**[COMP6490] Document Analysis**

**Assignment 1**

*Semester 2 - 2023*

**Name :** Huy Hoang Hoang

**University ID** : u7671528

# Question 1

## Methods

Generally, there are 3 phases in my approach for Boolean Retrieval problem:

**Parse Query**

**Retrieve Posting Lists**

**Commence Query Operations**

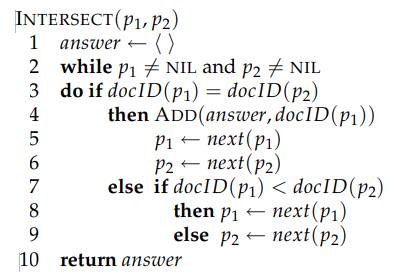
**Parse Query**: Initially, the query string is tokenized using whitespaces. And then, all tokens are separated into **terms[]** (for query terms) and **ops[]** (for operations like AND, OR). Each term is also converted to lower-case form. Furthermore, the query is also checked to ensure its validity.

**Retrieve Posting Lists**: Look for terms obtained from the previous phase in the index. As lookup of each team is independent, a thread pool is used to enhance execution time.

**Commence Query Operations**: All operations have the following form:

And as there is neither precedence nor bracket, we can execute the whole query using the following algorithm:

We can implement our code for **AND** operation based on the following algorithm:



**Figure 1** Algorithm for finding intersection of 2 lists o1 and p2. (Christopher D. Manning, et al., 2008)

For **OR** operation, our implementation does not change much. However, it is important to note that while the method for **AND**  operation returns as soon as we reach the end of a list, the method for **OR** must keep running till ends of both lists have been reached.

## Testing

File **tests/query\_boolean\_test.py** is for testing **union\_query()**, **intersect\_query()** and **parse\_query()**.

Below test cases are used:

|  |  |  |
| --- | --- | --- |
| **uniqon\_query()** | **intersect\_query()** | **parse\_query()** |
| * Normal case * Both lists are empty * One of the lists is empty * Both lists are the same * Overlapping elements at a list's ends * Overlapping elements at a list's middle | * Normal case * Both lists are empty * No intersect * One of the lists is empty | * Normal case * Operation at the query’s start * 2 operations next to each other * 2 terms next to each other * Only operations * Only terms |

Figure 1.1 Testcases for Boolean queries

## Results

Applying the method above, we get following results (**Table 1.2**):

|  |  |
| --- | --- |
| **Query** | **Result** |
| Workbooks | ./gov/documents/14/G00-14-2198849  ./gov/documents/36/G00-36-2337608  ./gov/documents/50/G00-50-0062475  ./gov/documents/69/G00-69-1400565  ./gov/documents/69/G00-69-2624147 |
| Australasia OR Airbase | ./gov/documents/21/G00-21-0639911  ./gov/documents/35/G00-35-2345443  ./gov/documents/92/G00-92-1917825  ./gov/documents/92/G00-92-3147280 |
| Warm AND WELCOMING | ./gov/documents/63/G00-63-2944034  ./gov/documents/97/G00-97-2878104 |
| Global AND SPACE AND economies | ./gov/documents/26/G00-26-1275394  ./gov/documents/42/G00-42-0794402  ./gov/documents/56/G00-56-3884873  ./gov/documents/60/G00-60-1685015  ./gov/documents/70/G00-70-0584855  ./gov/documents/70/G00-70-2258666  ./gov/documents/84/G00-84-0274223  ./gov/documents/96/G00-96-2539022 |
| SCIENCE OR technology AND advancement AND PLATFORM | ./gov/documents/29/G00-29-3700059  ./gov/documents/94/G00-94-0708359  ./gov/documents/99/G00-99-0289351 |
| Wireless OR Communication AND channels OR SENSORY AND INTELLIGENCE | ./gov/documents/33/G00-33-2857182  ./gov/documents/49/G00-49-0055139  ./gov/documents/65/G00-65-2419666  ./gov/documents/68/G00-68-4089689  ./gov/documents/84/G00-84-1559810 |

Table 1.2 Results for queries in Question 1

# Question 2

## Methods

The function for calculating IDF **calc\_idf()** has been implemented as it is used multiple times in this question. This promotes both code reusability and maintainability. If we need to use another variant of IDF, we just need to modify **calc\_idf()** method instead of modifying the whole **query\_tfidf.py** file.

Following is formula used to calculate TF-IDF. Suppose that the document is consisted of terms , , …, , is the term frequency of term in document , is the inversed document frequency of term in the corpus.

Then norm of can be obtained by following formula:

And similarity between and is given by:

## Testing

File **tests/query\_tfidf\_test.py** serves the purpose of testing **calc\_idf()** and **get\_doc\_to\_norms**(). Function **calc\_idf()** is tested against below cases:

* Normal case
* The term appears in all documents in the collection.
* The term cannot be found in the collection.
* The term appears in almost all documents (N – 1).

Function **get\_doc\_to\_norms()** is tested using the following index (**Table 2.1**):

|  |  |  |  |
| --- | --- | --- | --- |
|  | **car** | **insurance** | **auto** |
| **doc1** | 1 | 0 | 3 |
| **doc2** | 5 | 0 | 0 |
| **doc3** | 0 | 1 | 2 |
| **doc4** | 0 | 0 | 0 |

Table 2.1 Index used for testing TF-IDF cosine similarity.

## Results

We get top-5 most relevant documents corresponding to each query (**Table 2.2**):

|  |  |
| --- | --- |
| **Query** | **Results** |
| Is nuclear power plant eco-friendly? | ./gov/documents/76/G00-76-3273936 0.5802  ./gov/documents/30/G00-30-1518511 0.4348  ./gov/documents/01/G00-01-1806077 0.4302  ./gov/documents/55/G00-55-0690938 0.4136  ./gov/documents/72/G00-72-1085257 0.3936 |
| How to stay safe during severe weather? | ./gov/documents/99/G00-99-3847208 0.2812  ./gov/documents/38/G00-38-1132272 0.1961  ./gov/documents/75/G00-75-0577710 0.1534  ./gov/documents/11/G00-11-0266936 0.1382  ./gov/documents/66/G00-66-4088870 0.1356 |

Table 2.2 Results for queries in Question 2

## Evaluation

Evaluation result:

**map** : 0.2543

**Rprec** : 0.2423

**recip\_rank** : 0.4549

**P\_5** : 0.2467

**P\_10** : 0.1867

**P\_20** : 0.1200

Performance comparison between using TF and TF-IDF:

A graph of a performance metrics comparison

Description automatically generated

Figure 2 TF vs TD-IDF

Overall, the system with TF-IDF outperforms its TF counterpart as it overcomes in almost all metrics, especially in the **MAP** metric, which indicates the overall performance of a system. In terms of Recall, the difference between the 2 system is not clear (**R-Precision** and **Reciprocal Rank**). Focusing on Precision, the system with TF-IDF proves to be better at precision-related performance. We see that in the Top-K metrics (**P5**, **P10**,and **P20**) the system using TF-IDF performed considerably better than the system using TF. Especially, the Top-K improvement is inversely proportional to rank K.

# Question 3

## Choosing Metrics

Evaluation results provide following metrics:

* MAP
* R-Precision
* Reciprocal Rank
* P5
* P10
* P20

In this problem, we are working with a ranked retrieval system, so we put priority on how quickly we receive the first relevant result, and we also expect our system to have a high precision. Thus, MAP, which indicates the overall performance of a system, and Reciprocal Rank (Inverse of the rank that the first relevant result appears. In other words, this metrics indicates a system’s ability to quickly return a relevant result) are used as the main metrics.

## Employed Modifications

In this part, I will demonstrate how I gradually improved my token processing function, leading to a significant improvement in overall performance of the information retrieval system (**Figure 4** and **Table 3.1**).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | MAP | R-Precision | Reciprocal Rank | P5 | P10 | P20 |
| Original | 0.2224 | 0.2153 | 0.4331 | 0.2067 | 0.1533 | 0.11 |
| Modification 1 | 0.2343 | 0.2095 | 0.4135 | 0.2133 | 0.1567 | 0.11 |
| Modification 2 | 0.2515 | 0.2250 | 0.4729 | 0.2267 | 0.1700 | 0.10 |
| Modification 3 | 0.2583 | 0.2250 | 0.4979 | 0.2267 | 0.1767 | 0.1033 |
| Modification 4 | 0.2494 | 0.2183 | 0.4589 | 0.2333 | 0.1700 | 0.1050 |
| Modification 5 | 0.2406 | 0.2253 | 0.4934 | 0.2133 | 0.1733 | 0.1183 |

Table 3.1 Evaluation results of all modifications

A graph of different colored bars

Description automatically generated

Figure 3 Evaluation results of all modifications

**Original**

This system just uses a simple processing method of removing stopwords and converting each token into the lower-cased form. Hence this is also the system whose overall performance is worst (**Figure 4** and **Table 3.1**).

**Modification 1**

In the first modification, I tried to improve the performance by employing Porte Stemmer, which is the most popular stemmer. Apart from that, stopword removal and case-lowering was kept. However, these was only a marginal improvement in the performance of the system, let alone there were some metrics that indicated a performance worse than that of the original system (**Figure 4** and **Table 3.1**).

To pinpoint the root cause of this poor improvement, I investigated the index data and found following points:

* A large portion of tokens are emails and websites.
* Most tokens appear with punctuation marks (ex: “natural”?: ; “search)
* Some tokens appear with non-ASCII characters (ex: •bureau, •business)

Apparently, stemmers are unable to clear those marks or non-ASCII characters from stemmed tokens.

These points leaded to my second modification, which further improves the token processing phase.

**Modification 2**

In this modification, I attempted to resolve issues found in the first modification by filtering out unwanted characters.

Firstly, all non-ASCII characters will be removed.

Secondly, regular expressions are used to pick emails or websites from containing token (ex: [**?\_data\_analysis@anu.edu.au\_\_**](mailto:?_data_analysis@anu.edu.au__) would become [**data\_analysis@anu.edu.au**](mailto:data_analysis@anu.edu.au)). It is of no value to put emails and websites found in this stage under stemming process.

Lastly, occurrences of punctuation marks in each token will be removed. Specifically, following characters will be removed: . **? ! , ; : \ ‘ “ ( ) [ ] { } ‘s \_ ;** .

Changes applied in this modification brought about a considerably better result in all metrics (**Figure 4** and **Table 3.1**).

**Modification 3**

Here I tried another stemmer to see if the performance could be further improved.

Martin Porter, author of Porter Stemmer, claimed that Snowball Stemmer – sometimes considered as Porter Stemmer 2, had better performance, and should be used for practical work (Porter, 2006).

Thus, I changed from Porter Stemmer to Snowball Stemmer while retaining the filtering part. As anticipated, evaluation result once again showed a significant improvement comparing to the original, especially in Reciprocal Rank (**Figure 4** and **Table 3.1**).

**Modification 4**

This is an experiment with a rather unpopular stemmer – Lancaster Stemmer. Evaluation result presented inconsistence because while it was superior to all other modifications, except for Modification 3, in almost all metrics, its Reciprocal Rank dropped to 0.4589, which was even lower than that of Modification 2 (0.4729).

(**Figure 4** and **Table 3.1**).

**Modification 5**

Only filtering part was kept in this modification. It is clearly that this modification would not be able to surpass stemmer-integrated modifications. However, it had a good MAP (0.2406) and Reciprocal Rank (0.4934) – which is the second highest.

You may have a question about why lemmatization is not used here. The answer is very simple: Lemmatization requires the context (which is the whole sentence). In this case, all sentences have been tokenized before, so it is not appropriate to use lemmatization here.

## Conclusion

Looking at **Figure 3** and **Table 3.1**, we can conclude that the best modification is **Modification 3** since it showed consistent improvement in multiple metrics, especially in MAP and Reciprocal rank. However, when indexing time is prioritized, **Modification 5** might be a potential candidate as it does not use any stemmers, hence processing time is largely improved. Moreover, **Modification 5** also has an excellent Reciprocal Rank, which is crucial to ranked retrieval.

# References

* Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze, 2008. Algorithm for the intersection of two postings lists p1 and p2. In: *An Introduction to Information Retrieval.* s.l.:Cambridge University Press, p. 11.
* Porter, M., 2006. *The Porter Stemming Algorithm.* [Online]   
  Available at: https://tartarus.org/martin/PorterStemmer/index.html  
  [Accessed 8 2023].