Information Retrieval – First Homework

Ondřej Měkota

November 26, 2019

Introduction

My approach consists of implementing most variants (from the lecture) of term weighting, document frequency weighting, normalization, etc. and then running fairly extensive search over *many* combinations of these hyperparameters.

1 Description of my IR system

Written in Python 3, the source codes should be reasonably easy to read. I depend on many external libraries. Mainly on NumPy — a math library for storing numbers, matrices and providing interface for operations on them.

After parsing the documents from XML, and extracting desired information from them (tf, idf, etc.), The information is stored as a *sparse* matrix, where each document is assigned a column and each unique word a row. Topics are stored in the same way – topic has a row and words have columns. When performing cosine similarity retrieval, we simply do a dot product.

As we were required to implement document vector storage. Dense matrix would not fit into 16GB memory. Therefore I decided to implement a sparse matrix. It is a compressed sparse row matrix (and compressed sparse column) [1]. It provides fast matrix multiplication, given the other matrix is in suitable format.

2 Run-0: baseline

The system simply follows the restrictions from the assignment description.

[1] CSR matrix: `https://en.wikipedia.org/wiki/sparse_matrix`

3 Run-1: constrained

I have run approximately 18 000 experiments. I have used `neptune.ml` to keep track of my experiments. They can be seen on the webpage referenced in footnote [2] (there is a simple guide how to sort the results by mean average precision). I have not been able to export the measurements from the website.

The parameter optimization has been done in several *waves*. Starting from every combination of tf, df for both query and document separately, lowercasing and numbers processing; fixing such parameter settings which clearly ouperformed different sets of parameters regardless of other parameters — for example *log average* tf and *idf* have better result whether numbers processing is turned on or not.

Other experiments were performed to see how the best methods perform on english dataset (generally better than on the czech one). And to select other hyperparameters – normalization, *sort of* query expansion (taking every substring of lenghth at least 3 of every words in query and adding it to query), removing *garbage* words (nonsensical words containing a lot of interpunction and deleting stopwords). Except stopwords removal, most of the methods did not improve the performance significantly.

4 Run-2: unconstrained

This differs from run-1 only in allowing queries to be created also from *desc* and *narr* fields. Again, a search has been run to find which combination of fields is the best.

I have observed that for english the best system used all fields form query, however for czech it only used *title* and *desc*.

5 Results

Results (MAP and P_10) are as follows:

[2] `https://ui.neptune.ml/pixelneo/retrieval/experiments`

Run	Czech	English
Run-0	0.0567	0.0455
Run-1	0.3020	0.3516
Run-2	0.3156	0.3843

Table 1: Mean average precision of each run

Run	Czech	English
Run-0	0.0640	0.0720
Run-1	0.3480	0.3960
Run-2	0.3480	0.4280

Table 2: Precision of the first 10 documents (P_10)

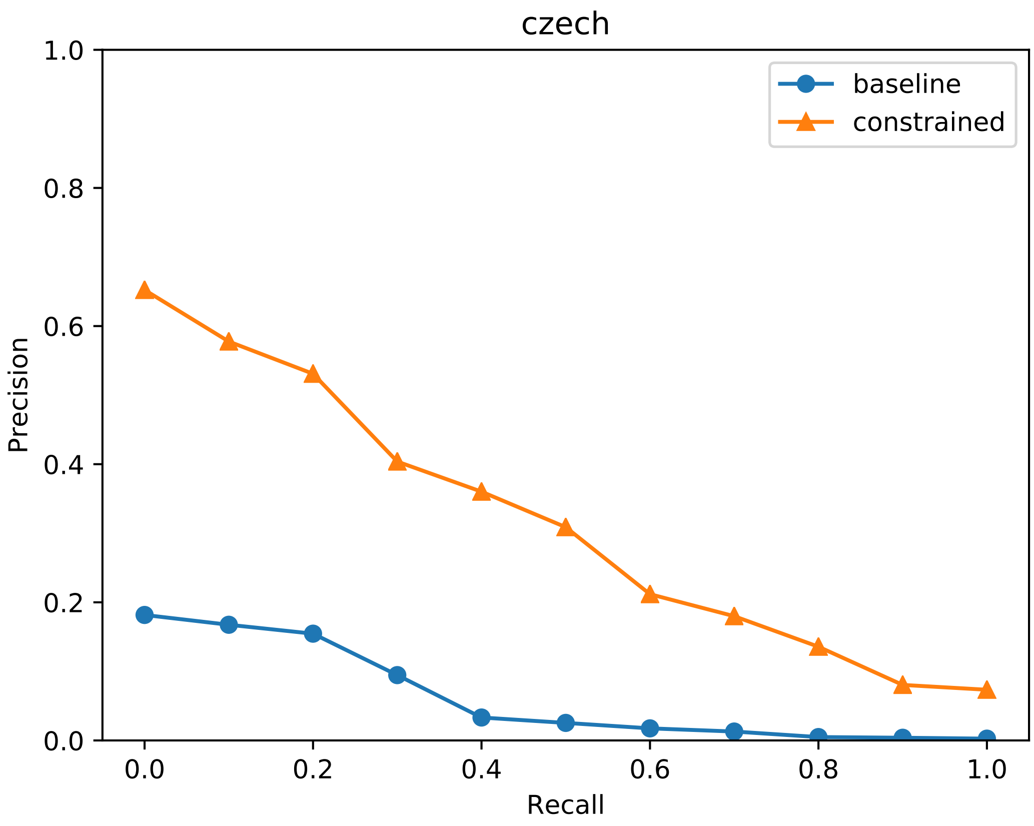

Figure 1: Precision-recall curve for the czech dataset.

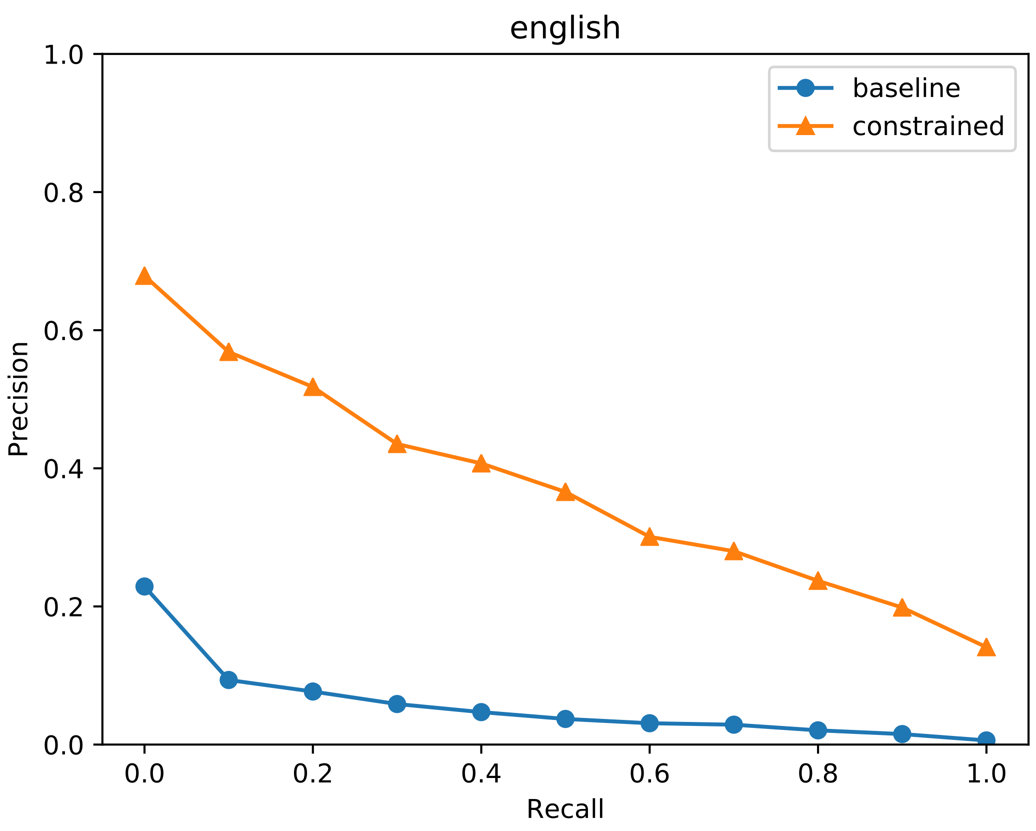

Figure 2: Precision-recall curve for the english dataset.

6 Details

Parameters for the final runs can be found in `args` directory. Parameters for each experiment are on the *neptune* website [3].

6.1 Instructions

There is a `README` file with instruction on how to run the retrieval system.

Computer has to have `xmllint` installed (all faculty computers in lab have it) and Python 3. Required packages are in file `requirements.txt` which can be installed by `pip3 install -r requirements.txt` Also morphodita files need to be downloaded.

All this can be done by running `make build`

[3]`https://ui.neptune.ml/pixelneo/retrieval/experiments`

The interface of the `run` program requires an optional argument, that is `-document_path` which should be the path to the folder with document files, it can be blank if the the list of documents (given by parameter `-d`) containes the whole (relative) path from `run` file to the individual documents.

Conclusion

I think that the results are reasonably good. Constrained and unconstrained achieved significantly better the performance compared to baseline.

Where I see a problem is the time performance of document parsing and sparse matrix creation. I do believe that it could run much faster, however I have not been able to identify the weak spot, and since time performance is not graded, I left it there.