БАКАЛАВРСЬКА РОБОТА

БР. ІП - 31.00.00.000 ПЗ

Група ІП-23-1К

Кувік Юрій

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Кувік Юрій Васильович

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Реалізація інтелектуальної системи підбору стеку читача

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач освітнього рівня Кувік Ю.В.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Піх Володимир Ярославович, к.т.н., доцент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Кувіку Юрію Васильовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) “ Реалізація інтелектуальної системи підбору стеку читача ”

керівник проекту (роботи) Піх В.Я., к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

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

1. Аналіз проблематики побудови інтелектуальних систем підбору книг
2. Дослідження алгоритмів рекомендації та парсерів контенту побудови системи підбору
3. Опис алгоритму ANNOY (Approximate Nearest Neighbors Oh Yeah)
4. Програмна реалізація інтелектуальної системи рекомендації та підбору стеку читача
5. Результати використання інтелектуальної системи рекомендацій та підбору стеку читача

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Макет файлу Titelbank (рис. 1.1)
2. ONIX файл (рис. 1.2)
3. Підключення до FTP бібліотеки (рис. 1.3)
4. Приклад реалізованого скрепера даних бібліотеки (рис. 1.4)
5. Дерево ключових слів з використанням структури tag_map_interests (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз проблематики побудови інтелектуальних систем підбору книг	02.05.2025	виконано
2	Дослідження алгоритмів рекомендації та парсерів контенту побудови системи підбору	10.05.2025	виконано
3	Опис алгоритму ANNOY (Approximate Nearest Neighbors Oh Yeah)	20.05.2025	виконано
4	Програмна реалізація інтелектуальної системи рекомендації та підбору стеку читача	28.05.2025	виконано
5	Результати використання інтелектуальної системи рекомендацій та підбору стеку читача	02.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	10.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 78 сторінок, 24 рисунки, список використаних джерел із 34 найменуваннями, 1 додаток.

Мета роботи - розробити та реалізувати інтелектуальну систему рекомендації та підбору книжок для читача з урахуванням семантичної релевантності, читабельності текстів та особистісних переваг користувача.

Об'єкт дослідження - процеси рекомендації та персоналізованого підбору книжок у цифровому середовищі.

Предмет дослідження - алгоритми, моделі та технології, що забезпечують інтелектуальний підбір книжок для читача.

В першому розділі проведено аналіз технічних та методологічних аспектів, що лежать в основі побудови системи рекомендацій, включаючи джерела даних, семантичний аналіз і метрики читабельності

В другому розділі визначено ключові вимоги до архітектури системи та підхід до формування бази знань

В третьому розділі реалізовано та протестовано ефективність різних алгоритмів рекомендацій, включно з високовимірними моделями та парсерами PDF

Висновок: реалізовано прототип системи, що включає ансамблеві підходи до побудови рекомендацій та вирішення проблеми високої розмірності простору ознак. Отримані результати демонструють ефективність запропонованого підходу в задачах автоматизованого підбору літератури

КЛЮЧОВІ СЛОВА: ІНТЕЛЕКТУАЛЬНА СИСТЕМА, РЕКОМЕНДАЦІЇ, СТЕК ЧИТАЧА, ЧИТАБЕЛЬНІСТЬ, ТЕМАТИЧНА РЕЛЕВАНТНІСТЬ, МЕТАДАНІ, ФІЛЬТРАЦІЯ, ANNOY, КЛАСИФІКАЦІЯ ТЕКСТІВ, СКЛАДНІСТЬ ТЕКСТУ

ANNOTATION

The bachelor's thesis consists of 78 pages, 24 figures, a reference list with 34 entries, and 1 appendix.

The aim of the work is to develop and implement an intelligent book recommendation and selection system for readers, taking into account semantic relevance, text readability, and user personal preferences.

The object of study is the processes of recommendation and personalized book selection in a digital environment.

The subject of study is the algorithms, models, and technologies that enable intelligent book selection for readers.

In the first section, an analysis of the technical and methodological aspects underlying the development of the recommendation system is conducted, including data sources, semantic analysis, and readability metrics.

In the second section, key requirements for the system architecture and the approach to forming the knowledge base are defined.

In the third section, the effectiveness of various recommendation algorithms, including high-dimensional models and PDF parsers, is implemented and tested.

Conclusion: A prototype of the system was implemented, incorporating ensemble approaches to recommendation building and addressing the issue of high-dimensional feature space. The obtained results demonstrate the effectiveness of the proposed approach in tasks of automated literature selection.

KEYWORDS: INTELLIGENT SYSTEM, RECOMMENDATION, READER STACK, READABILITY, THEMATIC RELEVANCE, METADATA, FILTERING, ANNOY, TEXT CLASSIFICATION, TEXT COMPLEXITY

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ANN (Approximate Nearest Neighbors) – Наближені найближчі сусіди

ANNOY (Approximate Nearest Neighbors Oh Yeah) – Бібліотека для пошуку наближених найближчих сусідів

AVI (Analytisch Vasthoudende Instructie) – Аналітично стійкі інструкції

CBF (Content-Based Filtering) – Контентна фільтрація

CF (Collaborative Filtering) – Колаборативна фільтрація

ISBN (International Standard Book Number) – Міжнародний стандартний номер книги

LSH (Locality Sensitive Hashing) – Хешування, чутливе до локальності

MAE (Mean Absolute Error) – Середня абсолютна похибка

NLP (Natural Language Processing) – Обробка природної мови

RMSE (Root Mean Squared Error) – Середньоквадратична похибка

RS (Recommender System) – Рекомендаційна система (або RecSys)

ВСТУП

У сучасному інформаційному суспільстві стрімке зростання кількості цифрових текстових ресурсів створює потребу в ефективних інструментах персоналізованого підбору літератури. Особливо це актуально в освітньому контексті, де важливо не лише задовольнити інтереси читача, а й підібрати тексти відповідно до його когнітивних можливостей, рівня читабельності та тематичних уподобань. Існуючі системи рекомендацій здебільшого орієнтовані на загальні алгоритми без врахування складності тексту та інтелектуальних характеристик користувача. Це знижує ефективність таких систем у контексті освітніх потреб та розвитку читацької компетентності. Тому розробка інтелектуальної системи, що враховує метадані текстів, структуру інтересів, семантичну релевантність і індекси складності – є актуальним науково-практичним завданням.

Персоналізовані рекомендаційні системи стають невід'ємною складовою багатьох галузей, зокрема освіти, електронного книговидання та цифрових бібліотек. У той час як класичні підходи до рекомендацій базуються переважно на історії переглядів або вподобаннях користувачів, у сфері підбору текстів для читання виникає потреба в глибшому аналізі текстового контенту. Такий аналіз повинен враховувати не лише тематику, але й рівень читабельності, семантичну складність та відповідність когнітивним характеристикам читача.

Актуальність роботи

Актуальність дослідження зумовлена необхідністю поліпшення процесу навігації користувача в інформаційному середовищі через автоматизовані рекомендаційні системи. Зважаючи на величезні обсяги цифрових бібліотек і електронних ресурсів, користувачі часто не здатні самостійно відібрати релевантні матеріали для читання. Існуючі системи рекомендацій не завжди враховують когнітивні особливості читача,

складність текстів або семантичну релевантність. Робота спрямована на розв'язання цих проблем через впровадження унікальних рішень, таких як використання індексів читабельності, аналіз тематичних інтересів та алгоритмів рекомендацій у високих вимірах.

Ця робота спрямована на створення інтелектуальної системи, здатної здійснювати підбір стеку читача на основі багатофакторного аналізу. Основними аспектами є збір метаданих з різних джерел, побудова ієрархії інтересів користувача, аналіз тематики за допомогою словників ключових слів, застосування метрик читабельності (AVI, CLIB) та використання сучасних алгоритмів машинного навчання й рекомендацій, зокрема Content-Based та Collaborative Filtering.

Мета роботи - розробити та реалізувати інтелектуальну систему рекомендації та підбору книжок для читача з урахуванням семантичної релевантності, читабельності текстів та особистісних переваг користувача.

Завдання дослідження

1. Провести аналіз методів збору та обробки текстових даних.
2. Вивчити сучасні алгоритми рекомендацій та оцінити їх ефективність.
3. Реалізувати механізм тематичної релевантності на основі ключових слів.
4. Інтегрувати індекси читабельності у систему підбору книг.
5. Розробити базу даних книжкового контенту та реалізувати програмну архітектуру системи.
6. Провести тестування системи та оцінити результати її роботи.

Об'єкт дослідження - процеси рекомендації та персоналізованого підбору книжок у цифровому середовищі.

Предмет дослідження - алгоритми, моделі та технології, що забезпечують інтелектуальний підбір книжок для читача.

Методи дослідження

В роботі використано контентний аналіз; спільна фільтрація; гібридні рекомендаційні алгоритми; метод ANNOY; методи обробки природної мови (NLP); індекси читабельності AVI та CLIB; алгоритми зниження розмірності.

Наукова новизна

У роботі запропоновано комплексне поєднання метрик тематичної релевантності, індексів читабельності та сучасних алгоритмів рекомендацій для формування персоналізованого стеку читача в інтелектуальній системі.

Практичне застосування

Розроблена система може бути впроваджена в електронних бібліотеках, освітніх платформах або онлайн-магазинах для поліпшення користувацького досвіду та підвищення ефективності пошуку книжкових ресурсів.

Дана дипломна робота охоплює також питання парсингу PDF-документів, обробки високовимірних векторних просторів, зменшення впливу домінуючих ознак, а також формування ансамблевої моделі рекомендацій. Запропонована система має потенціал бути впровадженою в освітні платформи та онлайн-бібліотеки для підвищення якості та релевантності читацьких рекомендацій.

Бакалаврська робота містить 78 сторінок, 24 рисунки, 3 розділи список використаних джерел із 34 найменуваннями, 1 додаток.

1. АНАЛІЗ ПРОБЛЕМАТИКИ ПОБУДОВИ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ ПІДБОРУ КНИГ

1.1. Передумови для програмної розробки

Дана бакалаврська робота присвячена розробці та реалізації інтелектуальної системи підбору читацького матеріалу (стеку читача). В процесі роботи над дослідженням були набуті ключові навички програмування, а також поглиблені знання щодо специфіки обробки та аналізу даних у високовимірних просторах, що є критично важливим для створення ефективних рекомендаційних систем.

Рівень читацької компетенції є фундаментальним показником успішності освітнього процесу та інтелектуального розвитку особистості. Однак сучасні дослідження вказують на глобальну тенденцію до зниження рівня читацької активності та розуміння текстів серед учнів та студентів. Ця ситуація може бути зумовлена низкою факторів, серед яких виділяють:

- Недостатня відповідність запропонованого читацького матеріалу індивідуальним здібностям, інтересам та особистісним характеристикам учнів.

- Обмежене розуміння педагогами рівня читацької компетенції та динаміки прогресу учнів, що перешкоджає ефективному моніторингу та корекції навчального процесу.

З метою вирішення зазначених проблем та підвищення задоволення від читання розробляються та впроваджуються інноваційні освітні платформи. Однією з таких розробок є система, що фокусується на оптимізації процесу підбору читацького матеріалу. Ключовою особливістю даної системи є наявність обширної бази даних книг, що містить метадані, такі як автор, рекомендований вік, опис, жанр тощо.

Система дозволяє освітнім закладам здійснювати цифровізацію своїх бібліотечних фондів, створюючи точний інвентар доступних книг. Це забезпечує можливість моніторингу читацької активності учнів, зокрема інформацію про поточні та попередньо прочитані книги. Шляхом реєстрації прогресу читання формуються цифрові читацькі щоденники, що надають педагогам розширені дані для аналізу читацької поведінки та динаміки розвитку кожного учня.

Наступним етапом розвитку подібних систем є створення інтелектуального модуля для персоналізованого підбору книг. Цей аспект становить центральний фокус даної роботи.

Основне питання роботи полягає в тому яким чином може бути розроблена система рекомендації книг для ефективного надання користувачам рекомендацій, що відповідають їхньому рівню читацької компетенції та інтересам?

Для систематизації дослідження та досягнення поставленої мети, основне питання деталізовано у наступних підпитаннях:

1. Які методи збору та обробки інформації про книги є найбільш ефективними?

2. Які ключові алгоритми та підходи використовуються у сучасних рекомендаційних системах?

3. Яким чином зібрана та оброблена інформація може бути інтегрована для реалізації функціональної системи рекомендацій?

Структура даної роботи представлена наступними розділами:

Розділ 1 присвячений аналізу методів збору та підтримки актуальності даних про книги в базі даних системи. В ньому детально розглядаються причини та методи агрегації інформації, а також визначаються найбільш релевантні атрибути для процесу підбору книг.

Розділ 2 надає огляд сучасних технік, що застосовуються у рекомендаційних системах, з ілюстративними прикладами. Зокрема,

проводиться детальний аналіз функціонування алгоритму ANNOY, як одного з ключових компонентів, реалізованих у фінальній версії рекомендаційної системи.

Розділ 3 демонструє процес підготовки зібраної інформації для використання у рекомендаційній системі та описує фінальну реалізацію самої системи. Додатково обговорюються перспективи подальшого розвитку та вдосконалення системи, зокрема з урахуванням потреби в аналізі користувацьких даних для оптимізації рекомендацій.

1.2. Методи збору даних

Перш ніж можна буде розробити систему рекомендації книг, важливо мати дані високої якості та кількості про книги. Основні метадані для книги - це її назва та зображення обкладинки книги, оскільки ці речі повинні відображатися в додатку. Однак, для розробки системи рекомендації книг потрібна значно більша кількість даних, особливо дана система спеціалізується на початкових школах. У початкових школах діти тільки почали читати, що призводить до того, що рівень читання книги є критичною інформацією. Адже книги для дітей групи 8 не повинні рекомендуватися дітям групи 3, оскільки вони тільки навчилися читати і, ймовірно, не зрозуміють багато слів. Для отримання всіх цих даних було використано багато різних методів, з яких найважливішим є Titelbank.

1.2.1. База даних Titelbank

Titelbank - це база даних, яка містить всі назви нідерландських книг, опублікованих з 1970 року. На сайті Titelbank доступна велика кількість даних. Наприклад, можна перевірити, чи існує назва, яку ви бажаєте дати своїй книзі, які автори написали які книги, ISBN книги (Міжнародний стандартний номер книги), який є унікальним ідентифікатором для книг і

зазвичай записується як ISBN-13 через його 13-значний формат, та різні інші змінні.

Titelbank також дозволяє запитати файл, який містить всю інформацію, яку вони зберігають про книги в своїй базі даних. "Освіта в дії" також запитала такий файл як основу для власної бази даних. Запитаний файл містив інформацію про приблизно 300 000 книг, і для більшості цих книг також було включено зображення обкладинки та файл PDF перших кількох сторінок.

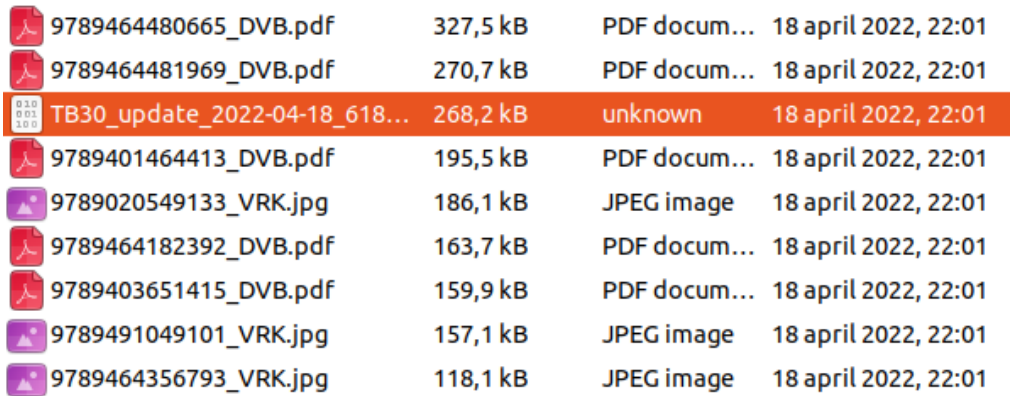

Рисунок 1.1 – Макет файлу Titelbank

Файл, який виділено, містить всю необхідну інформацію про книги у файлі ONIX, який є файлом у стандартизованому форматі XML для метаданих книг. ONIX (Online Information Exchange) — це міжнародний стандарт XML-формату, який використовується у книжковій індустрії для обміну розширеними метаданими про видавничі продукти. Він є ключовим інструментом для стандартизації та автоматизації передачі інформації між різними учасниками видавничого ланцюга постачання, такими як видавці, дистриб'ютори, оптові продавці та бібліотеки.

Основна мета файлу ONIX полягає в забезпеченні єдиного, структурованого та машиночитного способу опису книг, електронних книг, аудіокниг та інших видавничих матеріалів. Це дозволяє ефективно керувати

великими обсягами даних і мінімізувати ручне введення, що зменшує кількість помилок і прискорює процеси.

Розробкою та підтримкою стандарту ONIX займається міжнародна організація EDItEUR, яка постійно оновлює та вдосконалює його відповідно до потреб видавничої галузі. Існує кілька версій ONIX, наприклад, ONIX 2.1 та ONIX 3.0, причому версія 3.0 є більш сучасною та розширеною.

Рисунок 1.2 – ONIX файл

Основні компоненти структури ONIX включають:

1. <Product>: Це кореневий елемент для опису одного видавничого продукту (наприклад, однієї книги). Кожен файл може містити один або кілька таких елементів.

2. Ідентифікація продукту: Містить унікальні ідентифікатори, такі як ISBN (International Standard Book Number), EAN (European Article Number), а також внутрішні ідентифікатори видавця.

3. Опис продукту: Включає широкий спектр метаданих, таких як:

- Заголовок та підзаголовок.

- Автор(и), редактор(и), ілюстратор(и) та інші особи, пов'язані з продуктом.

- Анотація або короткий опис книги.

- Тематичні класифікації: Коди предметних рубрик (наприклад, BISAC, Thema, UDC).

- Характеристики видання: Формат (тверда обкладинка, м'яка обкладинка, електронна книга), кількість сторінок, розміри, мова видання.

- Видавець та імпринт.

- Ціна та валюта.

- Статус доступності (в наявності, передзамовлення, вилучено з друку).

- Пов'язані продукти: Посилання на інші видання тієї ж книги (наприклад, м'яка обкладинка, аудіокнига), або інші книги серії.

- Дати: Дати публікації, випуску та доступності.

- Схеми та версії: Існують різні версії стандарту ONIX (наприклад, ONIX 2.1, ONIX 3.0), які постійно оновлюються для врахування нових вимог індустрії та типів продуктів.

Загалом, ONIX-файл є комплексним контейнером даних, який забезпечує точний, деталізований та уніфікований опис видавничих продуктів, що є критично важливим для ефективної роботи сучасної книжкової індустрії.

1.2.2. Особливості FTP-підключення

Для забезпечення актуальності даних у системі реалізовано механізм щоденного оновлення інформації. Цей процес передбачає взаємодію із зовнішнім сервером протоколу передачі файлів (FTP), який функціонує як репозиторій для публікації регулярних пакетів оновлень. Використання FTP-сервера дозволяє здійснювати ефективний обмін файлами між віддаленими обчислювальними системами у мережевому середовищі. З метою автоматизації процедури отримання та інтеграції оновлень, розроблений

модуль використовує функціонал бібліотеки ftplib мови програмування Python для доступу до зазначеного FTP-ресурсу та завантаження актуальних даних до внутрішньої бази даних системи.

```
import ftplib
ftp = ftplib.FTP()
ftp.connect(host, port)
ftp.login(username, password)
files = ftp.nlst()
for file in files:
    # Зробити щось з файлами
ftp.quit()
```

Рисунок 1.3 – Підключення до FTP бібліотеки

1.2.3. Методологія збору додаткової інформації доповнення бази метаданих

Незважаючи на агрегацію первинних бібліографічних метаданих, виявлено суттєву неповноту критично важливої інформації, необхідної для функціонування інтелектуальної системи підбору читацького матеріалу. Зокрема, для ефективної персоналізації рекомендацій, особливо в освітньому контексті, неприпустимим є пропонування літератури, що не відповідає віковим та когнітивним особливостям реципієнтів. Наприклад, книги, призначені для досвідчених читачів або ті, що містять контент, неприйнятний для молодшої вікової категорії (наприклад, матеріали з елементами жаху або насильства), повинні бути виключені з рекомендаційного пулу.

З огляду на це, ключове значення для розробки адекватної рекомендаційної моделі набуває наявність таких змінних, як індекс читабельності (наприклад, AVI), що відображає рівень складності тексту, та віковий ценз. Ці параметри є фундаментальними для коректного співвіднесення складності книги з рівнем читацької компетенції користувача та його віковими обмеженнями. Додатковою важливою змінною є тематичні ключові слова, які дозволяють ідентифікувати змістову спрямованість книги. Наявність таких дескрипторів є критичною для реалізації механізмів

контентно-орієнтованих рекомендацій, де перевага надається книгам зі схожою тематикою (наприклад, рекомендація книг про автомобілі користувачу, який виявив інтерес до цієї теми). Методи асоціації ключових слів з книгами детально розглянуті зголом.

Не менш значущим аспектом є класифікація типу книги. У контексті навчальних закладів бібліотечний фонд включає різноманітні формати: від книг для спільного читання у класі та ілюстрованих видань з мінімальним текстовим наповненням до науково-популярної літератури без наративного сюжету. Адекватний облік цих типологічних відмінностей є обов'язковим для формування релевантних рекомендацій, що відповідають цілям та формату використання.

```
import requests
from bs4 import BeautifulSoup
try:
    page = requests.get(url, timeout=10)
except requests.exceptions.ReadTimeout:
    print("Timeout reached")
soup = BeautifulSoup(page.content, 'lxml')
```

Рисунок 1.4 – Приклад реалізованого скрепера даних бібліотеки

Для подолання проблеми неповноти даних та збагачення бази метаданих було застосовано метод веб-скрейпінгу. Цей підхід дозволив програмно збирати додаткові змінні з відкритих веб-ресурсів. Реалізація скрейперів здійснювалася з використанням мови програмування Python та відповідних бібліотек, зокрема requests для виконання HTTP-запитів до веб-сторінок та BeautifulSoup для парсингу HTML-контенту. Отримані дані з веб-ресурсів інтегрувалися з наявними бібліографічними відомостями, формуючи єдину, вичерпну базу даних, достатню для ефективного функціонування інтелектуальної системи рекомендацій.

Потім необхідна інформація на сторінці може бути знайдена за допомогою методу find_all(), який приймає як аргумент назву тегу, який ви

бажаєте знайти. Дані з Titelbank у поєднанні з даними зі скрейперів призводять до бази даних, яка містить всю необхідну інформацію для інтелектуальної системи рекомендацій.

1.3. Механізм асоціації ключових слів

Для ефективної реалізації системи рекомендацій, здатної ідентифікувати та групувати літературні твори за тематичною схожістю, було розроблено механізм асоціації ключових слів з книгами. Цей підхід дозволяє не тільки рекомендувати книги в межах однієї вузької теми (наприклад, "футбол"), але й враховувати ієрархічні зв'язки між ширшими категоріями (наприклад, "спорт" як узагальнення для "футболу", що є більш релевантним, ніж "мистецтво"). Для досягнення цієї мети було сконструйовано дві взаємодоповнюючі структури даних.

1.3.1. Словник тригерних термінів (tag_word_dict)

Перша структура даних, tag_word_dict, являє собою асоціативний масив, що зіставляє базові тематичні теги з набором тригерних термінів та їх індивідуальними коефіцієнтами релевантності. Коефіцієнти релевантності були емпірично визначені та налаштовані шляхом ітеративного тестування після початкової реалізації для оптимізації точності асоціацій. Важливим аспектом проектування цієї структури є використання не лише окремих слів, але й фраз, а також врахування контекстуальних маркерів (наприклад, пробілів перед термінами). Це дозволяє запобігти помилковим активаціям, коли частина слова може збігатися з тригерним терміном, але не відповідає бажаній тематиці. Наприклад, для активації тегу "штучний інтелект" використовуються повні терміни "штучний інтелект" або "ШІ", а не окремі слова "штучний" чи "інтелект", що підвищує специфічність та точність зв'язування.

1.3.2. Ієрархія тематичних інтересів (tag_map_interests)

Друга структура, tag_map_interests, реалізована як ієрархічне дерево залежностей, що моделює взаємозв'язки між тематичними тегами. Це дозволяє одному специфічному тегу активувати більш загальні теги вищого рівня.

```
TAG_MAP_INTERESTS = {
    'хобі': {
        'спорт': {
            'командний спорт': {
                'футбол': 1.5,
                'хокей': 1.5,
                'баскетбол': 1.5
            },
            'зимовий спорт': {
                'лижі': 1.5
            },
            'індивідуальний спорт': {
                'дартс': 1.1,
                'шахи': 1.1,
                'гольф': 1.3
            }
        }
    }
}
```

Рисунок 1.5 - Дерево ключових слів з використанням структури tag_map_interests

Наприклад, активація тегу "лижі" може поширюватися на теги "зимовий спорт" та "спорт", які, у свою чергу, можуть активувати "хобі". При поширенні активації вгору по ієрархії коефіцієнт релевантності тегу зменшується на кожному кроці (множиться на 0.5), відображаючи зниження специфічності зв'язку від вузької теми до ширшої категорії. Це забезпечує вищу релевантність для безпосередньо активованих, специфічних тегів порівняно з опосередковано активованими ширшими категоріями.

Щоб пов'язати ці ключові слова з книгами, використовується опис книги. Щоб забезпечити, щоб термін "гольф" був правильно активований, якщо він знаходиться поруч з розділовими знаками, деякі розділові знаки, такі як крапка, кома, знак питання та знак оклику, замінюються пробілом.

Для всіх ключових слів у tag_word_dict та для всіх тригерних слів на ключове слово застосовується наступний алгоритм:

- Підрахувати кількість входжень тригерного слова.

- Для кожного входження додати пов'язану релевантність, знайдену в tag_word_dict, до загальної суми релевантності на ключове слово.

- Якщо ця сума більша за 0, зберегти ключове слово та його пов'язану загальну оцінку релевантності в словник.

```
def recursive_func(tree):
    if isinstance(tree, dict):
        relevance_sum = 0
        for tag, tree_sub in tree.items():
            relevance = 0
            if isinstance(tree_sub, float):
                if tag in tag_dict:
                    relevance = tag_dict[tag] * tree_sub
                else:
                    # Перевірити, чи контекст містить підтеги
                    relevance = recursive_func(tree_sub)
                if relevance > 0:
                    tag_dict[tag] = relevance
                    relevance_sum += relevance
                return relevance_sum / 2
    tag_dict = {}
    for tag_obj in tag_list_custom:
        tag_dict[tag_obj['tag']] = tag_obj['relevance']
    recursive_func(config, TAG_MAP_INTERESTS)
```

Рисунок 1.6 – Код для рекурсивної функції

Результатом є словник з ключовими словами як ключами та оцінкою релевантності як значення. Рекурсивна функція викликається на цьому словнику для проходження дерева tag_map_interests, яка відстежує загальну накопичену релевантність на ключове слово. Якщо зустрічається тег, який зустрічається в словнику, його релевантність множиться на релевантність у дереві та також додається до всіх ключових слів у дереві вище, множачи на 0.5 кожного кроку вище в дереві. Це робиться, тому що ключові слова вище в дереві можуть бути активовані сотнями інших ключових слів, що призводить до дуже високої оцінки релевантності для ключових слів вище. Якщо всі релевантності нормалізуються в кінці, релевантності конкретних ключових

слів будуть дуже низькими, і тільки верхні ключові слова матимуть пристойну оцінку релевантності. Це також враховує той факт, що ключові слова, які були спеціально активовані, повинні бути більш релевантними, ніж непрямо активовані слова. Книга про футбол повинна отримати високу релевантність для ключових слів "футбол" і "спорт" і меншу релевантність для "хобі". Код для рекурсивної функції наведений на рисунку 1.6.

1.3.3. Алгоритм визначення тематичної релевантності

Процес асоціації тематичних тегів з книгами базується на аналізі текстового опису книги. Попередня обробка тексту включає заміну певних розділових знаків (наприклад, крапки, коми, знаки питання та оклику) на пробіли, що забезпечує коректну ідентифікацію тригерних термінів.

Алгоритм визначення тематичної релевантності реалізується у кілька етапів:

1. Початкова активація тегів.

Для кожного тригерного терміна, визначеного у структурі tag_word_dict, обчислюється його кількість входжень у тексті опису книги. Загальна сума релевантності для кожного відповідного базового тегу визначається шляхом множення кількості входжень на попередньо встановлений коефіцієнт релевантності цього терміна. Лише теги, сумарна релевантність яких перевищує нуль, зберігаються у проміжному словнику trigger_dict.

2. Ієрархічне поширення релевантності.

На другому етапі застосовується рекурсивний механізм для проходження ієрархічної структури tag_map_interests. Для кожного тегу, що міститься в trigger_dict (або є результатом активації), його релевантність поширюється вгору по дереву залежностей, активуючи батьківські теги. На кожному кроці поширення релевантність поточного тегу множиться на коефіцієнт релевантності, визначений у tag_map_interests (якщо тег є

безпосереднім дочірнім елементом), а потім значення зменшується (множиться на 0.5) для кожного наступного кроку вгору по ієрархії. Такий механізм деградації релевантності запобігає надмірному завищенню значень для загальних тегів вищого рівня, забезпечуючи при цьому вищу релевантність для безпосередньо активованих, специфічних тегів. Це дозволяє зберегти баланс між точністю (специфічні теги) та повнотою (ширші категорії).

3. Нормалізація.

На завершальному етапі всі отримані значення релевантності нормалізуються відносно максимального значення, що забезпечує шкалювання оцінок до діапазону [0,1] та їх порівнянність.

1.4. Показник читабельності AVI

Показник AVI (Analyse van Individualiseringsvormen) є метрикою, що характеризує рівень складності текстових матеріалів. Його застосування поширене в освітньому середовищі, зокрема у початковій школі, для фасилітації вибору літератури, відповідної індивідуальному рівню читацької компетенції учнів. Даний індикатор часто маркується безпосередньо на книжкових виданнях з метою спрощення навігації для читачів.

Визначення показника AVI для книги здійснюється на основі розрахункового значення CILT (Cito Index voor de LeesTechniek), згідно з наступною шкалою відповідності (таблиця 1.1).

Таблиця 1.1 - Шкала відповідності CILT до AVI

Рівень AVI	**Значення CILT**
Початок	<54.9
M3	54.9−56.9
E3	56.9−58.9
M4	58.9−61.9
E4	61.9−63.9

M5	63.9−65.9
E5	65.9−67.9
M6	67.9−69.9
E6	69.9−71.9
M7	71.9−73.9
E7	73.9−74.9
Плюс	>74.9

Значення CILT, що слугує як Індекс техніки читання, розраховується за емпіричною формулою:

$$CILT=150-(114.49+0.28\times Wfreq-12.33\times Lavg)$$

де:

Wfreq — відсоток слів у тексті, що входять до апріорно визначеного списку 1000 найбільш вживаних слів.

Lavg — середня довжина слів у тексті.

З огляду на відсутність публічно доступного еталонного списку високочастотних слів, що використовується для розрахунку CILT, у даному дослідженні був інтегрований альтернативний, загальнодоступний лінгвістичний корпус. Для адаптації цього корпусу до специфіки задачі було проведено попередню обробку, що включала видалення термінів, які містять службові символи (наприклад, _), та стандартизацію символів валют (наприклад, заміна $ на €).

1.5. Оцінка зрозумілості тексту за індексом CLIB

Окрім показника складності тексту, для повної оцінки читабельності використовується метрика CLIB-niveau (Cito LeesIndex Begrip), яка кількісно визначає ступінь зрозумілості текстового матеріалу. Цей індекс відображає легкість сприйняття та розуміння тексту читачем.

Рівень CLIB для конкретного тексту визначається на основі розрахункового значення індексу CLIB відповідно до наступної шкали (таблиця 1.2).

Таблиця 1.2 - Шкала відповідності CLIB

Рівень CLIB	**Значення CLIB**
Початок	≤ -12
3	$-12 < CLIB \leq 7$
4	$7 < CLIB \leq 20$
5	$20 < CLIB \leq 35$
6	$35 < CLIB \leq 48$
7	$48 < CLIB \leq 61$
8	$61 < CLIB \leq 74$
Плюс	$CLIB > 74$

Значення індексу CLIB обчислюється за наступною емпіричною формулою:

$$CLIB = 46 - 6.603 \times Lavg + 0.474 \times Wfreq - 0.365 \times TTR + 1.425 \times Savg$$

де:

Lavg — середня довжина слова, аналогічна параметру, що використовується в розрахунку CILT.

Wfreq — відсоток слів, що входять до списку 1000 найбільш вживаних слів, аналогічно параметру CILT.

TTR (Type-Token Ratio) — співвідношення кількості унікальних слів (типів) до загальної кількості слів (токенів) у тексті, виражене у відсотках. Цей показник характеризує лексичне різноманіття тексту.

Savg — середня кількість слів у реченні, що відображає синтаксичну складність тексту.

2. ДОСЛІДЖЕННЯ АЛГОРИТМІВ РЕКОМЕНДАЦІЇ ТА ПАРСЕРІВ КОНТЕНТУ ПОБУДОВИ СИСТЕМИ ПІДБОРУ СТЕКУ ЧИТАЧА

2.1. Метод вилучення текстових даних з PDF-документів

Для повноцінного функціонування інтелектуальної системи підбору читацького стеку, критично важливою є наявність точних показників читабельності книг, таких як AVI, CILT та CLIB. Проте, незважаючи на агрегацію доступних метаданих, інформація щодо цих показників не завжди є вичерпною, особливо для видань, орієнтованих на широкий віковий діапазон. Для всебічної оцінки складності та зрозумілості тексту, що виражається у значеннях CILT та CLIB, необхідно аналізувати повний текстовий зміст книги, а саме: відсоток вживання високочастотних слів, середню довжину слів, кількість унікальних та загальну кількість слів, а також середню кількість слів у реченні.

Для розв'язання проблеми неповноти даних та забезпечення необхідних вхідних параметрів для розрахунку показників CILT і CLIB, було використано ресурс, що містить цифрові прев'ю книжкових видань у форматі PDF (наприклад, перші декілька сторінок). Ці PDF-файли, попри їх фрагментарність, можуть містити достатній обсяг текстової інформації для проведення лінгвістичного аналізу.

Для програмного вилучення текстового контенту з таких PDF-документів застосовується спеціалізована бібліотека для парсингу PDF, така як Pdfminer у середовищі Python. Процес вилучення тексту включає програмне відкриття PDF-документа, ітерацію по його сторінках та подальший аналіз елементів макету кожної сторінки. На етапі обробки макету PDF-файлу ідентифікуються та обробляються різні типи об'єктів, що складають візуальне представлення документа.

```python
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfpage import PDFPage
from pdfminer.pdfparser import PDFParser
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import PDFPageAggregator
from pdfminer.layout import LAParams, LTTextBox, LTTextLine, LTFigure

fp = open('example.pdf', 'rb')
parser = PDFParser(fp)
doc = PDFDocument(parser)

rsrcmgr = PDFResourceManager()
laparams = LAParams()
device = PDFPageAggregator(rsrcmgr, laparams=laparams)
interpreter = PDFPageInterpreter(rsrcmgr, device)
for page in PDFPage.create_pages(doc):
    interpreter.process_page(page)
    layout = device.get_result()
    for obj in layout._objs:
        # Зробити щось з об'єктом
```

Рисунок 2.1 – Код PDF-парсера

До таких об'єктів належать текстові блоки (LTTextBox, LTTextLine), графічні елементи (LTFigure, LTImage), а також лінії та прямокутники (LTLine, LTRect). Ключовим завданням на цьому етапі є ідентифікація та вилучення саме текстових об'єктів для подальшого лінгвістичного аналізу. Це дозволяє отримати фрагменти тексту, необхідні для обчислення лінгвістичних параметрів, що входять до формул CILT та CLIB.

Оскільки є діти, які люблять читати книги з великою кількістю зображень, може бути корисно розрахувати співвідношення зображень на сторінку для книг, щоб рекомендувати книги з великою кількістю зображень цим дітям. Наступний код був реалізований для досягнення бажаного результату:

```python
if isinstance(obj, LTImage):
    images_count += 1
```

Щоб визначити кількість унікальних та загальних слів, потрібно підрахувати слова в PDF. Як можна побачити на рисунку 2.1, слова

зустрічаються в об'єктах LTTextBox. Кожен об'єкт LTTextBox містить кілька об'єктів LTTextLine, і кожен об'єкт LTTextLine представляє рядок слів у PDF.

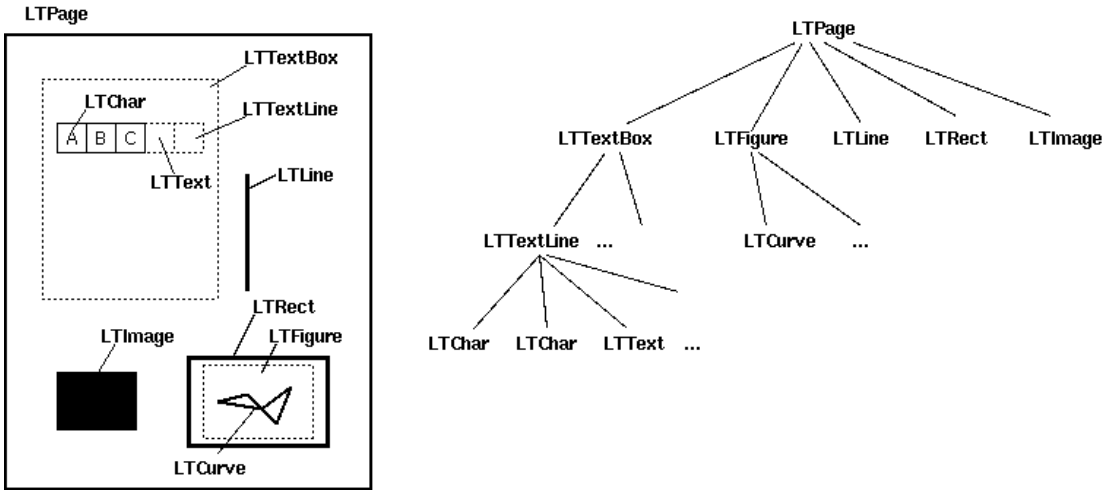

Рисунок 2.2 - Розташування об'єктів у PDF-файлі

Наведений нижче код виконує призначену мету:

```
if isinstance(obj, LTTextBox):
    for ob in obj:
        if isinstance(ob, LTTextLine):
            words = ob.get_text().lower().strip().split(' ')
            word_count += len(words)
```

Щоб отримати відсоток частих слів, використовується інший цикл для ітерації по всіх словах у рядку. Потім для кожного слова перевіряється, чи зустрічається воно в списку найчастіше вживаних слів, і відповідно оновлюється лічильник, як показано нижче:

```
words = ob.get_text().lower().strip().split(' ')
for word in words:
    if word in frequent_words:
        frequent_word_count += 1
```

Хоча це могло б здатися правильним, на практиці це не так. Це пов'язано з тим, що парсери завжди є складними і потребують додаткового коду для крайніх випадків. У цьому випадку код не враховує той факт, що в реченні є багато розділових знаків. Якщо слово 'дім' є одним з найчастіше вживаних слів, і в PDF є речення: "Ар'єн йде до дому.", функція split витягне "дім." як слово і перевіряє, чи зустрічається воно в списку частих слів, чого не відбувається! Щоб уникнути цього, всі розділові знаки повинні бути усунені, що досягається наступним кодом:

```
to_be_replaced = {',', '', '-', '', ',', ',', ',', ',', ',', ',', ',', ',', ',', ',', ',', ',', ',', ',', ',',
```

Кількість речень - це ще одна змінна, яку необхідно отримати для розрахунку показників CILT та CLIB. Це можна легко зробити, збільшуючи кількість речень на одиницю кожного разу, коли зустрічається один з наступних символів: '.', '!' або '?'. Однак це не враховує той факт, що існує багато скорочень, які містять символ '.'. Оскільки список усіх скорочень не був знайдений, код враховує лише ті скорочення, які зустрічаються в списку найбільш вживаних.

Остання проблема, яку необхідно вирішити, - це перенос слів. Розглянемо наступне речення:

```
Джордж зробив дивовижне відкриття і радіє розповісти новини своєму університетському відділу.
```

Як можна побачити, це речення охоплює два рядки. Це створює проблеми для наступного рядка в коді: "object.get_text().lower().strip().split(' ')". Коли цей рядок виконується і розділові знаки видаляються, останнє слово рядка буде "уні", а перше слово наступного рядка буде "верситет". Це не тільки призведе до неправильного підрахунку найчастіше вживаних слів, але

й до плутанини в загальному підрахунку слів, а отже, і до інших змінних, які залежать від загального підрахунку слів.

Простий підхід для вирішення цієї ситуації - спочатку об'єднати всі рядки зі всіх сторінок. Потім, щоб отримати кількість речень, використовується пакет NLTK. Цей пакет містить функцію "sent_tokenize()", яка приймає рядок як вхід і повертає список рядків як вихід, застосовуючи різні евристики та правила для визначення меж між реченнями. Оскільки кожен рядок у списку тепер відповідає реченню, кількість речень можна легко розрахувати.

Ще одна перевага цього підходу полягає в тому, що слово з переносом в кінці рядка тепер сприймається як одне слово з дефісом, що означає, що одне з речень після функції токенізації містить рядок "уні-верситет". Якщо функція ".lower().strip().split(' ')" застосовується після видалення розділових знаків, включаючи дефіси, для кожного речення, результат містить список, який містить лише слова без розділових знаків, включаючи дефіси, і слово 'університет' правильно знаходиться:

["Джордж", "зробив", "дивовижне", "відкриття", "і", "радіє", "розповісти", "новини", "своєму", "університетському", "відділу"]

Загальна кількість слів, а також відсоток частих слів тепер легко доступні, як показано в коді нижче на рисунку 2.3.

Цей фрагмент коду виконує комплекс операцій з обробки тексту, вилученого з PDF-файлу, з метою обчислення лінгвістичних метрик, необхідних для оцінки читабельності тексту (зокрема, показників CILT та CLIB, які обговорювалися раніше).

Цей код представляє собою реалізацію модуля для автоматичного лінгвістичного аналізу текстових даних, витягнутих з PDF-файлів. Його основна функція полягає у підготовці тексту (токенізація, очистка) та обчисленні ключових показників (відсоток частих слів, середня довжина слів, лексичне різноманіття, середня довжина речення), які є необхідними

вхідними даними для розрахунку індексів читабельності тексту, таких як CILT та CLIB.

```
import nltk

# Завантажити модель речень
nltk.download('punkt')
nltk.download('dutch')
pdf_text = ""

..... # Деякий код перед тим, щоб ітерувати по всіх об'єктах на сторінці
if isinstance(obj, LTTextBox):
    for line in obj:
        if isinstance(line, LTTextLine):
            pdf_text += line.get_text()
..... # Деякий інший код для розміру шрифту

sentences = nltk.sent_tokenize(pdf_text, language='dutch')
sentence_count = len(sentences)
words = pdf_text.lower().strip().split(' ')
for word in words:
    if word not in abbreviations:
        for key, value in to_be_replaced.items():
            word = word.replace(key, value)
word_count = len(words)
total_length = 0
unique_words = set()
for word in words:
    if word in frequent_word_list:
        frequent_word_count += 1
    total_length += len(word)
    unique_words.add(word)
percentage_frequent_words = (frequent_word_count/word_count)*100
percentage_unique_words = (len(unique_words)/word_count)*100
average_word_length = total_length/word_count
words_per_sentence = word_count/sentence_count
```

Рисунок 2.3 - Обчислення лінгвістичних метрик, необхідних для оцінки читабельності тексту

Наведемо детальний опис функціоналу коду.

1. Імпорт та завантаження ресурсів NLTK:

- import nltk: Імпортує бібліотеку Natural Language Toolkit (NLTK), яка є стандартним інструментом для роботи з текстом у Python.

- nltk.download('punkt'): Завантажує моделі токенізації 'punkt'. Ці моделі використовуються для розбиття тексту на речення.

- nltk.download('dutch'): Завантажує специфічні ресурси, що дозволяє sent_tokenize коректно розбивати речення.

2. Вилучення тексту з PDF:

- pdf_text = "": Ініціалізує порожній рядок, який буде містити весь витягнутий текст.

- Блок коду if isinstance(obj, LTTextBox): ... pdf_text += line.get_text(): Цей фрагмент (який є частиною більшого циклу, що ітерує по об'єктах сторінки PDF) відповідає за безпосереднє вилучення тексту. Він перевіряє, чи є поточний об'єкт PDF (з бібліотеки pdfminer) текстовим блоком (LTTextBox). Якщо так, то код проходить по всіх рядках у цьому блоці, перевіряє, чи є рядок текстовим (LTTextLine), і додає його вміст до змінної pdf_text. Таким чином, pdf_text накопичує весь текст, знайдений у PDF.

3. Токенізація речень:

- sentences = nltk.sent_tokenize(pdf_text, language='dutch'): Використовує функцію sent_tokenize NLTK для розбиття витягнутого pdf_text на окремі речення, застосовуючи правила токенізації.

- sentence_count = len(sentences): Обчислює загальну кількість виявлених речень.

4. Попередня обробка та токенізація слів:

- words = pdf_text.lower().strip().split(' '):

- pdf_text.lower(): Перетворює весь текст на малі літери для уніфікації та забезпечення коректного підрахунку слів без урахування регістру.

- .strip(): Видаляє початкові та кінцеві пробіли.

- .split(' '): Розбиває текст на список окремих слів за пробілом. Це є базовою токенізацією слів.

- Цикл for word in words: if word not in abbreviations: for key, value in to_be_replaced.items(): word = word.replace(key, value): Цей блок виконує

додаткову очистку та нормалізацію слів. Він пропускає слова, які знаходяться у списку відомих скорочень (abbreviations), та замінює певні символи або підрядки (key) на інші (value), згідно з словником to_be_replaced. Це необхідно для стандартизації тексту та запобігання помилкам у подальшому лінгвістичному аналізі (наприклад, для коректного порівняння зі списком частих слів).

- word_count = len(words): Обчислює загальну кількість слів після їх токенізації та початкової очистки.

5. Обчислення лінгвістичних метрик:

- total_length = 0, unique_words = set(): Ініціалізує змінні для накопичення сумарної довжини всіх слів та набору унікальних слів. (Змінна frequent_word_count також передбачається ініціалізованою перед циклом).

- Наступний цикл for word in words: ...:

- if word in frequent_word_list: frequent_word_count += 1: Перевіряє, чи входить поточне слово до попередньо визначеного списку частих слів (frequent_word_list), і збільшує лічильник, якщо воно там присутнє.

- total_length += len(word): Додає довжину поточного слова до загальної сумарної довжини.

- unique_words.add(word): Додає слово до набору unique_words. Оскільки набори зберігають лише унікальні елементи, це ефективно збирає всі унікальні слова в тексті.

- percentage_frequent_words = (frequent_word_count/word_count)-100: Розраховує відсоток частих слів у тексті. Ця метрика відповідає Wfreq у формулах CILT та CLIB.

- percentage_unique_words = (len(unique_words)/word_count)-100: Розраховує відсоток унікальних слів від загальної кількості слів (Type-Token Ratio, TTR). Ця метрика є ключовою для формули CLIB.

- average_word_length = total_length/word_count: Обчислює середню довжину слова в тексті (Lavg), яка використовується як у CILT, так і в CLIB.

- words_per_sentence = word_count/sentence_count: Розраховує середню кількість слів у реченні (Savg), що є однією зі змінних для CLIB.

2.2. Метод контентної фільтрації (Content-Based Filtering)

Контентна фільтрація (Content-Based Filtering) є методом рекомендації, що базується на аналізі внутрішніх атрибутів об'єктів (товарів, контенту) та профілю вподобань користувача. Основний принцип полягає в тому, що якщо користувач виявив інтерес до певного елемента (наприклад, прочитав книгу), то йому, ймовірно, сподобаються інші елементи, що мають схожі характеристики. Алгоритм контентної фільтрації будує профіль користувача на основі атрибутів контенту, який він споживав або оцінив позитивно, а потім рекомендує нові елементи, що демонструють високий ступінь подібності до цього профілю.

Таблиця 2.1 - Приклад атрибутів книг для контентної фільтрації

	Книга A	**Книга B**	**Книга C**	**Книга D**	**Книга E**
Наукова фантастика	✓			✓	
Фентезі	✓				
Екшен	✓		✓	✓	
Жахи			✓		✓
Трилер		✓			
Романтика		✓			
Автор A	✓		✓		
Автор B		✓			
Автор C				✓	✓

Подібність між елементами визначається на основі їхніх метаданих, таких як жанр, автор, ключові слова, тематика, рівень складності тощо. Наприклад, як показано у таблиці 2.1, для книги можуть бути визначені такі атрибути.

У випадку, якщо користувач виявив інтерес до "Книги А" (жанри: наукова фантастика, фентезі, екшен; автор: Автор А), система контентної фільтрації проаналізує характеристики інших книг. Наприклад, "Книга D" (жанри: наукова фантастика, фентезі, екшен; автор: Автор С) демонструє високий ступінь змістової подібності за жанрами, тоді як "Книга С" (жанри: наукова фантастика, екшен, жахи; автор: Автор В) має перетин за жанрами. На основі цих атрибутивних збігів, алгоритм визначить "Книгу С" та "Книгу D" як потенційно релевантні рекомендації для даного користувача, тоді як "Книги В" та "Е", що не мають значних перетинів за атрибутами, будуть вважатися менш релевантними.

2.3. Спільна фільтрація (Collaborative Filtering)

Метод спільної фільтрації (Collaborative Filtering) є однією з фундаментальних парадигм у побудові рекомендаційних систем. На відміну від контентної фільтрації, що базується на аналізі атрибутів самого контенту, спільна фільтрація генерує рекомендації шляхом виявлення закономірностей у поведінці та вподобаннях великої кількості користувачів.

Спільна фільтрація, часто називається фільтрацією співпраці користувач-користувач, є технікою фільтрації, заснованою на припущенні, що користувачі зі схожими смаками, ймовірно, сподобаються продукти, які споживаються іншими. Якщо користувач закінчив книгу і розмовляє з кимось, хто також прочитав цю книгу, вони можуть запитати один одного про хорошу рекомендацію, оскільки вони мають схожий смак. Хоча цей схожий смак заснований лише на читанні однієї книги, вони все ще можуть

віддавати перевагу абсолютно різним речам. Однак, маючи кількох користувачів зі схожими смаками, це звужується.

Таблиця 2.2 - Приклад спільної фільтрації

	Книга A	**Книга B**	**Книга C**	**Книга D**	**Книга E**
Користувач 1	✓	✓	✓	X	X
Користувач 2	✓	X	✓	X	✓
Користувач 3	X	X	X	✓	✓
Користувач 4	X	X	✓	✓	✓
Користувач 5	✓	?	?	?	?

Спостерігається, що користувачі 1, 2 та 5 всі прочитали книгу A. Якщо ці користувачі вважаються користувачами зі схожими смаками, можна створити наступну таблицю.

Таблиця 2.3 - Приклад спільної фільтрації (із змінами)

	Книга A	**Книга B**	**Книга C**	**Книга D**	**Книга E**
Користувач 1	✓	✓	✓	X	X
Користувач 2	✓	X	✓	X	✓
Кількість		1	2	0	1
Користувач 5	✓	?	?	?	?

✓ прочитано X не прочитано ? ймовірно, хороша пропозиція ? ймовірно, погана пропозиція ? ні хороша, ні погана

Дивлячись на таблицю, можна побачити, що книга C, ймовірно, хороша пропозиція для користувача 5. Також видно, що чим більше даних доступно, тим кращі будуть рекомендації. Припустимо, що не було інформації про користувача 2, тоді лише користувач 1 вважався б схожим, і книги B та C були б рекомендовані, навіть якщо книга B загалом читається рідше, ніж

книга C, а книга E навіть не розглядалася б, хоча вона читається частіше, ніж інші книги.

2.4. Алгоритм ANNOY (Approximate Nearest Neighbors Oh Yeah)

Вищезазначені методи фільтрації обидва вимагають від системи визначити найкращий збіг. Хоча знаходження найкращого збігу в прикладах здавалося легким, знаходження їх у більшому наборі даних з більшою кількістю вимірів і точок даних набагато складніше. Для вирішення цієї проблеми можна використовувати ANNOY.

ANNOY, що означає Approximate Nearest Neighbors Oh Yeah, є алгоритмом, заснованим на випадкових проекціях і деревах. Він був розроблений Еріком Бернхардссоном у 2015 році, який на той час працював у Spotify. Те, що робить цей алгоритм корисним, - це те, що він не намагається обчислити точні найближчі сусіди. Він, як і говорить назва, знаходить приблизно найближчі сусіди, що призводить до значного зменшення складності.

Це зменшення складності виникає з того факту, що з даних будується дерево, в якому пошук найближчого сусіда може бути виконаний за логарифмічний час. Це пояснюється більш детально нижче:

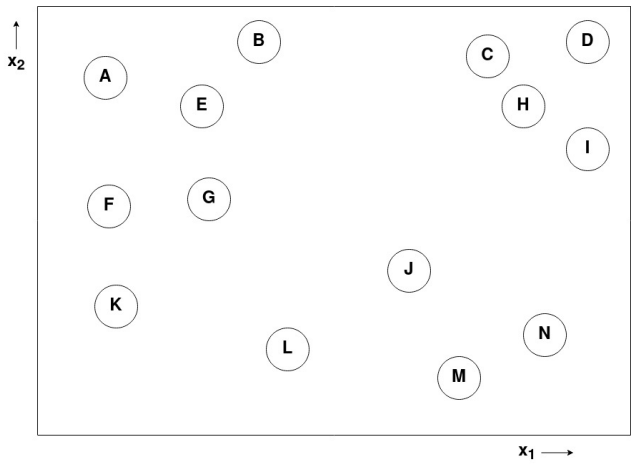

Рисунок 2.4 – Пояснення алгоритму ANNOY

Як можна побачити, є 14 точок даних у двовимірному просторі. Перше, що робиться, - це взяти дві випадкові точки, в цьому випадку A та N, і з'єднати їх лінією:

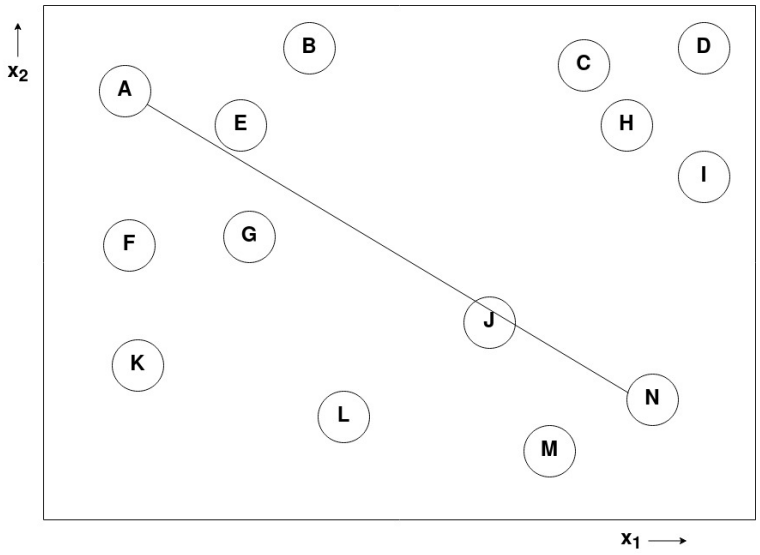

Рисунок 2.5 – Пояснення алгоритму ANNOY (крок 2)

Далі проводиться нова лінія, перпендикулярна до тієї, що від A до N, яка перетинає посередині:

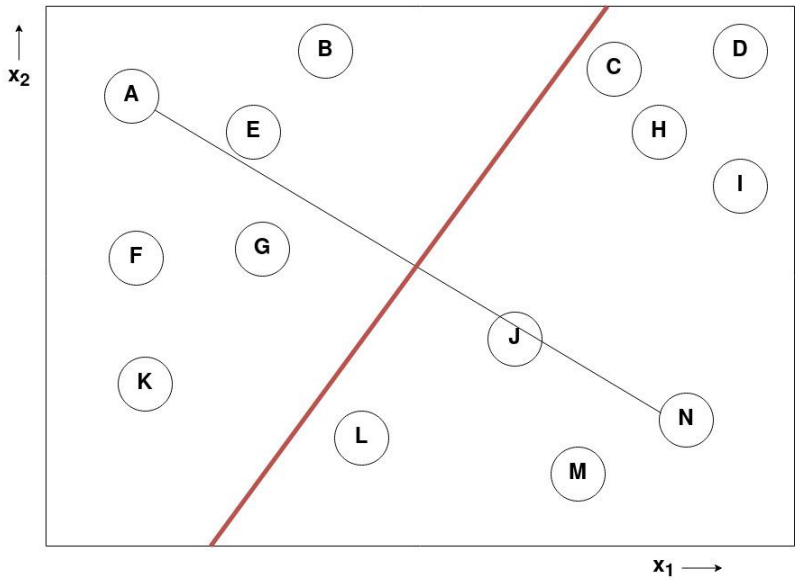

Рисунок 2.6 – Пояснення алгоритму ANNOY (крок 3)

Лінія між точками A та N видаляється, що призводить до двох різних областей. Одночасно генерується структура дерева, де (A,N) означає, що дані були розділені між точками A та N:

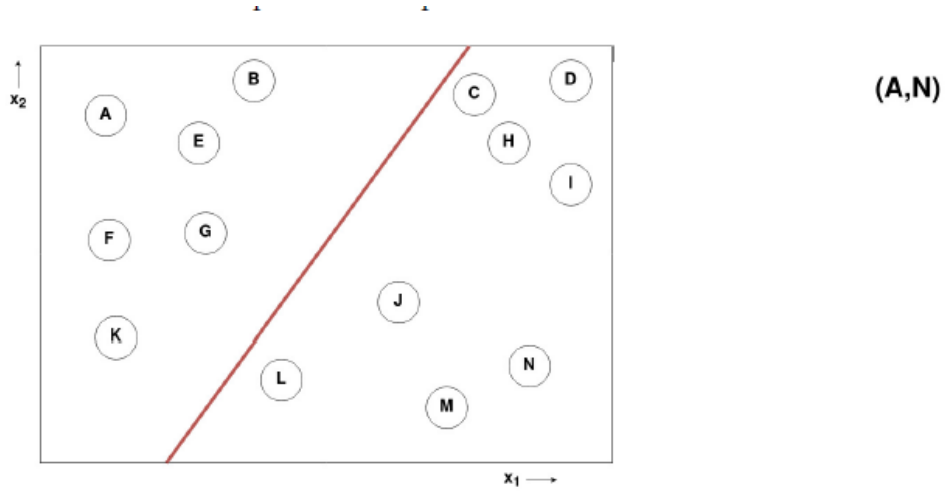

Рисунок 2.7 – Пояснення алгоритму ANNOY (крок 4)

Алгоритм продовжує розділяти кожну область, як це було зроблено вище, і завершується, коли область містить менше, ніж заздалегідь визначену кількість точок, 3 в цьому прикладі.

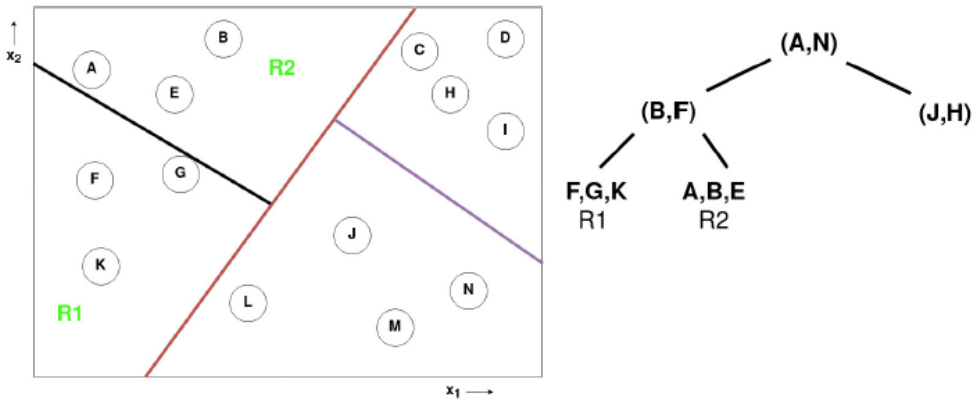

Рисунок 2.8 – Пояснення алгоритму ANNOY (крок 5)

Області R1 та R2 містять 3 або менше точок даних, тому розділення в цих областях завершується.

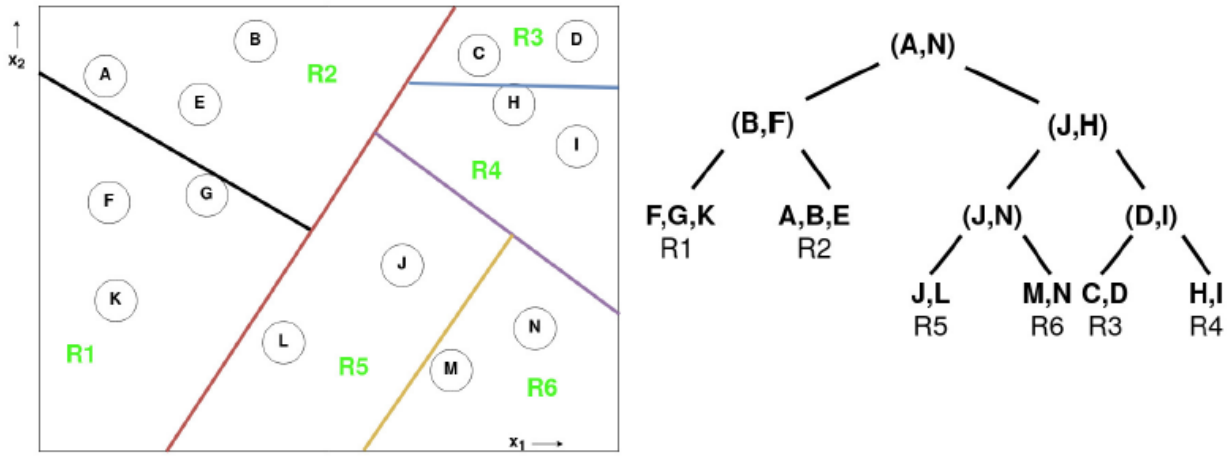

Рисунок 2.9 – Пояснення алгоритму ANNOY (крок 6)

Тепер всі області містять 3 або менше точок даних, і розділення точок даних на області завершується. Щоб визначити найближчого сусіда точки, алгоритм дивиться на область, в якій знаходиться точка, і обчислює відстань лише між точками в цій області. Отже, якщо потрібно визначити найближчого сусіда M, виконується наступне:

- Чи знаходиться M ліворуч чи праворуч від лінії між точками A та N? Праворуч, тому продовжуємо праворуч у дереві.

- Чи знаходиться M ліворуч чи праворуч від лінії між точками J та H? Ліворуч, тому продовжуємо ліворуч у дереві.

- Чи знаходиться M ліворуч чи праворуч від лінії між точками J та N? Праворуч, тому продовжуємо праворуч у дереві.

- Досягнуто кінця дерева, оскільки більше немає розділень у цій гілці. Найближчий сусід тепер визначається шляхом обчислення відстаней між M та всіма іншими точками в цій області. Оскільки N є єдиною іншою точкою в цій області, вона також є найближчим сусідом M.

Оскільки області генеруються випадково, може статися, що створене дерево не представляє кожну область однаково добре. Дивлячись назад на приклад вище, можна побачити, що C найближче до H. Також можна побачити, що ця точка не розглядається, оскільки вона знаходиться в іншій

області, і тому алгоритм повертає I як найближчого сусіда. Бажано мінімізувати ці помилки якомога більше, що вирішується шляхом генерації лісу з кількох дерев замість одного дерева. Ліс містить кілька дерев, кожне зі своїми розділами. Після генерації лісу порівнюються найближчі сусіди з усіх дерев, і повертається найкращий. У більшості реалізацій ANNOY сьогодні використовуються додаткові евристики для подальшого покращення алгоритму.

2.5. Інші варіації метрик

У прикладі вище використовувалася евклідова відстань як міра для визначення найближчих сусідів точок даних. Однак можна розглянути інші варіанти метрик, які підтримує ANNOY, щоб отримати найкращі результати. Ці різні варіанти метрик наведені нижче:

1. Евклідова відстань, яка є прямою відстанню між 2 точками. Евклідова відстань — це стандартна метрика відстані, що використовується для вимірювання прямої лінії між двома точками в Евклідовому просторі. Це найінтуїтивніший спосіб розуміння "відстані", який відповідає тому, як ми вимірюємо відстань у повсякденному житті, наприклад, довжину прямого шляху між двома об'єктами. Вона розраховується наступним чином:

$$\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

2. Відстань Манхеттена, яка є відстанню між двома точками, виміряною вздовж осей під прямим кутом. Відстань Манхеттена, також відома як метрика прямокутного міста, відстань "міських кварталів", відстань таксі або L1-норма, є метрикою відстані, яка визначає відстань між двома точками шляхом підсумовування абсолютних значень різниць їхніх координат уздовж кожної осі.

Вона розраховується наступним чином:

$$\sum_{i=1}^{n} |p_i - q_i|$$

Візуальне представлення подано на рисунку 2.10.

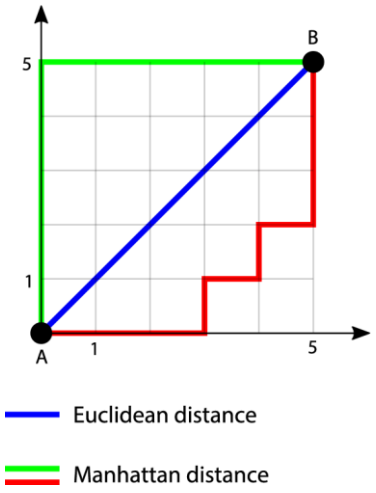

Рисунок 2.10 - Евклідова відстань та відстань Манхеттена

3. Кутова відстань, яка вимірює кут між двома векторами. Вона розраховується наступним чином:

$$\arccos\left(\frac{\langle p,q\rangle}{\|p\|\cdot\|q\|}\right)$$

4. Відстань точкового добутку, яка вимірює косинус кута між двома векторами. Вона розраховується наступним чином:

$$1 - \frac{\langle p,q\rangle}{\|p\|\cdot\|q\|}$$

Візуальне представлення подано на рисунку 2.11.

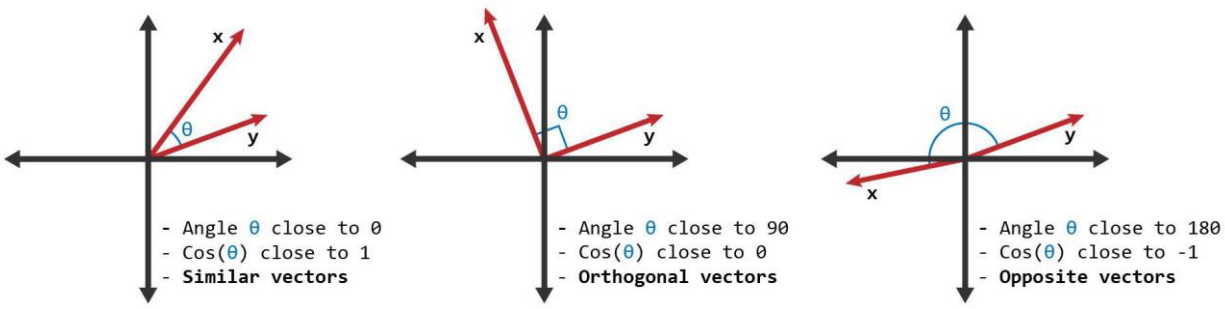

Рисунок 2.11 – Косинусна подібність

Як можна побачити вище, чим більше подібні вектори, тим менший кут і тим більше косинус кута наближається до 1. Оскільки міра відстані повинна бути найменшою для подібних векторів, використовується 1 мінус результат замість цього, щоб отримати хороше представлення відстані для відстані точкового добутку. З кутовою відстанню це не потрібно, оскільки міра відстані є самим кутом, який вже дорівнює 0 для подібних векторів і стає більшим, чим більше вектори відрізняються.

5. Відстань Гаммінга, яка є кількістю заміщень, необхідних для перетворення першого елемента в другий елемент. Відстань Гаммінга — це метрика, яка використовується для вимірювання різниці між двома послідовностями (зазвичай рядками символів або двійковими векторами) однакової довжини. Вона визначається як кількість позицій, у яких відповідні символи цих двох послідовностей відрізняються.

Для обчислення відстані Гаммінга необхідно посимвольно порівняти дві послідовності. Якщо символи на одній і тій же позиції різні, то лічильник відстані збільшується на одиницю Вона розраховується наступним чином:

$$\sum_{i=1}^{n}[p_i \neq q_i]$$

Візуальне представлення подано на рисунку 2.12.

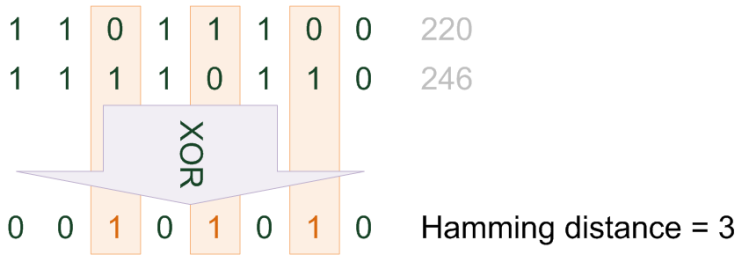

Рисунок 2.12 - Відстань Гаммінга

Як можна побачити вище, кількість заміщень також може бути отримана шляхом застосування операції XOR, якщо представлення елементів є двійковим числом.

2.6. Вибір розмірності метрики

Ознайомившись із різними варіантами метрик, які підтримує ANNOY, необхідно обрати найкращу. Для цього використано наступний приклад з векторами v1 = [1, 4], v2 = [8, 8] та v3 = [3, 1].

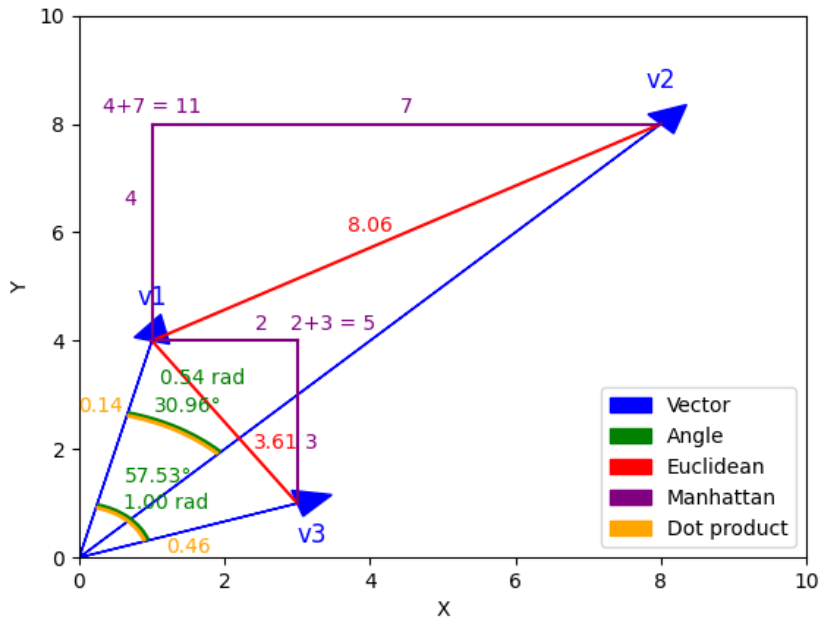

Рисунок 2.13 - Відстань за кутовою мірою порівняно з Евклідовою, Манхеттенською та скалярним добутком

Відстань Гаммінга не показана, але вона дорівнює 2 для всіх векторів, оскільки всі вектори мають різні координати x та y. Можна побачити, що евклідова відстань між v1 та v3 менша, ніж евклідова відстань між v1 та v2. Це також стосується відстані Манхеттена. Однак, дивлячись на кут між v1 та v3, можна побачити, що він більший, ніж кут між v1 та v2. Відстань точкового добутку показує це ще краще, оскільки відстань точкового добутку між v1 та v3 майже втричі більша, ніж між v1 та v2. Отже, чи вважається v1 найбільш схожим на v2 або v3?

Хоча всі обговорені варіанти є хорошими мірами, існує важливий фактор, який необхідно враховувати. Існують випадки, коли певні міри відстані не є хорошою мірою для відстані. Ці випадки виникають у високовимірному просторі. У високовимірному просторі дані стають більш розрідженими. Це збільшення розрідженості призводить до збільшення мінімальної відстані між двома точками в наборі даних. Іншими словами, для кожного додаткового виміру в даних мінімальна відстань збільшується до точки, де майже немає різниці між мінімальною та максимальною відстанню. Це явище відоме як прокляття вимірності.

Щоб зрозуміти, чому це відбувається, почнемо з простого прикладу з кількома точками даних в одномірному просторі.

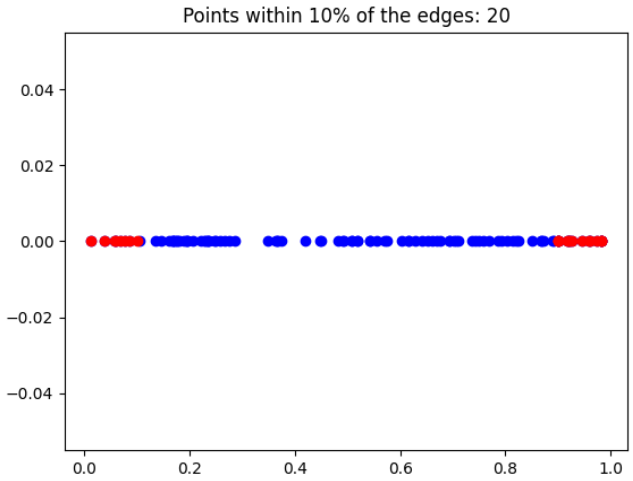

Рисунок 2.14 - Випадкові точки в одновимірному просторі

Зображення вище було створено шляхом генерації 100 випадкових точок в одномірному просторі. Всі точки поблизу країв, точки в межах 10% від краю, забарвлені в червоний колір. Як ми очікували, 20 точок потрапляють в межі 10% від країв, що еквівалентно 20% точок.

```python
import random
import matplotlib.pyplot as plt

# Генерація 100 випадкових точок в 1D просторі
points = [random.uniform(0, 1) for _ in range(100)]

# Підрахунок кількості точок в межах 10% від країв
count = sum(p <= 0.1 or p >= 0.9 for p in points)

# Отримання точок нижче 0.1 та вище 0.9
below = [p for p in points if p < 0.1]
above = [p for p in points if p > 0.9]

# Побудова точок
plt.scatter(points, [0] * 100, c='b')
plt.scatter([0.1, 0.9], [0, 0], c='r')
plt.scatter(below, [0] * len(below), c='r')
plt.scatter(above, [0] * len(above), c='r')

# Додавання заголовка та відображення графіка
plt.title(f"Точки в межах 10% від країв: {count}")
plt.show()
```

Рисунок 2.15 – Код для побудови випадкових точок в одновимірному просторі

Те ж саме робиться для двовимірного простору.

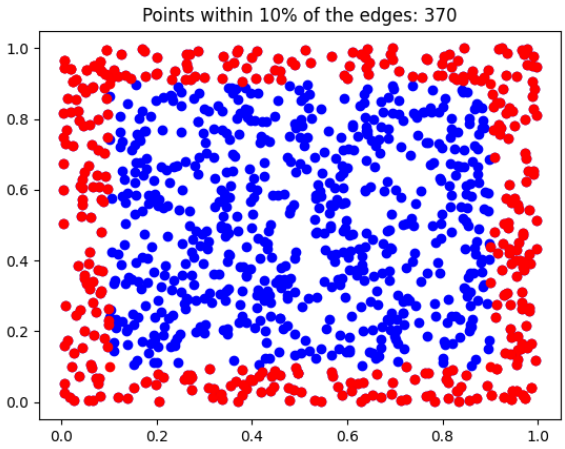

Рисунок 2.16 - Випадкові точки в двохвимірному просторі

Щоб зберегти ту ж саму розрідженість, тепер генерується 1000 випадкових точок замість 100. Як можна побачити, тепер є 370 точок в межах 10% від країв, що майже вдвічі більше.

```python
import random
import matplotlib.pyplot as plt

# Генерація 1000 випадкових точок в 2D просторі
points = [(random.uniform(0, 1), random.uniform(0, 1)) for _ in range(1000)]

# Підрахунок кількості точок в межах 10% від країв
count = sum(min(p) <= 0.1 or max(p) >= 0.9 for p in points)

# Отримання точок нижче 0.1 та вище 0.9
below = [p for p in points if min(p) < 0.1 or max(p) < 0.1]
above = [p for p in points if min(p) > 0.9 or max(p) > 0.9]

# Побудова точок
plt.scatter([p[0] for p in points], [p[1] for p in points], c='b')
plt.scatter([0.1, 0.9, 0.9, 0.1], [0.1, 0.1, 0.9, 0.9], c='r')
plt.scatter([p[0] for p in below], [p[1] for p in below], c='r')
plt.scatter([p[0] for p in above], [p[1] for p in above], c='r')

# Додавання заголовка та відображення графіка
plt.title(f"Точки в межах 10% від країв: {count}")
plt.show()
```

Рисунок 2.17 – Код для побудови випадкових точок в двовимірному просторі

Дивлячись на тривимірний простір, число продовжує зростати.

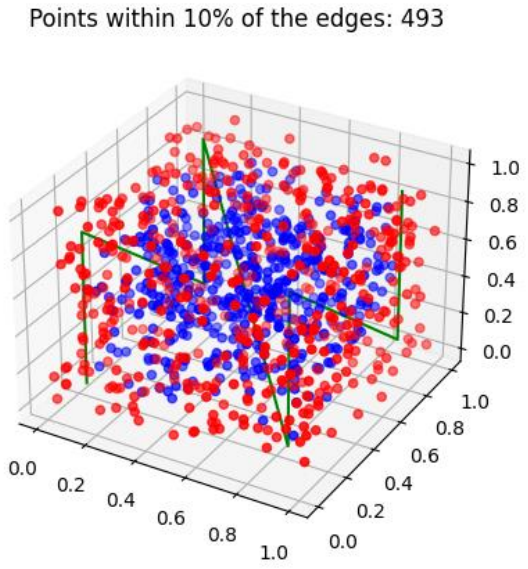

Рисунок 2.18 - Випадкові точки в тривимірному просторі

Це зображення було згенеровано за допомогою 1000 випадкових точок, оскільки з тією ж розрідженістю, як на попередніх двох зображеннях, зображення було б майже суцільним червоним блоком, і сині точки не були б видимі, оскільки вони були б за червоними точками. Як можна побачити, тепер є 493 точки поблизу країв. Це майже 50% від загальної кількості точок даних.

```
import random
import matplotlib.pyplot as plt

# Генерація 1000 випадкових точок в 3D просторі
points = [(random.uniform(0, 1), random.uniform(0, 1), random.uniform(0, 1)) for _ in range(100

# Підрахунок кількості точок в межах 10% від країв
count = sum(1 for point in points if any(p <= 0.1 or p >= 0.9 for p in point))

# Розділення точок всередині та зовні країв
edge_points = [point for point in points if any(p < 0.1 or p > 0.9 for p in point)]
area_points = [point for point in points if all(0.1 <= p <= 0.9 for p in point)]

# Побудова точок
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter([p[0] for p in edge_points], [p[1] for p in edge_points], [p[2] for p in edge_points
ax.scatter([p[0] for p in area_points], [p[1] for p in area_points], [p[2] for p in area_points

# Додавання країв до графіка
ax.plot([0.1, 0.1, 0.1, 0.1, 0.9, 0.9, 0.9, 0.9],
        [0.1, 0.1, 0.9, 0.9, 0.9, 0.9, 0.1, 0.1],
        [0.1, 0.9, 0.9, 0.1, 0.1, 0.9, 0.9, 0.1], c='r')

# Встановлення меж та міток графіка
ax.set_xlim([0, 1])
ax.set_ylim([0, 1])
ax.set_zlim([0, 1])
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# Додавання заголовка та відображення графіка
plt.title(f"Точки в межах 10% від країв: {count}")
plt.show()
```

Рисунок 2.19 – Код для побудови випадкових точок в тривимірному просторі

При подальшому дослідженні даних з більшою кількістю вимірів, кількість точок поблизу країв продовжує зростати як показано нижче в таблиці 2.4.

Таблиця 2.4 – Кількість точок із зростанням вимірів

Кількість вимірів	Відсоток даних в межах 10% від країв
4	60
5	68
6	74
7	80
8	82
9	87

2.7. Феномен концентрації даних на межі у високих вимірах

У контексті проблеми "прокляття розмірності" (Curse of Dimensionality) виникає виразний математичний патерн, що пояснює розподіл даних у просторах високої вимірності. Розглянемо гіперкуб зі стороною довжиною L.

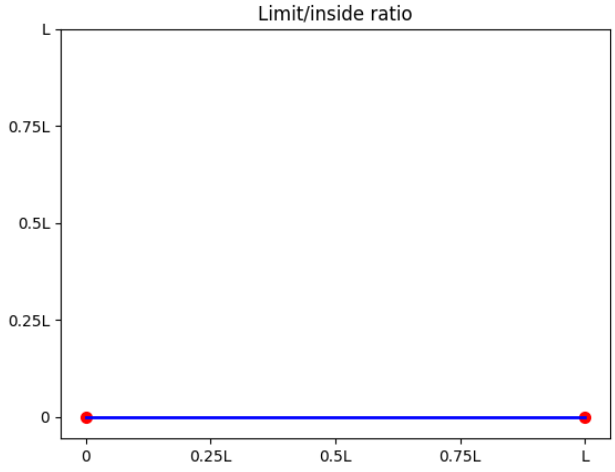

Рисунок 2.20 - Співвідношення межі до внутрішньої частини в одновимірному просторі

У одновимірному випадку (лінійний відрізок довжиною L), гранична область представлена двома кінцевими точками (0 та L), тоді як внутрішня область охоплює всі точки між ними. Співвідношення обсягу граничної області до обсягу внутрішньої області у 1D дорівнює 2/L.

```python
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Побудова точок
L = 1
fig, ax = plt.subplots()
ax.plot([0, 1], [0, 0], color='blue', linewidth=2)
ax.scatter(0, 0, color='red', linewidth=2)
ax.scatter(1, 0, color='red', linewidth=2)

# Встановлення міток для осей x та y
x_ticks = [0, 0.25*L, 0.5*L, 0.75*L, L]
x_ticklabels = ['0', '0.25L', '0.5L', '0.75L', 'L']
ax.set_xticks(x_ticks)
ax.set_xticklabels(x_ticklabels)

y_ticks = [0, 0.25*L, 0.5*L, 0.75*L, L]
y_ticklabels = ['0', '0.25L', '0.5L', '0.75L', 'L']
ax.set_yticks(y_ticks)
ax.set_yticklabels(y_ticklabels)

# Додавання заголовка та відображення графіка
plt.title("Співвідношення межі/внутрішньої частини")
plt.show()
```

Рисунок 2.21 – Код побудови співвідношення межі до внутрішньої частини в одновимірному просторі

Продовжуючи аналіз для просторів вищої вимірності, спостерігається аналогічна закономірність. У двовимірному випадку (квадрат зі стороною L), гранична область представлена периметром (4L), а внутрішня область – площею (L2). Співвідношення становить 4L/L2=4/L.

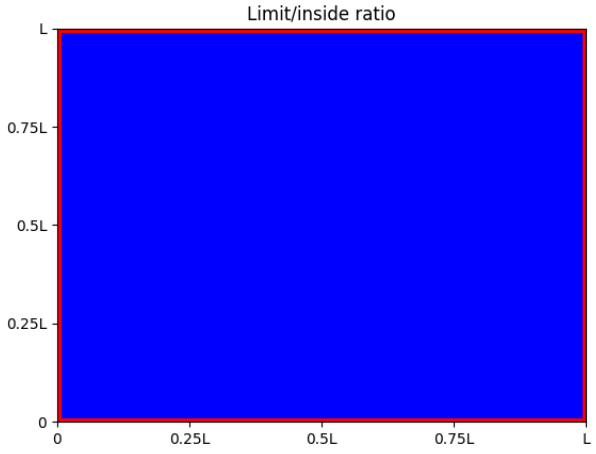

Рисунок 2.22 - Співвідношення межі до внутрішньої частини в двовимірному просторі

```python
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Побудова точок
L = 1
rect = patches.Rectangle((0, 0), L, L, facecolor='blue', edgecolor='red', linewidth=8)
fig, ax = plt.subplots()
ax.add_patch(rect)

# Встановлення міток для осей x та y
x_ticks = [0, 0.25*L, 0.5*L, 0.75*L, L]
x_ticklabels = ['0', '0.25L', '0.5L', '0.75L', 'L']
ax.set_xticks(x_ticks)
ax.set_xticklabels(x_ticklabels)

y_ticks = [0, 0.25*L, 0.5*L, 0.75*L, L]
y_ticklabels = ['0', '0.25L', '0.5L', '0.75L', 'L']
ax.set_yticks(y_ticks)
ax.set_yticklabels(y_ticklabels)

# Додавання заголовка та відображення графіка
plt.title("Співвідношення межі/внутрішньої частини")
plt.show()
```

Рисунок 2.23 – Код побудови співвідношення межі до внутрішньої частини в двовимірному просторі

Для тривимірного простору (куб зі стороною L), гранична область складається з 6 граней (кожна площею L2), що дає 6L2, а внутрішня область – об'єм (L3). Отже, співвідношення дорівнює 6L2/L3=6/L.

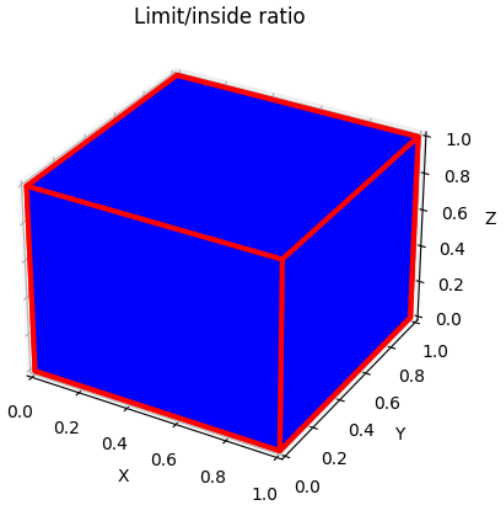

Рисунок 2.24 - Співвідношення межі до внутрішньої частини в двовимірному просторі

```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import numpy as np

# Визначення вершин куба
L = 1
verts = np.array([(0, 0, 0), (0, L, 0), (L, L, 0), (L, 0, 0),
                  (0, 0, L), (0, L, L), (L, L, L), (L, 0, L)])

# Визначення граней куба
faces = np.array([(0, 1, 2, 3), (0, 4, 5, 1), (1, 5, 6, 2),
                  (2, 6, 7, 3), (3, 7, 4, 0), (4, 7, 6, 5)])

# Створення графіка та додавання куба
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
cube = Poly3DCollection(
    [verts[faces[i]] for i in range(len(faces))],
    edgecolors='red',
    facecolors='blue',
    linewidth=3)
ax.add_collection3d(cube)

# Встановлення меж та міток графіка
ax.set_xlim([0, L])
ax.set_ylim([0, L])
ax.set_zlim([0, L])
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.title("Співвідношення межі/внутрішньої частини")

plt.show()
```

Рисунок 2.25 – Код побудови співвідношення межі до внутрішньої частини в тривимірному просторі

Ця закономірність узагальнюється для довільної кількості вимірів D. Співвідношення обсягу граничної області до обсягу внутрішньої області гіперкуба зі стороною L у D-вимірному просторі може бути виражене наступною формулою:

$$Ratio = \frac{2D \cdot L^{D-1}}{L^D} = \frac{2D}{L}$$

Де D — кількість вимірів, а L — довжина сторони гіперкуба.

З цієї формули випливає критично важливий висновок: зі збільшенням кількості вимірів (D), при фіксованій довжині сторони L, більша частка об'єму гіперкуба концентрується поблизу його границь. Це означає, що у високорозмірних просторах більшість точок даних статистично розташовуються ближче до "межі" простору, ніж до його "центру" або глибокої внутрішньої частини. Це явище має значні наслідки для ефективності алгоритмів машинного навчання, особливо тих, що базуються на відстанях та щільності даних, оскільки дані стають все більш "розрідженими" і "віддаленими" один від одного у вищих вимірах.

2.8. Порівняльний аналіз чутливості метрик відстані до проблеми розмірності

У контексті проблеми "прокляття розмірності" вибір адекватної метрики для оцінки подібності між векторами у високорозмірному просторі набуває особливої важливості. Для забезпечення ефективної ідентифікації найближчих об'єктів необхідна метрика, яка зберігає чутливість до незначних змін у векторах, незважаючи на високу розмірність. З метою емпіричної оцінки впливу розмірності на різні метрики відстані, було проведено порівняльний аналіз.

Для порівняльного аналізу чутливості метрик використовувалися два 512-вимірні бінарні вектори: v1 та v2. Вектор v1 складається з чергування значень 0 та 1 ([0,1,0,1,...,0,1]), тоді як вектор v2 характеризується чергуванням пар значень 00 та 11 ([0,0,1,1,...,1,1]). Були розраховані відстані між цими векторами з використанням різних метрик. Крім того, для оцінки чутливості до малих змін, було змодельовано зміну одного біта (перетворення 0 на 1) у векторі v1, після чого повторно обчислювалися відстані та визначався відсоток зміни.

1. Евклідова відстань

Початкова відстань між v1 та v2 була обчислена за формулою Евклідової відстані:

$$dEuclidean(v1,v2)=i=1\sum 512(v1i-v2i)2=256=16$$

При зміні першого біта v1 (0 на 1), відстань становить 257≈16.031. Це відповідає зміні приблизно на 0.19% від початкової відстані.

2. Відстань Манхеттена

Відстань Манхеттена між v1 та v2 становить:

$$dManhattan(v1,v2)=i=1\sum 512 \mid v1i-v2i \mid =256$$

У випадку зміни першого біта v1 (0 на 1), відстань збільшується до 257. Відносне збільшення становить приблизно 0.39%.

3. Кутова відстань

Кутова відстань розраховується як арккосинус косинусної подібності.

Початкова кутова відстань між v1 та v2:

$$dAngular(v1,v2)=arccos(\mid\mid v1\mid\mid \cdot \mid\mid v2\mid\mid v1 \cdot v2)=arccos(256 \cdot 256128)$$
$$=arccos(256128)=arccos(0.5)=60\circ$$

При зміні першого біта v1 (0 на 1), нова кутова відстань становить arccos(257⋅256128)≈60.60∘. Відносне збільшення складає приблизно 1.0%.

4. Відстань на основі скалярного добутку (Dot Product Distance)

Ця метрика часто визначається як 1−Cosine Similarity.

Початкова відстань:

$$dDotProduct(v1,v2)=1-\frac{\lVert v1 \rVert \cdot \lVert v2 \rVert}{v1 \cdot v2}=1-\frac{256 \cdot 256}{128}=1-0.5=0.5$$

При зміні першого біта v1 (0 на 1), нова відстань становить $1-\frac{257 \cdot 256}{128}\approx 0.50097$. Відносне збільшення становить приблизно 0.19%.

5. Відстань Гаммінга

Відстань Гаммінга між v1 та v2:

$$dHamming(v1,v2)=\sum_{i=1}^{512} I(v1i = v2i) = 256$$

У випадку зміни першого біта v1 (0 на 1), відстань збільшується до 257. Відносне збільшення становить приблизно 0.39%.

Результати проведеного порівняльного аналізу демонструють, що кутова відстань демонструє найбільшу чутливість до незначних пертурбацій у високорозмірному просторі, що відображається у найбільшому відсотковому збільшенні відстані після зміни одного біта. Це вказує на її відносну стійкість до феномену "прокляття розмірності" порівняно з Евклідовою, Манхеттенською, Гаммінга та відстанню на основі скалярного добутку, які показали менш виражені зміни. Оскільки для ефективної рекомендаційної системи необхідна метрика, яка зберігає чутливість до малих змін у векторах ознак у високорозмірному просторі, кутова відстань була обрана для використання в кінцевій реалізації системи рекомендацій та підбору стеку користувача.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ РЕКОМЕНДАЦІЇ ТА ПІДБОРУ СТЕКУ ЧИТАЧА

В цьому розділі розглядаються зібрані дані про книги, за якими слідує розділ, який описує, як ці дані попередньо обробляються для використання в системі підбору та рекомендацій. Останній розділ охоплює реалізацію самої системи підбору стека читача.

3.1. Представлення структури бази даних

Результатом використання вищезазначених методів для збору даних про книги з різних джерел є наступна модель бази даних.

Таблиця 3.1 - Змінні про книги в базі даних

Назва змінної	Тип змінної
ISBN13	CHAR(13)
Кількість сторінок	SMALLINT без знаку
Назва книги	VARCHAR(256)
Тип книги	TINYINT без знаку
Мова книги	TINYINT без знаку
Дислексія	BOOLEAN
Мінімальний вік	TINYINT без знаку
Максимальний вік	TINYINT без знаку
AVI	SMALLINT без знаку
Слів на сторінку	DECIMAL(8, 4)
Слів у реченні	DECIMAL(8, 4)
Зображень на сторінку	DECIMAL(8, 4)
Основний розмір шрифту	DECIMAL(8, 4)
Середній розмір шрифту	DECIMAL(8, 4)

Назва змінної	Тип змінної
Відсоток частих слів	DECIMAL(8, 4)
Співвідношення тип/токен	DECIMAL(8, 4)
Середня довжина слова	DECIMAL(8, 4)
Дата випуску	DATE
Ключові слова	ARRAY

Для типу книги, мови книги та AVI використовуються перерахування Python.

```
from enum import Enum

class BookType(Enum):
    LEESBOEK = 0
    INFORMATION = 1
    PRENTENBOEK = 2
    STRIPBOEK = 3
    DICHTBUNDEL = 4
    VOORLEESBOEK = 5
    ORIENTATIE_OP_LEZEN = 6
    ZOEKBOEK = 7
    AANWIJSBOEK = 8

class BookLanguage(Enum):
    DUTCH = 0
    ENGLISH = 1
    GERMAN = 2
    FRENCH = 3
    SPANISH = 4
    FRYSIAN = 5


class Avi(Enum):
    START = 0
    M3 = 1
    E3 = 2
    M4 = 3
    E4 = 4
    M5 = 5
    E5 = 6
    M6 = 7
    E6 = 8
    M7 = 9
    E7 = 10
    PLUS = 11
```

Рисунок 3.1 – Код, що визначає кілька переліків (Enums) у Python, використовуючи модуль enum

Переліки — це набір символічних імен (членів), прив'язаних до унікальних постійних значень. Вони роблять код більш читабельним і менш схильним до помилок, ніж використання "магічних" чисел або рядків без чіткого значення.

Наведемо детальний опис класів `Enum`:

1. `BookType`:

- Цей перелік використовується для класифікації різних типів книг.

- Кожен член (наприклад, `LEESBOEK`, `INFORMATION`) представляє конкретний тип книги та асоційований з цілим числом (наприклад, `LEESBOEK = 0`, `INFORMATION = 1`). Це дозволяє чітко ідентифікувати призначення книги.

- Приклади значень:

  - `LEESBOEK`: книга для читання.

  - `INFORMATION`: інформаційна книга.

  - `PRENTENBOEK`: книга з картинками.

  - `STRIPBOEK`: комікс.

  - `DICHTBUNDEL`: збірка віршів.

  - `VOORLEESBOEK`: книга для читання вголос.

  - `ORIENTATIE_OP_LEZEN`: орієнтація на читання (для новачків).

  - `ZOEKBOEK`: книга-пошуковик (наприклад, "Знайди Уоллі").

  - `AANWIJSBOEK`: книга, де потрібно вказувати (для малюків).

2. `BookLanguage`:

- Цей перелік призначений для ідентифікації мови книги.

- Він містить кілька поширених мов, кожна з яких також прив'язана до цілого числа.

- Наприклад: `DUTCH = 0` (нідерландська), `ENGLISH = 1` (англійська), `GERMAN = 2` (німецька) тощо.

3. `Avi`:

- Цей перелік визначає рівні читабельності AVI. Як обговорювалося раніше, AVI — це показник складності книг, що часто використовується в нідерландській початковій школі.

- Кожен член представляє конкретний рівень AVI (наприклад, `START`, `M3`, `E3`, `PLUS`) і асоційований з числовим значенням. Це дозволяє програмно працювати з цими рівнями замість використання рядків, що зменшує ймовірність помилок.

Такі переліки застосовуються для:

- Підвищення читабельності коду: Замість чисел або рядків, що не мають явного значення, використовується зрозуміле символічне ім'я (наприклад, `BookType.STRIPBOEK` замість `3`).

- Спрощення рефакторингу: Якщо значення, асоційоване з членом переліку, потрібно змінити, це робиться в одному місці, а не по всьому коду.

У контексті системи рекомендацій і підбору, ці переліки дозволяють ефективно і стандартизовано кодувати такі атрибути книг, як їхній тип, мова та рівень AVI, що потім може бути використано для фільтрації, пошуку та персоналізації рекомендацій.

3.2. Процес підготовки даних

Алгоритм ANNOY знаходить приблизно найближчі сусіди точок даних. Ідея полягає в тому, щоб відобразити всі книги в базі даних на точку даних в деякому високовимірному просторі і дозволити ANNOY визначити найближчі сусіди. Для досягнення цього додається вимір для кожної змінної в базі даних, крім ISBN-13 та назви книги:

isbn13 = [Кількість сторінок, Тип книги, Мова книги, Дислексія, Мінімальний вік, Максимальний вік, AVI, Слів на сторінку, Слів у реченні, Зображень на сторінку, Основний розмір шрифту, Середній розмір шрифту, Відсоток частих слів,

Співвідношення тип/токен, Середня довжина слова, Дата випуску, ключові слова]

Наприклад, у даній ситуації, коли є німецька книга з ISBN-13 9781234567890 на 400 сторінок, мінімальний вік 12 з середньою кількістю 250 слів на сторінку, генерується наступний вектор:

```
9781234567890 = [400, 0, 2, 0, 12, 0, 0, 250, 0, 0, 0, 0, 0,
                 0, 0, 0, 0]
```

Однак, існує велика різниця між змінними кількість сторінок та мінімальний вік. Це можна пояснити тим, що ці змінні працюють на різних шкалах, тому їх важко порівнювати одна з одною. Однак обидві ці змінні є важливими даними для книг, тому їх необхідно враховувати при рекомендації книг. Для вирішення цієї проблеми дані масштабуються таким чином, щоб всі значення були між 0 і 1 для всіх змінних. Рівень AVI завжди буде цілим числом між 0 і 11, тому масштабування цього буде легким. На жаль, це не працює для кількості сторінок. Якщо максимальна кількість сторінок встановлена для книги з найбільшою кількістю сторінок, може бути книга з більшою кількістю сторінок, і максимум доведеться коригувати знову. Іншою проблемою є масив ключових слів. Масив не може бути порівняний з цілими числами.

Для вирішення цього кожна змінна розділяється на кілька вимірів з 1 як максимальним значенням і 0 як мінімальним значенням, що можна побачити в додатку. Для віку був використаний наступний код.

Цей код призначений для створення вектора релевантності віку для певного елемента контенту (наприклад, книги). Цей вектор (age_vector) моделює, для яких вікових груп цей елемент є найбільш підходящим або релевантним, використовуючи 16 дискретних вікових категорій (від 0 до 15 років).

```
age_vector = 16 * [0]
if item['min_age'] is not None:
    min_age = cap(15, item['min_age'])
    if item['max_age'] is None:
        age_vector[cap(15, min_age - 2)] = 0.25
        age_vector[cap(15, min_age - 1)] = 0.5
        age_vector[cap(15, min_age + 1)] = 0.5
        age_vector[cap(15, min_age + 2)] = 0.5
        age_vector[cap(15, min_age)] = 1
    else:
        max_age = cap(15, item['max_age'])
        age_vector[cap(15, min_age - 2)] = 0.25
        age_vector[cap(15, min_age - 1)] = 0.5
        age_vector[cap(15, max_age + 1)] = 0.5
        age_vector[cap(15, max_age + 2)] = 0.5
        age_vector[min_age:max_age] = [1] * (max_age - min_age)
```

Рисунок 3.2 - Код для створення вектора релевантності віку

Цей код призначений для створення вектора релевантності віку для певного елемента контенту (наприклад, книги). Цей вектор (`age_vector`) моделює, для яких вікових груп цей елемент є найбільш підходящим або релевантним, використовуючи 16 дискретних вікових категорій (імовірно, від 0 до 15 років).

Розглянемо алгоритм роботи даного коду:

1. Ініціалізація вектора віку:

- `age_vector = 16 - [0]`: Створюється список з 16 нулів. Кожен індекс у цьому списку відповідає певному віку (наприклад, індекс 0 для віку 0, індекс 15 для віку 15 і старше).

2. Перевірка наявності мінімального віку:

- `if item['min_age'] is not None:`: Код виконується лише в тому випадку, якщо для елемента (`item`) вказано мінімальний вік. Якщо `min_age` відсутній, віковий вектор залишається нульовим.

- `min_age = cap(15, item['min_age'])`: Значення `min_age` береться з елемента `item`, але обмежується максимумом у 15. Функція `cap(15, value)` (якої немає у цьому фрагменті, але яка є типовою) гарантує, що вік не виходить за межі діапазону індексів вектора (0-15).

3. Обробка сценаріїв вікового діапазону:

- Якщо `max_age` відсутній (`item['max_age'] is None`):

- Цей блок обробляє контент, який має лише нижню вікову межу (наприклад, "для дітей від 7 років і старше").

- Вектор `age_vector` заповнюється значеннями релевантності, які піково збільшуються на `min_age` (значення `1`) і поступово зменшуються (до `0.5` і `0.25`) для віку на 1 або 2 роки до/після `min_age`. Це створює "розмиту" зону релевантності навколо мінімального віку, дозволяючи рекомендації бути трохи гнучкішими.

- Наприклад: `age_vector[min_age] = 1`, `age_vector[min_age - 1] = 0.5`, `age_vector[min_age + 1] = 0.5` тощо. Функція `cap(15, ...)` використовується для уникнення виходу за межі індексів.

- Якщо `max_age` присутній (`else` блок):

- Цей блок обробляє контент, який має чітко визначений віковий діапазон (наприклад, "для дітей 7-9 років").

- `max_age = cap(15, item['max_age'])`: Значення `max_age` також обмежується 15.

- `age_vector[min_age:max_age] = [1] - (max_age - min_age)`: Усім вікам в межах цього діапазону (від `min_age` до `max_age-1`) присвоюється найвище значення релевантності (`1`).

- Крім того, як і в попередньому випадку, додаються "розмиті" зони релевантності (зі значеннями `0.5` та `0.25`) для вікових груп, що знаходяться безпосередньо за межами цього діапазону (`min_age - 2`, `min_age - 1`, `max_age + 1`, `max_age + 2`).

Цей `age_vector` є важливою ознакою (feature) для даного елемента контенту. У системі рекомендацій він використовується для:

- Персоналізації рекомендацій - порівнюючи вік користувача з цим вектором, система може визначити, наскільки релевантним є даний контент для конкретного користувача.

- Фільтрації контенту - забезпечує, щоб дітям не рекомендувався контент, що явно не відповідає їхньому віку.

Таким чином, цей код ефективно перетворює вікові обмеження книги на числове представлення, яке може бути легко інтегроване в алгоритми машинного навчання для рекомендацій системи підбору стеку читача.

3.3. Ансамблевий метод рекомендацій щодо подолання проблеми домінування ознак

Незважаючи на можливість використання високорозмірних векторів (наприклад, 450-вимірних) у алгоритмах наближеного пошуку найближчих сусідів (наприклад, ANNOY), спостерігається проблема домінування певних груп ознак. У випадках, коли переважна більшість вимірів (наприклад, 400 з 450) відповідає одній категорії (наприклад, тегам), висока схожість за цією категорією може некоректно маскувати значні відмінності за іншими важливими атрибутами (наприклад, віком). Це може призвести до нерелевантних рекомендацій, як-от пропозиція дорослої літератури дитячій, що є неприйнятним для цільової системи.

Це означає, що книги з кількома схожими тегами вже краще відповідатимуть одна одній незалежно від змінної віку, що означає, що хтось, хто читає "Дольф'є Веерволф'є", може отримати рекомендацію "Сутінки".

Щоб гарантувати, що ключові слова не домінують у рекомендаціях, рекомендація розділена на кілька категорій за допомогою наступної формули:

$$\alpha 1 * x1 + \alpha 2 * x2 + \alpha 3 * x3 + \alpha 4 * x4$$

Де αi - вага, прив'язана до частини і формули, x1 - результат ANNOY для тегів, x2 - результат ANNOY для віку, x3 - результат ANNOY для оцінки

avi, а x4 - результат ANNOY для залишкових змінних, таких як кількість сторінок і результати парсера PDF, такі як слів на сторінку, кількість зображень тощо.

```python
from annoy import AnnoyIndex

class GeneratorAnnoy:
    def __init__(self, vector_dict: dict):
        self.vector_dict = vector_dict
        self.vector_list = list(self.vector_dict.values())
        self.vector_length = len(self.vector_list[0])
        # Генерація таблиць перетворення
        self.obj_id_to_index = {}
        self.index_to_obj_id = {}
        for index, obj_id in enumerate(list(self.vector_dict.keys())):
            self.obj_id_to_index[obj_id] = index
            self.index_to_obj_id[index] = obj_id

    def build_forest(self):
        # Ініціалізація списку векторів елементів
        self.annoy_index = AnnoyIndex(self.vector_length, 'angular')
        for i, vector in enumerate(self.vector_list):
            self.annoy_index.add_item(i, vector)
        # Побудова лісу з TREE_COUNT дерев
        self.annoy_index.build(config.TREE_COUNT)

    def save_annoy_index(self):
        """Зберегти ліс у файл"""
        self.annoy_index.save('test.ann')

    def compute_distances(self, obj_id: int, count: int):
        """
        Обчислення найближчих сусідів за заданим obj_id.
        :param int obj_id: ідентифікатор об'єкта, до якого повинні бути обчислені відстані.
        :return list indices: індекси найближчих сусідів
        :return list distances: відстані до найближчих сусідів.
        """
        index = self.obj_id_to_index[obj_id]
        result = self.annoy_index.get_nns_by_item(index, count)
        indices, distances = result
        return indices, distances

    def compute_similarities(self, count: int):
        similarities = {}
        for obj_id in self.vector_dict.keys():
            indices, distances = self.compute_distances(obj_id, count)
            # Створення словника подібностей
            # Шаблон: dict[base_id][recommendation_id] = distance
            similarities[obj_id] = {}
            for i in range(count):
                index = indices[i]
                similarities[obj_id][self.index_to_obj_id[index]] = distances[i]
        return similarities
```

Рисунок 3.3 – Використання бібліотеки Annoy

Для обчислення індивідуальних оцінок подібності в межах кожної категорії використовується бібліотека ANNOY (Approximate Nearest Neighbors Oh Yeah). Ця бібліотека дозволяє ефективно будувати індекси для швидкого пошуку наближених найближчих сусідів у високорозмірному просторі. Для кожної категорії ознак (теги, вік, AVI, залишкові змінні) окремо створюється незалежний ANNOY-індекс, який зберігає вектори, що представляють лише ознаки даної категорії. Після побудови лісу дерев (з визначеною кількістю дерев), система може оперативно обчислювати найближчих сусідів та відповідні відстані для кожного елемента контенту в межах кожної категорії. Результатом є набір словників, де для кожного ISBN-13 книги представлений список рекомендованих ISBN-13 разом з їхніми відстанями в межах конкретної категорії

Розділивши різні категорії таким чином, кожній категорії присвоюється власна вага для визначення, наскільки важливою є кожна категорія для кінцевого результату. Щоб забезпечити, що кінцевий результат знаходиться між 0 і 1, важливо, щоб Σαi від i=1 до 4 дорівнювало 1. Єдине, що потрібно змінити для досягнення цього, - розділити згенерований масив на 4 частини, по одній на категорію. Бібліотека Annoy використовується наступним чином (рис. 3.3).

Клас GeneratorAnnoy потім використовується наступним чином для обчислення найближчих сусідів на основі віку (рис. 3.4).

```
def compute_similarities(vector_dict: dict, count: int):
    count = count if count < len(vector_dict) else len(vector_dict)
    generator = GeneratorAnnoy(vector_dict)
    generator.build_forest()
    similarities = generator.compute_similarities(count)
    return similarities

age_similarities = compute_similarities(
    edition_to_age_vector_dict,
    config.CALCULATION_RECOMMENDATION_COUNT
)
```

Рисунок 3.4 – Використання класу GeneratorAnnoy

Результатом є словник з номерами ISBN-13 книги як ключами та списком номерів ISBN-13 рекомендацій у порядку як значенням. Те ж саме робиться для інших категорій, щоб отримати найближчі сусіди для кожного ISBN-13 на категорію. Щоб застосувати вищезазначену формулу, важливо пам'ятати, що всі категорії можуть мати різні рекомендації ISBN-13. Щоб правильно впоратися з цим, використовується наступний код (рис. 3.5).

Код (рис. 3.5) ітерує по всіх рекомендаціях на категорію та вставляє оцінку в словник keyword_recommendations під відповідною категорією. Оскільки бібліотека Annoy повертає оцінку між 0 і 2, оцінку необхідно розділити на 2, щоб отримати результат між 0 і 1. Також, оскільки більш схожі вектори мають нижчу оцінку, застосовується 1 - (оцінка / 2), щоб отримати бажаний результат, а саме схожі вектори без кута між ними мають оцінку 1.

```
recommendation_scores = {}
for id in keyword_recommendations.keys():
    recommendation_scores[id] = {'score': 0, 'tags': 0, 'age': 0, 'avi': 0, 'remainder': 0}
for id, score in keyword_recommendations.items():
    recommendation_scores[id]['tags'] = 1 - (score / 2)
for id in age_recommendations.keys():
    if id not in recommendation_scores:
        recommendation_scores[id] = {'score': 0, 'tags': 0, 'age': 0, 'avi': 0, 'remainder': 0}
for id, score in age_recommendations.items():
    recommendation_scores[id]['age'] = 1 - (score / 2)
for id in avi_recommendations.keys():
    if id not in recommendation_scores:
        recommendation_scores[id] = {'score': 0, 'tags': 0, 'age': 0, 'avi': 0, 'remainder': 0}
for id, score in avi_recommendations.items():
    recommendation_scores[id]['avi'] = 1 - (score / 2)
for id in remainder_recommendations.keys():
    if id not in recommendation_scores:
        recommendation_scores[id] = {'score': 0, 'tags': 0, 'age': 0, 'avi': 0, 'remainder': 0}
for id, score in remainder_recommendations.items():
    recommendation_scores[id]['remainder'] = 1 - (score / 2)

for id in recommendation_scores.keys():
    recommendation_scores[id]['score'] = (
        0.8 * recommendation_scores[id]['tags'] +
        0.1 * recommendation_scores[id]['age'] +
        0.05 * recommendation_scores[id]['avi'] +
        0.05 * recommendation_scores[id]['remainder']
    )
```

Рисунок 3.5 – Код для визначення категорії

Щоб отримати остаточну оцінку рекомендації, береться оцінка за категорією та множиться на відповідні ваги. Ці ваги були результатом багатьох тестів і можуть змінюватися з часом. Наприклад, коли вага для категорії віку була занадто високою, були деякі книги, які рекомендувалися майже для кожної іншої книги, що, ймовірно, було через те, що вони мали великий діапазон віку, який зробив їх відповідними більшості книг.

3.4. Результати використання інтелектуальної системи рекомендацій та підбору стеку читача

Після реалізації системи рекомендацій у додатку результати були обнадійливими.

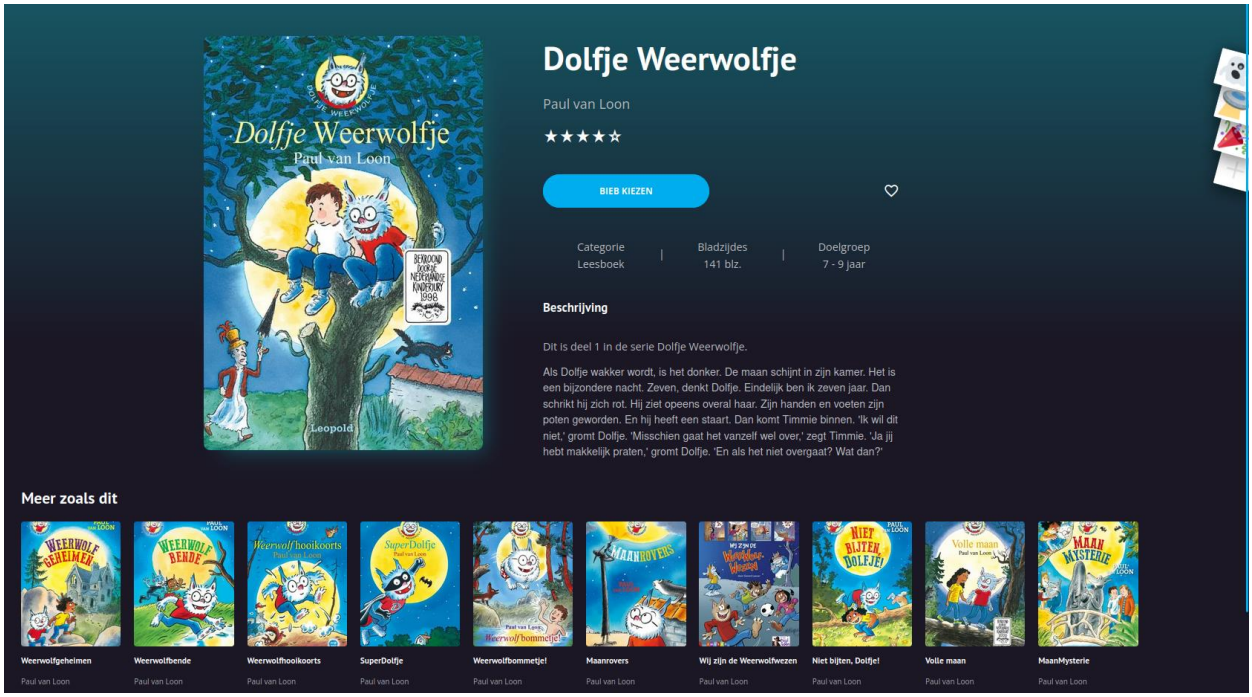

Рисунок 3.6 - Рекомендації "Дольф'є Вірвольф'є" в додатку

Як видно, книга про Дольф'є Вірвольф'є отримує рекомендації щодо інших книг про перевертнів, більшість з яких також є Дольф'є Вірвольф'є.

Переглядаючи базу даних, можна отримати оцінки, які рекомендація отримує в кожній категорії.

ibsn13_book	overall_score	tags_score	age_score	avi_score	remainder_score	sequence_number	isbn13
9789025861346	0.8221	0.7983	1.0000	0.6445	0.5685	0	9789025851200
9789025861346	0.8221	0.7983	1.0000	0.6445	0.5685	0	9789025867614
9789025861346	0.8182	0.7933	1.0000	0.6445	0.5685	1	9789025856663
9789025861346	0.7920	0.7402	1.0000	0.6445	0.8672	2	9789025862800
9789025861346	0.7920	0.7402	1.0000	0.6445	0.8672	2	9789025875626
9789025861346	0.7896	0.7402	1.0000	0.6445	0.8187	3	9789025861759
9789025861346	0.7896	0.7402	1.0000	0.6445	0.8187	3	9789025866402
9789025861346	0.7896	0.7402	1.0000	0.6445	0.8187	3	9789025881108
9789025861346	0.7860	0.7418	1.0000	1.0000	0.7224	4	9789025869380
9789025861346	0.7860	0.7418	1.0000	1.0000	0.7224	4	9789025877842
9789025861346	0.7810	0.7180	1.0000	0.6445	0.9896	5	9789025880897
9789025861346	0.7702	0.7287	1.0000	0.3334	0.6082	6	9789048736645
9789025861346	0.7665	0.7314	1.0000	0.6445	0.4941	7	9789025860097
9789025861346	0.7625	0.7213	1.0000	0.2929	0.5685	8	9789025845858
9789025861346	0.7625	0.7213	1.0000	0.2929	0.5685	8	9789025866648
9789025861346	0.7595	0.6983	1.0000	0.3334	0.8663	9	9789025870584
9789025861346	0.7590	0.7169	1.0000	0.2929	0.5685	10	9789025855277
9789025861346	0.7590	0.7169	1.0000	0.2929	0.5685	10	9789025880170
9789025861346	0.7581	0.7058	1.0000	1.0000	0.7223	11	9789025881122
9789025861346	0.7292	0.6509	1.0000	0.6445	0.9953	12	9789025877316
9789025861346	0.7253	0.6636	1.0000	0.6445	0.7208	13	9789025849245
9789025861346	0.7253	0.6636	1.0000	0.6445	0.7208	13	9789025863029
9789025861346	0.7253	0.6636	1.0000	0.6445	0.7208	13	9789025876159
9789025861346	0.7231	0.6754	1.0000	1.0000	0.4941	14	9789045120188
9789025861346	0.7201	0.6505	1.0000	0.6445	0.8185	15	9789025876814
9789025861346	0.7166	0.7190	0.6772	0.6445	0.8180	16	9789025860684
9789025861346	0.7151	0.6603	1.0000	0.6445	0.5685	17	9789048807154
9789025861346	0.7137	0.6423	1.0000	1.0000	0.8192	18	9789025873035
9789025861346	0.7137	0.6423	1.0000	1.0000	0.8192	18	9789025883133
9789025861346	0.7096	0.6432	1.0000	1.0000	0.7220	19	9789025842185
9789025861346	0.7096	0.6432	1.0000	1.0000	0.7220	19	9789025862787
9789025861346	0.7014	0.6425	1.0000	0.6445	0.5685	20	9789048802173

Рисунок 3.7 - Оцінки, збережені в базі даних для Dolfje Weerwolfje

Як можна побачити, всі рекомендовані книги збігаються на 100% за віком. Це можна пояснити тим, що вік має велику вагу в формулі рекомендації. Адже "Сутінки" не повинні рекомендуватися дітям 8 років. Також можна побачити, що іноді є кілька рекомендацій ISBN13 на номер послідовності. Це відбувається тому, що деякі книги мають кілька видань, що означає, що книги майже ідентичні, за винятком кількох змін. У цьому випадку номери ISBN-13 будуть відображені на ту саму книгу в системі рекомендацій. Це також запобігає тому, щоб інше видання тієї ж книги було рекомендоване. На момент написання, AVI не має великої ваги в формулі рекомендації, оскільки ці дані невідомі або неточні для більшості книг.

Дивлячись на рекомендації для частини 5 "Грайзе Ягер", рекомендації трохи різноманітніші.

Рисунок 3.8 - Рекомендації "Сірого Мисливця" в додатку

Як можна побачити, рекомендуються інші книги " Сірого Мисливця ". Також можна побачити, що рекомендуються інші книги у жанрі фентезі. Однак, дивлячись на оцінки в базі даних, перша рекомендована книга має значно вищу оцінку, ніж інші.

ibsn13_book	overall_score	tags_score	age_score	avi_score	remainder_score	sequence_number	isbn13_recommendation
9789025861346	0.7863	0.7242	1.0000	0.6445	1.0000	0	9789025744960
9789025861346	0.7863	0.7242	1.0000	0.6445	1.0000	0	9789025751944
9789025861346	0.6073	0.6402	0.5160	0.2985	0.4160	1	9789000374298
9789025861346	0.6047	0.6083	0.4756	0.2985	1.0000	2	9789085921660
9789025861346	0.5984	0.5876	0.5316	0.4439	1.0000	3	9789047713166
9789025861346	0.5977	0.6252	0.3605	0.6445	1.0000	4	9789020683547
9789025861346	0.5969	0.4799	1.0000	0.4439	1.0000	5	9789025752804
9789025861346	0.5839	0.5689	0.5316	0.6445	1.0000	6	9789047713173
9789025861346	0.5821	0.5697	0.6127	0.2929	0.6667	7	9781408855669
9789025861346	0.5787	0.5741	0.4784	0.3334	1.0000	8	9789054446415
9789025861346	0.5787	0.5741	0.4784	0.3334	1.0000	8	9789076174112
9789025861346	0.5787	0.5741	0.4784	0.3334	1.0000	8	9789076174129
9789025861346	0.5770	0.5985	0.3605	0.4439	1.0000	9	9789085922858
9789025861346	0.5765	0.6360	0.3285	1.0000	0.5227	10	9789025871642
9789025861346	0.5717	0.5658	0.4756	0.4439	1.0000	11	9789077826751
9789025861346	0.5717	0.5658	0.4756	0.4439	1.0000	11	9789085922025
9789025861346	0.5681	0.5763	0.5316	1.0000	0.5704	12	9789000374274
9789025861346	0.5661	0.5738	0.5316	0.2929	0.5681	13	9789025771508
9789025861346	0.5651	0.5445	0.5316	0.3334	1.0000	14	9789026156458
9789025861346	0.5642	0.6298	0.2929	0.3334	0.4966	15	9789025868000
9789025861346	0.5599	0.6066	0.3710	0.6445	0.4972	16	9789025868758
9789025861346	0.5591	0.6088	0.3576	1.0000	0.4930	17	9789025867621
9789025861346	0.5541	0.5399	0.6127	0.2929	0.5694	18	9789402707397
9789025861346	0.5528	0.5680	0.3576	1.0000	1.0000	19	9789056377144
9789025861346	0.5498	0.5335	0.4935	0.2929	1.0000	20	9789061697008
9789025861346	0.5498	0.5335	0.4935	0.2929	1.0000	20	9789061697015

Рисунок 3.9 - Оцінки, збережені в базі даних для "Сірого Мисливця"

Оцінки віку та AVI також трохи низькі, що можна пояснити тим, що книги, такі як " Сірого Мисливця " та "Гаррі Поттер", не написані спеціально для дітей початкових шкіл і також орієнтовані на старшу аудиторію. Це означає, що рекомендації в основному базуються на тегах та залишкових змінних, що також можна побачити на зображенні.

Однак, щоб отримати належні результати, щоб побачити, чи дійсно система рекомендацій покращує задоволення від читання для учнів, систему необхідно використовувати протягом кількох місяців, перш ніж можна буде спостерігати зміни. На момент написання, багато дітей починають книгу, але не закінчують її, або тому, що їм не подобається книга, або тому, що вона занадто легка або важка. Якщо після кількох місяців використання системи рекомендацій кількість дітей, які дійсно закінчують книгу, збільшиться, лише тоді можна буде стверджувати, що система рекомендацій книг працює належним чином.

Подальша оптимізація та розширення функціональних можливостей рекомендаційної системи є критично важливими для підвищення її ефективності та релевантності. Першочерговим завданням є забезпечення актуальності даних, що вимагає безперервного моніторингу та адаптації скрейперів до змін у структурі веб-сайтів-джерел. Окрім цього, існують кілька ключових напрямків для вдосконалення алгоритму рекомендацій.

1. Розширення простору ознак.

Необхідно інтегрувати додаткові змінні до вектора рекомендацій для більш повного представлення об'єктів контенту. Це може включати, але не обмежуватися, метаданими, що стосуються видавництва, року публікації, рецензій критиків, наявності ілюстрацій або інших специфічних характеристик, які можуть впливати на сприйняття користувачем.

2. Оптимізація семантичного ядра ключових слів.

Постійне оновлення та розширення словника ключових слів, а також диначна корекція їхніх вагових коефіцієнтів у дереві ключових слів, дозволить підвищити точність тематичної відповідності рекомендацій.

3. Збільшення обсягу та якості даних PDF-аналізу.

Розширення бази даних книг, для яких доступний аналіз перших кількох сторінок у форматі PDF, забезпечить багатший набір текстових та структурних ознак (наприклад, щільність слів на сторінку, кількість зображень). Це може бути досягнуто як шляхом автоматизованого онлайн-скрейпінгу, так і за допомогою ручного збору та сканування в бібліотеках.

Поточна архітектура системи спирається на емпірично визначені вагові коефіцієнти для різних категорій ознак та ключових слів. Впровадження методів штучного інтелекту (наприклад, еволюційних алгоритмів, нейронних мереж або методів оптимізації) дозволить автоматично визначати оптимальні значення цих ваг. Це, ймовірно, призведе до значного підвищення точності та релевантності рекомендацій, оскільки ШІ здатен виявляти складні залежності у даних, які важко ідентифікувати за допомогою ручної настройки.

Наявні дані про рівень AVI та рекомендований вік для книг часто базуються на інформації, отриманій з відкритих джерел, яка може бути неповною або неточною. Майбутні дослідження мають бути спрямовані на інтеграцію даних про взаємодії користувачів (наприклад, вік учнів, які читають певні книги) для уточнення цих параметрів. Використання поведінкових даних з додатку дозволить переоцінити та перезаписати існуючі значення, забезпечуючи більш точне відображення реального віку та рівня читабельності для кожної книги.

Для підвищення довіри користувачів та спрощення процесу налагодження системи, пропонується відображати індивідуальні оцінки за категоріями безпосередньо в інтерфейсі додатку. Наразі ці оцінки зберігаються у базі даних, але не є доступними для користувачів. Візуалізація факторів, що вплинули на конкретну рекомендацію, дозволить користувачам

краще розуміти логіку системи. Для розробників це надасть цінну інформацію для ідентифікації випадків, коли рекомендації є нерелевантними, що полегшить процес виявлення та усунення недоліків в алгоритмах.

Поточна система генерує рекомендації за принципом "об'єкт-до-об'єкта" (item-to-item), базуючись на схожості між книгами. Натомість, персоналізовані рекомендації передбачають створення індивідуального вектора читацьких вподобань користувача на основі історії прочитаних та/або покинутих книг. Цей персоналізований вектор може бути використаний для:

- рекомендацій "користувач-до-об'єкта". Використання ANNOY для пошуку книг, які найкраще відповідають індивідуальному профілю читача.

- рекомендацій "користувач-до-користувача" (Collaborative Filtering). Порівняння вектора читацьких вподобань активного користувача з векторами інших користувачів для виявлення "сусідів" зі схожими смаками. Це дозволить рекомендувати книги, які були високо оцінені схожими користувачами.

Таким чином, поточна система рекомендацій "об'єкт-до-об'єкта" слугує міцною основою для майбутнього переходу до більш складних та персоналізованих моделей.

ВИСНОВКИ

В дипломній роботі було здійснено комплексний аналіз проблематики побудови інтелектуальних систем рекомендацій з урахуванням особливостей предметної області – персоналізованого підбору стеку читача. Було визначено передумови для програмної реалізації, зокрема окреслено джерела даних, методи їх отримання та структурування. Значну увагу приділено побудові механізмів семантичної обробки, таких як асоціація ключових слів, побудова ієрархії інтересів та розрахунок релевантності на основі словників тригерних термінів.

Робота охопила методи оцінювання читабельності й зрозумілості текстів (індекси AVI та CLIB), що є важливими параметрами при формуванні персоналізованих рекомендацій. На цій основі здійснено поєднання лінгвістичних показників із профілем користувача, що дозволяє більш точно підбирати відповідний стек літератури.

У другому розділі проведено дослідження алгоритмів парсингу тексту, методів контентної та спільної фільтрації, а також аналіз ефективності використання алгоритму ANNOY у задачах побудови рекомендацій у векторному просторі. Окремо розглянуто проблему «прокляття розмірності», яка виникає у випадках високої кількості ознак, та запропоновано підходи до оптимального вибору метрик подібності та розмірності простору ознак.

У третьому розділі реалізовано прототип інтелектуальної системи рекомендацій, що поєднує дані з зовнішніх джерел, механізми попередньої обробки, а також ансамблеві підходи до рекомендацій для зменшення ефекту домінування окремих ознак. Представлено структуру бази даних та процес підготовки даних, здійснено тестування системи та отримано результати, що демонструють здатність системи формувати індивідуалізовані рекомендації стеків читача з урахуванням когнітивних характеристик, інтересів та тематичної релевантності текстів.

Отримані результати свідчать про доцільність застосування комплексного підходу до побудови інтелектуальних систем рекомендацій у сфері персоналізованого читання, що поєднує семантичні, статистичні та алгоритмічні методи аналізу даних.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 17(6), 734-749.
2. Ricci, F., Rokach, L., & Shapira, B. (2011). Recommender Systems Handbook. Springer.
3. Resnick, P., & Varian, H. R. (1997). Recommender systems. Communications of the ACM, 40(3), 56-59.
4. Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. Proceedings of the 10th International Conference on World Wide Web, 285-295.
5. Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, 43-52.
6. Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30-37.
7. Lops, P., de Gemmis, M., & Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. In Recommender Systems Handbook (pp. 73-105). Springer.
8. Pazzani, M. J., & Billsus, D. (2007). Content-based recommendation systems. In The Adaptive Web (pp. 325-341). Springer.
9. Bellman, R. (1961). Adaptive control processes: a guided tour. Princeton University Press. (Origin of "Curse of Dimensionality")
10. Aggarwal, C. C., Hinneburg, F., & Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional spaces. International Conference on Database Theory, 420-434.

11. Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity search in high dimensions via hashing. Proceedings of the 25th International Conference on Very Large Data Bases, 518-529.
12. Burke, R. (2002). Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction, 12(4), 331-370. 13. Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems (TOIS), 22(1), 5-53.
13. Mitchell, T. M. (1997). Machine learning. McGraw-Hill.
14. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.
15. Russell, S. J., & Norvig, P. (2010). Artificial intelligence: a modern approach. Prentice Hall.
16. Tan, P. N., Steinbach, M., & Kumar, V. (2005). Introduction to data mining. Pearson Education.
17. Shani, G., & Gunawardana, A. (2011). Evaluating recommender systems. In Recommender Systems Handbook (pp. 257-297). Springer.
18. Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing, 7(1), 76-80.
19. Zhou, Y., Cao, Y., & Zeng, S. (2010). Learning to rank for recommender systems. ACM.
20. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.
21. Deng, L., & Yu, D. (2014). Deep Learning: Methods and Applications. Foundations and Trends in Signal Processing, 7(3–4), 197–387.
22. Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). Mining of Massive Datasets. Cambridge University Press.
23. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

24. Cover, T. M., & Thomas, J. A. (2006). Elements of information theory. John Wiley & Sons.
25. Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. ACM computing surveys (CSUR), 31(3), 264-323.
26. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
27. Van der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. Journal of Machine Learning Research, 9(Nov), 2579-2605.
28. Duda, R. O., Hart, P. E., & Stork, D. G. (2001). Pattern classification. John Wiley & Sons.
29. Klabjan, D., & Li, R. (2009). Recommender Systems for the Netflix Prize. Springer.
30. Balabanovic, M., & Shoham, Y. (1997). Fab: content-based, collaborative recommendation. Communications of the ACM, 40(3), 66-72.
31. Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2002). Recommender systems for large-scale e-commerce: Challenges and solutions. Proceedings of the 18th National Conference on Artificial Intelligence, 170-175.
32. Bell, R. M., & Koren, Y. (2007). Lessons from the Netflix Prize. ACM SIGKDD Explorations Newsletter, 9(2), 57-61.
33. Manning, C. D., Schütze, H., & Hinrichs, K. (2014). Foundations of Statistical Natural Language Processing. MIT Press.
34. Rendle, S. (2010). Factorization machines. 2010 IEEE International Conference on Data Mining, 995-1000.

ДОДАТКИ

Додаток А

Попередня обробка даних

Назва змінної	Індекс	Значення
Кількість сторінок	0	1, якщо кількість сторінок < 100
	1	1, якщо 100 ≤ кількість сторінок < 250
	2	1, якщо кількість сторінок ≥ 250
Тип книги	3	1, якщо тип книги 0 (книга для читання)
	4	1, якщо тип книги 1 (інформаційна книга)
	5	1, якщо тип книги 2 (книга з картинками)
	6	1, якщо тип книги 3 (комікс)
	7	1, якщо тип книги 4 (збірка віршів)
	8	1, якщо тип книги 5 (книга для читання вголос, для вчителя читати класу)
	9	1, якщо тип книги 6 (книга для орієнтації на читання)
	10	1, якщо тип книги 7 (книга для пошуку, учень повинен знайти щось у книзі)
	11	1, якщо тип книги 8 (книга для вказівки, учень повинен вказувати на речі в книзі)
Мова книги	12	1, якщо мова книги 1 (нідерландська)
	13	1, якщо мова книги 1 (англійська)
	14	1, якщо мова книги 2 (німецька)
	15	1, якщо мова книги 3 (французька)
	16	1, якщо мова книги 4 (іспанська)
	17	1, якщо мова книги 5 (фризька)
Дислексія	18	1, якщо книга є версією для дислексії
AVI	19	0.1, якщо AVI - 2 == 0 0.5, якщо AVI - 1 == 0 1, якщо AVI == 0 0.5, якщо AVI + 1 == 0 0.5, якщо AVI + 2 == 0
	20	0.1, якщо AVI - 2 == 1 0.5, якщо AVI - 1 == 1 1, якщо AVI == 1 0.5, якщо AVI + 1 == 1 0.5, якщо AVI + 2 == 1
	21	0.1, якщо AVI - 2 == 2 0.5, якщо AVI - 1 == 2 1, якщо AVI == 2 0.5, якщо AVI + 1 == 2 0.5, якщо AVI + 2 == 2
	22	0.1, якщо AVI - 2 == 3 0.5, якщо AVI - 1 == 3 1, якщо AVI == 3

Назва змінної	Індекс	Значення
		0.5, якщо AVI + 1 == 3 0.5, якщо AVI + 2 == 3
	23	0.1, якщо AVI - 2 == 4 0.5, якщо AVI - 1 == 4 1, якщо AVI == 4 0.5, якщо AVI + 1 == 4 0.5, якщо AVI + 2 == 4
	24	0.1, якщо AVI - 2 == 5 0.5, якщо AVI - 1 == 5 1, якщо AVI == 5 0.5, якщо AVI + 1 == 5 0.5, якщо AVI + 2 == 5
	25	0.1, якщо AVI - 2 == 6 0.5, якщо AVI - 1 == 6 1, якщо AVI == 6 0.5, якщо AVI + 1 == 6 0.5, якщо AVI + 2 == 6
	26	0.1, якщо AVI - 2 == 7 0.5, якщо AVI - 1 == 7 1, якщо AVI == 7 0.5, якщо AVI + 1 == 7 0.5, якщо AVI + 2 == 7
	27	0.1, якщо AVI - 2 == 8 0.5, якщо AVI - 1 == 8 1, якщо AVI == 8 0.5, якщо AVI + 1 == 8 0.5, якщо AVI + 2 == 8
	28	0.1, якщо AVI - 2 == 9 0.5, якщо AVI - 1 == 9 1, якщо AVI == 9 0.5, якщо AVI + 1 == 9 0.5, якщо AVI + 2 == 9
	29	0.1, якщо AVI - 2 == 10 0.5, якщо AVI - 1 == 10 1, якщо AVI == 10 0.5, якщо AVI + 1 == 10 0.5, якщо AVI + 2 == 10
	30	0.1, якщо AVI - 2 == 11 0.5, якщо AVI - 1 == 11 1, якщо AVI == 11 0.5, якщо AVI + 1 == 11 0.5, якщо AVI + 2 == 11
Слів на сторінку	31	1, якщо слів на сторінку < 50 0.25, якщо 50 ≤ слів на сторінку < 100
	32	0.5, якщо слів на сторінку < 50 1, якщо 50 ≤ слів на сторінку < 100 0.5, якщо 100 ≤ слів на сторінку < 180

Назва змінної	Індекс	Значення
	33	0.5, якщо 50 ≤ слів на сторінку < 100 1, якщо 100 ≤ слів на сторінку < 180 0.5, якщо 180 ≤ слів на сторінку < 300
	34	0.5, якщо 100 ≤ слів на сторінку < 180 1, якщо 180 ≤ слів на сторінку < 300 0.5, якщо 300 ≤ слів на сторінку < 400
	35	0.5, якщо 180 ≤ слів на сторінку < 300 1, якщо 300 ≤ слів на сторінку < 400 0.75, якщо слів на сторінку ≥ 400
	36	0.5, якщо 300 ≤ слів на сторінку < 400 1, якщо слів на сторінку ≥ 400
Зображень на сторінку	37	1, якщо зображень на сторінку < 0.1
	38	1, якщо 0.1 ≤ зображень на сторінку < 0.67
	39	1, якщо зображень на сторінку > 0.67
Основний розмір шрифту	40	1, якщо (основний розмір шрифту - середній розмір шрифту) > 1.5, якщо середній розмір шрифту ≤ 12 1, якщо (основний розмір шрифту - середній розмір шрифту) ≤ 1.5, якщо основний розмір шрифту ≤ 12
	41	1, якщо (основний розмір шрифту - середній розмір шрифту) > 1.5, якщо основний розмір шрифту > 12 1, якщо (основний розмір шрифту - середній розмір шрифту) ≤ 1.5, якщо основний розмір шрифту > 12
Відсоток частих слів	42	Відсоток/100
Співвідношення тип/токен	43	Співвідношення тип/токен поділене на 100
Дата випуску	44	1, якщо дата випуску < 2000 0.5, якщо 2000 ≤ дата випуску < 2005
	45	0.5, якщо дата випуску < 2000 1, якщо 2000 ≤ дата випуску < 2005 0.5, якщо 2005 ≤ дата випуску < 2010
	46	0.5, якщо 2000 ≤ дата випуску < 2005 1, якщо 2005 ≤ дата випуску < 2010 0.5, якщо 2010 ≤ дата випуску < 2015
	47	0.5, якщо 2005 ≤ дата випуску < 2010 1, якщо 2010 ≤ дата випуску < 2015 0.5, якщо 2015 ≤ дата випуску < 2020
	48	0.5, якщо 2010 ≤ дата випуску < 2015 1, якщо 2015 ≤ дата випуску < 2020 0.5, якщо дата випуску > 2020
	49	0.5, якщо 2015 ≤ дата випуску < 2020 1, якщо дата випуску > 2020
Значення CILT	50	1, якщо значення CILT < 54.9 0.5, якщо 54.9 ≤ значення CILT < 56.9 0.25, якщо 56.9 ≤ значення CILT < 58.9

Назва змінної	Індекс	Значення
	51	0.5, якщо значення CILT < 54.9 1, якщо 56.9 ≤ значення CILT < 58.9 0.5, якщо 58.9 ≤ значення CILT < 61.9 0.25, якщо 61.9 ≤ значення CILT < 63.9
	52	0.1, якщо значення CILT < 54.9 0.5, якщо 54.9 ≤ значення CILT < 56.9 1, якщо 56.9 ≤ значення CILT < 58.9 0.5, якщо 58.9 ≤ значення CILT < 61.9 0.25, якщо 61.9 ≤ значення CILT < 63.9
	53	0.1, якщо 54.9 ≤ значення CILT < 56.9 0.5, якщо 56.9 ≤ значення CILT < 58.9 1, якщо 58.9 ≤ значення CILT < 61.9 0.5, якщо 61.9 ≤ значення CILT < 63.9 0.25, якщо 63.9 ≤ значення CILT < 65.9
	54	0.1, якщо 56.9 ≤ значення CILT < 58.9 0.5, якщо 58.9 ≤ значення CILT < 61.9 1, якщо 61.9 ≤ значення CILT < 63.9 0.5, якщо 63.9 ≤ значення CILT < 65.9 0.25, якщо 65.9 ≤ значення CILT < 67.9
	55	0.1, якщо 58.9 ≤ значення CILT < 61.9 0.5, якщо 61.9 ≤ значення CILT < 63.9 1, якщо 63.9 ≤ значення CILT < 65.9 0.5, якщо 65.9 ≤ значення CILT < 67.9 0.25, якщо 67.9 ≤ значення CILT < 69.9
	56	0.1, якщо 61.9 ≤ значення CILT < 63.9 0.5, якщо 63.9 ≤ значення CILT < 65.9 1, якщо 65.9 ≤ значення CILT < 67.9 0.5, якщо 67.9 ≤ значення CILT < 69.9 0.25, якщо 69.9 ≤ значення CILT < 71.9
	57	0.1, якщо 63.9 ≤ значення CILT < 65.9 0.5, якщо 65.9 ≤ значення CILT < 67.9 1, якщо 67.9 ≤ значення CILT < 69.9 0.5, якщо 69.9 ≤ значення CILT < 71.9 0.25, якщо 71.9 ≤ значення CILT < 73.9
	58	0.1, якщо 65.9 ≤ значення CILT < 67.9 0.5, якщо 67.9 ≤ значення CILT < 69.9 1, якщо 69.9 ≤ значення CILT < 71.9 0.5, якщо 71.9 ≤ значення CILT < 73.9 0.25, якщо 73.9 ≤ значення CILT < 74.9
	59	0.1, якщо 67.9 ≤ значення CILT < 69.9 0.5, якщо 69.9 ≤ значення CILT < 71.9 1, якщо 71.9 ≤ значення CILT < 73.9 0.5, якщо 73.9 ≤ значення CILT < 74.9 0.25, якщо значення CILT ≥ 74.9
	60	0.1, якщо 69.9 ≤ значення CILT < 71.9 0.5, якщо 71.9 ≤ значення CILT < 73.9 1, якщо 73.9 ≤ значення CILT < 74.9

Назва змінної	Індекс	Значення
		0.5, якщо значення CILT ≥ 74.9
	61	0.1, якщо 71.9 ≤ значення CILT < 73.9 0.5, якщо 73.9 ≤ значення CILT < 74.9 1, якщо значення CILT ≥ 74.9
Значення CLIB	62	1, якщо значення CLIB < -12 0.5, якщо -12 ≤ значення CLIB < 7 0.25, якщо 7 ≤ значення CLIB < 20
	63	0.5, якщо значення CLIB < -12 1, якщо 7 ≤ значення CLIB < 20 0.5, якщо 20 ≤ значення CLIB < 35 0.25, якщо 35 ≤ значення CLIB < 48
	64	0.1, якщо значення CLIB < -12 0.5, якщо -12 ≤ значення CLIB < 7 1, якщо 7 ≤ значення CLIB < 20 0.5, якщо 20 ≤ значення CLIB < 35 0.25, якщо 35 ≤ значення CLIB < 48
	65	0.1, якщо -12 ≤ значення CLIB < 7 0.5, якщо 7 ≤ значення CLIB < 20 1, якщо 20 ≤ значення CLIB < 35 0.5, якщо 35 ≤ значення CLIB < 48 0.25, якщо 48 ≤ значення CLIB < 61
	66	0.1, якщо 7 ≤ значення CLIB < 20 0.5, якщо 20 ≤ значення CLIB < 35 1, якщо 35 ≤ значення CLIB < 48 0.5, якщо 48 ≤ значення CLIB < 61 0.25, якщо 61 ≤ значення CLIB < 74
	67	0.1, якщо 20 ≤ значення CLIB < 35 0.5, якщо 35 ≤ значення CLIB < 48 1, якщо 48 ≤ значення CLIB < 61 0.5, якщо 61 ≤ значення CLIB < 74 0.25, якщо 74 ≤ значення CLIB < 69.9
	68	0.1, якщо 48 ≤ значення CLIB < 61 0.5, якщо 61 ≤ значення CLIB < 74 1, якщо 74 ≤ значення CLIB < 69.9 0.5, якщо значення CLIB ≥ 74
	69	0.1, якщо 48 ≤ значення CLIB < 61 0.5, якщо 61 ≤ значення CLIB < 74 1, якщо значення CLIB ≥ 74

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “ Реалізація інтелектуальної системи підбору стеку читача ”

Обсяг пояснювальної записки: 78 аркушів.

Дата закінчення роботи: 10 червня 2025 р.

Підпис студента _____