Hidden Gems in PIR

Course Project Proposal

509496 - Information Retrieval and Recommender Systems

Professors: Pasi G., Peikos G., Kasela P.

Brasca Federica - 513065
Pan Andrea - 514555
Zappietro Alessandro - 516169

Bachelor of Science in Artificial Intelligence

Abstract

The field of **Information Retrieval (IR)** has increasingly emphasized **personalization**, aiming to tailor search results to the unique user preferences and behaviors.

In this work, we develop an **Information Retrieval System (IRS)** that extends beyond conventional relevance-based retrieval by incorporating deeper personalization techniques. We introduce the concept of **Hidden Gems**—documents underexposed in traditional rankings but highly valued by niche audiences, identified through **high favorites-to-view ratios**. Additionally, by constructing **user profiles** from previously asked questions, our system enables **personalized recommendations**, suggesting related questions posed by other users.

Through rigorous evaluation using **precision, nDCG and MAP**, we assess our system's effectiveness in balancing **retrieval accuracy and content diversity**, ultimately enhancing the **user search experience**.

Introduction

Information Retrieval systems have evolved to prioritize **personalization**, addressing the need for tailored search experiences that reflect diverse user preferences. Building on previous research in community Question Answering (cQA), we explored methods to identify and highlight valuable but underrepresented content. Specifically, our study defined "Hidden Gems" as those documents with disproportionately high user appreciation (i.e., favorites) relative to their view counts. **Such content often goes unnoticed in conventional ranking systems but holds significant potential to enrich user satisfaction and engagement, while expanding content discovery**. By analyzing **user interaction patterns** and **document characteristics**, our approach aims to improve both **retrieval relevance and content diversity**, advancing personalization in IR systems.

Dataset

The dataset used in this study is a subset of the **SE-PQA (StackExchange - Personalized Question Answering) dataset**. SE-PQA is a curated resource derived from the publicly available StackExchange network, covering a broad range of topics across multiple communities. This dataset includes textual questions, answers and metadata capturing user interactions, such as scores, favorites and tags assigned to questions.

Main Discoveries from Statistics

The reduced dataset we used spans a decade (2009–2019) and includes 10,000 documents with an average length of 661 characters. **Contributions came from 93 users generating 37,782 questions and 35,714 answers**.

User engagement shows 71.9 million total views (**7,186 views/document**) and 8,997 favorites (**0.9/document**). **Popularity is concentrated**: 2,227 documents exceeded 5,000 views, while

6,450 have no favorites. **Scores average is 7.88**, with a moderate correlation (r = 0.66) between scores and favorites.
The top document metrics include 833,389 views, 170 favorites, and a score of 310, while outliers include negative scores (229 documents) and documents with as few as 23 views.
This dataset illustrates high variability in user interactions and content impact.

Task Definition

Our objective is to develop a search engine that **retrieves relevant answers from a cQA dataset** while integrating **personalization techniques**.
This **personalized search engine** leverages the dataset outlined in the previous section and combines traditional retrieval methods (**BM25**) with neural re-ranking (through a **bi-encoder**).

Key features:

- **Query expansion** using **Large Language Models (LLMs)**, improving retrieval breadth
- **Personalization** through **user historical data** and **contextual features**

Data preprocessing

Feature engineering is the process of transforming raw data into formats that improve usability, including techniques to ensure **data consistency and reliability**, handle missing values and standardize content. In this work, we applied the following feature engineering steps:

- **Text Preprocessing**: converted text to **lowercase**, removed **hyperlinks, HTML tags, emojis, symbols and extra whitespaces**, and applied **stopword removal** and **lemmatization**.
- **Numerical Data Processing**:
  - Missing values in the **favorites** column were replaced with **zero** and converted to **integers**.
  - The **timestamp** column was reformatted to retain only **dates**, while **rel_timestamps** was reduced to the **most recent entry**, assuming it represented the latest relevant update.
- **Data Cleaning**: **duplicate entries** were removed and **questions without valid answers** were filtered out, ensuring a clean and structured dataset for downstream tasks.

The Retrieval Model

We developed a `RetrieverModel` class to handle document retrieval and recommendation tasks efficiently. This model integrates three key components: a BM25 retriever, a bi-encoder and a LLM model able to generate given an instruction, T5-small.
The model begins by indexing all answer documents using the `indexing` method, which constructs the term-document frequency (TDF) matrix and an inverted index. Additionally, answer embeddings are precomputed for GPU-accelerated search, utilizing a dictionary that maps each document ID to its corresponding embedding index.

Document Retrieval

Document retrieval is performed through two primary methods, depending on the task: `search_answer` and `search_answer_batch`, which follow the same core process but differ in optimization and usage. Both methods include the following steps:

1. **Text preprocessing** for query cleaning and keywords extraction.
   → Possible **LLM query expansion**.
2. **BM25 Initial Retrieval** to retrieve up to 1000 documents (traditional ranking).
3. **Bi-encoder re-ranking** to get the top 100 relevant documents, ordered by rank.

The main difference between the methods lies in their optimization and number of features:

- `search_answer`: Designed for single-query retrieval, offering personalized recommendations based on user profile.
- `search_answer_batch`: Optimized for handling large batches of queries, providing retrieval efficiency for mass query processing. However, it does not incorporate personalized recommendations.

Further Reading Recommendation

Following the first query, personalized recommendations are possible if the user has a profile in the database (i.e., the user has previously asked questions). The `search_answer` method allows a `user` parameter to be passed, which triggers the extraction of a dictionary of **interest tags** from the user's past questions. These tags are stored in the format `(tag, count)` and represent the user's preferences.

To generate recommendations, the system performs the following steps:

1. **Tag Filtering**: The tags from the user's initial query are added to the existing interest tags. Documents are then filtered based on whether they share at least one tag from the union of the user's existing tags and the query's tag.
2. **Probabilistic Weighting**: Documents are then recommended based on the **tag counts** in the interest tags. For example, a document tagged with "food" with an associated user count of 5 will have five times more chances of being recommended compared to a document tagged with "reading" and a count of 1, assuming both tags are present in the user's interest tags.
3. **Hidden Gems**: If available, the recommendation also returns (using once again the method described above) a **Hidden Gem**, a document that has high user appreciation (favorites) relative to its views. A "Hidden Gem" is defined using a formula that balances **favorites** and **views**:

$$favorites \geq a^{x \cdot views - b} + c$$

where:

- a: Defines how quickly views impact the favorites count.
- b: Sets the tolerance region before a document starts being considered a hidden gem.
- c: The minimum number of favorites required to be considered a hidden gem.
- x: The scaling factor that determines how much views affect the growth of favorites.

This probabilistic approach ensures that both popular content and valuable underexposed content (hidden gems) are effectively recommended to users.

LLM query expansion

We utilize the **T5-small** model provided by Google, chosen for its ability to generate text based on given instructions, allowing a **dynamical query expansion**.
The expansion process is activated by setting the `expand` parameter to `True` in either of the search methods. When enabled, the following steps occur:

1. The original query is passed to T5-small with the instruction:
   "Expand search queries for:" followed by the user's query.
2. The LLM generates a response, typically starting with something like:
   "Sure, here's the expanded query: ..."
3. To clean the result, the text before the colon (":") is removed, leaving only the expanded query.

This expansion process is handled by the method `expand_query`, which returns the **expanded and cleaned query** ready for use in the retrieval process.
By leveraging query expansion in this way, we ensure that the search process benefits from a broader range of terms, enhancing the retrieval of relevant documents.

Evaluation

To evaluate the model's performance, we use the `evaluate` method, which makes predictions based on a **qrels** dictionary that maps question IDs to a single answer ID, considered the most relevant answer for each query. Each question's text is retrieved either from a provided pandas dataframe or, if not available, from the larger training dataset.

The evaluation metrics include **Precision@k**, **nDCG@k** and **MAP**, calculated at various **k-values**.

Due to the need for batch and parallel processing to ensure reasonable evaluation times, using **LLM query expansion** can increase processing time significantly even with a CUDA-capable GPU. Therefore, for evaluating expanded queries, it is recommended to use a limited set of queries (between 10 and 100) across three different **k-values**.

Performance Results

On the train dataset of 9216 entries, and the test dataset of 98 entries, our model obtained the following **scores**:

Train - 9216	**Time: 4m55s**	**Test - 98**	**Time: 3.8s**	**Test (LLM) - 98**	**Time: 3m48s**
Precision@1: 0.6979	nDCG@1: 0.6979	Precision@1: 0.7755	nDCG@1: 0.7755	Precision@1: 0.7449	nDCG@1: 0.7449
Precision@5: 0.8577	nDCG@5: 0.7857	Precision@5: 0.9082	nDCG@5: 0.8473	Precision@5: 0.9082	nDCG@5: 0.8346
Precision@10: 0.8954	nDCG@10: 0.7979	Precision@10: 0.9490	nDCG@10: 0.8606	Precision@10: 0.9388	nDCG@10: 0.8456
MAP: 0.8170	31 queries/s	MAP: 0.8776	25 queries/s	MAP: 0.8639	0.43 queries/s

Conclusion and Future Works

In conclusion, our personalized Information Retrieval System demonstrates strong performance in terms of precision, nDCG and user satisfaction. However, there are several areas for improvement. The limited success of query expansion, likely due to model constraints or query length limitations, suggests the need for more advanced language models or optimized preprocessing techniques. For this dataset, **query expansion with T5-small may have introduced irrelevant variations, rather than improving the quality of the results.**

Looking ahead, we recommend exploring larger or more efficient models for query expansion, refining the weighting mechanism for query history to balance diversity and relevance, and investing in computational resources to fully harness the potential of user data (SE-PQA's users.csv). Additional areas for future work could include evaluating document relevance based on more factors, such as **recency** (considering the publication dates), as well as scores and views (to assess **popularity**).

Another direction is the addition of a community column that extracts the community from the `question_id` in the dataset. This would enable further analysis by filtering based on individual communities, providing deeper insights into user behaviors and preferences.

Finally, to evaluate user satisfaction, a feedback retriever could be implemented. For a given query, once relevant documents are retrieved, users could provide feedback indicating whether the results are relevant or not. This feedback loop would allow the system to learn and refine its retrieval processes, improving subsequent results.

By addressing these challenges, the system can achieve even greater accuracy, diversity and user satisfaction, ultimately delivering a more effective and personalized information retrieval experience.

Roles and Personal Contributions

The project was carried out collaboratively, with each member contributing by leveraging their unique strengths.

1. Federica Brasca: Report and Code Documentation

Focused on the major part of the report, ensuring clarity, structure and alignment with project goals. Additionally, contributed to the code with a specific focus on preprocessing tasks, improving the data pipeline. Enhanced the readability of the code by adding detailed docstrings, comments and markdown sections to improve maintainability and understanding for future work.

2. Andrea Pan: Lead Developer

Led the majority of the code development, particularly the core retrieval and recommendation components. Contributed significantly to the implementation of the search and ranking mechanisms, integrating the BM25 retriever, bi-encoder ranking, LLM query expansion and recommendation techniques. Contributed in refining the report by providing insights into the technical aspects of the code

3. Alessandro Zappietro: Data Analyst

Responsible for the data analysis, including extracting insights from the dataset and generating various statistics and visualizations. This analysis helped inform the direction of the retrieval system and provided valuable feedback on the overall system performance.

This seamless collaboration enhanced our decision-making process and strengthened our ability to work as a cohesive and effective team.

References

Kasela, P., Pasi, G., & Perego, R. (2023). SE-PQA: Personalized Community Question Answering