User-Provided Relevance Feedback Information Retrieval System

COMS-E6111 Project 1

How to run

- Install program requirements with:

  ```
  $pip install -r requirements.txt
  ```

- Run the program with:

  ```
  $python3 main.py <google api key> <google engine id> <precision>
  <query>
  ```

  where:

  - `<google api key>` is your Google Custom Search JSON API Key
  - `<google engine id>` is your Google Custom Search Engine ID
  - `<precision>` is the target value for precision@10, a real number between 0 and 1
  - `<query>` is your query, a list of words in double quotes (e.g., "Aryana and Jasmine")

Description

Overall Structure

- `main(client_key, engine_key, query, precision)`
  - Entry point to the application, handling the google query iteration and collecting user input on google results
- `query_expansion(corpus, rel_idx, query)`
  - driver function for query expansion using rocchio's algorithm, handling vectorization of documents and running bigram reordering of query terms
- `rocchio(init_q_vec, doc_vecs, rel_idx)`
  - Implementation of Roccio's algorithm
- `reorder_query(related_docs, query, new_words)`
  - using bigram probability to reorder query terms

Details

Query expansion is implemented with ideas adapted from Chapter 9, "Relevance Feedback and Query Expansion," of the Manning, Raghavan, and Schütze *Introduction to Information Retrieval* textbook. We first use TF-IDF weights to create vector representations of the relevant documents and queries. We use Scikit-learn's `TfidfVectorizer` to calculate these weights.

We use then the Rocchio (1971) Algorithm as explained in the chapter to generate a modified query vector ($\overrightarrow{q_m}$):

$$\overrightarrow{q_m} = \alpha\overrightarrow{q_0} + \beta\frac{1}{|D_r|}\sum_{\overrightarrow{d_j}\in D_r}\overrightarrow{d_j} - \gamma\frac{1}{|D_{nr}|}\sum_{\overrightarrow{d_j}\in D_{nr}}\overrightarrow{d_j}$$

where q_0 is the original query vector, D_r and D_{nr} are the set of known relevant and nonrelevant documents respectively, and α, β, and γ are weights attached to each term. We use the weights $\alpha = 1$, $\beta = 0.75$, and $\gamma = 0.15$. This is because we place the greatest value on the information gained from the original query with information directly from the user. We similarly highly value the user's positive feedback on relevant documents. Since we do not have a high level of visibility into non-relevant documents, we weigh the information gained from that sector less heavily. These weights are also recommended in the chapter. The two words with the highest weights in the modified query (not including words from the previous query) are added to the new query. These terms are then reordered before a new search occurs.

Reordering of query terms is implemented using ideas adapted from Chapter 3, "N-gram Language Models," of Jurafsky and Martin's *Speech and Language Processing*. We used bigram probability to determine the best ordering of query terms. This is achieved by preprocessing the documents marked as related and removing stopwords and irrelevant characters. Then we leveraged the Natural Language Toolkit (`nltk`) to generate bigrams. While going through the document, we count up the occurrence of each word and each bigram which we can use to calculate P(word| previous_word). If there is no occurence of such bigram, we check the rest of the words and find if there exist a bigram we can generate with the remaining words in all the query terms (old + new query terms)

Imports

- `numpy`: vector space manipulation and other calculations
- `googleapiclient.discovery.build`: query Google
- `sklearn.feature_extraction.text.TfidfVectorizer`: calculate tf-idf weights, generate document vectors
- `sklearn.feature_extraction.text.ENGLISH_STOP_WORDS`: implement stop words
- `sklearn.metrics.pairwise.linear_kernel`: calculate cosine similarity
- `nltk`: used for stopword processing and bigram generation

About

Authors:

- Aryana Mohammadi (am5723)

- Jasmine Shin (yx2810)

Files included:

- `main.py`
- `README.md`
- `requirements.txt`
- `run`
- `stop_words.txt`
- `transcript.txt`

Keys:

- Google Custom Search Engine JSON API Key: AIzaSyDJZCCaWHD-kdmn1dENt6Pynp9mcykVHtpg
- Engine ID: c6f4622f0b9a14652

References:

- Chapter 9, “Relevance Feedback and Query Expansion,” of Manning, Raghavan, and Schütze’s *Introduction to Information Retrieval*

  https://nlp.stanford.edu/IR-book/pdf/09expand.pdf

- Chapter 3, “N-gram Language Models,” of Jurafsky and Martin’s *Speech and Language Processing*

  https://web.stanford.edu/~jurafsky/slp3/3.pdf

Notes:

- HTML and non-HTML files are taken into consideration for every search and precision calculation.