Όνομα: xx

ΑΜ: xx

Γλωσσική Τεχνολογία

Μέρος Α

Σχετικά αρχεία:

Spiders: Bbc.py, Articles.py,

news-web-similarity.py

Αρχικά, το scrapy έτρεξα με τον εξής τρόπο.

Στο terminal

```
$ pip install scrapy

$ scrapy startproject newsweb

$ cd newsweb

$ scrapy genspider Articles eu.usatoday.com

$ scrapy genspider Bbc www.bbc.com
```

Επειδή έχουμε 2 spiders πρέπει να τρέξουμε 2 αρχεία.

```
$ scrapy crawl -o Articles.json Articles

$ scrapy crawl -o Articles2.json Bbc
```

Στο σύνολο του μερους Α χρησιμοποίθηκαν

- nltk.stem.WordNetLemmatizer(): για τη δημιουργία λημμάτων.
- nltk.pos_tag(): για το POS Tagging.
- nltk.FreqDist(): για τον υπολογισμό του term frequency.
- Και φυσικά το scrapy.

Όλο το υπόλοιπο μέρος Α είναι δικιάς μου υλοποίησης

Έπειτα τρέχουμε το news-web-similarity.py

Το αρχείο αυτό έχει όλες τις συναρτήσεις για επεξεργασία των αρχείων που δημιουργήθηκαν από το scrapy.

- *build_from_scrapy(files=None):* Μπορούμε να φτιάξουμε το ανεστραμμένο ευρετήριο από τα αρχεία json που δημιούργησε το scrapy. Αν δεν δώσουμε ώρισμα χρησιμοποιεί όλα τα .json

αρχεια που βρισκονται στο ιδιο directory.

```
>>> inverted_index = build_from_scrapy()
Loading 221 data
Loaded 221 data
```

Μπορουμε να δώσουμε ώρισμα λιστα με τα ονόματα των αρχείων.

```
>>> build_from_scrapy(['newsweb/articles.json', 'newsweb/articles2.json'])
Loading 841 data
Loaded 841 data
```

Αποθήκευση και επαναφόρτωση ευρετηρίου.

- *write_to_xml(fname, inverted_index):* Αποθηκεύει το ανεστραμμένο ευρετήριο σαν ένα .xml αρχείο.

```
>>> write_to_xml('inverted_index.xml', inverted_index)
Saved inverted index to C:\Users\DX\PycharmProjects\news-web-similarity\inverted_index.xml
```

- *read_xml(fname):* Φορτώνει ένα .xml ανεστραμμένο ευρετήριο σαν dictionary.

```
>>> inverted_index = read_xml('inverted_index.xml')
file inverted_index.xml loaded containing 26284 indexes
```

Αξιολόγηση ευρετηρίου

- *search(string_of_lemmas, inverted index):* Κάνει αναζήτηση των λέξεων στο ανεστραμμένο ευρετήριο.
  print_dict(dictionary): Βοηθητική συνάρτηση, εμφανίζει ένα dictionary με ευανάγνωστο τρόπο (κατακόρυφα)

```
>>> result = search("google", inverted_index)
>>> print_dict(result)
https://eu.usatoday.com/story/tech/columnist/2020/03/31/wi-fi-mesh-system-bolster-your-home-connectivity/5032441002/ 7.3801384444990505
https://eu.usatoday.com/story/tech/columnist/komando/2021/08/05/how-recall-email-and-other-hidden-google-settings-save-time/5471193001/ 62.73117677824193
https://eu.usatoday.com/story/tech/reviews/2020/11/16/apples-homepod-mini-review-better-price-privacy-policy-than-google/6282049002/ 47.97089988924383
https://eu.usatoday.com/story/tech/columnist/2020/06/04/working-home-what-backdrop-video-call-says-you/3145490001/ 3.6900692222495253
https://eu.usatoday.com/story/tech/columnist/2021/07/30/fix-apple-imessage-android-problems-ipad/5418383001/ 3.6900692222495253
```

- *timeth(string_of_lemmas, times, inverted_index):* Χρονομετρεί times επαναλήψεις της συνάρτησης search σε ένα ανεστραμμένο ευρετήριο και εμφανιζει τον μέσο όρο.

```
>>> timeth("is", 20, inverted_index)
... timeth("is have", 20, inverted_index)
... timeth("is have man", 30, inverted_index)
... timeth("is is have man google", 30, inverted_index)
Average time of Query:"is" for 20 times: 0.0002991795539855957
Average time of Query:"is have" for 20 times: 0.0012467265129089355
Average time of Query:"is have man" for 30 times: 0.0012632131576538086
Average time of Query:"is is have man google" for 30 times: 0.001829377810160319
```

Μετρήσεις μέσων χρόνων για το ανεστραμμένο ευρετήριο

```
Average time of Query:"is" for 20 times: 0.0002991914749145508
Average time of Query:"is have" for 20 times: 0.0010970592498779296
Average time of Query:"is have man" for 30 times: 0.0010970830917358398
Average time of Query:"is is have man google" for 30 times: 0.0017287254333496094
```

Figure 1: Μέσοι χρόνοι για 1,2,3,4 λέξεις αντίστοιχα.

Μέρος Β

Σχετικά αρχεία:

document_classification_tf_idf.py

Βασικά υποσυστήματα

Προ-επεξεργασία των συλλογών Ε και Α

Για την προ-επεξεργασία χρησιμοποιώ τις συναρτήσεις:

```
>>> collection_E[0]
'From: darice@yoyo.cc.monash.edu.au (Fred Rice)\nSubject: Re:
```

- *depunctuate(list_of_strings):* Δικιά μου υλοποίηση, αφαιρεί διάφορους ειδικούς χαρακτήρες.

```
>>> depunctuate(collection_E)[0]
'From  darice yoyo cc monash edu au  Fred Rice \nSubject  Re
```

- *stemming(list_of_documents):* Με τη βοήθεια της συνάρτησης NLTK.stem.PortStemmer για θεματοποίηση.

```
>>> stemming(collection_E)[0]
['from', 'daric', 'yoyo', 'cc', 'monash', 'edu', 'au', 'fred', 'rice', 'subject', 're',
```

- *stop_words_removal(list_of_documents):* Δικιά μου υλοποίηση, αφαιρεί τα stopwords με τη βοήθεια του λεξιλόγιου NLTK.corpus.stopwords.words('english').

```
>>> stop_words_removal(collection_E)
['daric yoyo cc monash edu au fred rice subject islam dress
```

Δημιουργία χώρου χαρακτηριστικών

tf_idf(doc, top_N): Μερικώς δικιά μου υλοποίηση, με τη χρήση sklear.feature_extraction.text.TfidfVectorizer για τον υπολογισμό του μητρώου με τις τιμες tf-idf. Επιστρέφει τα κορυφαία N χαρακτηριστικά.

```
>>> doc1_S = tf_idf(collection_A, 4000)
>>> doc1_S
array(['god', 'thi', 'christian', ..., 'lourd', 'pictel', 'transgress'],
      dtype='<U78')
>>> len(doc1_S)
4000
```

Σύγκριση διανυσμάτων χαρακτηριστικών

def cosine_similarity(doc1, doc2): Μερικώς δικιά μου υλοποίηση, με τη βοήθεια του np.dot και np.liangl.norm.