COLUMBIA UNIVERSITY
CS6111 ADV. DATABASE
PROJECT1

Abstract:

In this project we implement an information retrieval system that exploits user-provided relevance feedback to improve the search results returned by Google Custom Search API.

Authors:

Zhonglin Yang, zy2496
Liping Hu, lh3009

Files:

Purpose	File Name
Main program	main.py
README file	README.pdf
3 successful runs of the program	run_transcript.pdf

Credentials:

Purpose	Credentials
Google Custom Search Engine Json API Key	AIzaSyCrJrAUdogkB9X5cqO7VbJDO5IR2de_fBM
Engine Id	aa148555cb784f522

To Run Our Program:

1. Run the command line to install numpy library - pip3 install numpy
2. Run the command line to install scikit-learn library - pip3 install -U scikit-learn

3. Use terminal command cd to get into the program directory

4. Run this in terminal:

   Python3 <google api key> <google engine id> <precision number> <query term>

   Eg:

   python3 main.py AIzaSyCrJrAUdogkB9X5cqO7VbJDO5IR2de_fBM aa148555cb784f522 0.9 "cases"

Internal Design:

- General structure:

  The program has two main components, the first one is to get query results from google custom search API, display top-10 results (url, title, and description snippets), and get relevance feedback of results from users. The second one is to augment the query by using the Rocchio Algorithm according to the query, relevant results and non-relevant results. We use the Rocchino Algorithm to add two words to the current query for the next iteration.

- External Libraries:

1. googleapiclient:

   From Googleapiclient we import a module called discovery which contains a function called build. Build constructs a Resource for interacting with the API which returns us with everything we need.

2. scikit-learn

   Scikit-learn is a free software machine learning library for python.
   We used the TfidfVectorizer class to initialize the TF_IDF Vectorizer so that raw documents can be converted into a matrix of TF-IDF features. We

Query Modification:

In order to improve the final result set according to the user's input query, one of the approaches is to involve the user in the retrieval process and have the user choose if the query is relevant or not. This is the main idea of relevance feedback(RF). More specifically, the user gives feedback on the relevance of the documents returned by our program as an initial set of results. Then the following procedures are:

- The user issues a (short, simple) query.
- The program returns an initial set of retrieval results.
- The user marks some returned documents as relevant or irrelevant.
- The program computes a better representation of the information needed based on the user feedback.
- The program displays a revised set of retrieval results.

In order to improve the query results using relevance feedback, we utilized Rocchio's algorithm.

- Rocchio's Algorithm:

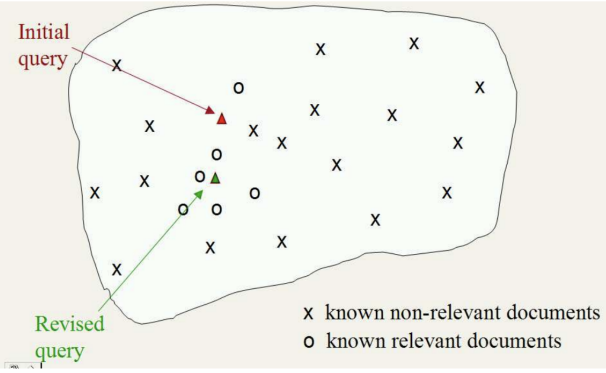

Figure 1: Some documents have been labeled as relevant and nonrelevant and the initial query vector is moved in response to our feedback program.

$$\vec{q}_m = \alpha\vec{q}_0 + \beta\frac{1}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \gamma\frac{1}{|D_{nr}|}\sum_{\vec{d}_j \in D_{nr}}\vec{d}_j$$

Figure 2: This is the underlying theory that our program is based on.

Here, q0 is the original query vector and |Dr| is the number of relevant queries and |Dnr| is the number of nonrelevant queries. Dr and Dnr are the set of relevant and nonrelevant documents

respectively. Vector dj is a single query vector. Also, α, β, and γ are weights attached to each term. Reasonable values are $\alpha = 1$, $\beta = 0.75$, and $\gamma = 0.15$ which are what we used.

In our program, we use titles and description snippets of google results as documents to augment query.

We use the scikit-learn library to deal with converting words (documents and query) to vectors. We use the TfidfVectorizer from scikit-learn to get a transformer based on TF-IDF.

Then, we use the transformer to convert the query (a string of words), the relevant and non-relevant documents (two lists of strings) to vectors.

With all the vectors we get, we can compute the augmented query vector using the Rocchio algorithm in Figure 2. Next, we inverse transform the augmented vector back to words. We use the argsort and np.filp functions to do a descending sort on the vector and get top n (the length of the old query) + 2 most possible words to construct a new query.

The new query's word order is determined by the ranking. Then, we check if all words in the old query are still all in the new query, if any of the words doesn't exist in the new query, we replace the least important words in the new query with these words.

References:

Chapter 9, "Relevance Feedback & Query Expansion," of the Manning, Raghavan, and Schütze Introduction to Information Retrieval textbook

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html