Personalized Information Retrieval project - 2024/25

Armanni Luca 509085

Ghiotto Alessandro 513944

Introduction

The goal of this project is to build a search engine designed to retrieve relevant answers to user queries from a community Question Answering dataset. The focus is on exploring simple statistical and lightweight models, as well as incorporating neural reranking with BERT-like models, which generate contextualized representations of the input text. The final step involves personalizing the results by integrating user-specific features.

Dataset

The SE-PQA dataset comprises a collection of questions and their corresponding answers. The objective is to identify the best answer for a given question from the collection of documents. Each question includes a title and a body of text, along with additional metadata that enables personalization, such as the author, timestamp, tags, and associated community. Below, we provide some statistics on the word counts of the documents.

	mean	std	median	25%	75%	99%	min	max
Answer	234.28	257.09	159.0	85.0	289.0	1222.03	1	4412
Question title	10.30	4.09	10.0	7.0	13.0	22.0	1	31
Question text	118.25	111.64	89.0	57.0	143.0	520.0	6	2549

Evaluation measures

Before entering into the methods that we use, the most important task is to decide the measures we are interested in. This highly depends on the tasks and what you really want from you application, the ones we have chosen are the following:

- **P@1 and P@3:** critical for question answering systems, as the correct answer should appear at the top.
- **nDCG@3, nDCG@10, and MAP@100:** provide a general overview of ranking performance.
- **R@100:** high recall is essential for selecting a first-stage retriever before applying re-ranking.

Lastly we have also looked at the **Mean Response Time (mrt)**, which reports the average time (in milliseconds) to conduct a query.

Methodology

1. Baseline Retrieval

Our Baseline Retrieval employs two classical models, **BM25** and **TF-IDF.** A comprehensive grid search was conducted to identify the optimal model and configuration. This process explored several preprocessing options, including plain text, stemming, stopwords removal, and their combination, alongside two query types: full question text and question titles.

Separate indexes were created for each preprocessing configuration using PyTerrier. Once the optimal model and configuration were determined, BM25 was optimized further by adjusting its weighting parameters: c and k_1

We performed a grid search over these parameters to find two models: one that maximizes precision and one that maximizes recall. The first is used in this first part where BM25 is used as a standalone model, the second is used instead when BM25 is used as a first stage retrieval in a re-ranking pipeline.

2. Neural Re-ranking

To enhance ranking quality, the pipeline employs Neural Re-ranking using dense embeddings. We selected the **Bi-encoder** architecture for its efficiency and scalability. Unlike Cross-encoders, which evaluate each query-document pair at runtime, Bi-encoders enable precomputing document embeddings, making them suitable for large-scale datasets. We used the **all-MiniLM-L12-v2** model as Bi-Encoder model. This model is pre-trained on StackExchange pairs, which ensures relevance to the SE-PQA dataset without requiring additional fine-tuning.

Dense embeddings for the document collection were precomputed and stored using **FAISS**, a library designed for fast similarity search. The pipeline is the following: **BM25 % k >> BiEncoder**.

1. **Candidate Retrieval**: BM25 retrieves a set of candidate documents.
2. **Neural Re-ranking**: The Bi-encoder re-rank these candidates based on their cosine similarity to the query embedding.

Experiments tested various cutoff values for BM25, with k = 100 delivering the best performance.

Weighted Combination (BM25 + Bi-encoder)

To further improve the re-ranking pipeline, we integrated scores from BM25 and the Bi-encoder. The output score is computed with the following weighted average:

$$\text{final\_score} = \lambda \cdot \text{BM25\_score} + (1 - \lambda) \cdot \text{BiEncoder\_score}$$

A systematic search over λ values (ranging from 0 to 1 with step 0.1) identified the optimal balance.

3. Query Expansion: Personalizing User Queries with Tags

Personalization was introduced into the pipeline through Query Expansion, enriching user queries with contextual information derived from their historical behavior. **Tags** from users' past interactions served as the foundation for this process. For each query, we compiled a set of tags reflecting the user's cumulative interests, sourced from metadata associated with their previous questions.

To perform the expansion, we used the **Phi-3-mini-4k-instruct** model, a decoder-only LLM. The LLM receives the tags and the query, and outputs a more comprehensive and personalized version of the query.

The pipeline is the following: **BM25 % 100 >> expand_query >> BiEncoder**

4. Personalized Information Retrieval

To align retrieval results with user preferences, we incorporated a **TagsScore** feature into the pipeline. This score quantifies the overlap between two users' historical data. The two users we are interested in are the one who wrote the question and the one who wrote the answer. It is calculated as follows:

$$TagsScore(u_q, u_a, t) = \frac{len(intersection(Tags(u_q, t), Tags(u_a, t))}{len(Tags(u_q, t)) + 1}$$

Where $Tags(u, t)$ is the union of the sets of tags associated to all the questions before t, written by the user u. u_q is the user who wrote the question and u_a is the one who has written the answer. The tags for the question at time t are included for u_q, but excluded for u_a.

The TagsScore is combined with BM25 and Bi-encoder scores using a **weighted average**:

$$\text{final\_score} = \lambda_1 \cdot \text{BiEncoder\_score} + \lambda_2 \cdot \text{BM25\_score} + \lambda_3 \cdot \text{tags\_score}, \text{ with } \lambda_1 + \lambda_2 + \lambda_3 = 1$$

The inclusion of the tags score in the final output enables a personalized ranking mechanism tailored to a specific query and user.

We tried to compute the tag score in other ways. The first is to represent the tags as count vectors, instead of just sets, and compute the cosine similarity between them. This approach also takes into account the number of occurrences of the tags, not just which ones. The second is to use the community associated to a question, instead of the tags. Both methods gave good results, but not better than the original one.

Cold Start Problem

Most users have written only a single question, leading to an insignificant tags score in the majority of cases. To address this, we experimented with alternative computations for the tags score that consider its relevance only when a user has contributed a substantial **number of questions**. For instance, we tried multiplying the tags score by the number of questions written by the user, applying the tags score only when the number of questions exceeds a certain threshold, and using two distinct weighting schemes depending on the number of questions.

Learning to Rank (LTR)

The LTR phase aims to improve the personalization of the retrieval pipeline by training machine learning models to effectively combine multiple features into a single ranking score. The model receives the following scores as **input features**: BM25 Score, Bi-encoder Score and Tags Score. These features were normalized to ensure a more stable training of the model.

During the training phase, the models used **relevance labels** (*qrels*) as the target output. This setup allowed the models to learn patterns that differentiated relevant documents from irrelevant ones based on the input features.

We explored two machine learning models:

1. **Random Forest**: This ensemble learning method uses multiple decision trees to predict relevance scores. It is robust to overfitting and captures non-linear relationships between features.
2. **XGBoost**: A gradient boosting framework optimized for speed and performance. It iteratively improves the model by focusing on errors in previous predictions.

The output was a predicted relevance score, which was used to re-rank the documents retrieved by BM25. The pipeline is the following: **BM25 % 100 >> BM25 ** BiEncoder ** TagsScore >> LearnToRank_Model**

Results

The performance on the val set of the models on their respective best configuration, is summarized in the following table.

n	name	P@1	P@3	nDCG@3	nDCG@10	R@100	AP@100	mrt
0	BM25	0.765	0.282	0.815	0.845	0.959	0.819	13
1	Bi-Encoder	0.908	0.310	0.921	0.938	**0.969**	0.927	23
2	Bi-Encoder + BM25	0.918	0.316	0.935	0.943	**0.969**	0.934	29
3	Query Expansion >> Bi-Encoder	0.857	0.310	0.898	0.902	**0.969**	0.891	12740
4	Bi-Encoder + BM25 + TagsScore	0.918	**0.320**	**0.943**	0.946	**0.969**	0.938	47
5	WA_H(Bi-Encoder, BM25, TagsScore)	**0.929**	0.316	0.940	**0.948**	**0.969**	0.**941**	63
6	RF(Bi-Encoder, BM25, TagsScore)	0.918	0.316	0.936	0.940	**0.969**	0.935	54

0. BM25

The baseline model applied to the full text of the question, with **stemming** and **stopwords removal**, already gives a solid starting point for a IR system. The tuning for precision lead to the parameters $\mathrm{c} = 1.0$ and $\mathrm{k_1}$=1.2

1. BM25 % 100 >> BiEncoder

This two-step approach increases the accuracy of the results without increasing the computational requirements too much, since the bi-encoder is applied only to a small fraction of the documents retrieved by BM25. When BM25 is used as a first stage retriever, have as parameters $c = 1.0$ and $k_1 = 2.5$, which gives the best recall.

2. BM25 % 100 >> .9*BiEncoder + .1*BM25

Assigning a small portion of the total score to BM25 contributes to a more robust model because a statistical model like BM25 consistently provides reliable scores, even though it is less powerful than a Bi-Encoder.

3. BM25 % 100 >> expand_query >> BiEncoder

This method was implemented as a straightforward approach to personalize a query. However, it's evident that utilizing LLMs solely for query rewriting is a waste of resources.

4. BM25 % 100 >> .7*BiEncoder + .1*BM25 + .2*TagsScore

Our grid search identified the best coefficients as the triple (0.7, 0.1, 0.2), indicating that the Tags Score contributes meaningfully to the information retrieval task.

$$\textbf{5.}\quad \textbf{BM25 \% 100} >> \begin{cases} .7 \cdot \text{BiEncoder\_score} + .1 \cdot \text{BM25\_score} + .2 \cdot \text{tags\_score}, & \text{if num\_questions} \geq 512 \\ .9 \cdot \text{BiEncoder\_score} + .1 \cdot \text{BM25\_score}, & \text{otherwise.} \end{cases}$$

This model (*WA_H*) employs a simple heuristic: if the number of questions written by the user exceeds 512, we apply the best personalized method; otherwise, we use the best non-personalized method.

6. BM25 % 100 >> (BiEncoder ** BM25 ** TagsScore) >> RandomForestRegressor

Both Random Forest and XGBoost performed well, with Random Forest performing slightly better. Our analysis of feature importance showed that the Bi-encoder score was the most significant contributor to the final ranking. In comparison, BM25 and especially TagsScore had a minor impact.

Best model: WA_H(Bi-Encoder, BM25, TagsScore), number 5.

Finally we look at the performances of our best model on the test set.

P@1	P@3	nDCG@3	nDCG@10	R@100	AP@100
0.878	**0.306**	0.901	0.915	0.969	0.902

These results can be considered quite good. When the best answer is retrieved by BM25 % 100 in the first-stage retrieval, it is almost always ranked within the top 3 positions.

Conclusions

Finally, we would like to say what impressed us the most. First, the ability of BERT-like models to understand text. It's incredible how good the results are by simply computing the cosine similarity between the embeddings of the question and the answers. Second, the fact that a lightweight statistical model can achieve such good results; this achievement opens up the possibility of creating a reranking pipeline, which is a very clever and simple way to reduce the computational load of a neural model.

We believe that the tag score doesn't have a significant impact because we observed that most users have only posted one question on the platform, which in many cases leads to a cold start problem. In addition, the powerful capabilities of the Bi-Encoder model result in an overly influential score, so the other scores don't have much of a chance to shine.