Project 1 Group 33

Rashad Barghouti (rb3074)
Parth Panchmatia (psp2137)

Project Files

- `dechi.py` — The implementation of the query expansion in Python 3.
- `README.md` : — this README file
- `stopwords.txt` — A file containing a sting of stop-words obtained from link provided in assignment description
- `keys` — a file containing that simplifies invocation of the program. It Google's API key and CSE id.

Dependencies

The following are the packages needed to run the submitted Python 3 implementation. The installation instructions for an Ubuntu VM as follows:

Python3

Python 3 should be already part of the Ubuntu VM. To check run

```
$ which python3
```

On Ubuntu 14.04 VM, this above command should return something like `/usr/bin/python3` . Nonetheless, to install or upgrade Python 3, the following sequence will put Python 3.6 on the system:

```
sudo add-apt-repository ppa:jonathonf/python-3.6
sudo apt-get update
sudo apt-get install python3.6
```

NumPy and SciPy

Python 3 versions of both can be installed/upgraded with python's `pip` command as follows:

- First upgrade `pip3` on the VM with:

  ```
  sudo -H pip3 install --upgrade pip
  ```

- To do a user-only installation, run:

  ```
  pip3 install --user numpy scipy
  # may need to add /home/<username>/.local/bin to PATH
  ```

- To install on the VM for all users, run

  ```
  sudo -H pip3 install numpy scipy

  # /home/<username>/.local/bin may need to be added to PATH after running
  # the above command
  ```

scikit-learn

To install, run

```
sudo -H pip3 install -U scikit-learn
```

- **Google API Python Client** for Python3

Python's `pip3` can also be used for this installation

```
sudo -H pip3 install --upgrade google-api-python-client
```

Running the Program

The Google API key and the custom search engine's id are:

API key: AIzaSyAlKLHe1eAmug6XeTlQ1DxzOsPI4zax7Ms
CSE ID: 006096712590953604068:qoxtr78cjow

To run the implementation, do the following:

- In a directory on the VM, uncompress the `group33-proj1.tar.gz` file

  ```
  $ tar xvf group33-proj1.tar.gz
  ```

- First ensure the Python program has its executable bits turned on. In the project's directory do:

  ```
  $ chomd +x dechi.py
  ```

- Run `dechi.py` in one of the following two ways (use the [-h | --help] to display usage info):

  ```
  $ dechi.py <google api key> <cse id> <precision@10> <query>
  ```

  or

  ```
  $ dechi.py @keys <precision@10> <query>
  ```

The `keys` file is included in `group33-proj1.tar.gz`

Code Design and the Query Augmentation Algorithm

The code has the following structure and sequence of operation:

1. `parse_cmdline()` — A simple parser that processes and validates the command-line arguments.
2. Data structure initializations in `main()` that set up an internal data structure and the search engine's service object
3. `process_query()` — a routine that (1) performs the initial Google search, (2) collects the relevance feedback from the user, and (3) computes the precision@10 value from the feedback data. It returns all relevant search results selected as relevant by the user and the top-ranked nonrelevant result.
4. `main()` — the main program component that performs the query expansion.

Algorithm and Implementation

The algorithm implemented in `dechi.py` is the Ide dec-hi variation of Rocchio work. Discussions of the algorithm can be found in chapter 9 of reference textbook by Christopher et al.: Information to Information Retrieval, as well as in the original papers by Ide E. The following are references for that work

> *IDE, E*. 1969. "Relevance Feedback in an Automatic Document Retrieval System." Report No. ISR-15 to National Science Foundation from Department of Computer Science, Cornell University, January
>
> IDE, E. 1971. "New Experiments in Relevance Feedback," in The SMART Retrieval System, ed. G. Salton, pp. 337-54. Englewood Cliffs, N.J.: Prentice Hall.

The algorithm adds to the original query the term frequencies from all relevant vectors and subtracts the term frequencies from the top-ranked non-relevant vector. The implementation is performed as follows:

- Build the index's vocabulary from both the relevant documents and the top-ranked document. From each document, the title string and the snippet string are combined to form the document's data.
- Vectorize the relevant data to form a term-document matrix based. Term frequencies were used as the index data. (Tfidf was implemented but had no advantage of tf indexing.)
- Compute cumulative term frequencies for each term in of the relevant vectors and subtract from the result the frequencies in the non-relevant vector
- Construct a list of (term, cumulative_freq) tuples.
- Sort the list in descending order on its aggregate frequency values
- While ignoring zero- or negative-frequencies, which result from the above subtraction, chose a maximum of 2 terms from the top of the sorted list to augment the query.

Additional Remarks

- Three variations of the Rocchio algorithm were implemented and tested: (1) Original query modified by positive feedback from all relevant data and negative feedback from all non-relevant data. Both feedback figures were scaled by beta and gamma values and normalized by the cardinality of their respective sets. The second (2) was an implementation with no negative feedback at all (gamma=0); and the third (3) is the Ide dec-hi presented here. There was no difference between all three, but Ide dec-hi was chosen because it is reputed to be more consistent with different types of queries

- The vectorizers are from sklearn's text feature extraction package, and both performed identically. The reasons for the lack of distinction between them are a combination of: (1) the (very) small size of the input document set and the lengths of the corresponding vectors, and (2) the short lengths of the queries (two of which are single-word terms). The first factor made document-frequency analysis and log-scaling not needed, because the index did not have any high-frequency terms that could obscure the presence of important rare terms. The second factor (the single-word queries) rendered document frequency scaling neutral. In the end, term-frequencies were sufficient to produce the high precision needed.

- Since idf scaling had no effect, stop-words removal was important to remove high-frequency words. A side effect of this was the loss of 'per' from the 'per se' query string. This, however, had no effect, since 'per' remained in the original query string, which was appended with the augmentation terms.