tdt4117_3

Pål Larsen (paaledwl) og Johannes Kvamme (johannkv)

1 | Data loading and preprocessing

1.0 Fix random numbers generator

This is done at the top of the run_me.py file

1.1 Open and load the file (it’s UTF-8 encoded) using codecs.

This is done at the very start in the main function

1.2. Partition file into separate paragraphs. Paragraphs are text chunks separated by empty line. Partitioning result should be a list of paragraphs. For us, each paragraph will be a separate document

This is done in the parseBookToList function. It iterates through each line in the files. If the line is empty it understands that it is a new paragraph and will finish that paragraph and append it to the book which is the main list.

1.3 Remove (filter out) paragraphs containing the word “Gutenberg” (=headers and footers).

This is done with the function removeParagraphsContaining which takes in a paragraph and returns a counter value of how many times the 'bad words' has appeared. If it is more than 0 it does not include it in the book. The 'bad words' are initialized at the start of the file as a list, letting more words be easily removed.

1.4. Tokenize paragraphs (split them into words). Now we should have a list of paragraphs where each paragraph is a list of words.

This is done in the parseBookToList function which splits each line into a list, which are appended to the current paragraph.

1.5. Remember to remove from the text punctuation (see string.punctuation) and whitecharacters ("\n\r\t"). Convert everything to lower-case.

This is first done in parseBookToList which goes through every word in a line. The word is then put through the function iterateWord which also calls the function removeWhiteCharacters which removes any of those characters in a word.

1.6. Using PorterStemmer stem words

This is done in the same way as 1.5, but iterateWord first sets the word to lower case then calls stemming which returns the word, but stemmed

2 | Dictionary building

2.1. Build a dictionary (a structure mapping words into integers).

This is done in the main function, after calling parseBookToList, which takes the new, processed book.

2.2 Filter out stopwords using the list from https://www.textfixer.com/tutorials/common-english-words.txt

This is done in the main function. First creating the stopWordsFile (which is the one downloaded from above), then calling the function getStopIds, which takes the file and book as parameters. The function just returns a list of those words. Then calling `dictionary.filter_tokens(stop_ids)` as well as `dictionary.compactify()` which removes the words from the dictionary.

2.3 Map paragraphs into Bags-of-Words using the dictionary.

This is done after 2.3 in main which calls the function getBagOfWords which takes the dictionary and book as paramters, returning the corpus. The function iterates every paragraph in the book which adds each paragraph (which is a list) into the dictionary's doc2bow function.

3 | Retrieval models

3.1 Build TF-IDF model using corpus (list of paragraphs) from the previous part.

This is done in the main function after #2.3 (setting the corpus). `tfidf_model = models.TfidfModel(corpus)`

3.2 Map Bags-of-Words into TF-IDF weights (now each paragraph should be represented with a list of pairs (word-index, word-weight)).

This is done after #3.1 with the code `tfidf_corpus = tfidf_model[corpus]`

3.3

This is done after #3.2 with the code `tfidf_index = similarities.MatrixSimilarity(tfidf_corpus)`

3.4 Repeat the above procedure for LSI model using as an input the corpus

with TF-IDF weights. Set number of topics to 100. In the end, each paragraph should be represented with a list of 100 pairs (topic-index, LSI-topic-weight)).

This is done after #3.3 in the main function. The index is created with the 3 lines

```
lsi_model = models.LsiModel(tfidf_corpus, id2word=dictionary, num_topics=100)
lsi_corpus = lsi_model[tfidf_corpus]
lsi_index = similarities.MatrixSimilarity(lsi_corpus)
```

4 | Querying

4.1 For the following query: "What is the function of money?"apply all necessary transformations: remove punctuations, tokenize, stem and convert to BOW representation in a way similar as in Part1.

The querying is done in the main function, in the first lines in the while loop. With the code

```
userQuery = input("User query: ")
        processedQuery = []
        if userQuery == 'break':
            break
        queryList = userQuery.split()
        for word in queryList:
            processedQuery.append(iterateWord(word))
        queryCorpus = dictionary.doc2bow(processedQuery)
```

First it takes in the user input as 'userQuery' and splits it into a list 'queryList'. Then it iterates through every word with the function iterateWord which removes unwanted parts of a word. At the end it create a 'queryCorpus' with the `dictionary.doc2bow(processedQuery)`

4.2 Convert BOW to TF-IDF representation. Report TF-IDF weights. For example, for the query "How taxes influence Economics?" TF-IDF weights are:

this is done after #4.1 in the while loop with the code `tfidf_query = tfidf_model[queryCorpus]` returning the index of the tax, influence and economics as well as their values.

4.3 Report top 3 the most relevant paragraphs for the query "What is the function of money?" according to TF-IDF model (displayed paragraphs should be in the original form –before processing, but truncated up to first 5 lines). For example for the query "How taxes influence Economics?"

This is done after #4.2 with the code

```
print("\nTFIDF-Model query")
tfidf_sims = tfidf_index[tfidf_query]
```

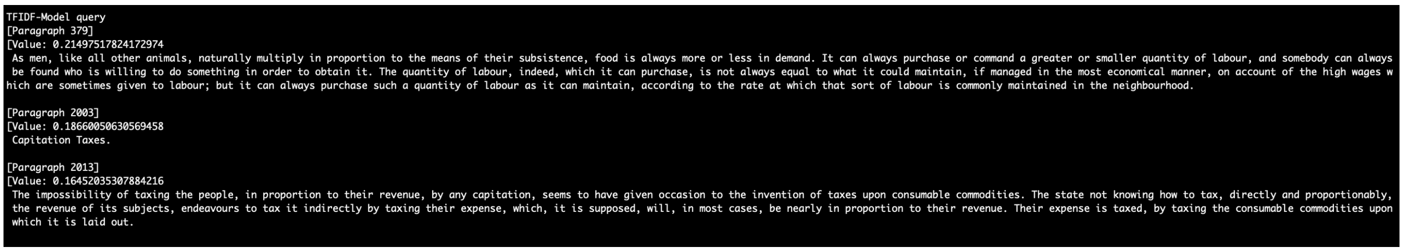
```
iterateSims(tfidf_sims, untouchedBook)
```

where iterateSims goes through the top 3 values in the similarities and calls printParagraph for each of the top 3 values. Which prints the paragraph in the wanted form.

4.4

The values returned from the LSI indexing are sometimes different, I assume it's from the random seeding.

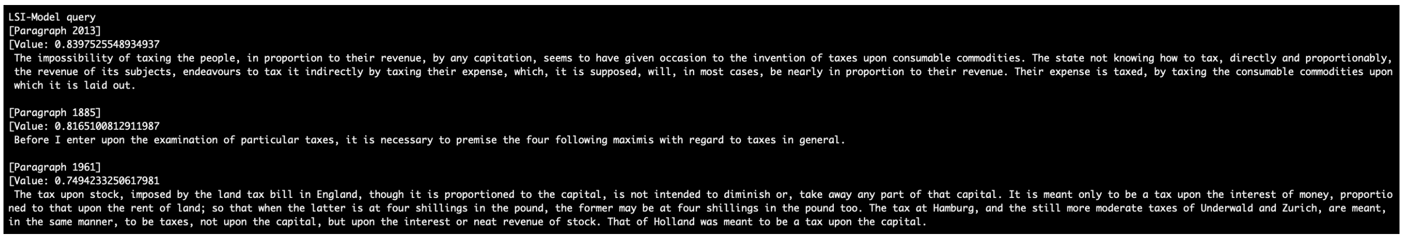
LSI will/should return the more relevant files since it implements semantic closeness of the words. They both return paragraph 2013, but the others are different. Paragraph 379 also has nothing to do with taxes and paragraph 2003 has two words in it, one being only taxes. While the paragraphs 1885 and 1961 talks abouts taxes in greater length.



```
TFIDF-Model query
[Paragraph 379]
[Value: 0.21497517824172974
 As men, like all other animals, naturally multiply in proportion to the means of their subsistence, food is always more or less in demand. It can always purchase or command a greater or smaller quantity of labour, and somebody can always
 be found who is willing to do something in order to obtain it. The quantity of labour, indeed, which it can purchase, is not always equal to what it could maintain, if managed in the most economical manner, on account of the high wages w
hich are sometimes given to labour; but it can always purchase such a quantity of labour as it can maintain, according to the rate at which that sort of labour is commonly maintained in the neighbourhood.

[Paragraph 2003]
[Value: 0.18660050630569458
 Capitation Taxes.

[Paragraph 2013]
[Value: 0.16452035307884216
 The impossibility of taxing the people, in proportion to their revenue, by any capitation, seems to have given occasion to the invention of taxes upon consumable commodities. The state not knowing how to tax, directly and proportionably,
 the revenue of its subjects, endeavours to tax it indirectly by taxing their expense, which, it is supposed, will, in most cases, be nearly in proportion to their revenue. Their expense is taxed, by taxing the consumable commodities upon
 which it is laid out.
```



```
LSI-Model query
[Paragraph 2013]
[Value: 0.8397525548934937
 The impossibility of taxing the people, in proportion to their revenue, by any capitation, seems to have given occasion to the invention of taxes upon consumable commodities. The state not knowing how to tax, directly and proportionably,
 the revenue of its subjects, endeavours to tax it indirectly by taxing their expense, which, it is supposed, will, in most cases, be nearly in proportion to their revenue. Their expense is taxed, by taxing the consumable commodities upon
 which it is laid out.

[Paragraph 1885]
[Value: 0.8165100812911987
 Before I enter upon the examination of particular taxes, it is necessary to premise the four following maximis with regard to taxes in general.

[Paragraph 1961]
[Value: 0.7494233250617981
 The tax upon stock, imposed by the land tax bill in England, though it is proportioned to the capital, is not intended to diminish or, take away any part of that capital. It is meant only to be a tax upon the interest of money, proportio
ned to that upon the rent of land; so that when the latter is at four shillings in the pound, the former may be at four shillings in the pound too. The tax at Hamburg, and the still more moderate taxes of Underwald and Zurich, are meant,
in the same manner, to be taxes, not upon the capital, but upon the interest or neat revenue of stock. That of Holland was meant to be a tax upon the capital.
```