Discussion of query refinement/relevance feedback techniques used

1. Query Refinement – Wordnet

* Nltk’s wordnet returns a list of Sysnets for a given word. We used wordnet to expand a given query before doing a VSM on the new query vector.
* So given a query, wordnet will return us some synonyms of the query. We then add these synonyms to the query vector and do VSM. For example, if the query given is ‘motorcar’, wordnet will return us ['car', 'auto', 'automobile', 'machine', 'motorcar']. This list then becomes our new query vector.
* On immediate effect of doing this is more documents being returned as our output.
  + With more documents, our precision drops because many documents returned does not contain any of the original queries
* Sometimes wordnet gives us synonyms that are irrelevant of what we are trying to find. For example, ‘good’ can mean both ‘superb’ or ‘commodity’, and these are words with very different meaning. Hence without being clear of what the user is trying to find, wordnet tends to result in too many irrelevant documents.
* There is a slight increase in the runtime taken when using wordnet
* We do not use wordnet to increase the constraint when trying to find a strict ‘AND’ query (step 1.b.2 in our README). E.g. ‘motorcar AND ocean’ will not become ‘motorcar AND car AND ocean’ because we will miss out on potential documents.
* Wordnet is turned off because of the many irrelevant documents that it returns.

1. Query Refinement – Relevance Feedback
   * For relevance feedback, we needed to store the document vectors of every document. So, while indexing, we store a separate dictionary which contains all the document vectors. To our surprise it was only slightly over 100MB.
   * After that we still had to convert the document vectors from term-frequency vectors into tf-idf vectors. We do this inside search.py.
   * So, when we search for a query, we look at the lines below it to find any other relevant documents. If they exist, we retrieve their document vectors and "push" the query accordingly using the Rocchio formula.
   * Vectors in our implementation are represented as python dictionaries and so we had to write add\_vectors which adds two vectors and multiply\_vector which multiplies a vector with a scalar. Both of these are implemented within our utils.py.
   * After having implemented these, we were able to update our query vector.
   * alpha and beta were two hyper parameters that we adjusted to see which pair gives us the best results. We decided to stick to 2 for alpha and 0.3 for beta after a considerable amount of experimentation.
   * Finally, we have turned off relevance feedback (can be switched on by setting the relevance\_feedback\_switch in search.py to True). This we did because after adjusting the query vector, many of its components had a non-zero value. This increased the computation time significantly for our search. Hence, we decided to let it be turned off by default. Should one decide to turn it on, he/she should expect about a 4-5 times slowdown in querying. However, it is likely that the documents retrieved will be more relevant.