In this session:

- We will program a simple User Relevance Feedback cycle on top of ElasticSearch.
- We will evaluate this strategy over a collection of documents.

1 Document relevance

To show more precisely how ElasticSearch works you have a script named `SearchIndexWeights.py` that it is similar to the script from the first session, but now only shows a specific number of hits (it shows the total number of documents that matches the query as the last line of the output) and also has a relevance score computed by ElasticSearch representing how well the document matches the query. The scripts has a `--nhits` flag that changes how many hits are shown and a `--query` flag that accepts a list of words (performs an AND query with all the words), this flag has to be the last one when invoking the script.

Play a little bit with this script, performing different queries and observing the scores. The syntax of the query allows using the fuzzy operator `~n`, but also the boost operator `^n` with n indicating how important is this term compared with the others, this changes the relevance score. For example with the `20_newsgroups` corpus try the following queries:

```
python SearchIndexWeights.py --index news --nhits 5 --query toronto nyc
python SearchIndexWeights.py --index news --nhits 5 --query toronto^2 nyc
python SearchIndexWeights.py --index news --nhits 5 --query toronto nyc^2
```

You will see that the scores and the positions of the documents change. Invent new queries and observe the results.

2 We will, we will Rocchio you

The goal of this session is to program a script `Rocchio.py` that implements a User Relevance Feedback system using Rocchio's rule. In fact, we will implement Pseudo-relevance Feedback since we will not ask the user which documents s/he finds relevant: simply we will assume that the first k documents are the relevant ones, for a k of our choice.

More precisely, we want our script to do the following:

1. Ask for a set of words to use as query
2. For a number of times (*nrounds*):
   (a) Obtain the k more relevant documents
   (b) Compute a new query applying Rocchio's rule to the current query and the Tf-Idf representation of the k documents
3. Return the k most relevant documents after the n iterations

You will have to implement the function that computes the new query applying the Rocchio's rule. Applying the Rocchio's rule to a query and a list of documents involves computing:

$$Query' = \alpha \times Query + \beta \times \frac{d_1 + d_2 + \cdots + d_k}{k}$$

You will have to compute the tf-idf vector for each document. To average all the documents you will have to sum vectors with many elements, you must use dictionaries to do it more efficiently instead of merging ordered vectors. What is the difference in computational cost of merging ordered vectors or using dictionaries for this operation? Discuss it in your report.

Also the resulting list of terms will be large. Consider pruning the list to only the R more relevant terms (larger weights).

Most of the elements to solve this you already have from the provided scripts and from your solutions from the previous sessions:

1. From `SearchIndexWeights.py` you have the code for building a query for a list of words and then retrieving the k more relevant documents.
2. Adding the weights computed using Rocchio's rule to each word in the search needs only concatenating the word, the boost operator (^) and the weight.
3. You will have to compute the tf-idf vector for a document, problem that you have solved in the previous session.

Observe that there are several parameters you can play with, at least:

- *nrounds*, the number of applications of Rocchio's rule
- k, the number of top documents considered relevant and used for applying Rocchio at each round
- R, the maximum number of new terms to be kept in the new query
- α and β, the weights in the Rocchio rule.

Please make easy to change them in the code (e.g., their values defined only once!).

3 Experimenting

Once you are done with your programming, try it out with the test collections from the previous sessions. Do the queries that pseudorelevance feedback produce make sense? For example, do the new terms seem related to what the user is looking for?

In which sense do the results improve? Recall? Precision?

Investigate to some extent the effect of each parameter. Do you get very different results if you change the parameters *nrounds*, k, R, α, β? Do you find, for each one, some value or value range that seems to be optimal in some sense?

4 Deliverables

To deliver: Write a short report (3-4 pages max) describing:

1. if you more or less followed the scheme suggested above to implement URF, or if you changed it in some significant way
2. any major difficulties you found
3. a few of the experiments you performed, and the results you observed
4. any thoughts or conclusions that depart from what we asked you to do – in fact, they'll be highly valued if they are intelligent and show that you can go beyond following instructions literally

Please do not explain (again) what Rocchio rule is or stuff that is explained in this document. We know it already!

PDF format is preferred. Make sure it has your names, date, and title.

Also, submit the code for the script that you created. Add everything to a `.zip` file and deliver it through the raco.

Rules: 1. You can solve the problem alone or with one other person. 2. No plagiarism; don't discuss your work with others, except your teammate if you are solving the problem in two; if in doubt about what is allowed, ask us. 3. If you feel you are spending much more time than the rest of the group, ask us for help. Questions can be asked either in person or by email, and you'll never be penalized by asking questions, no matter how stupid they look in retrospect.

To deliver: You must deliver a brief report describing your results and the main difficulties/choices you had while implementing this lab session's work. You also have to hand in the source code of your implementations.

Procedure: Submit your work through the raco platform as a single zipped file.

Deadline: Work must be delivered within **2 weeks** from the lab session you attend. Late submissions risk being penalized or not accepted at all. If you anticipate problems with the deadline, tell me as soon as possible.