Лабораторна робота №1

Побудова базового RAG-пайплайну. Векторизація та семантичний пошук

Мета роботи: опанування процесом перетворення неструктурованого тексту у векторні представлення (embeddings) та реалізація механізму знаходження інформації за змістом, а не за ключовими словами.

Стек технологій:

Python / Pandas *для обробки структурованих даних.*

LangChain *- фреймворк для побудови ланцюжків ШІ.*

ChromaDB *- векторна база даних (зберігає дані локально) з підтримкою індексації HNSW.*

Sentence-Transformers *нейромережева модель для генерації щільних векторів (створення ембедингів - перетворення тексту в цифрові вектори).*

Зміст роботи

Завдання 1. ***Дослідити ефективність застосування векторних ембедингів для аналізу текстового контенту шляхом створення інтелектуального сховища даних та проведення його дескриптивного і семантичного оцінювання.*** *Налаштувати середовища (LangChain, ChromaDB). Завантаження текстового датасету (напр. Netflix), створення ембедингів та реалізація найпростішого пошуку Top-K.*

Етапи роботи:

1. *Підготовка даних.* Імпорт та первинне опрацювання датасету .

2. *Статистичне обґрунтування*

– Розрахунок дескриптивних статистик для числових атрибутів.

– Інтерпретація ключових метрик: обсяг вибірки, заходи центральної тенденції та діапазони значень (min/max).

3*. Розгортання векторного сховища*

– Інтеграція бібліотеки ChromaDB у програмне середовище.

– Екстракція семантичного ядра (колонка description) для подальшої обробки.

– Ініціалізація бази даних та наповнення векторного простору текстовими ембедингами.

– Валідація цілісності бази шляхом порівняння кількості завантажених об'єктів.

4. *Оцінка ефективності.* Верифікація системи шляхом виконання серії контрольних семантичних запитів.

Завдання 2. ***Провести дослідження багатомовного семантичного аналізу.*** *Замінити модель all-MiniLM-L6-v2 на мультимовну (наприклад, paraphrase-multilingual-MiniLM-L12-v2). Провести експеримент: ввести запит українською мовою до англомовного датасету Netflix та проаналізувати релевантність результатів.*

Завдання 3. ***Порівняння метрик відстані.*** *Створити три різні колекції в ChromaDB з однаковими даними, але різними метриками схожості: l2 (евклідова), cosine (косинусна) та ip (внутрішній добуток). Порівняти Top-5 результатів для абстрактного запиту та обґрунтувати різницю в ранжуванні.*

Завдання 4. ***Динамічне оновлення та видалення об'єктів у векторній БД.*** *Розробити сценарій, у якому після ініціалізації бази необхідно:*

- видалити всі фільми певного режисера;

- оновити опис одного з фільмів;

- додати 5 нових записів.

Перевірити, чи змінюються результати пошуку після цих маніпуляцій.

Завдання 5. ***Опрацювання аномалій та семантичного шуму.*** *Додати до датасету 10 випадкових «сміттєвих» текстів (наприклад, рецепти страв або технічні інструкції). Дослідити, чи потрапляє це сміття в Top-K при абстрактних запитах.*

Завдання 6. ***Стрес-тестування продуктивності.*** *Штучно збільшити кількість документів у базі до 20 000 (50 000) (дублюванням або додаванням іншого датасету). Виміряти затримку пошуку. Побудувати графік залежності часу пошуку від кількості записів у базі.*

Методичні рекомендації

Загальна схема семантичного пошуку та аналізу даних з використанням ChromaDB та Pandas наведена на рисунку 1.

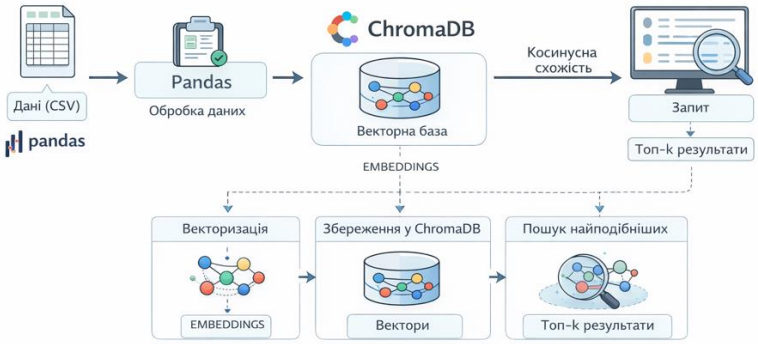

Рисунок 1. - Загальна схема семантичного пошуку

У прикладі використано датасет *netflix_titles.csv*.

В роботі використно *LangChain* як платформу для системної інтеграції *LLM* із зовнішніми базами знань. Фреймворк характеризується розвиненим рівнем абстракції, що забезпечує автоматизацію багатоетапних процедур, підтримку парадигми *Retrieval-Augmented Generation (RAG)* та створення саморегулівних агентів із розвиненою системою управління оперативною пам'яттю.

Налаштування середовища

```
! pip install langchain-community langchain-core chromadb sentence-
transformers
```

Підключення бібліотек

```
import pandas as pd
import os
import time
from langchain_community.vectorstores import Chroma
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_core.documents import Document
```

Підготовка та статистичний скринінг. Завантаження датасету та огляд його статистичних характеристик.

```
try:
    df = pd.read_csv('netflix_titles.csv')
except FileNotFoundError:
    print("Помилка: Файл netflix_titles.csv не знайдено.")

df = df.dropna(subset=['description']).drop_duplicates(subset=['title',
'description']).reset_index(drop=True)

print(df.describe(include='all'))
```

Створення об'єктів *Document* для *LangChain*

```
docs = []
for _, row in df.iterrows():
    # Формуємо метадані для подальшої фільтрації (Metadata Filtering)
    metadata = {
        "title": str(row['title']),
        "year": int(row['release_year']),
        "type": str(row.get('type', 'Unknown'))
    }
    docs.append(Document(page_content=row['description'],
metadata=metadata))

print(f"Дані очищено. Підготовлено {len(docs)} унікальних документів.")
```

Ініціалізація векторного сховища. Використовуємо спеціалізовану модель *Sentence-Transformers*, яка точніша за стандартну *DefaultEmbeddingFunction*. *all-MiniLM-L6-v2* - це одна з найпопулярніших та найефективніших моделей для генерації ембедингів (векторних уявлень тексту), що орієнтована на високу швидкість роботи при збереженні відмінної якості семантичного пошуку.

```
model_name = "sentence-transformers/all-MiniLM-L6-v2"
embeddings = HuggingFaceEmbeddings(model_name=model_name)

# PersistentClient дозволяє зберігати базу на диску
persist_directory = "./netflix_db_langchain"

# Створюємо або завантажуємо векторне сховище
vectorstore = Chroma.from_documents(
    documents=docs,
    embedding=embeddings,
    persist_directory=persist_directory,
    collection_metadata={"hnsw:space": "cosine"}
)
```

У більшості векторних баз даних, зокрема і у *ChromaDB*, за замовчуванням може бути встановлена евклідова відстань (*L2*). Для задач семантичного пошуку критично важливо явно задати косинусну схожість, оскільки вона нівелює вплив довжини тексту на результат пошуку.

Семантичний запит та аналіз результату. Виконуємо запит та аналізуємо відстані (distances). Чим менша відстань, тим вища релевантність**.**

```
query = "Dramatic story about high school friendship and secrets"
```

Пошук з поверненням оцінки схожості (*similarity scores*), Top-K пошук (k=3)

```
results = vectorstore.similarity_search_with_score(query, k=3)
```

```
print(f"{'#':<3} | {'Назва':<25} | {'Схожість':<10} | {'Рік'}")

for i, (doc, score) in enumerate(results, 1):
```

У *Chroma score* - це дистанція (*distance*). *Similarity = 1 - distance*

```
    similarity = round(1 - score, 4)
    title = doc.metadata.get('title', 'N/A')
    year = doc.metadata.get('year', 'N/A')

    print(f"{i:<3} | {title[:25]:<25} | {similarity:<10} | {year}")
    print(f"    Опис: {doc.page_content[:120]}...")
```

Додатково робимо фільтрацію за метаданими, що є гібридним підходом: семантика + структура

```
print("\nФільтрований пошук (тільки фільми після 2015 року)")
filtered_results = vectorstore.similarity_search_with_score(
    query,
    k=2,
    filter={"year": {"$gt": 2015}}
)

for doc, score in filtered_results:
    print(f"Знайдено (після 2015): {doc.metadata['title']}
({doc.metadata['year']})")
```

Результат пошуку, що стосується драматичних історій про шкільну дружбу та таємниці:

```
#   | Назва                     | Схожість   | Рік
------------------------------------------------------------
1   | Trinkets                  | 0.6121     | 2020
    Опис: A grieving teen finds an unexpected connection with two classmates at her new high school after they all land in the sam...
------------------------------------------------------------
2   | Pyaar Tune Kya Kiya       | 0.6022     | 2014
    Опис: From teen lovers defying social and religious taboos to a student falling for her teacher, these stories explore joy and...
------------------------------------------------------------
3   | My True Friend            | 0.5959     | 2012
    Опис: A bullied student forms a bond with the leader of a youth gang and discovers the power and meaning of true friendship....
```

Результат після використання фільтру (фільми що вийшли після 2015 року):

```
Знайдено (після 2015): Trinkets (2020)
Знайдено (після 2015): Detention (2020)
```

Обґрунтування обраних методів реалізації:

Статистичний аналіз. Використання методу df.describe() забезпечує первинне розуміння структури датасету, зокрема розподілу категоріальних даних (співвідношення фільмів та серіалів).

Семантична векторизація. Застосування ChromaDB дозволяє автоматично трансформувати текстові описи у багатовимірні вектори, відображаючи їхній зміст у математичному просторі.

Векторний пошук. Механізм запитів базується на обчисленні косинусної схожості.

Система ідентифікує найбільш релевантні фрагменти тексту, знаходячи вектори, що мають мінімальну відстань до вектора запиту користувача.

Контрольні запитання

1. Які задачі можна вирішувати за допомогою ChromaDB?
2. Як впливає наявність `NaN` значень у полі `description` на процес векторизації?
3. Поясніть різницю між пошуком за ключовим словом (SQLLike) та семантичним пошуком (ChromaDB Query).
4. Чому важливо використовувати `PersistentClient` замість звичайного клієнта в ChromaDB?
5. Як змінюється затримка (latency) пошуку при збільшенні обсягу даних з 8 тисяч до 1 мільйона записів?
6. Що таке Semantic Drift (семантичний зсув) і як він може вплинути на результати пошуку в медіа-бібліотеках?
7. Поясніть концепцію In-context Learning у контексті RAG-систем (Retrieval-Augmented Generation).
8. Як впливає нормалізація векторів на обчислення косинусної схожості?
9. Поясніть принцип роботи алгоритму HNSW у ChromaDB. Чому він швидший за повний перебір векторів?
10. Як розмірність вектора (наприклад, 384 проти 768) впливає на точність пошуку та обсяг споживаної пам'яті?

Самостійна робота до лабораторної роботи №1

Завдання: Дослідити вплив архітектури моделі та розмірності вектора на точність та швидкість RAG.

1. Порівняйте моделі *sentence-transformers/all-MiniLM-L6-v2* (розмірність 384) з *BAAI/bge-small-en-v1.5* (розмірність 384) або *sentence-transformers/all-mpnet-base-v2* (розмірність 768)

2. Методика дослідження:

- Створіть окремі колекції ChromaDB для кожної моделі.
- Виміряйте час індексації (створення бази) для 1000 документів.
- Виміряйте середній час пошуку на 50 випадкових запитах.
- Оцініть якість пошуку, чи змінився склад Top-3 результатів?

3. Дослідження розмірності вектора:

- Як впливає перехід з 384 на 768 вимірів на швидкість пошуку?
- Як змінюється об'єм пам'яті, що займає база даних?

4. Оформити результати досліджень у вигляді таблиці з полями модель, розмірність, час індексації, Latency (ms), об'єм БД (MB).

Зробити висновки.