Information retrieval project

Irene Brugnara

January 2021

Outline

Main features of the project:

- Probabilistic model: BM25
- Relevance feedback and pseudo-relevance feedback
- Free-form text queries
- Pure Python implementation
- Evaluation with a test collection: the LISA corpus

The dataset

The LISA collection contains titles and abstracts of 5999 documents from the Library and Information Science Abstracts database.

I simply merged together title and abstract fields.

Insight into the corpus

An example of document:

```
PRESERVATION OF READING MATERIALS IN LIBRARIES: A
PRACTICAL APPROACH.

THERE HAS BEEN AN ALARMING INCREASE IN THE DETERIORATION
OF READING MATERIALS IN LIBRARIES, DUE TO CARELESS
HANDLING BY LIBRARY STAFF AND READERS. CAREFUL HANDLING,
PROPER SHELVING AND TIMELY BINDING CAN CHECK FUTURE
DAMAGE. DESCRIBES SOME IMPORTANT PREVENTIVE MEASURES FOR
ENVIRONMENTAL AND INSECT CONTROL.
```

Insight into the corpus

Top 20 words with highest document frequency:

the, of, and, in, to, a, for, library, on, is, are, by, with, information, libraries, an, as, from, which, at

Text processing

Steps for pre-processing documents and queries:

- Normalisation: remove punctuation and lowercase
- Tokenisation: split by space
- Stop words removal: custom stop list obtained by taking the first 20 terms with highest document frequency (see previous slide)
- Stemming: Porter stemmer (NLTK library)

The ranking function

I used the following ranking function, in the framework of Okapi BM25 with relevance feedback. The Retrieval Status Value of a document d for a query q is the sum over all query terms of a weight which is the product of three contributions:

$$RSV_d = \sum_{t \in q, t \in d} w_{td}^{DTF} \times w_{tq}^{QTF} \times w_t^{RSJ}$$

The document term frequency weight

The first contribution accounts for term frequency tf_{td} in the document

$$w_{td}^{DTF} = \frac{(k_1 + 1)tf_{td}}{k_1((1 - b) + b(L_d/L_{ave})) + tf_{td}}$$

where k_1 is the parameter calibrating the document term frequency scaling and b is the parameter which regulates document length normalisation.

The query term frequency weight

The second contribution accounts for term frequency in the query, and it is useful in case of long queries, as is the case with my test queries (they are paragraph-long information needs)

$$w_{tq}^{QTF} = \frac{(k_3 + 1)tf_{tq}}{k_3 + tf_{tq}}$$

where k_3 is the parameter tuning term frequency in the query.

The Robertson-Sparck-Jones weight

The last contribution is a Robertson-Sparck-Jones weight with smoothing

$$w_t^{RSJ} = \log \left[\frac{(|VR_t| + 0.5)(N - |VR| - df_t + |VR_t| + 0.5)}{(df_t - |VR_t| + 0.5)(|VR| - |VR_t| + 0.5)} \right]$$

where $|VR|$ is the number of documents judged relevant by the user, $|VR_t|$ is the number of judged relevant documents containing term t, df_t is the document frequency for term t and N is the total number of documents in the collection.

The Robertson-Sparck-Jones weight

The RSJ weight derives from the Binary Independence Model: it can be written as a log odds ratio

$$w_t^{RSJ} = \log\left(\frac{p_t}{1-p_t}\frac{1-u_t}{u_t}\right)$$

where p_t is the probability of term t appearing in a document given that the document is relevant, and u_t is the probability of t appearing in a document given that the document is non-relevant, estimated in this way:

$$p_t = \frac{|VR_t| + 0.5}{|VR| + 1} \qquad u_t = \frac{df_t - |VR_t| + 0.5}{N - |VR| + 1}$$

The estimate for u_t assumes that all documents outside VR are non-relevant, which is a reasonable approximation because usually relevant documents are a small fraction of the collection.

A rough estimate

In absence of relevance information

$$|VR| = |VR_t| = 0$$

$$p_t = 0.5 \qquad u_t = \frac{df_t + 0.5}{N + 1}$$

$$w_t^{RSJ} = \log \frac{N - df_t + 0.5}{df_t + 0.5} \approx idf_t$$

I used this rough estimate for providing an initial set of results for the user, among which he can start searching for relevant documents.

Having removed stop words, $df_t > N/2$ and the weight is always positive.

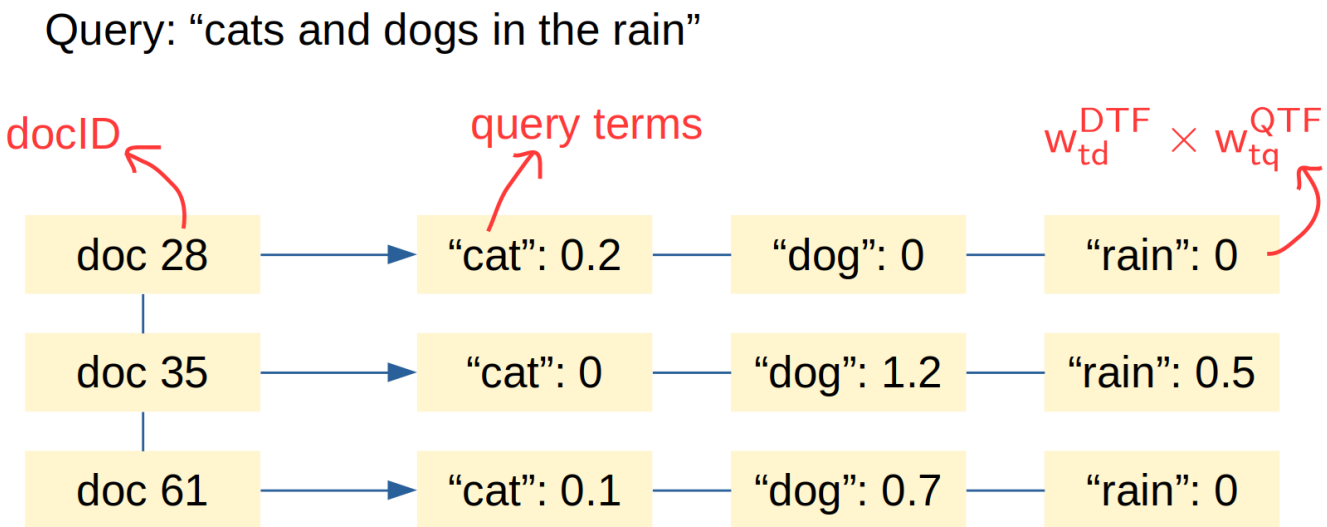
Relevance feedback

I applied relevance feedback in an iterative way: the system initially retrieves a rank of documents with the rough estimate of previous slide, the user provides a set of relevant documents VR, the system updates the results according to this evidence by recomputing the scores, the user finds some other relevant documents, and so on, until the user is satisfied.

I assume that the user does not change his information need after seeing new results from relevance feedback, therefore I keep memory of previously judged relevant documents across relevance feedback iterations.

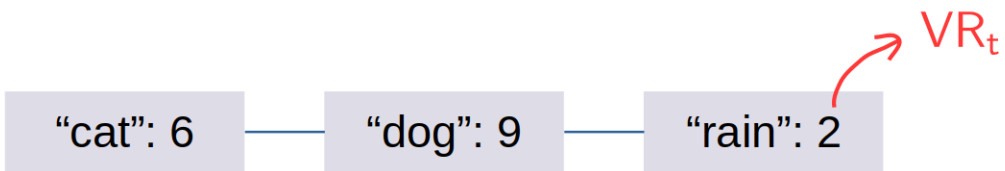
Implementation

The weights $w_{td}^{DTF} \times w_{tq}^{QTF}$ are computed when the user submits the query and they are stored in a dictionary whose keys are docIDs and whose values are dictionaries from query terms to these weights, because they don't change across relevance feedback iterations.
To build this dictionary, for each term t in the query I scan the corresponding posting list and save the score of each document d in the posting list.



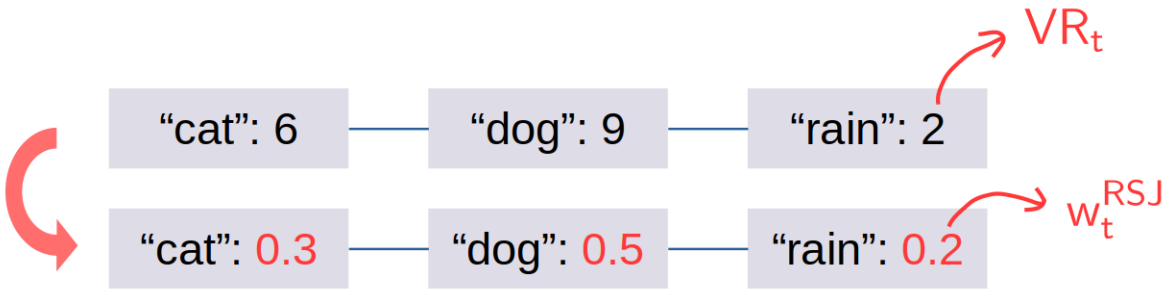
Implementation

I store the counts VR_t in a dictionary (with one item per term t) which is updated at every iteration of relevance feedback in this way: for all new documents d judged relevant and for all query terms t, I increment VR_t by 1 if d contains t, i.e. if $w_{td}^{DTF} \times w_{tq}^{QTF} > 0$ which I can check just by looking up the other dictionary.



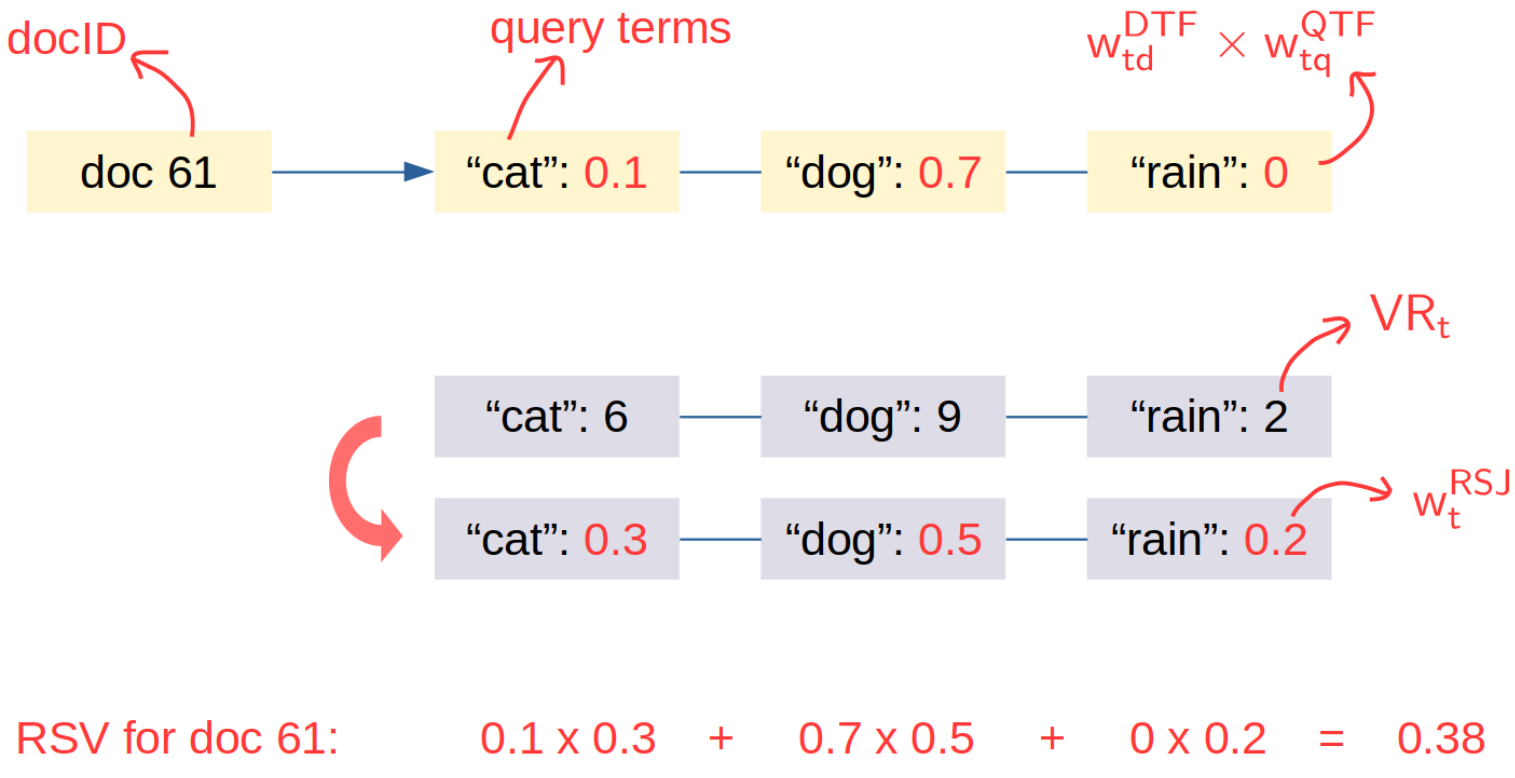
Implementation

I store the counts VR_t in a dictionary (with one item per term t) which is updated at every iteration of relevance feedback in this way: for all new documents d judged relevant and for all query terms t, I increment VR_t by 1 if d contains t, i.e. if $w_{td}^{DTF} \times w_{tq}^{QTF} > 0$ which I can check just by looking up the other dictionary.
Then, from VR_t I compute the weights w_t^{RSJ}

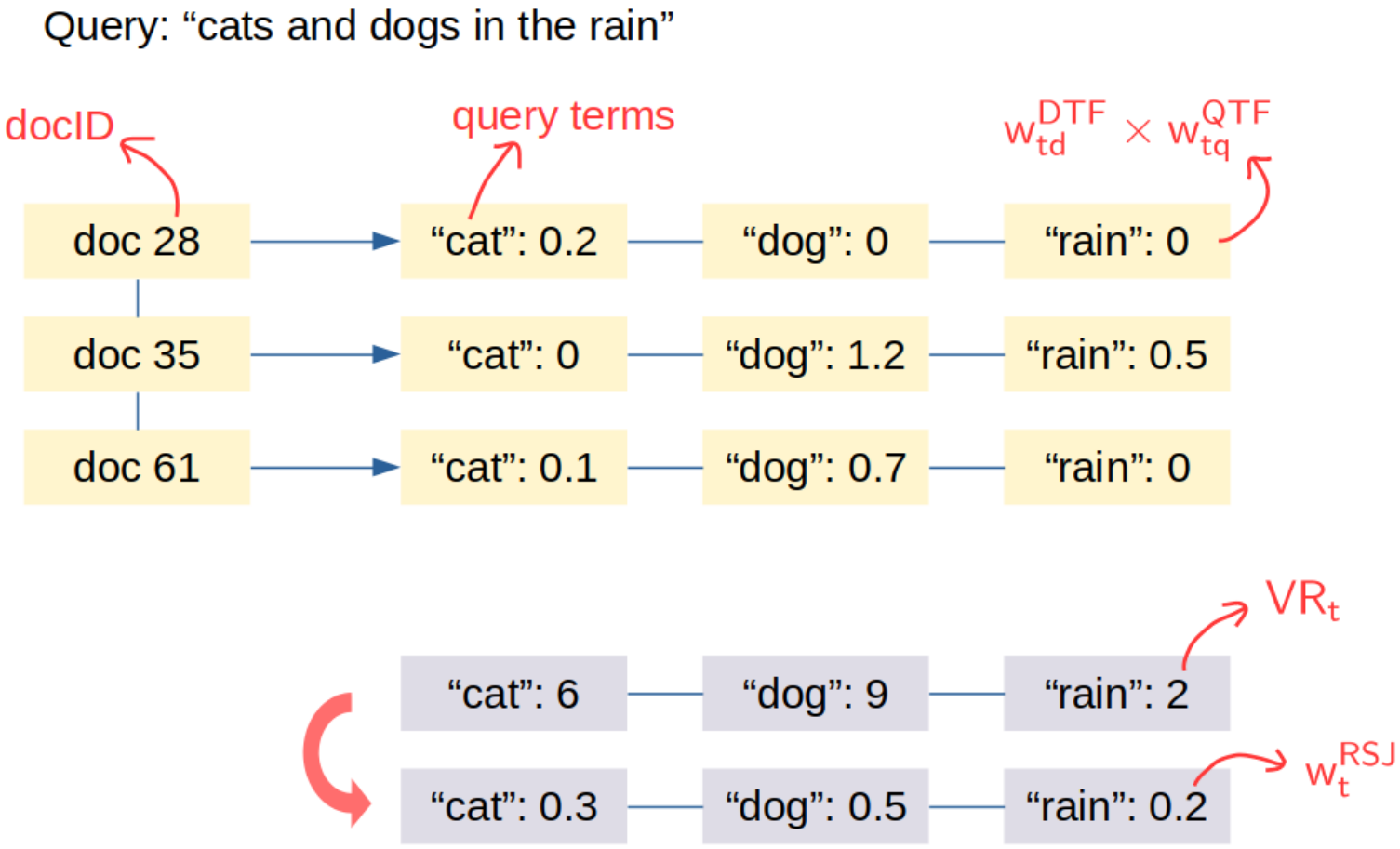


Implementation

Then, from VR_t I compute the weights w_t^{RSJ} and combine them with $w_{td}^{DTF} \times w_{tq}^{QTF}$ to get the total RSV for each document d in the rank.



Implementation



RSV for doc 61:	0.1 x 0.3	+	0.7 x 0.5	+	0 x 0.2	= 0.38
RSV for doc 35:	0 x 0.3	+	1.2 x 0.5	+	0.5 x 0.2	= 0.7
RSV for doc 28:	0.2 x 0.3	+	0 x 0.5	+	0 x 0.2	= 0.06

Pseudo relevance feedback

In the case of pseudo relevance feedback, I used the first k ranked documents as the set VR of relevant documents, where k is a parameter. This means that, unlike the case of relevance feedback with the user, I don't keep memory of which documents are relevant across iterations, as the relevant documents may become non-relevant whenever they exit the top k positions.

The convergence condition is that the first k ranked documents are the same in two consecutive iterations, because this means the whole ranking will remain the same in all subsequent iterations.

Evaluation

The LISA dataset provides 35 test queries with relevance judgements. For each query, there are between 1 and 53 relevant documents, 10 on average.

Disclaimer: the README file of the dataset says *"I would guess that probably 80% of the relevant documents have been identified."*

No parameter tuning because the set of test queries is too small to do a train/test split; in absence of optimisation, reasonable values are $k_1 \in [1.2, 2], b = 0.75$ [1]
Choice of parameters: $k_1 = k_3 = 1.5$, $b = 0.75$, and $k = 5$ for pseudo-relevance

[1] Schütze H., Manning C. D., Raghavan P. (2008). Introduction to information retrieval

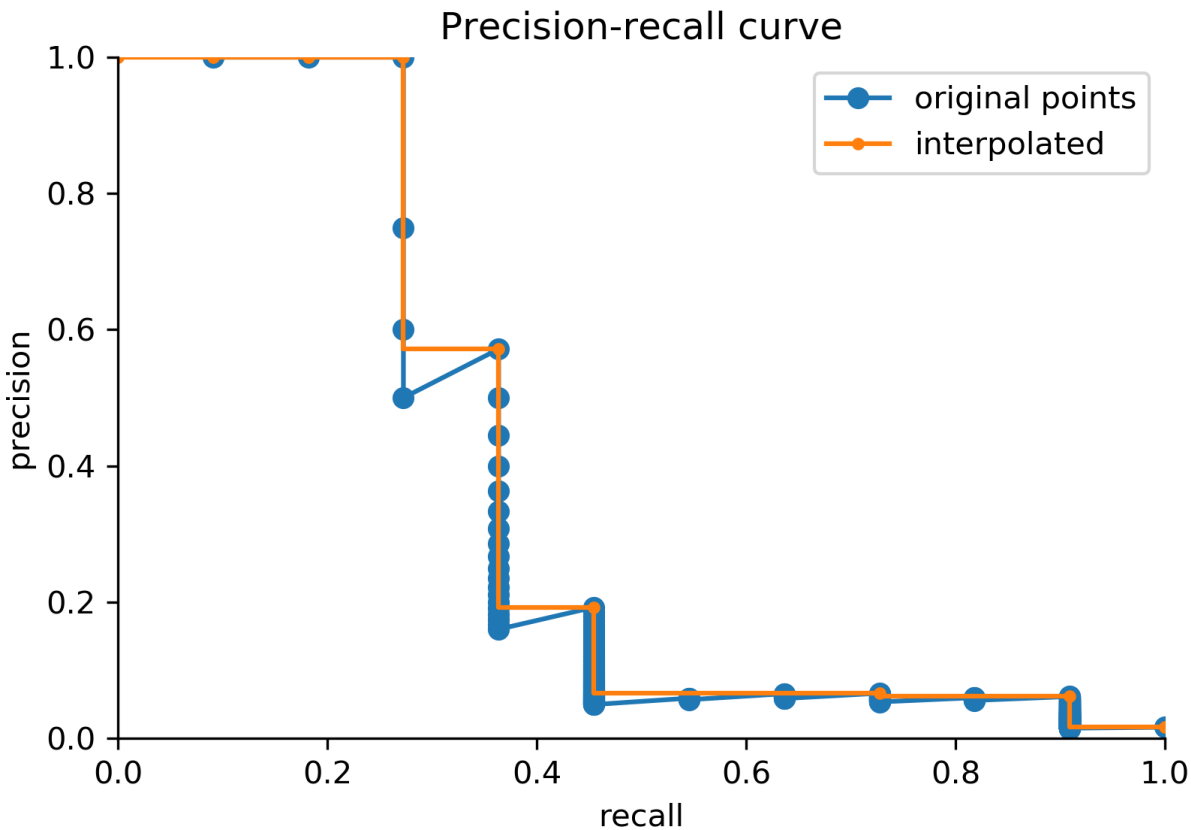
Test queries

Test queries are paragraph-long, written in natural language.
An example:

I WOULD BE PLEASED TO RECEIVE ANY PAPERS DESCRIBING FREE TEXT RETRIEVAL PACKAGES LIKE CAIRS, ASSASIN, STATUS, DECO OR STAIRS, ESPECIALLY IF THE FEATURES OF DIFFERENT PACKAGES ARE COMPARED, OR IF THE IMPLEMENTATION AND USE OF A SYSTEM IS DESCRIBED. I AM ESPECIALLY INTERESTED IN PACKAGES FOR MINIS AND MICROS AND IN ANY ASSOCIATION BETWEEN FREE TEXT RETRIEVAL PACKAGES AND DATABASE MANAGEMENT SYSTEMS. MINICOMPUTERS, MICROCOMPUTERS, DATABASE MANAGEMENT SYSTEMS, FREE TEXT RETRIEVAL

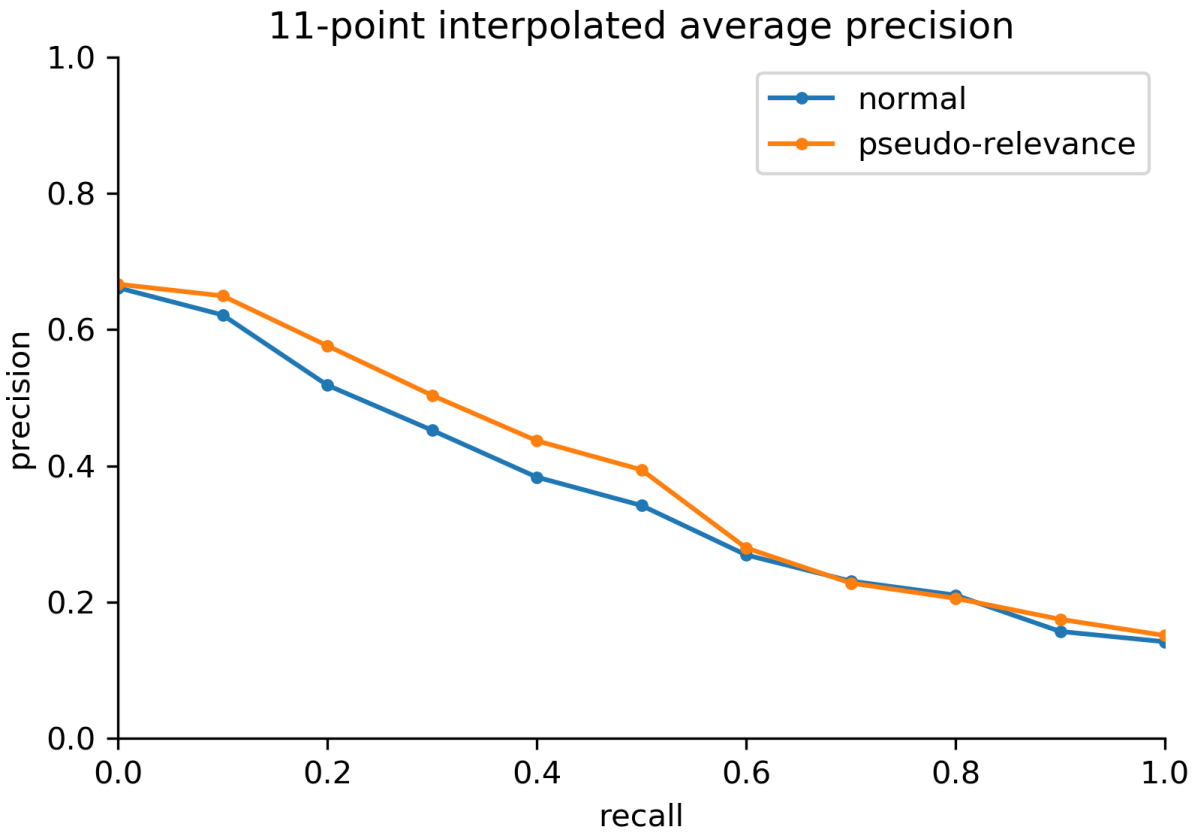
Precision-recall curves

I plotted precision-recall curves for each test query. Here is the result for the query of previous slide:



Average performance on test collection

Mean Average Precision (MAP): 0.348
MAP after pseudo-relevance feedback: 0.370



MAP with the Binary Independence Model: 0.195

Assessment of relevance feedback

How to assess the effectiveness of relevance feedback with the user, by using test queries?

I measured the MAP before and after giving the IR system one relevant document among the set of judged relevant documents (averaged across which relevant document to pick), excluding this relevant document from the computation of the MAP (both before and after) for fairness. The idea is that obviously the known relevant document will be ranked higher, but we would expect that also the other relevant documents go up in the rank.

MAP before feedback: 0.365
MAP after feedback: 0.383