CS6111 - Advance Database Systems Project 1

Authors

Arjun Mangla (am4409) and Tomás Larrain (tal2150)

Files

Name	Usage
`project1.py`	main program
`HttpResponse.py`	Google response class
`mock_response.py`	Mock Response for offline work
`requirements.txt`	Python packages to run the project
`query_transcripts.pdf`	Transcript of required queries
`query_tests.xlsx`	Test queries and their performance compared to reference implementation

Credentials

Credential	Detail
`AIzaSyC7QIDsftl4f8lLTNwGsevas5hOOC96FZ8`	Google API KEY
`015789145925826530430:flftxrkxktl`	Search Engine ID

Dependencies

To install, run:

```
bash $ cd cs6111-project1 $ pip install -r requirements.txt
```

How to Run

Under the project's root directory, run

```
$ python3 project1.py <google api key> <search engine id> <precision> <query>
```

Note: if `<query>` has multiple words, be sure to put them between quotes (e.g. `"per se"`).

Internal Design

The program will first get arguments such as *API key*, *search engine ID*, *precision@10*, and *query terms* from user input. The code is mainly divided in 3 main parts (with many methods each): 1. **Fetch Google Results**: This is mainly done by the `get_google_results()` method that is called once per iteration. This method takes in the query and the credentials (API key, search engine id), and makes a call to the Custom Search Engine API. From the results, the formatted version of each of the results are stored as dictionary objects. The method then returns a list of these dictionaries. 2. **Get relevance feedback**: The user must divide the results between relevant and not relevant through the terminal (summarized in `get_relevance_feedback()`). Just like the reference implementation, the user is presented, one by one, with the results that were extracted using the above method. For each result, the user provides feedback as to whether it is relevant or not. With the relevance feedback, `compute_precision_10()` checks if the desired precision was achieved. If the precision has been reached, the program terminates after presenting this outcome to the user. If not, the program continues to the next step. 3. **Compute augmented query**: With the results of the relevance ready, the system procedes to compute the augmented query in `get_augmented_query()`. This method returns the full augmented query, which replaces the original one to start a new fetch and feedback loop. This loop happens a maximum number of times dictated by the MAX_ATTEMPTS variable, currently set to 10.

Query-modification Method

The query-modification method is based on the `get_augmented_query()` method of the code. It first calls sciki-learn's TfidfVectorizer [1] using the parameters `analyzer='word', stop_words='english'` to use words as the building blocks, and to remove english stop words from the vectors. Using this class, we transform our relevant and non-relevant documents corpus to a tf-idf vector, and we do the same with our current query.

Having all these 11 vectors (10 for each query result and the query itself), we proceed to implement Rocchio's algorithm. By fixing parameters `ALPHA = 1`, `BETA = 0.75` and `GAMMA = 0.15`, we compute our augmented query vector using Rocchio's equation (9.3) in [2].

With the augmented query, we go to `get_best_words()` to look for the words with the highest tf-idf index. The two highest words (that aren't already in the query), are the ones used to augment the query, and call the procedure again.

External references

[1] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

[2] Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval. Cambridge university press.