Final Project

Submission Deadline: August 23rd, 23:59

Task Description

In this final project, the task is to develop and evaluate a two-stage information retrieval model that given a query returns the n most relevant documents and then ranks the sentences within the documents. For the first part, you should implement a baseline document retriever with tf-idf features. To get full credits, in the second part you should improve over the baseline of the document retriever with an advanced approach of your choice. The third part extends the model to return the ranked sentences. The answer to the query should be found in one of the top-ranked sentences.
In addition to the source code, you should submit a 4-6 page report that describes the problem and why it is interesting/challenging in your own words, the preprocessing steps, the models you have developed, your evaluation results and an analysis of the results.
The data you will use is given in marked up language with tags:

```
<DOC>
<DOCNO> CR93H-1 </DOCNO>
<DOCID>H01AP3-2</DOCID>
<TTL>DESIGNATION OF SPEAKER PRO TEMPORE (House - April 01, 1993)</TTL>
<DATE>19930401</DATE>
<TEXT>
The SPEAKER pro tempore laid before the House the following communication from
the Speaker:
...
...
</TEXT>
</DOC>
```

Before you can pre-process the texts, you have to extract the document numbers and texts.

1 Baseline Document Retrieval Model

As a baseline for the document retriever, you should implement a Tf-Idf model. Therefore, you should create an index that contains the tf and idf values for your terms.

(a) Extract the text and document numbers from the corpus. You may use libraries such as BeautifulSoup[1] or xml[2] to parse the files. Note that the documents contain the text within `<text>` tags. The LA-Times documents, whose document numbers start with `LA`, use additional `<P>` tags.

(b) Pre-process the texts from the documents by applying tokenisation, lower-casing, removing punctuation tokens. You may use `NLTK` to tokenise the documents. However, for the rest

[1]https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[2]https://docs.python.org/3/library/xml.etree.elementtree.html

of Section 1 you are not allowed to use any external libraries that offer functions to obtain the following weights and scores.

(c) Compute and store the inverse document frequency (idf) of the terms:

$$idf(term_i) = log(\frac{N}{n(term_i)}) \tag{1}$$

where N is the total number of documents and $n(term_i)$ is the number of documents that contain the $term_i$.

(d) Compute and store the term frequency (tf) of the terms per document along with the document ID:

$$tf(term_i, doc) = \frac{n(term_i, doc)}{max(term_j, doc)} \tag{2}$$

where $n(term_i, doc)$ is the number of times $term_i$ appears in the document and $max(term_j, doc)$ is the number of times the most frequent term of the document appears in the document.

(e) Given a list of query terms, you can compute and store the tf-idf weights for these terms as the product of the term's idf and the tf-value of the term in the respective document:

$$tf - idf(term_i, doc) = tf(term_i, doc) * idf(term_i) \tag{3}$$

(f) Based on the tf-idf weights, you can determine the relevance of the documents for the query based on their cosine similarity with the query using dot products. To calculate the tf-idf weights for the query, you should treat it like a document.

$$cos(\overrightarrow{q}, \overrightarrow{d}) = \frac{q \cdot d}{|\overrightarrow{q}||\overrightarrow{d}|} \tag{4}$$

(g) Sort the similarity scores and output the top 50 most relevant documents for a query along with their scores.

(h) Write a function to evaluate the performance of your document using the evaluation metric precision at r with $r = 50$:

$$Precision = \frac{\#(relevant\ and\ retrieved\ documents)}{\#(retrieved\ documents)} \tag{5}$$

Precision at r computes the percentage of relevant documents from the top r retrieved documents. You should ignore the documents ranked lower than r

(i) Use the `precision at r` function to evaluate your performance as mean of precisions over the queries given in `test_questions.txt`. First, you must extract the queries from the descriptions `<desc>`. A document is relevant, if it contains the answer. You can check this by using the regex patterns for the answers to the corresponding queries in `patterns.txt`.

2 Advanced Document Retriever with Re-Ranking

In this section, you will improve over the performance of the baseline information retrieval model by first ranking and returning the top 1000 documents for a query with the baseline retriever from Section 1. Then, you should develop your own method to re-rank these 1000 documents to return the top 50 documents, which should improve over the top documents returned by your baseline model in Section 1.

(a) Use the baseline model from Section 1 and return the top 1000 documents given a query.

(b) Re-rank the top 1000 documents with a more advanced approach. As inspiration, you can look at more advanced scoring functions such as BM25[3], SVMs for ranking[4] or experiment with neural re-ranking methods. Your re-ranker should return the top 50 documents based on the top 1000 documents returned in **(a)**.

3 Sentence Ranker

In this section, you will use the top 50 documents from Section 2 to rank the sentences in these documents.

(a) Split the top 50 documents into sentences. You may use an existing function such as `sent_tokenize` from `NLTK`.

(b) Using the same approach you developed in Section 2, you can treat the sentences like documents to rank them and return the top 50 sentences.

(c) Evaluate the performance of your model using the mean reciprocal rank function (MRR) on the test queries Q:

$$MRR = \frac{1}{|Q|} \sum_{q=1}^{Q} \frac{1}{rank(q)} \tag{6}$$

where $rank(q)$ is the rank of the first relevant sentence for the query q. Use the regex patterns to determine whether a sentence is relevant or not.

Grading (100 Points)

- 25 points - Baseline Document Retrieval Model
- 25 points - Advanced Document Retrieval Model
- 25 points - Sentence Ranker
- 25 points - Report is well-written, presents meaningful analysis with evaluation of different models, and contains all required information

Submission Instructions

The following instructions are mandatory. Please read them carefully. If you do not follow these instructions, the tutors can decide not to correct your exercise solutions.

- You have to submit the solutions of this exercise sheet as a team of 2 students.
- Please use Python unless otherwise agreed upon with your tutor.
- When developing the baseline document retrieval model, please implement the functions from scratch. The use of libraries that come with these functions is prohibited. For developing the advanced document retriever, you may use any libraries and tools of your choice.
- Make a single `ZIP` archive file of your solution with the following structure
  - A `source_code` directory that contains your well-documented source code and a `README` file with instructions to run the code and reproduce the results.

[3]https://radimrehurek.com/gensim/summarization/bm25.html
[4]https://en.wikipedia.org/wiki/Ranking_SVM

  - A `PDF` report with your model descriptions and comparisons, evaluation and analysis.
  - A `README` file with group member names, matriculation numbers and emails.

- Rename your `ZIP` submission file in the format

  `exercise02_id#1_id#2.zip`

  where `id#n` is the matriculation number of every member in the team.

- Your exercise solution must be uploaded by only one of your team members under *Assignments* in the *General* channel on Microsoft Teams.

- If you have any problems with the submission, contact your tutor before the deadline.