Project_part1 Report

In class *InvertedIndex*, there are three functions:

index_document(documents),
split_query(Q, DoE),
max_score_query(query_splits, doc_id)

Spacy, itertools and math.log were imported in this file.

Implementation:

1. *index_document(documents)*
   For given documents, each document was iterated to obtain the separate lists of entities and tokens. Then for each token and entity in that specific document, the counts of their occurrence were recorded in dictionaries respectively for the particular document in terms of term frequency. After finishing the counting of the tokens or entities for all documents, idf for tokens and entities was calculated.

   Algorithm:
   1) Iterate through input documents.
      For each document, extract entities in a list using Spacy, so does tokens (check for the occurrence of single-word entities, stop words and punctuations and some special cases that might happen).
   2) Iterate through tokens for specific document.
      For each entity, count its occurrence in current document and update the counts as values of the dictionary with doc_id as keys, and then update the dictionary as the value for *self.tf_entities* whose keys are entities, so does tokens.
   3) Iterate through *self.tf_entities* and *self.tf_tokens*.
      Compute idf for each entity and token.

2. *split_query(Q, DoE)*
   The combination of DoE was computed first, and the occurrence of each token in different combinations was checked to make sure the query has increasing order of tokens for certain entity combinations.

   Algorithm:
   1) Split the query into tokens.
   2) For each entity in DoE, check whether the terms in the specific entity exist in the query.
   3) Use *itertools.combinations* to find out all the combinations for entities in *DoE*.
   4) Iterate through the combinations.
      Use itertools.permutations to find all permutation for the specific combination.
      In the iteration for all the permutations, if
         one of the permutations was added to the final entity list, then break and continue the next combination.
      Else,
         Split the entities into a list of *entity_term.*
         Find the index of sequential elements in *entity_term* list and delete the respective token from *token_list*.

If there is an entity whose term does not exist in *token_list*, it will not be added into the return list and vice versa.

3. *max_score_query(query_splits, doc_id)*
   For each *query_split*, the normalized term frequencies for tokens and entities were calculated, and then based on the normalized term frequencies, the combined score was computed. To find out the best split for query, the maximal combined score was selected and the corresponding split is the optimal split for query.

   Algorithm:
   1) Iterate through *query_splits*.
      For each entity of each split, compute the normalized entity score.
      For each token of each split, compute the normalized token score.
      Combine two scores with certain proportion.
   2) For each combined score, find out the maximal one and return the score and the corresponding split.