Project Information Retrieval – Part 2

Adham Mohamed, Veronica Orsanigo

1 Introduction

In this second part we present some techniques to improve performances of a search engine. There are some issues that occur while searching: the same word can have different meanings (polysemy), two different words can have the same meaning (synonymy), vocabulary of searcher may not match that of the documents. Since the request made within a query could need some more results than the only ones where the query terms perfectly match with document terms, we have to make our search engine able to find correspondences based on a larger similarity. In particular we are interested in retrieving also documents where synonyms or words close to the ones of our query occur, we are interested in semantic too, in the topic where the query is inserted.

For example, if we consider the query *"plane fuel"*, exact matching will miss documents containing *aircraft*, *airplane* or *jet*.

There are two main different ways to improve performances:

- Feedback relevance:
  It is local, based on the analysis of the single query, it tries to make a new query from the starting one analyzing the feedback from the retrieved relevant and not relevant documents. It works with query and document vectors, so it modifies the final query vector giving a larger weight to the relevant documents.
- Query expansion:
  It is global, it uses thesaurus to keep a list of possible terms that can expand the one from the starting query. Thesaurus can be manual or automatically derived, in the first case for each term of the query there are words that the thesaurus lists as semantically related with it, in the second case the generation of the thesaurus is automatic, it follows similarity terms rules of co-occurrence or grammatical relation with same words.

The code and the complete results on GitHub:

https://github.com/adham95/IR_Assignment

1.1 Feedback relevance

This method is grounded on relevance of document retrieved, this feedback can be given explicitly (explicit feedback) by users or implicitly (implicit feedback), for example checking if a document is selected for viewing, the duration of time spent viewing a document, or page browsing or scrolling action. Finally there is also pseudo-relevance, where the feedback action is made automatically, assuming that the k-top ranked documents are relevant. [1]

A particular well known algorithm for feedback relevance is Rocchio algorithm. Its aim is to find the optimal query $\vec{q}_{opt}$, so to move the query vector towards relevant documents and away from nonrelevant documents.

Rocchio algorithm's formula is:

$$\vec{q}_m = \alpha\overrightarrow{q_0} + \beta\frac{1}{|D_r|}\sum_{\overrightarrow{d_J} \epsilon D_r}\overrightarrow{d_J} - \gamma\frac{1}{|D_{nr}|}\sum_{\overrightarrow{d_J} \epsilon D_{nr}}\overrightarrow{d_J}$$

Where:

$\vec{q}_m$: modified query vector

$\overrightarrow{q_0}$: original query vector

D_r and D_{nr} : sets of known relevant and nonrelevant documents

α, β, γ : weights attached to each component.

1.2 Query expansion

The query is modified based on global resource, so not query-dependent. Often the problem aims to find (near-)synonyms.

This is possible thanks to different methods:

- Use of a controlled vocabulary that is maintained by human editors
- Manual thesaurus
- Automatically derived thesaurus
- Query reformulations based on query log mining

The automatic query expansion based on co-occurrence data has been studied for nearly three decades, there are different methods that can be classified into four groups:

- Simple use of co-occurrence data. The similarities between terms are calculated based on the association hypothesis and then used to classify terms by setting a similarity threshold value. So a set of index terms is subdivided into classes of similar terms and a query is expanded by adding all the terms of the classes that contain query terms.
- Use of document classification. Documents are first classified. Infrequent terms found in a document class are considered similar and clustered in the same term class.
- Use of syntactic context. The term relations are generated on the basis of linguistic knowledge and co-occurrence statistics. The method uses a grammar and a dictionary to extract for each term t a list of terms.
- Use of relevance information. Relevance information is used to construct a global information structure. A query is expanded by means of this global information structure.

2 Data and results

In this section we want to compare different methods to search for documents that match our queries.

- Standard Searcher [2]

  This searcher is the one implemented following the standard given on Lucene web site and used for examples from the previous report (part 1).

- Rocchio algorithm [3]

  To create Rocchio algorithm we first needed to implement the tf_idf for terms and so create the corresponding vector of the query.

  Later we implemented Rocchio algorithm giving more weight to the relevant documents, in order to obtain the new optimal query to pass to the searcher.

  In particular we proceeded as follows:

  - We represented each query and document as tf-idf vector
  - For each query we retrieved top 50 documents using search.py (presented in part 1), with StandardAnalyzer and BM25Similarity
  - We applied Rocchio algorithm to update each query vector by considering top 10 retrieved documents as relevant ones (so applying pseudo-relevance, even if we know it can have some limits with some queries), since we previously saw that BM25Similarity worked not that bad and to make the procedure more automatic. We set $\alpha = 1, \beta = 0.65, \gamma = 0$
  - From the updated query vector, we picked up the top 10 terms to obtain the updated query
  - We retrieved again documents, but now for the updated query

- Query expansion [4]

  We made query expansion using WordNet, a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations.
  So we could use it to have the words to use to expand the queries.
  Later we obtained the expanded query from the chosen title and we searched for it to have the new results.

Since the set of documents is very large, our approach was to take a subset of it, so we took some titles as queries and we worked on them.

Also here, to evaluate the system using different searching techniques, we were interested in precision and recall.

P = measure of “how much correct found”

R = measure of “how much of correct found”.

3 Examples

We analyze here how the system evolves using different techniques to modify the original query.

As first approach we manually checked the results for the following queries, in order to decide which were the correct answers before and after applying Rocchio algorithm and query expansion.

Y = correct answer for the query

N = not correct answer for the query

Red = document with title equal to the query

Technique	Document	Top-10	Query (original/modified)
	Q1171		What is the most efficient graph data structure in Python?
BM25		Q1171.xml Y Q852334.xml N Q262232.xml N Q10025584.xml N Q1005494.xml Y Q902910.xml N Q463240.xml Q1094215.xml Q238008.xml Q872290.xml	
Rocchio		Q852334.xml N Q872290.xml N Q1171.xml Y Q902910.xml N Q262232.xml N Q628192.xml Y Q671403.xml Q990469.xml Q10025584.xml Q1005494.xml	structure graph efficient what most in data python the is
Query expansion		Q1139449.xml N Q1321062.xml N Q1457045.xml N Q1368822.xml N Q572808.xml N Q561949.xml N Q1392604.xml Q1074200.xml Q1171.xml Q598931.xml	What efficient effective graph graphical_record chart data information datum structure construction anatomical_structure python Python
	Q1350396		Error while inserting pointer to a vector
BM25		Q914460.xml Y Q1222926.xml Y Q485358.xml Y Q555223.xml N Q1052135.xml N	

		Q947394.xml N Q1084251.xml Q488729.xml Q313432.xml Q995281.xml	
Rocchio		Q914460.xml Y Q485358.xml Y Q947394.xml N Q1280513.xml N Q1222926.xml N Q488729.xml N Q621233.xml Q62340.xml Q1436020.xml Q313432.xml	a inserting vector pointer to error while function strongpwrite have
Query expansion		Q1384301.xml N Q326922.xml N Q891913.xml Y Q1381840.xml N Q1472019.xml N Q405617.xml N Q995281.xml Q1039627.xml Q901704.xml Q739870.xml	mistake error fault insert infix enter arrow pointer cursor vector transmitter
	Q10880		Any good advice on using emacs for C++ project?
BM25		Q671412.xml Y Q557555.xml N Q79210.xml Y Q10880.xml Y Q1078069.xml N Q1416882.xml N Q445595.xml Q623040.xml Q1231397.xml Q495579.xml	
Rocchio		Q671412.xml Y Q557555.xml N Q1114305.xml Y Q842918.xml Y Q1209497.xml N Q79210.xml Y Q10880.xml Q1078069.xml Q1416882.xml Q24109.xml	on good advice any using emacs for c project isn
Query expansion		Q454664.xml N Q708684.xml N Q396074.xml N Q1209497.xml N Q807846.xml N Q1436545.xml N Q314553.xml Q1027785.xml Q1239433.xml Q883332.xml	Any good goodness commodity advice exploitation victimization victimisation degree_centigrade degree_Celsius C undertaking project task
	Q2933		How can I create a directly-executable cross-

			platform GUI app using Python?
BM25		Q818736.xml N Q1147199.xml N Q2933.xml Y Q1459087.xml N Q205062.xml N Q1060679.xml N Q450136.xml Q5313.xml Q716524.xml Q796364.xml	
Rocchio		Q1147199.xml N Q818736.xml N Q2933.xml Y Q1459087.xml N Q5313.xml Y Q1060679.xml N Q205062.xml Q450136.xml Q1029435.xml Q81584.xml	gui i platform can a app cross directly executable python
Query expansion		Q758023.xml N Q13607.xml Y Q636990.xml Y Q722098.xml N Q692566.xml Y Q211830.xml N Q977653.xml Q707491.xml Q1043114.xml Q402598.xml	create graphical_user_interface GUI
	Q638360		Python How to calculate equal parts of two dictionaries
BM25		Q327311.xml Y Q1154378.xml N Q526125.xml Y Q399957.xml N Q638360.xml Y Q285938.xml N Q1317410.xml Q631463.xml Q1055410.xml Q1165352.xml	
Rocchio		Q327311.xml Y Q1085617.xml N Q592746.xml N Q1456617.xml N Q38987.xml Y Q1154378.xml N Q526125.xml Q399957.xml Q638360.xml Q285938.xml	equal calculate dictionaries two how parts python to of other
Query expansion		Q431082.xml N Q526125.xml Y Q1165352.xml Y Q327311.xml Y Q285938.xml N Q49137.xml N	python Python calculate cipher cypher peer equal match parts part portion two 2 II dictionary lexicon

		Q581209.xml Q638360.xml Q609972.xml Q1024847.xml	
	Q1124667		What is the best practice in deploying application on Windows
BM25		Q261829.xml Y Q455904.xml N Q585165.xml N Q1208963.xml N Q1222626.xml Y Q1121725.xml N Q1424169.xml Q941809.xml Q467899.xml Q848126.xml	
Rocchio		Q261829.xml Y Q455904.xml Y Q585165.xml N Q1208963.xml N Q1222626.xml Y Q1121725.xml N Q1424169.xml Q941809.xml Q467899.xml Q848126.xml	application is the what in windows practice best deploying on
Query expansion		Q1305632.xml N Q401118.xml N Q797771.xml N Q646427.xml N Q1365729.xml N Q651358.xml N Q1409192.xml Q1161618.xml Q259634.xml Q1367130.xml	best topper Best practice pattern exercise deploy application practical_application coating Windows window windowpan
	Q827393		Default value for bool in C
BM25		Q827393.xml Y Q1059630.xml Y Q412611.xml Y Q1142209.xml N Q663724.xml N Q529210.xml N Q1123725.xml Q886729.xml Q1409454.xml Q365198.xml	
Rocchio		Q827393.xml Y Q1185923.xml N Q565765.xml Y Q183606.xml N Q609937.xml N Q1166729.xml N Q1057724.xml Q1110658.xml Q913020.xml Q54867.xml	bool in default value c for other about function have
Query expansion		Q620137.xml N Q563221.xml N	default nonpayment nonremittal value economic_value

		Q1059630.xml Q687718.xml Q1123725.xml Q827393.xml Q1142209.xml Q1190112.xml Q202718.xml Q507971.xml	Y N Y Y	time_value degree_centigrade degree_Celsius C

We showed the top-10 documents retrieved to see if the document corresponding to the title used as query appears in that set, but we calculated precision (as in the previous report, we could say precision@6) and recall considering the top-6 documents.

As denominator for the recall we use "X" since we don't know the number of the relevant documents.

Queries:

- "What is the most efficient graph data structure in Python?"

  BM25:
  Precision = 2/6
  Recall = 2/X
  Rocchio:
  Precision = 2/6
  Recall = 2/X
  Query expansion:
  Precision = 0/6
  Recall = 0/X

- "Error while inserting pointer to a vector"

  BM25:
  Precision = 3/6
  Recall = 3/X
  Rocchio:
  Precision = 2/6
  Recall = 2/X
  Query expansion:
  Precision = 1/6
  Recall = 1/X

- "Any good advice on using emacs for C++ project?"

  BM25:
  Precision = 3/6
  Recall = 3/X
  Rocchio:
  Precision = 4/6
  Recall = 4/X
  Query expansion:

Precision = 0/6
Recall = 0/X

- "How can I create a directly-executable cross-platform GUI app using Python?"

  BM25:
  Precision = 1/6
  Recall = 1/X
  Rocchio:
  Precision = 2/6
  Recall = 2/X
  Query expansion:
  Precision = 3/6
  Recall = 3/X

- "Python How to calculate equal parts of two dictionaries"

  BM25:
  Precision = 3/6
  Recall = 3/X
  Rocchio:
  Precision = 2/6
  Recall = 2/X
  Query expansion:
  Precision = 3/6
  Recall = 3/X

- "What is the best practice in deploying application on Windows"

  BM25:
  Precision = 2/6
  Recall = 2/X
  Rocchio:
  Precision = 3/6
  Recall = 3/X
  Query expansion:
  Precision = 0/6
  Recall = 0/X

- "Default value for bool in C"

  BM25:
  Precision = 3/6
  Recall = 3/X
  Rocchio:
  Precision = 2/6
  Recall = 2/X
  Query expansion:

Precision = 3/6
Recall = 3/X

Since this approach is restrictive (we can not manually check a large number of queries), we decided to make also a larger-scale experiment, that is the same as the experiment number 2 of the report 1 (the complete description is written there).
We looked at the average position of the document that has the query as title using different techniques: normal searcher with BM25Similarity, Rocchio algorithm, query expansion.
We used 100 queries (for Rocchio and query expansion we used the updated query) and we calculated the average position on them, retrieving the top-50 documents.

BM23Similarity: average position = 15.2

Rocchio algorithm: average position = 19.6

Query expansion: average position = 35.08

4 Analysis of results

If we look at the manual results, we can see how relevant documents are given in greater number sometimes using Rocchio algorithm, sometimes using query expansion, but with both the techniques we lose precision, since the modified queries can bring us away from our goal. Query expansion, in particular, seems not to help so much if we use sentences as query (titles), the expansion is not so accurate and we lose important documents, as we can see in results that are zero.

If we look at the results of the second experiment, on a larger-scale, we can see how the average position of the document that has the query as title is higher if we use BM23Similarity, then there is Rocchio algorithm and finally query expansion, with a big gap between it and query expansion.

So we could say that Rocchio algorithm can increase recall, but we lose in precision, as we expected.

If we analyze query expansion, we can see how deeply it changes the query, so that we often lose a large part of documents, so, for our queries, that are long, since they are sentences and not single words, this technique is not optimal.

Also here we observed that the performance of the two techniques is dependent on the particular query we are dealing with, suggesting us an oscillating behavior, in particular this seems to be true for the query expansion, where the synonyms taken from WordNet thesaurus do not always satisfy the request of the queries, maybe with a too specific and technical language.

5 Conclusion

The two techniques we examined here work on modifying queries, changing and adding terms to them. This approach can increase recall, but at the same time can be dangerous, since it can bring us far from our aim, as seen for query expansion. Moreover, since we have not a robust set of relevant documents (we do not have a real feedback by users), it is not always easy to make Rocchio algorithm work at its best.

Bibliography

[1] https://www.cl.cam.ac.uk/teaching/1617/InfoRtrv/lecture7-relevance-feedback.pdf .

[2] https://svn.apache.org/viewvc/lucene/pylucene/trunk/samples/SearchFiles.py?view=markup.

[3] https://github.com/siddumsp1997/Relevance-feedback-and-Query-expansion.

[4] https://github.com/ellisa1419/Wordnet-Query-Expansion/blob/master/wordnet.py.