RISHABH SACHDEVA – IR CMSC 676 – HOMEWORK 2 – BM25 APPROACH

Approach:

I extended my own pre-processor I wrote for previous assignment. I used REGEX to separate words. One of the important reason for not using libraries for this job is program speed. Another reason is language constraints. I used JAVA and it does not have support for good libraries, when compared to python. But, running time is the clear advantage. All stop words were removed for the score calculations.

Output of my program are the property files corresponding to each html have word vs bm25 score.

How to run:

javac BM25.java

java BM25 <input_dir> <output_dir>

TERM WEIGHTING SCHEME: BM25

Word Frequency:

The tuning that appeared relevant in calculating TF is not to use word_frequency, instead use the square root of its value. The major motivation behind this approach is that "if the *word* is appearing twice in one document, then it actually doesn't make it twice the relevant." So, proceeding with square root appeared as a good idea to begin with.

TF: In traditional TFIDF scheme, TF refers refers to the frequency of the word. It is important to take into account the length of document for normalization.

In BM25 approach, the TF score is influenced by whether the document is above or below the average length of the document in corpus. Two new variables are introduced in the formula, b and k to incorporate the mentioned conditions.

TF = (word_freq_sqrt * (k + 1)) / (word_freq_sqrt + k * (1 - b + b * doc_size / avg_doc_size));

"*doc_size / avg_doc_size"* in the formula corresponds to "*how long is the document when compared to avg document size of the corpus.*" The constant '**b**' allows to tune how much influence the "relative size" of the document should have on the score. In traditional TFIDF, such tuning factors are not set, which clearly play an important role to judge relevance, as in BM25.

IDF:

IDF = ln(num_docs/docFreq+1);

The +1 factor is introduced in the log, which make sure that negative value is not computed.

Overall Formula for TF-IDF:

Math.log(num_docs/(docFreq+1)) * (word_freq * (k + 1)) / (word_freq + k * (1 - b + b * doc_size / avg_doc_size));

Choosing k1 and b:

It is widely known that there is no "best" values for b and k1 for all the corpus sets. The general followed approach is to try tuning the score with different values that make sense, and choose the best one.

Optimal b is generally around 0.3-0.9 range. Its an important normalizing factor as it allows to tune how much influence the "relative size" of the document should have on the score. I tried with very low values like 0.1 and 0.2, but it clearly doesn't make sense if we are completely ignoring the influence of relative size of the corpus. In my program, I finally settled-up with 0.75 value of b. This is generally considered to be a good value with works well with most of the corpuses. K1 is typically between 0 between 3. I used the value 2 which is again the default one and works well with most corpuses. I did played with these values, and studied how the tuning can affect results.And, I clearly saw the score differences, and understood how these normalizing factor can be used in an effective manner.

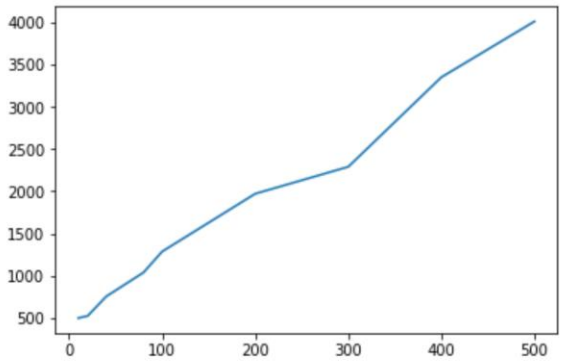
In short, there are no perfect values but BM25 with k1 = 1.2 and b = 0.75 seem to give very good results for most cases.

RUNTIME ANALYSIS

Num_Files	Running_Time (ms)
10	501
20	526
40	757
80	1040
100	1288
200	1972
300	2291
400	3351
500	4011

```
In [1]: import matplotlib.pyplot as plt

In [2]: plt.plot([10,20,40,80,100,200,300,400,500],[501,526,757,1040,1288,1972,2291,3351,4011])
Out[2]: [<matplotlib.lines.Line2D at 0x1eda2357b00>]
```



Effect of Stop words:

In previous assignment, 1.properties file had 589 tokens and the current generated has 447 tokens.

In older 2.prop file, there were 1092 token and current one has 896.

Same effect was seen in all the other files. Hence, the removing stop word effect was clear. Attaching the 2 sample prop files for reference.