Department of Computer Science
The University of Melbourne
COMP90042 WEB SEARCH AND TEXT ANALYSIS (Semester 1, 2018)

Workshop exercises: Week 11

Discussion

1. Discuss the concepts of **query expansion** and **relevance feedback** and how they are related.

   Solution:

   - Query expansion improves query *recall* by dealing with the *vocabulary mismatch* problem where terminology in the query and terminology in the document collection don't match (e.g. poison vs toxin).
   - Relevance feedback is one way to expand/reformulate the query by incorporating feedback into the query process. There are many different ways to incorporate and retrieve relevance feedback (discussed below).
   - However, there are other query expansions methods such as using a thesaurus or wordnet which do not require relevance feedback.

2. Discuss and give an example of the following forms of relevance feedback.

   (a) User relevance feedback
   (b) Pseudo relevance feedback
   (c) Indirect relevance feedback

   Solution:

   (a) User feedback comes directly from the user issuing the query. This might be in the form of clicking on a result or swiping left/right on your phone. In the lecture we looked at Pinterest where we could refine the results returned for the query "watch" by clicking on watches we like. HOW this feedback is incorporated into the search process depends. One way could be to do query expansion.

   (b) Pseudo relevance feedback (or sometimes called blind feedback) is an automated system that does not require any actions by the user. It is based on the assumption that after running the original user query, the top documents will likely be relevant. So we analyze these documents to extract additional informative/important terms from these documents. Different methods can be used to determine these additional terms. We might look at the topic models of these documents to determine additional terms. Another methods might look at the frequency of terms in the documents relative to their occurrence in the overall collection. For example, in the top documents of the query "melbourne public transport", the terms "tram" or "bus" will likely occur with higher frequency than what we would expect from other documents in the collection. Thus, these terms might be candidates for query expansion. If these terms are included in a secondary query, we would be able to find documents talking about "melbourne trams" even though they might not mention "public transport". Thus, the *recall* would increase.

(c) User feedback is problematic because users might be reluctant to provide manual feedback. Pseudo relevance feedback has the drawback that no actual feedback from any human is incorporated in the process. Indirect user feedback seeks to incorporate implicit feedback from users. That might be statistics about what pages users visit after being displayed a results for a certain query. Thus, the feedback would not be from the current user seeking information, but other users who searched for the same query in the past. These information are collected in large quantities in query click logs at most major web search engines and e-commerce platforms. Note that these information might not be used to perform query expansion but rather re-rank documents directly based on this click feedback.

3. Is it possible to perform **query expansion** without **relevance feedback** and vice versa? Discuss!

   Solution:

   (a) Query expansion can be performed using global resources such as WordNet which do not require relevance feedback.

   (b) Relevance feedback from a user (direct or indirect) can be incorporated without performing query expansion. This might be in the form of re-ranking existing documents or discovering documents that are similar to documents (the "more like this" button) to the ones users consider relevant.

4. Discuss the process of static **inverted index construction** and how it can be used to perform in incremental index construction.

   Solution:

   - Space efficient inverted index construction splits a static document collection into blocks.
   - For each block an individual inverted index is constructed (space efficiently)
   - At the end, the inverted indexes for each block are merged to create a single large inverted index for the whole collection.
   - In the incremental setting we are working with a similar setting. We have a collection that is already indexed (somewhere stored on disk or in memory). As new documents arrive we form a block and build an inverted index for those.
   - At query time we have to query all of the inverted indexes at the same time and merge the results

5. Why is a **logarithmic** index layout useful? What are the disadvantages of such an index structure?

   Solution:

   - Lets say we would index $N = 10$ billion postings of the course of a year.
   - However, at each time we only store $n = 1$ million postings in RAM.
   - If the index becomes too large we store it to disk by merging it with a larger inverted index containing the postings we have stored so far.

- Thus, we would want to merge the small index containing n postings with the larger index.
- At the start of the year we start with having nothing on disk and n things in memory.
- Merging the small index to disk would just mean storing n postings to disk.
- The next time we have n postings in memory, we also have n postings on disk. Thus merging them requires writing out $2n$ postings.
- The next time we have n postings in memory, we would have to write $3n$, $4n$, $5n$, ..., N postings as we do the merging throughout the year until at the end we have an index on disk containing roughly $N = 10$ billion postings.
- However, because we throughout the year we performed lots of merge operations which amounts to $N^2/n \approx n + 2n + 3n + 4n \ldots N$ IO operations. This is very expensive.
- Instead we keep a logarithmic number of indexes of different size. For example, our first level stores at most n postings. The next level stores at most $2n$ postings, the next level stores at most $4n$ postings and so on. If the a level fills up we only merge with the next higher level. So to incorporate n new postings, we merge with the $2n$ level. If that fills up we merge that with the $4n$ level etc. As the sizes of at the different level increases merging becomes cheaper as less IOs are required in total.

Programming

1. Work on the project! :-)

Catch-up

- What is top-K retrieval and why is it useful?
- What are some of the reasons a search engine would want to offer a query completion service?

Get ahead

- The lecture links to a blog post discussing a more advanced index merging scheme used by Lucene. Read up on why Lucene uses this method compared to the logarithmic method discussed in the lecture.
- In general, similarity measures that we have discussed up till now are quite simplistic as they only take into consideration term frequency statistics. Modern search engines use many more advanced features to determine the relevance of a document. Can you think of some features that might be useful?