COMP6714 Project-1 (Part 1)

Amritesh Singh
z5211987

Implementation Details of Q1

- **Initialization:**

  Dictionaries to store TF, IDF, TF_{norm}, and TF-IDF, one each for both tokens and entities, as well as spaCy's English language model (***en_core_web_sm***), are initialized. Let tokens and entities be represented by ***t*** and ***e*** respectively.

- **TF-IDF Index Construction:**

  Iterating over each element in the document dictionary ***D***, ***TF(e)*** dictionary is constructed by extracting all elements obtained after passing the string to the spaCy model and iterating over its ***ents*** attribute and setting these elements' text as keys, while setting their frequency in the string as the values. The starting index in the string for each entity is also stored. Iterating over each token in the spaCy-processed string, the starting index of each token is kept track of. If the token is a stop word, punctuation, or its starting index matches that of a single-word entity, it is skipped over. ***TF(t)*** dictionary is constructed by taking the remaining tokens' text as keys and their frequency as values.

  Iterating over ***TF(t)***, for each key, an entry is added in ***IDF(t)*** dictionary with the same key, and the value being calculated using the formula given in the assignment specifications. Further iterating over the values in ***TF(t)***, the $TF_{norm}(t, key)$ and ***TF-IDF(t, key)*** dictionaries (where ***key*** is the key corresponding to each element of the ***TF(t)*** values) are constructed using the given formulae. The whole process is repeated for the ***TF(e)*** to construct the $TF_{norm}(e, key)$ and ***TF-IDF(e, key)*** dictionaries.

- **Query Split:**

  Iterating over each element in ***DoE***, let ***x*** and ***y*** be the split lists for **Q** and the key respectively. Iterating over ***x***, whenever the element at the current index of ***x*** matches the element at the current index of ***y*** (starting from index 0), the current index of ***y*** is shifted to its succeeding index. After each iteration of ***x***, if the current index of ***y*** equals the length of ***y***, i.e., all the elements of ***y*** have been visited in increasing order, the corresponding key is added to the list of valid entities.

  Iterating over all combinations of the entities list, those combinations are filtered out that yield a combined number of all the tokens for all entities in the entities list exceeding the number of tokens in **Q**. For the remaining combinations, each separate entity in the combination list is removed from the split list of **Q** (named ***y***) in increasing order of the individual words in the

entity. This is accomplished for each entity in the combination list by setting initial index to 0, finding the index of the current word in the entity (which must be greater than the current index), updating the current index, and removing the current word from ***y***. In case a combination fails to remove elements from ***y*** in the desired fashion, another permutation for the combination list is tried out, since the order in which entities are deleted can also change the resulting ***tokens*** list. For each combination list of ***entities***, if any permutation succeeds in forming a ***tokens*** list, they are added as entries on the value side to the result dictionary, with the key being a simple counter. This finally results in the construction of a dictionary that consists of each combination of ***tokens*** and ***entities*** list as values.

- **Max Score Computation:**

  Iterating over each element of the ***query_splits*** dictionary, for each entry in the value ***tokens***, the corresponding value in the ***TF-IDF(t, doc_id)*** dictionary, where ***doc_id*** is the given document ID, is taken and added to the ***tokens_score***. The process is repeated for each entry in the value ***entities*** (using the ***TF-IDF(e, doc_id)*** dictionary), resulting in the computation of the ***entities_score***. The combined score is calculated as ***(entities_score + 0.4 * tokens_score)*** and this combined score and the corresponding key are stored as a tuple in the list ***score_list***. Sorting this list in descending order of the scores and returning the first element of the sorted list gives the entity and token list combinations with the maximum score.