Matthew Bejtlich and Tong Zhang
2040
Part 2
Report

1.) Can inverse document frequency equal zero? Why or why not? (2 sentences)

    Yes, the inverse document frequency can be equal to zero. If a single word appears in every document of the corpus, then the result of log_e(total number of words/number of documents with term t in it) is equal to 0.

2.) Describe the format of your inverted index justifying your choice. You should use Big-O analysis in your argument. (4 sentences)

    I used a nested dictionary structure (dictionary in a dictionary). A python dictionary is a hashmap, with worst-case O(n) and average time complexity O(1). The dictionary provides an efficient way to read from memory and access positing lists corresponding to specific words in the corpus. The outer dictionary key represents a stemmed word, while the inner dictionary holds the document ID and a list of word positions in the document. The inverted index was stored in a JSON file format, because it provided me with a highly structured data file that I could easily read in my query.py program.

3.) What is the time and space complexity of your create.py program? Explain your answer. (5 sentences)

    The time complexity is O(*T*), where *T* is the total number of terms in the corpus (entire XML page). This alternatively can be represented by O(*n***p*), where *n* is the number of words on a specific page in the XML file and *p* is the number of terms in that specific page. To parse and tokenize the terms in a given XML document, we first enumerate across each page in the corpus in a for loop, and then extract a block of text associated to that page in another for loop by using the iter method. The space complexity of the create.py program is *k***l*, where *k* is the total number of words in the dictionary, and *l* is the length of the posting list associated to each word at a given page id.

4.) What changes to your program would you need to make if your query.py can not load the entire inverted index into memory? (3 sentences)

    If the inverted index were too large to load in the query.py file, I would break it apart into multiple smaller dictionaries. I would have a dictionary for each word in the alphabet (26 total), and all of the terms starting with the respective letter would go into the dictionary (arranged by alphabetical order). When given a specific query, I would only read the dictionary into memory corresponding to the first letter of the query.

5.) What aspects of tf-idf make it useful for ranking search results? How would you modify our tf-idf ranking algorithm if boolean queries contained NOT's? (7 sentences)

The tf-idf is a helpful metric for specifying how important a word is to a document contained within a larger corpus. Higher scores are given to words that are especially unique to certain pages, while lower scores are attributed to words that appear frequently across all pages. Using the tf-idf metric is especially usual for handling multiple word queries. In this case, we use cosine similarity and form a vector space model for the documents and the query. The document vectors are composed of tf scores only corresponding to each query term, and the query vector is composed of idf values for each query term. The cosine similarity score (between 0 and 1) represents the relevancy of that document given the query (with 1 being the highest score). I would modify the tf-idf ranking algorithm by assigning lower tf scores (e.g. negative value) to query terms that the "not" is affecting. For instance, with the query "snow AND not white," I would penalize the only the tf terms corresponding to white. The documents with lower proportions of white would score higher, and the documents with higher proportions of white would score low. Alternatively, I could make "NOT" a Boolean operator, and have that condition taken into account in the boolparser function. In this way, I would not have to adjust the tf-idf scores already computed and stored in the inverted index.

6.) Try your search engine on the given collection with your own queries, a few of each type. Which type of queries does it perform best/worst on, in terms of result relevance? How does this depend on the query words (common/uncommon/specific/etc.)? Provide at least 2 examples for each query type, one positive and one negative

One word:

nemo

```
File:Nemo.jpg: 1.8937609297970355
Nemo (character): 1.8937609297970355
File:Finding nemo marlin nemo.jpg: 1.5150087438376285
Category:Finding Nemo: 1.262507286531357
File:Album-nemo.jpg: 1.262507286531357
```

toystory

```
File:Title-toystory.jpg: 2.0525885336873104
File:Album-toystory.jpg: 2.0525885336873104
File:Credits-toystory.jpg: 2.0525885336873104
File:Voice-toystory.jpg: 2.0525885336873104
File:Toystory severalcharacterfrom12ad3.jpg: 2.0525885336873104
```

I am using a normalized tf method for one-word queries. “Toystory” and “nemo” appear to return document titles that represent images. This is expected, since documents that have fewer words will give a greater normalized tf-weight compared to documents that are larger. For instance, “nemo” will get a higher weight if it is in a document of 3 total words, compared to a document of 100 total words. In this case, it would be helpful to filter out documents that contain .jpg in the title. I think the “nemo” query performs better than the “toystory” query, because it returns two documents that don’t have jpg.

Free text:

finding nemo

```
User blog:Lightening McQueen/Finding Nemo 2?: 0.999998522843486
Tad: 0.9999904457723348
Finding Nemo: Losing Dory: 0.9999904457723348
Anglerfish: 0.999954716832774
2003: 0.999954716832774
```

actor nemo

```
User talk:Aleal: 0.9998805100543425
Monsters, Inc.: 0.9998805100543423
Ellen DeGeneres: 0.9998805100543423
Austin Pendleton: 0.9998805100543423
Carl Fredricksen: 0.9998805100543423
```

Based on these two results, it seems that “finding nemo” returns the more relevant results. For instance, Tad is a fish in the movie. “2003” is the year of the release. “Actor nemo,” provides less relevant results, because the actor who played the voice of Nemo, Albert Brooks, isn’t contained within the first 5 results. Rather, Ellen DeGeneres, who played Dory, is in the third position. Possibly, DeGeneres is more famous than Brooks, and that is why she is mentioned more frequently in the documents.

Phrase:

“Buzz lightyear”

```
Cars Trivia: 0.9994295871849915
42: 0.9994295871849915
7597 Western Train Chase: 0.9994295871849915
Toy Story Midway Mania!: 0.9994295871849915
User: The Buzz Lightyear: 0.9994295871849915
```

“Lightyear Buzz”

```
42: 0.9994295871849915
User:The Buzz Lightyear: 0.9994295871849915
Up Trivia: 0.9994295871849914
```

User:Dhamma1000: 0.9994295871849914
User blog:Gameboyz829/The Garage Sale: 0.998309002224471

The results for “Lightyear Buzz” appear to be far less relevant than “Buzz lightyear.” The first search result is “42,” which isn’t a strong result. No document results yield direct association to the movie Toy Story. In contrast, “Buzz lighyear” yields 2 results in the top 5 directly associated to Toy Story. I would expect there to be a proportion of articles related to Toy Story at the top of the results list. Since “Lighyear buzz” is more uncommon than “Buzz lightyear,” this could be a reason why the search results are worse. “Buzz lighyear” yielded far more search results.

Boolean:

nemo AND finding

User blog:Lightening McQueen/Finding Nemo 2?: 0.999998522843486
Tad: 0.9999904457723348
Finding Nemo: Losing Dory: 0.9999904457723348
Anglerfish: 0.999954716832774
2003: 0.999954716832774

nemo AND buzz

Pixar Wiki: 0.9999802943880018
Buy n Large: 0.9999802943880018
John Lasseter: 0.9999802943880018
WALL•E Trivia: 0.9999802943880018
The Art of Ratatouille: 0.9999802943880018

The first query, “nemo AND finding,” returns more relevant search results than the second query, “nemo and buzz”. All five of the results are specific to the movie “Finding Nemo.” With the second query, I am selecting a character from one Pixar movie and a character from another Pixar movie. Thus, I would expect more general search results that mention several movies of the Pixar franchise. This is exactly what I am observing from the results. The third result is the chief creative officer of Pixar, John Lasseter. The 4th and 5th queries are not relevant to the query tokens of nemo and buzz, however. Likely, this combination of words in a document is more rare than with the first query, “nemo AND finding.”