Project_part1

Author: Yinghong Zhong(z5233608)

Date: 2019/10/31 (T3)

Project Part 1 is separated into three sections in which I import Spacy, collections and itertools libraries to implement my solution.

Section 1: TF-IDF index construction for entities and tokens

At first, I use Spacy to parse the documents into tokens and entities and then aggregate entities and tokens into the ent_dict and token_dict, which format is {entity: {docID: count}} and {token: {docID: count}} respectively. When I am aggregating entities, I collect the single-word entities into a list named single_word_ent, and these single-word entities will be excluded during token aggregating part. This step is important to ensure that single-word entities occurring in ent_dict will not occur in token_dict.

Based on the construction of TF-token and TF-entity, I normalize TF of token and entity, and calculate the idf_tokens and idf_entities following the formula.

Section 2: Split the query into entities and tokens

Step 1: select the probable entities

Firstly, I split the query into single-word tokens, and if these single-word tokens occur in DoE, I would append them to the entitylist which is used to store the

probable entities.

Secondly, I construct combinations of tokens and find multi-word entities. I use for_loop (from 2 to size of the query) and combinations syntax to find every combination of tokens. If the phrase formed by the words in the combination occur in DoE, I would append them to the entitylist.

In this part, I would get the probable entities including single-word and muti-word entities, which are not duplicate. And I also append the split with all words as tokens and no words as entity in the query_splits list.

Step 2 and step 3: Find the combinations of probable entities and check token count of each combination doesn't exceed Q_count.

I use for_loop (from 1 to size of the entitylist) and combinations syntax to find subsets of probable entities.

For each subset, I count the words in it by using Counter syntax, and then compare the subset_Couter and Q_Counter by key to check if the word count of subset exceeds word count of query

If subset of entity doesn't exceed the query, the remaining part of the query would be the token part which is calculated by Q_count minus subset_Counter. Now I can get the correct split of the query and append it to query_splits list.

Part 3: Query score computation

In this part, I enumerate the query_splits list and calculate entity score, token score and total score for each split. During calculation, I have to check if the

ent/token and doc_id in our TF-IDF index.

After calculation, I would find the max total score and return the max_score and its split.