Advance Data Bases – Project 1

Query Reformulation System

Abhinav Bajaj (ab3900), Rafica AR (ra2688)

List of Files for submission -

- main.py - File with the code of the main framework of the system.
- engine.py - File with the code of the key word engine responsible for extending the query with new terms.
- README.pdf - This file
- Makefile - File containing the steps to compile and run the code on CLIC machines
- Transcript.pdf - File containing the results of 3 test cases

Running the program -

The project is in Python.

Run Command -

```
python main.py <ACCOUNT_KEY> <TARGET_PRECISION> <QUERY>
```

NOTE: In case when query has multiple words then they should be in single quotes. For example,

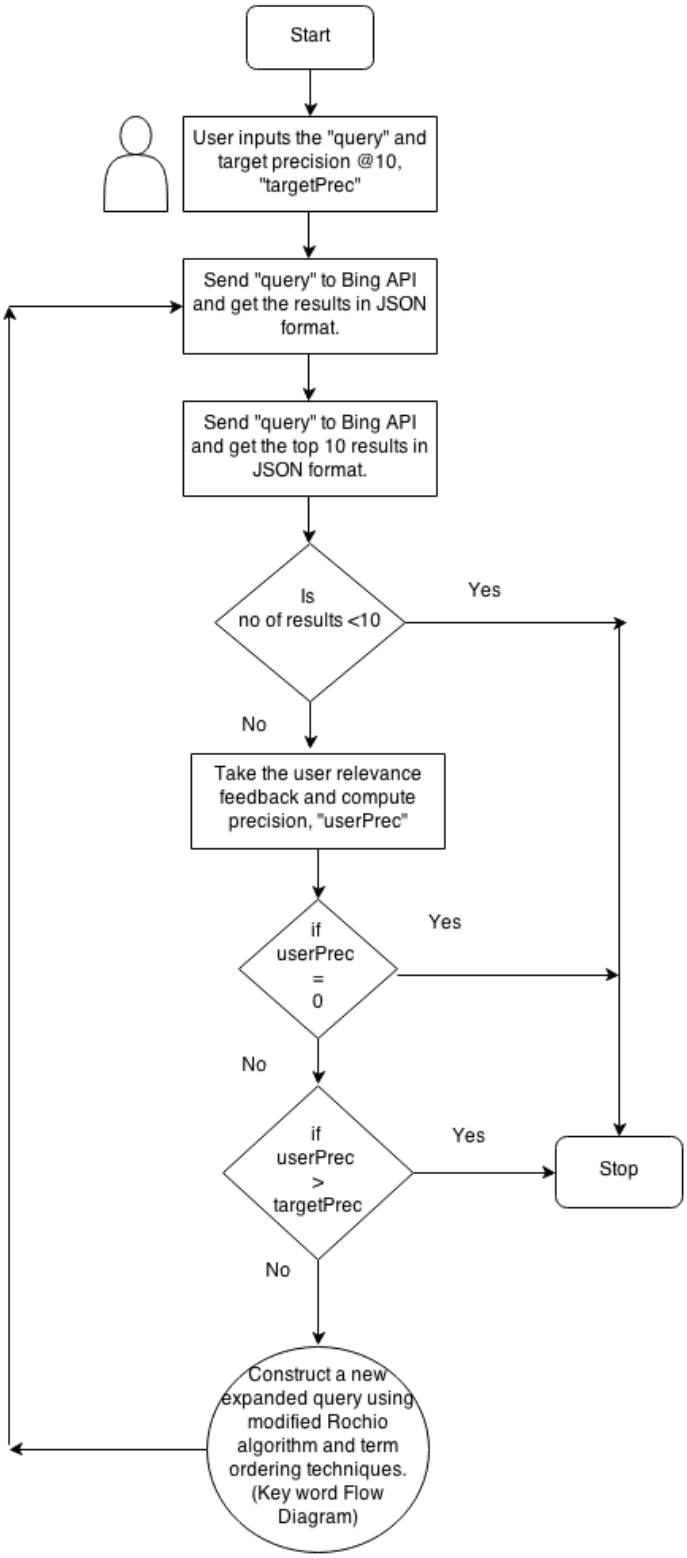
```
python main.py JsV9AIVwzY0l654YiaIXAppMcpvpm7lvkcYdmzJrNcs 0.9 'snow leopard'
```

Bing Search Account Key -

`JsV9AIVwzY0l654YiaIXAppMcpvpm7lvkcYdmzJrNcs`

Design Description -

The major work flow or design of the program is presented in the flow diagram below



Design of Query - Modification method

The basic algorithm behind the system is Rocchio's Algorithm. With each iteration the next query terms are found using the weights calculated using the Rocchio's formula as below -

$$\overrightarrow{Q_m} = (a \cdot \overrightarrow{Q_o}) + \left(b \cdot \frac{1}{|D_r|} \cdot \sum_{\overrightarrow{D_j} \in D_r} \overrightarrow{D_j}\right) - \left(c \cdot \frac{1}{|D_{nr}|} \cdot \sum_{\overrightarrow{D_k} \in D_{nr}} \overrightarrow{D_k}\right)$$

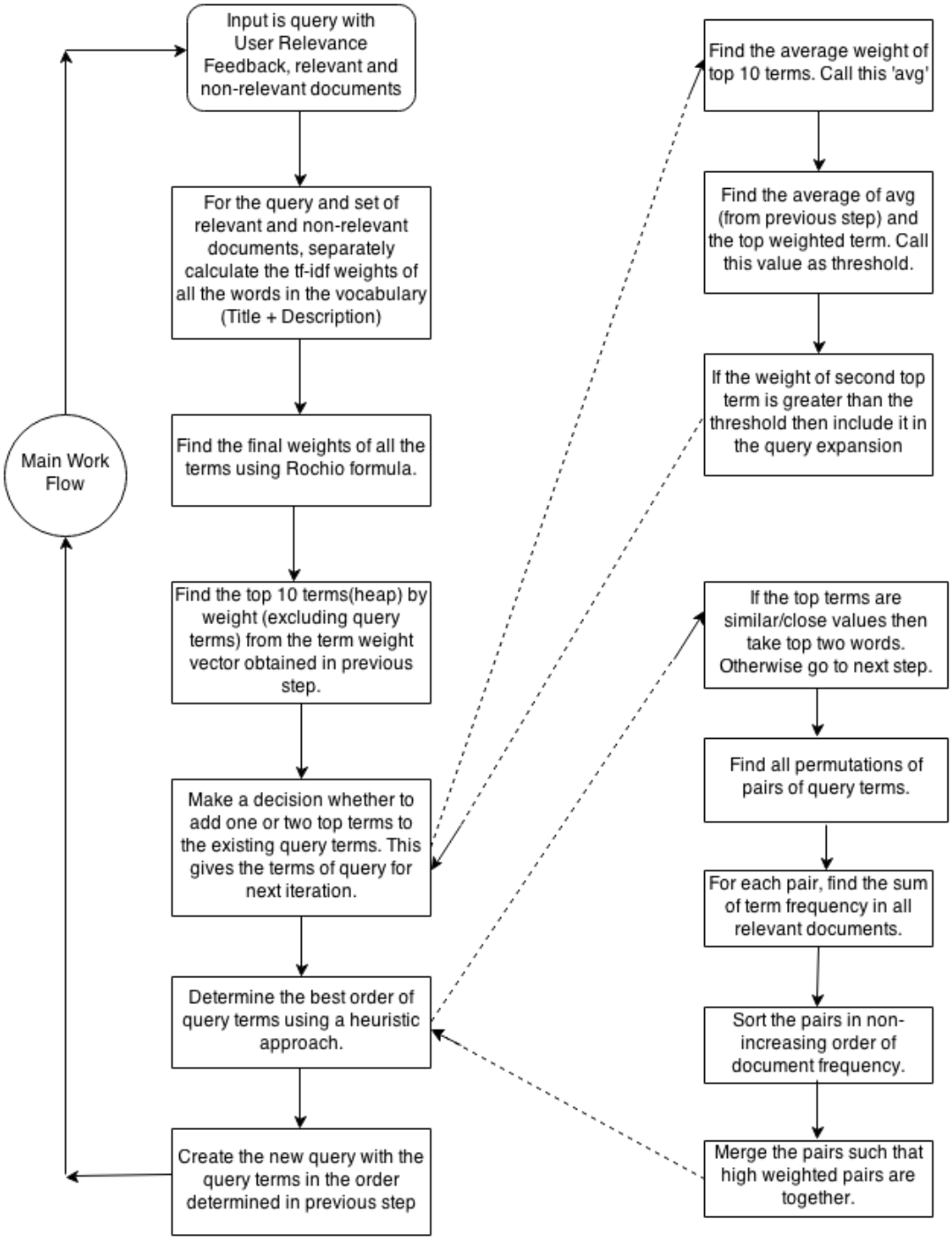
Variable	Value
$\overrightarrow{Q_m}$	Modified Query Vector
$\overrightarrow{Q_o}$	Original Query Vector
$\overrightarrow{D_j}$	Related Document Vector
$\overrightarrow{D_k}$	Non-Related Document Vector
a	Original Query Weight
b	Related Documents Weight
c	Non-Related Documents Weight
D_r	Set of Related Documents
D_{nr}	Set of Non-Related Documents

Source: Wikipedia

The values in the system are - a(alpha) = 1.0, b(beta) = 0.75, c(gamma) = 0.15

The system uses many techniques to determine the term weight vector for the query, related documents and non-related documents, used in the Rocchio's formula, to give better results. These techniques are explained later in the document.

The major work flow or design of the query modification system is presented in the flow diagram below -



Improvement Techniques

Calculation of Term Weight Vector

a. **More Weightage for Quality documents** - The algorithm calculates the term frequency, tf, for all the terms in the vocabulary. The vocabulary consists of all the terms in title and description of the documents (query, relevant or non-relevant set documents). The special case is for handling of terms that appears in relevant quality documents. The system maintains a list of quality documents and checks if the term is from a quality document then the weightage of that term is increased by 20%. Presently the system has only Wikipedia in the list of quality documents. We can add new document(or URLs) to this list. So, for all the terms that appears in a relevant Wikipedia document then the contribution of these terms to term frequency is increased.

b. **More Weightage for Title terms** - The algorithm calculates the term weights using tf-idf formula for all the terms in the set of documents. Afterwards, we increase the tf-idf weightage of the terms that appear in the title of the document. The strategy here is to calculate a relative increase depending on how many times the term appears in the title of the set of documents. The relative increase here is capped at 20%. So for example, if there is a term which appears in all the relevant documents then its weightage(tf-idf) value will be increased by 20%. But if the term appears in title half of the relevant documents then its weightage is increased by 10%.

Decision to select top one or two terms

Once the term weight vector is available, the system has to decide whether to add one or two top terms(highest by weights) to the query. The system uses a heuristic approach in making this decision. The idea behind this approach is to handle various scenarios of weights of top terms in the term weight vector. Below are few examples of these scenarios, showing the weights in non-increasing order -

5, 4.8, 4.8, 3.3, 2.7, 2.5, 2.0, 1.8, 1.7, 1.3, 1 - System selects top two terms with weights 5 & 4.8.

a. 5, 4.1, 2.9, 2.7, 2.5, 2.3, 1.9, 1.4, 1.1, 0.4, 0 - System selects top two terms with weights 5 & 4.1.
b. 5, 4.5, 4.5, 4.3, 3.3, 2.3, 1.5, 1.2, 1.0, 0, 0, 0 - System only selects top term with weight 5.
c. 5, 3.3, 3.2, 2.6, 2.2, 1.8, 1.4, 1.2, ,1.2, 1, 1 , 1, 0, 0, - System only selects top term with weight 5.

Algorithmic Approach -

If the top terms are similar/close values then take top two words. Values are similar is the below condition is true -

```
weight(top_term)  - (0.05 * weight(top_term)) < weight(2nd top term)
```

Otherwise, Find the average weight of top 10 terms. Call this 'avg'.

a. Find the average of avg (from previous step) and the top weighted term. Call this value as threshold.

b. If the weight of second top term is greater than the threshold then include it in the query expansion

NOTE : We are always adding top term to the query. The decision has to be taken for the second highest term.

Determining the best order of terms

Once the system has found all the terms in the new query (terms in previous iteration + new terms), it will use a heuristic approach to determine a best possible ordering of those terms.

Algorithmic Approach -

c. Find all permutations of pairs of query terms.

d. For each pair, find the sum of term frequency in all relevant documents.

e. Sort the pairs in non-increasing order of document frequency.

f. Merge the pairs such that high weighted pairs are together.

For example -

Lets say the query terms are 'columbia', 'university', 'campus', and the weightage (sum of term frequency in all relevant documents) of the pairs of these terms is below -

('columbia', 'university') - 12

('university', 'columbia') - 2

('campus', 'university') - 6

('university', 'campus') - 7

('campus', 'columbia') - 4

('columbia', 'campus') - 10

The algorithm will sort the above pairs w.r.t. the weight. It will take the highest pair and then try to combine the next highest pair with it and in this case the result will be ('columbia', 'university', 'campus').