CS3245 Homework 4 bonus document

Describe the information related to the query expansion techniques you have implemented. You may include tables / diagrams in this document.

Student Number	A0136070R
Student Email	E0005572@u.nus.edu

Contents

Query expansion methods implemented

I implemented the following query expansion methods:

1. Manual thesaurus-based method with WordNet
2. Automatic thesaurus-based method with term-term co-occurrence values
3. Semi-automatic thesaurus-based method with WordNet + term-term co-occurrence filter

Manual thesaurus-based method with WordNet

Method (1) is as follows:

i. For each lemma in the query, get the synsets from WordNet.
ii. Get the lemmas for these synsets and remove duplicates.
iii. Add all these lemmas to the original query.

In my opinion, we should restrict the number / percentage of new lemmas to be added. There is no magic formula for this number / percentage decided in my program, but it ought to be small to balance between expansion and drifting.

Automatic thesaurus-based method with term-term co-occurrence values

Method (2) is as follows:

i. Follow the first two steps of method (1).
ii. Instead of adding the synset lemmas blindly, fetch the nltk.Text of the top k relevant documents from the index.

In my program, k is computed from a function of the total number of relevant documents (see Line 116 in search.py). One round of retrieval should be done before expanding the query (otherwise, there will be no "relevant documents" determined in the first place).

iii. Get the union of lemmas across the top k documents that co-occur with each query lemma.
iv. Add all these lemmas to the original query.

Semi-automatic thesaurus-based method with WordNet + term-term co-occurrence filter

Method (3) is an upgrade of the manual thesaurus-based method, inspired by method (2).

In method (2), there are many parameters for us to decide whether a new term should be part of the query expansion, but in method (3), these parameters are replaced by the set intersection condition explained below.

Method (3) is as follows:

v. Follow the first two steps of method (1) and then the first three steps of method (2).
vi. Instead of adding the synset lemmas or the co-occurring lemmas blindly, intersect these co-occurring lemmas with the synset lemmas from WordNet.
vii. Add this intersection to the original query.

Additional notes about query expansion

Note that we may or may not want to subtract the lemmas from the query expansion which already exist in the original query. Not subtracting them would mean an increased importance on the term which exists in both the query and the query expansion. In my implementation, these "duplicate" lemmas are subtracted.

Complexity analysis

Increase in time complexity

The increase in runtime by the query expansion methods are dependent on the various parameters inside each of these methods.

For method (1), the expected runtime increases linearly with the total number of terms in the expanded query. Let O(x) be the time complexity of getting the cosine similarities between 1 term

and every document in the collection in sorted order, where x is some variable that models the complexity of my retrieval pipeline (i.e. boolean retrieval then vector space retrieval).

Then for an original query with j terms and a query expansion of k terms, the time complexity of the retrieval process is O((j+k)*x).

For methods (2) and (3), there is a need to first retrieve the most relevant documents for the original query before expanding the query. (Otherwise, we do not know which documents to get the co-occurrence values from.)

As such, using the same set of variables defined above, the time complexity of method (2) and (3) is O(j*x + (j+k)*x + y) where y is a constant that differentiates between method (2) and (3), since these two methods process co-occurrence differently.

Increase in space usage

The increase in space complexity by the query expansion methods are as follows:

For method (1), we need to store WordNet. The WordNet downloaded from the Princeton University website is about 40 MB.

For method (2), we need to store either the word_tokenize-ed text or the nltk.Text version of every document. Storing nltk.Text consumes more space than storing simply the word_tokenize-ed text because nltk.Text contains other class-specific fields for ease of processing. On the other hand, if we store the latter, we will need to incur computational costs during the search phase to convert to nltk.Text type. The nltk.Text of every document in the CSV file occupies about 1.7 GB of space.

For method (3), since both WordNet and word_tokenize-ed / nltk.Text resources are used, the increase in space usage is the sum of additional space usages by method (1) and (2).

Conclusion

In conclusion, query expansion increases the complexity of the retrieval. The more the expected precision / robustness of the query expansion method, the more the resources used. My personal inclination for query expansion would be the following:

- Should skip query expansion for longer queries as they are more likely to have all the relevant terms inside already (need to define what is “longer”)
- Should further limit the total number of terms in the query expansion to balance against query drifting (e.g. maximum of two query expansion terms per query term)
- Should use a more niche thesaurus depending on the domain of the collection (i.e. use a law thesaurus for a collection of court files)
- Should use a word sense disambiguation framework to get the most appropriate sense for each query term and get the synonyms for those senses (effectiveness is now largely dependent on the framework)