Report

Aryan Singh | 19CS30007 | IR Default Term Project Part 2

We used indexer.py and parser.py to generate model_queries_10.bin and queries_10.txt.

Task 2A

Files that were built in this part : Assignment2_10_ranker.py

My contribution as a team member :-

- To design an efficient vector representation using dictionaries.

  My team had implemented the tf-idf vector using an on-demand technique. It only computed TF-IDF when a computation related to the document was required. As a result, the space complexity is reduced from O(Docs + Vocabs) to O(Vocabs). They were able to handle all the vocabulary in the corpus. But the time required for ranking using this method was about 3 minutes per query.
  I then used multi-threading to bring down the time from 3 minutes to 1.5 minutes.

  On close inspection of the code, I and my teammates observed that using np.arrays in this case is slow. This is due to the fact that the numpy arrays are scarcely populated.

  After extensive discussion over this, we decided to move forward with the design choice of using a dictionary-based approach to represent the tf-idf vector, storing only the non-zero values. This method brought the time down from **1.5 minutes to 0.5 seconds**. Thus, this was very fruitful because now we can work and check the performance in real time.

My contributions as an individual :-

- Implementing the custom cosine similarity

  We finally used dictionary-based vectors, due to which the numpy dot product was rendered powerless. Hence, to calculate the cosine similarity, I designed a custom function.

  def custom_cosine_sim(tokensA, tokensB):
  This took two normalized dictionaries and then traversed keys common to both and accumulated the product of their values.

Task 2B

Files that were built in this part : Assignment2_10_evaluator.py

My contributions as an individual :-

- Parsing and storing Ranked List in an effective manner.
  Problem Faced : Using values directly from the ranked_list file would have been very cumbersome.
  Approach : parsing the ranked list once and storing it in list-of-lists fashion.

- Computing the NDCG value
  This part had two sub-parts. For every query, I had to do the following:
  1. calculate DCG for the first k terms in ranked_list
  2. Sort the term judgment score to find the top k relevant scores.
  3. Calculate the ideal DCG.
  4. Divide the DCG/ Ideal DCG to compute the NDCG.