README.md

VSM-with-TF-IDF

An implementation of the Vector Space Model using TF-IDF

CS F469 - Information Retrieval

Domain Specific Information Retrieval

Contributors -

- Anirudh Achanta - 2016A7PS0022H
- Ashwin Chakravarti Nallan - 2016A7PS0013H
- Sasmit Mati - 2016A7PS0118H
- Pranav Taneja - 2016A7PS0096H

Language - Python 3.6.6

Requirements -

Running this program requires `nltk` to be installed on your system. Do this by - `pip3 install nltk`

Running the Program

The actual python code is stored in ./scripts, with . being the root folder of the program. To run the program, simply run the `run.sh` file in the root folder.

Once the program starts, choose the type of search you need out of the given four, Title, Author, Description and Year of Publication. And then enter your search query when prompted. The top 10 (or the most relevant if its a Year or Author search) pop up on the screen. You can either continue searching for other queries or type 'quit' (without the quotes) to exit the program.

Working

- `run.sh` calls `main.py` inside the scripts folder. `main.py` is responsible for printing the choices available to the user. `main` is also responsible for taking user input and then calls `search_result()` from `calc` module and prints the result to the screen.
- `search_result()` reads the query and option (i.e., the type of search selected) and processes it accordingly. Year of Publication search doesn't use TF-IDF to get results, it directly returns the top 10 (or less) papers published in that specific year. All other types of search follow this general pattern to process the query :-
  - Process query by using `query_vectorizer()` to tokenize and convert it into a TF-IDF vector and then normalize it by calling `normalize()`
  - For optimization, we will only be calculating the TF-IDF scores of the query tokens in the documents, and not the tokens we get from tokenizing the document. This saves time as all those terms that are not there in the query contribute nothing to the final calculation of similarity between the document vector and query vector. `doc_vectorizer()` precisely does that. It stores the TF-IDF vector of each document in a dictionary.
  - `tf` function calculates the term frequency as `1 + math.log(tf)` and the `idf` function calculates the Inverse Document Frequency as `log(N/df)`
  - We then create another dictionary that stores the `cosine_similarity()` values between the query vector and each individual document vector. This dictionary is then sorted and the top 10 results are displayed on the screen.
  - This entire process repeats for each search query till a `quit` is sent which then exits the program.