Information Retrieval: Assignment 2

Marcel Aguilar Garcia, Id: 20235620

November 21, 2021

Question 1

In the classical Rocchio method for relevance feedback, searchers are given an option to give feedback of a binary nature (either positive or negative relevance feedback). Consider designing a system where users can give feedback in a range 1 to 5, where 1 means very non-relevant and 5 means very relevant. Modify the Rocchio approach to allow for such a feedback mechanism.

Answer Question 1 The original Rocchio method considers only documents that are relevant and documents that are not relevant. If we denote by D_R the collection of relevant documents and by D_N the collection of non relevant documents, Rocchio method provides a new query vector given by:

$$\vec{q} = \alpha\vec{q} + \frac{\beta}{|D_R|}\sum_{d_j \in D_R} d_j - \frac{\gamma}{|D_N|}\sum_{d_j \in D_N} d_j$$

where α, β, γ are constants which determine the importance of feedback, and the relative importance of positive feedback over negative feedback.

Let's consider a system where users can give feedback in a range 1 to 5, where 1 means very non-relevant and 5 means very relevant. In this case, we can consider five different categories. We can denote by D_i the collection of documents that have been given a feedback of i. This means that there will be two categories for relevant documents D_5 and D_4, two categories for non-relevant documents D_2 and D_1, and one neutral category, D_3. As in the original method, we would want our query vector to move closer to the relevant documents and further from the non-relevant documents. It seems common sense that neutral documents, D_3, should not have an impact to this. In this way, we can extend the original formula to:

$$\vec{q} = \alpha\vec{q} + \frac{\beta}{|D_5|}\sum_{d_j \in D_5} d_j + \frac{\mu}{|D_4|}\sum_{d_j \in D_4} d_j - \frac{\lambda}{|D_2|}\sum_{d_j \in D_2} d_j - \frac{\gamma}{|D_1|}\sum_{d_j \in D_1} d_j$$

where α, μ, λ, β, γ are constants which determine the importance of feedback and the relative importance of positive feedback over negative feedback.

Finally, by choosing constants correctly, the expression above would provide good results. However, by default, it is not taking into account that documents in D_5 are more relevant than documents in D_4 and, analogously, that documents in D_2 are more relevant than documents in D_1. We could give weights that are relative to the feedback to make sure that the constants are aligned to the weight of each category:

$$\vec{q} = \alpha\vec{q} + \frac{5\beta}{|D_5|}\sum_{d_j \in D_5} d_j + \frac{4\beta}{|D_4|}\sum_{d_j \in D_4} d_j - \frac{4\gamma}{|D_2|}\sum_{d_j \in D_2} d_j - \frac{5\gamma}{|D_1|}\sum_{d_j \in D_1} d_j$$

Question 2

Given a query that a user submits to an IR system and the top N documents that are returned as relevant by the system, devise an approach (high level algorithmic steps will suffice). To suggest query terms to add to the query. Typically, we wish to give a large range of suggestions to the users capturing potential intended query needs, i.e., high diversity of terms that may capture the intended query context/content.

Answer Question 2 Let $\mathcal{T} = \{t_1, ..., t_n\}$ be the collection of terms from the top N documents and let M be an n x n matrix defined as $M_{i,j} = h(t_i, t_j)$ where $h : \mathcal{T} \text{x} \mathcal{T} \to \mathbb{R}^+$ is a function that assigns a correlation to two terms. For the purpose of this exercise, we can use one of the metrics from the lectures $h(t_i, t_j) = \sum\limits_{t_i, t_j \in D} \frac{1}{dis(t_i, t_j)}$ where $dis(t_i, t_j) = \infty$ if t_i and t_j are in different documents.

Now, given a query $Q = (t_{i_1}, ..., t_{i_k})$ we would like to find an algorithm that will suggest new query terms.

The suggested terms should be somehow correlated with the query terms. First, I thought of using the union of sets of terms that are highly correlated to each term t_{i_j} for $j = 1, ..., k$ and then, take the one with highest correlation as the first suggestion. However, this could lead to the situation in where a term t_r has a high correlation with, e.g., t_{i_1} but no correlation at all with all the other terms. Therefore, I have decided to define the correlation of a term t_r with Q as the average of correlations of t_r and query terms:

$$\text{corr}_Q(t_r) := \frac{\sum\limits_{j=1}^{k} h(t_{i_j}, t_r)}{k} = \frac{\sum\limits_{j=1}^{k} M_{i_j,r}}{k}$$

Note that as we do not want to suggest terms that are already in the query, we can assume $r \neq i_1, ..., i_k$.

We can now define the first query suggestion $\hat{t}_1$ as the term that has the maximum query correlation as defined above, this is $\text{corr}_Q(\hat{t}_1) = \max\limits_{t \in \mathcal{T} \setminus \mathcal{Q}} \text{corr}_Q(t)$. After this first step, we have expanded the query to have one more term $Q_1 := (t_{i_1}, ..., t_{i_k}, \hat{t}_1)$.

We could apply this step iteratively by using corr_{Q_1} in the next step. However, by doing this, we could be including terms that are very similar to $\hat{t}_1$ which would make these suggestions useless. For this reason, it seems more reasonable to keep using the correlation of each term with the original query $\text{corr}_Q(t)$. Additionally, we can try to take terms that are high correlated to the query Q and, at the same moment, have a low correlation with the suggested terms.

Let's define S^i as the set of suggested terms that we have found in step i following this algorithm. For example, in step one, we have $S^1 := \{\hat{t}_1\}$. Analogously to corr_Q, we can define

$$\text{corr}_{S^i}(t_r) := \frac{\sum\limits_{\hat{t} \in S^i} h(\hat{t}, t_r)}{|\mathcal{S}^i|}$$

for all terms $t \in \mathcal{T} \setminus (\mathcal{Q} \cup \mathcal{S}^i)$.

Now, in step $i + 1$ we would like to find a term $t \in \mathcal{T} \setminus (\mathcal{Q} \cup \mathcal{S}^i)$ such that tries to maximise $\text{corr}_Q(t)$ while tries to minimise $\text{corr}_{S^i}(t)$. While there are probably better solutions to find the equilibrium between these two, minimising $\text{corr}_{S^i}(t)$, is equivalent to maximising $\frac{1}{\text{corr}_{S^i}(t)}$, we could summarise this problem by finding a term t that maximises the expression above in the step $i + 1$:

$$\text{corr}_{Q,S^i}(t) = \text{corr}_Q(t) + \frac{1}{\text{corr}_{S^i}(t)}$$

Note that overall we may still want to consider terms that are correlated to the query so we could consider maximising this expression while considering only the subset of terms $t \in \mathcal{T} \setminus (\mathcal{Q} \cup \mathcal{S}^{i-1})$ s.t. $\text{corr}_Q(t) > 0.7$.

Finally, the algorithm could stop after the number of added terms have reached a certain limit, when all $\text{corr}_Q(t)$ are too low (below a threshold), or when all $\text{corr}_{S^i}(t)$ are too high (above a threshold).

In summary:

1. Compute term correlation matrix M
2. First suggestion is a term $\hat{t_1}$ such that $\text{corr}_Q(\hat{t_1}) = \max\limits_{t \in \mathcal{T} \setminus \mathcal{Q}} \text{corr}_Q(t)$. This can be computed by calculating $\text{corr}_Q(t)$ for all terms that are not in the query and taking the one with the maximum value. Matrix M can be used to compute each $\text{corr}_Q(t)$ as seen in the the definition.
3. In step i, the suggestion will be the term $\hat{t_i}$ such that $\text{corr}_Q(\hat{t_i}) > 0.7$ and $\text{corr}_{Q,S^{i-1}}(\hat{t_i}) = \max\limits_{t \in \mathcal{T} \setminus (\mathcal{Q} \cup \mathcal{S}^{i-1})} \text{corr}_{Q,S^{i-1}}(t)$. Matrix M can be used to compute each $\text{corr}_{Q,S^{i-1}}(t)$ as seen in the the definition.
4. The process will keep going until $|S^{i-1}| >$ threshold_corr_set, or $\text{corr}_Q(t) >$ threshold_corr_Q or $\text{corr}_{S^i}(t) <$ threshold_corr_S where thresholds are parameters that can be set by the user.

1 Question 3

Consider the following scenario: a company search engine is employed to allow people to search a large repository. All queries submitted to the system are recorded. A record that contains the id of the user and the terms in the query is stored. The order of the terms is not stored and neither is any timestamp. Each entry in this record is effectively and id and a set of terms. Any duplicate terms in a query is ignored.The designers of the search engine, decide to use this information to develop an approach to make query term suggestions for users, i.e., at run time once a user an entered their queries terms, the system will suggest potential extra terms to add to the query. Given the data available, outline an approach that could be adopted to generate these suggested terms. A brief outline is sufficient that captures the main ideas in your approach. Identify advantages and disadvantages of your approach (briefly)

Answer Question 3 If we were to assume that all users have similar interest, we could come up with an approach that ignores users and focuses only on query terms. However, this is usually not the case, and distinguishing users will probably lead to better results.

At the end, I would like to provide suggestions to a user based on queries of similar users. We could start defining a matrix M that stores the similarity between each user. There are many ways to define this similarity. For example, we could consider all possible pair combinations of each query:

$$\{t_i, t_j, t_k\} \longrightarrow \{(t_i, t_j), (t_i, t_k), (t_j, t_k)\}$$

Then, add up the frequency of each pair per user across all queries as seen in the following table (the same table normalised by the total number of occurrences per user would probably lead to better results):

user	(t_1, t_2)	(t_1, t_3)	...	(t_{n-1}, t_n)
1	6	2	...	5
2	7	4	...	3
3	2	3	...	9
4	5	2	...	10
5	3	12	...	3

This tells us how many times a user has used two terms in the same query. We can now define the matrix M by calculating the correlation between the vectors of two users. This is $M_{ij} = \text{corr}(\text{user i}, \text{user j})$ where corr could be, e.g., Pearson correlation.

When user i searches for a query, we could based the suggested terms only on users that have a correlation greater than 0.7 (or, if there are none, the top N, e.g. top 3 correlated users).

Now, we just have to find a way to select the suggested terms within the set of similar users. In order to do this, we could use the table provided above to consider all pairs of terms where one, and only one of the terms is in the query. Finally, for each pair, we could do the weighted average across all users by using each user correlation with user i and finally, returning the terms with highest average. For example, if the user doing the query is related to, e.g, user j with a correlation of 0.8 and user k with a correlation of 0.87. Then, if the the query is $Q = (t_1, t_2)$ and $\mathcal{T} = \{t_1, t_2, t_3, t_4\}$ we are going to consider the following table

user	(t_1, t_3)	(t_1, t_4)	(t_2, t_3)	(t_3, t_4)
j	1	3	6	5
k	8	4	2	3

and finally do the weighted average taking the correlations of each user

(t_1, t_3)	(t_1, t_4)	(t_2, t_3)	(t_3, t_4)
$1 \cdot 0.8 + 8 \cdot 0.87$	$3 \cdot 0.8 + 4 \cdot 0.87$	$6 \cdot 0.8 + 2 \cdot 0.87$	$5 \cdot 0.8 + 3 \cdot 0.87$

which is:

(t_1, t_3)	(t_1, t_4)	(t_2, t_3)	(t_3, t_4)
7.76	5.88	6.54	6.62

Finally, we could return the term with the highest weight for t_1, which is t_3 with a weighted of 7.76, and the term with the highest weight for t_2, which is t_4 with a weighted average of 6.62.

Note that this is just an example to show how the calculations can be done and in such as small space $\mathcal{T}$ the results are not very reasonable.

The main advantage of this approach is that, it suggests terms based on users that are, in general, doing similar queries. Therefore, we would expect the suggested terms to be quite precise. However, this algorithm does not have a great time complexity as it has to count the frequency of all users for all possible combination of terms and add them up every time that a suggestion has to be done. A good idea could be to locally save the correlation matrix until certain date and used it for some time. On top of that, terms that are returned could be highly correlated and therefore, an extension of this algorithm could be added to consider only terms with low correlation in a similar way as in Question 2.