Pseudo Relevance Feedback: Algorithmic Details

Sidharth Agarwal   2019CS50661

October 10, 2021

So I have only implemented the Rocchio Method of pseudo relevance feedback in Python for re-ranking the top k documents retrieved from an unknown retrieval system on TREC Covid dataset.

Vector Space Model:

So for calculating the weights of each term I have used the standard $weight = tf * idf$, where tf is the term frequency and idf is the adjustment for terms which are usually present in every document. So for calculating the idf I have randomly selected 10k documents out of original 192k documents available in the TREC Covid dataset. These many documents were found to be sufficiently good for calculating the IDF values of all terms, since when I experimented with 15k documents there was not much improvement in retrieval and only increased the time of computation. Those which didn't appear in the 10k documents were not frequently ocuring terms so I gave them half of the maximum value of IDF. For tfs, I only computed them for the term frequencies in the top 100 retrieved documents of each query, since those are only which we need. For representing each vector I used *dictionary* object of python where average time complexity of insert, search is $O(1)$.

Relevant and Non-Relevant Documents Vector:

Since this is pseudo relevance feedback I assumed all the top k documents retrieved by the previous retrieval system to be relevant. For the non-relevance, I assumed that the documents that were not present in the top 100 retrieved documents were not relevant. So in order to to reduce computation, I randomly selected 10k documents which are non relevant for all the queries to compute this vector, and this vector will be same for all the queries, hence reducing the computation.

Fine Tuning and Optimizations:

I implemented the modern nDCG and MAP metrics by myself in python in order to fine tune my parameters.

- *alpha and beta:* I used grid search to compute the optimal value of alpha and beta for a fixed value of $gamma = 0.15$ which is suggested to be the most opitimal value by many previously conducted experiments.
- *gamma:* After getting optimal values of alpha and beta which turned out to be $alpha = 1$ and $beta = 0.25$, I also checked for variations in the value of gamma, but $gamma = 0.15$, still remained best.
- *fields of xml/json:* Since the documents provided to us have been properly segregated into different sections or fields, I decided to choose only *title, authors, abstract* and *body text* terms into consideration. Also I assigned weightage to the terms appearing in title much more than abstract and title, and those appearing in abstract and title much more than in body text, by multipling with factors to their weights which were later fine tuned.
- *different forms of query:* so as in the data, we can choose any one out of the three fields of query which are 'query','question' and 'narrative'. I decided to choose 'query' since it contains only important terms which came out to be useful for my model. I used BeautifulSoup library here for parsing xml representation of queries.
- *Others:* I added common and appropriate delimiters that I could see in the collection to lex the sentences into tokens. I also used PorterStemmar to stem the tokens.