Université de Genève

Information Retrieval
14x011

TP 2 : BIR and LSI

Author: Sajaendra Thevamanoharan

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

Author: Deniz Sungurtekin

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

28 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 *Probabilistic model* and *Latent semantic indexing model* 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

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

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).

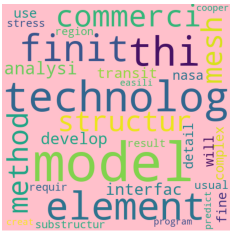

Figure 1: Tag Cloud of the article emt11895.txt

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.

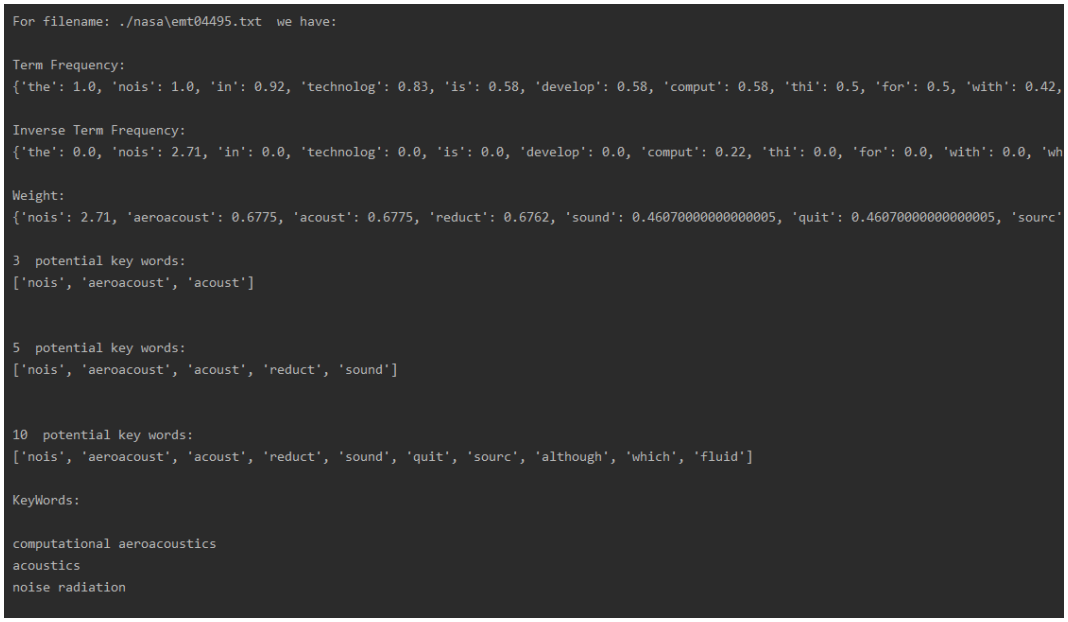

Figure 3: tf_{ij}, idf_i, w_{ij} for emt04495.txt

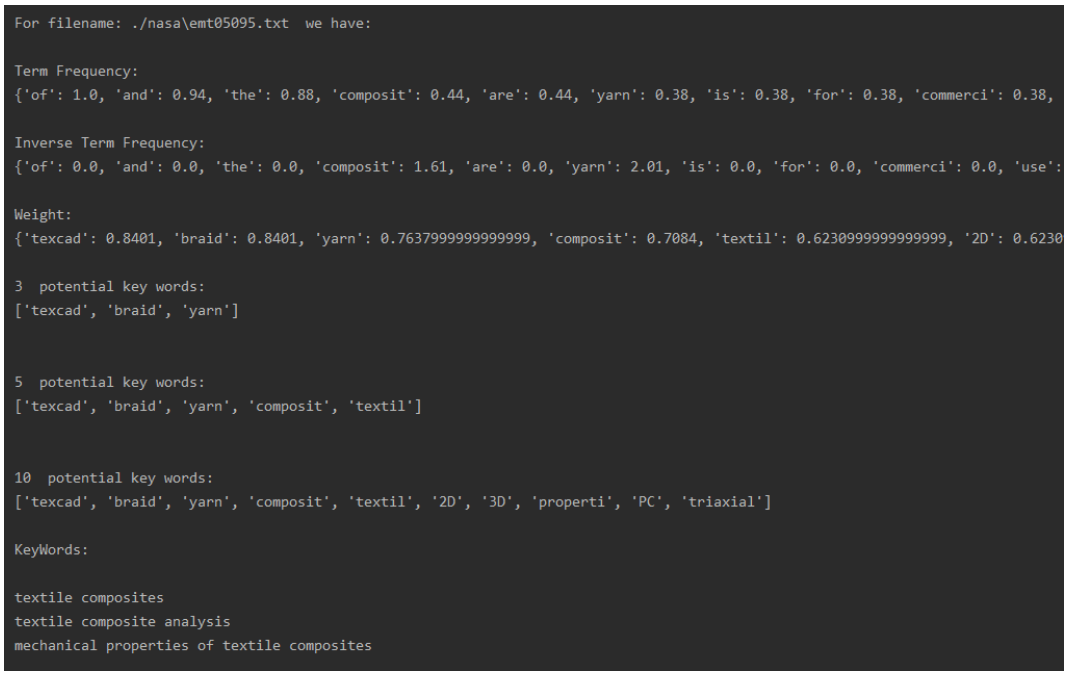

Figure 4: tf_{ij}, idf_i, w_{ij} for emt05095.txt

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.

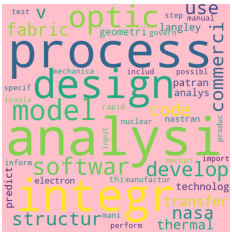

Figure 6: Tag Cloud of the article mt01995.txt and without stopwords

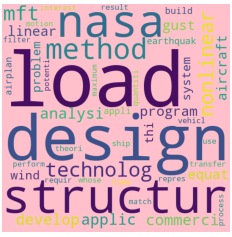

Figure 7: Tag Cloud of the article mt02495.txt and without stopwords

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

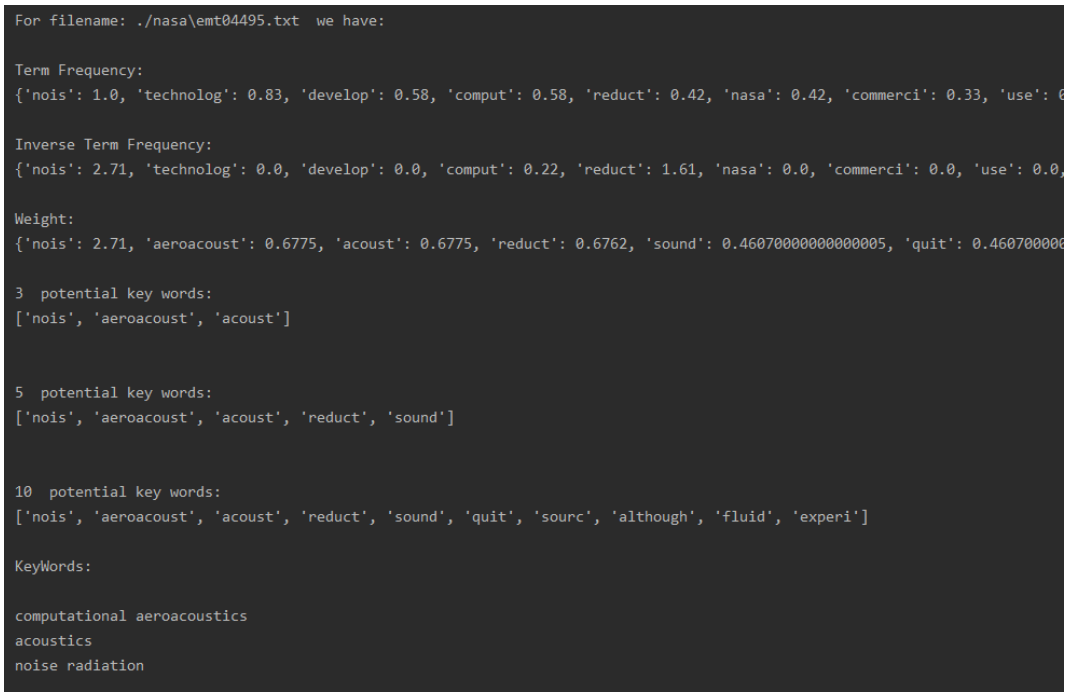

Figure 8: tf_{ij}, idf_i, w_{ij} for emt04495.txt without stop words

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 *probabilistic* and *latent semantic indexing* models for this corpus in order to give a ranking of the documents based on the similarity to a certain query.

Probabilistic model

We consider a user query and a set of documents that contains only relavant document for this query. We attempt to estimate the probability that the user will find the document d_j as interesting. Let R be the set of initially good guessed documents to be relevant. Let $\bar{R}$ to be the complement of set R. The similarity measure $sim(d_j, q)$ between the document and the d_j and the query q is defined as the ratio

$$sim(d_j, q) = \frac{P(R|d_j)}{P(\bar{R}|d_j)} \tag{5}$$

with $P(R|d_j)$, the probability that the document d_j is relevant to the query q and $P(\bar{R}|d_j)$, the probability that the document d_j is non-relevant to the query q. Assuming independance of *index terms* and taking logarithms, we can finally write :

$$sim(d_j, q) = \sum_{i=1}^{k} w_{iq}.w_{ij} \left(\log \frac{P(t_i|R)}{1 - P(t_i|R)} + \log \frac{P(t_i|\bar{R})}{1 - P(t_i|\bar{R})} \right) \tag{6}$$

As we dont know the probabilities and the set R and $\bar{R}$, we assume the following initial values.

$$P(t_i|R) = 0.5, P(t_i|\bar{R}) = \frac{n_i}{N} \tag{7}$$

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

We can update these rules as follows

$$P(t_i|R) = \frac{|V_i|}{|V|}; P(t_i|\bar{R}) = \frac{n_i - |V_i|}{N - |V|} \tag{8}$$

$$P(t_i|R) = \frac{|V_i| + 0.5}{|V| + 1}; P(t_i|\bar{R}) = \frac{n_i - |V_i| + 0.5}{N - |V| + 1} \tag{9}$$

$$P(t_i|R) = \frac{|V_i| + \frac{n_i}{N}}{|V| + 1}; P(t_i|\bar{R}) = \frac{n_i - |V_i| + \frac{n_i}{N}}{N - |V| + 1} \tag{10}$$

with $|V|$, be a subset of the documents initially retrieved and ranked by the probabilitic model. Let $|V_i|$ be the subset of V composed of the documents in V which contain the *index term* t_i.

Results

Raw Data

We generate a random query q. We consider our *Term-document matrix* as described before. (cf. def TermDocumentMatrixProbModel(...))

```
First elements of the query:  [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
Result:
File  0  with score  13.846702104582276
File  8  with score  9.776624299653696
File  6  with score  9.366199621038367
File  5  with score  8.345427351341158
File  11  with score  8.03972931698755
File  4  with score  7.2124958175552765
File  10  with score  4.6611426025702976
File  13  with score  3.913802974920665
File  12  with score  3.7316049765294323
File  9  with score  3.2976974211190333
File  7  with score  3.0920983024921167
File  14  with score  2.808094269044016
File  2  with score  2.34007855753668
File  3  with score  1.9959747548878362
File  1  with score  0.0
```

Figure 9: Results with stop words

This result is obtained by using the second updating rules because with the first one the probabilities are too small to compute the logarithm used in equation 6. Moreover, we have the same issue with the third one because adding $\frac{n_i}{N}$ is not enough to solve the problem.

Without StopWords

We generate a random query q. We consider our *Term-document matrix* as described before. (cf. def TermDocumentMatrixProbModel(...))

```
First elements of the query:  [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
Result:
File  0  with score  18.62264270317566
File  11  with score  17.198201651398573
File  5  with score  15.159354845625272
File  6  with score  13.764887262613026
File  13  with score  13.01817874002615
File  8  with score  10.832163926008285
File  2  with score  8.599100825699287
File  10  with score  7.455438474454501
File  4  with score  7.24258631045716
File  7  with score  6.733288297405787
File  9  with score  4.31748811353631
File  12  with score  4.31748811353631
File  3  with score  3.932368173808871
File  14  with score  3.627121764181854
File  1  with score  0.0
```

Figure 10: Results without stop words

For the same query, we obtain a different result without stop words. We can clearly see that the score of the first document is much bigger which is good because the query contain all his p words. Some stop words can really impact the search because they do have a good frequency without necessarily being present in others document which is a problem because if their inverse frequency is not big enough, they will be taken as one of the p selected words of the document which will have big influence on the result.

Latent semantic indexing model

This approach maps each document and query vector into a lower dimensional space which is associated with concepts. Let k be the number of index terms in the collection of documents and N be the total number of documents. We define M as a matrix with k rows and N columns. Each element M_{ij} of this matrix is assigned a weight w_{ij} associated with the index term t_i and the document d_j.

Latent semantic indexing model proposes to decompose the *term-document* matrix using SVD as follow

$$M = S.\Delta.D^T \tag{11}$$

S,D are the matrix of eigenvectors obtained respectively from $M.M^T$ and $M^T.M$. Δ is a $r \times r$ diagonal matrix of singular values where $r = min(k, N)$ is the rank of M.

We consider now only the l largest singular values of Δ and keep with hteir corresponding columns in S and D, respectivly. The result is matrix M_l given by

$$M_l = S_l.\Delta_l.D_l^T \tag{12}$$

The relation between any two documents in the reduced space of dimensionnality l can be derived from matrix $M_l.M_l^T$ where element (i,j) quantifies the relationship between documents d_i and d_j.

Results

We take the first row of the matrix which will correspond to a boolean query as we did in the probabilistic model. We generate a query which will find the first document as the most relevant. We plot score wise the corresponding documents.

Raw data

```
Result:
File  12  with score  2.7563634795179466
File  8  with score  2.58093148502261
File  9  with score  2.510936832683253
File  4  with score  2.4253164786148633
File  6  with score  2.337358830523316
File  0  with score  2.3047606197127246
File  14  with score  2.0307731637251325
File  11  with score  1.517481843144812
File  3  with score  1.4310619765521442
File  5  with score  1.4079777697245734
File  10  with score  1.3948689583896463
File  1  with score  1.3860933311450048
File  7  with score  1.360794984739237
File  2  with score  1.167544637938398
File  13  with score  1.1268173601694336
```

Figure 11: LSI with StopWords

Without StopWords

```
Result:
File  12  with score  2.778179090999753
File  8  with score  2.5755323540231365
File  9  with score  2.5446218173016018
File  4  with score  2.415369477294094
File  6  with score  2.323934083555419
File  0  with score  2.2751409059485894
File  14  with score  2.0534139257615776
File  11  with score  1.5333712402703774
File  3  with score  1.4415169914521033
File  7  with score  1.387534276129956
File  1  with score  1.3873847568093245
File  5  with score  1.3815918676468635
File  10  with score  1.3506241891116237
File  2  with score  1.167969853216841
File  13  with score  1.0911595518194337
```

Figure 12: LSI with StopWords

We can see that concept wise, we have different results from the probabilistic model in both cases. We can also notice that the document 0 is very far in the list. It means that the document is not having the same concept ideas than itself. It seems to be like very strange to have these results. This difference may come from the fact that we are here considering not the frequency of the model, but the concept of these documents.

Conclusion and Remarks

In conclusion, we have seen that it is important to be careful with the probabilistic model because the stop words can have a big impact on the result, moreover the way we update our probabilities can generate some issues in practice. In regards to Latent semantic indexing model, we had different results because we considered a concept wise model instead of a probabilistic one. In this method, it is very easy to compute and see the correlation between all the documents to quickly see the similarities between them. Finally, this method shows efficiency toward stop words, we clearly see that both results are very similar.