**A0172868M-A0171426J-A0168355W-A0164816Y**

**QUERY REFINEMENT AND**

We experimented with different types of query refinement techniques. They are mostly implemented in QueryExpansion.py.

## Relaxing boolean and phrasal queries

The first stage of query expansion, as explained above, involves relaxing the restrictions placed on the query from phrases and boolean operators. Since the terms in the user queries may not be the exact terms desired, we need to relax the AND portion of the query, so that even if the term given is not correct, the results for other parts of the query can still be returned. To achieve a baseline tf-idf framework, all boolean operators and phrase markers were stripped from the query string.

Every query can one of the following four types:

1. +phrase, + boolean: e.g. "fertility treatment" AND damages  
2. +phrase, -boolean: e.g. "fertility treatment" damages  
3. -phrase, +boolean: e.g. fertility AND treatment AND damages  
4. -phrase, -boolean: e.g. fertility treatment damages (basic free text query without phrases or boolean operator)

For an original query string with both phrases and the "AND" boolean operator, query expansion can allow us to relax these restrictions in order to produce the other 3 combinations. When the original query either does not have phrases or the boolean operator, it can still be relaxed to the free text query. For a maximally complex query of type 1 (including boolean operator and phrases), the orders of search between these four types of queries can be permuted and experimented with to determine the importance of preserving the additional information of phrases and boolean operators.

For 3. We note that the set of documents containing the phrase “fertility treatment” is a subset of the set of documents containing (fertility AND treatment).

## WordNet/Thesaurus Query Expansion

The NLTK WordNet was used as a thesaurus to implement query expansion. In particular, the synonyms of the terms were found using the synset feature of WordNet. Additionally, we also experimented with using hypernyms and hyponyms to return related words, however, because too many of such words were returned, we decided to stick with synonyms. The additional synonyms retrieved were appended onto the free text version of the original query to create a longer free text query. Due to time constraints we were unable to implement a potential improvement, which involves ensuring for a boolean query that the synonyms were intersected. For example, a query 'quiet AND phone call' could be expanded to '(quiet OR silent) AND (phone call OR telephone call)'.

## Relevance Feedback and Rocchio Algorithm

For relevance feedback based on the Rocchio Algorithm, our system makes use of the top 1000 returned documents from the basic search which are assumed to be relevant, on top of the list of documents identified as relevant in the original query file. The document IDs are then used to retrieve precomputed and stored document vectors in the document properties file, which are then combined to give the centroid vector of the relevant documents. This is done such that there is no need to traverse the postings file to build the document vector for each relevant document, which would be extremely expensive. We incorporated this requirement into our indexing phase, using properties\_helper.py and vectors.txt.

An additional optimisation involves storing the vectors as sparse vectors using a dictionary mapping terms to the tf values. This is necessary since the vectors would include many 0 terms if the dimension of the vector was the size of the entire dictionary. Furthermore, even after computing the centroid vector, there will still remain many non-zero dimensions in the centroid. In order to improve on efficiency, there is a need to remove some of the non-zero terms. To do this, each component was multiplied with idf in order to reduce the value of more common and hence less useful terms. The top 50 terms (excluding stopwords and punctuations) were chosen for the final centroid vector, as 1. Too many terms may slow down the query and 2. Useless dimensions that would not contribute much to the query can be removed.

The new centroid vector can then be added to the original query vector to derive a new query vector used for VSM retrieval. For simplicity, the original query vector is made to be a free text query such that boolean operators are removed and phrases are converted to single word terms. The additional documents found from relevance feedback are appended after the already returned documents.

# **Experimental Results**

We use notations to denote the order we executed the query refinement techniques.

1. POSITIVE LIST
2. +PHRASE, + BOOLEAN ("fertility treatment" AND damages)
3. +PHRASE, - BOOLEAN ("fertility treatment" damages)
4. -PHRASE, -BOOLEAN (fertility treatment damages, baseline)
5. +WORDNET, -BOOLEAN
6. ROCCHIO ALGORITHM
7. -PHRASE, +BOOLEAN

### QUERY RETRIEVAL

To understand our performance against the different query types, we retrieved the query strings from the competition framework:

|  |  |
| --- | --- |
| q1 | quiet AND "phone call" |
| q2 | prostitute AND "forced sex" AND payment |
| q3 | pretend to be officer |
| q4 | "fertility treatment" AND damages |
| q5 | publish hurt son |
| q6 | "good grades" AND exchange AND scandal |

### APPROACHING BASELINE

We first implemented a baseline system disregarding any query refinement (-phrase, -boolean), so ("fertility treatment" AND damages) becomes (fertility treatment damages).

|  |  |  |
| --- | --- | --- |
|  | BASELINE | OURS (JSON) |
| Q1 Average F2 | 0.0108595077894779 | 0.0108201093105916 |
| Q2 Average F2 | 0.362745098039216 | 0.362745098039216 |
| Q3 Average F2 | 0.0113111117361595 | 0.0112925624835047 |
| Q4 Average F2 | 0.496296296296296 | 0.496296296296296 |
| Q5 Average F2 | 0.104510939130271 | 0.103200491131526 |
| Q6 Average F2 | 0.3 | 0.30050505050505 |
| Mean Average F2 | 0.214287158831903 | 0.214143267961031 |

As expected, we were close to the baseline tf-idf model. However, we were still slightly below and we hypothesised that the baseline model appended the title to the content before processing, so we did that as well.

### INITIAL TEST

We did an initial test of the techniques above:

1. positive list 2. +boolean, +phrase 3. -boolean, -phrase 4. +phrase -boolean 5. Wordnet expansion 6. Rocchio expansion

|  |  |
| --- | --- |
| Q1 Average F2: 0.0318471337579618 | +193% |
| Q2 Average F2: 0.276839007986549 | -24% |
| Q3 Average F2: 0.00854730742939909 | -24% |
| Q4 Average F2: 0.516624579124579 | +4% |
| Q5 Average F2: 0.103200491131526 | -1% |
| Q6 Average F2: 0.172659817351598 | -42% |
| Mean Average F2: 0.184953056130269 | -14% |

This performed worse than the baseline tf-idf. There was only improvement to one type of query, over the baseline results.

### IMMEDIATE WORDNET EXPANSION

We analysed the submissions made by other teams, as well as the given positive list. Interestingly, we found that the positive list for “quiet phone call” did not contain the exact terms. Rather, “telephone call” was in the document.

This led us to the revelation that it may be better to do a wordnet expansion first. To better isolate our results, we attempted to only do the wordnet expansion. That is, we only return the results from the wordnet expansion (not including positive list or the hard conjunctions). We did this because an accurate wordnet expansion should still retrieve the positive list results. By right, the wordnet expansion should also return the results of the original query (we append the original query string to the wordnet expansion).

Unfortunately, we received very bad results on this experiment.

|  |  |
| --- | --- |
| Q1 Average F2: 0.00198346589850713 | -82% |
| Q2 Average F2: 0.00152853996309595 | -100% |
| Q3 Average F2: 0.00109824660167018 | -90% |
| Q4 Average F2: 0.00148902389641845 | -100% |
| Q5 Average F2: 0.00166292668856411 | -98% |
| Q6 Average F2: 0.000682437441147597 | -100% |
| Mean Average F2: 0.00140744008156724 | -99% |

### IMMEDIATE ROCCHIO FEEDBACK

Given the same concerns as earlier (that is, the returned documents in the positive list do not contain the exact query terms) we try another way to return the documents, using Rocchio feedback. In theory, Rocchio feedback will find the documents containing “quiet phone call”. It is possible that these documents also synonymously use “telephone call” in some instances, so we will get “telephone call” in our centroid.

|  |  |
| --- | --- |
| Q1 Average F2: 0.00198346589850713 | -82% |
| Q2 Average F2: 0.00152853996309595 | -100% |
| Q3 Average F2: 0.00111074417157611\*\* | -90% |
| Q4 Average F2: 0.00148902389641845 | -100% |
| Q5 Average F2: 0.00166292668856411 | -98% |
| Q6 Average F2: 0.000682437441147597 | -100% |
| Mean Average F2: 0.00140952300988489 | -99% |

We received very interesting results, almost identical to what we got from an immediate wordnet expansion. Only the third query (pretend to be officer) returned a slightly different result. We believe this is the case because both Rocchio and WordNet return a very very large number of documents (the entire corpus). In the end, the precision suffers a lot that we hit minimum precision.

Both the Rocchio algorithm and WordNet expansion used on their own have negative results. We continue to experiment with different ways to improve our F2 score. We looked at the leaderboard and saw some teams using part-of-speech tagging (pos) tagging. We did not attempt to try this because pos tagging is likely to face the same issues we have with Rocchio expansion and wordnet, which is returning almost the entire corpus.

Currently, we want our query expansion to return more documents but not the entire corpus. We introduce tweaks to our Rocchio algorithm to make it return fewer number of documents. Besides adding a cutoff on the number of documents we use to generate the Rocchio query, we also add a cut-off for which terms should appear in the new query.