Information Retrieval Assignment 1

Alexey Shapovalov (20235952)

November 21, 2021

1 Question 1

- For this answer an assumption is made that the history of a users ratings is unknown.
- There are two challenges with the 1-5 range based feedback mechanism:
  1. How to classify a document as relevant or non-relevant.
     - To do this a simple if statement can be used:

     ```
     if (document.userRating > 3) then document is relevant
     else document is not relevant
     ```

     - The reason I would consider a 3 as non-relevant is purely subjective, I would prefer documents that I would rate as a 3 to non be included in the relevance feedback. In a real world system this could be evaluated.
  2. How to incorporate the level of relevance/non-relevance of the range, e.g. a 5 is more relevant than a 4, yet a 4 is still relevant.
     - My solution to this would be to **multiply the document vector** by `x = 1 / document.userRating` for the non-relevant documents and `y = document.userRating / 5` for the relevant documents.
     - Rating = 1: Document is considered **irrelevant** from the if statement above. Multuplier is `x = 1 / 1 = 1`.
     - Rating = 2: Document is considered **irrelevant** from the if statement above. Multiplier is `x = 1 / 2 = 0.5`.
     - Rating = 3: Document is considered **irrelevant** from the if statement above. Multiplier is `x = 1 / 3 = 0.33`.
     - Rating = 4: Document is considered **relevant** from the if statement above. Multiplier is `y = 4 / 5 = 0.8`.
     - Rating = 5: Document is considered **relevant** from the if statement above. Multiplier is `y = 5 / 5 = 1`.
     - By multiplying the document vectors via the scheme above it reduces the impact of documents with scores that are not either very relevant (5) or very irrelevant (1).
- Based on this the modification would be to multiply the documents by **x** (for elements in the set D_r) and **y** (for elements not in the set D_r) during the summations. The if statement is then used to place a document into set D_r.

2 Question 2

- Initially a term to term matrix is created. This matrix specifies how much each term correlates to every other term. Each term is vectorized in some way and then compared to each other with a correlation measure, examples include the pearson correlation and vector similarity.

Each cell in the matrix would hold the correlation measure between the word in the column header and the word in the row header.

- From this matrix my solution would consist of three steps, one step to return a set of terms that are maximally correlated, another step to try to find more diverse terms and the last step to combine both lists.
- The first step would be straightforward, for each term in the query: extract the top **x** terms that are above a threshold **t** with respect to the correlation to the term. The output of this step is stored in a list `top_suggestions`.
- To handle diversity, stoted in `diversity_suggestions`, an algorithm could be:

```
find_diversity_suggestions():
    y = ? # how many terms to extract for each query term

    # max correlation score to be considered unsimilar
    # to a search term (or added suggestion)
    dissimilarity_threshold = ?

    diversity_suggestions = []
    for each term in query_terms:
        term_suggestions = []
        candidates = sort all unique terms by their correlation score to term
        for candidate in candidates:
            # In every iteration after the first iteration
            # term_suggestions will not be empty
            targets = (query_terms - term) + term_suggestions
            if correlation_max(candidate, targets) < dissimilarity_threshold:
                term_suggestions.append(term)
            if len(term_suggestions) == y:
                break
        diversity_suggestions += term_suggestions

    return diversity_suggestions
```

- Essentially the algorithm extracts the most correlated terms that are somewhat uncorrelated to all the other query terms and to the suggestions found so far.
- The last step would be to combine the lists `top_suggestions` and `diversity_suggestions` by adding the lists together and removing duplicate terms. Note: if these terms are actually displayed to the user (e.g. as suggestions in a UI) they could be sorted by recommending terms that are in both lists higher than those that are not.
- This approach requires choosing reasonable values for **x**, **t**, **y** and **dissimilarity_threshold**. These would need to be found through experimentation and are impacted by a number of factors such as the corpus and this size of N.

3 Question 3

- The goal for this question is how to incorportate both the user id and the search terms into the information retrieval system. My approach is as follows:
- First create a user by user correlation matrix where each cell represents the correlation score

between the user in the column header and the user in the row header.
  - This is done by first vectorizing each user in some way, for example with a vector of size unique_words each value could correspond to the number of times the word is contained in a search query divided by the number of search queries.
  - Next the matrix is created by evaluating for each user a correlation measure to every other user, for example pearson correlation, vector similarity.
- With this matrix when a user starts a search term the system could find the top **N** most correlated users to the target user. This could then be used to filter the log records to only search queries for the most correlated users, denoted as T.
- The search queries in T could then be used to generate suggestions for extra query terms. This essentially reduces the search space down to queries that are done by users that are correlated to the current user.
  - This could be done in the exact same manner as question 2 where the search queries are treated as the documents.
  - A term by term correlation matrix is created from the search queries in T.
  - For each term in the user query, extract the top **x** terms that are above a threshold **t** with respect to the correlation to the term. Return these terms as the suggestions.
  - **Note:** In addition to this, for the same reasons as in question 2 (diversity) you could further enhance this with the algorithm in question 2.

3.1 Advantages

- The main advantage is that both piece of information are utilised, the user id and the query terms. By utilising the user id the outcome is that the suggestions should be based on search terms that are quite similar to the target users search terms. As the users come from the same company they may come from different departements. When a user from one department searches for a given term it could have a completely different context from when a user from a different department searches for the same term.

3.2 Disadvantages

- The system suffers from a sort of "cold start" problem. When a new user enters the system there is not enough information to correlate to other users and as such the top **N** users are not very meaningful.
- By only considering the search terms in the top **N** correlated users some diversity is lost. Maybe the search query is quite generic and the context of the department is not very relevant. In this case filtering to the top **N** could remove users that are not correlated and as a result give back less than optimal suggestions.