Université de Genève

Information Retrieval
14x011

TP 1 : Text Retrieval

Author: Sajaendra Thevamanoharan

E-mail: Sajaendra.Thevamanoharan@etu.unige.ch

Author: Deniz Sungurtekin

E-mail: Deniz.Sungurtekin@etu.unige.ch

Mars 2020

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

Introduction

In this exercise, we will try to understand how to find a discriminative set of *index terms* (keywords) for collection of documents. We will create two basic models in information retrieval namely, the *boolean* and *vector* models.

We have a set of documents : a collection of 141 short articles. We need to perform certain algorithm to prepare the collection of documents in order to analyse this set of documents.

NLTK

The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. NLTK includes graphical demonstrations and sample data.

Ref : //www.nltk.org

Information Retrieval

Raw data

In this first part, we load our set of documents (15 articles from the NASA corpus), as a text file into a variable using Python. We use also the NTLK librairies to tokenize the document. *Tokenisation* is about cutting a whole text into a *token* , often into words.

For each document of the corpus, we perform the tokenisation as follows :

```
Part of the output after tokenizing the emt01995.txt:

['Integration', 'of', 'Mechanical', 'Design', ',', 'Analysis', ',', 'and', 'Fabrication', 'Processes', 'Mechanical'
```

Stemming using Porter Algorithm

Once the documents are tokenized, we need to perform the stemming to extract the meaning of the document. A stem is a part of word to which affixes (suffixes and prefixes) can be attached to form a new words. Stemming is the proces of extracting for each given word its corresponding stem. Here we use the stemming algorithm of the library NLTK. (nltk.stem.porter).

We store the set obtained by the stemming in a variable, which is linked to the documents. We are going to perform the rest of our steps (tag clouds, term frequency and inverse documents frequency) in this output of stemming.

```
Part of the output after stemming the token obtained by tokenizing the emt01995.txt:

['integr', 'of', 'mechan', 'design', ',', 'analysi', ',', 'and', 'fabric', 'process', 'mechan'
```

Visualisation using tag clouds

Once we perfomed the stemming, we visualize the frequency of words using the tag clouds for 50 most frequent words. Tags will be represented by different font size based on the importance of each word. To compute the font size s_i for a word i, we will use the formula

$$s_i = f_{max} \cdot \frac{t_i - t_{min}}{t_{max} - t_{min}} \tag{1}$$

where f_{max} is the maximum font size, t_i is the word count, t_min is the minimum count, and t_{max} is the maximum count.

We have some tag clouds for some documents from the collection(NASA).

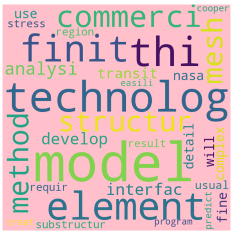


Figure 1: Tag Cloud of the article emt11895.txt

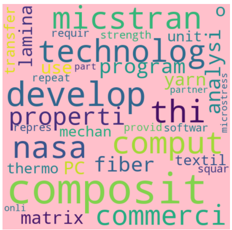


Figure 2: Tag Cloud of the article emt14395.txt

Term Frequency and Inverse Document Frequency

We have to measure two quantities from the text and the corpus to evualte the overall set. The *term frequency* tf_{ij} is the weight of a term t_i in a documents d_j computed according to

$$tf_{ij} = \frac{freq_{ij}}{max_k freq_{kj}} \tag{2}$$

where $freq_{ij}$ is the number of occurences of the term t_i in the document d_j.

We have also the *inverse document frequency* idf_i is a measure of the general importance of the term t_i in the whole corpus. It is computed according to

$$idf_i = log(\frac{N}{n_i}) \tag{3}$$

N, the total number of documents and n_i is the number of documents in which the term t_i appears.

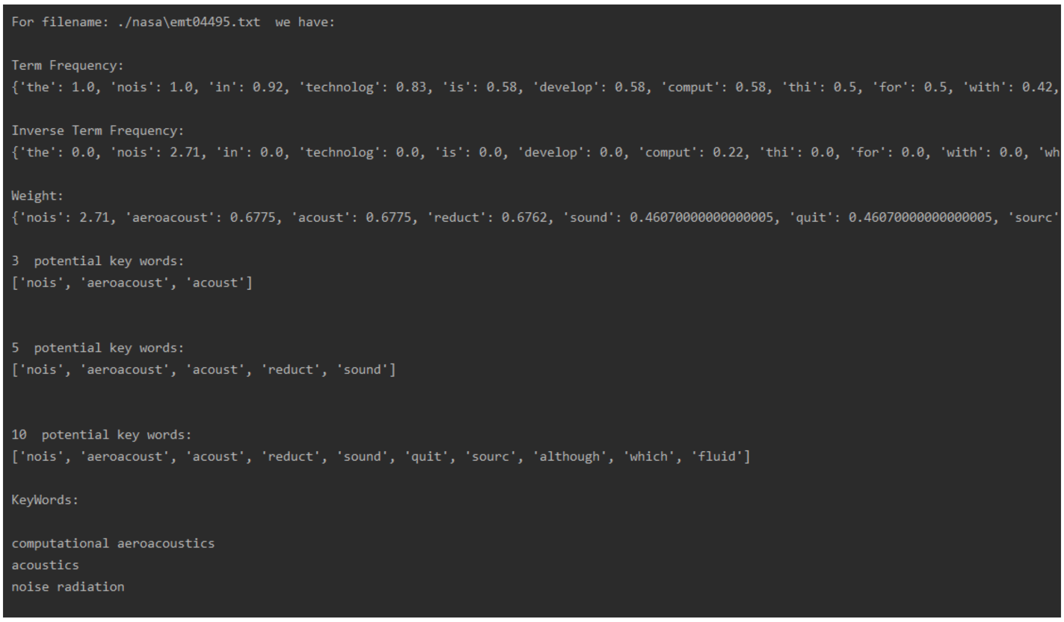
Using *the term frequency*tf_{ij} and the *inverse document*$frequency idf_i$ as described above, we obtain the term-weighting scheme in the text retrieval defined as :

$$w_{ij} = tf_{ij}.idf_i \tag{4}$$

Raw Data

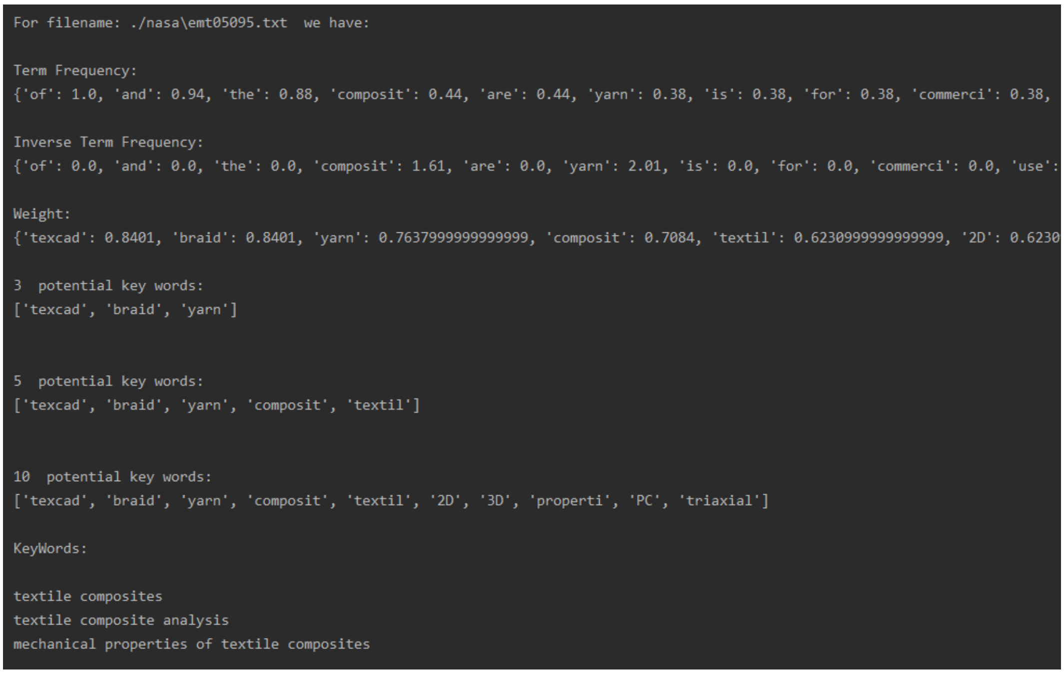
In this section, we take a corpus of 15 documents from the NASA article collection as a raw data (after the *tokensisation*). After the *stemming* step, we compute the *term frequency* tf_{ij} and *the term frequency* tf_{ij} and the w_{ij} for each document.

We obtain the following results for two of the articles from the corpus.



```
For filename: ./nasa\emt04495.txt  we have:

Term Frequency:
{'the': 1.0, 'nois': 1.0, 'in': 0.92, 'technolog': 0.83, 'is': 0.58, 'develop': 0.58, 'comput': 0.58, 'thi': 0.5, 'for': 0.5, 'with': 0.42,

Inverse Term Frequency:
{'the': 0.0, 'nois': 2.71, 'in': 0.0, 'technolog': 0.0, 'is': 0.0, 'develop': 0.0, 'comput': 0.22, 'thi': 0.0, 'for': 0.0, 'with': 0.0, 'wh

Weight:
{'nois': 2.71, 'aeroacoust': 0.6775, 'acoust': 0.6775, 'reduct': 0.6762, 'sound': 0.46070000000000005, 'quit': 0.46070000000000005, 'sourc'

3  potential key words:
['nois', 'aeroacoust', 'acoust']


5  potential key words:
['nois', 'aeroacoust', 'acoust', 'reduct', 'sound']


10  potential key words:
['nois', 'aeroacoust', 'acoust', 'reduct', 'sound', 'quit', 'sourc', 'although', 'which', 'fluid']

KeyWords:

computational aeroacoustics
acoustics
noise radiation
```

Figure 3: tf_{ij}, idf_i, w_{ij} for emt04495.txt



```
For filename: ./nasa\emt05095.txt  we have:

Term Frequency:
{'of': 1.0, 'and': 0.94, 'the': 0.88, 'composit': 0.44, 'are': 0.44, 'yarn': 0.38, 'is': 0.38, 'for': 0.38, 'commerci': 0.38,

Inverse Term Frequency:
{'of': 0.0, 'and': 0.0, 'the': 0.0, 'composit': 1.61, 'are': 0.0, 'yarn': 2.01, 'is': 0.0, 'for': 0.0, 'commerci': 0.0, 'use':

Weight:
{'texcad': 0.8401, 'braid': 0.8401, 'yarn': 0.7637999999999999, 'composit': 0.7084, 'textil': 0.6230999999999999, '2D': 0.6230

3  potential key words:
['texcad', 'braid', 'yarn']


5  potential key words:
['texcad', 'braid', 'yarn', 'composit', 'textil']


10  potential key words:
['texcad', 'braid', 'yarn', 'composit', 'textil', '2D', '3D', 'properti', 'PC', 'triaxial']

KeyWords:

textile composites
textile composite analysis
mechanical properties of textile composites
```

Figure 4: tf_{ij}, idf_i, w_{ij} for emt05095.txt

For the results obtained, we will analyse each of these quantites : the term frequency , the inverse term frequency, the weight.

For both of these articles, we can see that the term frequency gives us all words including the common words like 'the', 'of', etc. The inverse document frequency gives us the commonness of the words among the corpus the find only the discriminative words.

A high weight w_{ij} is reached by a high frequency of the term t_i in the document d_j and a low frequency of the term t_i in the whole collection of documents. Hence, the weights will tend to filter out common terms that appear in many documents in the collection.

We then take the top $p \in \{3, 5, 10\}$ stems according to the term frequency and the inverse document frequency and compare the extracted keywords among all documents. The keywords are given for each document in a *.key* file. We can see that when $p \in \{3, 5\}$, we don't get enough of matching of the p stems and the extracted key words. For the futur queries, we fixe $\mathbf{p = 10}$.

We can clearly say that, if we were based on only the term frequency or only the inverse document frequency, we would get either the basic words as 'the', 'of' as potential keywords, or the strict discriminative words of each document in the corpus.

We can conclude that the weight scheme ($w_{ij} = tf_{ij}.idf_i$) is a healthy measure for information retrieval.

Without Stop Words

Definition : Stop words are those words which are judged to be so common that they have very little information or discrimination power. A list of English stop words is given in the file **english.stop**.

We use the ntlk tokenization (element not in stopwords.words()) to get rid of these stop words. We perform now the tokenization algorithm and we have :

```
Without StopWord:  ['Integration', 'Mechanical', 'Design', ',', 'Analysis', ',', 'Fabrication', 'Processes', 'Mechanical',
```

Figure 5: Tokenization without stop words

To compare, we show the tokensization with the stop words.

```
 Part of the output after tokenizing the emt01995.txt:

['Integration', 'of', 'Mechanical', 'Design', ',', 'Analysis', ',', 'and', 'Fabrication', 'Processes', 'Mechanical'
```

We can notice that this time, it removed all the words which are judged to be so common that they have very little informationor discrimination power.

We now perform the stemming in these words obtained after the tokenization and we visualize them using tag clouds.

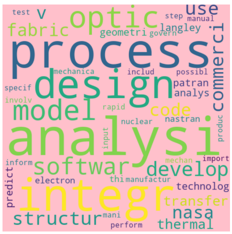


Figure 6: Tag Cloud of the article mt01995.txt and without stopwords

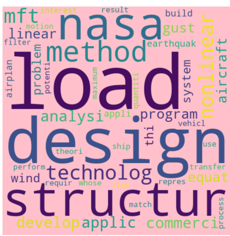
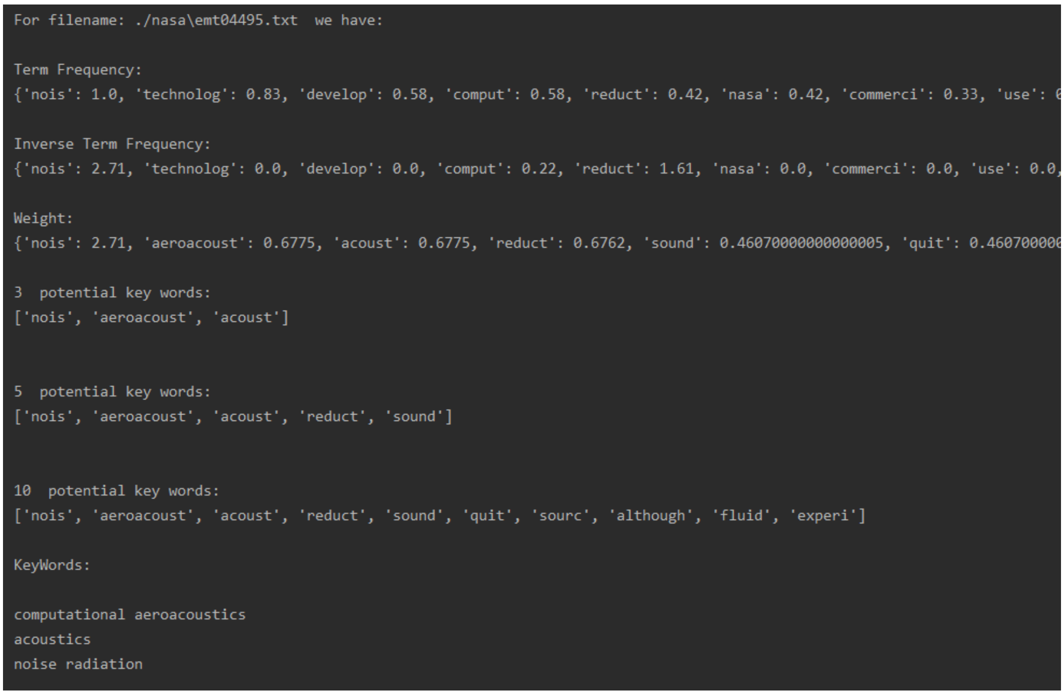


Figure 7: Tag Cloud of the article mt02495.txt and without stopwords

We can notice that we get more important indexes than the one with the stepwords.

In this section, we take a corpus of 15 documents from the NASA article collection as a raw data (after the *tokensisation*). After the *stemming* step, we compute the *term frequency* tf_{ij} and *the term frequency* tf_{ij} and the w_{ij} for each document without stop words. We have



```
For filename: ./nasa\emt04495.txt  we have:

Term Frequency:
{'nois': 1.0, 'technolog': 0.83, 'develop': 0.58, 'comput': 0.58, 'reduct': 0.42, 'nasa': 0.42, 'commerci': 0.33, 'use': 0

Inverse Term Frequency:
{'nois': 2.71, 'technolog': 0.0, 'develop': 0.0, 'comput': 0.22, 'reduct': 1.61, 'nasa': 0.0, 'commerci': 0.0, 'use': 0.0,

Weight:
{'nois': 2.71, 'aeroacoust': 0.6775, 'acoust': 0.6775, 'reduct': 0.6762, 'sound': 0.46070000000000005, 'quit': 0.46070000

3  potential key words:
['nois', 'aeroacoust', 'acoust']


5  potential key words:
['nois', 'aeroacoust', 'acoust', 'reduct', 'sound']


10  potential key words:
['nois', 'aeroacoust', 'acoust', 'reduct', 'sound', 'quit', 'sourc', 'although', 'fluid', 'experi']

KeyWords:

computational aeroacoustics
acoustics
noise radiation
```

Figure 8: tf_{ij}, idf_i, w_{ij} for emt04495.txt without stop words

We can notice that, even with a small **p**, we get closer to the extracted key words for this document. This is the case for the other documents too.

We will now construct a vector and boolean models for this corpus in order to give a ranking of the documents based on the similarity to a certain query.

Boolean and vector models

Before the explanation of our implementation of our boolean and vector models, its necessary to define the term document matrix which contain N row of size p containing the p most discriminative stemmed word for each document. Furthermore we define T being the corresponding vector containing all the words of the term document matrix. With those two element we can easily build our two following models:

The boolean model is a simple retrieval model where the query is made of stemmed words (element in T), our "compQueryBoolean" function simply count the number of times a stemmed word in the query appear in each document, this count will be the score of a document. Then this function return a descending list of document ordered by their score value.

As regards of the vector model, each query is represented by a **N*p** vector q where q_i is 1 if the query contain the word T_i and 0 if not. Furthermore, we also define each document as a **N*p** vector d where each dimension correspond to a term and its value is the term's weight. For example d_1 have his p first element set to the weight of his p term and the rest of the vector is set to 0 because it correspond to other document's weight. Then if we want to compute the corresponding result we need to build a similarity measure between our query vector q and our document vector d, this similarity is define as:

$$sim(\vec{d}, \vec{q}) = \frac{< \vec{d}, \vec{q} >}{||\vec{q}|| \cdot ||\vec{q}||} \tag{5}$$

This formula compute the cosinus of the angle between the two vector so for each document we compute this similarity and we return a list of document in a descending way depending on their similarity value.

For example, to visualize our result we took a boolean query made of the p most discriminative words of the emt01995.txt. Then we computed the corresponding vector query to observe the obtained result for the boolean and vector model:

```
Boolean Query:  ['optic', 'integr', 'fabric', 'code-v', 'patran', 'electron', 'analys', 'nastran', 'thermal', 'possibl']
Same Boolean Query as Vector Query:  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

Result from Boolean model:  [['./nasa\\emt01995.txt', 10], ['./nasa\\emt07895.txt', 1], ['./nasa\\emt10395.txt', 0], ['./nasa\\emt10195.txt', 0], ['./nasa\\emt07295.txt'
Result from Vector model:  [[['./nasa\\emt01995.txt', 0.8615843347999571], ['./nasa\\emt07895.txt', 0.07497681287813095], ['./nasa\\emt04895.txt', 0.053839973406613964],
```

Figure 9: A chosen example of boolean and vector query and the obtained result

We can see that we obtain the expected result because we have a score of 10 for the first document because we used his p = 10 most discriminative word to build our query. As regards of the vector model, we observe that the similarity between our query vector and our first document vector is much larger than any other similarity measure.

Raw data

Here we generate a random boolean query and its corresponding vector query:

```
Boolean Query:  ['low', 'polici', 'concept', 'sensit', 'nastran', 'maintain', 'SD']
Same Boolean Query as Vector Query:  [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

Result from Boolean model:  [['./nasa\\emt02695.txt', 3], ['./nasa\\emt07295.txt', 2], ['./nasa\\emt10195.txt', 1], ['./nasa\\emt01995.txt', 1], ['./nasa\\emt10395.txt', 0]
Result from Vector model:  [[['./nasa\\emt02695.txt', 0.3263631619477078], ['./nasa\\emt07295.txt', 0.31164205755251967], ['./nasa\\emt10195.txt', 0.09160404309361213], ['.
```

Figure 10: A random example of boolean and vector query and the obtained result

We can see that both queries give the same result almost with the same order.

Without Stop Words

Here we are doing the chosen query without including the stop words:

```
Boolean Query:  ['optic', 'integr', 'fabric', 'code-v', 'patran', 'electron', 'analys', 'nastran', 'thermal', 'possibl']
Same Boolean Query as Vector Query:  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

Result from Boolean model:  [['./nasa\\emt01995.txt', 10], ['./nasa\\emt07895.txt', 1], ['./nasa\\emt10395.txt', 0], ['./nasa\\emt10195.txt', 0],
Result from Vector model:  [[['./nasa\\emt01995.txt', 0.859128714833714], ['./nasa\\emt07895.txt', 0.07528940707024624], ['./nasa\\emt04895.txt',
```

Figure 11: A chosen example of boolean and vector query and the obtained result

The difference is only visible in the similarity value of the vector model because without the stop word there is a slight change in the weight value used to define our document vector.

Conclusion

We perform the stemming with and without stop words. We notice that the stop words does not influence too much the result if we take the weight as a discriminative measure. We can also notice that for a large value of p, the number of stems, we get closer to their extracted keywords. We can clearly see that the stop words doesn't influence in the interaction with the bollean and vectors models.

Work distribution

- Sungurtekin Deniz : Vector model implementation, tokenization, stemming, computation of frequency.
- Thevamanoharan Sajaendra : Boolean model implementation, tag clouds , computation of inverse frequency.
- Both : Report