Ahmed Alhaj
Phase 2 term weighting
CMSC476

In the first phase, I fall in the mistake of making a poor implementation preprocessor, which was little to this costly for this phase. My initial preprocessor was pretty hard to build on top of it which caused to restart all I over again. As I start, The first goal In this phase I decided to move to a decent preprocessor pipe line structure can be easily to work with. Building new a preprocessor and an inverted index is sort of inescapable after seeing the whole picture as to where the project is going to be about.

The new preprocessor has three parts, a files pipeline, document processor, and tokenizer. The pipeline open the set of the file filter the html syntax and use re.compile to read only characters and discard whitespace and symbols and the pass the file content a document processor. The document processor has parser to tokenize the file from the common words. The document processor also register the token in the inverted index, and lastly, it must keep track of document ID and it is content.

The inverted index keep tracks of the token and its frequency in the documents. The idea is to register a token and the documents in the in which the token show up in, in a nested dictionary in way the parent dictionary holds the token and list of documents and the child dictionary holds the document ID and the frequency of the token in this document. This in turn, allows us to apply for formula and algorithms pretty easily. The inverted index also supports other operation

for calculating tf-idf for a document and normalized tf-idf for term. It support operation of concatenate to another inverted indexes. The performance cost for the process, including writing token weight into a file, is around 9 seconds that is in case running without html filter, however, in case of html filter, it doubles up a 18 second.

Term weighting is a very important procedure in text indexing process, In this project we were supposed to calculate the term weight in each documents in order to assess the value of each term to the document, it should represent the important of the term in a document. Aside from a good formula for assessing a

With respect the term weight I used maximum normalization technique so I can normalize the tf weights of all terms occurring in a document by the total tf in that document. For each document

ntft,idf = a + (1 − a)* (log10(tf/number(tf-in-doc))

The whole point is smoothing by the contribution of second term, which can be viewed as a scaling down if the tf by largest tf value in document. The most important point of a maximum normalization is to reduce the deviation of following expected term. However my implementation seems to be unstable when I tried with running html-syntax filter vs without html-with-filter there

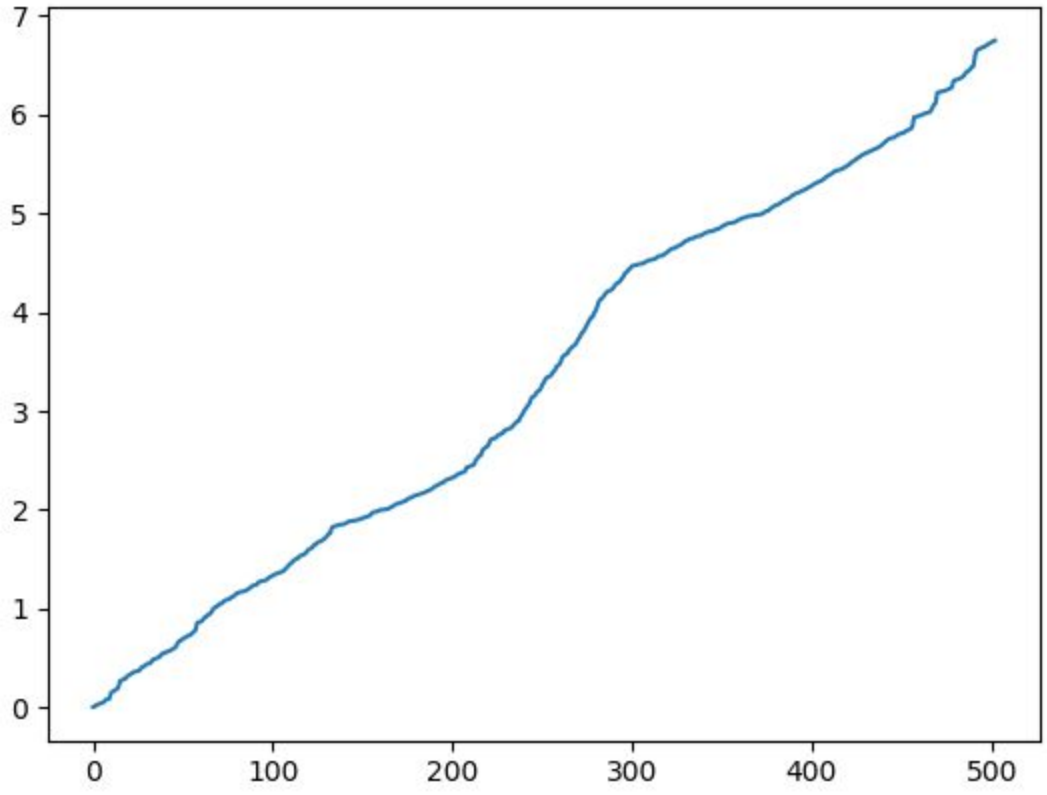
so huge change in the term weightings. Also it the term weighting gets highly affected by content of document if the weird.

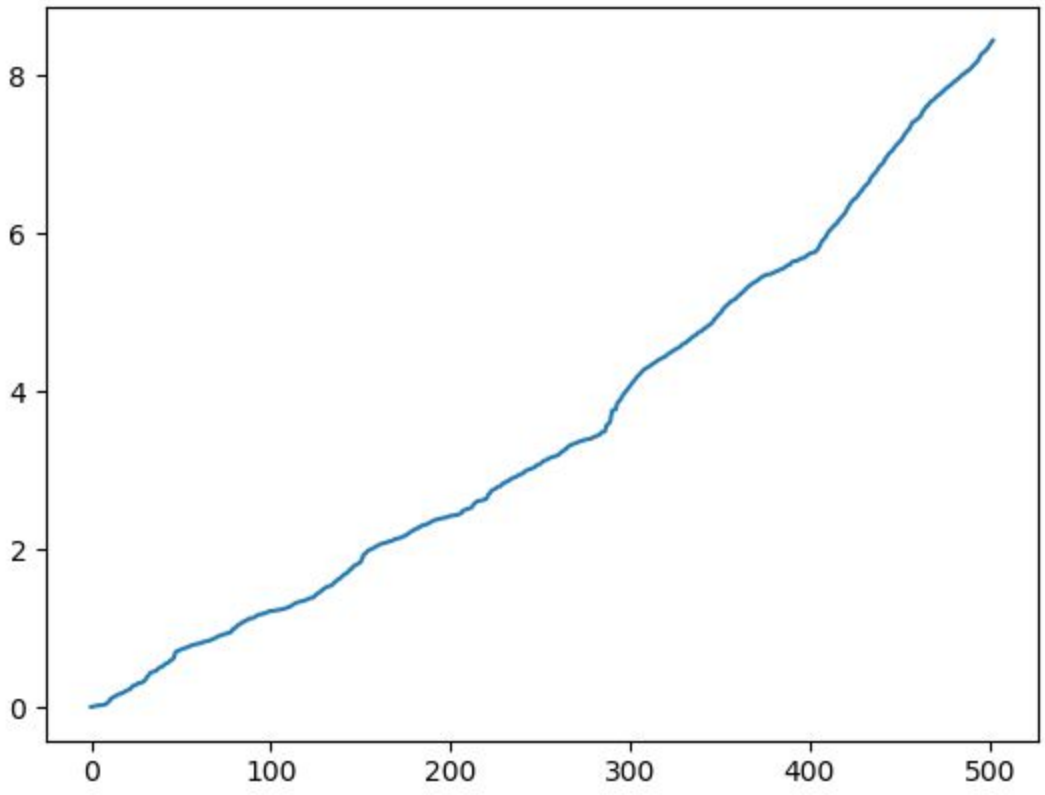
Performance Analysis

It takes around 8 seconds to read and tokenize the file and calculate the term weight without the html handler. If I used the html filter it doubles the time.

The First plot show the time for processing the file html filter and calculating term weight

The second plot shows the for processing the file with html filter and calculating their term weight.


7
6
5
4
3
2
1
0
0
100
200
300
400
500


8
6
4
2
0
0
100
200
300
400
500