Explanation and Evaluation

1. TF-IDF/ Vectorial Model

This search engine was based on the tf-idf vectorial model such that all documents are transformed in vectors according to their tf-idf representation. Therefore, when a query is submitted, we transform this query into tf-idf vector by counting the frequency of the words in the query and multiplying by their inverse document frequency compared to the corpus. This allows us to compute the scalar product between the query vector and the vector of all documents in the corpus. By computing this scalar product, we can retrieve an angle. We assume that the smaller this angle is, the more pertinent the document is to the query.

A dictionary was created in order to properly store all the vectorial representation of all documents on the collection. The keys of this dictionary are the documents' name and the words appear in this document. Each key refers to a value of the tf-idf product of that word in that document. Therefore, when we compute the scalar products of the whole collection, we loop over all documents and for each document we go through the words in the query and access the generated dictionary to compute the product between the tf-idf value of the word in that query and the tf-idf value of that word in the document, computing the scalar product between all documents in the collection in O(n*m) time. Where n is the number of documents in the collection and m is the number of words in the query.

However, after the scalar product has been computed and the angle between the query vectors and the document vectors is found, it is still necessary to sort them by angle value which is done is O(n*logn) time. This is clearly the upper bound of the search engine.

2. The programs

Contruct_TF_IDF_Matrix.py - This program constructs the TF-IDF dictionary by using the TF of each document and the IDF of each word. It stores the result on key which is a combination of the document name with the studied word. This program stores the result in form of pickle file.

Contruct_TF_Matrix.py - This is responsible for counting the word frequency of each word inside every document. If the word is not in the document (frequency zero) we simply do not store anything. Stores the result in a pickle file.

GetAbsoluteValueFromDocumentVector.py - This calculates the absolute value of the vector generated by the tf-idf formulation of every document. This is done to facilitate the calculations on the angle based on the scalar product. This program stores the result in form of pickle file.

GetCollection.py - This retrieves the names of all the documents in the collection and stores the result in a pickle file as a list. Allowing other programs to loop easier through the whole collection.

GetStats.py - This gets the stats of all documents and gets the idf calculation for every word in the vocabulary. This program stores the result in form of pickle file.

GetVocabulary.py - This returns the vocabulary in the collection allowing other program to easily loop through the words in the vocabulary. This program stores the result in form of pickle file.

SearchQueries.py - This is the program that searches the collection based on the formulations described above. Just type the query as a parameter in the beginning of the program.

3. Evaluation and selection of the threshold

Our search engine returns a set of documents and their scores. In order to guarantee the quality of our output, we tried to set up a threshold for selecting the most pertinent files. Firstly, we studied the "Rappel-precision" relation. From Fig. 1, we can see that the precision drops sharply with the increase of rappel. It's almost impossible to guarantee the precision and the rappel of a query at the same time.

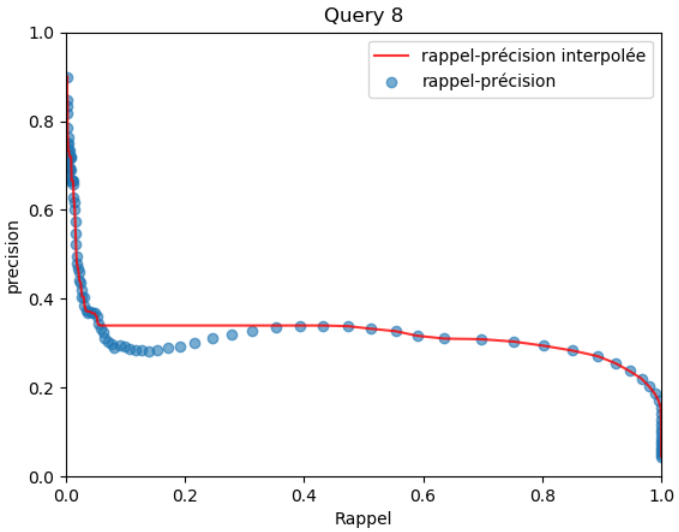


Fig. 1 Rappel-Precision curve and Rappel-interpolated precision curve of query 8

We then continued to analyze the relation between query score and the precision based on all 8 given develop queries and got Fig 2. We can find that the global precision drops when the score exceeds around 1.34. With this observation, we set a threshold of 1.344 for the search engine. Only the documents with lower than 1.344 will be considered. As a reminder, the score represents the angle between the query vector and the document vector. We keep the most pertinent documents for the output. However, our search engine is not sensitive to some queries (query 1 for example). In other words, the idea of setting a threshold to filter output files fails to some queries.

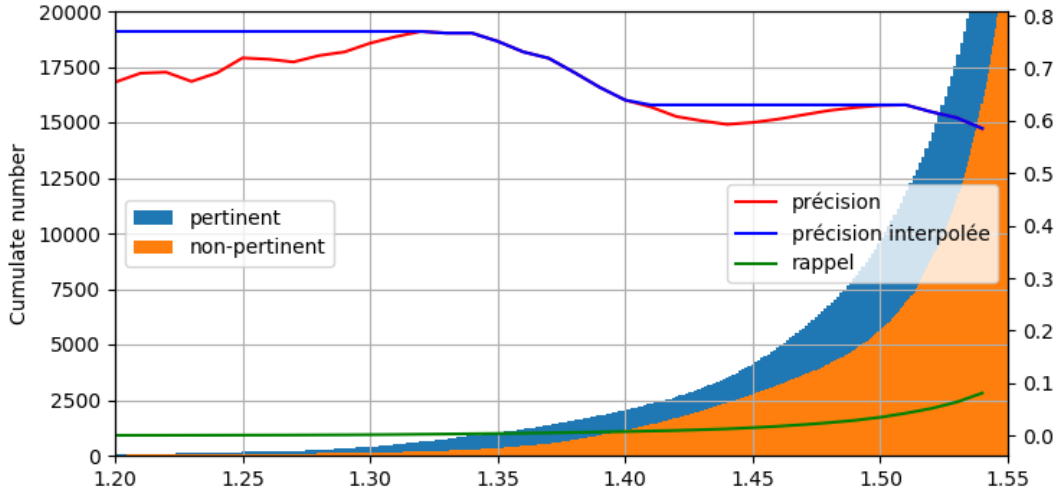


Fig. 2. Analysis of threshold value based on 8 develop queries. X-axis is the score of the query, the lower the score, the higher the pertinence.

We have also visualized how the number of input files influences the accuracy of the query. From Fig. 3 we have the conclusion that the corpus without stop words(Fig.3 left) has a better performance than that with stop words (Fig.3 right) For the purpose of generalization, we have finally chosen a more traditional method for our search engine, which is to limit directly the number of output to 15.

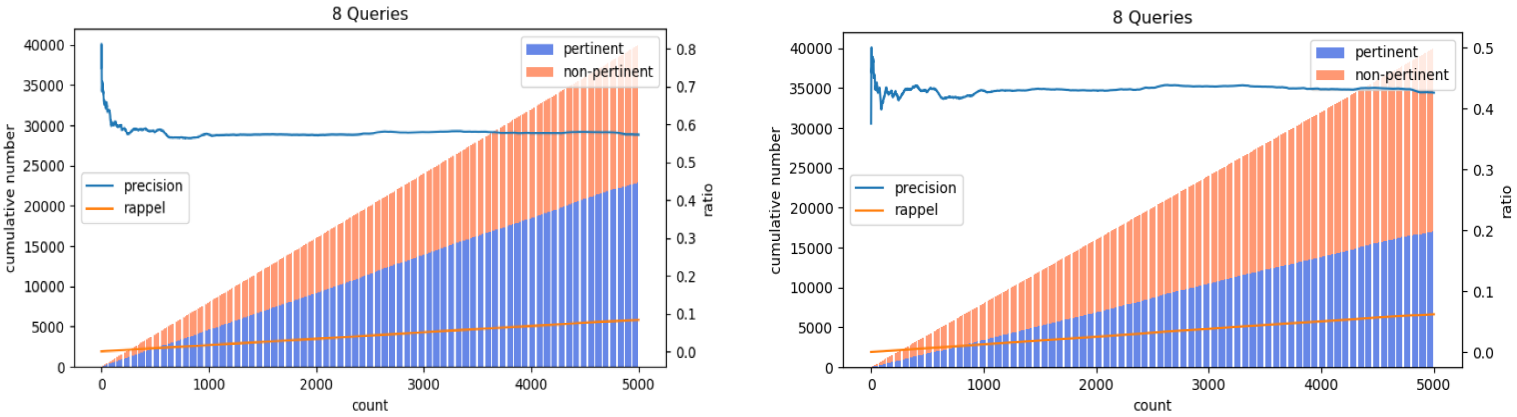


Fig. 3. Analysis of number of output files based on 8 develop queries. X-axis is the number of output files of each query. Without stop words(left), with stop words(right)

After fixing all parameters, we randomly chose some words from the corpus to build up queries. The first output given by our search engine was always the correct document from which we collected the words, suggesting that our search engine performs very well for long query (enough information).

Appendix:

Fig. 1

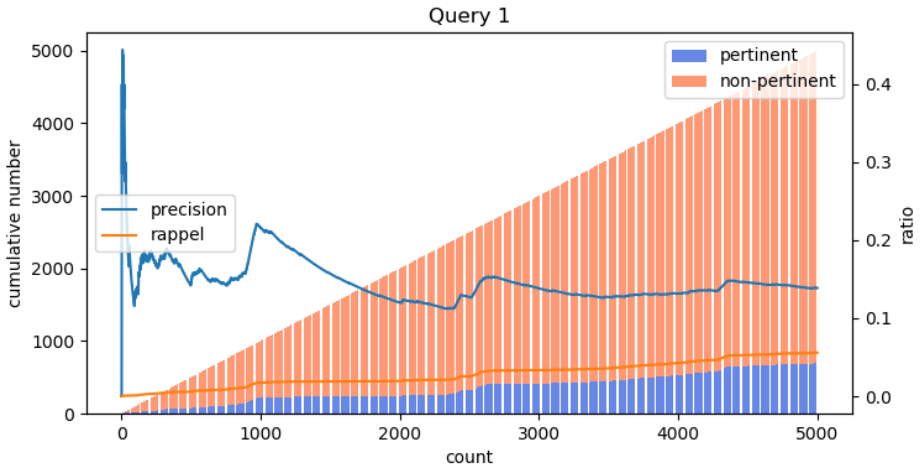


Fig. 2

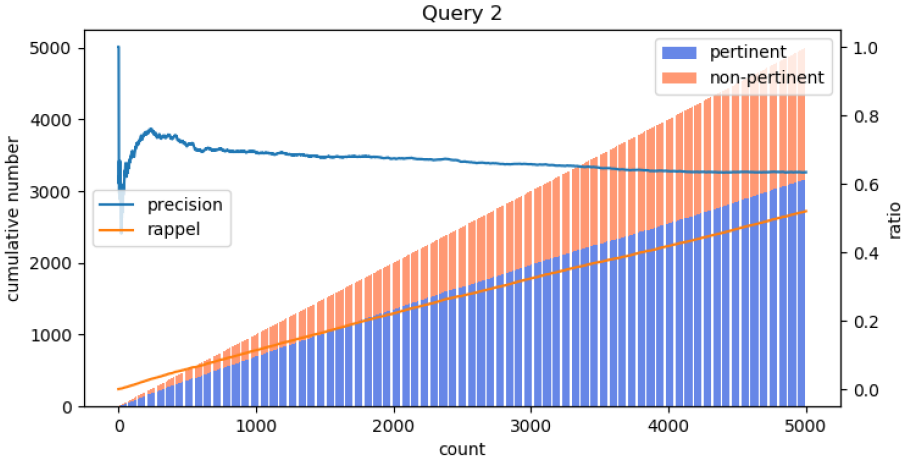


Fig. 3

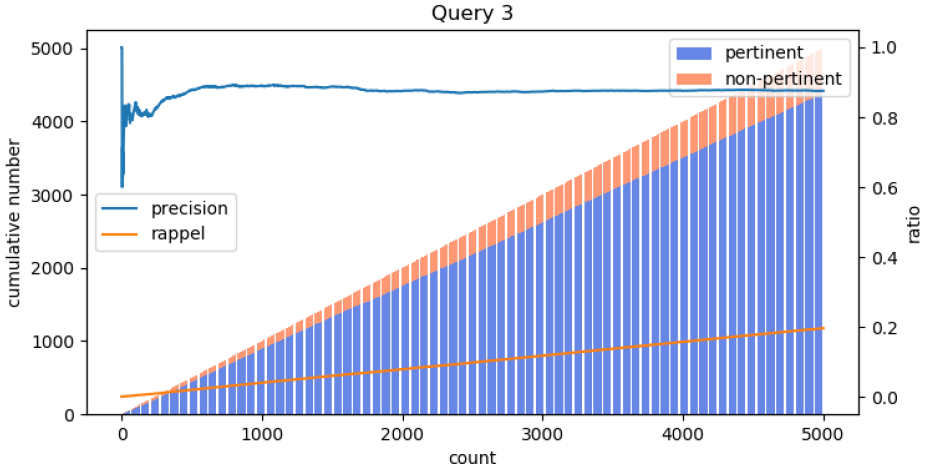


Fig. 4

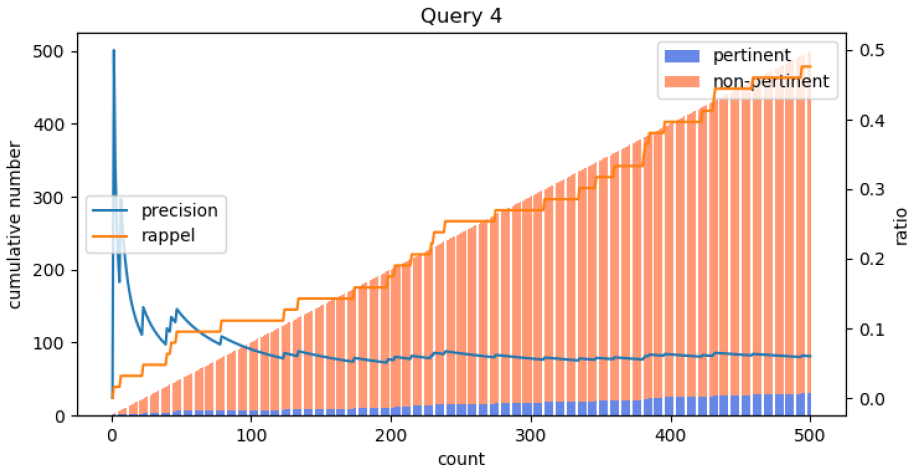


Fig. 5

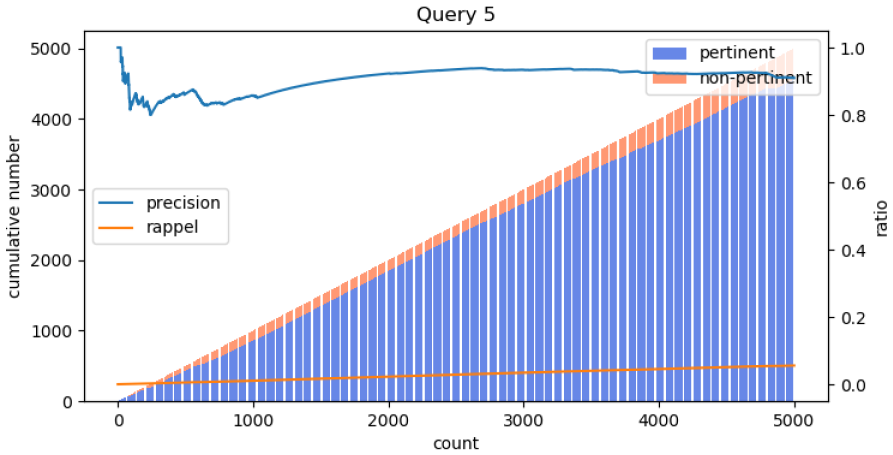


Fig. 6

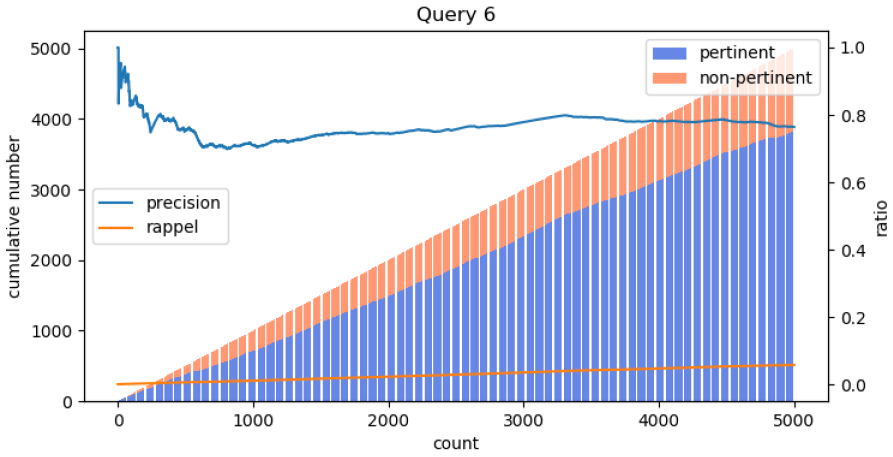


Fig. 7

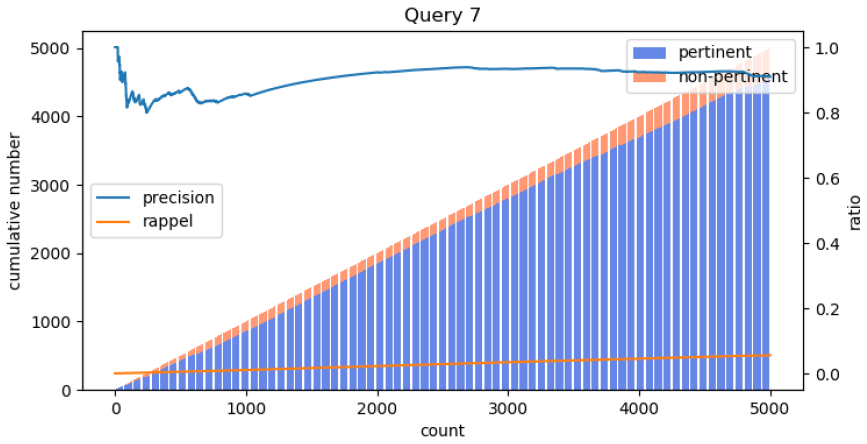


Fig. 8

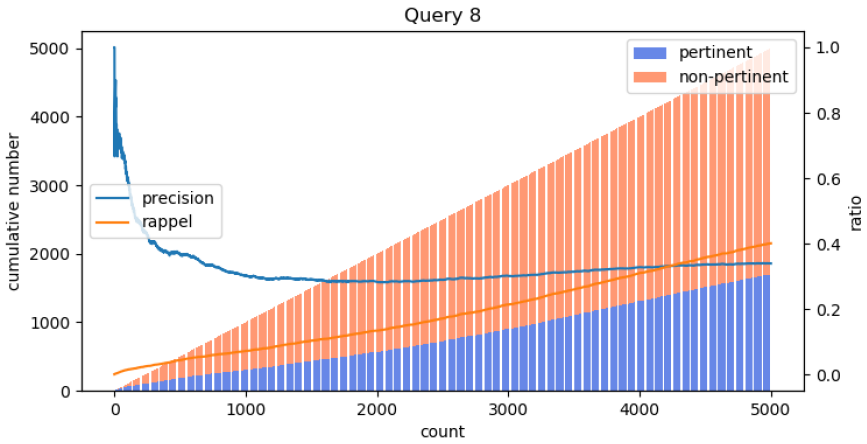
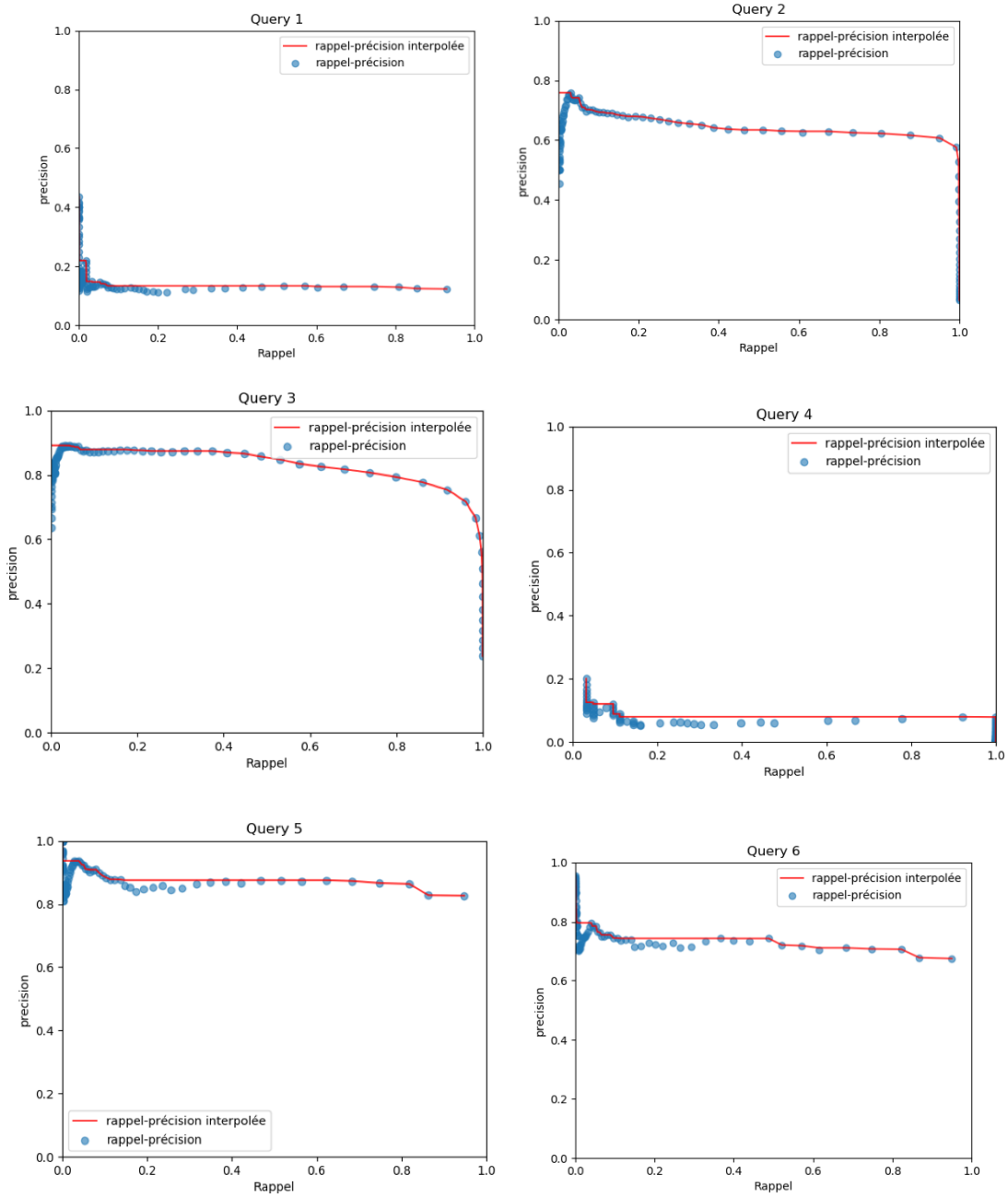
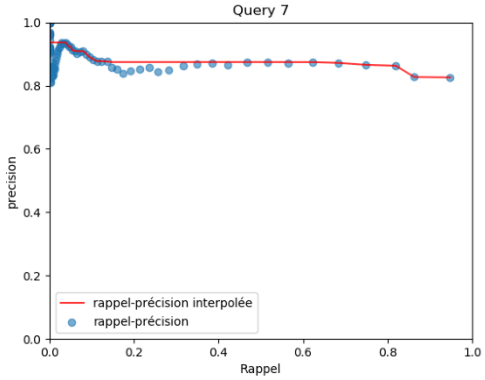
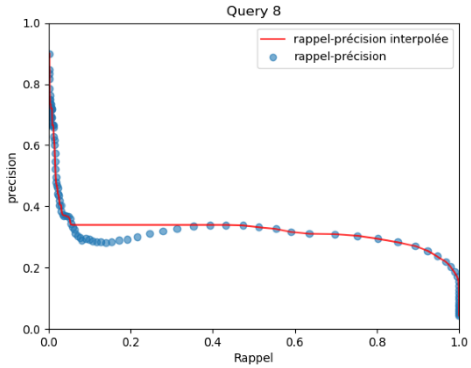


Fig. 9


Query 1
Query 2
Query 3
Query 4
Query 5
Query 6
rappel-précision interpolée
rappel-précision
precision
Rappel


Query 7
precision
Rappel
rappel-précision interpolée
rappel-précision
0.0
0.2
0.4
0.6
0.8
1.0


Query 8
precision
Rappel
rappel-précision interpolée
rappel-précision
0.0
0.2
0.4
0.6
0.8
1.0