

Prototype 2 Submission Report.

Gaurav Patil.

Implemented TFI δ p \times 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 0th and 1st 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.

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

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 L_q 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, L_c 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.