CMSC 676 Information Retrieval - Phase 2 project report

Mihir Kelkar

University of Maryland, Baltimore County

Background on Phase One (Pre-processing): In Phase one of the project we dealt with tokenizing HTML. A corpus of 503 html files was provided which was to be tokenized. In the first phase, I had implemented the tokenizer by using regular expressions to identify parts which matched text patterns in HTML. The main principle used in the first tokenizer was to use HTML's structure to help identify text, then filter identified text by clearing out punctuation marks, digits, special symbols and non-ascii characters. After the process of clearing the text is completed, the tokenized text for an input HTML file was written to its corresponding output file.

Improvements in the pre-processing phase : Instead of trying to identify text from HTML , I decided to use the HTML Parser library in Python to simply clear out everything which matches html tags and syntax constructs, thus leaving the page with text and numerals and symbols. Then I filtered the remaining of the corpus and produced pure text not containing numbers or punctuation marks. I however decided to keep some special characters like é since the entire corpus is not in English. By leaving out special characters from Spanish and other identifiable languages, I could parse the corpus better.

Introduction to the concept of term frequency, inverse document frequency: The, tf-idf for a term is a numerical description of how important a term is to a document in a given corpus. The tf-idf value of a word is directly proportional to the number of times a word appears in a document, but it has to be normalized with the length of the documents as well as considering the occurrence of that word in other documents in the same corpus. A high tf-idf value for a term in a given document indicates that if a query contains that term, then the document is more likely to be returned as a result. Hence, very common words will have a low tf-idf value. Whereas, words which are unique to a particular document will have a high tf-idf value.

In this TF-IDF scheme, if a word occurs in every document in the corpus the IDF value turns out to be zero. Thus, it is representative of the fact the particular term is practically useless since it can never isolate a document for query and search purposes. However, if a given term is frequent in a document and occurs only in that document, it will have a high TF-IDF value and will be able to isolate the document as a potential result during a search.

Description of TF-IDF scheme used in the code :

Term Frequency

For a given token t, f(t,d) represents the number of time a the token t occurs in the document d.

f(t, d) = frequency of t in document d.

However,considering non-normalized term frequency has an inherent bias towards longer documents. Hence, the length of the document should be used to offset the value.

Lets call this normalized term frequency as augmented frequency :

tf (f, d) = **0.5 + ((0.5 + f (t, d)** / ***Max frequency any term in the document***)

Inverse Document Frequency

The inverse document frequency for a given term is a measure of how common that term is across all the documents in a given corpus.

Let **N** be the total number of documents in the entire corpus, and let $\mathbf{n_t}$ the total number of documents in which the term **t** occurs provided the term t is present at least once in the entire corpus. The inverse document frequency value is calculated as follows

: $\mathbf{IDF = Log(N/n_t)}$

Implementation Details :

The code for this phase of the project can be segregated into the following distinct sections:

1) **Pre-processing**: This part includes the process of tokenizing and cleaning the HTML inputs to retrieve text and their respective frequencies.

2)**Removing additional unimportant words**: After a list of key-words is obtained from the pre-processing phase, it is further processed to remove words of length one and all those words which occur only once in the entire corpus. This process takes **O(n) time** where n is the total number of key-words.

3) **Calculating word-document associations**: An index is created which maps every key-word to the number of documents which contain the key-word.This process is **O(nm)** in time where n is the number of key-words and m is the number of documents in the corpus.

4) **TF-IDF Calculation** : Iterate through the key-word dictionary for every input file, calculate the tf-idf values and then write them back to a new file. This process should be linear in time and an additional overhead of the file write operations

Analysis : The entire process of calculating TF-IDF for key-words was repeated for number of document increments of 10 and readings were recorded. About 10 such readings for varying number of documents were recorded and then average readings for time taken per number of documents were used to plot a graph.

The observable trend suggests that the time taken for processing increases almost linearly with increase in number of documents. However, there are two points of substantial increase in time.

The first peak occurs around document no 340 and shows a considerable increase on processing time for every 10 documents till document no 370. The sharp increase in processing times for these documents can be attributed to the fact that these documents are not in the English language and contain several words and punctuation marks. Several of these words occur only once in the entire corpus and most of them are just single letters separated by whitespaces. Both of these cases need to be weeded out after the initial pre-processing and this adds an additional time overhead.

The second sharp increase in processing times occurs from document number 430 to 440. The increase in processing time in this case is due to the fact that these documents account for about 16% of the total corpus size and contain a lot more data to process. Moreover, a lot of the contents of these files are simply numerical values which eventually get weeded out and very few key-words are actually weighted despite the enormous pre-processing required.

The rest of the graph shows a steady trend of increase in time with increase in number of documents.

(Graph on last Page)

An illustration : The parser and the cleanup procedures successfully clean out most of the words which have little to no effect on the scheme. In order to improve the efficiency of write operations, I choose not to write key-words which do not occur in the file(have a weight zero)to its corresponding .tws file. Lets consider an example of the scheme. Consider the word ***california***. The word occurs in a total of 16 files in the entire corpus, in the first file it occurs 3 times. The most frequent word that occurs in that file has a frequency of 16. By the adopted tf-idf scheme, the tf value of california for file 1 is 0.59. The idf value for california is log(503/16). The base of the logarithm does not matter since it contributes a constant multiplicative factor to all of the tf-idf values and the ranking in that case would be relative. So the tf-idf value for california in file 1 turns out to be 2.8682. Now for file no 8, California has a tf-idf value of 2.978. Which means that for a search query consisting of california, the file 8 will be ranked higher than file 1 since the word has a higher ”prominence ” in file 8 than it has in file 1.

Time taken vs Number of Documents to be processed

X-axis : Number of documents

Y-axis: Time taken for processing.(S)