Search Engine Python Project

By Luc Bos & Samed Balcioglu (Group 10)

Introduction:

We're creating a search engine that can search a query (one query at a time) in a map of documents. We're keeping it relatively simple and easy to use. We created a simple search tool in which a user can put the desired query and when the "search" button gets hit, it retrieves and ranks the 5 most relevant items based on the query that was put in. We used a set of documents, which we will discuss in the section "Dataset description". The query must loop through the documents and when it finds a match, it ranks them based on tfidf. If a query does not match a single document it will not give a ranked output of the results, but instead will show an interface that shows that the query the user has put in did not find a match. To make it user friendly we made this message as follows: 1 of meerdere zoektermen niet gevonden in documentencollectie zoek opnieuw. (this is in dutch)

Dataset description:

As the dataset to test our search engine we decided to use descriptions of the bachelor information science and computer science. These descriptions were gathered from the websites of Dutch universities and 2 other websites with general information about bachelors. In total we gathered 10 documents, all of them are in Dutch. The length of the documents range from around 200 to more than 400, with an average of 260 words. Other than interpunction and stop words, the documents did not contain any information that was not usable for the search engine. A problem that we encountered was caused by interpunction: when we wanted to preview a result by showing words around a queried word, an error appeared when searching for certain words. This was caused by the fact that there was a punctuation mark after the searched word, and therefore the preview function couldn't recognize the queried word in the document. For example, when searching for 'informatiekunde', the preview couldn't be shown for a file because the text looked as following: 'Informatiekunde: automatisch informatie verwerken Informatie wordt wel eens met olie vergeleken: met ruwe olie kan je niks, met ruwe informatie evenmin.'. We solved this by removing punctuation marks for the entire preview.

Implementation:

Preprocessing:

1. We created a list containing the texts of all the documents in our dataset. We end up with a list of lists. We remove the stop words from the document texts, using a file containing stop words.
2. We make a function to create a dictionary with word counts for each document, then we turn this dictionary into a dataframe. For example part of the word count dictionary: {'informatica leiden.txt': {'informatica': 5, 'bsc': 1, etc.
3. Using the data from the dataframe we first calculate the inverse documents frequencies for every term (this is calculated by (log2(y/g)) in this function y is the amount of documents, g is the amount of documents in which a term is present), then we multiply the idf’s with term weights which gives us term weights (tfidf). The term weights are added to a dataframe.
4. A function is defined to calculate vector lengths (this is calculated by adding the squaring all term frequencies, adding them together and taking the square root of the total)
 we use this function to create vector lengths for each document, and for the query.

Search engine:

1. We take the rows from our term weight dataframe that contain the term weights of words in the query, for each document these term weights are added to each other to get the document weight. Because the query frequency is either 0 or 1, there is no need for multiplication in this step
2. We calculate the cosine similarity between each document and the query (this is done by document weight / document length * query length). For this step we use the document weights, and the vector lengths for both the query and the documents.
3. Then we use a function to rank the documents by cosine similarity from high to low

Graphical User Interface:

1. For display purposes, a list is made containing ranking number, cosine similarity, name and a text preview for each document.
2. Graphical User Interface we use 3 html files, 1 as a homepage, 1 as a result page and 1 page for no results.
3. All 3 html sites contain a search bar, when a query is entered the words are used in the search engine part of the code.

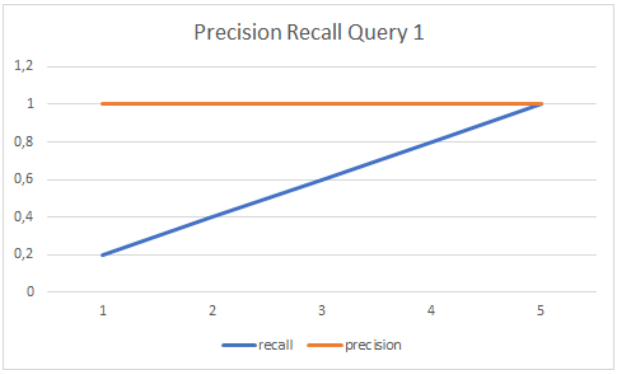
Evaluation:

We used alot of different queries to test our search engine. This helped us tweak some things inside the program that we wrote. When we started testing first the relevant documents were being ranked, but it showed the ranking of all 10 documents instead of the first 5. We fixed this. Later on we struggled with implementing the preview of alteast 1 keyword (of the searched query) and a set of words surrounding this keyword. We found a solution at first, but if a keyword in the query was followed by punctuation, the preview of the keyword + surrounding words did not show. We later fixed this. Also, one of the bigger problems we came across was that if one of the keywords of the query (when used 2 or more keywords) did not match to any documents, the whole query was showed as "does not match any documents". We fixed this later by using a try except block in our code.

To finish our evaluation of the search engine we chose 4 different queries, 1 that performs well, 2 that performed OK-ish, 1 that returned (almost) no relevant documents. Query 1 performed well, for this query we used 3 words. Query 2 performed Ok-ish, for this query we used 2 words. Query 3 returned (almost) no relevant documents, only 2. For this query we used 2 words. Query 4 performed OK-ish, for this query we used 2 words. We have discussed the documents and words that we used in the section "Dataset description"

Here are the results of the 4 queries that we used:

Query 1: informatica software ICT



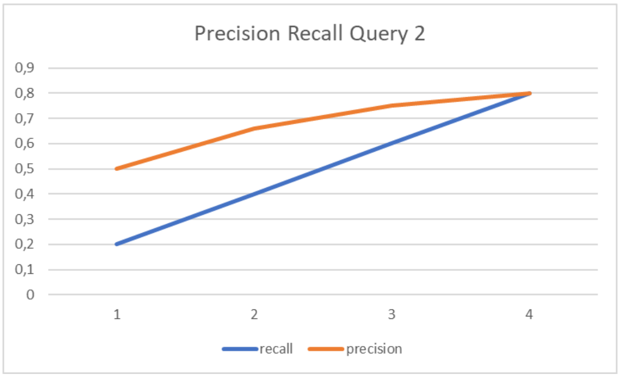
Precision @ 1: 1

Precision over entire output: 0,5

Recall over entire output: 0,3

Average precision: 1

Query 2: informatiekunde software



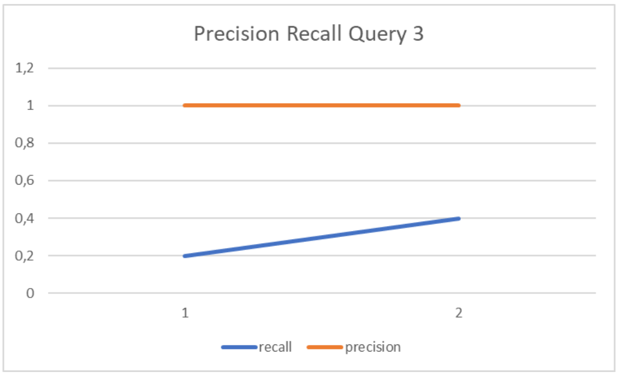
Precision @ 1: 0

Precision over entire output: 0,27

Recall over entire output: 0,2

Average precision: 0.68

Query 3: information science



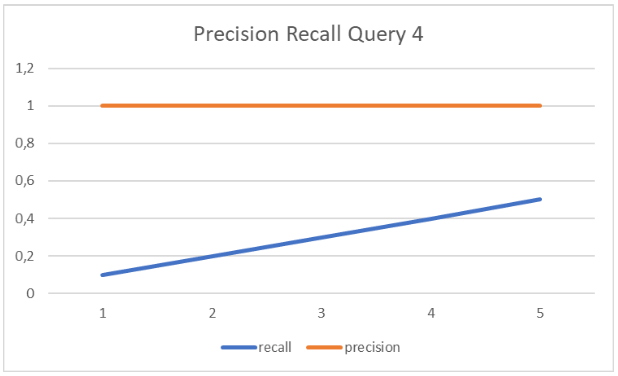
Precision @ 1: 1

Precision over entire output: 0,2

Recall over entire output: 0,06

Average precision: 1

Query 4: software engineer



Precision @ 1: 1

Precision over entire output: 0,5

Recall over entire output: 0,15

Average precision: 1

In the results we can see that query 1 works very well. The query is: informatica software ict. The top 5 ranked results are all in the documents that are about the study informatica. This is good because it shows that even though the other query words "software" and "ict" appear in more than only these 5 documents, it ranks these 5 as the highest because the query word "informatica" does not match to the other 5 documents. This outcome is as expected.

In query 2 we can see that it performed OK-ish. The query is: informatiekunde software. This does not perform as well as query 1 because the word "informatie" is being used (in informatiekunde). This means that even though it should only return 5 documents of with the subject "informatiekunde", in the top 5 ranked results it returns a document that has the word "informatie" in it, even though this document is about "informatica" and not "informatiekunde". Therefore, this query did not perform that well.

In query 3 we can see that this query only returned 2 relevant documents. The query is: information science. This query should only return every document about "informatiekunde", but instead it returns only 2 documents that are relevant. This means that this query did not do well at all.

In query 4 we can see that this query performed OK-ish. The query is: software engineer. The word software matches all of the documents, but the word engineer should have this query only return documents of "informatica" in the top 5 results. You can only become a software engineer with the study "informatica" and not with "informatiekunde". Therefore, this query did not perform as well as query 1, but did not perform as poorly as query 3.

Conclusion:

All in all, the project was a succes. Our search engine works as it should. We both put in the hours together to get to our final result. We did a lot of testing and debugging of our code to make sure that we fixed any possible problems. A few things that we could do better the next time are:

- All documents are opened and added to a list manually, it would be better to write a loop that opens every file in a given directory and adds their text to a list.
- No result is given when the query contains a word that isn't apparent in the whole dataset, also when the query does contain another word that is apparent in the datasent.. It would be better to have the code run normally and not consider the non-apparent word.
- Our dataset was relatively small and all about the same subject so it was hard to evaluate our search engines results. This is because it was hard to come up with a query that would be relevant for a small amount of documents.