UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

INFORMATION RETRIEVAL AND RECOMMENDER SYSTEMS

Lab 2: Programming in ES

Mohana Fathollahi, Tanja Bien

11th October, 2022

1 Modifying index behaviour

In the following sections we investigate the influence of the tokenizer and filter options. The implementation can be found in the index_behaviour.ipynb.

1.1 Influence of tokenization

At first we investigated the influence of the tokenizer. We always used the same filter (lowercase) and the following tokenizers:

- Whitespace: divides text into terms whenever it encounters any whitespace character.
- Classic: grammar based tokenizer for the English Language.
- Standard: divides text into terms on word boundaries, as defined by the Unicode Text Segmentation algorithm and removes most punctuation symbol.
- Letter: divides text into terms whenever it encounters a character which is not a letter.

As can be seen in 1, the tokenizer has a large impact on the number of tokens: the whitespace tokenizer returns more than twice as many words as the other tokenizers. This is to be expected since whitespace uses a fairly simple rule for tokenization. Also, no punctuation is removed. This becomes especially clear when looking at the types that are included in the whitespace index but not in the standard index, e.g. "see?–that", "plagiarists." or "(smith)". Thus, many types that are directly adjacent to a punctuation symbol are included as a separate type. On the other hand, there are types that exist only with punctuation, for example, the classical index contains the type "spiritus", while the space indexes contains only "spiritus.". This example illustrates the problem that arises from using such a simple rule.

Table 1: Influence of the tokenizer on the word types and most frequent words of the novel corpus

tokenizer	**filter**	**n of types**	**most frequent word**
whitespace	lowercase	167,063	the(203231)
classic	lowercase	59,569	the(206706)
standard	lowercase	61,825	the(206546)
letter	lowercase	54,455	the(206706)

A limitation of the letter tokenizer is the handling of words containing non-alpha characters, e.g. the word "wouldn't". Only the word "wouldn" can be found in the letter index. We can also notice differences between the advanced tokenizer classic and the standard. One big difference is the keeping of the character "_". But in general both tokenizers are able to accept words containing non-alphabetic characters like "wouldn't" or "your@login".

The most common terms (when applying the lowercase filter only and using the letter tokenizer) vary for the datasets. The most common terms for the novel dataset are "the", "of", "and" (in that order), while for the news dataset they are "the", "to", "of", and for the

Arxiv dataset they are "the", "of", "and". As expected, the most frequent terms are stop words.

We did not evaluate other tokenizers like "lowercase" or "uax_url_email" in this report, but included them in our implementation.

As a final note on tokenization, we want to remark that the path in the IndexFilesPreprocess.py file has been specially configured as a keyword. For this reason, it is excluded from tokenization and we can search for documents by pathname.

1.2 Influence of filter

In the second step we investigated the following filters:

1. lowercase: changes token text to lowercase
2. asciifolding: converts alphabetic, numeric, and symbolic characters that are not in the Basic Latin Unicode block (first 127 ASCII characters) to their ASCII equivalent
3. stop: removes stop words
4. snowball: filter that stems words using a Snowball-generated stemmer

The influence of the filters on the number of types and the most frequent word can be seen in table 2. We notice that after applying all the filters, only about a third of the original number of types remains. We understand this better if we recheck which terms are in one index but not in the other. The lowercase-only index contains words like "noshemã" that were converted to "noshema" using the ascifolding filter. The stop filter removed 33 stop words such as "a", "the", or "but". The most radical impact on the number of types was made by the stemer. We have examined the various stemmers, but will use the snowball steamer as a representative in this report. While this index contains terms like "snowi", the stop index contains the unstemmed workds like "snowing" and "snowiest". The stop filter also has an influence on the most frequent term, because without it "the" is the most frequent term, but because it is a stop word it is filtered and "i" is the new most frequent term.

Table 2: Influence of the filter on the word types and most frequent words of the novel corpus

filter	**tokenizer**	**n of types**	**most frequent word**
lowercase	letter	54,455	the(206706)
lowercase, asciifolding	letter	40,296	the(126390)
lowercase, asciifolding, stop	letter	40,263	i(29687)
lowercase, asciifolding, stop, snowball	letter	20,918	i(39348)

After removing the stop words, the most common terms for the records have changed. For Arxiv, these are now "we," "of," and "which," and terms such as "model" and "data" are also included in the top 10. These words are consistent with expectations regarding the scientific nature of the text. For the novel dataset, the most common terms are "i", "its", "he", while for the news dataset they are "i", "you" and "have".

2 TF-IDF

Based on the formula that we have for "tfidf", weight for each token has been calculated with respect to code in below.

2.1 toTFIDF function

```
    for (t, fd),(_, df) in zip(file_tv, file_df):
        #1) compute tf(d,i). Use f(d,i) = file_td[t] = fd and max = max_freq
        tf = fd / max_freq
        #2) compute idf(i). log2(D/df(i)). D=dcount. df(i) = file_df[t]=df)
        idf = np.log2(dcount / df)
        w = tf * idf
        tfidfw.append((t, w))
    return normalize(tfidfw)
```

In the next step we applied normalization based on sequence root of the sum of the component squared. Then we calculated cosine similarity based on normalized weights that we have from previous step. Codes related to this part exist in "TFIDFViewr.py" and it passed tests and we did not have a problem on that.

3 Experimenting

All experiments are implemented in the comparison_btw_within_groups.ipynb.

Table 3: Similarity of different documents

File 1	**File 2**	**Similarity**	**Comment**
arxiv math 0	arxiv math 1	0.01441	same domain
arxiv cs 0	arxiv cs 1	0.02587	same domain
arxiv physics 0	arxiv physics 1	0.05931	same domain
arxiv physics 0	arxiv cs 1	0.02292	different domain
arxiv math 0	arxiv cs 1	0.06139	different domain
DickensAChristmasCarol	DickensGreatExpectations	0.01716	same author
PoeWorksVol1	PoeWorksVol2	0.24176	same author
DickensAChristmasCarol	LondonCallofTheWild	0.00456	different author
DickensAChristmasCarol	DarwinOriginofSpecies	0.00277	different author
alt.atheism 0	comp ms-windows 2000	0.00676	different subset
comp ms-windows 2000	comp ms-windows 2001	0.00152	same subset
comp ms-windows 2000	comp ms-windows 2010	0.00626	same subset
autos 6050	motorcycles 8000	0.00086	similar subset
politics guns 11926	politics guns 11939	0.03807	same subset
politics guns 11926	politics mideast 16011	0.02426	similar subset

We performed some experiments and compared the similarity of different documents, as shown in table 3. First, we started with the arxiv data set and compared documents from the same domain (mathematics, physics or computer science). We found similarities between 0.01 and 0.06. Next we started comparing documents from different domains (physics vs. computer science or mathematics vs. computer science) and still observed similar similarities (0.02 and 0.06). Although the documents are from different fields, the fields themselves are not really different, but there are many similarities between computer science, mathematics and physics, which explains the results. The style and vocabulary of the text is also very similar due to its scientific nature.

In the next steps, we used the novel data set and compared novels by the same author and novels by different authors. As expected, we obtain higher similarities for the same authors (0.25 and 0.2) compared to different authors (0.004, 0.003). In particular, the works of Edgar Allan Poe are very similar, which we explain by the fact that he wrote many books of the same genre (detective novels). The similarity between the different authors is very low, which is probably due to the different narrative styles, topics and genres.

Finally, we also took a look at the newsgroup corpus. These texts are special because they do not have a fixed form like scientific texts or novels and also might be very short. We compared different texts from the same computer sub-corpus and also compared them to a completely different sub-corpus (alt.atheism).

In all cases, we encountered quite low similarities. When we investigated the texts, we learned that they are very specialized and also quite short. Therefore, the vocabulary is very different and the similarity is accordingly low (0.001 - 0.007). The same is true for the documents on cars and motorcycles. For this reason, we looked at the politics subset. Since people write more here and the vocabulary used is not as specialized, we expected higher similarities. The results were consistent with this expectation and also with the assumption that documents from the same topic area (e.g., weapons) are more similar than those from different subsets (weapons vs. Middle East).

All experiments have the limitation that we used only one pair of randomly selected documents. It has given us more intuition, but to get more meaningful results, we have to experiment with more sample of pairs. So in the next step, we investigated the newsgroup using more pairs for more meaningful results.

In each group of messages, five documents were randomly selected and the similarities between each group and another group were calculated and the mean of the five results for each group was obtained. We encountered the maximum similarity in "comp.sys.ibm.pc.hardware" with 0.062391, suggesting that this group is more homogeneous than the other groups. The group "sci.electronic" has the lowest similarity or homogeneity, which is 0.006514. Moreover, after applying the similarity between the groups, we find the highest similarity between: "talk.politics.guns" and "talk.politics.misc" with 0.047720, which is in line with our expectations since both topics are from the political domain. On the other hand, we have the lowest similarity between "comp.windows.x" and "rec.autos" with 0.002747. This is also consistent with our previous results. All results, including the ordered table of the within and between similarities can be found in the notebook.