Vector Space Model with PyTerrier

David Burian

January 2, 2023

1 Introduction and choice of the framework

My IR system is implemented with PyTerrier[1], which is a python interface to a java IR system called Terrier[2].

Apart from PyTerrier, I've considered also other IR frameworks. Here I briefly discuss why I've decided to use PyTerrier. I've looked at the following systems sorted according to my preference in decreasing order:

- Xapian[3] – C++ IR system with bindings to python with good documentation. The only drawback I was afraid of was that I would have to write much more code than with PyTerrier.
- Lucene[4] – Popular and largely used Java IR end-to-end system. I trust this would be the number one choice for developing large IR system. The only two drawbacks from my point of view were the language used and having to deal with complexities around the deployment of such system, which was not really the focus of this assignment.
- Anserini[5] – Java IR built with Lucene with a python interface called Pyserini. Both look like well built projects, though lacking documentation.
- Solr[6], ElasticSearch[7] – These are complete IR deployment-ready systems that are built on top of Lucene. Both offer high-level features and require minimal configuration. Because of this these systems are in my opinion the wrong choice for this assignment, where we require high configurability and do not concern ourselves with deployment and user interface.

To sum up I chose PyTerrier because of:

- good documentation,
- trusted and large enough project, which did the heavy lifting (Terrier),
- good balance of the amount of configurability and amount of code needed to get fully functional system,
- an existence of python interface.

2 Experiments

I have carried out 34 experiments which I will describe in the following paragraphs. I've split the experiments into sections, where each section is focused on a particular aspect of an IR system. At the end of each section I will include the experiments' results on the training topics.

[1] `https://github.com/terrier-org/pyterrier`
[2] `https://github.com/terrier-org/terrier-core`
[3] `https://xapian.org`
[4] `https://lucene.apache.org`
[5] `https://github.com/castorini/anserini`
[6] `https://solr.apache.org`
[7] `https://www.elastic.co/elasticsearch/`

2.1 Baselines

run-0 Baseline implemented according to the assignment instructions.

- *tokenizer*: regex tokenizer spliting at any of the following characters `-,.;;?!␣\t\n[]()'"`
- *stop words*: -
- *stemmer*: -
- *weight model*: term frequency

run-0-tfidf Improved baseline. I wanted a more robust weighting model before tinkering with indexation. Do note that the TF-IDF used is not pure TF-IDF. But a version given in Jones [1972] with Robertson's TF where the TF-IDF of document d and query q is defined as:

$$TF - IDF_{d,q} = \frac{tf * k_1 * log(\frac{N}{df})}{tf + k_1 * (1 - b + b * \frac{|d|}{avg_{d'}|d'|})},$$

where $k_1 = 1.2$ and $b = 0.75$ are constants.

- *tokenizer*: regex tokenizer spliting at any of the following characters `-,.;;?!␣\t\n[]()'"`
- *stop words*: -
- *stemmer*: -
- *weight model*: Robertson's TF-IDF

	Czech		English	
run ID	MAP	P@10	MAP	P@10
run-0	0.0433	0.0680	0.1011	0.1360
run-0-tfidf	0.1905	0.1800	0.2359	0.2720

Table 1: Results of **run-0** and **run-0-tfidf**.

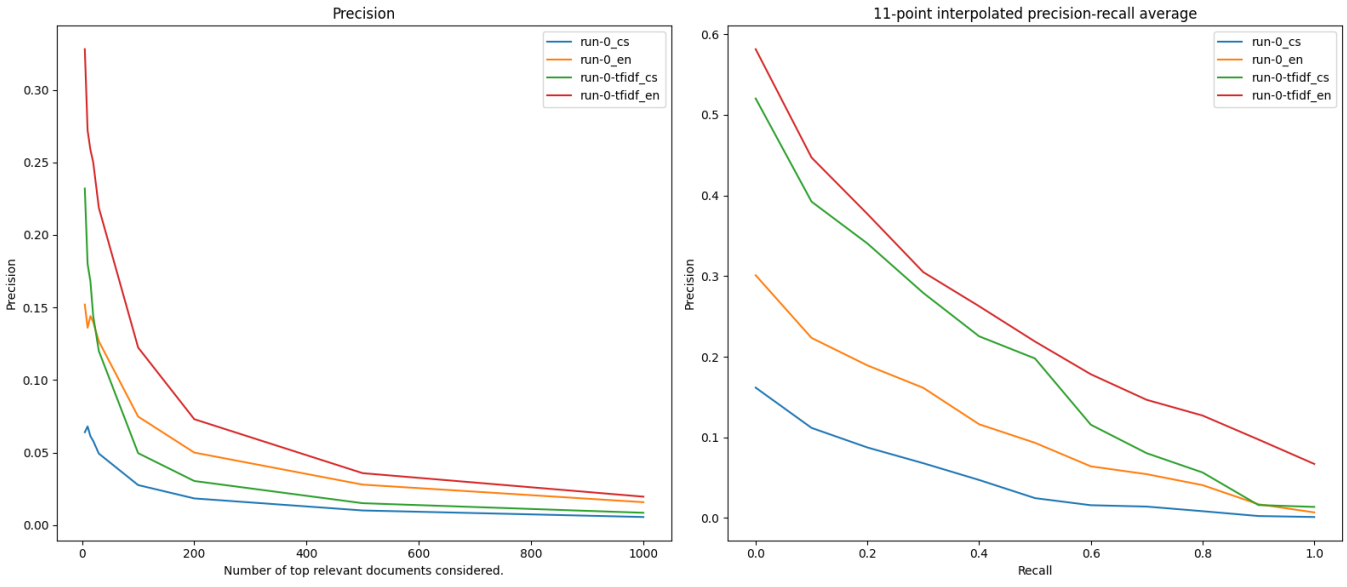

Figure 1: Precision and 11-point interpolated precision-recall curves for **run-0** and **run-0-tfidf**.

As can be seen from the results TF-IDF improved results considerably by more than doubling both MAP and P@10.

2.2 Tokenization

run-0-pyterrier-tok Improved tokenization using PyTerrier's builting tokenizers. 'english' is specialized tokenizer for English, while 'utf' is general tokenizer applicable not only to Czech.

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: -
- *stemmer*: -
- *weight model*: Robertson's TF-IDF

run-0-nltk-tok Improved tokenization using nltk's tokenizers specialized per each language.

- *tokenizer*: nltk's tokenizers[8]; 'english' for English dataset, 'czech' for Czech dataset
- *stop words*: -
- *stemmer*: -
- *weight model*: Robertson's TF-IDF

	Czech		English	
run ID	MAP	P@10	MAP	P@10
run-0-pyterrier-tok	0.2540	0.2560	0.3472	0.3960
run-0-nltk-tok	0.1893	0.1800	0.2110	0.2560

Table 2: Results of **run-0-pyterrier-tok** and **run-0-nltk-tok**.

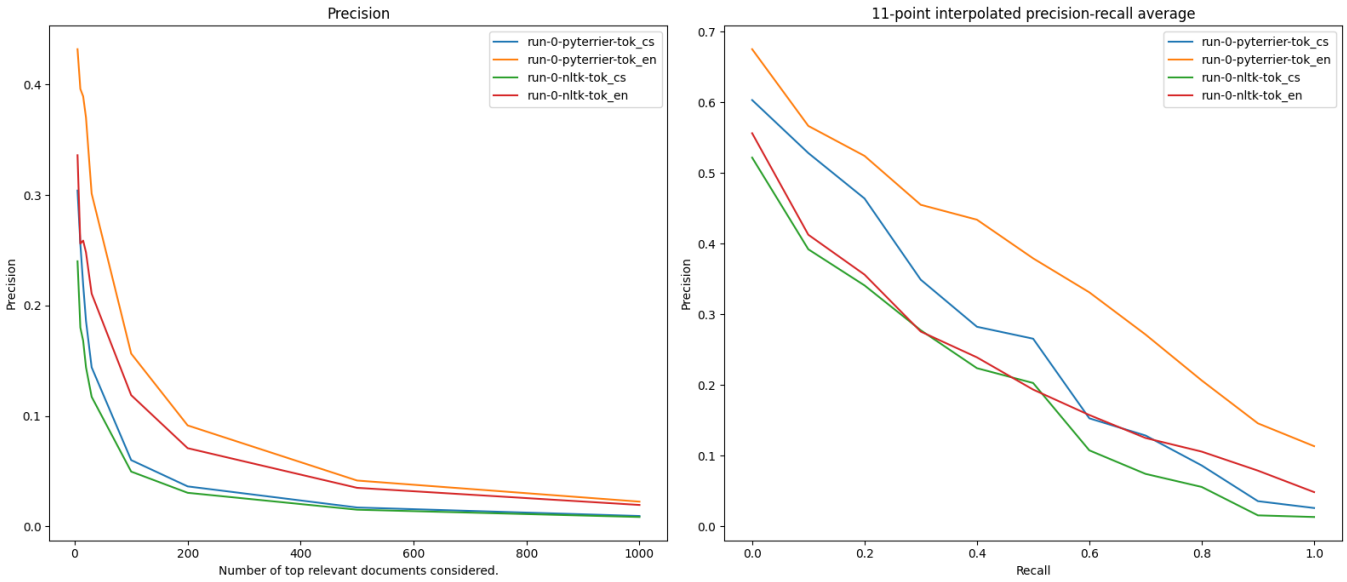

Figure 2: Precision and 11-point interpolated precision-recall curves for **run-0-pyterrier-tok** and **run-0-nltk-tok**.

As can be seen from the results using specialized tokenization algorithm instead of splitting with regex has number of advantages. Surprisingly nltk's tokenization does not perform as well as PyTerrier's. And so, going forward I will use PyTerrier's 'english' tokenization for English dataset and 'utf' for Czech dataset.

[8] `https://www.nltk.org/api/nltk.tokenize.html`

2.3 Stopwords

run-0-pyterrier-stop Incorporating PyTerrier's English stopword list. Since there is no Czech alternative, I run this experiment only for the English dataset.

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: builtin PyTerrier's stopwords list for English dataset only
- *stemmer*: -
- *weight model*: Robertson's TF-IDF

run-0-nltk-stop Using nltk's English stopword list. Same as with PyTerrier's list, there is no Czech alternative and so I again ran this experiment only for the English data.

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: nltk English stopword list
- *stemmer*: -
- *weight model*: Robertson's TF-IDF

run-0-kaggle-stop Using random stopword list found online. The main goal of this experiment was to find some stopwords for the Czech dataset and so it was ran only against the Czech documents.

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list[9]
- *stemmer*: -
- *weight model*: Robertson's TF-IDF

	Czech		English	
run ID	MAP	P@10	MAP	P@10
run-0-pyterrier-stop	-	-	0.3443	0.4080
run-0-nltk-stop	-	-	0.3442	0.4080
run-0-kaggle-stop	0.2576	0.2640	-	-

Table 3: Results of **run-0-pyterrier-stop**, **run-0-nltk-stop** and **run-0-kaggle-stop**.

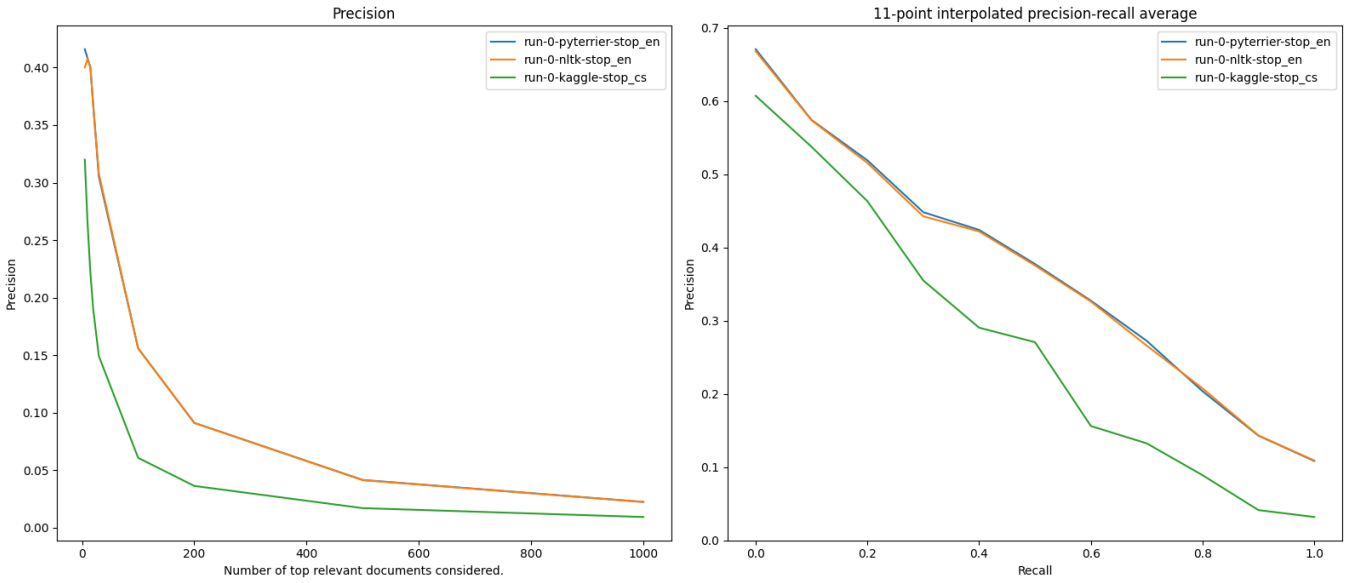

Figure 3: Precision and 11-point interpolated precision-recall curves for **run-0-pyterrier-stop**, **run-0-nltk-stop** and **run-0-kaggle-stop**.

Clearly stopword lists do not make much of a difference. In the case of the English dataset they slightly worsen the results, while for the Czech dataset the effect is opposite. Going forward I will use the Kaggle Czech stopword list for the Czech data and no stopwords for the English data.

[9] `https://www.kaggle.com/datasets/heeraldedhia/stop-words-in-28-languages?select=czech.txt`

2.4 Stemming\Lemmatization

run-0-porter-stemm Stemming documents for both languages with Porter stemmer.

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list for Czech data only
- *stemmer*: PyTerrier's Porter stemmer
- *weight model*: Robertson's TF-IDF

run-0-snowball-stemm Stemming documents for both languages with Snowball stemmer.

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list for Czech data only
- *stemmer*: PyTerrier's Snowball stemmer
- *weight model*: Robertson's TF-IDF

run-0-udpipe-lemm I tried to use lemmatization as well using UDPipe 2. While I found the installation and running of the software quite easy, I lacked the hardware resources to run the experiments efficiently. I was able to run lemmatization two times (both took more than 8h) for the Czech dataset only.

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list for Czech data only
- *stemmer*: Lemmatization using UDPipe 2[10] with the 'czech' model
- *weight model*: Robertson's TF-IDF

run-0-czech-stemm To improve my results for the Czech dataset I searched for stemmer online. The only stemmer I found was one implemented by Prof. Jacques Savoy. Even though the stemmer is designed for Czech I ran it also against the English dataset.

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list for Czech data only
- *stemmer*: Czech stemmer[11] implemented by Prof. Jacques Savoy, 'light' version
- *weight model*: Robertson's TF-IDF

	Czech		English	
run ID	MAP	P@10	MAP	P@10
run-0-porter-stemm	0.2573	0.2600	0.3929	0.4240
run-0-snowball-stemm	0.2576	0.2640	0.3960	0.4240
run-0-udpipe-lemm	0.0000	0.0000	-	-
run-0-czech-stemm	0.3117	0.3240	0.3430	0.3800

Table 4: Results of **run-0-porter-stemm**, **run-0-snowball-stem**, **run-0-udpipe-lemm** and **run-0-czech-stemm**.

The best stemmer for English seems to be the Snowball stemmer, though the difference is only marginal. The Czech stemmer by Jacques Savoy did improve the results considerably for the Czech dataset, while expectedly worsen the score for the English dataset. Unfortunately I was not able to find the bug in my **run-0-udpipe-lemm** run.

Going forward I will use the Czech stemmer for Czech and Snowball stemmer for English.

[10] `https://ufal.mff.cuni.cz/udpipe/2`
[11] `http://members.unine.ch/jacques.savoy/clef/index.html`

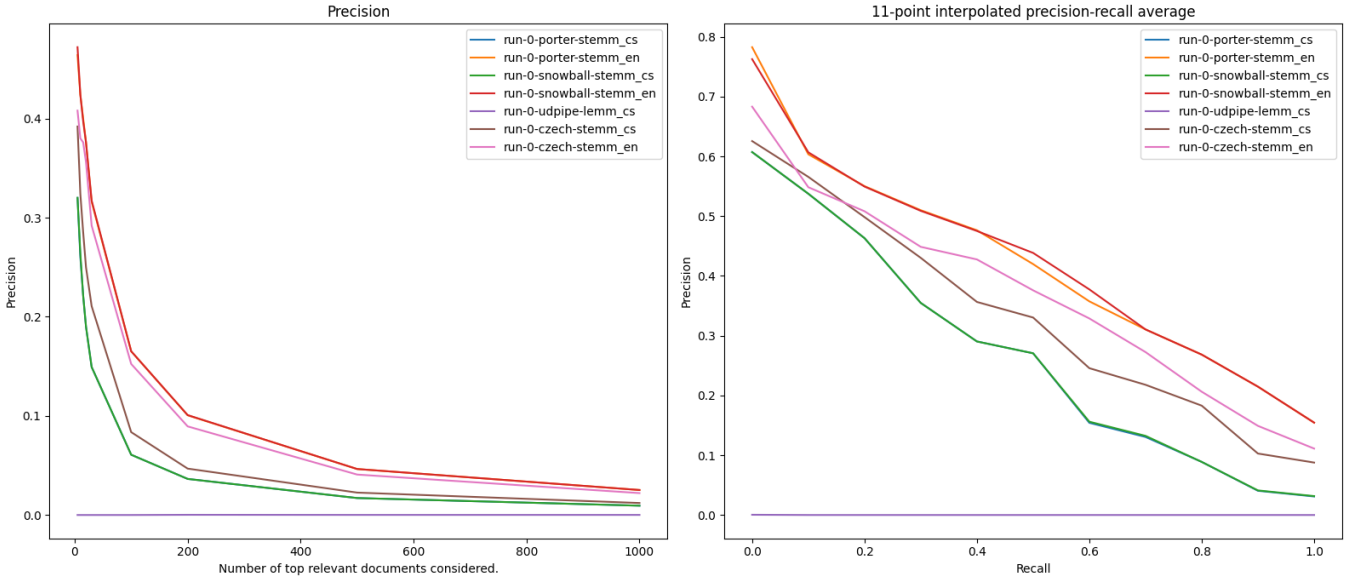

Figure 4: Precision and 11-point interpolated precision-recall curves for **run-0-porter-stemm**, **run-0-snowball-stem**, **run-0-udpipe-lemm** and **run-0-czech-stemm**.

2.5 Weighting models

run-0-tfidf-pivoted

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list for Czech data only
- *stemmer*: Czech stemmer implemented by Prof. Jacques Savoy, 'light' version for Czech; Snowball stemmer for English
- *weight model*: TF-IDF with Pivoted length normalization

run-0-tfidf-pivoted-robertson Instead of the usual length normalization, PyTerrier offers also a Robertson's normalization Robertson and Walker [1994].

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list for Czech data only
- *stemmer*: Czech stemmer implemented by Prof. Jacques Savoy, 'light' version for Czech; Snowball stemmer for English
- *weight model*: TF-IDF with Pivoted Robertson's normalization

run-0-bm25

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list for Czech data only
- *stemmer*: Czech stemmer implemented by Prof. Jacques Savoy, 'light' version for Czech; Snowball stemmer for English
- *weight model*: BM25

run-0-pl2

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list for Czech data only
- *stemmer*: Czech stemmer implemented by Prof. Jacques Savoy, 'light' version for Czech; Snowball stemmer for English
- *weight model*: PL2

run-0-lemur-tfidf

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list for Czech data only
- *stemmer*: Czech stemmer implemented by Prof. Jacques Savoy, 'light' version for Czech; Snowball stemmer for English
- *weight model*: Lemur[12]'s version of TF-IDF, implemented by Terrier

	Czech		English	
run ID	MAP	P@10	MAP	P@10
run-0-tfidf-pivoted-robertson	0.3121	0.3320	0.3925	0.4200
run-0-tfidf-pivoted	0.2799	0.2840	0.3738	0.4160
run-0-pl2	0.2901	0.3160	0.3821	0.3960
run-0-bm25	0.3082	0.3240	0.3806	0.3760
run-0-lemur-tfidf	0.2947	0.3000	0.3877	0.4280

Table 5: Results of **run-0-tfidf-pivoted**, **run-0-tfidf-pivoted-robertson**, **run-0-bm25**, **run-0-pl2** and **run-0-lemur-tfidf**.

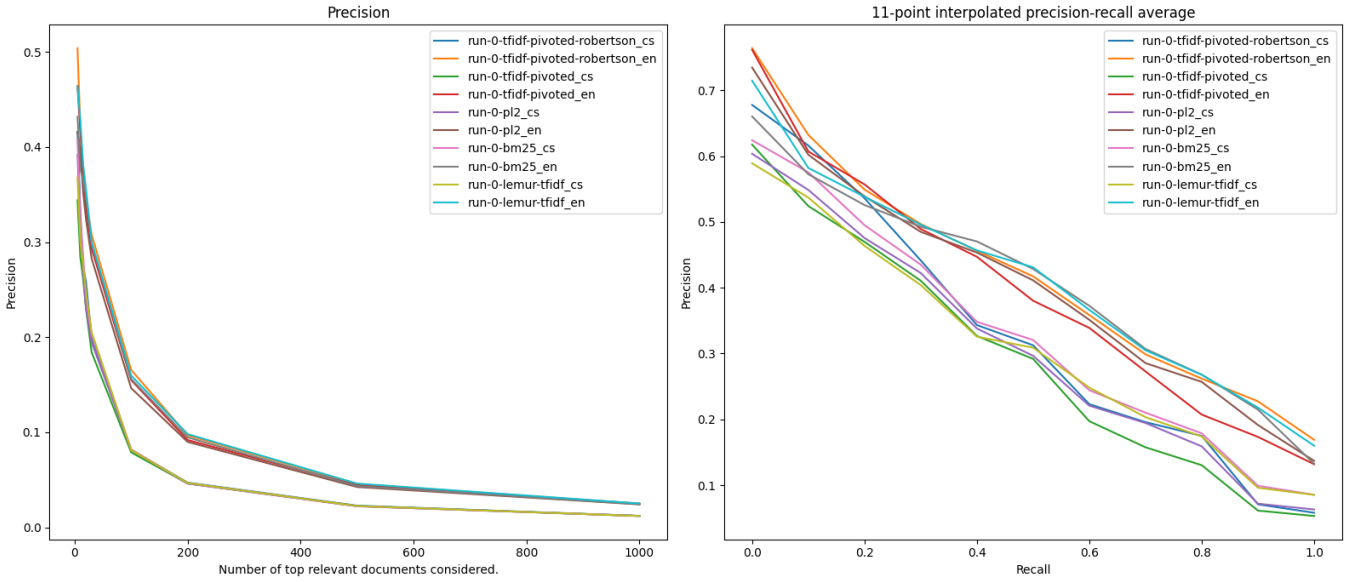

Figure 5: Precision and 11-point interpolated precision-recall curves for **run-0-tfidf-pivoted**, **run-0-tfidf-pivoted-robertson**, **run-0-bm25**, **run-0-pl2** and **run-0-lemur-tfidf**.

Unfortunately none of the tried weighting models were able to improve the benchmark set by the initial Robertson's TF-IDF.

Before we move to query expansion, let us define **run-1** composing the best configuration yet to be found.

run-1 Represents the best configuration I have came across in my experiments.

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list for Czech data only
- *stemmer*: Czech stemmer implemented by Prof. Jacques Savoy, 'light' version for Czech; Snowball stemmer for English
- *weight model*: Robertson's TF-IDF

[12] `http://www.lemurproject.org`

2.6 Query expansion

run-2 **run-1** with query expansion.

- *tokenizer*: builtin PyTerrier tokenizers; 'english' for English dataset, 'utf' for Czech dataset
- *stop words*: Kaggle Czech stopword list for Czech data only
- *stemmer*: Czech stemmer implemented by Prof. Jacques Savoy, 'light' version for Czech; Snowball stemmer for English
- *weight model*: Robertson's TF-IDF
- *query expansion*: divergence from Randomness query expansion model using PyTerrier's built in 'Bo1' model

	Czech		English	
run ID	MAP	P@10	MAP	P@10
run-2	0.3548	0.3520	0.3983	0.4360

Table 6: Results of **run-2**.

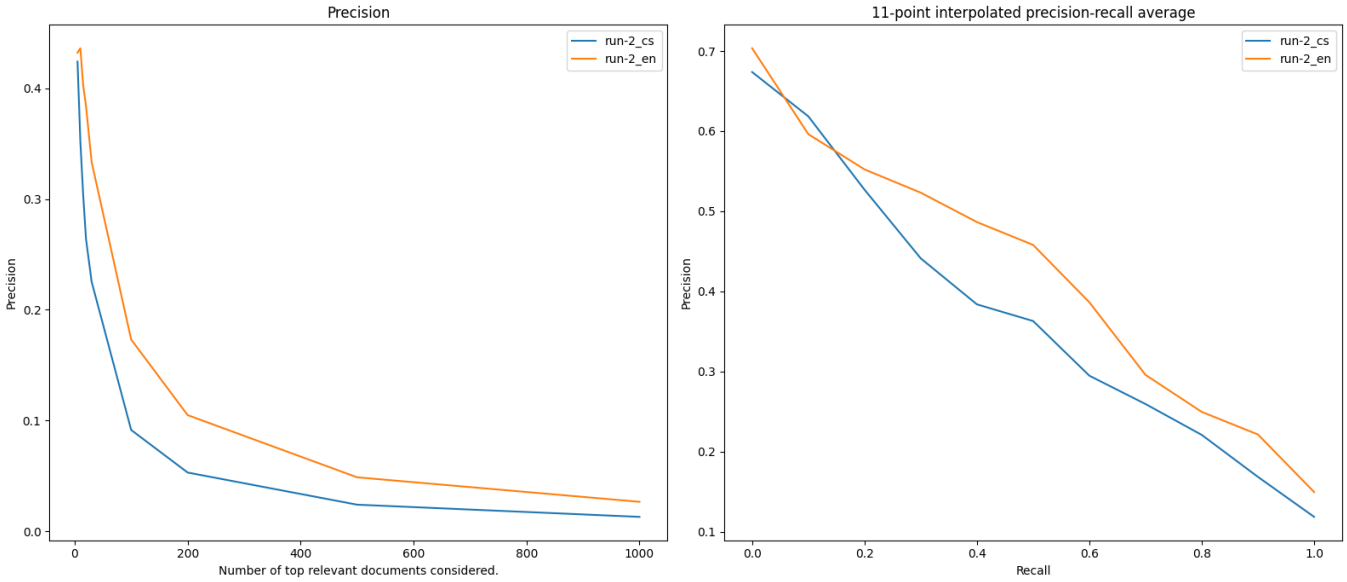

Figure 6: Precision and 11-point interpolated precision-recall curves for **run-2**.

As can be seen from the results query expansion helped tremendously to the Czech dataset, but improved results for the English dataset only marginally.

3 Conclusion

I have built an IR system using PyTerrier and Terrier. Throughout my experiments I managed to improve MAP scores by almost 0.41 for the Czech dataset and by almost 0.3 for the English dataset.

	Czech		English	
run ID	MAP	P@10	MAP	P@10
run-0	0.0433	0.0680	0.1011	0.1360
run-2	0.3548	0.3520	0.3983	0.4360

Table 7: Results of **run-0** and **run-2**.

References

Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972.

Stephen E Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94*, pages 232–241. Springer, 1994.

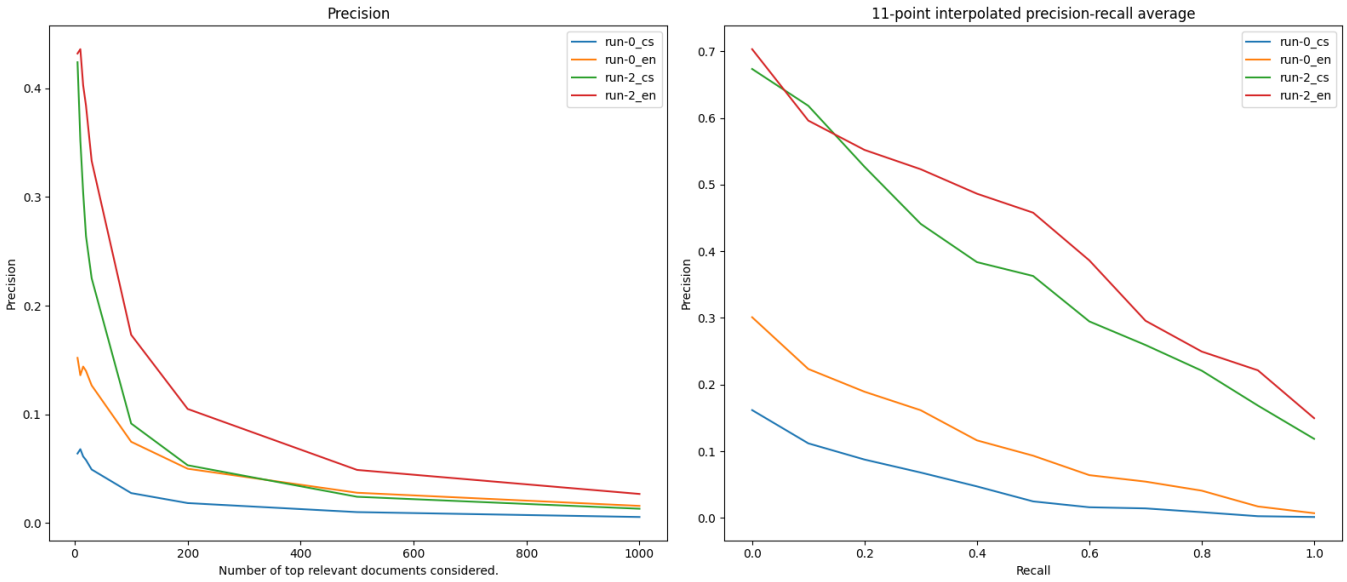

Figure 7: Precision and 11-point interpolated precision-recall curves for **run-0** and **run-2**.