Lab 2

Programming with ElasticSearch

Bartomeu Perelló Comas
David Carballo Montalbán

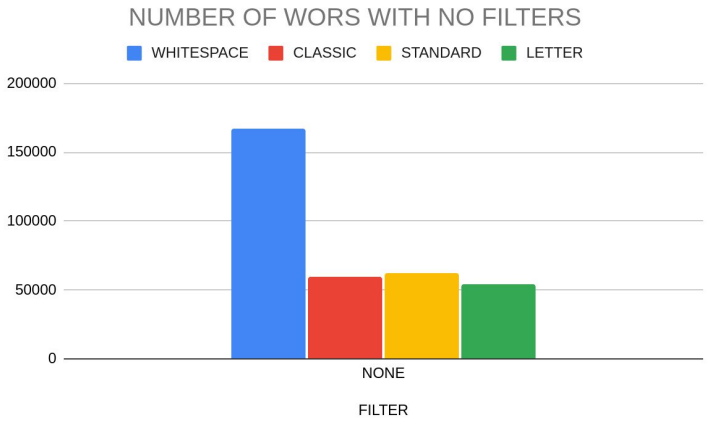
22/10/2020

CAIM - GEI FIB

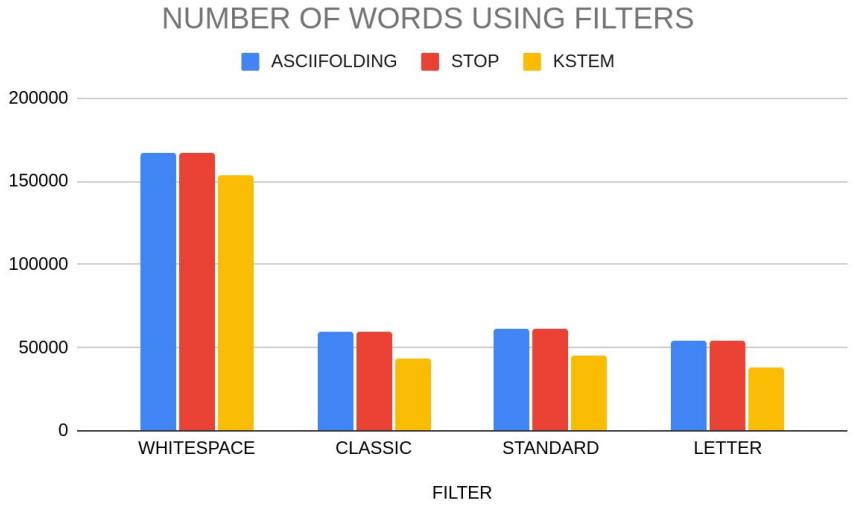
Index behavior

In this first section of the session we are asked to use different tokens in the novel collection and compare the results between them.

If we only compare the use of the tokenizer, it's not surprising that the *whitespace* one gives us the most number of total words, because everytime it encounters something between two whitespaces it is considered as a word, afterwards we have the *standard* which uses the Unicode Text Segmentation algorithm, finally the *classic* and the *letter* are pretty close in the number of different words they keep being *letter* the most restrictive one.



Finally we tried different filters to see how much they would affect the number of words, *asciifolding* did pretty much nothing, until we use the *kstem* filter there are no noticeable differences in the amount of words we end up with.



For example, the *stop* filter only removes the most common stop words on a specific language, so as we learned in the previous season, stopwords are just a few which have higher frequencies, this filter should be used when trying to check if zipf's law is true on a text because it removes the noise from those words.

Computing tf-idf ’s and cosine similarity

On the other hand, we had to complete the TFIDFViewer.py script which uses an index of the database and the path of the documents we want to compare that are related to the index.

The first function to complete was *toTFIDF*, we had calculate the term frequency and the inverse document frequency to obtain the weight of each term on the document following the following steps:

Each weight is a product of two terms

$$w_{d,i} = tf_{d,i} \cdot idf_i.$$

The term frequency term tf is

$$tf_{d,i} = \frac{f_{d,i}}{\max_j f_{d,j}}, \qquad \text{where } f_{d,j} \text{ is the frequency of } t_j \text{ in } d.$$

And the inverse document frequency idf is

$$idf_i = \log_2 \frac{D}{df_i}, \qquad \text{where } D = \text{number of documents}$$

and df_i = number of documents that contain term t_i.

Afterwards we had to complete the *normalize* function, which transforms each weight of the vector into another one such that the new norm of the vector has value 1 so then it’s easier to compare to other vectors.

Cosine_similarity function calculates the scalar product between two weight vectors. And finally, the *print_term_weight_vector* function which printed the vector.

Down below, there are the tests we made with the script to calculate the similarity between different 20_newsgroups documents.

Documents of different groups

Similarity (alt.atheism/0000001 – sci.space/0014782) = 0,02199
Similarity (alt.atheism/0000011 – sci.space/0014094) = 0,01142
Similarity (alt.atheism/0000028 – sci.space/0014464) = 0,00242
Similarity (alt.atheism/0000620 – sci.space/0014782) = 0,00879
Similarity (alt.atheism/0000620 – sci.space/0014464) = 0,02132
Similarity (alt.atheism/0000028 – sci.space/0014782) = 0,01011

Documents of sci-space group:

Similarity (0014000 – 0014001) = 0,01557
Similarity (0014000 – 0014094) = 0,02639
Similarity (0014001 – 0014002) = 0,09955
Similarity (0014001 – 0014094) = 0,07624

Documents of alt.atheism group:

Similarity (0000001 – 0000000) = 0,25966
Similarity (0000001 – 0000620) = 0,17102
Similarity (0000011 – 0000620) = 0,03206
Similarity (0000028 – 0000620) = 0,02974

As we can appreciate from the results above, between files in the same theme have higher similarity than documents from others. For example, in the case of alt.atheism group, we obtained a similarity of up to 0,2, but when the comparison was done with documents from different groups the highest similarity was 0.021

This being said, we cannot confirm that the similarity between two different groups will always be inferior than the similarity between documents in the same one.