TDT4117 Information Retrieval - Autumn 2022

Assignment 3 – genism

Deadline for delivery on Blackboard is 20.10.2022

Introduction

Gensim is a Python library for *topic modelling*, *document indexing* and *similarity retrieval*. Target audience is the *natural language processing* (NLP) and *information retrieval* (IR) community. [https://pypi.python.org/pypi/gensim]

NLTK is a leading platform for building Python programs to work with human language data. It provides processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, and wrappers for industrial-strength NLP libraries. [http://www.nltk.org/]

Installation

The advised working environment for this assignment is the latest version of python (for example appropriate anaconda distribution from https://www.anaconda.com/download/). Necessary python modules can be installed using pip:

```
$ pip install gensim
```

```
$ pip install nltk
```

Data set

In the assignment we will index and query content (paragraphs) of the book: 'An Inquiry into the Nature and Causes of the Wealth of Nations' by Adam Smith. Text file containing the book should be downloaded from http://www.gutenberg.org/cache/epub/3300/pg3300.txt

Task

The task is to index and query book paragraphs. For example: a query "*How taxes influence Economics?*" should (among the others intermediate results) should result in printing 3 the most relevant paragraphs (up to 5 lines each) from the book according to LSI (over TF-IDF) model. Also plot a graph for frequency distribution of top 15 words of preprocessed data:

```
[paragraph 2013]
 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,

[paragraph 1885]
 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 1942]
 The first tax of this kind was hearth-money; or a tax of two shillings
```

```
upon every hearth. In order to ascertain how many hearths were in the
house, it was necessary that the tax-gatherer should enter every room
in it. This odious visit rendered the tax odious. Soon after the
Revolution, therefore, it was abolished as a badge of slavery.
```

Notice that the text is in the original form (no preprocessing or transformation used in print out).

1. Data loading and preprocessing

In this part we load and process (clean, tokenize and stem) data.

1.0. Fix random numbers generator with the following code:

```
$ import random; random.seed(123)
```

1.1 Open and load the file (it's UTF-8 encoded) using codecs. Some useful code:

```
$ import codecs
```

```
$ f = codecs.open("pg3300.txt", "r", "utf-8")
```

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.

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

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

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

```
$ import string
```

```
$ print(string.punctuation+"\n\r\t")
```

1.6. Using PorterStemmer stem words. Some useful code:

```
$ from nltk.stem.porter import PorterStemmer
```

```
$ stemmer = PorterStemmer()
```

```
$ word = "Economics"
```

```
$ print(stemmer.stem(word.lower()))
```

1.7. The class FreqDist of NLTK works like a dictionary where the keys are the words in the text and the values are the count associated with that word. For example, if you want to see how many words "tax" are in the text, you can type:

```
$ print(freqDist["tax"])
```

Remark1: *When processing data remember to keep a copy of original paragraphs (we will need them*

at the end to display querying results).

Remark2: This part result should be a list of paragraphs where each paragraph is a list of processed, lower-case words. For example for document with M paragraphs, where paragraph 1. has N words and paragraph M. has K words:

```
[["paragraph1word1", "paragraph1word2", ... , "paragraph1wordN"],
...,
["paragraphMword1", "paragraphMword2", ... , "paragraphMwordK"]]
```

2. Dictionary building

In this part we filter (remove stopwords) and convert paragraphs into Bags-of-Words.

For more details see: https://radimrehurek.com/gensim/tut1.html

2.1. Build a dictionary (a structure mapping words into integers). Some useful code:

```
$ import gensim
$ dictionary = gensim.corpora.Dictionary(...)
```

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

```
$ stopword = "this"
$ stopword_id = dictionary.token2id[stopword]
$ ...
$ dictionary.filter_tokens(stop_ids)
```

2.2. Map paragraphs into *Bags-of-Words* using the dictionary. Some useful code:

```
$ dictionary.doc2bow(…)
```

Remark1: This part result should be a corpus object (=list of paragraphs) where each paragraph is a list of pairs (word-index, word-count).

3. Retrieval Models

In this part we convert Bags-of-Words into TF-IDF weights and then LSI(Latent Semantic Indexing) weights.

For more details see: https://radimrehurek.com/gensim/tut2.html and https://radimrehurek.com/gensim/tut3.html

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

```
$ tfidf_model = gensim.models.TfidfModel(...)
```

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

```
$ tfidf_corpus = tfidf_model[...]
```

3.3 Construct MatrixSimilarity object that let us calculate similarities between paragraphs and queries:

```
$ ... = gensim.similarities.MatrixSimilarity(...)
```

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)). Some useful code:

```
$ ... = gensim.models.LsiModel(tfidf_corpus, id2word=dictionary,
num_topics=100)
```

```
$ lsi_corpus = lsi_model[...]
```

```
$ ... = gensim.similarities.MatrixSimilarity(...)
```

3.5 Report and try to interpret first 3 LSI topics. Some useful code:

```
$ lsi_model.show_topics()
```

4. Querying

In this part we query the models built in the previous part and report results.

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. Some useful code:

```
$ query = preprocessing("What is the function of money?")
```

```
$ query = dictionary.doc2bow(query)
```

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:

```
(tax: 0.26, econom: 0.82, influenc: 0.52)
```

Some useful code:

```
$ ... = tfidf_model[...]
```

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?"* results are (paragraph numbers are optional):

```
[paragraph 379]
 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,

[para
 grap
 h
 2003
 ]
 Capi
 tati
 on
 Taxe
 s.

[paragraph 2013]
 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,
```

Some useful code:

```
$ doc2similarity = enumerate(tfidf_index[tfidf_query])
$ print(sorted(doc2similarity, key=lambda kv: -kv[1])[:3] )
```

4.4 Convert query TF-IDF representation (for the query *"What is the function of money?"*) into LSI-topics representation (weights). Report top 3. topics with the most significant (with the largest absolute values) weights and top 3. the most relevant paragraphs according to LSI model. Compare retrieved paragraphs with the paragraphs found for TF-IDF model.

For example, for the query *"How taxes influence Economics?"* top 3 LSI topics are:

```
[topic 3]
 0.496*"tax" + 0.229*"rent" + -0.177*"capit" + -0.177*"labour" + -
0.175*"employ" + 0.162*"u [topic 5]
 -0.360*"tax" + -0.258*"capit" + -0.165*"foreign" + -0.142*"consumpt" + -
0.137*"trade" + -0 [topic 9]
 -0.286*"tax" + -0.254*"coloni" + 0.204*"bank" + 0.170*"export" + 0.165*"duti" + -0.164*"la
```

Some useful code:

```
$ lsi_query = lsi_model[tfidf_query]
$ print( sorted(lsi_query, key=lambda kv: -abs(kv[1]))[:3] )
$ print( lsi.show_topics() )
$ doc2similarity = enumerate(lsi_index[lsi_query])
$ print( sorted(doc2similarity, key=lambda kv: -kv[1])[:3] )
```

Delivery

Make sure to document every step of the assignment, including the description of everything you have done and the source code.