog
 = stylesheet_link_tag 'application', media: 'all'
 = javascript_include_tag 'application'
 = csrf_meta_tags
 body
 nav.navbar.navbar-default
 .container
 .navbar-header
 a.navbar-brand[href="/"]
 | Mazurik Blog
 - if current_user
 ul.navbar-nav.nav
 = active_link_to 'Articles', posts_path, 'posts'
 = active_link_to 'Sections', sections_path, 'sections'
```

```
 - if current_user.present? && current_page?(controller: 'posts', action:
'index')
 = search_form_for @q, html: { class: 'navbar-form navbar-left' } do |f|
 .form-group
 = f.search_field :title_cont, placeholder: 'Search posts...', class: 'form-
control'
 = f.submit 'Search', class: 'btn btn-default'
 ul.navbar-nav.nav.navbar-right
 - unless current_user
 li
 = link_to 'Sign up', new_user_registration_path
 li
 = link_to 'Sign in', new_user_session_path
 - else
 li.dropdown
 a.dropdown-toggle[href="#" data-toggle="dropdown" role="button" aria-
haspopup="true" aria-expanded="false"]
 = current_user.email
 |  
 span.glyphicon.glyphicon-option-vertical
 ul.dropdown-menu
 li
 = link_to 'My posts', manage_path
 li.divider
 li
 = link_to 'Sign Out', destroy_user_session_path, method: :delete


 .container
 - if notice
 p.alert.alert-warning
 = notice
 = yield
 footer
 .text-center
 | © Aleksey Mazurik. 2016
```

Лістинг файлу app/views/posts/_form.html.slim

```
= simple_form_for(@post) do |f|
```

```
 .form-group
 label.control-label
 | Author:  
 = current_user.email
 = f.input :title, placeholder: 'Title...', hint: false
 = f.input :annotation, placeholder: 'Annotation...'
 = f.input :section_id, collection: @sections, selected: @post.section_id
 = f.input :user_id, as: :hidden, input_html: { value: current_user.id }
 .form-group
 label.control-label for='post[tag_list][]'
 | Tags:
 select.form-control.js-select2-select multiple="multiple"
name="post[tag_list][]"
 - ActsAsTaggableOn::Tag.all.pluck(:name).each do |tag|
 - if tag.in?(@post.tag_list)
 option value="#{tag}" selected='selected' #{tag}
 - else
 option value="#{tag}" #{tag}
 = f.input :image_src, placeholder: 'Image URL...'
 = f.input :text, placeholder: 'Blog text type here...', input_html: {rows:
'30'}
 = f.button :submit
```

Лістинг файлу app/views/posts/index.html.slim

```
- @posts.each do |post|
 h3
 = post.section.name
 |  
 | &#8594;
 |  
 = link_to post do
 = post.title
 |  
 p
 - post.tags.each do |tag|
 span.label.label-default
 = tag.name
```

```
 - if post.image_src.present?
 img src="#{post.image_src}" alt="Post image" width="100%"
 p
 = post.annotation
 = link_to 'Learn more &#8594;'.html_safe, post, class: 'btn btn-default'
 hr
```

Лістинг файлу app/views/posts/edit.html.slim

```
h1
 | Edit Post
= render 'form'
```

Лістинг файлу app/views/posts/manage.html.slim

```
h1
 | Posts  
 = link_to 'New Post', new_post_path, class: 'btn btn-sm btn-default'
- unless @posts.empty?
 table.table
 thead
 tr
 th Title
 th Annotation
 th Section
 th
 tbody
 - @posts.each do |post|
 tr
 td.min-200
 = link_to post.title, post
 td
 = post.annotation
 td.min-100
 = post.section.name
 td.min-130
 .text-left
 - if (current_user.id == post.user_id || current_user.admin?)
 = link_to 'Edit', edit_post_path(post), class: 'btn btn-sm btn-warning'
```

```
 |  
 = link_to 'Destroy', post, method: :delete, data: { confirm: 'Are you sure?' },
class: 'btn btn-sm btn-danger'
- else
 p
 | No posts yet.
```

Лістинг файлу app/views/posts/new.html.slim

```
h1
 | New Post
= render 'form'
```

Лістинг файлу app/views/posts/show.html.slim

```
h2
 = @post.section.name
 |  
 | &#8594;
 |  
 = @post.title
 |  
 - if (current_user.id == @post.user_id)
 = link_to 'Edit', edit_post_path(@post), class: 'btn btn-sm btn-default'

- if @post.image_src.present?
 = image_tag @post.image_src, alt: 'Post image', width: '100%'

p
 | By 
 strong
 = @post.author

p
 strong
 | Created at:  
 = str_date @post.created_at

h4
```

```
 - @post.tags.each do |tag|
 span.label.label-default
 = tag.name


p
 = @post.text.html_safe


.like-form-section
 - unless current_user.likes?(@post)
 = form_for @post, url: like_post_path, remote: true, method: :post, html: {
class: 'form-inline like-form' } do |f|
 = f.button class: 'btn btn-default' do
 span.glyphicon.glyphicon-thumbs-up
 |  Like
 |  
 - else
 = form_for @post, url: unlike_post_path, remote: true, method: :delete, html:
{class: 'form-inline like-form'} do |f|
 button.btn.btn-success type='submit'
 span.glyphicon.glyphicon-thumbs-up
 |  Like
 |  
 - unless current_user.dislikes?(@post)
 = form_for @post, url: dislike_post_path, remote: true, method: :post, html:
{class: 'form-inline like-form'} do |f|
 button.btn.btn-default type='submit'
 span.glyphicon.glyphicon-thumbs-down
 |  Dislike
 |  
 - else
 = form_for @post, url: undislike_post_path, remote: true, method: :delete,
html: {class: 'form-inline like-form'} do |f|
 button.btn.btn-danger type='submit'
 span.glyphicon.glyphicon-thumbs-down
 |  Dislike
```

```
 - if current_user.recommended_posts.size > 0 &&
!(current_user.recommended_posts.include?(@post) &&
current_user.recommended_posts.size == 1)
 .also-like
 .panel.panel-default
 .panel-body
 h3 You may also like:
 - current_user.recommended_posts(4,0).each do |post|
 - if @post != post
 .media
 - if @post.image_src
 .media-left
 = link_to post do
 img.media-object width="100" src="#{post.image_src}" alt='Post image'
 .media-body
 = link_to post do
 h4.media-heading #{post.title}
 p #{post.annotation}
```

Лістинг файлу app/views/sections/index.html.slim

```
.sections
 .main-section
 h3.title
 | Technical sciences
 = image_tag 'computer_sciences.png'
 .sub-sections.row
 .section.col-md-4
 = image_tag 'social_media.jpg'
 .section.col-md-4
 = image_tag 'economics.jpg'
 .section.col-md-4
 = image_tag 'legal_sciences.jpg'

table.table.table-hover
 thead
 tr
 th #
```

```
 th Name
 - if current_user.admin?
 th Options
 tbody
 - if current_user.admin?
 tr.info
 td
 td
 = simple_form_for @section, method: :post, html: { class: 'form-inline' } do
|f|
 .form-group
 = f.input :name, required: true, placeholder: 'Section name...', label: false
 = f.button :submit, 'Add Section', class: 'btn-sm'
 - @sections.each do |section|
 tr
 td #{section.id}
 td class='hidden js-edit-form' data-id="#{section.id}"
 = simple_form_for section, method: :put, html: { class: 'form-inline' } do |f|
 .form-group
 = f.input :name, placeholder: 'Section name...', required: true, label: false
 = f.button :submit, 'Update', class: 'btn btn-sm btn-primary'
 .form-group
 button class='btn btn-default js-cancel btn-sm' data-id="#{section.id}" Cancel
 td class='js-info-name' data-id="#{section.id}" #{section.name}
 - if current_user.admin?
 td.text-left
 button class='js-edit btn btn-sm btn-warning' data-id="#{section.id}"
 | Edit
 |  
 = link_to 'Destroy', section, method: :delete, data: { confirm: 'Are you sure?'
}, class: 'btn btn-sm btn-danger'
```

Лістинг файлу app/views/devise/sessions/new.html.slim

```
.row
 .col-md-6.col-md-offset-3
 h2
 | Sign in
```

```
 = simple_form_for(resource, as: resource_name, url: new_user_session_path) do
|f|
 = f.input :email, required: false, autofocus: true
 = f.input :password, required: false
 = f.input :remember_me, as: :boolean if devise_mapping.rememberable?
 .text-center
 = f.button :submit, 'Sign in', class: 'btn-success btn-lg'
 .text-center
 = render 'devise/shared/links'
```

Лістинг файлу app/views/devise/registrations/new.html.slim

```
.row
 .col-md-6.col-md-offset-3
 h2
 | Sign up
 = simple_form_for(resource, as: resource_name, url:
registration_path(resource_name)) do |f|
 = f.input :email, required: true, autofocus: true
 = f.input :password, required: true
 = f.input :password_confirmation, required: true
 .text-center
 = f.button :submit, 'Sign up', class: 'btn-success btn-lg'
 .text-center
 = render 'devise/shared/links'
```