**Implementing Techniques**

Our team decided to implement techniques in the form of (manual) thesaurus expansion and Rocchio algorithm. We have also performed some other optimisations and tweaks to accommodate their effects/improve performance.

**Thesaurus Expansion**

Thesaurus expansion in our project is done via the WordNet corpus from NLTK. Using that, we can easily find synonyms for every word in the query. However, it must also be acknowledged that using query expansion for every term in the query can be a bit excessive, especially if said query is very long. This is one of the drawbacks of query expansion that we learnt in lecture.

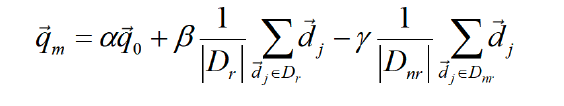
To mitigate this, and to search for the words which seem more likely, we try to find the query weight, via one additional round of cosine scoring on the term. Although this may seem like a lot of work, we get a good idea of which terms we should be expanding. Then, we can only choose to expand the terms which meet a certain threshold, as these are the terms which are judged to be more important. This way, we avoid the issue of expanding irrelevant or terms which are perhaps less related to the query at hand.

Admittedly, there are some drawbacks to this approach, namely that there is an assumption that the system works in the first place, else the terms which we have expanded may not be the relevant ones.

We tried experimenting with different values for the threshold weight (which determines whether a term is suitable for expansion), and after much testing we decided to use our current value indicated in the search.py file. We also tried with no threshold (hence applying it every time), but it was not optimal.

**Rocchio (Relevance feedback)**

The original Rocchio algorithm takes the form of:



For the purposes of the project, we ignore the irrelevant documents centroid, since it is going to be comprised of the majority of the documents that we have.

Instead, we only focus on the relevant vectors. Normally, for these document vectors, the vector used (Dj) is generated using the document count vector. However, in the interest of saving space, we only store the top K terms (in terms of frequency) of each vector. We do this using the assumption that anything after that is probably less relevant as a search keyword/term and hence should not affect the score much. Of course, this is an approximation and definitely does not hold true all the time, but we felt that the trade-off (to reduce index space and additional terms to search during the search phase) was acceptable. The query vector was computed as per the lecture. Note that we calculate this term-wise – that is, we calculate the entries of the refined query vector row by row.

Some key points of what we did (our version of the algorithm) is given below:

1. For each term appearing in the query, calculate the centroid’s entry/value of the term in the relevant documents. Use this value in the Rocchio Algorithm formula to calculate the refined query vector’s value for that term’s entry.
2. There are some terms that can be found in the relevant documents but not in the query. To account for this, we gather the set of terms which were not seen in the query, and then calculate the centroids for those terms as well. We then do the normal cosine scoring (via dot product) on these terms with the postings. This is essentially similar to that performed for terms in the query, just that we begin with a initial query vector value of 0 and add the centroid’s value to it.
3. In the process of calculating the centroid value of the relevant documents, each document’s contribution is divided by the document length to consider the distribution of the term in that document. This is done before averaging (that is, before dividing by total number of relevant documents). The magnitude of the length of each document vector is readily available as it is stored during indexing.

The parameters for the Rocchio algorithm were varied according to our experimental data and what we thought fit the leaderboard the best. (EMPHASIS\_ON\_ORIG as a multiplier for the original query in the Rocchio algorithm forumla, and EMPHASIS\_ON\_RELDOC as a multiplier for the relevant documents’ centroid in the forumla).

Here, we tried playing around with all sorts of values for the emphasis values, and also for the value of K for the top K most common terms in each document; and it turns out that our current emphasis values and K values tend to give better performance.

It seems placing a lower emphasis/significance on relevant-marked documents (value roughly 0.6-0.8) and higher of that on original query (value roughly 1) gives better performance. This is proven to be better, in general, than if they were equally emphasised.

We noticed that for K values, if the value of K falls too low, there is a reduction in performance. Also, if it is too high, the performance falls as well. It seems that K value of around 12-14 gives better results based on the leaderboard. This could be because with low K, we may not be getting the terms which are significant and therefore perform the Rocchio Algorithm with them. Moreover, with high K, we have too many terms and then we have too many documents which our system marks as relevant. Having a high K value also skyrockets the number of document IDs returned.

Moreover, we have also performed the following optimisations and tweaks (and experimented with their relevant values):

* For Boolean queries, if we have fewer results than a certain threshold, we will then expand the AND search to become an OR search between the terms. If still not enough results, then we will perform a free-text query search for individual terms in any phrases as well. This last one is not too bad as the phrases are at most a length of 3. This gives us more results if we have insufficient ones, so we are more likely to capture some relevant documents by casting our net wider.
* For occurrences in different zones/fields, we will boost the score accordingly to their importance via a multiplier (different for each type) and this will affect the ranking. These multiplier values were determined by playing around with during experiments, and also keeping in mind how likely the importance of the different zones/fields are to the user performing the search.

For instance, say user enters a number, which can be like a date. Here, we find that law documents may contain lots of dates of situation’s happenings, but these may not be the same as the document’s publication date. Hence, as lawyers are more likely to search for happenings by date rather than the document’s publication date, content > date.

Another instance, say a user enters a word/keyword. This term may appear in the title, court, or content. However, since we don’t really memorise content, it is more likely that this search term relates to the title or the court. Thus, title&court > content. Moreover, there is usually less utility in searching for a court than a title. Hence, title > court.

Therefore the importance we decided to go with, in descending order is: title > court > content > date

* Since we recognise that the users/lawyers enter search terms because they are more important, we reflect the higher importance of these terms by doing post-processing in that documents having these exact search terms will have slightly higher scores via a multiplier (EMPHASIS\_ORIG\_MULTIPLIER\_POSTPROCESSING ; greater than 1) as compared to those without these terms.
* Remove stop words in our index. In general, compared to without removing stopwords, the performance tends to improve when we remove stopwords. This is why we continued with it for the indexing phase.