Research Paper Recommender System

Hsin-Chun Yin

Abstract

This paper details the development of a recommender system for suggesting research papers to authors. By employing OpenAI's embeddings and utilizing SQLight databases, the system efficiently matches authors with relevant papers. Challenges in multi-threading and embedding methods were explored, with Semantic Scholar's approach serving as a comparison. The work concludes with a forward-looking discussion on improvements in recommendation strategies, providing a foundation for further research in the field of recommender systems.

1. Introduction

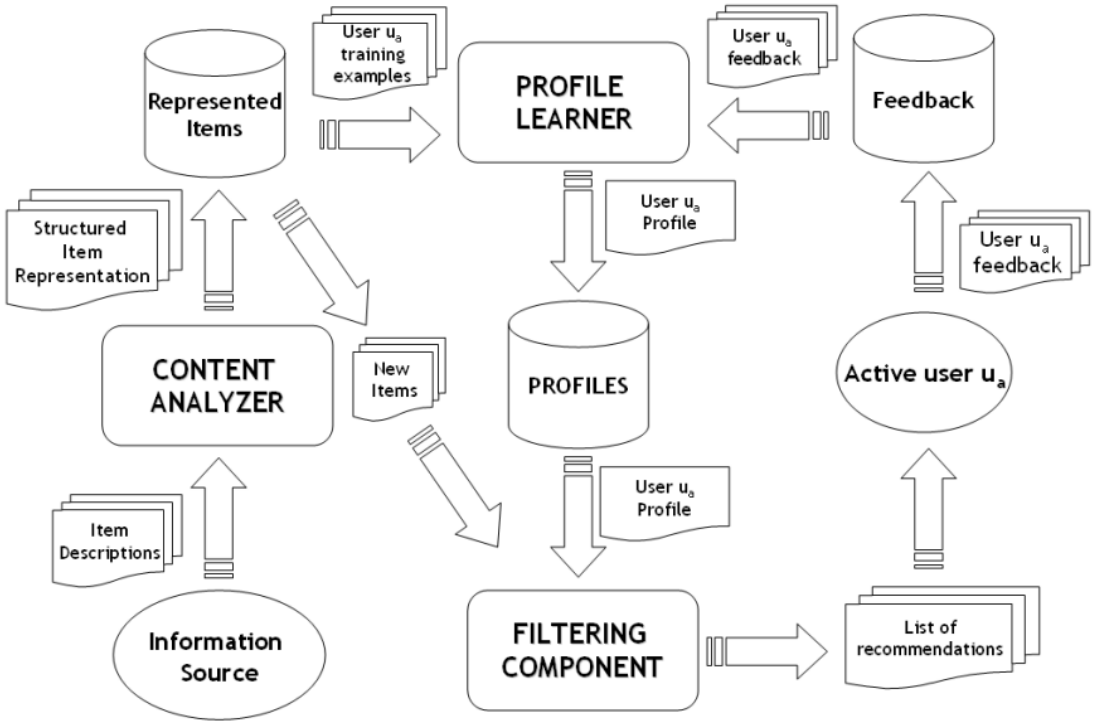
Recommender systems have become an indispensable tool in various fields, assisting users in discovering content aligned with their interests and needs. In the academic domain, these systems help researchers navigate the ever-growing body of scholarly literature.

2. Project Description

The project is a recommender system designed to suggest scholarly papers to researchers. Within this system, each user is treated as an author that writes or references research papers. These authors, who are the default users for the system, receive research paper recommendations based on their writing and referencing patterns. If an author likes a particular recommendation, the system logs this information, dynamically adapting future suggestion feeds to better align with the author's interests.

The structure of the recommender system is depicted in the image below[1]. Upon initialization, the program imports paper files downloaded from Semantic Scholars, representing each paper in a tabular format , and then storing it in the content database. Subsequently, it utilizes these papers to extract author information, in order to form the default authors in the profile database. The system then uses OpenAI embeddings to assign an embedding vector to each paper within the content database. Then we represent the author by another embedding, that is determined by taking the average of the embeddings of all the papers that the author wrote, referenced, or liked.

[1] Recommender Systems Handbook, Editors: Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor, 2011, Publisher: Springer New York, NY



Once a user selects an author ID, the recommender system retrieves the corresponding author from the profile database and compares their embedding with the embeddings of all papers in the content database. It then recommends to the author the papers with the top 10 highest cosine similarity scores.

An alternative method of recommendation employed by the system takes into consideration not only the rank of cosine similarity but also factors such as the publication year, reference count, citation number, and influential citation count. By calculating the weighted average of these ranks, it generates a new ranking for the papers and recommends the top 10 highest-ranked papers to the user. Importantly, the weights assigned to each rank can be customized by the user, allowing for a more personalized recommendation process. The formula is given by:

$$R_{final} \equiv W_{cos} \times Rank_{cos} + W_{ref} \times Rank_{ref} + W_{cite} \times Rank_{cite} + W_{year} \times Rank_{year} + W_{influential} \times Rank_{influential}$$

When the user reviews the recommendations, they have the option to select the papers they like, at which point the system updates the user's liked papers in the profile database. Should the user modify the set of papers they are associated with, the user's embedding will change accordingly, and consequently, so will the recommendations tailored for that user. To enhance efficiency, the author's embedding is calculated during runtime, ensuring a responsive and adaptive recommendation process.

A comprehensive visual depiction of the user interface can be found in a youtube video[2].

Technical details

The system concatenates the title and abstract of a paper and inputs it into the OpenAI embedding interface. Should the text prove too long for OpenAI to embed—an issue that has not arisen previously—I employ a sliding window technique to calculate the embedding for each section and then take their average. This approach was inspired by the paper[3] "Increasing Textual Context Size Boosts Medical Image-Text Matching". To enhance efficiency, I utilize python library “joblib” to cache the function calls of the OpenAI embedding process.

Currently, the database of the recommender system is restricted to papers on the topics of machine learning. However, the system is designed to allow for easy addition of papers on other subjects.

I have selected SQLite as my database of choice, with the majority of the work being handled through database queries. Python functions can be encoded into database functions using SQLite's features, further streamlining the process.

The project is highly scalable, capable of accommodating more than 10,000 papers with ease. Should expansion become necessary, transitioning to a larger database such as PostgreSQL would be feasible without significant modifications to the existing SQL code.

Failed attempts:

Paper embedding

Many of my early challenges stemmed from attempts to embed entire papers or find suitable ways to represent the data. Initially, I considered embedding the whole paper and even thought about dividing it into sections such as the title, introduction, body, and references. However, embedding the entire paper proved prohibitively expensive, costing roughly $0.1 per paper. By contrast, embedding only the title and abstract significantly reduced the cost to approximately $0.4 for 10,000 papers.

I also experimented with embedding papers using Specter embedding, which is the proprietary embedding method of Semantic Scholar. To this end, I downloaded Semantic Scholar's embedding for each paper. However, during searches in which I embedded keywords and compared the cosine similarity with the paper's embedding, the performance was subpar. The

[2] https://www.youtube.com/watch?v=4E4XfvXzOOo&ab_channel=Hsin-ChuYin
[3] Increasing Textual Context Size Boosts Medical Image-Text Matching, authors: Idan Glassberg, Tom Hope, arXiv:2303.13340 [cs.LG], Thu, 23 Mar 2023 15:20:05 UTC

average cosine similarities were around 10%, in stark contrast to OpenAI's 90%. One possible explanation for this discrepancy could be that Semantic Scholar embeds different parts of the paper than I did, possibly even including the entire paper's content. To rectify this issue, I would need to embed each paper's title and abstract using Specter embedding. Since Specter runs on transformers, this operation would be quite costly to execute on aquarium computers. A potential path forward could involve running the program on the school's computer cluster. This comparison between Specter and OpenAI's embedding methods could be an intriguing area for future research.

Paper Parser

Given that the system now focuses solely on embedding the title and abstract, the need to separate the content of the paper has been eliminated. Nevertheless, I did explore various methods for dividing the paper, including GPT-2, Grobid[4], and Cermine[5]. GPT-2, while innovative, was found to be imprecise and time-consuming, rendering it unsuitable for processing large numbers of papers. Grobid's age and updated dependencies have rendered it unusable. However, Cermine remains a viable option, capable of processing papers quickly and efficiently.

Database system

At first, I relied on files as my database, loading all the papers into pandas for processing. On an aquarium computer, pandas could handle approximately 30,000 papers in RAM at a time, and its processing speed outpaced that of an actual SQLite database. However, considering the future scalability of the project and the potential efficiency challenges as the number of papers increases, I transitioned to a database design.

I explored the use of a PostgreSQL database, but it could not be initialized on the school's computers. SQLite emerged as a suitable choice, primarily because it can save the database file under file systems, providing a balance between functionality and compatibility.

User interface

I employed Streamlit for the user interface, taking advantage of its multi-threading capabilities to enhance the refresh rate of the interface. However, this presented a challenge with SQLite, as it does not support multi-threading access and would become locked if two programs attempted to query the database simultaneously. One solution to this problem was to strategically sequence the database queries, prioritizing quicker ones so they would likely finish before the more time-consuming queries. An alternative approach involved using a try-and-except construct to catch any errors and reattempt the query until successful, ensuring a smooth user experience despite the technical constraints.

---

[4] Grobid: https://github.com/kermitt2/grobid
[5] Cermine: https://github.com/CeON/CERMINE

Another challenge with Streamlit is that it refreshes the entire page after every action on the screen. So, if there's a demanding query from the database – such as a request for recommendations – the recommendation sorting algorithm would run after each click on any button. The solution to this problem was to implement caching for the results of recommendations. By doing so, if nothing changes in the search box, the cached results are displayed, thereby optimizing the efficiency of the system and providing a more responsive user experience.

Future work

The roadmap for future work consists of two main areas: enhancing the existing content-based recommendation system and exploring a different recommendation model known as the collaborative-based recommendation system.

Content-Based Recommender System Improvements

Under the existing framework, there are promising opportunities for enhancing data representation. This could be achieved through the exploration of alternate embedding methods, or by parsing and embedding different sections of a paper. Such enhancements would allow for more nuanced comparisons, adding depth and sophistication to the analysis.

In the realm of recommending methods, a particularly promising strategy is the implementation of contrastive learning[6]. This approach, utilized by Semantic Scholars, takes into consideration both the papers an author likes and dislikes when forming recommendations. This contrasts with the current system, which bases recommendations solely on the papers that the author has shown a preference for and excluding the ones he has disliked, thereby potentially broadening the scope and relevance of the suggested readings.

Collaborative-Based Recommender System

Proven to be an effective method of recommendation, Collaborative-Based Recommender System offers several promising directions to explore. In addition, Applying Collaborative Recommender system to research papers is unprecedented, so it can be a direction of innovation. Some common collaborative approaches are listed in the handbook[7].

This system could incorporate techniques such as user clustering to increase efficiency, alleviate problems of scalability in recommender system[8] and offer personalized recommendations.

---

[6] https://www.semanticscholar.org/faq#fine-tune-research-feeds
[7] Recommender Systems Handbook, Editors: Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor, 2011, Publisher: Springer New York, NY
[8] Singh, M. Scalability and sparsity issues in recommender datasets: a survey. Knowl Inf Syst 62, 1–43 (2020). https://doi.org/10.1007/s10115-018-1254-2

Neural Networks and hybrid recommender systems

Although I will not delve into neural network-based recommendation systems, such as those seen in open-source Twitter code[9], due to a lack of research on the subject, this area offers fertile ground for further investigation.

More on Recommender system

Evaluation of the Recommender System[10]

While the current project has achieved high cosine similarity and seemingly accurate recommendations, no formal evaluation has been conducted. The assessment of recommender systems remains an open question in the field, though some attempts have been made to develop evaluation techniques.

Fairness in Recommender Systems[11][12][13]

The subject of fairness in recommender systems presents another compelling direction for exploration, with potential implications for ethical design and bias mitigation.

Decentralized Recommender System[14]

A decentralized recommender system offers personalized recommendations without relying on a central server. It addresses privacy concerns by keeping user data locally and employs techniques like federated learning. This approach maintains data security while still generating personalized suggestions based on collective knowledge. The field is rapidly evolving, exploring architectures and algorithms for efficient collaboration among devices while preserving privacy.

---

[9] Twitter recommender system: https://github.com/twitter/the-algorithm
[10] Evaluating Recommender Systems: Survey and Framework, Authors: Eva Zangerle, Christine Bauer ACM Computing SurveysVolume 55 Issue 8 Article No.: 170pp 1–38 https://doi.org/10.1145/3556536
[11] Fairness in Recommendation system: A Survey (2022) https://arxiv.org/pdf/2205.13619.pdf
[12] A Survey on the Fairness of Recommender Systems (2022) https://www.semanticscholar.org/paper/A-Survey-on-the-Fairness-of-Recommender-Systems-Wang-Ma/63a9d47f42d1d1e8bd8fb1183b88ca82297c37bb
[13] A Survey on Fairness-aware Recommender Systems https://www.semanticscholar.org/paper/A-Survey-on-Fairness-aware-Recommender-Systems-Jin-Wang/47d819f79d44d9cb5145b92e438f7b2a773451e8
[14]https://www.semanticscholar.org/paper/An-Efficient-Blockchain-Based-Privacy-Preserving-Casino-Patsakis/ea3a176672ec55c88c6d613a565fef9870da171a

Conclusion

In summary, the future work related to this project encompasses a wide range of possibilities, each with the potential to advance the field of recommender systems. By exploring new techniques and critically assessing current practices, significant strides can be made toward creating more effective, fair, and transparent systems for recommending academic papers.