Language Technology - EDAN20
Assignment 1 - Fall 2020

Building an inverted index
Tf-Idf value

Hicham Mohamad (hi8826mo-s)

October 21, 2020

1 Introduction

In this assignment, we need to write a Python program that first collects all the words from the corpus of Selma Lagerlöf, extracted from Lagerlöf arkivet at Litteraturbanken. Then, we will learn how to build an index from the collected words. At the end, we will get familiar with the Tf-Idf value to represent documents. An additional task consists in reading about the history of Google indexing by Jeff Dean.

2 Programming the Indexer

The task here is to implement an **indexer** that takes a corpus as input and outputs all the words with their positions in each document.

In this way, the returned index will consist of **rows** with one word per row and **posting list**, i.e. the list of files and positions, where this word occurs. The **position** of a word is encoded as the number of characters from the start of the file, i.e the word offset.

```
word1: file_name pos1 pos2 pos3... file_name pos1 pos2 ...
word2: file_name pos1 pos2 pos3... file_name pos1 pos2 ...
```

we split the implementation into **three steps** where **dictionaries** are used to represent the postings:

1. Index one file
2. Read the content of a folder
3. Create a master index for all the files

2.1 Indexing one file

To solve this essential task, we write a program that reads one document/text file and outputs an index file. It is nice to show the fundamental parts of this implementation:

- we represent this index as a **dictionary** where the keys is the **unique words** and the values, the lists of positions
- To extract the words, we implement the function `tokenize()` using `regex`. With **Unicode regular expressions**, we can explicitly define the **content of words** and consequently extract a sequence of words, where a word is defined as a sequence of contiguous letters.
- `finditer()` is used to find the positions of the words. This returns **match objects**, where we can get the matches and the positions with the `group()` and `start()` methods.

- we use the **pickle package** to write dictionary in an file.

After downloading the Selma folder, we run the implemented Python code on it. Below is an excerpt of the index of the `bannlyst.text` for the words *gjord*, *uppklarnande*, and *stjärnor*. The data is stored in a dictionary:

```
{...
'gjord': [8600, 183039, 220445],
'uppklarnande': [8617],
'stjärnor': [8641], ...
}
```

2.2 Reading the content of a folder

Using the Python **os package**, we write the function `get_files()` that reads all the files in a folder with a specific suffix (txt) and returns the file names in a list.

After reusing this function, `get_files('Selma', '.txt')`, we could output the following content of Selma:

```
['bannlyst.txt',
 'gosta.txt',
 'herrgard.txt',
 'jerusalem.txt',
 'kejsaren.txt',
 'marbacka.txt',
 'nils.txt',
 'osynliga.txt',
 'troll.txt']
```

2.3 Creating a master index

At the end, we complete our program with the creation of master index, where we associate each word of the corpus with the **files** (where it occurs) and its **positions** (a posting list).

We build the master index of corpus Selma with a loop over the list of files, where we first tokenize each file and then returns a dictionary of (words : positions) using the implemented function `text_to_idx()`. Below is an excerpt of the master index with the words *samlar* and *ände*:

```
'samlar':
            {'troll.txt': [641880, 654233],
             'nils.txt': [51805, 118943],
             'osynliga.txt': [399121],
             'gosta.txt': [313784, 409998, 538165]}
'ände':
            {'troll.txt': [39562, 650112],
             'kejsaren.txt': [50171],
             'marbacka.txt': [370324],
             'nils.txt': [1794],
             'osynliga.txt': [272144]}
```

2.4 Concordances

As an additional task, we get familiar with the important master index and see how to use it in order to find concordances in the text. Concordances are used to read **occurrences** of the given terms (or expression) in their respective context.

For this task, we write the function `concordance(word, master_index, window)` to extract the concordances of a word within a window of characters. Below is an output of the concordances with the word *samlar*:

```
gosta.txt
        om ligger nära Borg, och samlar ihop ett litet mid
```

```
        lika förstämda. Men hon samlar upp allt detta som
        n ensam i livet. Därmed samlar han korten tillhop
nils.txt
         bara, att du i all hast samlar ihop så mycket bos
        ar stannat hemma, och nu samlar de sig för att int
osynliga.txt
         till höger i kärran och samlar just ihop tömmarna
troll.txt
        en örtkunnig läkare, som samlar in markens växter
        älper dem, och medan hon samlar och handlar för de
```

3 Representing documents with tf-idf

After creating the index, we need here to represent each document in the corpus as a **word vector**. You will define the value of a word in a document with the **tf-idf metric**. In this assignment, we adopt the following definitions:

- tf is the **relative frequency** of the term in the document.
- idf is the **logarithm** base 10 of the inverse document frequency.

Conceptually, the **tf-idf representation** is a vector, where all the words in the corpus are keys: each dictionary will include all the words of the corpus as keys. The value of the key is then possibly 0, meaning that the word is not in the document or is in all the documents as shown in the following example. In our example, the word *gås* has the value 0 in *bannlyst.txt* and the value 0.000101001964 in *nils.txt*

```
nils.txt
  känna    0.0
  gås    0.00010100196417506702
  nils    0.00010164426900380124
  et    0.0

bannlyst.txt
  känna    0.0
  gås    0.0
  nils    0.0
  et    0.0
```

4 Comparing Documents

Using the **cosine similarity**, we need in this task to compare all the pairs of documents with their **tf-idf representation** and present the results in a matrix, as illustrated in Table 1. Then, looking at the table we can see that the most similar novels are `herrgard.txt` and `jerusalem.txt` with similarity: 0.370689423873392.

	bannlyst	gosta	herrgard	jerusalem	kejsaren	marbacka	nils	osynlig	troll
bannlyst	1.0	0.049	0.0009	0.0065	0.024	0.0368	0.051	0.0521	0.0886
gosta	0.049	1.0	0.0031	0.0043	0.048	0.0802	0.1048	0.1248	0.1957
herrgard	0.0009	0.0031	1.0	0.3707	0.0007	0.0036	0.0051	0.0048	0.0041
jerusalem	0.0065	0.0043	0.3707	1.0	0.0018	0.0049	0.0045	0.0283	0.0071
kejsaren	0.024	0.048	0.0007	0.0018	1.0	0.0711	0.0497	0.0511	0.1813
marbacka	0.0368	0.0802	0.0036	0.0049	0.0711	1.0	0.0847	0.0932	0.1472
nils	0.051	0.1048	0.0051	0.0045	0.0497	0.0847	1.0	0.1106	0.1885
osynlig	0.0521	0.1248	0.0048	0.0283	0.0511	0.0932	0.1106	1.0	0.1926
troll	0.0886	0.1957	0.0041	0.0071	0.1813	0.1472	0.1885	0.1926	1.0

Table 1: The computed similarity matrix between the documents of the corpus Selma.

5 Reading

After reading the slides about the history of **Google indexing** by Jeff Dean: *Challenges in Building Large-Scale Information Retrieval Systems* in [6], we can conclude that our **index encoding** is related to what Google did in slides 16 and 45 where the indexing partitioning are described in two way as:

- **By doc**: each shard has index for subset of docs
- **By word**: shard has subset of words for all docs

That "By word" way can be the most similar indexing technique and then in slide 45 it is described more in details.

6 Appendix

References

[1] Pierre Nugues, *Language processing with Perl and Prolog*, 2nd edition, 2014, Springer.

[2] P. Nugues, Language Technology - EDAN20, Lectures notes: `https://cs.lth.se/edan20/`

[3] VanderPlas, A Whirlwind Tour of Python. O'Reilly 2016. Online reading: `https://jakevdp.github.io/WhirlwindTourOfPython/`

[4] Tf-Idf measure: `https://en.wikipedia.org/wiki/Tf-idf`.

[5] Geron, Jupyter notebook on linear algebra from Machine Learning and Deep Learning in Python using Scikit-Learn, Keras and TensorFlow: `https://github.com/ageron/handson-ml2`

[6] Jeff Dean, Google Indexing: *Challenges in Building Large-Scale Information Retrieval Systems*, `https://static.googleusercontent.com/media/research.google.com/en//people/jeff/WSDM09-keynote.pdf`