Search Engine

Paweł Konopka

1. Dataset

Dataset consists entirely of English **Wikipedia articles** somehow connected to **history**. Their contents and titles were downloaded using wikipedia python library as queries with following keywords:

```
KEYWORDS = [
    "world history",
    "human history",
    "evolution",
    "homo",
    "medieval",
    "middle ages",
    "renaissance",
    "war",
    "battle",
    "revolution",
    "invasion",
    "europe",
    "asia",
    "america",
    "africa",
    "poland",
    "european union",
    "united states",
    "united kingdom",
    "history",
    "archeology",
    "leader",
    "king",
    "queen",
    "emperor",
    "general",
]
```

In total **10,178 articles** were collected. Even though there is no obstacle to download any number of wikipedia pages, I was forced to stop here because of RAM limitations on my PC.
The whole code regarding Wikipedia crawling is located in **`wikidata/wiki_download.py`**.

2. Data pre-processing

a. Splitting

In order to efficiently use the dataset, it was required to pre-process its content. First step was to split each article into words (called later terms). I decided to simply split each article string with any of the following characters as separator:

```
re.split(r'[  \t\n\r\v\f,.!:;\-=/\\()\[\]\'"]', line)
```

The fact that helped a lot is that Wikipedia articles are free of HTML tags, that could be a problem otherwise. After splitting only those words are taken into consideration, whose length is greater than one and all characters are alphabetic. This way all numbers are ingored, which may not be the best solution (for example for differentiating between World War I and World War II). Next all words are being transformed to lowercase.

b. Stemming

Word stemming was achieved by using Porter Stemmer from **nltk** Python library.

c. Stop words removal

Words from **nltk** English stop words list were manually removed from each article processing.

Functions implementing aforementioned functionalities can be found in **`engine/text_analysys.py`**.

3. Creating vocabulary

Vocabulary was created by counting all usages of each unique term (after pre-processing) in all articles. Vocabulary size was arbitrarily set to **5,000**. Again RAM limitations were the only obstacle in increasing this number. All other words than the 5,000 most used in articles after pre-processing were simply ignored later on by the search engine.

4. Term frequency

Term frequency (**tf**) is given as a share each term holds in each article. Formally we have:

$$tf(t,d) = \frac{count(t,d)}{length(d)}$$

Where: t - term, d - document
Term frequency value is always in range: $[\mathbf{0}, \mathbf{1}]$.
In Python it is represented as a list of dictionaries. List index corresponds to article index of being read. Dictionary keys are terms, and values their frequencies for given documents.

5. Document frequency

Document frequency (**df**) is given as a number of articles in which certain terms is used (at least once). The equation goes as follows:

$$df(t) = \sum_{d \in D} \frac{\mathbb{1}(count(t,d) \geq 1)}{|D|}$$

Where: D – set of all documents
Term frequency value is also always in range: $[\mathbf{0}, \mathbf{1}]$.
In python it is represented as a dictionary where each term is a key, and its document frequency a value.

6. Inverse document frequency

Inverse document frequency (**idf**) is given as:

$$idf(t) = \log \frac{|D|}{df(t) + 1}$$

Since *df* is a measure of how often does certain word appear in the text, higher *idf* value means that given term is quite rare and unique. Logarithmic function is used to smoothen this out, which tends to present better results.
Dictionary Python representation remains the same.

7. TF-IDF: Inverse Document Frequency

The final value that will be taken into consideration during computation is **tf-idf**. It is given by a formula:

$$tf\text{-}idf(t,d) = tf(t,d) \cdot idf(t)$$

It is basically *tf* value (how often does certain term appear in certain document) multiplied by its "uniqueness" (*idf* value).
In the code *tf-idf* is initially represented as a list of dictionaries (same as *tf*). Later on it is being transformed into $|T| \times |D|$ sparce matrix, where T stands for set of all terms (vocabulary). Each matrix column was also normalized

8. Singular Value Decomposition

Using Sigular Value Decomposition we are able to transfer matrix M as product of three matrices:

$$M = U\Sigma V^T$$

Truncation of SVD allows us to reduce matrix size. I used ready implementation from **sklearn** Python library. It returns two matrices: $(U\Sigma)_{|T|\times k}$ and $V^T{}_{k\times|D|}$.
Since it is an approximation, their product will not be exactly equal to *tf-idf* matrix.

9. Frontend layout

Frontend layout was added using **Django Python framework** with plain **HTML** and **CSS**.

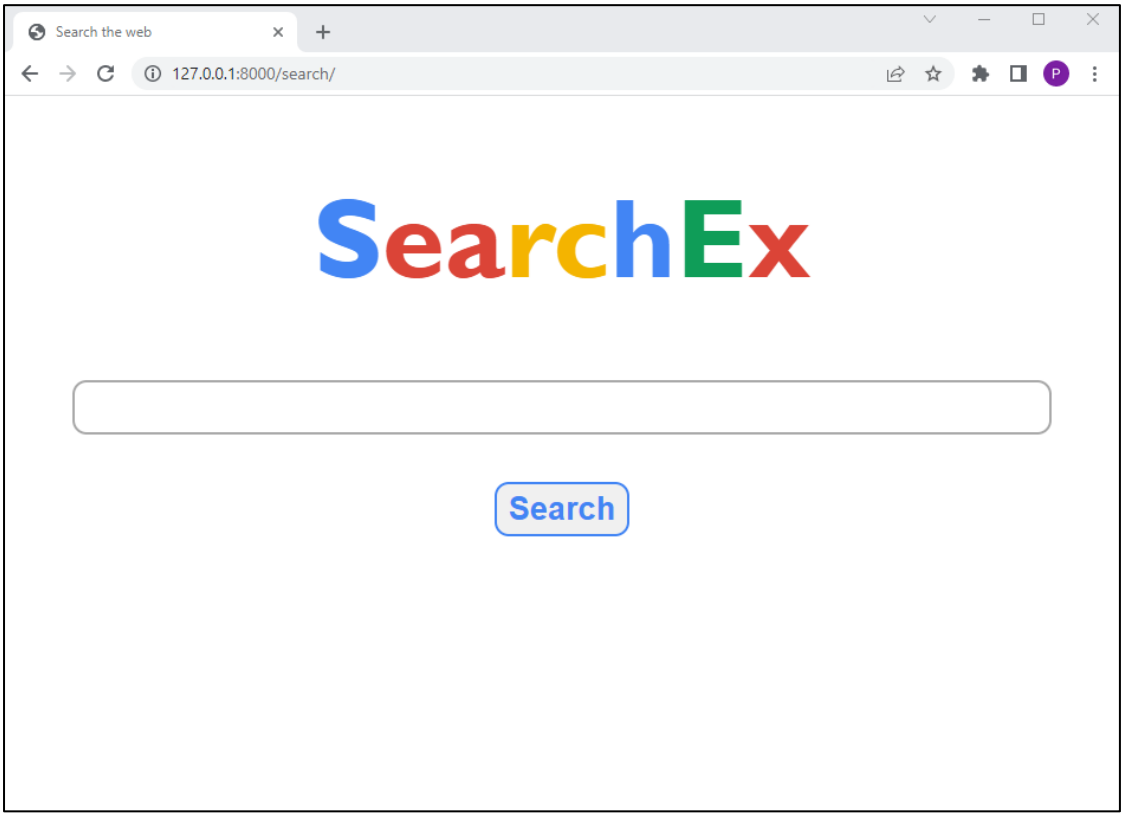

Figure 1: Search page graphical view

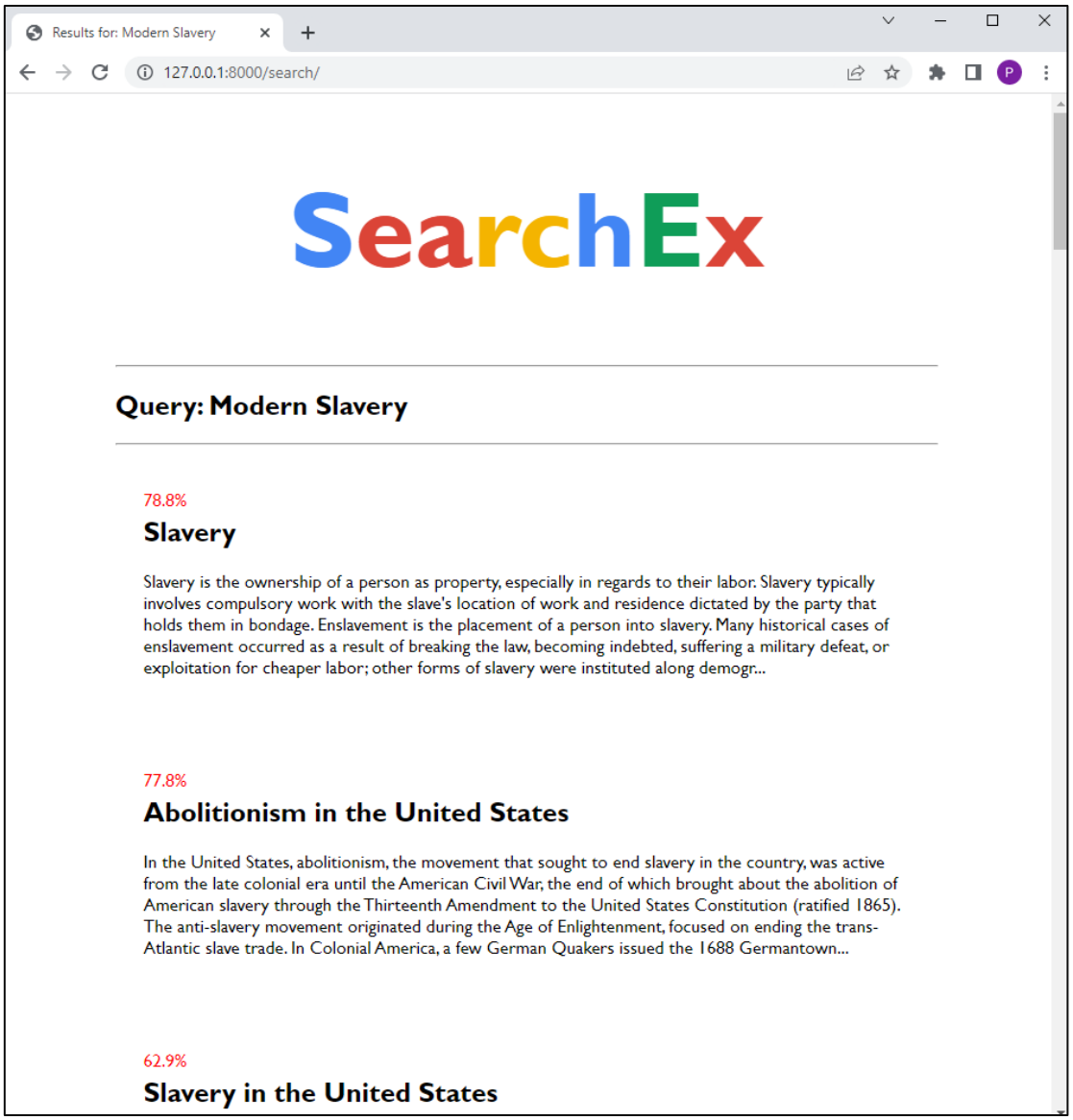

Figure 2: Search page graphical view

10.Result comparison using different parameters

a. Query: Ukrainian people and culture

i. No SVD search

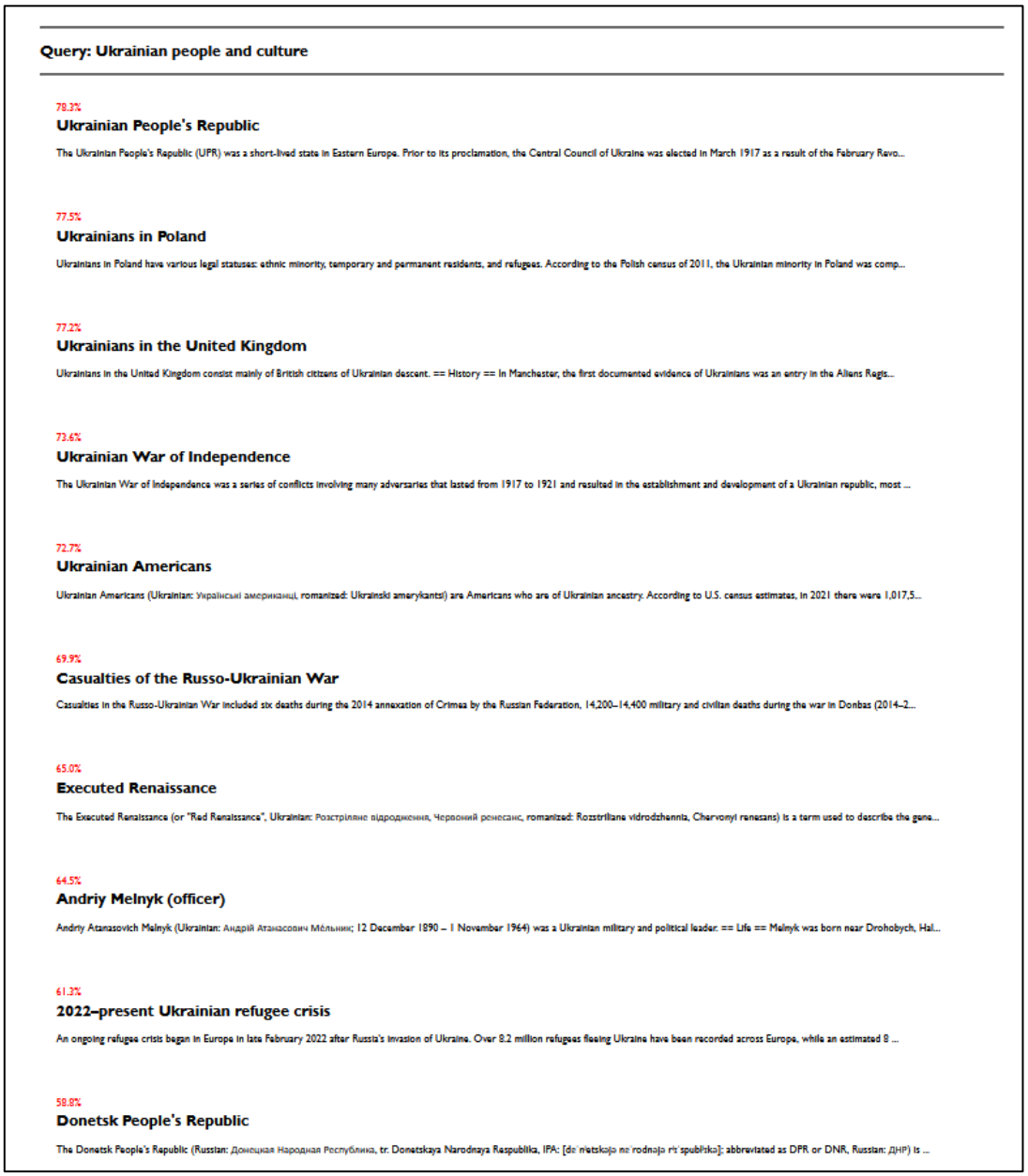

Figure 3: Results for query “Ukrainian people and culture” with SVD turned off

ii. SVD search, k = 100

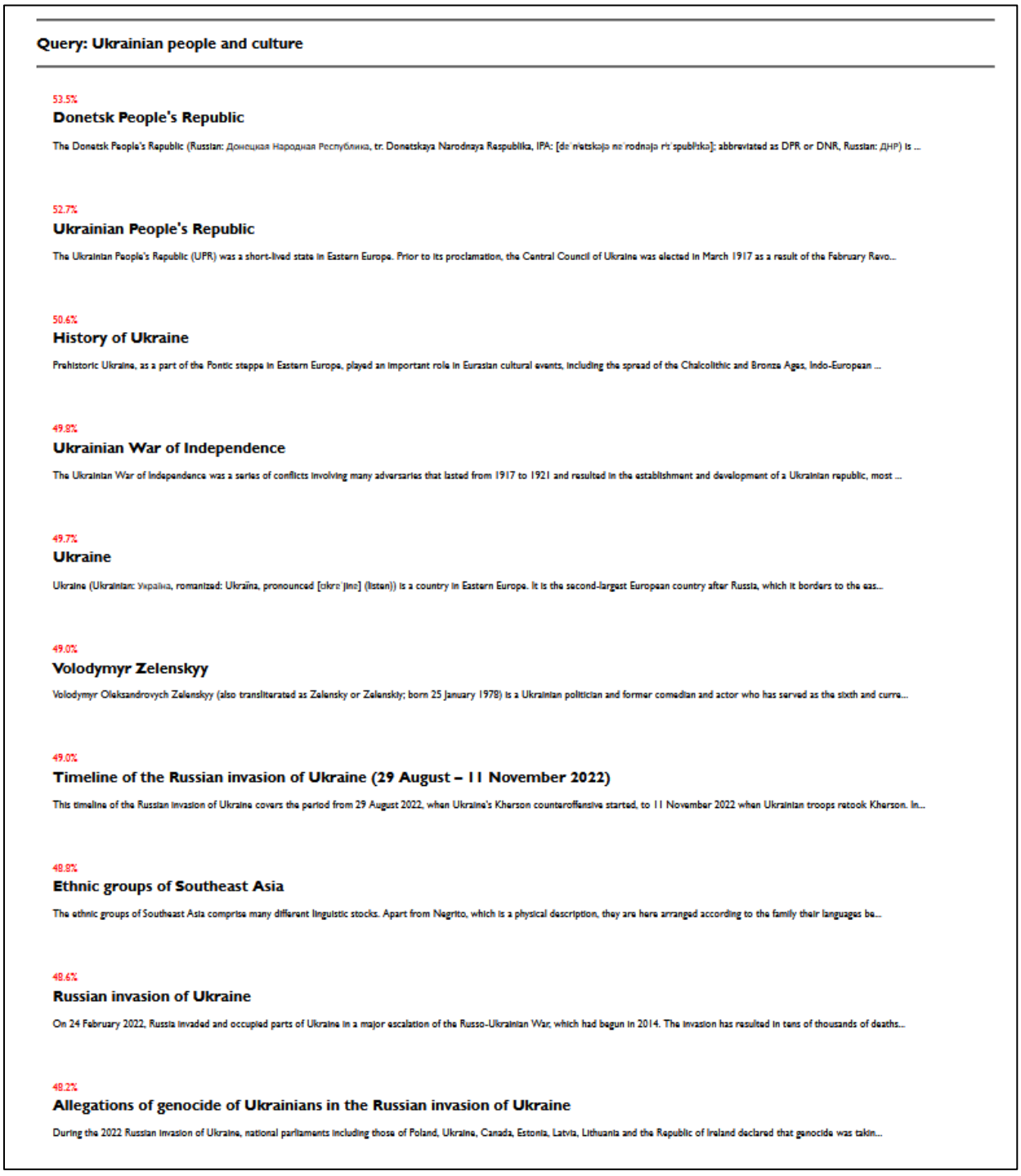

Figure 4: Results for query "Ukrainian people and culture" with SVD k=100

iii. SVD search, k = 1000

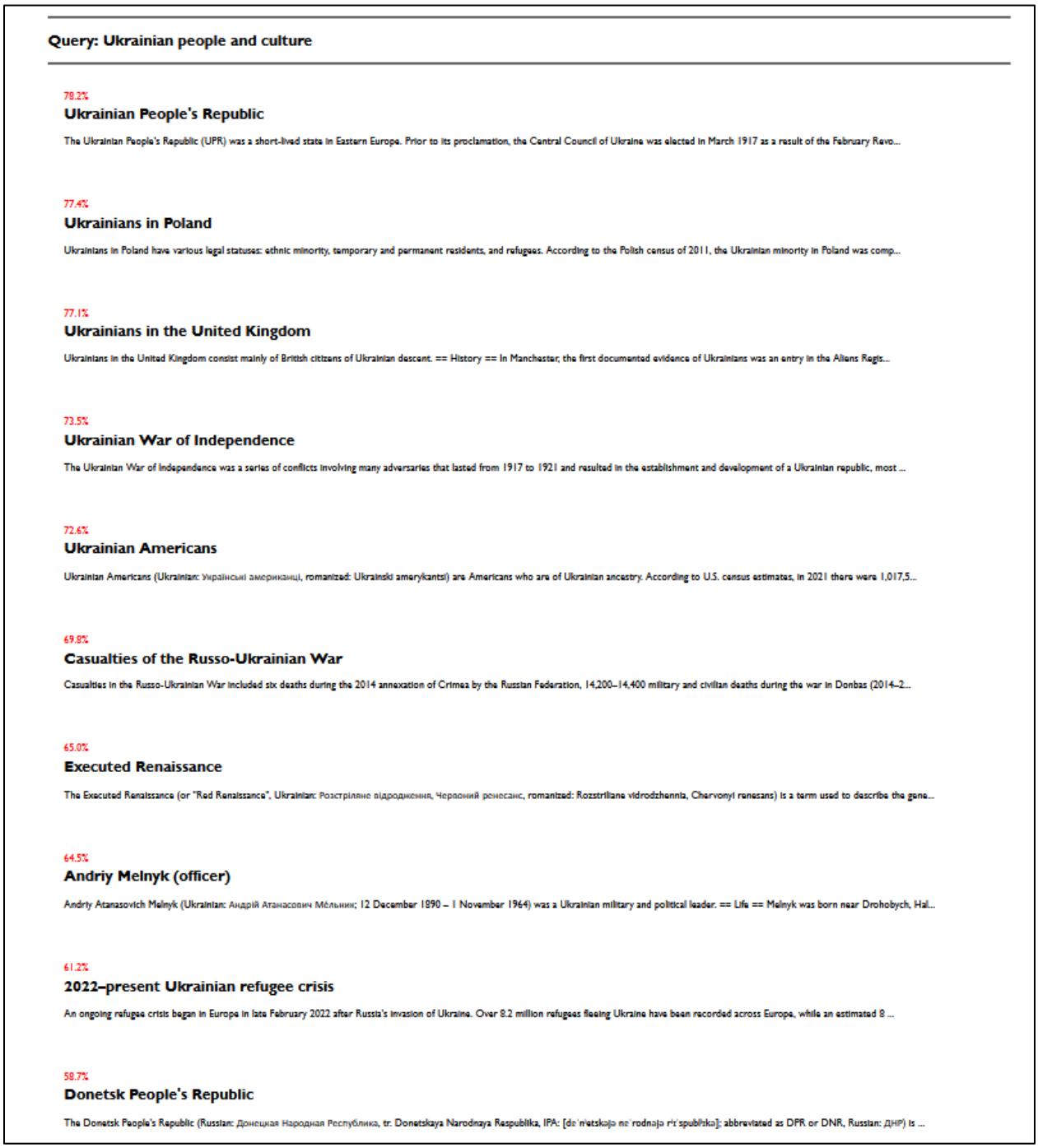

Figure 5: Results for query “Ukrainian people and culture” with SVD k=1000

b. Query: German holy military order

i. No SVD search

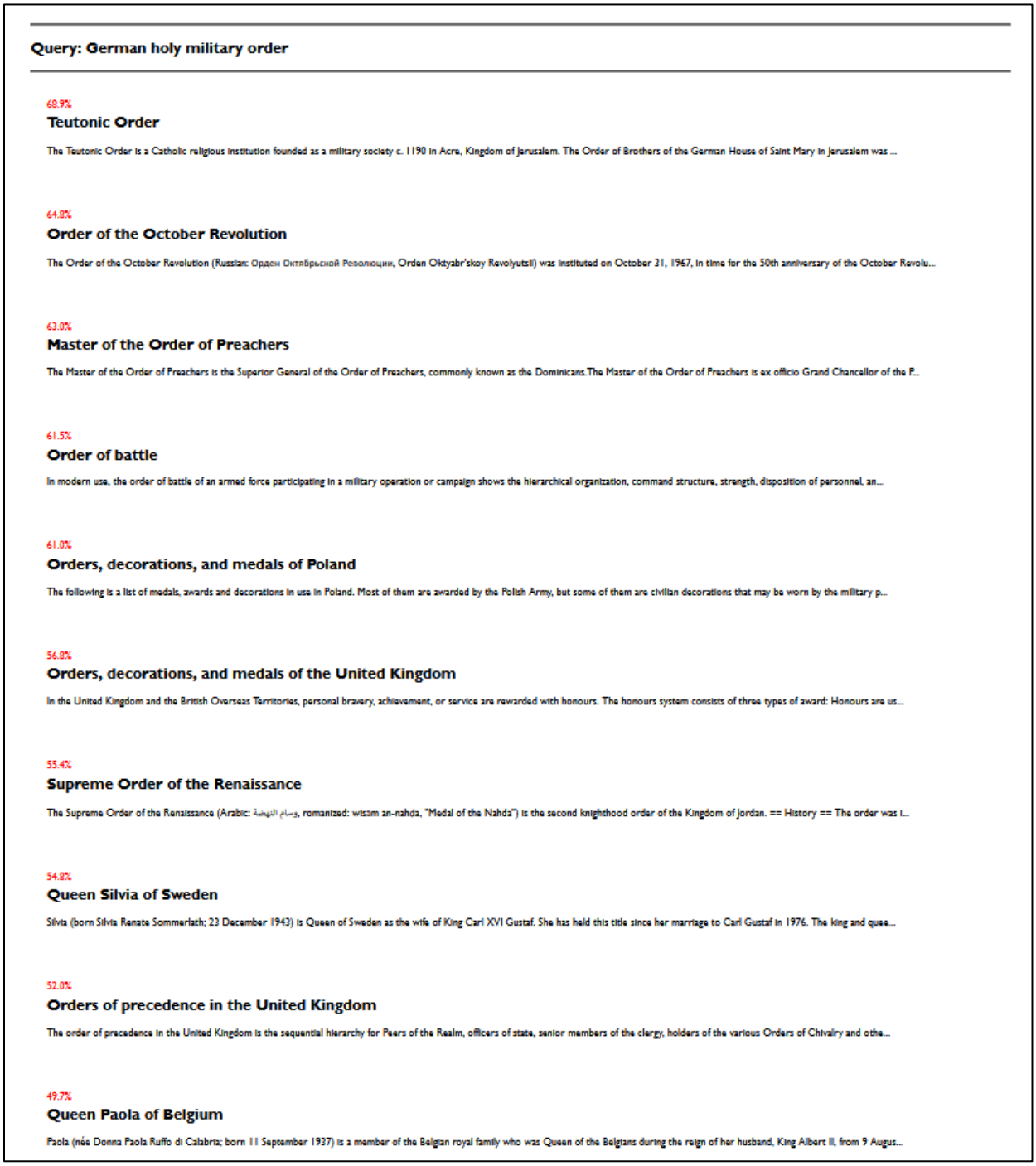

Figure 6: Results for query "German holy military order" with SVD turned off

ii. SVD search, k = 100

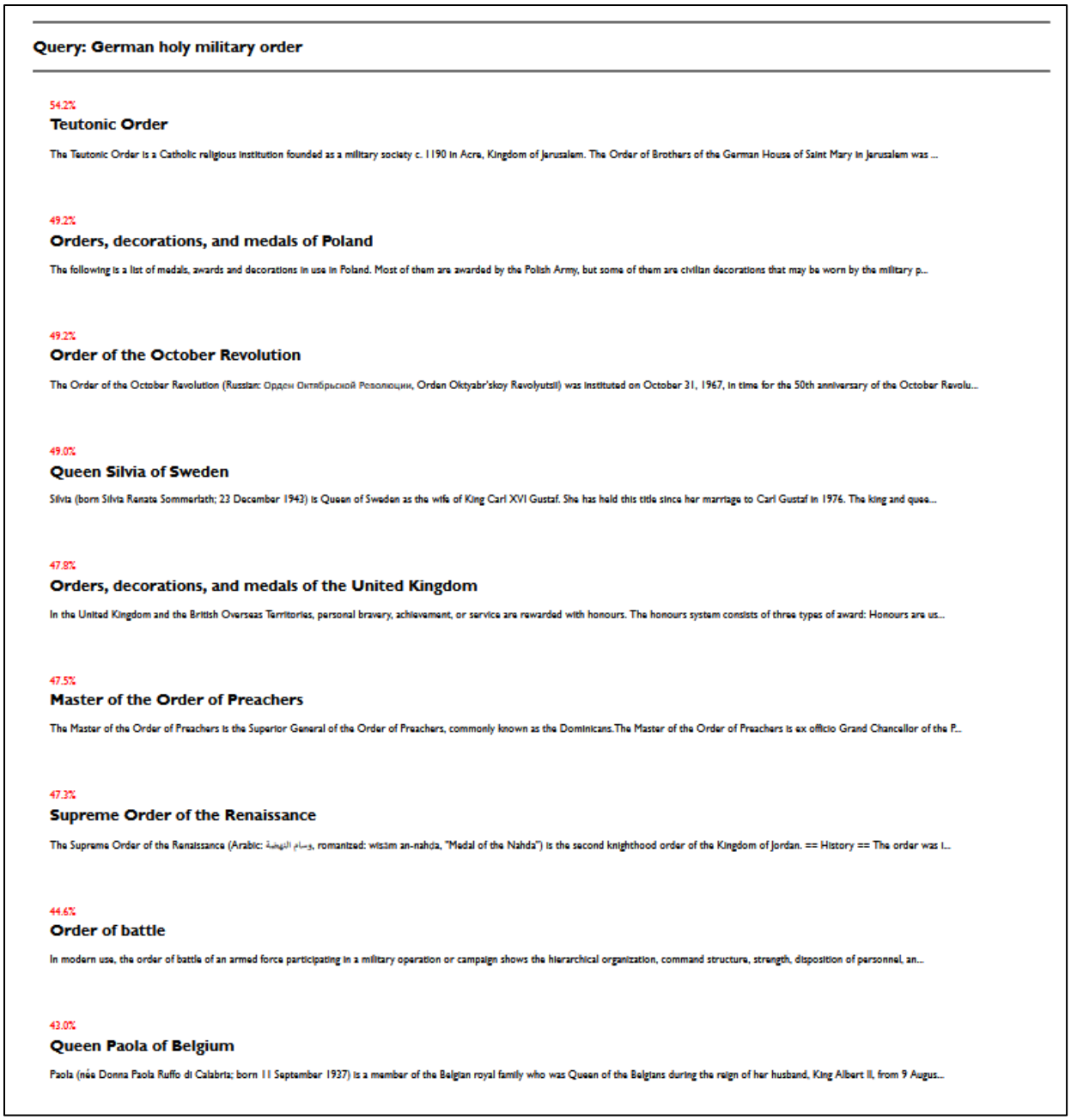

Figure 7: Results for query “German holy military order” with SVD k=1000

iii. SVD search, k = 1000

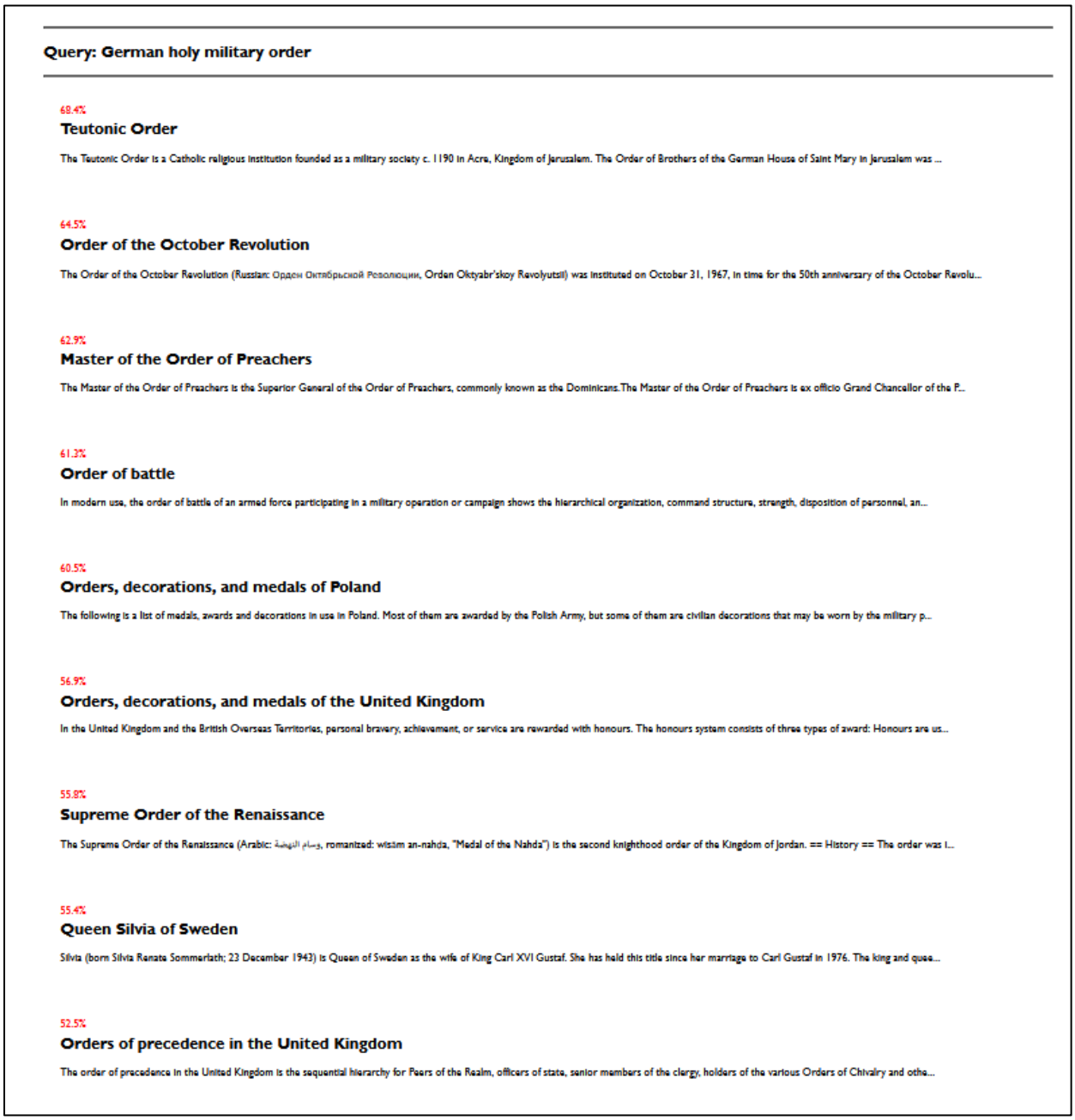

Figure 8: Results for query “German holy military order” with SVD k=1000

11. Subjective comparison between different search parameters

Using shown above examples and more testing SVD search with high k values seem to be giving very similar results to no-SVD search. In particular circumstances SVD may prove to better “understand” query meaning. On the other hand, low k values searches provide rather unrelated results.