Patrick Yeh (psy2107)
Sameer Saxena (ss6167)

Project 1 README

Submitted files:

- project1.py
- transcript
- README

How to run program:

1. Run 'python3 project1.py'
2. Follow the instructions on the terminal
    a. Note that 'Y/N' will have 'Y/y' be mapped to a 'yes' and anything else mapped to a 'no'

Dependencies:

The program uses the following libraries: sys, pprint, itertools, json, and googleapiclient.discovery. In particular, the last one should be installed using 'pip'.

Internal design:

At a high level, the program works as follows. We first receive the initial query and desired precision values from the user, and then we will begin our main feedback loop that's designed to iteratively improve on this initial query. The loop works as follows. We make a call to the Google API to retrieve search results, receive feedback from the user on which documents are relevant, and then we will use the relevant and irrelevant documents to further refine our query. Queries are represented as a list of tokens/words that are then put in a specific order and concatenated to form a phrase. We track relevant documents using a list and concatenate all their snippets/titles into an overall corpus which will be used for query optimization. Once we have modified the query, we will loop back and repeat the process of getting the search results and asking the user for feedback, and either reaching our desired precision or taking steps to improve on it. Our implementation follows this basic structure, and accounts for other important tasks involving formatting/processing data and accounting for edge cases mentioned in the writeup (for example, the case where the precision is 0 or we don't have 10 initial results.

Query modification:

One responsibility is to augment the query by adding at most two words. We do so by getting term frequencies from all of the relevant documents and, following a schema similar to the Rocchio algorithm and the ide-dec-hi formula we found in one of our sources (in particular, the fourth on the list explains it well), we will choose an arbitrary irrelevant document to induce some negative penalties on terms that appear in such documents. Currently, all such counts and penalties are done plainly without normalization. Once that is achieved, we will get the top two scoring terms and add them to our query. Currently, ties are broken deterministically, simply by iterating through a sorted list storing our counts until we have found what we need, but in the future it could be feasible to randomly or arbitrarily break ties to produce better results. Nonetheless, this means of adding terms to our query has been quite good empirically from our tests. Another consideration is that the Rocchio algorithm we based our approach on includes weight parameters alpha and beta that are used to fine-tune our learning. Though we may be able to more finely weight our scoring/penalty system (instead of using raw counts) perhaps through a normalization or tf-idf scheme, the current implementation benefits from its straightforward implementation/intuition without sacrificing the integrity of its results.

The other feature we implemented is finding the optimal ordering of the query terms. We do so by iterating through all possible permutations of our query and assigning a score using a bigram model. Essentially, we will take an ordering and slide a two-word window across the potential query. We look at how often this two-word phrase appears in our relevant documents, and the two-word frequency is then added onto the score of the overall permutation. The permutation with the highest score is then chosen as our optimal ordering (with ties once again broken deterministically in a fixed order). Besides adopting a different tie-breaking strategy, other avenues of improvement could entail extending from a bigram to a larger n-gram such as (n=3), creating penalties for appearances in irrelevant documents (though this arguably isn’t as important a concern as in the word-augmentation case), and using a finer scoring

algorithm that uses more nuanced weights than the raw counts we are currently using.

At this point, we have reformulated a new query, so we will start the loop over again by running the query, receiving user feedback, and using that feedback to once again modify the query under the previous process until we have reached our desired precision@10.

JSON API Key:
AIzaSyBOskhE799tyaHkMxJc08i3YLZiJj6vubw

Engine ID:
10e635b85174848d2

Sources used:

1. https://web.stanford.edu/~jurafsky/slp3/3.pdf
2. https://www.cs.ucr.edu/~vagelis/classes/CS242/publications/jasistSalton1990.pdf
3. https://nlp.stanford.edu/IR-book/pdf/09expand.pdf
4. https://redirect.cs.umbc.edu/~ian/irF02/lectures/11Relevance-Feedback.pdf
5. https://www.cs.columbia.edu/~gravano/cs6111/Readings/singhal.pdf
6. Plus other class materials