

**ΓΛΩΣΣΙΚΗ ΤΕΧΝΟΛΟΓΙΑ  
PROJECT 2021-2022  
ΑΝΑΦΟΡΑ**

Αθηνά Φουσέκη, Ε' Έτος

AM: 1059623

email: [st1059623@ceid.upatras.gr](mailto:st1059623@ceid.upatras.gr)

## **A' ΜΕΡΟΣ**

Για την υλοποίηση του πρώτου μέρους της εργασίας χρησιμοποίησα τις εξής βιβλιοθήκες της python:

```
1 import json
2 import os
3 import pandas as pd
4 import string
5 import nltk
6 from nltk.stem import WordNetLemmatizer
7 from pprint import pprint
8 import math
9 import re
10 from pymongo import MongoClient
11 import pickle
```

- json: Χρησιμοποίησα τη βιβλιοθήκη json γιατί τα άρθρα που έκανα scrape με τα δύο spiders που έφτιαξα, τα αποθήκευσα με την built-in εντολή του scrapy shell σε json αρχεία, συνεπώς, τη χρειαζόμουν για την διαχείριση του περιεχομένου των αρχείων αυτών.
- os: Χρησιμοποίησα τη βιβλιοθήκη os για την διαχείριση των json αρχείων και την αυτόματη ανάγνωσή τους.
- pandas: Για την επεξεργασία των κειμενικών δεδομένων χρησιμοποίησα dataframes και αποθήκευσα προσωρινά πληροφορία σε αυτά.
- string: Τη χρησιμοποίησα για διάφορες συναρτήσεις για συμβολοσειρές.
- nltk: Χρησιμοποίησα τη βιβλιοθήκη nltk, καθώς διαθέτει πληθώρα συναρτήσεων για την επεξεργασία φυσικής γλώσσας, π.χ. stemmers, lemmatizers, PoS-taggers κ.α.
- pprint: Τη χρησιμοποίησα για να περάσω το ανεστραμμένο αρχείο/ευρετήριο σε ένα αρχείο txt σε μία πιο ευανάγνωστη μορφή.
- math: Για τη συνάρτηση `log()`, την οποία χρησιμοποίησα για τον υπολογισμό του βάρους tf-idf.
- re: Χρειάστηκε για την συνάρτηση καθαρισμού του κειμένου από emojis.
- Pymongo: Για την μόνιμη αποθήκευση πληροφορίας σε βάση δεδομένων χρησιμοποίησα την MongoDB, συνεπώς χρειάστηκα τον driver pymongo για την επικοινωνία με τη βάση.
- Pickle: Αποθήκευσα το ανεστραμμένο αρχείο σε ένα αρχείο τύπου pickle, για να το χρησιμοποιήσω για την διεκπεραίωση των queries που ζητείται από την εκφώνηση.

### **Υποσυστήματα**

#### **1) Προσκομιστής Ιστοσελίδων**

Για τον προσκομιστή ιστοσελίδων, χρησιμοποίησα το scrapy.

```
(LinguisticTech) athinafus@pikachu:~/Documents/LinguisticTech$ scrapy version  
Scrapy 2.4.1
```

Έφτιαξα ένα environment, το LinguisticTech μέσω του anaconda, για το στήσιμο του scrapy.

Το ακόλουθο είναι το tree με τα αρχεία για το scrapy, καθώς και διάφορα αρχεία που προέκυψαν κατά την υλοποίηση της εργασίας.

```
(LinguisticTech) athinafus@pikachu:~/Documents/LinguisticTech/lingtech/lingtech$ tree  
.  
├── data.pkl  
├── drafts.py  
├── __init__.py  
├── inversed_file0.xml  
├── inversed_file.pkl  
├── inversed_file.xml  
├── items.py  
├── make_inv_file.py  
├── middlewares.py  
├── mongo_test.py  
├── output2_new2.txt  
├── output2_new.txt  
├── output2.txt  
├── output_new2.txt  
├── output_new.txt  
├── output.txt  
├── pipelines.py  
├── __pycache__  
│   ├── __init__.cpython-38.pyc  
│   └── settings.cpython-38.pyc  
├── queries_to_invFile.py  
├── settings.py  
├── spiders  
│   ├── articles_spider.py  
│   ├── __init__.py  
│   ├── __pycache__  
│   │   ├── articles_spider.cpython-38.pyc  
│   │   ├── __init__.cpython-38.pyc  
│   │   └── vox_spider.cpython-38.pyc  
│   └── vox_spider.py  
├── students.xml  
├── time_screenshots  
│   ├── Screenshot_20220106_000358.png  
│   ├── Screenshot_20220106_000531.png  
│   ├── Screenshot_20220106_000714.png  
│   ├── Screenshot_20220106_000757.png  
│   ├── Screenshot_20220106_000849.png  
│   ├── Screenshot_20220106_000922.png  
│   ├── Screenshot_20220106_000958.png  
│   ├── Screenshot_20220106_001039.png  
│   ├── Screenshot_20220106_001119.png  
│   └── Screenshot_20220106_001219.png
```

Σε έναν διαφορετικό φάκελο έχω περάσει τα αρχεία με τα άρθρα που έκανα scrape.

Οι ιστοσελίδες που διάλεξα για την συγκομιδή των άρθρων είναι οι:

- <https://english.elpais.com/>

## CORONAVIRUS CRISIS

### Spain's hospitals now treating more Covid-19 patients than during fourth and fifth waves

The infection curve keeps rising and 12,942 patients have been admitted into healthcare centers. Of these, 1,983 are in intensive care, close to the peaks of earlier surges

## CRIME

### Police arrest 37 for sexually exploiting underage girls in Madrid

PATRICIA PEIRO | Madrid

Ten minors have been freed from a ring that contacted them on social media and offered them money in exchange for selling drugs



## ANIMALS IN SPAIN

### Law change sees pets in Spain considered sentient beings, with welfare taken into account should a couple separate

MIGUEL ÁNGEL MEDINA | Madrid

More reforms are planned for this year, with the government aiming to bring about a change in citizens' relationships with animals

## Opinion

### OPINION

#### The European economy searches for its Michelin star

ÓSCAR GUINEA / ISABEL PÉREZ DEL PUERTO

### OPINION

#### An uninhabitable Earth?

MOISÉS NAIM

### OPINION

#### Germany's experiment in political philosophy

JEREMY CLIFFE

El Roto



- <https://www.vox.com/>

Vox

BIDEN ADMINISTRATION CORONAVIRUS RECODE THE GOODS FUTURE PERFECT THE HIGHLIGHT MORE ▾

Give



## Across All Majors and Gold

We take the lead when it comes to exceptional trading conditions by setting the standards. Tickmill™

Sign Up

## TOP STORIES



### The thirst for Jack Harlow, Gen Z's breakout white rapper, explained

The 23-year-old star is the latest product of the white rapper industrial complex.

By Terry Nguyen

### Democrats still have a Joe Manchin problem

The party's internal divides could trip up their agenda again in 2022.

By Li Zhou

### The stakes in the Supreme Court's vaccine cases are even bigger than they seem

The Court doesn't just threaten the



## THE GOODS

### America doesn't have enough teachers to keep schools open

The problem started way before omicron.

By Anna North

### The GOP's masculinity panic

David French on the cult of toughness on the Trumpist right.

By Sean Illing



### Hog farming has a massive poop problem

Inside North Carolina's search for solutions for its thousands of pig manure lagoons.

By Laura Bult

### The biggest fast food rollout of meatless fried chicken is happening next week

KFC will soon serve up Beyond Meat chicken at its 4,000 US locations.

By Kenny Torrella

### The great population growth slowdown

The pandemic has only accelerated a decline in US birth rates, says a

Για κάθε ιστοσελίδα έπαιξα ένα spider, στο οποίο έδινά ως start\_urls links από τις διάφορες κατηγορίες άρθρων, ανά σελίδα.

Για την El Pais:

```
class ArticlesSpider(scrapy.Spider):  
  
    name = "article"  
    start_urls = ['https://english.elpais.com/society/9/', 'https://english.elpais.com/international/9/',  
                  'https://english.elpais.com/culture/9/', 'https://english.elpais.com/sports/9/',  
                  'https://english.elpais.com/science-tech/9/', 'https://english.elpais.com/usa/9/',  
                  'https://english.elpais.com/economy-and-business/9/']
```

Για την Vox:

```
class ArticlesSpider(scrapy.Spider):  
  
    name = "article2"  
    start_urls = ['https://www.vox.com/business-and-finance/archives/14',  
                  'https://www.vox.com/culture/archives/14',  
                  'https://www.vox.com/policy-and-politics/archives/14',  
                  'https://www.vox.com/science-and-health/archives/14',  
                  'https://www.vox.com/world/archives/14',  
                  'https://www.vox.com/technology/archives/14', 'https://www.vox.com/energy-and-environment/archives/14']
```

Με την ακόλουθη εντολή, έτρεχε το κάθε spider κι αποθήκευα τα άρθρα σε ένα json αρχείο.

```
scrapy crawl article2 -o vox_articles_3_1_2-2022_p14.json
```

π.χ. Με αυτή την εντολή έτρεχα το spider που θα έκανε crawl τις ιστοσελίδες της vox, το οποίο έχει όνομα article2. Τα δεδομένα που θα συγκέντρωνε τα αποθηκεύονταν στο αρχείο vox\_articles\_3\_1\_2-2020\_p14.json.

Από κάθε άρθρο κρατάω τον τίτλο, το κειμενικό περιεχόμενο και το url της ιστοσελίδας.

Το κάθε spider επισκέπεται τα start\_urls που έχω ορίσει, σε αυτά εντοπίζει τα υπάρχοντα άρθρα κι έπειτα επισκέπεται κάθε ένα link που έχει βρει και αντιστοιχεί σε άρθρο. Από εκεί κάνει scrape τα δεδομένα που θέλουμε.

Ακολουθούν screenshots με τους κώδικες που κάνουν scrape τα δεδομένα από την ιστοσελίδα του κάθε άρθρου, για την El Pais και την Vox αντίστοιχα.

```
def parse_article(self, response):  
    #url = response.css('div.w_rs_i #btn_share_link_97::attr(href)').get().split('#?')[0]  
    arts = response.css('article.a_g._lg._g-o')  
  
    for art in arts:  
        try:  
            yield {  
                'title': art.css('h1.a_t::text').get(),  
                'url': art.css('a._btn_rs_l ::attr(href)').get().split('#?')[0],  
                'text': ' '.join(art.css('div.a_c_clearfix > p::text').getall()).replace("\n", ' ').replace("\r", ' ').replace(" ", ' ')  
            }  
        except:  
            continue
```



```
def parse_article(self, response):
    arts = response.css('article.l-segment.l-main-content')

    for art in arts:
        try:
            yield {
                'title': art.css('h1.c-page-title ::text').get(),_# <=====
                'url': response.request.url,
                'text': ' '.join(art.css('div.c-entry-content ::text').getall()).replace("\n", ' ').replace("\r", ' ').replace(' ', ' ')
            }
        except:
            continue
```

Τα αρχεία με την scraped πληροφορία μοιάζουν κάπως έτσι:

```
1 |
2 ~ 'title': 'Netflix's misguided Night Stalker series treats its cops like gods', 'url': 'https://www.vox.com/culture/22248673/netflix-night-stalker-docuseries-frank-salerno-gil-carrillo', 'text': " The climactic moment of Netflix's true crime
~ docuseries Night Stalker: The Hunt for a Serial Killer , is probably supposed to feel cathartic. In the final minutes of the four-part series' third installment, San Francisco detective Frank Falzon recalls how he tracked down a friend of the
~ California serial killer whose string of attacks throughout 1984 and 1985 made him a household name among true crime followers. Falzon describes this moment with relish almost four decades later. In his recounting, the friend - who'd originally
~ contacted police himself with a tip about the Night Stalker's identity - balked when Falzon asked him to reveal the Night Stalker's full name. So Falzon forcibly dragged the friend-turned-informant into his police car, threatened him, and
~ punched him in the face. "It wasn't my best punch, but it definitely wasn't my worst," Falzon says. After further threats, Falzon says, he lunged toward the informant, who cringed away from him. "threw his hands up in a cross," and stammered
~ out: "Richard Ramirez. Richard Ramirez. Richard Ramirez." As Falzon repeated the name, the music swelled and grew more ominous. The episode cut to the docuseries' cliffhanger end credits. And all I could think was how terrified this person must
~ have been of the police. To their credit, Night Stalker producers Tiller Russell and James Carroll have created a series that attempts to do exactly what true crime media should do: demystify the perpetrator and elevate the people impacted by
~ their crimes. Although Ramirez's lurid nickname gets the title credit, there's very little of him in Night Stalker 's narrative, which is almost entirely focused on the Los Angeles and San Francisco communities Ramirez prowled during his intense
~ 16-month period of home invasion, robbery, sexual assault, violence, and murder. But the strident erasure of Ramirez from this story of his crimes has also made Night Stalker a deeply confusing entry point for anyone who is unfamiliar with the
~ case. And it's opened up the series to other criticism: In its determination to avoid glorifying Ramirez, it instead glorifies the police who caught him. I'm not sure that version of the story is any less troubling. Night Stalker goes out of
~ its way to avoid including Ramirez in its narrative. Night Stalker is emphatically not about Richard Ramirez. In fact, the series has done a stellar job of assembling dozens of people who encountered Ramirez over the course of the investigation
~ into his crimes, from random witnesses to people who survived his attacks. Especially prominent are the police who hunted him, in particular Los Angeles County Sheriff's detectives Gil Carrillo and Frank Salerno. Carrillo and Salerno are
~ essentially the stars of Night Stalker ; as the younger, junior cop, Carrillo's perspective takes center stage, while Salerno, who was already a minor celebrity when Ramirez became active because of his prior work on the Hillside Strangler
~ murders , is introduced as a veritable denigro. Salerno is framed as larger than life, with stirring musical cues. And even decades later, Carrillo's voice retains a tinge of awe in recalling Salerno. These two are unquestionably our heroes.
~ But there's a huge hole at the center of this story in the shape of the Linky, hollow-eyed Ramirez. As a true crime fan, I've complained in the past about media that goes too far in the direction of glorifying the killer and allowing them to
~ control the narrative surrounding their crimes - Ted Bundy being the glaring, trope-setting standard . It's easy to see why Night Stalker 's producers would want to avoid this with Ramirez; he's one of the "big" names in true crime, a serial
~ killer whose name (or at least nickname) you're likely to recognize, in the same category as Bundy, BTK, or the Zodiac. So it's plausible that the intended audience for the documentary is assumed to be familiar with Ramirez's case, and that they
~ didn't feel it necessary to rehash too many details. But while Ramirez was headline news for much of the '80s (his attacks took place in 1984 and 1985, his trial began in 1988, and he was convicted in 1989), by the time of his death from cancer
~ in 2013 while on California's death row, he had largely faded from the public awareness. So the brief glimpses of him that Night Stalker offers are actually more confusing than enlightening - they feel like odd interruptions, cameos from a
~ strange minor character who inexplicably pops up occasionally, waving pentagrams on his palms. Why pentagrams? Why is Ramirez embarking on his crime spree? What makes his crimes in particular so memorable in the annals of serial killing? Night
~ Stalker never makes any of this clear, nor does the series attempt to provide any context for either Ramirez's motives or the impact of his crimes on the communities he terrorized. Growing up in Texas, Ramirez experienced severe abuse from
~ multiple family members, including one who groomed him to be a sexual predator and another who filled Ramirez's head with lingering horrific memories and images from the Vietnam War at an impressionable early age. Ramirez began to claim he
~ worshipped Satan while still a teenager, and was committing sexual assault by his 20s. The element of Satanism is the lurid, headline-grabbing aspect of the Night Stalker case, but that had much more to do with the way Ramirez's claims fed the
```

Είναι μία λίστα από json records.

## 2) Προεπεξεργασία Δεδομένων

Με τον κώδικα, διαβάζω όλα τα json αρχεία και περνάω το περιεχόμενο, μετά από ένα μικρό clean-up, στο dataframe df. Στο dataframe περνάω ως πληροφορία:

```
counter = 1
for item in file_list:
    print(item)
    with open(item) as f:
        #data = json.load(f)
        list_of_jsons = f.readlines()
        #print(type(list_of_jsons))
        #print(list_of_jsons[2])
        for article in list_of_jsons[1:]:
            #print(article)
            try:
                response = article.replace('\n', '')
                response = response.replace('{}', '{}').replace('}', '}')
                response = response.replace(' ', ' ')
                #response = "[" + response + "]"
                #print(response)
                y = json.loads(response)
                #print(y)
                df = df.append({'Document': 'd{}'.format(counter), 'Url': y['url'], 'Path': str(item), 'Title': y['title'], 'Article': str(y['text'])},
                                ignore_index=True)
                counter += 1
            except:
                continue
```

- ένα id για το κάθε άρθρο, της μορφής di, όπου i είναι ο αύξων αριθμός του κειμένου/άρθρου.
- Το url της ιστοσελίδας του άρθρου.
- Το path για το αρχείο json που περιέχει το άρθρο.
- Τον τίτλο του άρθρου.
- Το περιεχόμενο του άρθρου.

Αφού έχω περάσει όλα τα άρθρα, αφαιρώ τυχόν duplicates, βάσει του τίτλου του άρθρου. Έπειτα, ανεβάζω τα δεδομένα στη βάση μου.

```
lengthBefore = len(df)
df.drop_duplicates(subset=['Title'], inplace=True, keep="first")
# length after removing duplicates
lengthAfter = len(df)
print(lengthAfter)
print(df.head(10).to_string())

# inserting the articles in the database without the duplicates
try:
    for index, row in df.iterrows():
        collection.insert_one({'_id': row['Document'], 'Url': row["Url"], 'Path': row["Path"], 'Title': row['Title'],
                                'Article': row['Article']})
except:
    print("PROBLEM OCCURRED DURING THE INSERTION OF ARTICLES TO THE DATABASE")
```

Έπειτα, “καθαρίζω” το κείμενο του κάθε άρθρου στις παρακάτω γραμμές κώδικα.

```
df["Article"] = [clean_article(i) for i in df['Article']] # cleaning article
df["Article"] = [remove_punct(i) for i in df['Article']] # removing punctuation from the article
df["Article"] = [remove_emoji(i) for i in df['Article']] # removing emojis from the article

df['Article'] = df['Article'].apply(lambda x: lemmatize(x)[0]) # lemmatize each word
df['Article'] = df['Article'].apply(lambda x: remove_one_lengthed(x)) # remove words of one character
df['Article'] = df['Article'].apply(lambda x: lower_letters(x)) # make all words lowercase
df['Article'] = df['Article'].apply(lambda x: calculate_freq(x)) # count frequency of each lemma in a document
df['Article'] = df['Article'].apply(lambda x: list(set(x))) # remove duplicate words
```

Οι συναρτήσεις που χρησιμοποίησα είναι:

1) **clean\_article**:

```
def clean_article(text):
    people_names = []
    doc = nlp(text)
    if len(doc.ents) != 0: # if there are named entities in the text
        # --SYNTAX--[f(x) for x in sequence if condition]-----
        [people_names.append(X.text) for X in doc.ents
         if X.label_ in ['QUANTITY', 'DATE', 'TIME', 'PERCENT', 'CARDINAL']]
    else:
        print('No named entities in text')
    name_regex = re.compile(r'\b%s\b' % r'\b|\b'.join(map(re.escape, reversed(people_names))))
    text = name_regex.sub("", text) # removing the named entities from the text
    text = text.replace("_", ' ')
    text = text.replace("-", ' ')
    text = text.replace("'", ' ')
    text = ''.join([i for i in text if not i.isdigit()]) # removing numbers from the text
    text = re.sub(' +', ' ', text) # removing too many spaces from text
```

Με την clean\_article αφαιρώ named entities που έχουν σχέση με αριθμούς, χρονολογίες, ημερομηνίες, ποσοστά κ.λπ.. Επίσης, αφαιρώ διάφορα συμβολα, πολλά συνεχόμενα κενά και όσους αριθμούς δεν αφαιρέθηκαν κατά την αφαίρεση των named entities.

## 2) `remove_punct`:

```
def remove_punct(text):  
    text.replace("'", '')  
    table = str.maketrans("", "", string.punctuation)  
    return text.translate(table)
```

Με αυτή τη συνάρτηση αφαιρώ τα σημεία στίξης από το κείμενο.

## 3) `remove_emoji`:

```
def remove_punct(text):  
    text.replace("'", '')  
    table = str.maketrans("", "", string.punctuation)  
    return text.translate(table)
```

Με αυτή τη συνάρτηση αφαιρώ τυχόν emojis που μπορεί να υπάρχουν σε ένα άρθρο.

## 4) `lemmatize`:

```
def lemmatize(text):  
    lemmmed = []  
    l1 = [(w, get_wordnet_pos(w)[0]) for w in nltk.word_tokenize(text)] # list that contains the PoS-tagged article  
    l2 = [(w, get_wordnet_pos(w)[1]) for w in nltk.word_tokenize(text)] # list that contains the FULL PoS-tagged article  
    for el in l1:  
        if el[1] != 'stopw':  
            ll = wordnet_lemmatizer.lemmatize(el[0], el[1])  
            lemmmed.append(ll)  
    for tup in l2[:]:  
        if tup[1] == 't':  
            l2.remove(tup)  
    return (lemmed, l2)
```

Με αυτή τη συνάρτηση κάνω ταυτόχρονα το PoS-Tagging (μορφοσυντακτική ανάλυση), την αφαίρεση των stopwords και το lemmatization. Στην αρχή, για κάθε λέξη του άρθρου, καλείται η `get_wordnet_pos`, η οποία επιστρέφει το PoS-tag της λέξης για την οποία έχει κληθεί, σε δύο μορφές: η πρώτη είναι η κατάλληλη για να δοθεί ως συμπληρωματικό γνώρισμα στον lemmatizer, ενώ η δεύτερη είναι το PoS-tag στην αρχική του μορφή, έτσι ώστε να το χρησιμοποιήσω αργότερα, για την εγγραφή των PoS-tagged άρθρων σε ένα αρχείο, όπως ζητείται από την εκφώνηση. Στα for loops που βλέπουμε, ελέγχω αν η λέξη που επεξεργάζομαι κάθε φορά είναι stopword, έτσι ώστε αν είναι, να μην την κρατήσω. Όσες δεν είναι stopwords, τις περνάω στην λίστα lemmmed. Τέλος, η `lemmatize` με τη σειρά της επιστρέφει δύο στοιχεία, την λίστα lemmmed με τις lemmatized λέξεις, η οποία αποθηκεύεται στο `df['Article']`, και τη λίστα l2, η οποία αποτελείται από tuples της μορφής (λήμμα, PoS-tag) και αποθηκεύεται στο `df_posT['Article']`.



#### 5) `get_wordnet_pos`:

```
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tagg = nltk.pos_tag([word])
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}

    #print(tagg[0][1])
    closed_class_category_tags = ['CC', 'DT', 'EX', 'IN', 'LS', 'MD', 'PDT', 'POS', 'PRP', 'PRP$', 'RP',
                                  'TO', 'UH', 'WDT', 'WP', 'WP$', 'WRB']

    if tagg[0][1] in closed_class_category_tags: # detecting stopwords
        return ("stopw", tagg[0][1])
    else:
        return (tag_dict.get(tag, wordnet.NOUN), tagg[0][1])
```

Η `get_wordnet_pos` κάνει το part-of-speech tagging, χρησιμοποιώντας την `pos_tag` από την βιβλιοθήκη `nltk`. Μέσω της δημιουργίας ενός dictionary αντιστοιχεί τα PoS tags που επιστρέφει η `pos_tag` σε κλειδιά, τα οποία είναι strings που αναγνωρίζει ο `WordNetLemmatizer` για να βελτιώσει το lemmatization. Επίσης, σε αυτή τη συνάρτηση, εντοπίζω ποιες λέξεις είναι stopwords, ώστε να έχει την πληροφορία η `lemmatize` που αναφέραμε παραπάνω, και να μην τις λαμβάνει υπόψη της. Τα tags των `closedclasscategories` τα πήρα από την <http://www.infogistics.com/tagset.html>, η οποία μας δινόταν στην εκπόνηση. Τέλος, η συνάρτηση επιστρέφει τα PoS-tags στις μορφές που αναλύσαμε παραπάνω.

#### 6) `remove_one_lengthed`:

```
def remove_one_lengthed(article_list):
    [article_list.remove(x) for x in article_list[:] if len(x) == 1]
    return article_list
```

Η `remove_one_lengthed` δέχεται ως όρισμα μια λίστα από λέξεις ( η οποία κάθε φορά αντιστοιχεί σε ένα άρθρο) και αφαιρεί όλες τις λέξεις μήκους 1. Τέλος, επιστρέφει αυτή τη νέα λίστα.

#### 7) `lower_letters`:

```
def lower_letters(word_list):
    word_list = [word.lower() for word in word_list]
    return word_list
```

Η `lower_letters` δέχεται ως όρισμα μια λίστα από λέξεις ( η οποία κάθε φορά αντιστοιχεί σε ένα άρθρο) και σε κάθε λέξη στη λίστα εφαρμόζει τη συνάρτηση `lower()` της `python`, η οποία μετατρέπει όλα τα γράμματα της λέξης σε πεζά. Τέλος, επιστρέφει αυτή τη νέα λίστα.

#### 8) `calculate_freq`:

```
def calculate_freq(word_list):
    temp_list = [0]*len(word_list)
    for i in range(0, len(word_list)):
        freq = word_list.count(word_list[i])
        temp_list[i] = (word_list[i], freq)
    return temp_list
```

Η `calculate_freq` δέχεται ως όρισμα μια λίστα από λέξεις ( η οποία κάθε φορά αντιστοιχεί σε ένα άρθρο) και υπολογίζει την συχνότητα εμφάνισης κάθε μίας από αυτές στο κείμενο. Τέλος, επιστρέφει μία λίστα από tuples, όπου κάθε tuple είναι της μορφής (λήμμα, συχνότητα εμφάνισης του λήμματος)

#### 9) **set**:

Τέλος, χρησιμοποιώ της συνάρτηση της python `set()` στη λίστα των tuples, έτσι ώστε να αφαιρέσω tuples που επαναλαμβάνονται, καθώς μία λέξη μπορεί να εμφανίζεται περισσότερες από μία φορές μέσα σε ένα άρθρο.

### 3) Δημιουργία του Ευρετηρίου

Σε αυτή τη φάση, φτιάχνω το ανεστραμμένο ευρετήριο. Χρησιμοποιώ τη δομή δεδομένων της python: dictionary.

Κάθε στοιχείο του dictionary έχει τη μορφή:

`"lemma" : [[ 'di', 1.23], [ 'dj', 4.56], ..., [ 'dk', 7.89]]`

Τα κλειδιά είναι τα λήμματα που υπάρχουν σε όλη αυτή τη συλλογή κειμένων. Το value κάθε κλειδιού είναι μία λίστα, η οποία αποτελείται από υπο-λίστες. Κάθε υπο-λίστα περιέχει ένα id εγγράφου/άρθρου, στο οποίο υπάρχει το κλειδί-λήμμα, και (αρχικά, για λόγους ευκολίας την συχνότητα του όρου στο κείμενο) το βάρος tf-idf του λήμματος στο κείμενο.

```
dict_index = {}
#creating the inverted file
for index, row in df.iterrows():
    for tup in row['Article']:
        try:
            dict_index[tup[0]].append([row['Document'], tup[1]])
        except:
            dict_index[tup[0]] = [[row['Document'], tup[1]]]

dict_index = dict(sorted(dict_index.items(), key=lambda item: item[1]))

list_for_up=[]
for item in dict_index:
    term_idf = int(lengthAfter)/int(len(dict_index[item]))
    for i in range(0, len(dict_index[item])):
        #element = [doc_id, freq]
        dict_index[item][i][1] = (1 + math.log(int(dict_index[item][i][1]),2))*term_idf
        # inserting the lemmas to the database
        list_for_up.append(
            {'Lemma': item, 'inDocument': dict_index[item][i][0], 'tfidf': dict_index[item][i][1]})
```

Μετά την ολοκλήρωση του “χτισίματος” του ευρετηρίου, ανεβάζω τα δεδομένα στη βάση.

```
collection2.insert_many(list_for_up) # uploading the inverted file information to the database
```

Τέλος, αποθηκεύω το ανεστραμμένο ευρετήριο σε αρχεία: με `pprint()` σε ένα αρχείο `txt`, σε ένα αρχείο `pickle` και σε ένα αρχείο `xml`, χτίζοντας το `string` βάσει των προδιαγραφών που ορίζει η εκφώνηση.

```
with open('output2_new2.txt', 'wt') as out:
    pprint(dict_index, stream=out)

# saving dictionary in pickle file
a_file = open("inversed_file.pkl", "wb")
pickle.dump(dict_index, a_file)
a_file.close()

# building the xml string for the inversed file
xml_string = "<inverted_index>\n"
for item in dict_index:
    xml_string+="<lemma name=\"{}\">\n".format(item)
    for lists in dict_index[item]:
        xml_string+="<document id=\"{}\" weight=\"{}\"/>\n".format(lists[0], lists[1])
    xml_string+="</lemma>\n"

xml_string += "</>\n"
f = open("inversed_file.xml", "w") # xml file with inversed file
f.write(xml_string)
f.close()
```

#### 4) Αξιολόγηση Ευρετηρίου

Για την αξιολόγηση του ευρετηρίου έπαιξα ένα άλλο script, με το οποίο διαβάζω το ευρετήριο από το pickle αρχείο που έχω αποθηκεύσει.

Με σκοπό να εξοικονομήσω χρόνο, τα queries γίνονται αυτόματα, διαβάζοντας λέξεις από ένα αρχείο (για τις περιπτώσεις των ερωτημάτων των 2, 3 και 4 λημμάτων).

Κάθε λέξη έχει υποβληθεί στην ίδια προεπεξεργασία που υπεβλήθει και το κάθε άρθρο.

Για τα ερωτήματα 1 λέξης:

```
start = time.time() # start time for one word queries
for lem in one_word:
    #print("LEMMA: {}".format(lem))
    start = time.time()
    doc_list = inverted_file[lem] # list of documents that contain the lemma
    sorted_tfidf = sorted(doc_list, key=lambda x: x[1], reverse=True) # list sorted based on tf-idf score

    #print("Lemma: {}".format(lem))
    #for item in sorted_tfidf:
        #print("{} {}".format(item[0], item[1]))
        #x = collection.find_one({'_id': item[0]}, {'Path': 0, 'Article': 0})
        #print(x)

end = time.time()
```

Βρίσκω την κάθε λέξη στο ανεστραμμένο αρχείο και σε ποια έγγραφο περιέχεται. Ταξινομώ τα κείμενα βάσει του tf-idf. Χρονομετρώ αυτή την διαδικασία. Εκτός χρονομέτρησης, στέλνω στη βάση το κατάλληλο query έτσι ώστε να μου επιστραφούν το id, ο τίτλος και το url του άρθρου, έτσι ώστε ο χρήστης να έχει τη δυνατότητα να επισκευτεί την ιστοσελίδα με το άρθρο που έχει “αναζητήσει”.

Για τα ερωτήματα πολλαπλών λέξεων:

Βρίσκω τα κείμενα στα οποία περιέχεται το κάθε λήμμα και μετά βρίσκω την τομή αυτών των συνόλων. Έπειτα, ταξινομώ την τομή βάσει του αθροίσματος των tf-idf scores των λημμάτων σε κάθε έγγραφο. Χρονομετρώ τη διαδικασία για κάθε σύνολο ερωτημάτων με βάση το μήκος τους.

Παρομοίως, εκτός χρονομέτρησης στέλνω στη βάση το κατάλληλο query για να μου επιστρέψει τα επιπλέον δεδομένα.

### Χρόνοι

Οι χρόνοι προκύπτουν αφού έχω τρέξει το πρόγραμμα 10 φορές κι έχω βγάλει Μ.Ο. για κάθε είδος query.

Μήκος Query	Χρόνος
1 word	0.0001 ms
2 words	87,41 ms
3 words	187,33 ms
4 words	43,57 ms

Τα αποτελέσματα αφού έχω τρέξει το πρόγραμμα:

```
Elapsed time for 1-word queries: 0.0001430511474609375 ms
Elapsed time for 2-word queries: 88.93730640411377 ms
Elapsed time for 3-word queries: 187.04302310943604 ms
Elapsed time for 4-word queries: 44.257656733194985 ms

Process finished with exit code 0
```

```
Elapsed time for 1-word queries: 0.00011920928955078125 ms
Elapsed time for 2-word queries: 84.18169021606445 ms
Elapsed time for 3-word queries: 181.95132414499918 ms
Elapsed time for 4-word queries: 37.31997807820638 ms

Process finished with exit code 0
```

```
Elapsed time for 1-word queries: 0.00015497207641601562 ms
Elapsed time for 2-word queries: 86.70270442962646 ms
Elapsed time for 3-word queries: 184.33037598927817 ms
Elapsed time for 4-word queries: 42.75266329447428 ms

Process finished with exit code 0
```



```
Elapsed time for 1-word queries: 0.00010728836059570312 ms
Elapsed time for 2-word queries: 91.57081842422485 ms
Elapsed time for 3-word queries: 198.55639934539795 ms
Elapsed time for 4-word queries: 46.10440731048584 ms
```

```
Process finished with exit code 0
```

```
Elapsed time for 1-word queries: 0.00013113021850585938 ms
Elapsed time for 2-word queries: 87.65677213668823 ms
Elapsed time for 3-word queries: 186.8321657180786 ms
Elapsed time for 4-word queries: 43.78319581349691 ms
```

```
Process finished with exit code 0
```

```
Elapsed time for 1-word queries: 0.00011920928955078125 ms
Elapsed time for 2-word queries: 92.19517707824707 ms
Elapsed time for 3-word queries: 190.018359820048 ms
Elapsed time for 4-word queries: 44.11606788635254 ms
```

```
Process finished with exit code 0
```

```
Elapsed time for 1-word queries: 0.00011920928955078125 ms
Elapsed time for 2-word queries: 90.5333399772644 ms
Elapsed time for 3-word queries: 187.2894525527954 ms
Elapsed time for 4-word queries: 48.44662348429362 ms
```

```
Process finished with exit code 0
```

```
Elapsed time for 1-word queries: 0.00015497207641601562 ms
Elapsed time for 2-word queries: 86.3666296005249 ms
Elapsed time for 3-word queries: 193.53524843851724 ms
Elapsed time for 4-word queries: 40.44107596079508 ms
```

```
Process finished with exit code 0
```

```
Elapsed time for 1-word queries: 0.00013113021850585938 ms
Elapsed time for 2-word queries: 85.49504280090332 ms
Elapsed time for 3-word queries: 183.55349699656168 ms
Elapsed time for 4-word queries: 44.2262331644694 ms
```

```
Process finished with exit code 0
```

```
Elapsed time for 1-word queries: 0.00010728836059570312 ms
Elapsed time for 2-word queries: 80.51506280899048 ms
Elapsed time for 3-word queries: 180.32963275909424 ms
Elapsed time for 4-word queries: 44.35296058654785 ms
```

```
Process finished with exit code 0
```

Επιπλέον, έχω αναπτύξει κώδικα που επιτρέπει στον χρήστη από την γραμμή εντολών να κάνει δικά του queries και να του επιστρέφονται τα κατάλληλα άρθρα από τη βάση, σε ένα pretty table. Ο χρήστης μπορεί να κάνει επαναληπτικά ερωτήματα, και όταν αποφασίσει ότι θέλει να σταματήσει, γράφει “no” στη γραμμή εντολών όταν το σύστημα τον ρωτάει τι θα ήθελε να κάνει.

Παράδειγμα εκτέλεσης του προγράμματος:

Title	Url
One Good Thing: An unsung drama of the 2010s you should stream right now	https://www.vox.com/culture/22634896/er-robot-review-rmi-malek-netflix-grime-video
How hatred of gay people became a key plank in Hungary's authoritarian turn	https://www.vox.com/22547228/hungary-orban-lgbt-law-pedophilia-authoritarian
Rockefeller Foundation president Raj Shah: the Code Conference interview (transcript)	https://www.vox.com/recode/2019/7/1/19238719/rockefeller-foundation-raj-shah-code-conference-interview-transcript-teddy-schleifer-philanthropy
Why it's so satisfying to root for Disney villains	https://www.vox.com/culture/22453479/disney-villains-cruella-ur-sala-maleficent-scar-fans-jung-archetypes
Lil Nas X's evil gay Satanic agenda, explained	https://www.vox.com/22356438/lil-nas-x-satan-shoes-nike-montero-video-gay-agenda-christian-controversy
Ripple CEO Brad Garlinghouse explains why big banks should get into cryptocurrencies	https://www.vox.com/recode/2019/5/28/18613886/ripple-brad-garlinghouse-banks-bitcoin-blockchain-cryptocurrency-xrp-kara-swisher-podcast-interview
Plagues, famine, torture: historians try to set the record straight on the Middle Ages	https://english.elpais.com/usa/2021-07-19/plagues-famine-torture-historians-try-to-set-the-record-straight-on-the-middle-ages.html
What's in a name? For some brands, a racist history primed to be toppled.	https://www.vox.com/the-highlight/21398236/cleveland-indians-washington-redskins-name-racism-aunt-jentima-dixie-chicks-fair-and-lovely
The anti-American right	https://www.vox.com/22605599/olypics-conservatives-silence-bites-anti-american
"There was no Reconquest. No military campaign lasts eight centuries"	https://www.vox.com/vox-conversations-podcast/2021/11/8/22763282/vox-conversations-sebastian-junger-freedom
The complicated case of Allen v. Farrow, in one timeline	https://english.elpais.com/arts/2020-02-28/henry-kamen-there-was-no-reconquest-no-military-campaign-lasts-eight-centuries.html
Why 2020 presidential candidate Andrew Yang doesn't want to break up Google	https://www.vox.com/2020/7/19/20781175/andrew-yang-2020-presidential-race-google-breakup-tech-warren-kara-swisher-recode-decode-podcast
How voice actors are fighting to change an industry that renders them invisible	https://www.vox.com/culture/22955298/andy-allen-sia-farrow-timeline-hbo
How to fix Lebanon's political crisis	https://www.vox.com/2020/7/22/21326024/white-voice-actors-black-characters-cartoons-whitewashing
Is there an uncontroversial way to teach America's racist history?	https://www.vox.com/policy-and-politics/22464746/critical-race-theory-anti-racism-jarvis-givens
One Good Thing: Groundbreaking new novel Detransition, Baby lays bare the innermost thoughts of trans women	https://www.vox.com/22233339/detransition-baby-review-torrey-peters
More and more companies have monopoly power over workers' wages. That's killing the economy.	https://www.vox.com/the-big-idea/2018/4/6/17344880/wages-employers-workers-monopoly-growth-stagnation-inequality
Patricia Lockwood takes a scalpel to the extremely online brain in her excellent debut novel	https://www.vox.com/culture/22289358/no-one-is-talking-about-this-review-patricia-lockwood
Development plans threaten valuable Tartessos necropolis in southern Spain	https://english.elpais.com/culture/2021-06-11/development-plans-threaten-valuable-tartessos-necropolis-in-southern-spain.html

\*Αντέγραψα το κείμενο σε έναν text editor, καθώς είναι αρκετά μεγάλο σε πλάτος σε δεν μπορούσα να βγάλω screenshot απευθείας από το pycharm.

## 5) Αποθήκευση HTML

```
import requests
from pymongo import MongoClient

r = requests.get("https://example.com")
#print(r.text)







client = MongoClient("mongodb+srv://athinafus:<MYPASSWORD>.bv1w.mongodb.net/Lingtech?retryWrites=true&w=majority")
db = client['Article_Data']
collection = db['Articles']

#collection.delete_many({}) # deletes all entries

myresult = collection.find({}, {"_id": 0, "Article": 0}).limit(500)

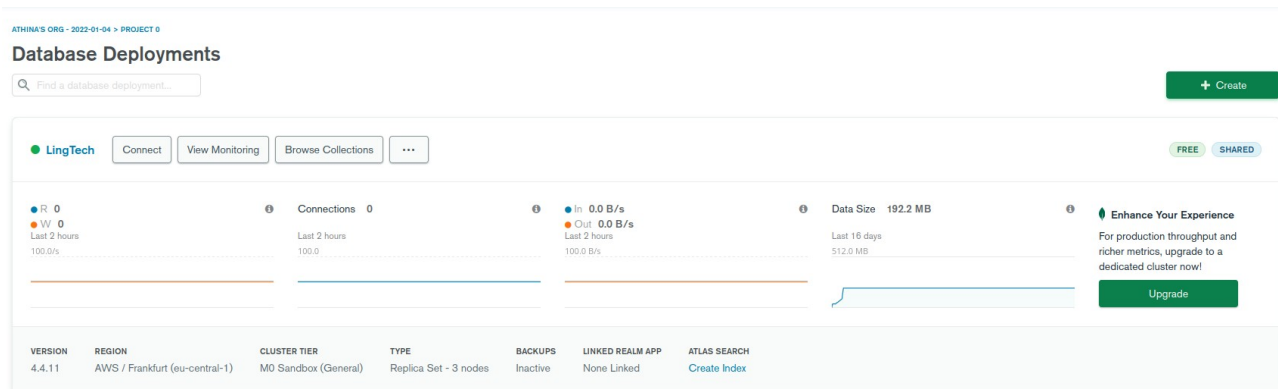
for x in myresult:
    print(x['Url'])
    r = requests.get(x['Url'])
    f = open("/home/athinafus/Documents/LinguisticTech/lingtech/lingtech/html_articles/{}.html"
            .format(x['Title'].replace(' ', '_'), "w"))
    f.write(r.text)
    f.close()
```

Με τον παραπάνω κώδικα, παίρνω τα urls των πρώτων 500 άρθρων που έχω αποθηκεύσει στη βάση και με την get() από τη βιβλιοθήκη requests παίρνω τον HTML κώδικα της ιστοσελίδας κάθε άρθρου και τον αποθηκεύω σε αρχείο html. Αποθηκεύω όλα τα αρχεία στον φάκελο html\_articles και κάθε αρχείο έχει ως όνομα τον τίτλο του εκάστοτε άρθρου.

					
Billionaires_are_racing_to_sidestep_President_Biden's_plan_to_raise_their_taxes.html	What_the_“Fauci_Gate”_emails_tell_us_about_Covid-19_and_American_politics.html	Democratic_voters_are_divided_on_whether_Biden_should_crack_down_on_Israel.html	Barack_Obama_sounds_the_alarm_about_America's_democratic_erosion.html	Ask_a_Book_Critic_Books_to_kick_off_the_new_year.html	TikTok's_Trump_problem_is_now_TikTok's_Biden_problem.html

## Βάση Δεδομένων

Για να ψτιάξω τη βάση δεδομένων μου χρησιμοποίησα τη MongoDB, και συγκεκριμένα την cloud εφαρμογή, MongoDB Atlas.



Η βάση μου ονομάζεται LingTech και αποτελείται από 2 collections: την Articles και την Lemma.

Η Articles περιέχει τα άρθρα που έχουμε κάνει scrape από τις ειδησεογραφικές ιστοσελίδες που έχουμε επιλέξει. Για κάθε άρθρο έχω αποθηκεύσει ένα id, τον τίτλο του, το περιεχόμενό του, το url του και το path του στον υπολογιστή μου.

Η Lemma περιέχει την πληροφορία του ανεστραμμένου αρχείου.

### Article\_Data.Articles

COLLECTION SIZE: 32.37MB TOTAL DOCUMENTS: 3749 INDEXES TOTAL SIZE: 76KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

FILTER { field: 'value' }

QUERY RESULTS 1-20 OF MANY

```
{
  "_id": "d1",
  "Url": "https://www.vox.com/22364056/promising-young-woman-oscars-best-picture...",
  "Path": "/home/athinafus/Documents/LinguisticTech/lingtech/vox_articles_3_1_2-2...",
  "Title": "Promising Young woman's explosive ending and Best Picture chances, exp...",
  "Article": " This year, eight films are in the running for Best Picture , the mos..."
}
```

```
{
  "_id": "d2",
  "Url": "https://www.vox.com/22346528/lord-of-the-rings-where-to-stream-rent-20...",
  "Path": "/home/athinafus/Documents/LinguisticTech/lingtech/vox_articles_3_1_2-2...",
  "Title": "One Good Thing: The Lord of the Rings trilogy is the perfect late-quar...",
  "Article": " There's a quote from Peter Jackson's 2001 film The Fellowship of the..."
}
```

### Article\_Data.Lemma

COLLECTION SIZE: 132.62MB TOTAL DOCUMENTS: 1799030 INDEXES TOTAL SIZE: 2709MB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

FILTER { field: 'value' }

QUERY RESULTS 1-20 OF MANY

```
{
  "_id": ObjectId("61d5564e48801221f2e17d08"),
  "Lemma": "long",
  "inDocument": "d17",
  "tfidf": 4.797184900831734
}
```

```
{
  "_id": ObjectId("61d5564e48801221f2e17dd4"),
  "Lemma": "long",
  "inDocument": "d9",
  "tfidf": 4.797184900831734
}
```

## B' ΜΕΡΟΣ

Σκοπός του Β μέρους της εργασίας είναι η υλοποίηση ενός συστήματος κατηγοριοποίησης κειμένων, τα οποία ανήκουν σε προκαθορισμένες θεματικές κατηγορίες.

Η συλλογή των κειμένων με τα οποία εργάστηκα είναι το **20 Newsgroups data set**. Συγκεκριμένα, επέλεξα τη δεύτερη δυνατή επιλογή που προσέφερε το site που μας υπέδειξε η εκφώνηση.

- [20news-19997.tar.gz](#) - Original 20 Newsgroups data set
- [20news-bydate.tar.gz](#) - 20 Newsgroups sorted by date; duplicates and some headers removed (18846 documents)
- [20news-18828.tar.gz](#) - 20 Newsgroups; duplicates removed, only "From" and "Subject" headers (18828 documents)

Το αρχείο αυτό περιείχε δύο φακέλους, έναν με τα κείμενα που θα χρησιμοποιούσα για training (την συλλογή E από την εκφώνηση) και έναν με τα κείμενα που θα χρησιμοποιούσα για testing (την συλλογή A από την εκφώνηση).

### ΠΡΟΕΠΕΞΕΡΓΑΣΙΑ ΚΕΙΜΕΝΩΝ

Χρειάστηκε να προεπεξεργαστώ τα κείμενα, έτσι ώστε να τα "καθαρίσω".

Διαβάζοντας κάποια από τα κείμενα, παρατηρήθηκε ότι είχαν μια σχεδόν standard μορφή, τουλάχιστον στην αρχή του περιεχομένου τους. Παρατήρησα ότι οι πρώτες γραμμές περιείχαν πληροφορία που δεν θα ήταν χρήσιμη για την κατηγοριοποίηση των κειμένων, οπότε αποφάσισα να την αφαιρέσω.

Ένα παράδειγμα κειμένου είναι:

**/home/athinafus/Documents/LinguisticTech/lingtech/lingtech/20news-bydate-test/comp.sys.ibm.pc.hardware/61114**

From: k4bnc@cbnewsh.cb.att.com (john.a.siegel)  
Subject: Can't set COM4 - G2Ks answer  
Organization: AT&T  
Distribution: usa  
Keywords: ATI conflict  
Lines: 14

Gateway service has confirmed my suspicion, echoed by a couple of people who responded to the original request for help. The ATI VLB video board uses the addresses for COM 4. They could suggest no work around. I will be returning the DF IO card they supplied for COM 4 (even though it could not possibly work) for credit against a bus mouse. This will free up the COM port I need - too bad the original salesman who suggested either the DF IO card or the bus mouse would solve my need for a port didn't know enough about the hardware.

Otherwise I must say that the 486DX2/66 system has worked very well - no problems with any other hardware or software.

John Siegel  
k4bnc@cbnewsh.att.com

Μέχρι και τη γραμμή που αναφέρεται το πλήθος των γραμμών, η πληροφορία δεν θα είναι χρήσιμη, οπότε, χρήσει regex, αφαιρώ αυτές τις γραμμές.

```
res = re.split("Lines: [1-9][0-9]", text) # removing lines before "Lines: <number of lines>
```



Κατόπιν, θέλω να εντοπίσω τα named entities στα κείμενα και να αφαιρέσω κάποια από αυτά. Για αυτό θα χρησιμοποιήσω το

```
nlp = en_core_web_sm.load()
```

από το spaCy.

```
doc = nlp(text)
if len(doc.ents) != 0: # if there are named entities in the text
    # -----[f(x) for x in sequence if condition]-----
    [people_names.append(X.text) for X in doc.ents
     if X.label_ in ['PERSON', 'QUANTITY', 'DATE', 'TIME', 'PERCENT', 'CARDINAL']]
else:
    print('No named entities in text')
name_regex = re.compile(r'\b%s\b' % r'\b|\b'.join(map(re.escape, reversed(people_names))))
text = name_regex.sub("", text) # removing the names from the text
text = re.sub(r'^\w|', ' ', text) # removing symbols and empty lines
text = text.replace("-", ' ')
text = big_regex.sub("", text) # removing the stopwords from the text
text = ''.join([i for i in text if not i.isdigit()]) # removing numbers from the text
text = re.sub(' +', ' ', text) # removing too many spaces from text
```

Στην μεταβλητή doc περνάω τις named entities του υπό επεξεργασία κειμένου. Εφόσον υπάρχουν named entities στο κείμενο, κρατάω μόνο όσες ανήκουν στις κατηγορίες PERSON, QUANTITY, DATE, TIME, PERCENT και CARDINAL και τις περνάω στην λίστα people\_names, χρήσιμη list comprehension. Εάν δεν υπάρχουν named entities, το εκτυπώνω. Στη συνέχεια, χρησιμοποιώντας regex αφαιρώ τις προαναφερθείσες named entities. Έπειτα, αφαιρώ σύμβολα και κενές γραμμές. Τέλος, αφαιρώ τα stopwords:

```
prohibitedWords = stopwords.words('english')
big_regex = re.compile(r'\b%s\b' % r'\b|\b'.join(map(re.escape, reversed(prohibitedWords))))
```

Έχω χρησιμοποιήσει τα stopwords από το nltk.corpus. Τέλος, αφαιρώ αριθμούς που μπορεί να έχουν ξεφύγει και πολλά συνεχόμενα κενά.

## ΔΗΜΙΟΥΡΓΙΑ ΧΩΡΟΥ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ

Για τον υπολογισμό των βαρών tf-idf των θεμάτων (stems) όλων των λέξεων, όλων των εγγράφων της συλλογής χρησιμοποίησα την TfidfVectorizer (από την βιβλιοθήκη scikit-learn), η οποία μετατρέπει μια συλλογή κειμένων σε ένα matrix από tf-idf βάρη για κάθε λέξη των κειμένων.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Επειδή η εκφώνηση ζητούσε να πάρουμε τα stems κάθε λέξης, έκανα overriding στον build\_analyser() για την TfidfVectorizer.

```
class StemmedTfidfVectorizer(TfidfVectorizer):
    def build_analyzer(self):
        analyzer = super(StemmedTfidfVectorizer, self).build_analyzer()
        return lambda doc: ([snowball.stem(w) for w in analyzer(doc)])
```

```
stfidfvectorizer = StemmedTfidfVectorizer(max_features=4000)
```

\*το max\_features=4000 είναι παράδειγμα.

Για την επιλογή του μεγέθους του χώρου των χαρακτηριστικών θα γίνουν δοκιμές για την ακρίβεια της κατηγοριοποίησης.

Επειδή η διαδικασία διαβάσματος των διαφόρων εγγράφων είναι σημαντικά χρονοβόρα, για να εξοικονομώ χρόνο, αποθηκεύω τα διανύσματα σε dataframes της βιβλιοθήκης pandas, ένα train κι ένα test dataframe, και τα αποθηκεύω σε αρχεία pickle, χρησιμοποιώντας την εντολή to\_pickle().

```
sdf_tfidfvect_TEST.to_pickle('sdf_tfidfvect_test.pickle')
```

Έχω αποθηκεύσει σε αρχεία τα dataframes από τα οποία φτιάχνω τα διανύσματα για κάθε έγγραφο. Έχω 4 διαφορετικές εκδόσεις, ανάλογα με το μέγεθος του διανυσματικού χώρου: 2000 χαρακτηριστικά, 4000, 8000 και 10000.

Για την κατηγοριοποίηση έχω χρησιμοποιήσει 3 διαφορετικές τεχνικές:

1. Χρήση αλγορίθμων machine learning
  1. Naive Bayes Classifier
  2. Support Vector Machine
2. Χρήση νευρωνικού δικτύου
3. Χρήση μετρικών σχετικότητας
  1. Cosine Similarity
  2. Manhattan Distance
  3. Euclidean Distance

Για κάθε διανυσματικό χώρο χαρακτηριστικών θα μετρήσω την ακρίβεια κάθε τεχνικής.

ACCURACIES						
Model	max_features	2000(1753)	4000(3426)	8000(6439)	10000(7845)	12000(9187)
Naive Bayes		71.08	74.53	76.3	76.86	76.93
Linear Kernel SVM		70.12	75.0	76.9	77.44	77.56
RBF Kernel SVM		71.62	75.7	77.05	77.78	77.84
Neural Network		61.15	65.53	66.20	68.26	67.51
Cosine Similarity		67.17	70.92	72.51	72.81	73.17
Manhattan Dist.		17.75	15.35	14.42	13.41	13.37
Euclidean Dist.		65.37	68.94	70.20	70.27	70.31

Για να μπορέσω να κάνω την κατηγοριοποίηση, έπρεπε τα διανύσματα που αντιστοιχούν στα διαφορετικά κείμενα να ανήκουν στον ίδιο διανυσματικό χώρο. Συνεπώς, έχοντας φτιάξει τον διανυσματικό χώρο για τη συλλογή E και τον διανυσματικό χώρο για την συλλογή A,

κρατάω την τομή μεταξύ των δύο συνόλων όρων. Το τελικός πλήθος όρων είναι μέσα στην παρένθεση.

```
# =====INTERSECTIONING THE STEMS IN EACH DATAFRAME=====
stems_intersection = list(set(sdf_tfidfvecr_TRAIN.columns) & set(sdf_tfidfvecr_TEST.columns)) # find intersection of columns

sdf_tfidfvecr_TRAIN = sdf_tfidfvecr_TRAIN.reindex(stems_intersection, axis=1).dropna(how='all', axis=1)
sdf_tfidfvecr_TEST = sdf_tfidfvecr_TEST.reindex(stems_intersection, axis=1).dropna(how='all', axis=1)

arr_TEST = sdf_tfidfvecr_TEST.values
arr_TRAIN = sdf_tfidfvecr_TRAIN.values

sdf_tfidfvecr_TEST['Vector']_ = pd.DataFrame([[i] for i in arr_TEST]) # _____ +++
sdf_tfidfvecr_TRAIN['Vector']_ = pd.DataFrame([[i] for i in arr_TRAIN]) # _____ $$$
```

### **Naive Bayes Classifier**

Για την κατηγοριοποίηση με τον αλγόριθμο Naive Bayes χρησιμοποίησα τον MultinomialNB(). Χρησιμοποίησα τα διανύσματα με τα tf-idf βάρη ως X, και η κατηγορία/class που άνηκε κάθε κειμένο ήταν αυτό που ήθελα να προβλέψει ο αλγόριθμος, ήταν το target variable. Επειδή το class ήταν categorical variable, εφάρμοσα Label Encoding, για να μπορεί να τη διαχειριστεί ο Naive Bayes.

```
Train_X = sdf_tfidfvecr_TRAIN["Vector"].tolist()
Train_Y = sdf_tfidfvecr_TRAIN["Class"].tolist()

Test_X = sdf_tfidfvecr_TEST["Vector"].tolist()
Test_Y = sdf_tfidfvecr_TEST["Class"].tolist()

Encoder = LabelEncoder() # to encode the target variable, because it's string and the model will not understand it
Train_Y = Encoder.fit_transform(Train_Y)

Test_Y = Encoder.transform(Test_Y)

# fit the training dataset on the NB classifier
Naive = naive_bayes.MultinomialNB()
Naive.fit(Train_X, Train_Y) # predict the labels on validation dataset
predictions_NB = Naive.predict(Test_X) # Use accuracy_score function to get the accuracy
```

### **Support Vector Machine**

Για τον αλγόριθμο Support Vector Machine δοκίμασα δύο διαφορετικούς Kernels, τον Linear Kernel και τον RBF Kernel. Για τον Linear, χρησιμοποίησα παραμετροποίηση που προτινόταν στο διαδίκτυο, ενώ για τον RBF χρησιμοποίησα cross validation και κατέληξα στην συγκεκριμένη παραμετροποίηση για το C και gamma.

Cross Validation:

```
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001], 'kernel': ['rbf']}
grid = GridSearchCV(svm.SVC(), param_grid, refit=True, verbose=3)
# fitting the model for grid search
grid.fit(Train_X, Train_Y)
print(grid.best_params_)
print(grid.best_estimator_)
```

Οι αλγόριθμοι:

```
SVM = svm.SVC(C=1.0, kernel='linear', degree=3, max_iter=3000)
print('Created the Linear Kernel SVM')
SVM.fit(Train_X, Train_Y) # predict the labels on validation dataset
print('Fitted the Linear Kernel SVM')
predictions_SVM = SVM.predict(Test_X) # Use accuracy_score function to get the accuracy
```

```
SVM = svm.SVC(C=10.0, gamma=1.0)
print('Created the RBF Kernel SVM')
SVM.fit(Train_X, Train_Y) # predict the labels on validation dataset
print('Fitted the RBF Kernel SVM')
predictions_SVM = SVM.predict(Test_X) # Use accuracy_score function to get the accuracy
```

\*Ο RBF Kernel είναι η default επιλογή, για αυτό και δεν φαίνεται ως όρισμα στο προηγούμενο screenshot.

## **Neural Network**

Για το νευρωνικό δίκτυο χρησιμοποίησα ένα απλό sequential model. Ως activation function χρησιμοποίησα τη LeakyRelu στα hidden layers, καθώς έχει όλα τα πλεονεκτήματα της ReLu, και καταπολεμά διάφορα από τα μειονεκτήματά της, όπως ότι δεν προκαλεί saturation σε νευρώνες του δικτύου, είναι εύκολη να υπολογιστεί και είναι πιο κοντά σε zero-centered functions από την ReLu. Επιπλέον, χρησιμοποίησα και κάποια Dropout Layers, για να αποφύγω την υπερεκπαίδευση του μοντέλου. Τέλος, στο εξωτερικό layer από νευρώνες (20, όσες και οι διαφορετικές κατηγορίες εγγράφων) χρησιμοποίησα την softmax, καθώς είναι η πλέον καταλληλότερη για το πρόβλημα του multiclass classification που είχα να επιλύσω.

```
model = Sequential()
model.add(Dense(12, input_dim=7845, activation='relu'))
model.add(layers.Dense(80)) # no activation here
model.add(layers.LeakyReLU(alpha=0.3)) # activation layer here instead
model.add(layers.Dropout(0.2))
model.add(layers.Dense(60)) # no activation here
model.add(layers.LeakyReLU(alpha=0.3)) # activation layer here instead
model.add(layers.Dropout(0.2))
model.add(layers.Dense(20, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=[['accuracy']])
```



Ως loss function χρησιμοποίησα την sparse categorical crossentropy, γιατί θεωρείται η πιο κατάλληλη για το multiclass classification.

### **Cosine Similarity, Manhattan & Euclidean Distance**

Για αυτές τις μετρικές χρησιμοποίησα την έτοιμη συνάρτηση από την scikit-learn και τη βιβλιοθήκη scipy:

```
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial import distance
```

```
cs = cosine_similarity(row_test['Vector'].reshape(1, -1), vec_train.reshape(1, -1))
dst_m = distance.cityblock(row_test['Vector'], vec_train)
dst_e = distance.euclidean(row_test['Vector'], vec_train)
```

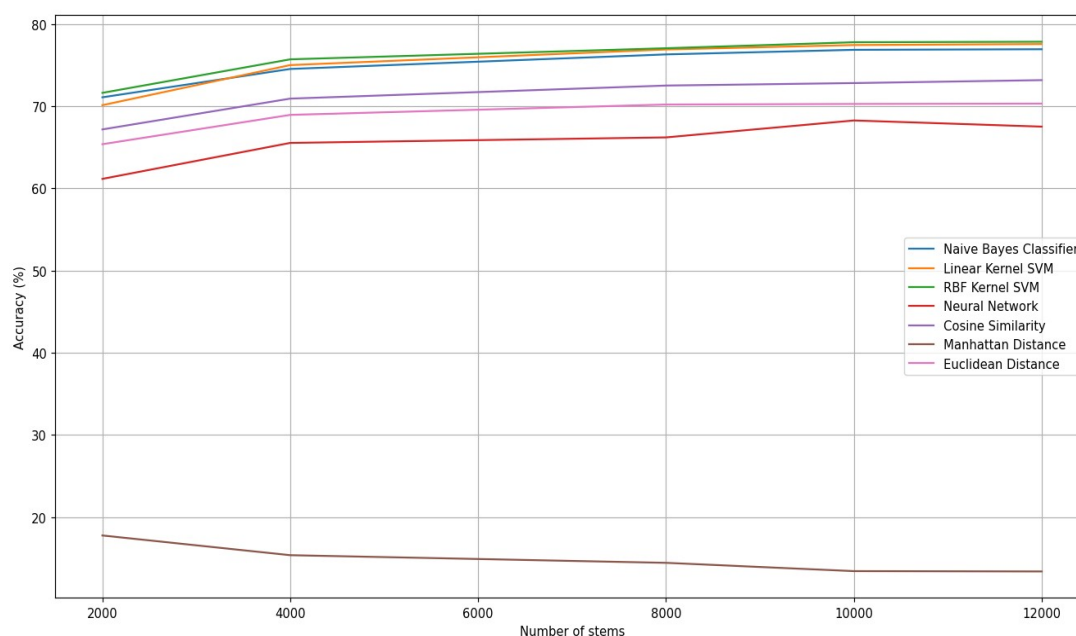
Για να μην συγκρίνω όλα τα διανύσματα της συλλογής E με την συλλογή A, έφτιαξα ένα διάνυσμα για κάθε κατηγορία/class για την συλλογή E και συνέγκρινα με αυτά τα διανύσματα της A. Αυτά τα διανύσματα-εκπρόσωποι της κάθε κατηγορίας, τα έφτιαξα χρησιμοποιώντας τους M.O. των βαρών tf-idf που είχε για κάθε λέξη/θέμα η κάθε κατηγορία.

```
sdf_tfidfvect_TRAIN_mean = sdf_tfidfvect_TRAIN.groupby(['Class']).mean()
```

Χρησιμοποίησα την groupby() για να χωρίσω ανά κατηγορία τις εγγραφές στο dataframe και στη συνέχεια εφάρμοσα τη συνάρτηση mean() για να πάρω τον M.O..

Δημιούργησα 3 νέα dataframes και σε κάθε ένα από αυτά αποθήκευσα τα αποτελέσματα της σύγκρισης για 1000 κείμενα της συλλογής A.

### **Συμπεράσματα**



Από τον πίνακα με την ακρίβεια για κάθε μέθοδο που χρησιμοποίησα, παρατηρούμε ότι καλύτερη απόδοση είχαν οι αλγόριθμοι SVM, και συγκεκριμένα το μοντέλο με τον RBF Kernell.

Παρατηρούμε ότι όταν τα max\_features ξεπεράσουν τα 10.000, δεν έχουμε σημαντική αύξηση της ακρίβειας, κάτι που με κάνει να πιστεύω ότι δεν χρειάζεται να αυξήσουμε την πολυπλοκότητα αυξάνοντας τον αριθμό των features, καθώς δεν θα έχουμε σημαντικό κέρδος. Στην περίπτωση του νευρωνικού δικτύου, μάλιστα, έχουμε μείωση της ακρίβειας.

Επιπλέον, βλέπουμε ότι όσο αυξανόταν το πλήθος των όρων, η απόσταση Manhattan ως μετρική είχε χειρότερη απόδοση, ενώ όλες οι άλλες είχαν ανοδική πορεία. Γενικά, η απόσταση Manhattan δεν έδινε καλό accuracy, οπότε δεν θα τη πρότεινα ως μέτρο σύγκρισης για την επίλυση ενός προβλήματος multiclass classification.

Καλύτερη απόδοση είχαν οι αλγόριθμοι SVM, αλλά ήταν και οι πιο ακριβοί σε υπολογισμούς και χρόνο. Ο Naive Bayes δίνει παρόμοια αποτελέσματα και είναι πολύ λιγότερο computationally expensive.

Το νευρωνικό δίκτυο είχε μέτρια απόδοση, κάτι που με κάνει να πιστεύω ότι θα μπορούσε να γίνει καλύτερη παραμετροποίηση, όσον αφορά τα dropout layers, τις activation functions και το πλήθος των layers.

## **BIBΛΙΟΘΗΚΕΣ**

Για την υλοποίηση του Β' μέρους χρησιμοποίησα τις εξής βιβλιοθήκες:

```
import pandas as pd
import os
import re
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem import SnowballStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
import en_core_web_sm
```

- pandas: Χρησιμοποίησα dataframes για το manipulation των δεδομένων μου
- os: Την χρειάστηκα για την προσπέλαση φακέλων με τα αρχεία που περιείχαν τα κειμενικά δεδομένα.
- Nltk: Την χρησιμοποίησα για να έχω πρόσβαση στη λίστα με τα stopwords και για τον stemmer.
- CountVectorizer και TfidfVectorizer: Για την μετατροπή των εγγράφων σε πίνακες που περιέχουν το πλήθος εμφάνισης κάθε λέξης σε ένα έγγραφο και το βάρος tf-idf για κάθε λέξη σε ένα έγγραφο, αντίστοιχα.
- en core web sm: Το χρησιμοποίησα για να αφαιρέσω διάφορα named entities που μπορεί να υπήρχαν σε κάθε έγγραφο, τα οποία δεν είχαν σημασιολογική αξία που θα βοηθούσε στην κατηγοριοποίηση των κειμένων.

```
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial import distance
from sklearn.metrics import accuracy_score
```

- Χρησιμοποίησα τις παραπάνω μετρικές για τις συγκρίσεις των διανυσμάτων/εγγράφων για την διαδικασία της κατηγοριοποίησης.

```
from sklearn import naive_bayes, svm
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.model_selection import GridSearchCV
```

- naive\_bayes, svm: Για τη δημιουργία των κατηγοριοποιητών.
- LabelEncoder: Η target variable για την κατηγοριοποίηση ήταν categorical και οι κατηγοριοποιητές δεν θα μπορούσαν να τη διαχειριστούν. Συνεπώς, χρησιμοποίησα label encoding για να αντιστοιχίσω κάθε κατηγορία εγγράφου σε έναν ακέραιο αριθμό.
- Sklearn.metrics: Για τις μετρικές που θα αξιολογούσα τους κατηγοριοποιητές.
- Sklearn.model\_selection: Για να βελτιστοποιήσω τον SVM κατηγοριοποιητή με RBF Kernel, έκανα cross validation, οπότε χρησιμοποίησα την GridSearchCV.

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras import layers
```

- Keras.models: Για να ορίσω τον τύπο του νευρωνικού δικτύου, το οποίο ήταν Sequential.
- Keras.layers: Για να ορίσω τον τύπο των layers του νευρωνικού δικτύου → Dense.
- LabelEncoder: Για τον ίδιο λόγο που αναφέρθηκε και παραπάνω.
- Tensorflow.keras layers: Για να ορίσω τα layers στο νευρωνικό δίκτυο.
- Matplotlib: Για τη δημιουργία του γραφήματος με τα accuracies για κάθε μέθοδο κατηγοριοποίησης.

```
import matplotlib.pyplot as plt

plt.plot([2000, 4000, 8000, 10000, 12000], [71.08, 74.53, 76.3, 76.86, 76.93], label='Naive Bayes Classifier')
plt.plot([2000, 4000, 8000, 10000, 12000], [70.12, 75.0, 76.9, 77.44, 77.56], label='Linear Kernel SVM')
plt.plot([2000, 4000, 8000, 10000, 12000], [71.62, 75.7, 77.05, 77.78, 77.84], label='RBF Kernel SVM')
plt.plot([2000, 4000, 8000, 10000, 12000], [61.15, 65.53, 66.20, 68.26, 67.51], label='Neural Network')
plt.plot([2000, 4000, 8000, 10000, 12000], [68.6, 72.0, 73.3, 73.9, 74.7], label='Cosine Similarity')
plt.plot([2000, 4000, 8000, 10000, 12000], [52.0, 47.4, 44.6, 42.2, 41.8], label='Manhattan Distance')
plt.plot([2000, 4000, 8000, 10000, 12000], [65.8, 70.6, 71.0, 71.1, 71.2], label='Euclidean Distance')
plt.ylabel('Accuracy (%)')
plt.xlabel('Number of stems')
plt.legend()
plt.grid(True)
plt.show()
```

## Αρχεία

### A' Μέρος

- **Αρχεία κώδικα**
  1. **make\_inv\_file.py**: Φτιάχνω το ανεστραμμένο αρχείο.
  2. **queries\_to\_invFile.py**: Ερωτήματα προς το ανεστραμμένο αρχείο, διαβάζοντας λέξεις από αρχεία.
  3. **queries\_to\_invFile\_from\_user.py**: Ερωτήματα προς το ανεστραμμένο αρχείο από τον χρήστη.
  4. **get\_HTML\_code.py**: Παίρνω HTML κώδικα από την ιστοσελίδα κάθε άρθρου.
  5. **articles\_spider.py**: Spider για crawling στην El Pais.
  6. **vox\_spider.py**: Spider για crawling στην Vox.
- **Αρχεία αποτελεσμάτων**
  1. **inversed\_file\_20\_jan.xml**: Αρχείο xml που περιέχει το ανεστραμμένο αρχείο.
  2. **inversed\_file\_20\_jan.pkl**: Αρχείο pickle που περιέχει το ανεστραμμένο αρχείο.
  3. **output\_20\_jan.txt**: Αρχείο txt που περιέχει το ανεστραμμένο αρχείο.
  4. **pos\_tagged\_articles.csv**: Αρχείο που περιέχει τα άρθρα με PoS tags.
  5. **output\_to\_user\_query.txt**: Αρχείο txt που περιέχει την έξοδο του συστήματος όταν ο χρήστης κάνει ένα query στο ευρετήριο.
  6. **The\_Queen's\_Gambit\_beauty\_debate,\_explained.html**: Παράδειγμα καταβασμένου κώδικα HTML ιστοσελίδας άρθρου.
  7. **four\_word\_queries.txt, three\_word\_queries.txt, two\_word\_queries.txt**: Αρχεία για την υποβολή ερωτημάτων στο ανεστραμμένο αρχείο.

### B' Μέρος

- **Αρχεία κώδικα**
  1. **partB.py**: Κώδικας για τη δημιουργία των πινάκων με τα διανύσματα με τα βάρη tf-idf για κάθε document
  2. **partB\_withFiles.py**: Κώδικας με τα μοντέλα μηχανικής μάθησης Naive Bayes και SVM, όπου διαβάζει τα δεδομένα από τα αρχεία με τους πίνακες διανυσμάτων που δημιούργησε το παραπάνω αρχείο.
  3. **partB\_ekfwnhsh.py**: Κώδικας για τη σύγκριση διανυσμάτων με την cosine similarity, manhattan distance και euclidean distance.
  4. **partB\_neural\_network.py**: Κώδικας για τη δημιουργία και εκπαίδευση νευρωνικού δικτύου.
  5. **graphs\_for\_evaluation.py**: Κώδικας για τη δημιουργία γραφήματος με τα accuracies ανά μέθοδο κατηγοριοποίησης και ανά μέγεθος διανυσματικού χώρου.
- **Αρχεία αποτελεσμάτων**: Όλα τα ακόλουθα αρχεία pickle περιέχουν τους πίνακες/dataframes με τα διανύσματα με τα tf-idf βάρη για τα stems του κάθε document.
  1. **sdf\_tfidfvect\_test12000.pickle**
  2. **sdf\_tfidfvect\_test10000.pickle**
  3. **sdf\_tfidfvect\_test8000.pickle**
  4. **sdf\_tfidfvect\_test4000.pickle**
  5. **sdf\_tfidfvect\_test2000.pickle**
  6. **sdf\_tfidfvect\_train12000.pickle**



7. `sdf_tfidfvect_train10000.pickle`
8. `sdf_tfidfvect_train8000.pickle`
9. `sdf_tfidfvect_train4000.pickle`
10. `sdf_tfidfvect_train2000.pickle`
11. `svm_cross_validation_results.txt`: Αρχείο που περιέχει τα αποτελέσματα από το cross validation για το μοντέλο SVM.