**Assignment 1: CSE 494/598**

**By: Udai Arora**

**4 points: Submit a print out of the top 10 documents ranked by Vector Space model (once for task 1, and once for task 2) for the 5 example queries below. Print ONLY the document numbers.**

“grades”

|  |  |
| --- | --- |
| tf  22149 | Tf-idf  22156 |
| 22156 | 233 |
| 233 | 22149 |
| 22919 | 19822 |
| 18699 | 19590 |
| 1851 | 22913 |
| 19590 | 21047 |
| 18750 | 1851 |
| 22913 | 22936 |
| 19822 | 18699 |

“newsletter”

|  |  |
| --- | --- |
| tf  1701 | Tf-idf  1701 |
| 1718 | 13778 |
| 13778 | 2161 |
| 1309 | 21523 |
| 16653 | 1238 |
| 2161 | 1309 |
| 21523 | 1718 |
| 18378 | 16653 |
| 1287 | 18378 |
| 1238 | 18372 |

“carl hayden”

|  |  |
| --- | --- |
| tf  21827 | Tf-idf  15128 |
| 16874 | 16676 |
| 16276 | 17678 |
| 15128 | 15130 |
| 16689 | 16276 |
| 17429 | 16904 |
| 16911 | 21827 |
| 18362 | 18362 |
| 16853 | 17429 |
| 18343 | 15114 |

“fall semester”

|  |  |
| --- | --- |
| tf  845 | Tf-idf  871 |
| 882 | 5 |
| 103 | 103 |
| 20258 | 3762 |
| 3595 | 30 |
| 30 | 882 |
| 88 | 79 |
| 275 | 21231 |
| 3599 | 20258 |
| 871 | 845 |

“stimulant web”

|  |  |
| --- | --- |
| tf  13973 | Tf-idf  26 |
| 14348 | 2085 |
| 26 | 18590 |
| 10895 | 2094 |
| 13988 | 13973 |
| 13984 | 14348 |
| 10768 | 18600 |
| 10749 | 13948 |
| 13948 | 18602 |
| 10750 | 13988 |

**2 points: A few sentences explaining your algorithm.**

Hash Map and arrays are the main data structures used in the program- idf, query words, result and sorted results are essentially stored in a map. The pre-computation begins by going over each term in the reverse index. For each term, every document corresponding to that term is visited and the “un-squarerooted” |D| for tf and idf is calculated. After the outer loop completes, the values are square-rooted to get the exact results. Now the query is inputted along with the type of similarity calculation to be performed. The appropriate function is then called to calculate the similarity. The result from the function is stored in a map. The function to sort a map is then called on this map. The output is displayed to the user using a print function.

The formula used for calculating-

|D| (for tf): (tf12+ tf22+ tf32+ tf32+ tf42+ tf52+….)1/2

|D| (for tf-idf): ((tf1 \*idf1 )2+(tf2 \*idf2 )2+(tf3 \*idf3 )2+….)1/2

|Q| = (w12+w22+…)1/2

The tf\_calculator function loops over every result document for each word w in query. The tf-similarity, idf-similarity is calculated as:

*tf\_similarity*=(**double**)tdocs.freq()/((**double**)*mod\_q* \* (**double**)*mod\_d\_tf*[tdocs.doc()]);

*tfidf\_similarity=(****double****)tdocs.freq()\* idf /((****double****)mod\_q \* (****double****)mod\_d\_idf[tdocs.doc()]); //where tdocs.freq=tf*

(tdocs.freq() gives the term freq for that word. |Q| and |D| are precomputed)

**5 points: How long did it take to compute document norms? How long did it take to get the results? To sort them?**

**The time taken to get the document norms is almost constant for every run. The average time taken to compute the document norms**: 4.65544 seconds [Range: 4.62331 to 4.69884]

**Sort Times:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | grades | newsletter | carl hayden | fall semester | stimulant web |
| tf | 2.616809 | 2.812247 | 13.093881 | 10.202947 | 19.006613 |
| idf | 2.722012 | 2.64033 | 13.681049 | 11.211781 | 19.626282 |

**Total Times:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | grades | newsletter | carl hayden | fall semester | stimulant web |
| tf | 3.498202 | 3.70091 | 19.233697 | 14.795233 | 24.106803 |
| idf | 3.794138 | 3.659427 | 19.091288 | 16.248679 | 26.49824 |

**5 points: Plot a bar chart showing the time taken to perform a search of the five queries above for TF and for TF/IDF. Is the difference significant? Is it expected?**

The query terms are shown on the left and the time in millisecond is shown in the lower axis. The graph is constructed from the values given in the last answer.

As we can see above, the difference is not significant. The time taken to calculate TF-IDF is very similar- although, a little more than the time taken to calculate TF. **This is expected** as the algorithm to compute IDF is exactly same as the one for TF- except the fact that there is an extra multiplication involved in calculating idf based similarity (multiplying tf by the idf).

**5 points: Plot a chart showing the time taken to perform a search vs the number of words in the query. Explain the result.**

The graph is shown below:

As the number of words in the query increase, the time increases. After a certain value (3 in this case), most of the documents would be covered (by the 3 words- “arizona”, “state” and “university” in this case). The document array would already be in the memory cached when the latter query words try to access them- and hence it stabilizes later due to faster reads.

**(Extra credit 3 points): Plot a curve showing the time taken to do a query vs the number of results in the query. To do this question, you will need to figure out query keywords of your own that have various number of results, so that you can plot a meaningful curve.**

The time taken to calculate the similarity of the query vs the number of documents is shown below.

As we see, the result is intuitive and expected- time taken increases with the number of results for that query.

**(Extra credit 3 points): Artificially restrict the size of the document set by ignoring any documents with id above a certain number 'n'. Plot the time taken to do the query against n. Explain the result.**

The graph is shown below:

Since we have to loop through lesser documents if we restrict the size of the corpus to a lower value, the graph is of a positive slope and is linearly increasing.

**(Extra credit 3 points): Provide a theoretical complexity analysis of the algorithm.**

For Pre-computing |D|: The outer loop traverses through each of the terms in the reverse index. The inner loop goes over the document indexed to that term for each term. Assuming that the get and put functions will have a constant time, the worst case, if the total number of indexed terms are t and total number of documents are d, the worst case complexity O(t\*d)

After the pre-computations, the query is input. Suppose there are q words in the query. Now the complexity creating a hash map for the query to calculate |q| will be O(q) as it takes constant time to put values into the hash table.

This is followed by a call to the function which calculates the TF or the TF-IDF. For each word in the query with the total number of word=q, it goes over the documents containing that word and calculates the tf/tf-idf. Assuming total docs is d, this has a complexity of O(q\*d)

The function for sorting takes O(s2) time if there are s entries in the parsed hash map. Similarly, the function to print will take linear time O(p).

Since t\*d would be much much greater than the other worst cases, the program has a worst case of O(t\*d). (This can be verified by the fact that pre-computation takes 5 to 6 seconds and getting, sorting and printing results are in the order of milliseconds.)

**5 points: Are the TF results relevant, as judged by a human? Are the TF/IDF results relevant? Which is better?**

The analysis of this part was done using the sample query “downtown campus degree type”.

For TF, on doing a manual analysis of the first few results, the third result seems less relevant as compared to other top 10 results as it just gives the tour of downtown campus and doesn’t talk about the degrees.

The top 10 results from TF-IDF look to be equally relevant- they have the query words and in the same context.

Comparing the results from TF and TF-IDF, the first 2 results are the same. The 3rd IDF result is the 4th one for TF. Comparing the 3rd result for TF and TF-IDF, the TF-IDF one seems more appropriate. The TF one gives a tour of the downtown campus whereas the IDF one talks about the inauguration of the downtown campus which includes what kind of degrees they planned to offer at that time.

TF is also missing a very relevant result in top 10, which occurs 7th for IDF- about the University Center of the downtown campus- which lists all the degrees the campus is offering.

Hence we see that the results by IDF are more relevant, judging from a user’s perspective.

**5 points: Compare TF vs TF/IDF: Is the order of the results different with a single keyword? two keywords?**

The order of results is different for tf and tf-idf for both single word and multi-word queries. Consider any single word query “xyz”. The only difference between calculating tf and idf similarity besides multiplying by idf of the query term in the numerator is the calculation of |D| :

|D| (for tf): (tf12+ tf22+ tf32+ tf32+ tf42+ tf52+….)1/2

|D| (for tf-idf): ((tf1 \*idf1 )2+(tf2 \*idf2 )2+(tf3 \*idf3 )2+….)1/2

These |D| will obviously have different values. As a result, tf will differ from idf by (idf/|D|for idf ) – (1/|D|for tf)

This argument can be extended for queries containing more than 1 words also.

**5 points: Which term(s) have the lowest IDF in the corpus? Is this expected?**

The lowest IDF is for the term “html” is 5.5894919010238E-4. The low idf is because it occurs in 25040 out of 25054 documents.

This is an expected result as almost all of the documents are html documents (25040 out of 25054) and contain the html tag(<html>). This is verified by opening many docs which are indexed Hence it has a very minimal idf as expected.