Γλωσσική Τεχνολογία

Εργασία 2020-2021

έΥΑ ντουρου (παλαιοσ αμ 235854, νεοσ αμ 1041567)

2021

Περιεχόμενα

[Μέρος Α 2](#_Toc82266258)

[Προσκομιστής Ιστοσελίδων 2](#_Toc82266259)

[Προεπεξεργασία Δεδομένων 2](#_Toc82266260)

[**Politico** 3](#_Toc82266261)

[**Euronews** 3](#_Toc82266262)

[**BBC** 3](#_Toc82266263)

[**Aljazeera** 4](#_Toc82266264)

[**Eureporter** 4](#_Toc82266265)

[Μορφοσυντακτική Ανάλυση 4](#_Toc82266266)

[Αναπαράσταση ιστοσελίδων στο Μοντέλο Διανυσματικού Χώρου. 5](#_Toc82266267)

[Δημιουργία του ευρετηρίου 7](#_Toc82266268)

[Αξιολόγηση Ευρετηρίου 8](#_Toc82266269)

[Ερωτήματα της μιας λέξης 8](#_Toc82266270)

[Ερωτήματα των δύο λέξεων 9](#_Toc82266271)

[Ερωτήματα των τριών λέξεων 10](#_Toc82266272)

[Ερωτήματα των τεσσάρων λέξεων 11](#_Toc82266273)

[Μέρος Β 12](#_Toc82266274)

[Προ-επεξεργασία των συλλογών Ε και Α 12](#_Toc82266275)

[Δημιουργία χώρου χαρακτηριστικών 13](#_Toc82266276)

[Δημιουργία διανυσμάτων χαρακτηριστικών 13](#_Toc82266277)

[Σύγκριση διανυσμάτων χαρακτηριστικών 13](#_Toc82266278)

[Αναφορές 16](#_Toc82266279)

Ο κώδικας και τα περισσότερα αρχεία, με εξαίρεση όσα ήταν πολύ μεγάλα σε μέγεθος, μπορούν να βρεθούν στο repository [**https://github.com/evedour/scraper\_tintin**](https://github.com/evedour/scraper_tintin)

# Μέρος Α

## Προσκομιστής Ιστοσελίδων

Ο προσκομιστής ιστοσελίδων υλοποιήθηκε με τη χρήση του Scrapy (1), το οποίο λαμβάνει ως πηγές τα RSS feeds που φαίνονται στον πίνακα 1 και εξάγει το περιεχόμενό τους σε μορφοποίηση html, το οποίο και αποθηκεύει στο directory του project. Η υλοποίησή του φαίνεται στο αρχείο parker.py. (./scraper\_tintin/Scraper/NewsScrape/NewsScrape/spiders/parker.py)

Πίνακας 1 Πηγές του crawler

|  |  |
| --- | --- |
| **Όνομα πηγής** | **RSS feed Link** |
| Eureporter | https://www.eureporter.co/feed/ |
| Aljazeera | https://www.aljazeera.com/xml/rss/all.xml |
| Euronews | http://feeds.feedburner.com/euronews/en/home/ |
| BBC | http://feeds.bbci.co.uk/news/world/rss.xml |
| Washington Post | http://feeds.washingtonpost.com/rss/world |
| Politico | https://www.politico.eu/feed/ |
| NY Times | https://www.nytimes.com/svc/collections/v1/publish/https://www.nytimes.com/section/world/rss.xml |

Από τις παραπάνω ιστοσελίδες, οι NY times δεν επέτρεπαν πρόσβαση σε scraper μέσω του robots.txt.

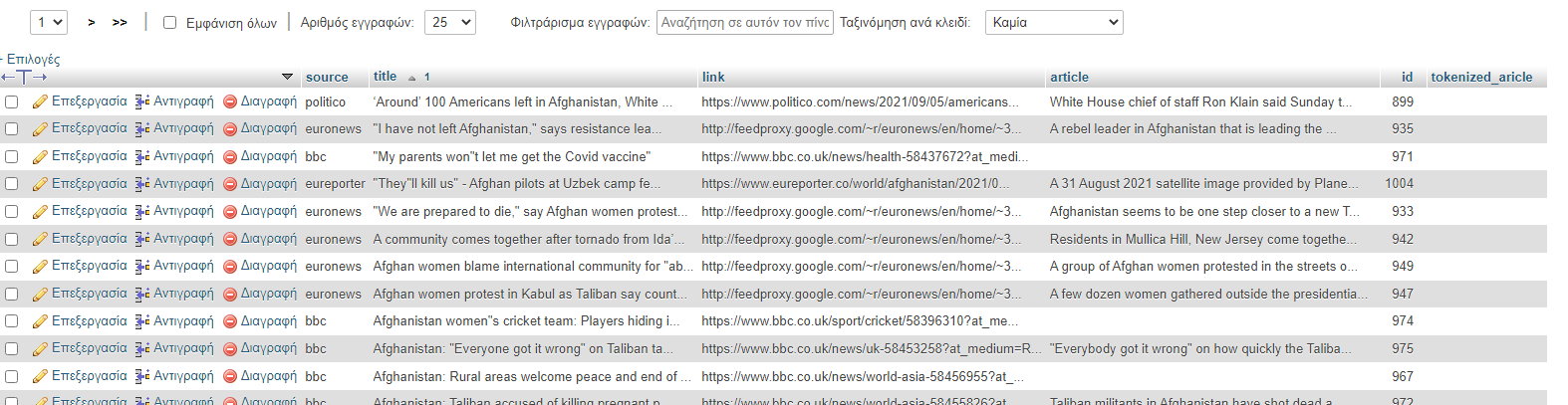
Οι σελίδες αυτές μετά το scrape, αποθηκεύονται σε αρχεία html, για την προεπεξεργασία.

## Προεπεξεργασία Δεδομένων

Για την προεπεξεργασία δεδομένων παίρνουμε ξεχωριστά την κάθε αποθηκευμένη σελίδα και με χρήση του εργαλείου Beautiful Soup, εξάγουμε από αυτή τους τίτλους. Συγκεκριμένα, ο τρόπος εξαγωγής για κάθε σελίδα φαίνεται στην αντίστοιχη παράγραφο παρακάτω.

Στο τέλος αυτής της διαδικασίας έχουμε για κάθε σελίδα τρεις λίστες titles, links, articles οι οποίες περιλαμβάνουν αντίστοιχα τους τίτλους, τα urls προς τα άρθρα, τα οποία χρησιμοποιούνται ως μοναδικά κλειδιά στις επόμενες φάσεις, για να ξεχωρίζουμε τα άρθρα και τέλος τα άρθρα.

Στη συνέχεια, εισάγουμε με τη σειρά τις παραπάνω λίστες σε localhost database, με χρήση MySQL και phpmyadmin. Η βάση δεδομένων έχει τη μορφή που φαίνεται στην εικόνα 1 και περιλαμβάνει και στήλη για την αποθήκευση του tokenized άρθρου, το οποίο γίνεται σε επόμενα ερωτήματα.



Εικόνα 1

### **Politico**

Κάθε άρθρο περιλαμβάνεται υπό ένα <item> tag.

Οι τίτλοι και τα links των άρθρων βρίσκονται στα αντίστοιχα tags.

To RSS του politico περιλαμβάνει και ολόκληρα τα άρθρα, στο tag <content:encoded>, το οποίο το Beautiful Soup μετατρέπει απλά σε <encoded>. Το εξάγουμε μέσω της findAll() και στη συνέχεια το μετατρέπουμε σε καλύτερη μορφή, αφαιρώντας τα tags :

article = re.sub('<.\*?>', '', article)

και τα newlines:

article = ''.join(article.strip().split('\\n'))

Αξίζει να σημειωθεί πως ανάλογη επεξεργασία γίνεται και στα links, καθώς περιλαμβάνουν ειδικούς χαρακτήρες \r, \n οι οποίοι δυσκολεύουν την χρήση τους.

### **Euronews**

Αντίστοιχα με το Politico, οι τίτλοι και τα links εξάγονται από τα αντίστοιχα tags.

Το Euronews δεν περιλαμβάνει ολόκληρα άρθρα στο RSS feed του, επομένως χρησιμοποιούμε το κάθε link για να κάνουμε εκ νέου request και στη συνέχεια να εξάγουμε το άρθρο από τη σελίδα του μέσω του Beautiful Soup.

r = requests.get(link)  
good\_soup = BeautifulSoup(r.content, 'html.parser')

Συγκεκριμένα εξάγουμε όλα τα div του class 'c-article-content' και στη συνέχεια από αυτά όλα τα p tags, δηλαδή τις παραγράφους του άρθρου. Έπειτα της συνδέουμε όλες για να αποθηκεύσουμε ολόκληρο το άρθρο.

divs = good\_soup.find\_all('div', class\_ = 'c-article-content')

for div in divs:  
 for p in div.find\_all('p'):  
 article.append(p.text)

### **BBC**

To RSS feed γίνεται parse ακριβώς όπως στο Euronews.

Στη συνέχεια, πρέπει να πάρουμε το άρθρο από το link του, με εκ νέου scrape του. Συγκεκριμένα γίνεται ένα request στο link για το περιεχόμενο της σελίδας κι έπειτα λαμβάνουμε με χρήση του Beautiful Soup το άρθρο, εξάγοντας αρχικά το article tag με class "ssrcss-xalfp3-ArticleWrapper e1nh2i2l6", το οποίο περιέχει το άρθρο μαζί με πιθανές εικόνες κι έπειτα στο div με class 'ssrcss-uf6wea-RichTextComponentWrapper e1xue1i87' υπό το article tag εξάγουμε όλα τα paragraphs (p tags), τα οποία και περιέχουν το κείμενο του άρθρου.

### **Aljazeera**

Το parsing γίνεται όπως στο Euronews, με την εξαίρεση ότι το όνομα του class που αναζητούμε στο div tag είναι 'wysiwyg wysiwyg--all-content css-1vsenwb'.

### **Eureporter**

Το RSS feed αυτής της σελίδας περιλαμβάνει και άρθρα, επομένως το parsing γίνεται όπως και του Politico.

Για την αποθήκευση των άρθρων επιλέχθηκε SQL database γιατί έτσι η ανάκτησή τους γίνεται με ένα query και το link, δεν χρειάζεται να διατηρείται ανοιχτό αρχείο json ή xml και να γίνεται συνέχεια scrape, καθώς και είναι εύκολο να προστεθεί απλά στην εγγραφή του αντίστοιχου link το tokenized άρθρο στην συνέχεια.

### Μορφοσυντακτική Ανάλυση

Στη συνέχεια, με χρήση του εργαλείου NLTK, γίνεται tokenization και pos tagging για κάθε κείμενο της βάσης δεδομένων μας. Αρχικά, συλλέγουμε τα links και τα αντίστοιχα άρθρα από τη βάση, με ένα SELECT ερώτημα. Έπειτα, για κάθε link και άρθρο:

Αφαιρούνται οι χαρακτήρες \r,\t,\n οι οποίοι προκύπτουν από το scraping:

article = cleaner.clean\_link(article)

Το άρθρο γίνεται tokenize:

tokens = nltk.word\_tokenize(article)

και στη μεταβλητή tokens έχουμε λίστα από τις λέξεις tokens που εμφανίζονται σε αυτό. Το αποτέλεσμα αυτό επεξεργαζόμαστε ώστε να αφαιρέσουμε τα σημεία στίξης και τους χαρακτήρες που δεν αφορούν λήμματα ως εξής:

punctuations = "?:!.,;"  
keys = ['token', 'tag']  
for link, article in content:  
 print(f'Tokenizing {link}')  
 # remove tab, newlines etc  
 article = cleaner.clean\_link(article)  
 # save links  
 links.append(link)  
 # use nltk to tokenize and tag article  
 article = article.lower()  
 article = article.replace('{html}', "")  
 cleans = re.compile('<.\*?>')  
 cleantext = re.sub(cleans, '', article)  
 rem\_url = re.sub(r'http\S+', '', cleantext)  
 rem\_num = re.sub('[0-9]+', '', rem\_url)  
 tokenizer = RegexpTokenizer(r'\w+')  
 tokens = tokenizer.tokenize(rem\_num)  
 insert\_tokenized(tokens, link)

Έπειτα αφαιρούνται τα stopwords:

no\_stopwords = [word.lower() for word in tokens if not word in stopwords.words()]

Τέλος, γίνεται μορφοσυντακτικός σχολιασμός μέσω του pos\_tag() του NLTK και τα tokens, με τα pos tags σε μορφή διανύσματος αποθηκεύονται σε αρχείο json.

tagged = nltk.pos\_tag(no\_stopwords)  
entities = nltk.chunk.ne\_chunk(tagged)  
tokenized\_articles.append(tagged)

Το διάνυσμα για κάθε άρθρο είναι της μορφής:

[[“token”, “POS\_TAG”], [“token”, “POS\_TAG”],…,[“token”, “POS\_TAG”]]

και συνολίκα το αρχείο json στο οποίο αποθηκεύονται είναι της μορφής:

{ “link” : [[“token”, “POS\_TAG”], [“token”, “POS\_TAG”],…,[“token”, “POS\_TAG”]],

…..,

“link”: [[“token”, “POS\_TAG”], [“token”, “POS\_TAG”],…,[“token”, “POS\_TAG”]] }

Σε επίπεδο κώδικα, έχουμε μια λίστα links, η οποία περιλαμβάνει τα links και αντίστοιχα μια λίστα λιστών tokenized\_articles, η οποία, με αντιστοιχία στις θέσεις των links έχει την αντίστοιχη λίστα tokens, με το POS σχολιασμό τους, ως tuples.

## Αναπαράσταση ιστοσελίδων στο Μοντέλο Διανυσματικού Χώρου.

Όπως αναφέρθηκε στην παραπάνω παράγραφο, η αναπαράσταση των άρθρων ως διάνυσμα και η αποθήκευσή τους σε αρχείο για μετέπειτα χρήση γίνεται μαζί με τη μορφοσυντακτική ανάλυση τους, στο αρχείο tokenizer.py.

Στη συνέχεια, στο αρχείο lemmatizer.py γίνεται η αφαίρεση των closed class categories, το stemming και το lemmatization.

Η συνάρτηση που αφαιρεί τα closed class categories είναι η εξής:

def remove\_closedclasscategories(tagged\_dic):  
 closedclasscategories = ['.', ',', ':', 'CD', 'CC', 'DT', 'EX', 'IN', 'LS', 'MD', 'PDT', 'POS', 'PRP', 'PRP$', 'RP', 'TO', 'UH', 'WDT', 'WP', 'WP$', 'WRB', ".", ",", ":", "D" "C" "T" "X" "N" "S" "D" "DT" "OS" "RP" "RP$" "P" "O" "H" "DT" "P" "P$" "RB"]  
 for key in tagged\_dic:  
 article = tagged\_dic[key]  
 for pair in article:  
 if pair[1] in closedclasscategories:  
 pair.clear()  
 tagged\_dic[key] = article  
 return tagged\_dic

Δέχεται ένα dictionary με pos tagged tokens και διατηρεί σε λίστα τα tags που αποτελούν closed class categories. Στη συνέχεια κάνει iterate το dictionary, ελέγχοντας εάν το tag του κάθε token βρίσκεται στη λίστα και αν βρίσκεται το αφαιρεί από το άρθρο.

Στη συνέχεια αποθηκεύεται και το άρθρο χωρίς τα closed class categories σε αρχείο json για μετέπειτα χρήση:

with open('Results/pos\_tags\_noclosed.json', 'w')as json\_out:  
 json.dump(tagged\_dictionary, json\_out)  
lemmas = get\_lemmas(tagged\_dictionary)

Το lemmatization γίνεται με χρήση του WordNetLemmatizer, και το stemming με SnowballStemmer. Με το stemming μπορούμε να λάβουμε τις ρίζες των λέξεων μας, κάτι που διευκολύνει τα ερωτήματα, καθώς, για παράδειγμα οι λέξεις “terrorism”, “terrorist” ανάγονται και οι δύο στην “terror”, επομένως στο ευρετήριο αναζητούνται ευκολότερα.

Δημιουργούμε τα λήμματα εφαρμόζοντας lemmatizer και stemmer σε κάθε token του κάθε άρθρου και αποθηκεύουμε τη λίστα λημμάτων σε txt αρχείο για μετέπειτα χρήση:

def get\_lemmas(data):  
 # stem and lemmatize  
 lemmatizer = WordNetLemmatizer()  
 stemmer = SnowballStemmer('english')  
 lemmas = []  
 for text in data:  
 lemmas.append(stemmer.stem(lemmatizer.lemmatize(text, pos='v')))  
  
 # remove single character lemmas  
 for lemma in lemmas:  
 if len(lemma) < 3:  
 lemmas.remove(lemma)  
 # remove duplicates  
 lemmas = list(dict.fromkeys(lemmas))  
 for lemma in lemmas:  
 if lemma not in nltk.corpus.words.words():  
 lemmas.remove(lemma)  
 return lemmas

Η συνάρτηση δέχεται το άρθρο, το οποίο είναι σε μορφή λίστας tokens.Ελέγχοντας ανα token, δημιουργεί λήμματα και τα προσθέτει στη λίστα λημμάτων, η οποία στη συνέχεια θα επιστραφεί συνάρτηση που δημιουργεί το ευρετήριο.

Επιπλέον, αφαιρούνται τυχόν λήμματα που προκύπτουν πολύ μικρά:

for lemma in lemmas:  
 if len(lemma) < 3:  
 lemmas.remove(lemma)

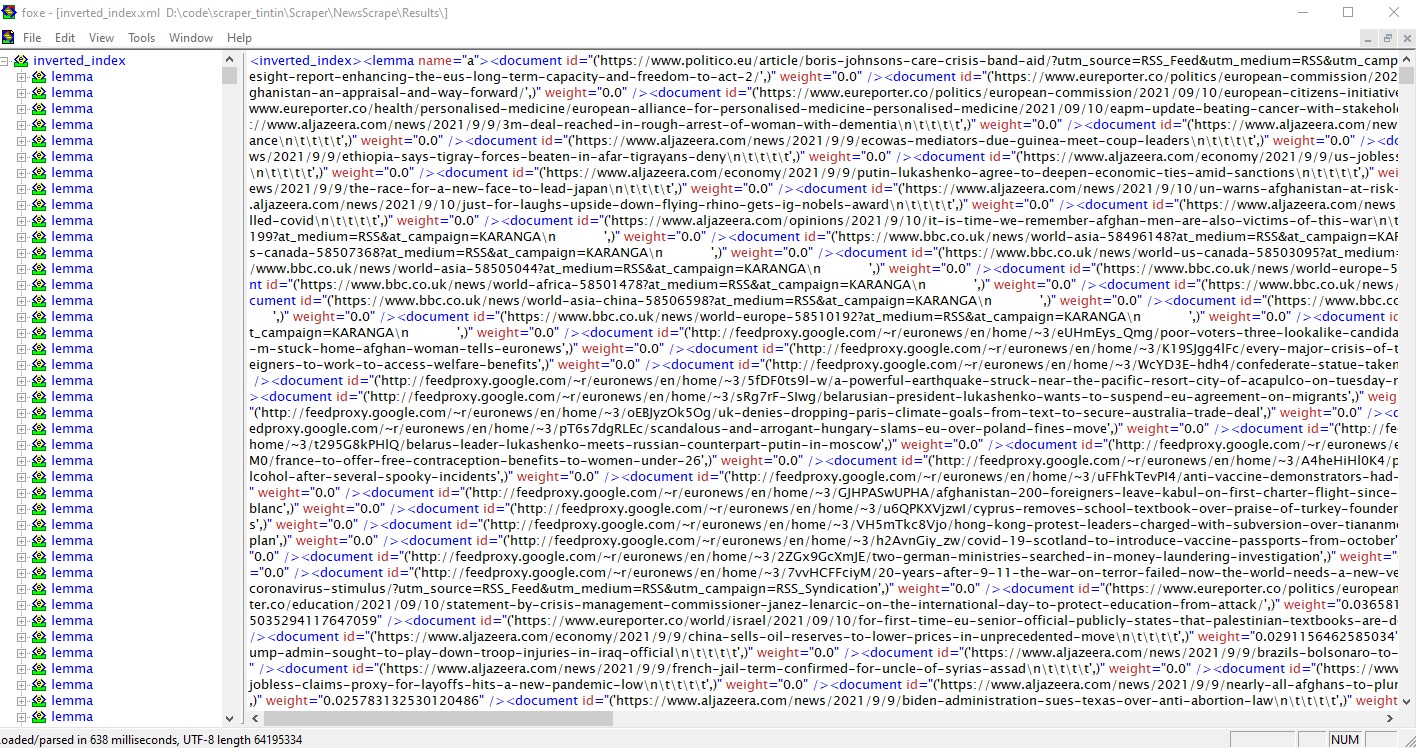
## Δημιουργία του ευρετηρίου

Έπειτα από τις παραπάνω διεργασίες, έχουμε στα χέρια μας μια λίστα λημμάτων, ένα genism dictionary, το οποίο κάνει map το κάθε μοναδικό token σε ένα id καθώς και διατηρεί το document frequency και τις εμφανίσεις των όρων συνολικά στα κείμενα, και ένα corpus, το οποίο διατηρεί τις εμφανίσεις των όρων στο κάθε κείμενο. Επομένως μπορούμε να μετρήσουμε τις συχνότητες των όρων στα κείμενα και να δημιουργήσουμε το ανεστραμμένο ευρετήριο. Για τον υπολογισμό της μετρικής TF-IDF χρησιμοποιείται το αρχείο indexer.py. Οι σχέσεις που χρησιμοποιούνται είναι οι (1), (2). Τη συχνότητα του όρου στο κάθε κείμενο λαμβάνουμε από το corpus, ελέγχοντας το κάθε tuple για κάθε document.

1. Το συνολικό αριθμό λέξεων στο κείμενο λαμβάνουμε από το μέγεθος της λίστας εμφάνισης όρων που αντιπροσωπεύει το κάθε κείμενο.
2. Το συνολικό αριθμό κειμένων λαμβάνουμε από το genism dictionary, το οποίο το διατηρεί στη μεταβλητή num\_docs. Τον αριθμό των κειμένων που περιέχουν το λήμμα λαμβάνουμε από το dictionary, το οποίο τον διατηρεί στη λίστα dfs.

Ο έλεγχος γίνεται για κάθε λήμμα και παράλληλα με χρήση της βιβλιοθήκης cElementTree του εργαλείου xml για python, γίνεται εγγραφή του ευρετηρίου στο output file”inverted\_index.xml”.

Στην εικόνα 2 φαίνεται κομμάτι του ευρετηρίου inverted\_index.py που δημιουργήθηκε:



Εικόνα 2

## Αξιολόγηση Ευρετηρίου

### Ερωτήματα της μιας λέξης

|  |  |  |
| --- | --- | --- |
| **Α/Α** | **ΕΡΩΤΗΜΑ** | **ΑΠΟΚΡΙΣΗ** |
| 1 | COVID | 14,79253339767450 |
| 2 | afghanistan | 14,30079984664910 |
| 3 | rights | 14,17088580131530 |
| 4 | Europe | 13,71914625167840 |
| 5 | DELTA | 18,21859693527220 |
| 6 | Korea | 14,26682353019710 |
| 7 | strain | 14,18088078498840 |
| 8 | Biden | 14,27883172035210 |
| 9 | states | 13,17845797538750 |
| 10 | London | 13,90203452110290 |
| 11 | terror | 14,76253724098200 |
| 12 | terrorist | 13,72712540626520 |
| 13 | China | 14,20287156105040 |
| 14 | Africa | 14,29881334304800 |
| 15 | president | 14,02797150611870 |
| 16 | minister | 13,66617846488950 |
| 17 | tech | 14,58265328407280 |
| 18 | covid-19 | 16,13476490974420 |
| 19 | health | 13,63933682441710 |
| 20 | hospitals | 14,63661909103390 |

### Ερωτήματα των δύο λέξεων

|  |  |  |
| --- | --- | --- |
| **Α/Α** | **ΕΡΩΤΗΜΑ** | **ΑΠΟΚΡΙΣΗ** |
| 1 | COVID pandemic | 15,85592365264890 |
| 2 | Taliban Afghanistan | 15,81095051765440 |
| 3 | women's rights | 15,21929001808160 |
| 4 | Europian news | 15,51411914825430 |
| 5 | DELTA variant | 15,48613405227660 |
| 6 | North Korea | 15,67102622985830 |
| 7 | virus strain | 14,74257349967950 |
| 8 | Joe Biden | 15,56109118461600 |
| 9 | united states | 14,83850669860830 |
| 10 | London calling | 14,98940467834470 |
| 11 | terror ensues | 15,36720347404470 |
| 12 | terrorist attacks | 15,71000623703000 |
| 13 | China bans | 15,12134528160090 |
| 14 | Africa news | 15,73599028587340 |
| 15 | president announcement | 16,12676787376400 |
| 16 | prime minister | 15,00241327285760 |
| 17 | tech university | 15,80295372009270 |
| 18 | covid-19 outbreak | 16,20572376251220 |
| 19 | health risk | 15,32822513580320 |
| 20 | hospitals capacity | 16,07181382179260 |

### Ερωτήματα των τριών λέξεων

|  |  |  |
| --- | --- | --- |
| **Α/Α** | **ΕΡΩΤΗΜΑ** | **ΑΠΟΚΡΙΣΗ** |
| 1 | COVID pandemic outbreak | 17,37605166435240 |
| 2 | Taliban overtake Afghanistan | 15,94185423851010 |
| 3 | women's rights fight | 15,86791682243340 |
| 4 | Europian political news | 18,46041846275320 |
| 5 | DELTA variant vaccine | 19,26496911048880 |
| 6 | covid vaccination presentage | 16,59648489952080 |
| 7 | new variant discovered | 16,77140021324150 |
| 8 | new variant deadly | 16,53053808212280 |
| 9 | novel virus strain | 16,53653454780570 |
| 10 | virus strain contagious | 16,34264492988580 |
| 11 | UniTED staTES politics | 17,05324029922480 |
| 12 | united states military | 16,53953456878660 |
| 13 | terror in east | 15,98984742164610 |
| 14 | terrorist attacks city | 16,00583696365350 |
| 15 | China bans site | 27,46463274955740 |
| 16 | Africa news world | 22,67925214767450 |
| 17 | president new announcement | 20,40989661216730 |
| 18 | prime minister virus | 20,76812410354610 |
| 19 | tech univeristy america | 20,71095919609060 |
| 20 | covid-19 new outbreak | 18,56021189689630 |
| 21 | high health risk | 18,31223988533020 |
| 22 | low health risk | 19,53516244888300 |
| 23 | hospitals maximum capacity | 19,00201344490050 |
| 24 | intensive care unit | 18,72801995277400 |
| 25 | authorities take action | 18,31529593467710 |
| 26 | war in east | 18,86723208427420 |
| 27 | COVID pandemic outbreak | 17,37605166435240 |
| 28 | Taliban overtake Afghanistan | 15,94185423851010 |
| 29 | women's rights fight | 15,86791682243340 |
| 30 | Europian political news | 18,46041846275320 |

### Ερωτήματα των τεσσάρων λέξεων

|  |  |  |
| --- | --- | --- |
| **Α/Α** | **ΕΡΩΤΗΜΑ** | **ΑΠΟΚΡΙΣΗ** |
| 1 | COVID pandemic outbreak danger | 19,59774875640860 |
| 2 | Taliban overtake Afghanistan Kabul | 20,10395503044120 |
| 3 | women rights fight protest | 18,88401317596430 |
| 4 | Europian political news England | 19,35007047653190 |
| 5 | DELTA variant vaccine efficient | 18,79439640045160 |
| 6 | covid vaccination presentage low | 18,19108080863950 |
| 7 | new variant discovered danger | 18,59502029418940 |
| 8 | new variant deadly young | 18,43780565261840 |
| 9 | novel virus strain delta | 20,42220139503470 |
| 10 | virus strain contagious younger | 18,27339315414420 |
| 11 | UniTED staTES politics NEWS | 18,52916407585140 |
| 12 | united states military war | 18,63567590713500 |
| 13 | terror in middle east | 18,51684212684630 |
| 14 | terrorist attacks city Kabul | 19,06617903709410 |
| 15 | China bans web site | 18,56027436256400 |
| 16 | Africa news world today | 18,13614964485160 |
| 17 | president new covid announcement | 19,33592326641080 |
| 18 | prime minister virus lockdown | 17,91204166412350 |
| 19 | texas tech univeristy america | 19,59161305427550 |
| 20 | covid-19 new outbreak contagion | 18,84943675994870 |
| 21 | high health risk old | 17,69494771957390 |
| 22 | low health risk younger | 19,56598401069640 |
| 23 | hospitals maximum capacity vaccine | 17,24323010444640 |
| 24 | healthcare intensive care unit | 19,41208577156060 |
| 25 | authorities take action law | 19,08716678619380 |
| 26 | war in east US | 18,63139748573300 |
| 27 | NATO announcement afghanistan aid | 18,05481386184690 |
| 28 | send aid war work | 18,22178125381460 |
| 29 | Kabul street protests rights | 18,99018764495840 |
| 30 | US investigate report war | 18,43180322647090 |

Μέση απόκριση: **17,17728240323060** seconds

# Μέρος Β

Για την υλοποίηση του δεύτερου μέρους χρησιμοποιήθηκαν οι συναρτήσεις του πρώτου, σε συνδυασμό με προσαρμοσμένα κομμάτια κώδικα, λόγω της διαφοράς του τύπου των δεδομένων.

Η συλλογή κειμένων φορτώθηκε από τη βιβλιοθήκη sklearn, ώστε να μπορεί να γίνει έλεγχος των αποτελεσμάτων της σύγκρισής μας, χρησιμοποιώντας τη μεταβλητή target για το κάθε κείμενο.

Λόγω έλλειψης υπολογιστικών πόρων, χρησιμοποιήθηκαν 100 άρθρα από την κάθε συλλογή (συνολικά 200), αντί των 20000 τα οποία περιέχει η συλλογή. Η επεξεργασία ολόκληρης της συλλογής απαιτούσε υπερβολικά πολλές ώρες και εξαντλούσε τη μνήμη του IDE που χρησιμοποιήθηκε.

## Προ-επεξεργασία των συλλογών Ε και Α

Η εξαγωγή των θεμάτων ξεκινά με το tokenization και lemmatization και των δύο συλλογών. Η διεργασία αυτή γίνεται στο αρχείο preprocessor.py. Η μέθοδος extract\_themes δέχεται τη λίστα δεδομένων (data) με τα κείμενα της κάθε συλλογής, και το όνομά της, το οποίο χρησιμοποιείται στην αποθήκευση των λημμάτων, και επιστρέφει λίστα των ίδιων κειμένων, tokenized και lemmatized.

# tokenize and lemmatize all documents in collection E  
print('Extracting lemmas in collection E: \n')  
e\_processed = preprocessor.exctract\_themes(collection\_e.data, 'e')  
print('\nExtracting lemmas in collection A: \n')  
a\_processed = preprocessor.exctract\_themes(collection\_a.data, 'a')

Στη συνέχεια, οι λίστες αυτές χρησιμοποιούνται για να δημιουργήσουν τα genism dictionaries, corpus, τα οποία είναι δομές οι οποίες διατηρούν, στη μορφή λίστας και λεξικών, συχνότητες όρων στη συλλογή και τα κείμενα, καθώς και τα μοναδικά λήμματα, mapped σε id. Τα λεξικά αυτά αποθηκεύονται, καθώς η κατασκευή τους απαιτεί πολύ χρόνο και resources.

# create dictionary based on the created lemmas  
# dictionary.token2id has id for tokens  
e\_dictionary = gensim.corpora.Dictionary(e\_processed)  
a\_dictionary = gensim.corpora.Dictionary(a\_processed)  
e\_dictionary.save('Data/e\_dictionary.txtdic')  
a\_dictionary.save('Data/a\_dictionary.txtdic')  
# create a dictionary reporting appearances of each word in each doc of collection e  
# this holds the number of appearances for each word in dictionary  
e\_cor = [e\_dictionary.doc2bow(doc) for doc in e\_processed]  
a\_cor = [a\_dictionary.doc2bow(doc) for doc in a\_processed]

Έπειτα δημιουργείται και index, όπως στο Μέρος Α της εργασίας. Για τη δημιουργία του χρησιμοποιείται ο indexer του Α μέρους.

e\_index = indexer.make\_index(e\_dictionary, e\_cor, range(len(e\_cor)), name='e')  
a\_index = indexer.make\_index(a\_dictionary, a\_cor, range(len(a\_cor)), name='a')



Τέλος, τα θέματα αποθηκεύονται sorted βάσει των εμφανίσεών τους στο λεξικό themes\_dictionary. Έτσι μπορούμε να πάρουμε μόνο κάποια από αυτά για το χώρο χαρακτητιστικών παρακάτω.

e\_dictionary =corpora.Dictionary.load('Data/e\_dictionary.txtdic')  
a\_dictionary =corpora.Dictionary.load('Data/a\_dictionary.txtdic')  
themes\_dictionary = {k: v for k, v in sorted(e\_dictionary.dfs.items(), key=lambda item: item[1])}

## Δημιουργία χώρου χαρακτηριστικών

Ως χώρος χαρακτηριστικών επιλέγονται τα πρώτα N θέματα με τις περισσότερες εμφανίσεις από το themes\_dictionary. Για τα τεστ σε 1000 κείμενα κάθε συλλογής, επιλέχθηκαν τα πρώτα 100.

## Δημιουργία διανυσμάτων χαρακτηριστικών

Οι συλλογές αναπαρίστανται ως λίστες λιστών της μορφής

e\_weights = [[(<term\_id>, <weight\_in\_document),…]….]

Η το index για κάθε εμφωλευμένη λίστα του e\_weights είναι το index του κειμένου στη συλλογή. Στο κάθε διάνυσμα στο αντίστοιχο index αποθηκεύεται tuple του id του λήμματος και του βάρους του στο αντίστοιχο κείμενο.

e\_weights = weights.get\_weights('inverted\_index\_e.xml', collection\_e, themes\_dictionary, e\_dictionary)  
a\_weights = weights.get\_weights('inverted\_index\_a.xml', collection\_a, themes\_dictionary, a\_dictionary)

## Σύγκριση διανυσμάτων χαρακτηριστικών

Ως μετρικές σύγκρισης επιλέχθηκαν οι cosine, Jaccard. Το αρχείο που περιέχει τη μέθοδο σύγκρισης compare() είναι το compare.py. Ο χρήστης δίνει το id του κειμένου της συλλογής Α που θέλει να κατηγοριοποιήσει και η μέθοδος επιστρέφει τα αποτελέσματα και με τις δύο μεθόδους, στη μορφή λίστας των κορυφαίων αποτελεσμάτων.

result\_cosine, result\_jaccard = compare.compare(a\_weights[int(user\_in)], e\_weights)

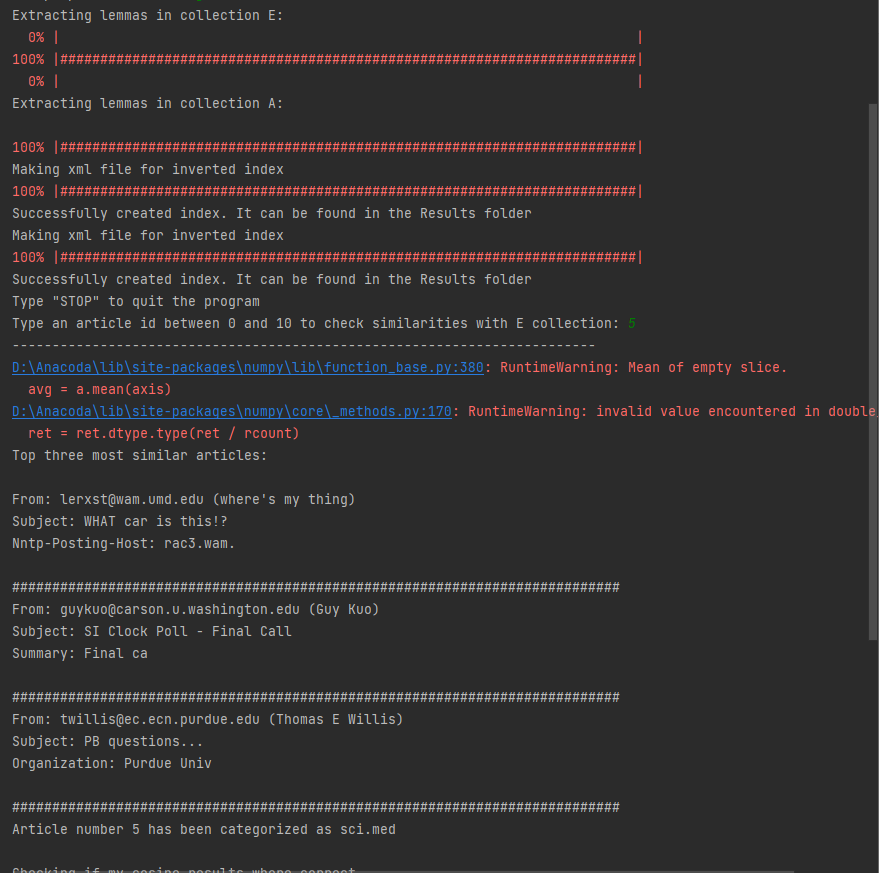
Το πρόγραμμα στη συνέχεια τυπώνει την κατηγοριοποίηση που είχε δοθεί στο κείμενο αρχικά από το dataset, και τις κατηγορίες που προέκυψαν από τη σύγκριση.

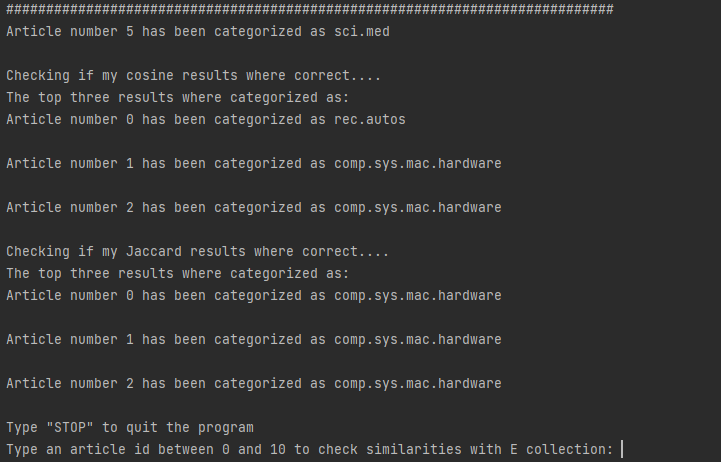
Η μέθοδος compare μετατρέπει τη λίστα από tuples σε διάνυσμα, και στη συνέχεια το συγκρίνει με όλα τα αντίστοιχα διανύσματα που προκύπτουν από τη διανυσματική αναπαράσταση των κειμένων της συλλογής Ε.

result\_cosine = []  
result\_jaccard = []  
vec\_in = [float(item) for t in vector for item in t]  
  
for article in collection:  
 index = collection.index(article)  
 art\_in = [float(item) for t in article for item in t]  
 if len(vec\_in) > len(art\_in):  
 max\_idx = len(art\_in)  
 del vec\_in[max\_idx:]  
 elif len(art\_in) > len(vec\_in):  
 max\_idx = len(vec\_in)  
 del art\_in[max\_idx:]  
 result\_cosine.append([compare\_by\_cosine(vec\_in, art\_in), index])  
 result\_jaccard.append([compare\_by\_jaccard(vec\_in, art\_in), index])

Για την υλοποίηση συγκρίσεων χρησιμοποιήθηκε η βιβλιοθήκη scipy και συγκεκριμένα οι μέθοδοι υπολογισμού απόστασης μεταξύ διανυσμάτων του module scipy.spatial.distance cosine() και Jaccard().

def compare\_by\_cosine(vector, article):  
 res = spatial.distance.cosine(vector, article)  
 return res  
  
  
def compare\_by\_jaccard(vector, article):  
 res = spatial.distance.jaccard(vector, article)  
 return res

**Παράδειγμα αποτελεσμάτων:**

****

# Αναφορές

1. **Scrapy.** Scrapy. [Ηλεκτρονικό] https://scrapy.org/.

2. **Crummy.** Beautiful Soup 4. [Ηλεκτρονικό] https://www.crummy.com/software/BeautifulSoup/.

3. **NLTK.** Natural Language Toolkit. [Ηλεκτρονικό] https://www.nltk.org/.

4. **Dustman, Andy.** Welcome to MySQLdb's documentation! *mysqlclient.* [Ηλεκτρονικό] [Παραπομπή: 4 9 2021.] https://mysqlclient.readthedocs.io/.

5. **XML Processing Modules. *Python.* [Ηλεκτρονικό] Python Software Foundation. [Παραπομπή: 7 9 2021.] https://docs.python.org/3/library/xml.html.**

**6. developers, SciPy. SciPy.org. *SciPy.org.* [Ηλεκτρονικό] scipy.org.**

**7. Python Software Foundation. -jsonJSON encoder and decoder. [Ηλεκτρονικό] https://docs.python.org/3/library/json.html.**