Part 1 Report

z5242692 Chenqu Zhao

This report illustrates my implementation details for project part 1, following the specifications step by step.

Step 1: TF-IDF index construction for Entities and Tokens

First, two empty dictionaries are initiated for both tokens and entities to store TF and IDF score separately. To start indexing documents, I implement a for loop to traverse values of the document dictionary. During each loop, tokens are picked first, and then entities are picked. The brief idea of indexing is explained below.

```
for each doc:
    initiate token_count and entity_count(dict)
    for each token in doc:
        if is stop word or punctuation:
            continue
        if appeared before:
            corresponding token count + 1
        else:
            add a new key to dict, token count = 1
    for each ent in doc:
        if appeared before:
            corresponding entity count + 1
        else:
            add a new key to dict, entity count = 1
        if is single word entity:
            reduce corresponding token count by 1

    update tf_tokens, tf_entities through token_count and entity_count
```

After this procedure, IDF of entities and tokens could be calculated by traversing.

Step 2: Split the Query into Entities and Tokens

To improve the readability of my code, a sub-function is defined for this step. How it works with the initial function is introduced below.

```
def handle_token(token_list, entity_list, DoE):
    initiate id_com to store entity id combination
    for entity in entity_list:
        record current token_list as token_mark
        for each word in current entity:
            if word appears earlier than last word or cannot find this word:
                reset token_list to token_mark
                break
            else:
                delete this word

        if token_list changes after the previous loop:
            add entity id combination to id_com
            sort id_com

    return a tuple(token_list, entity_list, id_com)

def split_query(Q, DoE):
    initiate id_list to store DoE id combinations
    initiate split_result to store split results
    do permutations for DoE.keys()
    for each entity combination:
        split query into list of words as initial token list
        call handle_token(token_list, entity_combination, DoE)
        if entity_combination not in id_list:
            add to id_list
            collect split details and add to split_result

    return split_result
```

Step 3: Query Score Computation

First, call split_query to get query_splits(list), then do for loop as below.

```
for split in query_splits:
    retrieve self.tf_tokens, self.idf_tokens, then compute tfidf for token
    retrieve self.tf_entities, self.idf_entities, then compute tfidf for entity
    score = 0.4*token_score + entity_score
    if score > max_score:
        score = max_score
        store corresponding split operation

return a tuple(max_score, split operation)
```