Query Expansion

Assignment 2

CT4100

Liselotte Lichtenstein (24240597)

Jonas Michel (24238749)

25th November 2024

1 Question 1 (10 Marks)

Given a query that a user submits to an IR system and the top N documents that are returned as relevant by the system, devise an approach (high-level algorithmic steps will suffice)

to suggest query terms to add to the query. Typically, we wish to give a large range of suggestions to the users capturing potential intended query needs, i.e., high diversity of terms that may capture the intended query context/content.

Answer:

The goal of the query expansion (QE) heuristic designed here is to enhance search results by adding terms that increase the specificity of a query, proposing both relevant and diverse terms for the user to select from. The objective is to design an interactive QE technique by suggesting a range of terms that the user can then add to the query to refine the search results. Our solution integrates a global concept-based technique with local pseudo-relevance feedback (PRF), combining relevance and diversity to generate more effective query suggestions [1]. Our approach is defined in high-level algorithmic steps below in Algorithm 1, and described in detail in the following.

Step 0 (Input and Output):
As a prerequisite, the document collection is preprocessed to have embeddings and clustering for the terms in the collection. This can be done offline to reduce runtime complexity. The input to the algorithm is the query terms q_{terms} and the document collection with embeddings and clustering. It should be noted that during the algorithm, we also need to embed the individual query terms, so the embedding models must be available. The output is a predefined number of k terms that can be used to expand the original query.

Step 1 (Query Embedding):
The query needs to be embedded in order to further process it. This is done by embedding the whole query in the same space as the terms in the document collection. In our case, we want to find terms similar to the entire query, rather than just similar to the individual query terms. Therefore, we obtain the embedding for each individual query term and apply mean pooling to get the embedding for the whole query [2].

Step 2 (Get Relevant Terms):
After embedding the query terms, the next step is to extract relevant terms r_{terms}. There are two types of relevant terms: the most similar terms from the embedding space and the co-occurrence terms from the top N returned documents for the original query. This is done to ensure that we have both semantically similar terms as well as terms that co-occur with the original query terms. The semantically similar terms are selected from the embedding space to ensure a search within the entire vocabulary and, therefore, enhance diversity. The co-occurrence terms are selected from the top N returned documents for

the original query, utilizing a pseudo-feedback (PRF) approach [1]. This allows us to add relevant terms that might occur in the same documents but are not necessarily semantically similar. Each of the candidate terms is assigned a weight based on its co-occurrence with the original query terms, and then the top m terms are selected as co-occurrence terms [1].

Step 3 (Get Corresponding Clusters):
As we stated in Step 0, the collection vocabulary has to be clustered. After obtaining the relevant terms, we can now retrieve the corresponding clusters for these terms. This is done by selecting all clusters that contain the terms in r_{terms}.

Step 4 (Select Top k Clusters):
Since Step 3 can give us a large number of clusters, we must select a certain subset of k clusters from which to extract suggesting terms. We want to select large clusters, as they seem relevant, but the clusters should also be spread out in the embedding space to ensure diversity. These two measures are balanced in a score to rank the clusters, as defined by:

$$\text{score} = \alpha \cdot s + \beta \cdot d$$

where s is the normalized size of the cluster and d is the distance of the cluster centroid to the other cluster centroids. The parameters α and β can be used to adjust the importance of size and distance. Based on this score, the top k clusters are selected.

Step 5 (Extract Terms from Selected Clusters):
The selected top k clusters now represent the most relevant and diverse topics related to the original query. In order for the user to select one of these topics, we must propose a term that is descriptive of the cluster. This is done by selecting the term that is closest to the centroid of the cluster [3, 4]. If this selected term is already in the original query, the next closest term is selected.

Step 6 (Return QE Suggestions):
The terms that are selected in Step 5 are then returned as the query expansion suggestions. The number of returned terms is defined by the number of top clusters k. The number k is a restriction to ensure that the user has a limited number of terms to select from to be not overwhelmed with suggestions.

Reflection:
By precomputing the vocabulary embedding and clustering, as well as the embedding model for the query, the runtime complexity is significantly reduced. This allows for a fast response time for the user.

In cases where our query and the resulting relevant terms are so specific that we only get one high-level cluster (e.g., "jaguar engine horsepower" $\rightarrow$ cluster: "cars"), we might instead want low-level clusters, such as the different engines or Jaguar models. As a solution, instead of precomputing only one clustering, multi-resolution clustering can be used to address different levels of specificity in the query [5]. The level of specificity would

Algorithm 1 Query Expansion (QE) Suggestions

Input: Query terms q_{terms} and document collection with embeddings and clustering
Output: QE suggestion terms

Step 1: Get query embedding from q_{terms} (mean pooling)
Step 2: Get r_{terms} relevant terms (union of s_{terms} and c_{terms})
  Step 2.1: Get n most similar terms from embedding space (s_{terms})
  Step 2.2: Get m co-occurrence terms from top N returned documents on the original query (c_{terms})
Step 3: Get corresponding clusters for r_{terms}
Step 4: Select top k clusters (weighted with size and centriod distance)
Step 5: Get one descriptive term for each cluster (e.g., closest to centroid)
  Step 5.1: Check that the descriptive term is not in the original query
Step 6: Return the descriptive terms as QE suggestions

need to be selected during runtime within our algorithm before Step 3.

In general, many hyperparameters in our algorithm can be adjusted to the specific use case and also need to be examined in general studies to determine the best values. Possible tunable parameters include the number of similar terms n and co-occurrence terms m to select, the number of clusters k to choose, the weighting parameters α and β for the cluster selection, and the level of specificity for the possible multi-resolution clustering.

2 Question 2 (10 Marks)

Consider the following scenario: a company search engine is employed to allow people to search a large repository. All queries submitted to the system are recorded. A record that contains the id of the user and the terms in the query is stored. The order of the terms is not stored and neither is any timestamp. Each entry in this record is effectively an id and a set of terms.

The designers of the search engine, decide to use this information to develop an approach to make query term suggestions for users, i.e., at run time, once a user has entered their query terms, the system will suggest potential extra terms to add to the query.

Given the data available, outline an approach that could be adopted to generate these suggested terms. A brief outline is sufficient to capture the main ideas in your approach. The designers of the system wish to take into account previous queries and any similarities between users. Identify the advantages and disadvantages of your approach (briefly).

Answer:

The goal is to develop a system that suggests additional terms for an entered query q based on the recorded history of user queries H (log). Our approach incorporates three levels of scoring.

In the whole vocabulary, we have candidate terms that we might want to suggest to the user. We rank these terms based on a score that consists of three components: one based on the user's own search history, one based on the search history of similar users, and one based on the global set of all queries. By adjusting the weights of these components, the score of the term can be fine-tuned. The score is defined as:

$$\text{score}(t) = \alpha \cdot \text{user\_h\_score}(t) + \beta \cdot \text{similar\_u\_score}(t) + \gamma \cdot \text{global\_h\_score}(t)$$

Where α, β, and γ are the weights for the three components. The three subscores are based on the frequency with which a term occurs given the entered query q. This means that for the user_h_score(t), a co-occurrence score is calculated based on how often the term t occurs with the terms in the query q in the user's history. The same logic applies to the other two scores in their respective contexts (similar users and global history).

While the user history and global history are clearly defined, similar users need to be identified as such in order to generate the context (queries from similar users). We consider two users to be similar if they have a high similarity in their query history. For simplicity, this is calculated as the average query similarity between two users, given by:

$$\text{UserSimilarity}(A, B) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} \text{QuerySimilarity}(q_{Ai}, q_{Bj})}{m \cdot n}$$

The QuerySimilarity(q_{Ai}, q_{Bj}) function is used to calculate the similarity between two queries and can be determined by applying metrics like the Jaccard similarity or cosine similarity to the query representations. A threshold can be set to define when two users are considered similar. This results in a set of similar users for each user, which can then

be used to calculate the $similar\_u\_score(t)$. The $similar\_u\_score(t)$ works analogously to the $user\_h\_score(t)$ but is based on the query history of similar users.

Based on the defined score, all terms in the vocabulary can be ranked, and the top k terms can be suggested to the user. The weights α, β, and γ can be adjusted to the specific use case to give more weight to the user's history, similar users, or the global history. This weighting can also be adjusted dynamically based on available data. This allows for relevant suggestions even for new users or users with very few similar users, by giving more weight to the global history. This third component adds a global context to the term score and, depending on its weight, can give 'new to the user' terms a chance to be suggested. It also provides a fallback in cases where a user enters unusual terms that are not in the history of the user or similar users, or in the case of a user being new and having no or very limited history.

Since the third component (global history) encompasses the first and second, occurrences in the user's own history and similar users' history receive an additional boost. This can be handled by excluding the user's own history and similar users' history from the global history. However, this is not necessary, as we assume that these components are more relevant and the boost has no negative influence on performance.

We chose to group similar users together based on their average query similarity, to differentiate between two users using the same terms across all queries and two users using the same terms in the same combinations. We weight users as similar if they use the same terms in the same combinations, as they are more likely to have similar interests and therefore similar queries. It would also be possible to cluster users based on their query history.

Advantages:

- **Adaptability:** Weights can be adjusted based on the use case.
- **Global Aspect:** Does not require extensive user history to provide suggestions (user-independent component).
- **Scalability:** User similarity can be calculated offline and stored for quick access.

Disadvantages:

- **Cold Start for New Terms:** Rare or novel terms may not be suggested effectively, as they may not appear frequently in the dataset (global and user-based).
- **Outdated History:** Inheriting old search history and user profiles can lead to entirely irrelevant suggestions. For example, if a user changes their position within the company.

References

[1] Hiteshwar Kumar Azad and Akshay Deepak. "Query Expansion Techniques for Information Retrieval: A Survey". In: *Information Processing & Management* 56.5 (2019), pp. 1698–1735.

[2] Qian Chen, Zhen-Hua Ling, and Xiaodan Zhu. "Enhancing Sentence Embedding with Generalized Pooling". In: *arXiv preprint arXiv:1806.09828* (2018). arXiv: 1806.09828.

[3] Gaetano Rossiello, Pierpaolo Basile, and Giovanni Semeraro. "Centroid-Based Text Summarization through Compositionality of Word Embeddings". In: *Proceedings of the Multiling 2017 Workshop on Summarization and Summary Evaluation across Source Types and Genres*. 2017, pp. 12–21.

[4] Ilyes Khennak et al. "Clustering Algorithms for Query Expansion Based Information Retrieval". In: *Computational Collective Intelligence: 11th International Conference, ICCCI 2019, Hendaye, France, September 4–6, 2019, Proceedings, Part II 11*. Springer, 2019, pp. 261–272.

[5] Artem Lutov, Mourad Khayati, and Philippe Cudré-Mauroux. "Accuracy Evaluation of Overlapping and Multi-Resolution Clustering Algorithms on Large Datasets". In: *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 2019, pp. 1–8.