Vector Space Model

David Burian

December 5, 2022

1 Installation

My solution is implemented in python `3.10.2`. All necessary packages are listed in `requirements.txt`.

The main program is in `main.py` and is run according to the assignment's instructions:

```
python main.py -q <topics> -d <docs> -r <run> -o <out-file>
```

For my own convenience I wrote a Makefile, which assumes:

- The whole assignment folder is placed under `DATA_DIR`
- Evaluation program `trec_eval` can be found under `TREC_EVAL_BIN`

Both constants can be set in the first few lines. Here is a summary of what the Makefile can do:

```
# Generates .res files for default run (run-0), both languages and both data splits
make res

# Generates .res files for run-1, both languages and both data splits
make res run=run-1

# Generates .res files for run-1, both languages and train data split
make res run=run-1 mode=train

# Generates .res file for run-1, cs language and train data split
make res run=run-1 mode=train lan=cs

# Generates corresponding .res file (if they are needed) and evaluates them using trec_eval
make eval run=run-1 lan=en
```

2 Experiments

2.1 run-0

The basic solution behaves according to the instructions and equally for both languages:

1. Parses all text found in a document (between tags, of course)
2. Splits the text using any of the following characters ␣`\t\n,.?!;:`
3. Uses inverted index to store term-document counts
4. From every query parses `<title>` and splits it to words identically
5. Computes similarity between query and document using term-at-a-time processing and min-heap
6. Uses cosine similarity as similarity measure

Results are depicted in Figure 1 and Table 1

	Czech				English			
run ID	MAP	#relevant	#rel. & ret.	MRR	MAP	#relevant	#rel. & ret.	MRR
run-0	0.1114	382	153	0.3571	0.1228	782	296	0.3217

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

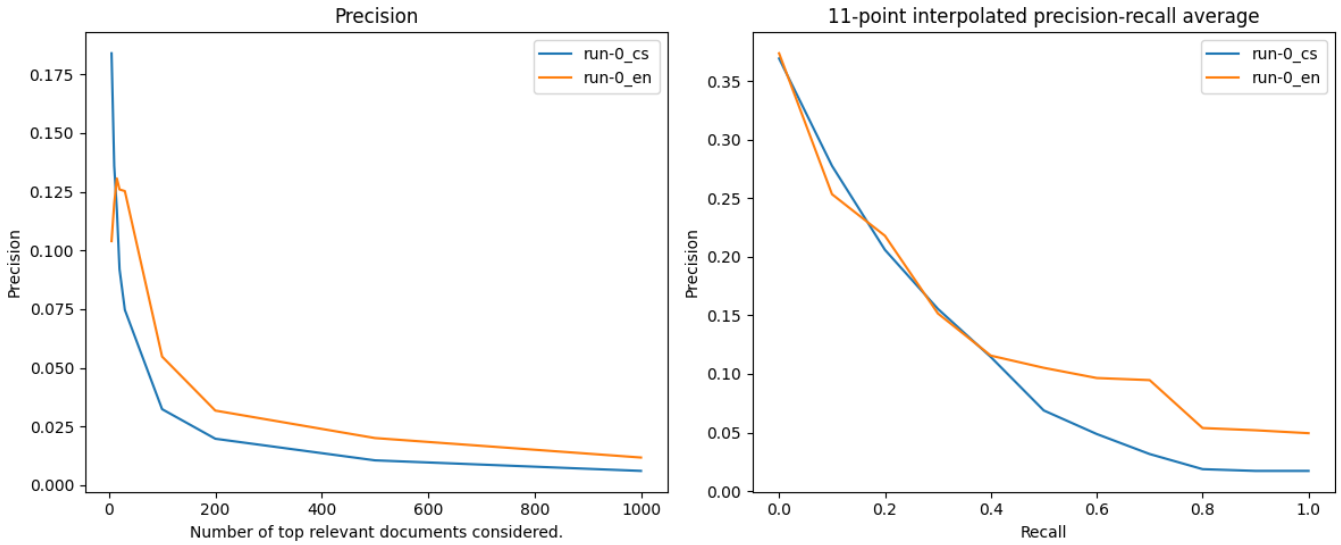

Figure 1: Precision and 11-point interpolated precision-recall curves for **run-0**.

As can be seen in Figure 1 **run-0** is for the Czech dataset initially much more precise, but its precision falls down more quickly than for the English one. For the English dataset the system is more forgiving, meaning there are more relevant documents for given queries. I believe this is the reason why for English **run-0** achieves higher MAP and has higher tail in the int. precision-recall curve.

2.2 run-1

After the basic solution I decided to find the best settings by iterative improvements. These will be described in the following paragraphs. Please note that not all of these experiments can be replicated with just specifying the run ID and running `make`. For some of these experiments it is necessary to change the source code or other files.

run-0-tfidf Instead of the default natural weighting I decided to use tf-idf. My reasons were that tf-idf weights terms according to their frequency of use and therefore is not as sensitive to everyday words such as *every*, *day* or *word*.

	Czech				English			
run ID	MAP	#relevant	#rel. & ret.	MRR	MAP	#relevant	#rel. & ret.	MRR
run-0	0.1114	382	153	0.3571	0.1228	782	296	0.3217
run-0-tfidf	0.1169	382	161	0.3989	0.1309	782	312	0.3856

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

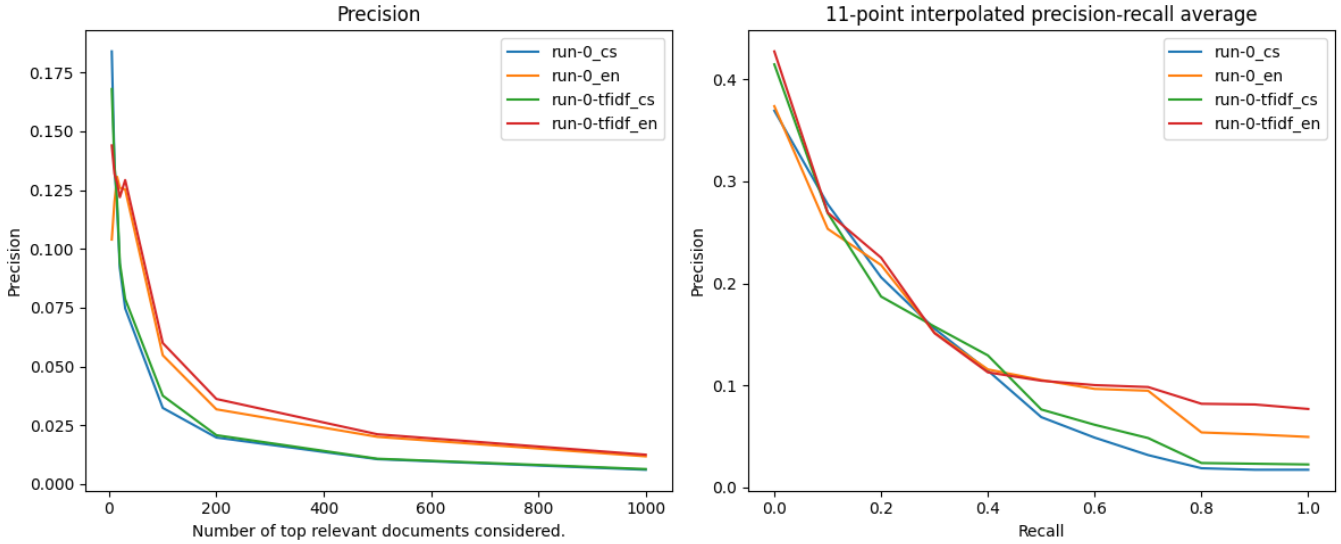

Figure 2: Precision and 11-point interpolated precision-recall curves for **run-0-tfidf** and **run-0**.

As can be seen from Figure 2 and from Table 2 tf-idf did not help much, but the system was able to return a little bit more relevant documents, than with natural weighting.

run-0-seps The default set of separators is rather small and I wanted to try to extend it before I started experimenting with stop words. I tried four extensions:

1. **run-0-seps-quot** – quotation marks: `"'`,
2. **run-0-seps-punc** – more punctuation characters: `-_`,
3. **run-0-seps-par** – parentheses: `[]()`,
4. **run-0-seps-quot-par** – quotation and parentheses sets combined.

To generate these results **run-0-tfidf** was used with different separators set.

	Czech				English			
run ID	MAP	#relevant	#rel. & ret.	MRR	MAP	#relevant	#rel. & ret.	MRR
run-0-tfidf	0.1169	382	161	0.3989	0.1309	782	312	0.3856
run-0-sep-par	0.1166	382	159	0.3971	0.1316	782	315	0.3856
run-0-sep-punc	0.1108	382	156	0.3890	0.1063	782	303	0.3222
run-0-sep-quot-par	0.1202	382	162	0.3965	0.1345	782	327	0.3583
run-0-sep-quot	0.1202	382	159	0.3983	0.1336	782	325	0.3583

Table 3: Results of **run-0-sep-punc**, **run-0-sep-quot**, **run-0-seps-par**, **run-0-seps-quot-par** and **run-0-tfidf**.

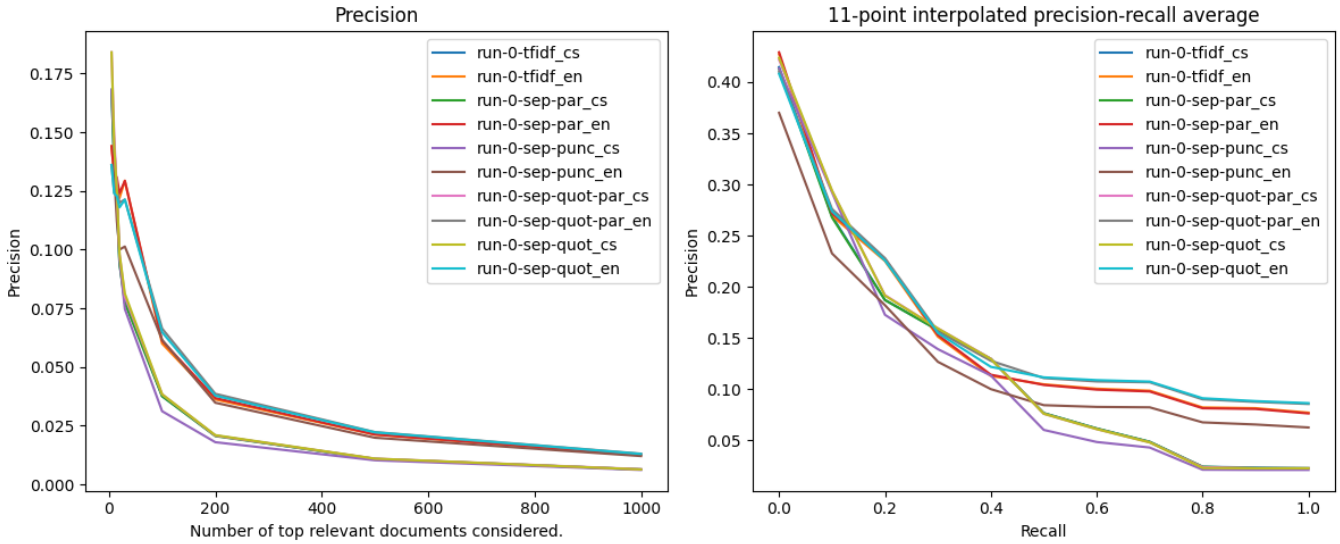

Figure 3: Precision and 11-point interpolated precision-recall curves for **run-0-sep-punc**, **run-0-sep-quot**, **run-0-seps-par**, **run-0-seps-quot-par** and **run-0-tfidf**.

As can be seen from Table 3 and Figure 3 the best set of separators for both languages is default set plus both the quotation marks and parentheses extensions. I'll use these sets going forward.

run-0-stopwords To speed up the system and to make it less sensitive to words that do not carry meaning (or that carry little of it) I used stop words. Initially I used Kaggle dataset[1] which resulted in worse scores than without stop words. Then I tried generating stop words from the dataset using a threshold of token frequency per document. Meaning a token was considered stop word if it occurred more than given times per document on average. Stop words generated this way represent more the document set than the semantics of the individual words.

The different runs and their corresponding stop word sets are visible in the Table 4.

As can be seen from Figure 4 and Table 5 the introduction of stop words only worsen the systems' scores for Czech. On the other hand the English document set seems to benefit from filtering out frequent words. Additionally one can see that the Kaggle stop word set I used was probably created with more care for English than for Czech, as the performance dip is more pronounced for the Czech document set.

Going forward I'll omit stop words for the Czech dataset and apply the 600-threshold stop word set for the English dataset.

[1] `https://www.kaggle.com/datasets/heeraldedhia/stop-words-in-28-languages`

run id	#Czech stop words	#English stop words
run-0-stopwords-kaggle	256	1298
run-0-stopwords-200	34	71
run-0-stopwords-600	8	21

Table 4: Stop word sets considered for given runs.

	Czech				English			
run ID	MAP	#relevant	#rel. & ret.	MRR	MAP	#relevant	#rel. & ret.	MRR
run-0-sep-quot-par	0.1202	382	162	0.3965	0.1345	782	327	0.3583
run-0-stopwords-200	0.1023	382	162	0.3297	0.1498	782	293	0.3392
run-0-stopwords-600	0.1023	382	162	0.3297	0.1498	782	293	0.3392
run-0-stopwords-kaggle	0.0925	382	167	0.2356	0.1324	782	293	0.3059

Table 5: Results of **run-0-stopwords-kaggle**, **run-0-stopwords-200** and **run-0-stopwords-600**.

run-0-tagblacklist Finally I've wanted to omit some SGML tags from the documents. Especially for the English dataset there are plenty of tags, which do not actually tell us anything about the document. The hypothesis is that by filtering out these additional words, the percentage of words bearing meaning is increased for each document. The system will therefore have less noisy documents to work with, which should increase its performance. The omitted tags are the following:

- for Czech documents: `DOCNO`, `DOCID`,
- for English documents: `DOCNO`, `DOCID`, `SN`, `PD`, `PN`, `PG`, `PP`, `WD`, `SM`, `SL`, `CB`, `IN`, `FN`.

	Czech				English			
run ID	MAP	#relevant	#rel. & ret.	MRR	MAP	#relevant	#rel. & ret.	MRR
run-0-sep-quot-par	0.1202	382	162	0.3965	-	-	-	-
run-0-stopwords-600	-	-	-	-	0.1498	782	293	0.3392
run-0-tagblacklist	0.0881	382	167	0.2295	0.1495	782	293	0.3392

Table 6: Results of **run-0-stopwords-600** on English, **run-0-seps-quot-par** on Czech and **run-0-tagblacklist** on both datasets.

Based on the results in Figure 5 and Table 6 we can say that tag-filtering again helped with the system's performance on the English dataset while it hindered its performance on the Czech dataset. It is so, despite the fact that the filtering was much less critical – only the document identifiers where filtered out. Though the system was able to retrieve more relevant documents overall, its precision plummeted. This would definitely require more attention as it is now unclear why this is the case.

3 Conclusion

I've managed to increase my system mean average precision from the initial 0.1114 and 0.1228 to 0.1202 and 0.1498 for the Czech and English dataset respectively. The increase in performance is much smaller for the Czech dataset and I'd suspect this is due to the fact there are much more unique word forms in the Czech dataset than in the English one. This could be probably solved by employing lemmatizer. I've looked into MorphoDiTa, but unfortunately I've had difficulties with setting it up.

4 Glossary

Throughout the text I used the following terminology:

- MAP – *mean average precision*
- #relevant – *total number of relevant documents*
- #rel. & #ret. – *total number of relevant documents returned by the system*
- MRR – *mean reciprocal rank*
- tf-idf – *term frequency-inverted document frequency term weights*

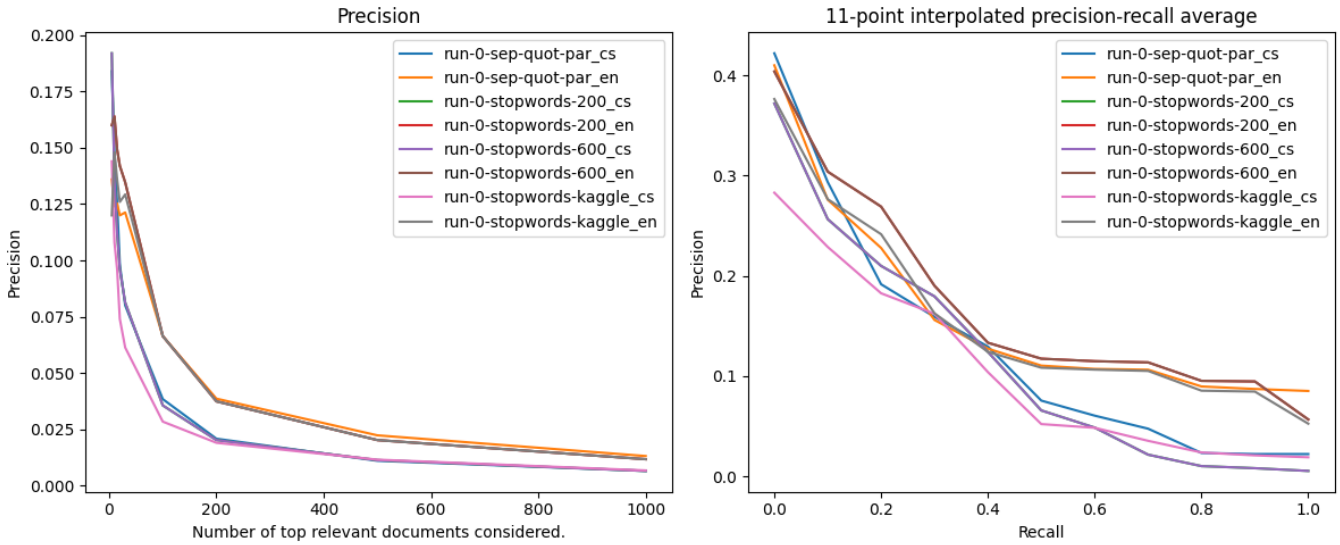

Figure 4: Precision and 11-point interpolated precision-recall curves for **run-0-stopwords-kaggle**, **run-0-stopwords-200** and **run-0-stopwords-600**.

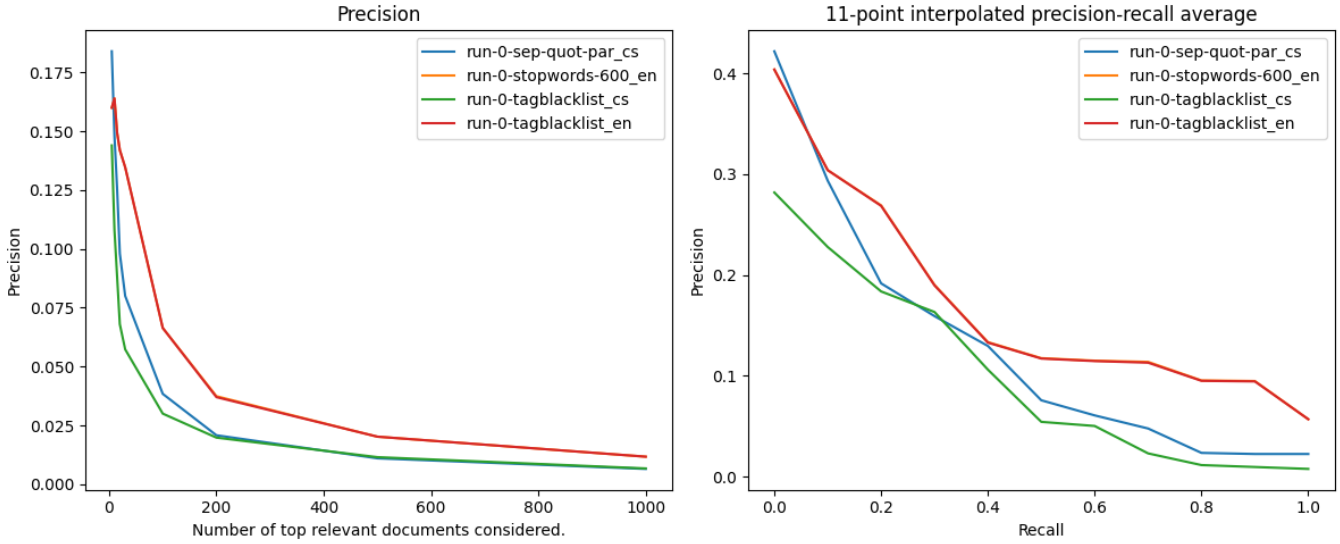

Figure 5: Precision and 11-point interpolated precision-recall curves for **run-0-seps-quot-par** on Czech and **run-0-tagblacklist** on both datasets.