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.

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

Step 0 (Input and Output):
As 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 has to be noted that during the algorithm we have to embedde the individual query terms as well, so the embedding models has to available. The output is predifined 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 embedde the whole query in the same space as the terms in the document collection. In our case we want to find later on terms similar to the whole query instead of just similar to the individual query terms. Therefore we get 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 on the original query. This is done to ensure that we have semanticly 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 on the original query, utilizing a pseudo-feedback (PRF) approach [1]. This allowed us to add relevant terms that might occur in the same documents but are not semantically similar. Each of the candiate terms is assigned a weight based on the 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 vocalbulary has to be clustered. After getting the relevant terms, we can now get 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 back a large number of clusters, we have to select a certain subset of k clusters to extraxt suggesting terms from. We want to select large clusters, since they seem relevant. But the clusters shoudl also be spead out in the embedding space to ensure diversity. These two measures are balanced in a score to ranke the clusters as defined by:

$$score = \alpha \cdot s + \beta \cdot d$$

where s is the normalised size of the cluster and d is the distance of the cluster centroid to the other cluster centriodes. 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 represent now the most relevant and diverse topics relating to the original query. In order for the user to select one of these topics we have to propose one term that is descriptive for the cluster. This is done by selecting the term that is closest to the centroid of the cluster [3, 4]. If this selected terms 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.

Algorithm 1 Query Expansion (QE) Suggestions

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

---

Reflection:

By precomputing the vocabulary embedding and clustering as well as the embedding model for the query runtime complexity is significantly reduced. Which allowes for a fast response time for the user.

In the case our query and the resulting releveant terms are so specidify that we only get one highlevel cluster (e.g., "jaguar engine horsepower" → cluster: "cars"). In that case we might want lowlevel clusters like the different engines or jaguar models. As a solution instead of just precomputing one clustering, multi-resolution clustering can be used to address different levels of specificity in the query [5]. The level of specificity would need to be seleted during runtime wihtin our algorithm before Step 3.

In general a lot of hyperparamters in our algorithm can be adjusted to the specific use case and also need to be examined in general studies to find the best values for them. Possible tunable parameters are, the number of similar terms n and co-occurrence terms m to select, the number of clusters k to select, 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)

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.