# CS3245 AY21/22 S2 HW4 Bonus

A0188493L-A0218271J-A0200025H-A0206154N

In this homework, we have implemented 3 query refinement techniques: 1 based on query expansion and 2 based on relevance feedback. As they did not perform up to expectation, they have not been combined with the main submission code, but the code can be found in the folder bonus/.

In particular, the logic can be found in:

* bonus/query.py under QueryRefiner class
* Part of the logic of extracting the relevant documents can be found in bonus/search.py ‘s run\_search
* The normalising logic in rocchio can be found in QueryDetails’s count method, which is marked with the @property decorator

## Query Expansion

The query expansion relies on the NLTK’s Wordnet library. The high-level idea is as follows:

1. For every of the original tokens (before applying any pre-processing), we use a hybrid between Wordnet’s synonym sets and our own custom curated set of words.
2. The custom set of words are curated from reading through the documents, and attempting to understand the context and grouping together words that frequently come together in the same context
3. Suppose we don’t recognise the word:
   1. We will use wup\_similarity in order to give some form of scoring to the synonym sets. The base of the comparison is by simply taking the first set given by NLTK. This is empirically chosen as it seems that the first set tends to give us the set that we want, but there are worst case examples where this might not be a good approach.
   2. Based on a parameter k, we will take the first k synonyms (not synonym sets) that we see from the synonym sets.
   3. For example, if I have 3 terms “hello information retrieval”, and k=2, every term will be given 2 new terms. For example, “hi greetings” for “hello”, “details knowledge” for “information”. “reclaim recapture” for “retrieval”. Thus the refined query is as if we are running “hello information retrieval hi greetings details knowledge reclaim recapture”
4. Else, we do recognise the word being part in our custom curated set of words:
   1. We will simply append the set of words that is associated with that word

As justified in readme.txt: We feel that the custom set of words should be more informative in the context of legal case retrieval than relying on a generic thesaurus, since our goal is to build a system for the specific purpose of legal case retrieval, thus it should make sense to make an attempt at optimisation by studying the nature of the documents.

## Relevance Feedback

In order to perform relevance feedback and tuning the query vector, we require the document vectors. In order to support this, our indexing has to be modified as the postings is a mapping from a term to document ids. That is, given a document id, we have no information about what the terms we have in that document are.

A naive approach is to simply build the inverse mapping, but this would take a large amount of space. Our solution is to take the top 5 most frequent words in the document and consider them to be the “most important” terms in that document. We took care to remove stopwords, as well as only take words of length 3 and above (we noticed that there are a couple of single and double-letter words that are not representative of the document at all).

Thus, during searching, we will retrieve the top 5 most frequent words and use it in order to refine the query vector

### Naive Pseudo Relevance Feedback

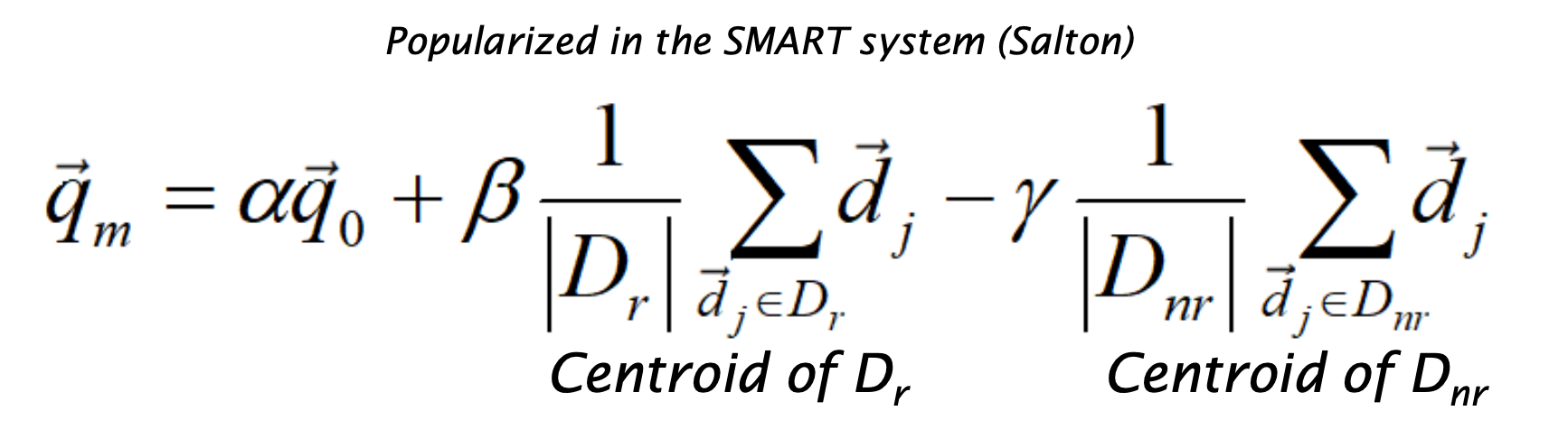
We first perform free-text retrieval in the Vector Space Model by using the original query. Then, we retrieve the top 5 document IDs, and their corresponding most important words.

Afterwards, these words are naively appended to the query (much like how we appended the words after query expansion)

### Pseudo-Rocchio for Pseudo Relevance Feedback

Like the naive version, we first perform free-text retrieval in the Vector Space Model by using the original query. Then, we retrieve the top 5 document IDs, and their corresponding most important words.

Similar to Rocchio in lecture:



We perform a modified version of the calculation, by setting alpha = 0.8, beta = 0.2 and gamma = 0. Furthermore, we are interpreting the vectors as the counts first before computing tf-idf weights.

Thus specific to our context (alpha and beta values above, with 5 document IDs), the formula becomes:

qm = 0.8q0 + (0.2 / 5) \* sum(dj)

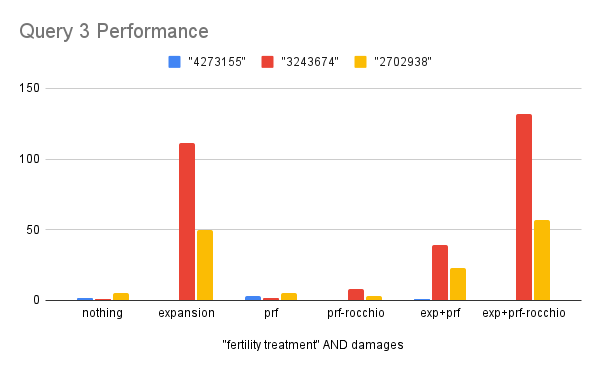
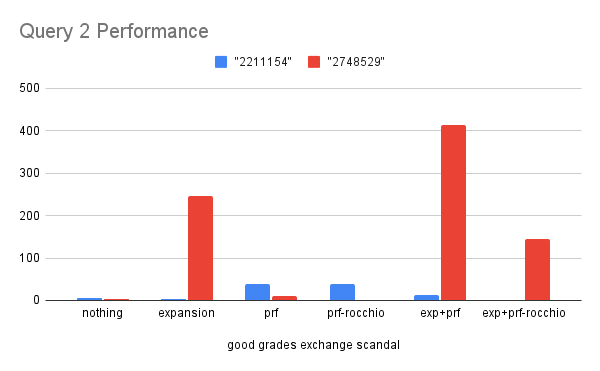
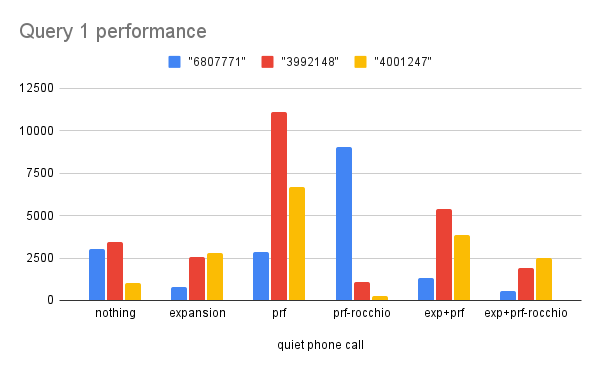
## Experiment

We used the sample queries q1-3 to us as well as the relevant documents given. As we have limited relevant documents, we are unable to compute an accurate precision and recall score to give a more holistic evaluation. We are only left to compare how well these different heuristics are able to return these relevant documents.

We experimented the following (which are our axis labels in the chart below):

* nothing: No query refinement used
* expansion: Query expansion used
* prf: Naive Pseudo Relevance Feedback used
* prf-rocchio: Rocchio used
* exp+prf: Query expansion and Naive relevance feedback used
* exp+rocchio: Query expansion and rocchio used

What we have done is to run all these different approaches to query refinement, and let’s say q1 has been marked with the relevant document "6807771", we will then find the ranking of "6807771" in the output for our q1. Eg if it was the top document returned, then the rank is 1. **Note that this means that the lower the value, the better**, as it means our system was able to rank it very highly (low number). This has been plotted in the following bar chart:

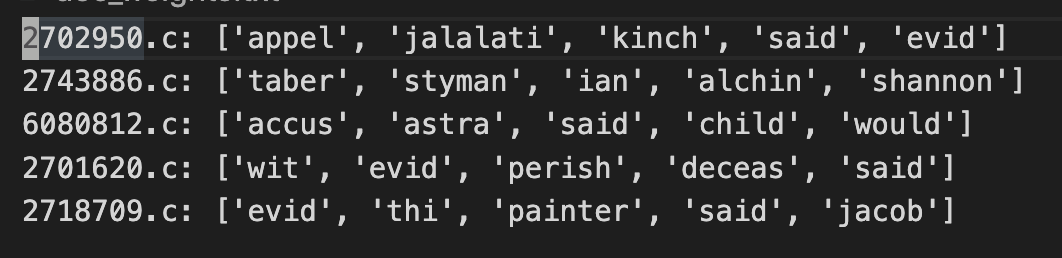


## Analysis

At a glance, there does not seem to be any clear winner. For Query 1: exp+prf-rocchio performs the best, but it performs badly for Query 3. For Queries 2 and 3, using prf and prf-rocchio seems to have reasonable performance, but it performs pretty bad for Query 1. Using nothing does well for Queries 2 and 3 as well, and seems to have reasonable performance in Query 1 as well.

#### Query 1

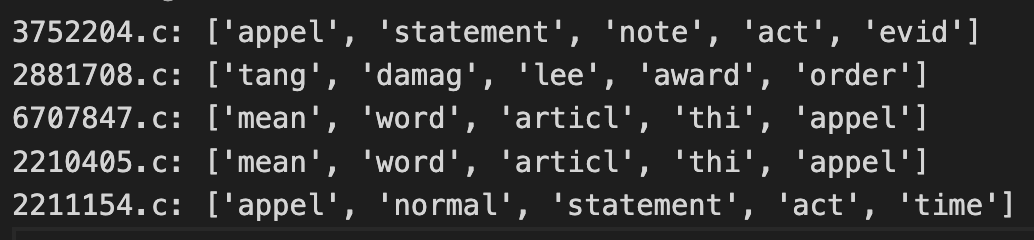
In general, Query 1 seems to be a “difficult” query in that none of our implementation ranks the relevant documents highly. Let us study this carefully and compare the performance of the different approaches. After scrutinising the documents marked as relevant, we quickly notice that the terms used more often are “telephone”, rather than “phone” which the query suggests. Query expansion allows us to introduce this new word “telephone”, as indeed after manual checking we see that the synonym sets returned by Wordnet contains “telephone”. This might explain why expansion has a slight edge over most the other implementations (especially against ‘nothing’). We see that prf and prf-rocchio performs quite poorly, and after scrutinising the initial top returned documents before doing the relevance feedback, we see that the top words seem to be nothing related all to ‘quiet’ ‘telephone’ or ‘call’:

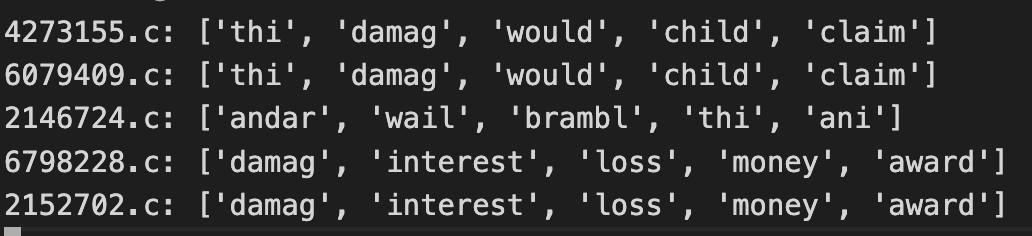


This might have drifted the query by a significant amount, which explains the poor performance of pure prf and prf-rocchio. Combining this approach with query expansion might have dampened the impact of these additional words, especially for prf-rocchio since we are not simply adding all these terms (like in prf) to the query, but rather we are putting smaller weights to these terms than query expansion.

#### Query 2 and 3

The following are the top few document IDs and their important terms which will be passed to the pseudo relevant feedback:

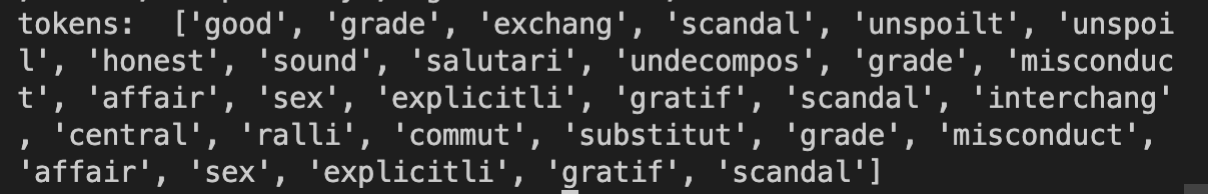


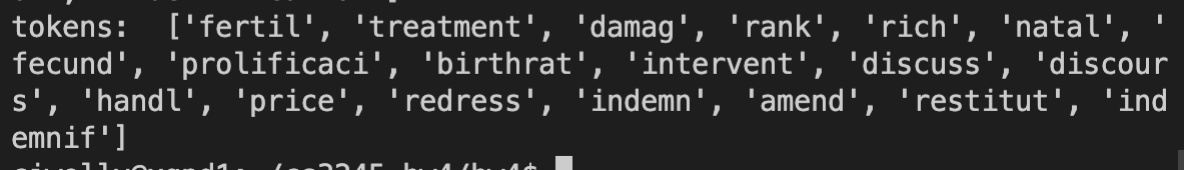


It is most interesting that some documents actually have the exact same set of important terms (and further, in the exact same order!). Upon scrutiny, documents 6707847 and 2210405 are exactly the same case! Likewise for document pairs (4273155, 7079409) and (6798228, 2152702).

Our suspicion is that we happen to be lucky and the top terms of these duplicate cases happen to match exactly the ones we need to obtain the relevant document suggested by the teaching team. Thus, we have artificially boosted our query that brings us very close to what we need for these two queries.

As for our query expansion, these are the final set of tokens after expanding:





We can only speculate that the expanded terms might have been very irrelevant (we also see some duplicate terms, which we should have deleted perhaps). This causes the query to drift from the intended outcome.

#### Synthesis

Overall, while prf and prf-rocchio seem to have occasionally decent performance, we would attribute it to being lucky with retrieving duplicate documents that are very relevant. Furthermore, much refinement might be needed to reduce the court-related terms as the ‘important terms’. For example, we see a couple of ‘appeals’ (‘appel’ after processing), ‘statement’, ‘note’, ‘article’. This is typically irrelevant in terms of query as most court-related text would involve these anyway.

Thus, it leaves us the choice of using either no query refinement at all or query expansion. Purely from the data presented, doing no query refinement seems to be the way to go. However, from the time of collecting experimental data to the submission, we have managed to continuously refine our curated set of words such that it performs relatively well in the leaderboard (with an average F2 score of 0.295). Unfortunately, we have not had the chance to make an updated comparison with the other approaches due to the tight deadline, but the prior results should suffice to suggest to us that it might not be worth pursuing pseudo-relevance feedback for our system.

As such, our final submission will make use of an updated query expansion that is an improvement compared to the version with data presented as above.