FACOM- UFU
Teacher: Wendel Melo

# Second work on Information Organization and Retrieval 2023-1

## Description

This work consists of the implementation of the vector IR model, which should use the TF-IDF weighting (with a base 10 logarithm).

$$TF\text{-}IDF_{ij} = (1 + \log f_{ij}) \log\left(\frac{N}{n_i}\right),\ if\, f_{ij} > 0$$

$$TF\text{-}IDF_{ij} = 0,\ if\, f_{ij} = 0$$

Only a **single** program developed in Python 3 that performs the two tasks described should be delivered. The program should only use the standard Python 3 libraries, i.e. the libraries that already come with the standard installation of the language interpreter, with the exception of the nltk library, which should be used for removing *stopwords* and extracting radicals (as described above).

**Important note:** if copying or any kind of cheating between assignments is detected, everyone involved will be penalized with a zero grade. So think carefully before asking to copy your classmate's work, as they could be punished too!

It is mandatory to use the nltk package to extract radicals from vocabulary terms and obtain a valid list of *stopwords*. The details of the weighting and the model are described below. **It is important to read carefully and follow all the details of the specification under penalty of losing points in the work grade!**

## The *stopwords*

*Stopwords* are terms that, taken in isolation, do not contribute to understanding the meaning of a document. Note then that *stopwords* should not be taken into account when generating the inverted index or processing queries! Your program should consider removing *stopwords* as specified in paper 1.

## Consultations

The queries to be answered by the system are made up of terms connected only by the & (AND) operator. Thus, the system must be able to answer queries such as the following:

| | |
|---|---|
| • sambaRead | : samba |
| • dog & catRead | : dog AND cat |
| • car & wheel & engineRead | : (car AND wheel AND engine) |

It's important to note that the vector model provides for the ranking of documents based on their similarity to queries. Thus, your program must present the documents in the correct order, taking into account the similarity between the document and the respective query it answers.

## Entering the program

Your program should receive two arguments as input **from the command line**. The first argument specifies the path of a text file that contains the paths of all the files that make up the base, each on one line. The second argument specifies the path of a text file containing a query to be answered.

**Example:** Let's assume that our base is made up of files *a.txt*, *b.txt* and *c.txt*. Let's also assume that our program is called *vector_model.py*. We would then call our program from the command line by doing:

```
> python vector_model.py base.txt query.txt
```

where the *base.txt* file contains the paths to the files that make up the document base, as follows:

```
a.txt
b.txt
c.txt
```

*base.txt*

```
home & love
```

*query.txt*

*The query.txt* file contains a query to be answered by the IR system, written on a single line in the format specified above.

## Leaving the program

The program should generate two output files, with names and contents exactly as follows:

- *weights.txt* : file containing TF-IDF weights for each document
- *answer.txt* : file with the names of the documents that answer the user's query

### The weights.txt file

The program must generate a text file called weights*.txt* which contains the weights of each term in each document according to the TF-IDF weighting (with base 10 logarithm calculation). Each line of this file must contain the non-zero weights of the terms in one of the documents in the database. For example, consider the example we saw in class, where we assume that W, X and Y are the terms in our vocabulary, after removing *stopwords* and extracting radicals.

The document base has the following content:

| Document | Contents |
|---|---|
| doc1 | W W X |
| doc2 | W W Y |
| doc3 | W W |
| doc4 | X X |

As calculated in class, the TF-IDF weight vectors of the documents are given by:

| Document | Weight vector |
|---|---|
| doc1.txt | (0.1845, 0 .3010, 0) |
| doc2.txt | (0.1625, 0, 0 .6021) |
| doc3.txt | (0.1625, 0 , 0) |
| doc4.txt | (0, 0.3916, 0 ) |

Thus, the contents of the *weights.txt* file will be given by (note again that only non-zero weights should be represented):

```
doc1.txt:  W, 0.1845     X, 0.3010
doc2.txt:  W, 0.1625     Y, 0.6021
doc3.txt:  W, 0.1625
doc4.txt:  X, 0.3916
```

*weights.txt*

Note that, for each document in the database, we have a list of t,q pairs where t is the term, and q is its respective weight according to the weighting adopted. So, for doc2.txt, we have the pair W,0.1625. indicating that the term W has a weight of 0.1625 in that document, and the pair Y,0.6021 indicating that the term Y has a weight of 0.6021 in that document. **Note that only non-zero weights should be represented in the weights.*txt* file.** You should use TF-IDF weighting to calculate the similarities in the vector model. However, your program should only save the weights.*txt* file without ever opening it for reading. Once the weights have been calculated, keep them in memory for the similarity calculations. **Logarithms must be calculated in base 10.**

## The *answer.txt* file

The answer.*txt* file contains the answer to the query contained in the query file, in our example, *query.txt*. The first line of this file should contain the number of documents that satisfy the query. The other lines contain the files in the database that satisfy the query, with their respective degree of similarity, one per line, as in the following example, where we answer the query *"W & Y"*. Note that you will need to disregard *stopwords* present in the query and also extract the radicals o f its terms:

```
3
doc2.txt  0.9983
doc3.txt  0.2031
doc1.txt  0.1061
```

*answer.txt*

Consider that only documents with a similarity greater than or equal to 0.001 can answer the query, i.e. 0.001 is the minimum threshold for a document to be considered relevant. (**Don't forget that, along with each document in the answer, you need to represent its similarity**)

**Examples**

Take a look at the examples of real terms on the site. Don't forget to use the automatic corrector to check your code. Assuming your source file is called vector_model.*py*, just use it on the command line:

```
> python     waxm_vector_corrector.pycbase. txt query.txt vector_model.py
```

or

```
> py     waxm_vector_corrector.pyc      base.txtquery .txt vector_model.py
```