Python 1: Search Engine

Overview

The project is contained in one file: “softwareAssignment.py”. The code follows the skeleton laid out by the initial methods and comments given. The idf, tf, and tf.idf data is created or read from file in the “__init__” method. The actual searching part is done in the “executeQuery” method, which creates the query vector from the user input and returns the relevant documents. The “executeQueryConsole” method handles user input and prints out information to the user.

In lieu of vectors, data is represented within the project as dictionaries. For the purposes of the cosine similarity function, zero values are equivalent to missing value. Dictionaries are an elegant way to represent vectors instead of requiring each document vector to be as large as our vocabulary size (idf). The idf file is just a normal dictionary. The tf file is a list of dictionaries, with a dictionary for each dictionary. See below an example of its structure:

```
[
	{
		'id': 'NYT_ENG_19950101.0001',
		'words': {
			'until': 0.0425531914893617,
			'colleg': 0.0425531914893617,
			...
		}
	},
	{
		'id': 'NYT_ENG_19950101.0002',
		'words': {
			'left': 0.044444444444444446,
			'10th': 0.022222222222222223,
			...
		}
	},
	...
]
```

The data structure can be simplified to a dict of dicts, but I chose to create this as a list, because document order matters for reading and writing the contents of tf to file. The tf.idf data follows the same structure as the tf. The query vector is also ultimately converted to this structure in order to reuse the tf/idf computation methods.

Project Execution

To run the program, execute the “softwareAssignment.py” file with Python 3. To change the XML dataset, specify the name of the XML file without the extension in the SearchEngine initialization of the main method (currently “nytsmall”).

Problems

Parsing

I implemented the “xml.sax” package. 99% of the parsing was correct, but I got some weird errors. In some lines, the handler would split the line in the middle of a word. These were seemingly random. There was a split in the middle of the word “vi|brate” to “vi” and “brate” (where the separator ‘|’ is), and the other in “low-f|at” to “lowf” and “at” (“low-f” changed to “lowf” after punctuation was removed). I decided to do some simpler parsing via regular expressions instead. The solution is not as elegant nor as flexible as an XML parser, but works for the line-by-line format in the given input files.

Query vector

It was difficult to understand how to create the query vector from the instructions in the PDF file. Specifically, I was not sure how to calculate the tf.idf of the query vector. By looking up examples in other implementations[1], I learned how to create the query vector properly.

Malformed input

The user is able to type their query, allowing possibly unexpected input. I wrote code to handle most of the normal cases I could think of, but I do not explicitly sanitize user input. The user is assumed to input valid words.

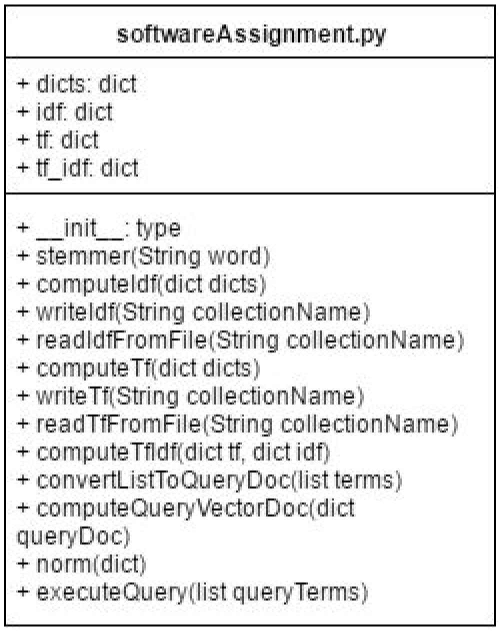
Verifying results

I used the provided PDF to verify my answers, but I also compared results with my friends also working on this project to ensure we arrived at the same result. We compared tf and idf files, and diff’ing the content of these files using diffchecker[2] was very useful. I detected problems with my files by doing this and am thankful that I could compare my results with my peers. I would not have found these problems otherwise, since my numbers already matched the PDF examples.

---

[1] http://www.site.uottawa.ca/~diana/csi4107/cosine_tf_idf_example.pdf
[2] https://www.diffchecker.com/diff

UML Class Diagram



softwareAssignment.py

+ dicts: dict
+ idf: dict
+ tf: dict
+ tf_idf: dict

+ __init__: type
+ stemmer(String word)
+ computeIdf(dict dicts)
+ writeIdf(String collectionName)
+ readIdfFromFile(String collectionName)
+ computeTf(dict dicts)
+ writeTf(String collectionName)
+ readTfFromFile(String collectionName)
+ computeTfIdf(dict tf, dict idf)
+ convertListToQueryDoc(list terms)
+ computeQueryVectorDoc(dict queryDoc)
+ norm(dict)
+ executeQuery(list queryTerms)