EDAN20 - Assignment 1
Building an inverted index

Hugo Mattsson
hu5174ma-s

September 2024

1 Objectives and dataset

1.1 Objectives

The objectives of the assignment were to:

- Write a program that collects all the words from a set of documents
- Build an index from the words
- Find some way of comparing the similarity between documents
- Read about an industrial system

1.2 Dataset

The dataset was a corpus consisting of some of Selma Lagerlöf's works: Jerusalem, Nils Holgerssons underbara resa genom Sverige, Kejsarn av Protugallien, Troll och människor, Mårbacka, Osynliga länkar, Gösta Berlings saga, En herrgårdssägen, and Bannlyst

2 Method and program structure

The program was written inside a jupyter notebook and is structured into the following flow:

2.1 Finding files

Given a folder and a wanted file type ending. Find all files of that type

2.2 Indexing

For every file, the regex `r'\p{L}+'` is run to tokenize the file into words and to get their positions. This is then combined into a master index which for a given word can tell in which files it occurs and at what position(s).

2.3 Vectorizing

The documents are then vectorized, i.e. being represented as an n-dimensional vector, where n is the number of unique words in the document. All unique words are scored using tf-idf, where tf was the relative frequency of the word in that document, and idf defined as $log_{10}(\frac{N}{d})$ where N is the total amount of documents and d is the amount of documents which include that word.

2.4 Comparison

The final part of the program is comparing the documents to each other in order to find out how similar they are. Since all documents are now represented as vectors in some n-dimensional room. They can now easily be compared using the cosine of two vectors at a time. This grants us a number between 0 and 1, where 0 means that the documents have nothing in common, and 1 that they are very similar or even identical works.

3 Results

After vectorizing and computing the cosine similarity of all documents below is the similarity matrix that was computed. In it we can see that the most similar works were *herrgard* and *jerusalem* with a similarity score of 0.3707

	troll	kejsaren	marbacka	herrgard	nils	osynliga	jerusalem	bannlyst	gosta
troll	1.0000	0.1813	0.1472	0.0041	0.1885	0.1926	0.0071	0.0886	0.1957
kejsaren	0.1813	1.0000	0.0711	0.0007	0.0497	0.0511	0.0018	0.0240	0.0480
marbacka	0.1472	0.0711	1.0000	0.0036	0.0847	0.0932	0.0049	0.0368	0.0802
herrgard	0.0041	0.0007	0.0036	1.0000	0.0051	0.0048	**0.3707**	0.0009	0.0031
nils	0.1885	0.0497	0.0847	0.0051	1.0000	0.1106	0.0045	0.0510	0.1048
osynliga	0.1926	0.0511	0.0932	0.0048	0.1106	1.0000	0.0283	0.0521	0.1248
jerusalem	0.0071	0.0018	0.0049	0.3707	0.0045	0.0283	1.0000	0.0065	0.0043
bannlyst	0.0886	0.0240	0.0368	0.0009	0.0510	0.0521	0.0065	1.0000	0.0490
gosta	0.1957	0.0480	0.0802	0.0031	0.1048	0.1248	0.0043	0.0490	1.0000

4 Conclusion

This has been a useful in assignment in learning about indexing, vectorizing and comparing different texts.

5 Answer to possible questions

The indexing/encoding technique used in this lab is called a *posting list*, which means that each indexed word is associated with its positions one or more files:

```
word1: file_name1 pos1 pos2 pos3... file_name2 pos1 pos2...
word2: file_name1 pos1 pos2 pos3... file_name2 pos1 pos2...
...
```

The technique used by Google that I find most similar is found on slide 45 labeled *Index Encoding circa 1997-1999*. Which, for every indexed word, stores the related document id and number of hits for the given word. This is followed by the hits which includes the word's position as well as other relevant attributes(see below figure 1).

Figure 1: Google's Index encoding 1997-1999

The differences between mine and Google's approach is that I don't store any word attributes other than position, and that Google explicitly encodes the number of hits for a word, which in my case is handled implicitly as the length of the python list.