**Search-Retrieval System**

**Project Summary**:

Project includes the implementation of basic Search-Retrieval System based on the topics included in CSC# 575.

Project involves creation of term indexes using TF.IDF weights and matching documents and query using cosine similarity. Application also provides two basic UIs for testing the engine.

**Deliverables**:

1. Project Report
2. Compiled Code
3. Test Collection - <http://ir.dcs.gla.ac.uk/resources/test_collections/time/>
4. Output Files Generated for Test Collection

**Application Description:**

Application is developed in Python 3.5 and uses NLTK package. To run the application, please make sure to install nltk library.

**Files**:

1. Constants.py → Declares the constants used in the application
2. DocumentReference.py → Includes the DocumentReference class, which stores information about the document
3. TokenInfo.py → Includes the TokenInfo class, which includes information about the tokes
4. IIREngine.py → Includes the Engine class, which is the heart of the application. This class has methods which creates the indexes and processes queries
5. Start.py → File provides a simple text based UI for running the application. This can be used for executing one query at a time.
6. testBot.py → File execute the queries in batch. This program requires a query file.

**Important Data structures**:

1. docDictionary → {Absolute Path: DocumentReference Object}
2. tokensDictionary → {token : TokenInfo Object}
3. dictionary → {token: {Absolute path : Term Frequency In Document} }

**How to execute**:

1. Unzip the submitted file
2. For Start.py:
   1. Open Start.py in Idle and run the module
   2. In Shell, run function start ()
   3. Follow all the instructions
3. For testBot.py:
   1. Open testBot.py in Idle and run the module
   2. In Shell, run function startTest (queryFilePath, docPath)
      1. queryFilePath → Absolute Path where query File is stored
      2. docPath → Path where all the documents are stored
   3. Follow the instructions
   4. At the end of execution testOutFile.txt will be generated in the executing folder

**Test Collection**:

Application requires that the documents in the document collection should be in text format and should be stored locally.

For the testing the application I have downloaded the test collection from: <http://ir.dcs.gla.ac.uk/resources/test_collections/time/>

In the collection, all the documents are in the same file. So I have written a simple Python script createsfile.py, to parse the file and generate different files from the master file. Following document structure would be found in the zipped folder:

1. Source folder/TestCollection/OrginalCollection → Contains downloaded test collection
2. Source folder/ Files → Contains document created by the createsfile.py from the master document
3. Source folder/Query → Contains query.txt that contains 83 queries – one query per line. This could be used for batch testing.

**Test Results**:

I have tested the developed application on the test collection mentioned above and have analyzed results of first 20 queries but the result of rest of the 63 queries is in the file testOutFile.txt.

Output File Folder Path: Source Folder/ OutputFiles

(Files will be generated at the execution path. I have copied the files in to a separate folder)

Following is results for the 20 queries:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Query | Total Retrieved | Relevant Retrieved | Relevant Not Retrieved | Precision | Recall |
| 1 | 154 | 2 | 5 | 0.013 | 0.286 |
| 2 | 287 | 2 | 0 | 0.007 | 1.000 |
| 3 | 271 | 2 | 2 | 0.007 | 0.500 |
| 4 | 239 | 1 | 4 | 0.004 | 0.200 |
| 5 | 134 | 2 | 3 | 0.015 | 0.400 |
| 6 | 205 | 2 | 7 | 0.010 | 0.222 |
| 7 | 293 | 2 | 0 | 0.007 | 1.000 |
| 8 | 210 | 1 | 1 | 0.005 | 0.500 |
| 9 | 124 | 0 | 8 | 0.000 | 0.000 |
| 10 | 423 | 6 | 0 | 0.014 | 1.000 |
| 11 | 299 | 2 | 0 | 0.007 | 1.000 |
| 12 | 210 | 1 | 7 | 0.005 | 0.125 |
| 13 | 315 | 2 | 1 | 0.006 | 0.667 |
| 14 | 423 | 1 | 0 | 0.002 | 1.000 |
| 15 | 250 | 3 | 2 | 0.012 | 0.600 |
| 16 | 345 | 1 | 2 | 0.003 | 0.333 |
| 17 | 97 | 1 | 1 | 0.010 | 0.500 |
| 18 | 342 | 1 | 0 | 0.003 | 1.000 |
| 19 | 333 | 3 | 2 | 0.009 | 0.600 |
| 20 | 299 | 1 | 0 | 0.003 | 1.000 |

Table Analysis of First 20 Queries

Recall values are in acceptable range as the retrieval system is picking up major chunk of the documents. As the search system is using cosine similarity between query vector and term vector, there are a lot of documents that are matching for larger queries. The relevance judgment values might be based on the context analysis of the queries, which I am not considering here. This might be reason we can a major impact on precision values.

Apart from the standard test collection, I have tested the application on sample test collection I created. Application provides corrects results for the queries I tested it for.