Language Technology - EDAN20
Assignment 1 - Fall 2020

Building an inverted index
Tf-Idf value

Hicham Mohamad (hi8826mo-s)

October 22, 2022

1 Objectives

The objectives of this assignment are to:

- Write a program that collects all the words from a set of documents
- Build an index from the words
- Know what indexing is
- Represent a document using the Tf-Idf value
- Write a short report of 1 to 2 pages on the assignment
- Read a short text on an industrial system

2 Organization and location

The first lab session will take place on September 10 and 11.

There can be last minute changes. Please always check the official times here:

`https://cloud.timeedit.net/lu/web/lth1/ri1Q5006.html`

You can work alone or collaborate with another student.

Each group will have to:

- Write a Python program.
- Check the results and comment them briefly.

3 Preparation

1. Download the `Selma folder` and uncompress it. It contains novels by **Selma Lagerlöf**. The text of these novels was extracted from *Lagerlöf arkivet* at *Litteraturbanken*.
2. Read the description of the **tf-idf measure** on Wikipedia (`https://en.wikipedia.org/wiki/Tf-idf`).

4 Programming the Indexer

Your program will take a corpus as input (here the Selma novels) and produce an index of all the words with their positions. you should be able to run it this way:

```
$ python indexer.py folder_name
```

Conceptually, the index will consist of rows with one word per row and the list of files and positions, where this word occurs. Such a row is called a **posting list**. You will encode the position of a word by the number of characters from the start of the file.

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

To make programming easier, you will split this exercise into **three steps** and you will use **dictionaries** to represent the postings.

Indexing one file

Write a program that reads one document `file_name.txt` and outputs an index file: `file_name.idx`:

1. The index file will contain all the **unique words** in the document, where each word is associated with the list of its positions in the document.
2. You will represent this index as a **dictionary** where the keys will be the words and the values, the lists of positions
3. As words, you will consider all the **strings of letters** that you will set in lower case. You will not index the rest (i.e. numbers, punctuations, or symbols).
4. To extract the words, you will use **Unicode** regular expressions. Do not use `\w+`, for instance, but the Unicode equivalent.
5. The word **positions** will correspond to the number of characters from the beginning of the file. (The **word offset** from the beginning)
6. You will use `finditer()` to find the positions of the words. This will return you match objects, where you will get the matches and the positions with the `group()` and `start()` methods.
7. You will use the **pickle package** to write your dictionary in an file, see `https://wiki.python.org/moin/UsingPickle`.

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], ...
}
```

The word gjord occurs three times in the text at positions 8600, 183039, and 220445, uppklarnande, once at position 8617, and stjärnor, once at position 8641.

Reading the content of a folder

Write a function that reads all the files in a folder with a specific suffix (txt). You will need the Python **os package**, see `https://docs.python.org/3/library/os.html`. You will return the file names in a list.

You can reuse this function:

```
def get_files(dir, suffix):
    """
    Returns all the files in a folder ending with suffix
    :param dir:
    :param suffix:
    :return: the list of file names
    """
    files = []
    for file in os.listdir(dir):
        if file.endswith(suffix):
            files.append(file)
    return files
```

Creating a master index

Complete your program with the creation of master index, where you will associate each word of the corpus with the **files**, where it occur and its **positions**. (a posting list)

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]}
```

The word *samlar*, for instance, occurs three times in the file *gosta.txt* at positions 313784, 409998, and 538165.

5 Representing documents with tf-idf

Once you have created the index, you will represent each document in your corpus as a **word vector**. You will define the value of a word in a document with the **tf-idf metric**. Tf will be the relative frequency of the term in the document and idf, the logarithm base 10 of the inverse document frequency.

You have below the tf-idf values for a few words. In our example, the word *gås* has the value 0 in *bannlyst.txt* and the value 0.000101001964 in *nils.txt*

```
troll.txt
  känna    0.0
  gås    0.0
  nils    2.148161748868631e-06
  et    0.0
kejsaren.txt
  känna    0.0
  gås    0.0
  nils    8.08284798629935e-06
  et    8.273225429362848e-05
marbacka.txt
  känna    0.0
  gås    0.0
  nils    7.582276564686669e-06
  et    9.70107989686256e-06
herrgard.txt
  känna    0.0
```

```
  gås    0.0
  nils    0.0
  et    0.0
nils.txt
  känna    0.0
  gås    0.00010100196417506702
  nils    0.00010164426900380124
  et    0.0
osynliga.txt
  känna    0.0
  gås    0.0
  nils    0.0
  et    0.0
jerusalem.txt
  känna    0.0
  gås    0.0
  nils    4.968292117670952e-06
  et    0.0
bannlyst.txt
  känna    0.0
  gås    0.0
  nils    0.0
  et    0.0
gosta.txt
  känna    0.0
  gås    0.0
  nils    0.0
  et    0.0
```

6 Comparing Documents

Using the **cosine similarity**, compare all the pairs of documents with their tf-idf representation and present your results in a matrix. You will include this matrix in your report.

Give the name of the two novels that are the most similar.

7 Reading

Read the text: *Challenges in Building Large-Scale Information Retrieval Systems*, about the history of **Google indexing** by Jeff Dean. In your report, tell how your **index encoding** is related to what Google did. You must identify the slide where you have the most similar indexing technique and write the slide number in your report.

8 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