Prototype 2 Submission Report.

Gaurav Patil.

Implemented TFl◦δ◦p×IDF:

Source:

Rousseau, F., M. Vazirgiannis, Composition of TF normalizations: new insights on scoring functions for ad hoc IR, SIGIR 2013, p. 917-920.

Description:

Rousseau and Vazirgiannis suggest that nonlinear gain from observing an additional occurrence of a term in a document should be modeled using a log function. Following BM25+, they add (delta) δ to ensure there is a sufficient gap between the 0^{th} and 1^{st} term occurrence. For document length normalization, they prefer the pivoted component from BM25. The combined equation is as follows.

$$rsv_q = \sum_{t \in q} \ln \frac{N+1}{df_t} \cdot \left(1 + \ln\left(1 + \ln\left(\frac{tf_{td}}{1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)} + \delta\right)\right)\right)$$

Code:

https://github.com/gauravpatil93/complex-answer-retrieval/blob/master/tc_TFIDF_IMPROVED.py

Implemented BM25+:

Source:

Lv, Y., C. Zhai, Lower-bounding term frequency normalization, CIKM 2011, p. 7-16.

Description:

Lv & Zhai go on to observe that the penalization of long documents occurs not only in BM25 but other ranking functions. They propose a general solution to this, which is to lower-bound the contribution of a single term occurrence. That is, no matter how long the document, a single occurrence of a search term contributes at least a constant amount to the retrieval status value. The equation is as follows.

$$rsv_q = \sum_{t \in q} \log\left(\frac{N+1}{df_t}\right) \cdot \left(\frac{(k_1+1).tf_{td}}{k_1 \cdot \left((1-b) + b \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} + \delta\right)$$

Code:

https://github.com/gauravpatil93/complex-answer-retrieval/blob/master/tc_TFIDF_IMPROVED.py

Implemented LM-DS (Dirichlet Prior Smoothing):

Source:

Petri, M., J.S. Culpepper, A. Moffat, Exploring the magic of WAND, ADCS 2013, p. 58-65.

Description:

Dirichlet smoothing is a standard method of interpolating between the collection model and the document model. Where Lq is the length of the query, u(mew) is a tuning parameter, tf_{tq} is the number of times the term occurs in the query, Lc is the length of the collection (in terms), and cft is the number of times the term occurs in the collection (the collection frequency).

Code:

https://github.com/gauravpatil93/complex-answer-retrieval/blob/master/tc_DIRICHLET.py

Implemented TextProcessing Module:

- *Regular expressions to remove noise from text*
- *Filtering list of stop words using NLTK corpus*
- *Stemming Words using Porter2*
- *Word ranking*

Caching:

The data for the ranking functions can be created once and used across different ranking functions in my implementation. Serialization is used for storing complex data structures. However even after using a C based implementation of 'pickle' the library was unable to store the index for 7 million passages.

Running the Code:

```
The code implements 4 types of ranking functions BM25, BM25+, TFIDF(Delta), DIRICHLET

Caching mechanism implemented for TFIDF(Delta) and DIRICHLET SMOOTHING METHODS so
once the cache is generated it can used for both of these methods.

If running tests on TFIDF(Delta) and DIRICHLET
```

Generating Cache

```
tc_generate_document_cache.py [outlines file] [paragraphs file] [no of passages to
extracts from pagagraph file]
```

Generating trec_eval compatible results file

```
tc_generate_document.py [outlines file] [paragraphs file] [output file] [ranking
function] [cache] [no of passages to extract]

The aforementoned arguments can take the following value:

[ranking function]          : BM25, BM25+, TFIDFIMPROVED, DIRICHLET
[cache]                     : no_cache, cache ( Note 'cache' only works if
tc_generate_document_cache.py is run first on same number of passages )
[no of passages to extract] : an integer

For first run: The repo already includes a cached collection of 50,000 passages so to
test either TFIDF(Delta) or DIRICHLET
just run the following command

tc_generate_document.py all.test200.cbor.outlines release-v1.4.paragraphs
output.DIRICHLET.run DIRICHLET cache 50000
tc_generate_document.py all.test200.cbor.outlines release-v1.4.paragraphs
output.DIRICHLET.run TFIDFIMPROVED cache 50000
```

NOTE

```
Important note about caching. cPickle has limitations to the amount of data that it
can serizalize and store so after running a test on 7 million passages trying to
cache them proved that indexes of such a large data set cannot be cached

For the next iteration a pure pythonic full text indexing library can be used such as
https://pypi.python.org/pypi/Whoosh/
```

or an alternative to this would be to serialize and store the data in a relational or no-sql database.