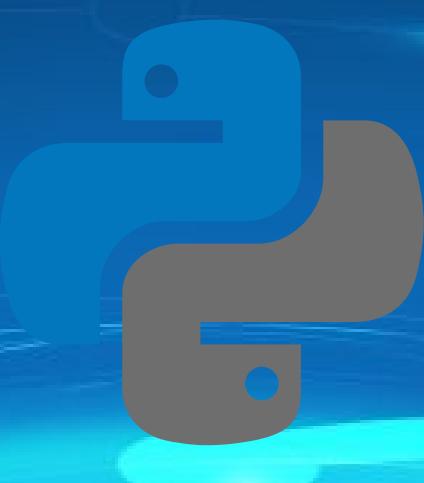




Topic Modeling Report



Report progetto

Web Analytics e Analisi Testuale

Topic modeling: Analisi delle tendenze della stampa italiana e statunitense nell'ultimo triennio, e impatto sulle notizie del verificarsi di due grandi eventi covid-19 e invasione russa in Ucraina



A.a. 2021/2022
Corso di studi: Network Economy

Prof. Stefano Matta

*Davide Locci 11/82/00197
Alessandro Piroddi 11/82/00205*

Indice

Introduzione

Analisi

Estrazione dei tweets

File main.py

File Scraping.py

Pre-processamento dei tweets

File main.py

File Italian_cleaning.py

File English_cleaning.py

Analisi preliminari

File main.py

File topwords.py

File word_trends.py

File tweets_reweets_likes.py

Topic modeling

File main.py

File topic_skl.py

File topic_gsm.py

Conclusioni

Introduzione

Obiettivo del progetto

Il progetto prevede di prendere in considerazione tre diversi periodi temporali:

- Il periodo antecedente al diffondersi della pandemia da COVID-19.
- Il periodo di tempo che è intercorso dal diffondersi della pandemia all'annuncio di invasione delle forze russe in Ucraina.
- Il periodo che intercorre tra l'annuncio dell'invasione russa in Ucraina fino ai giorni attuali.

Sulla base di questi 3 periodi temporali, l'obiettivo è quello di confrontare, al verificarsi dei due grandi eventi che hanno segnato l'ultimo triennio (appunto diffusione della pandemia e invasione russa) come e in quale misura si siano modificati gli argomenti trattati dalla stampa italiana. Eventualmente l'obiettivo è anche quello di cercare di capire se, e in quale misura, ci sia una tendenza a polarizzare l'attenzione su una singola tematica, quella più in voga, cannibalizzando tutte le altre.

Strumento di analisi

Lo strumento di analisi scelto per raggiungere il nostro obiettivo, è la Topic Modeling, che ci permetterà di rilevare i topic delle notizie per i tre diversi periodi temporali presi in considerazione, e di confrontarli.

Fonte testuale

Abbiamo scelto di procedere nel recuperare le fonti testuali della stampa italiana dal social network Twitter. Prenderemo in considerazione tre testate giornalistiche, in modo tale da considerarne una per ogni orientamento politico (sinistra, centro, destra). I tweet verranno estratti dagli account Twitter ufficiali delle tre testate giornalistiche prese in considerazione (La Repubblica, Corriere della Sera, Il Giornale). Al momento dell'estrazione, considereremo tutti i tweets di ognuna delle tre testate, senza filtrare per argomento, in modo tale che sia poi la topic modeling a stabilire quelli che sono i topic affrontati dalla stampa.

Abbiamo ritenuto inoltre essere interessante ripetere l'analisi con un'altra nazione, nel caso specifico gli Stati Uniti. In questo modo si potrebbe confrontare se, le tendenze dei media italiani in tema di argomenti trattati nei tre diversi periodi presi in considerazione, corrispondono con quanto avviene negli USA e rispondere a domande come: Quali erano i topic principali in Italia e negli USA prima dello scoppio della pandemia? Ci sono differenze? Quali sono i topic principali in Italia e negli USA nel periodo di diffusione della pandemia? Rispetto a quanto accadeva prima, c'è stato un appiattimento delle differenze degli argomenti trattati? Con quali differenze, in termini di risonanza mediatica, è percepito lo scoppio della guerra in Italia e negli USA? Ad esempio negli Stati Uniti potrebbe esserci un maggior distacco e una minore percezione di quanto sta accadendo dovuta alla lontananza geografica dalla zona del conflitto.

Analisi

Si è scelto di suddividere l'analisi in quattro step:

- Estrazione dei tweet necessari allo svolgimento del progetto
- Pulizia dei tweet estratti
- Analisi preliminari alla topic modeling con conteggi e grafici
- Topic modeling

Per ognuno di questi step è stato creato un branch; i quattro branch risultano così nominati: tweets-scraping, tweets-pre-processing, tweets-preliminar-analysis, tweets-topic-modeling.

Nello svolgimento del codice, si è scelto di utilizzare un paradigma di programmazione orientata agli oggetti. Abbiamo ritenuto fondamentale l'utilizzo delle [classi](#) e degli [oggetti](#), in quanto si presta al meglio con la natura stessa del nostro progetto.

Dovendo infatti analizzare cinque diversi giornali e tre diversi periodi temporali, accadrà in maniera ricorrente, per le varie operazioni da svolgere, di dover utilizzare i medesimi metodi per ciascuno dei cinque giornali e per ciascuno dei tre periodi. Utilizzando una soluzione ad oggetti, sarà possibile creare una classe che svolga di volta in volta tutti i compiti di cui avremo bisogno, chiamando ogni volta lo specifico oggetto di quella classe relativo al singolo giornale ed al singolo periodo.

Estrazione dei tweets: branch Tweets Scraping

Il primo branch è quello relativo all'estrazione dei tweet. Come anticipato nell'introduzione, l'obiettivo è quello di estrarre i tweet relativi ai seguenti periodi temporali: fase antecedente il diffondersi del covid19, fase che intercorre tra il diffondersi del covid19 e l'invasione russa in Ucraina, fase che intercorre tra l'invasione russa ed i giorni attuali.

Nel codice sono state etichettate queste tre fasi rispettivamente come: precovid, covid, war.
Di seguito le date che contraddistinguono ogni periodo:

- **precovid**: dal 01-07-19 al 31-12-19

Note: tale data di inizio è stata scelta al fine di estrarre i tweet relativi ai sei mesi antecedenti al 2020, anno nel quale inizia a diffondersi il virus già dai primissimi mesi.

- **covid**: dal 01-01-20 al 31-01-22

Note: si è optato per il 01-01-20 in quanto questa data è immediatamente successiva al 31-12-19, giorno in cui c'è stata l'ammissione dei primi casi di covid e la segnalazione della Commissione Sanitaria Municipale di Wuhan all'Organizzazione Mondiale della Sanità (OMS).

- **war**: dal 01-02-22 al 17-05-21

Note: Abbiamo scelto come data di inizio il 01-02-22, ovvero l'inizio del mese di febbraio. L'offensiva militare delle forze armate russe è infatti iniziata ufficialmente il 24 febbraio 2022, ma già da diverse settimane, l'argomento aveva attirato l'attenzione mediatica.

Come sopracitato, sono stati presi in considerazione i tweet pubblicati dai profili ufficiali di tre giornali italiani e due statunitensi.

Per quanto riguarda la scelta dei giornali italiani, questi sono stati selezionati secondo il proprio orientamento politico (sinistra, centro, destra).

I giornali scelti per ogni orientamento sono Repubblica, Il Corriere della Sera ed Il Giornale, e sono stati selezionati in base ai seguenti criteri:

- tiratura
- seguito su Twitter
- frequenza di pubblicazione su Twitter

Mentre i due giornali americani, il New York Times e il Wall Street Journal, sono stati selezionati in base a seguito e frequenza di pubblicazione su Twitter.

Per l'estrazione dei tweet ci siamo trovati di fronte a due possibili alternative.

La prima alternativa consiste nell'utilizzo delle [WEB API](#) ufficiali di Twitter, la seconda invece prevede di utilizzare delle librerie non ufficiali. La scelta è ricaduta sull'utilizzo delle librerie non ufficiali, e nello specifico della libreria [Snscreape](#).

Le API ufficiali di Twitter infatti prevedono tre grossi limiti:

- occorre effettuare una richiesta formale in modo tale da ottenere le chiavi che permettano di accedere allo scraping dei tweet.
- la richiesta formale dovrà essere esaminata da persona fisica e ciò comporterebbe una dilatazione in termini di tempo di attesa più lungo rispetto all'utilizzo delle librerie non ufficiali.
- è possibile estrarre tweet antecedenti solamente a sette giorni dalla data in cui si estrae.

In particolar modo la limitazione temporale dei sette giorni ci ha fatto decisamente propendere su snscreape: per la nostra analisi è infatti fondamentale ottenere tweet che vadano a ritroso anche di tre anni antecedenti la data di estrazione.

Sappiamo che le API di Twitter prevedono dei [rate limits](#) relativi allo scraping, consultabili nella documentazione di Twitter. Il superamento di questi limiti porta l'utente ad essere "blacklistato" e inibito dall'estrarrre tweet per un determinato periodo di tempo.

Per quanto riguarda snscreape invece, non esistono documenti ufficiali che attestino l'esistenza di rate limits, tuttavia, sul web è possibile trovare diverse testimonianze di utenti che sono stati soggetti a blocchi temporali in seguito all'estrazione di grandi mole di tweets.

Per questo, nel codice si è deciso di permettere all'utente, al raggiungimento di una data soglia di tweet estratti, di interrompere per un determinato intervallo di tempo l'estrazione dei tweet. Sarà l'utente stesso, in base a quanto sceglie di cautelarsi, ad impostare il limite di tweet da raggiungere prima di sospendere l'estrazione, e sarà sempre l'utente a scegliere per quale intervallo sospendere l'estrazione.

Ulteriormente, è possibile impostare un intervallo di sospensione anche tra la chiamata ad un oggetto ed un altro, nel caso si decidesse di chiamare più oggetti uno di seguito all'altro.

Il tutto viene effettuato grazie al modulo time ed alla sua funzione [sleep\(\)](#), che ci permette di bloccare l'esecuzione del codice specificando come argomento i secondi di interruzione.

Il branch Tweets Scraping si compone, oltre che dei file [requirements.txt](#) e [README.md](#), dei seguenti file:

- main.py
- scraping.py

File main.py

Dal file main partiranno le chiamate alla nostra classe Tweetscraper, che si occuperà dell'estrazione dei tweets. Chiameremo la classe Tweetscraper per ogni specifico giornale e per ognuno dei tre periodi precedentemente indicati.

Nella figura seguente è possibile vedere le prime due chiamate (relative al giornale Repubblica ed ai periodi precovid e covid):

```
#####
# scraping features

newspaper = 'repubblica'
date_start, date_finish = '2020-01-01', '2022-01-31'
period = 'covid'
tweets2sleep = 3200
sleeping_time_internal = 1800
sleeping_time_external = 1800
lang = 'it'

##### scraping call

df_repubblica_covid = Tweetscraper(newspaper=newspaper, tweets2sleep=tweets2sleep,
                                      sleeping_time=sleeping_time_internal,
                                      start=date_start, end=date_finish,
                                      path=f'./scraped_tweets/tweets_{period}',
                                      period=period, lang=lang).dump_tweets()

print(df_repubblica_covid.head(5))

##### sleeping

print('to scrape other tweets please wait, i am sleeping')
time.sleep(sleeping_time_external)
print('scraping tweets...')

#####
# scraping features

newspaper = 'repubblica'
date_start, date_finish = '2020-01-01', '2022-01-31'
period = 'covid'
tweets2sleep = 3200
sleeping_time_internal = 1800
sleeping_time_external = 1800
lang = 'it'

##### scraping call

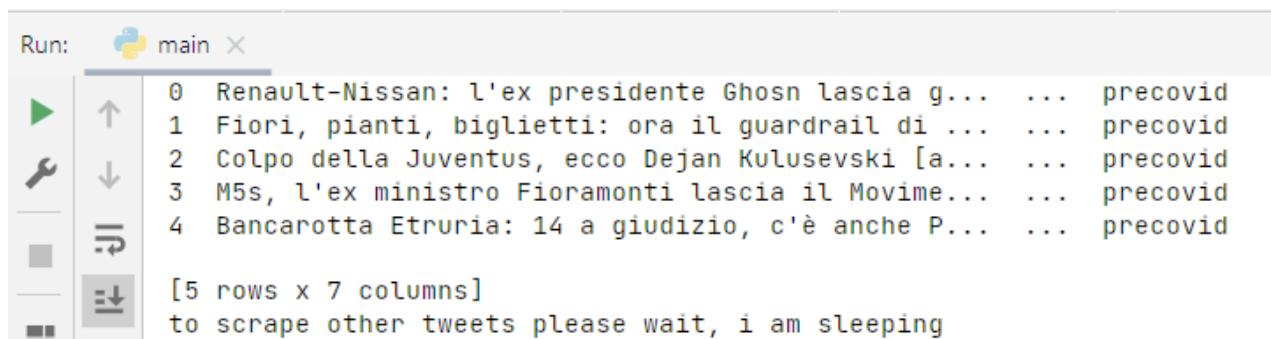
df_repubblica_covid = Tweetscraper(newspaper=newspaper, tweets2sleep=tweets2sleep,
                                      sleeping_time=sleeping_time_internal,
                                      start=date_start, end=date_finish,
                                      path=f'./scraped_tweets/tweets_{period}',
                                      period=period, lang=lang).dump_tweets()

print(df_repubblica_covid.head(5))

##### sleeping

print('to scrape other tweets please wait, i am sleeping')
time.sleep(sleeping_time_external)
print('scraping tweets...')
```

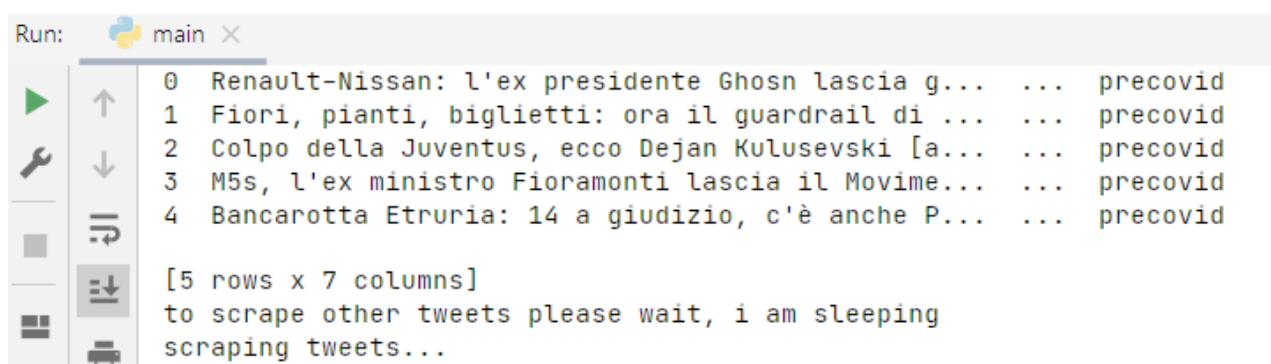
Una volta partita la chiamata, verrà stampata la frase “i am scraping”; una volta terminata l'estrazione relativa all'oggetto chiamato, come anticipato, si attiva il modulo time con la funzione `sleep()`, che ci permette di bloccare il codice per un numero di secondi pari a quelli specificati come argomento. Di seguito l'output nel momento in cui viene terminato il codice relativo alla chiamata di uno specifico oggetto:



```
Run: main ×
[5 rows x 7 columns]
to scrape other tweets please wait, i am sleeping
```

The screenshot shows a Jupyter Notebook cell titled "main". The code has printed five tweets from the Twitter API. Below the tweets, there is a message: "to scrape other tweets please wait, i am sleeping". This indicates that the program is currently paused due to the sleep function.

Di seguito l'output nel momento in cui, terminato il codice relativo alla chiamata di uno specifico oggetto, il tempo di sospensione si esaurisce e viene chiamato l'oggetto successivo:



```
Run: main ×
[5 rows x 7 columns]
to scrape other tweets please wait, i am sleeping
scraping tweets...
```

The screenshot shows a Jupyter Notebook cell titled "main". The code has printed five tweets from the Twitter API. Below the tweets, there is a message: "to scrape other tweets please wait, i am sleeping" followed by "scraping tweets...". This indicates that the program has moved on to the next object after the sleep period has ended.

Si può notare come da schermo l'utente sia avvisato della ripresa nell'estrazione dei tweet. Al momento della chiamata di ogni oggetto, come visibile nella figura sopra, occorre settare i seguenti parametri:

- `newspaper`: l'account ufficiale di twitter dal quale vogliamo prendere i tweet
- `date_start, date_finish`: le date nel cui intervallo vogliamo estrarre i tweet
- `period`: il periodo storico a cui ci stiamo riferendo nelle date di estrazione dei tweet
- `tweets2sleep`: il numero limite di tweet estratti prima di sospendere l'estrazione per un determinato intervallo di tempo
- `sleeping_time_internal`: tempo di sospensione, espresso in secondi, una volta raggiunto il limite specificato in `tweets2sleep`
- `sleeping_time_external`: tempo di sospensione, espresso in secondi, una volta conclusa la chiamata ad un singolo oggetto
- `lang`: la lingua dei tweet che stiamo estraendo
- `path`: specificazione del file csv nel quale Tweetscraper salverà i tweet estratti.
- `n_tweets`: il numero massimo di tweet che vogliamo estrarre. È stato utile soprattutto nell'esecuzione delle prove di estrazione. Nell'estrazione vera e propria invece estraiamo tutti i tweet, senza limite, nelle date indicate. Per questo `n_tweets` è impostato di default ad infinito.

File scraping.py

Nel file scraping.py è contenuta la nostra classe Tweetscraper, chiamata di volta in volta con i vari oggetti in main.py:

```
class Tweetscraper:  
    """  
        Class Tweetscraper allows, in the specified time interval,  
        to scrape tweets from a specific official  
        account of a newspaper.  
  
        Courtesy notice: this code allows, in case of need,  
        to stop manually scraping and obtain the date  
        of last extracted tweet. If you want to exploit this opportunity:  
        If your operating system is Linux:  
            the date is returned automatically when you stop the code  
        If your operating system is Windows:  
            please go to Run-->edit configuration -->emulate terminal in output console  
            to stop the code press CTRL+c  
    """  
  
    def __init__(self, newspaper, start, end, period, tweets2sleep, sleeping_time, lang, path, n_tweets=float('inf')):  
        self.newspaper = newspaper # the official newspaper account from which you want to scrape tweets;  
        self.start = start # date since you want to scrape tweets  
        self.end = end # date until you want to scrape tweets  
        self.period = period # historical period of the chosen dates;  
        self.tweets2sleep = tweets2sleep # limit of tweets to be scraped before suspending scraping;  
        self.sleeping_time = sleeping_time # suspension time when the specified tweets2sleep limit is reached;  
        self.Lang = lang # language of tweets you want to scrape;  
        self.path = path # path of csv file where you want to save scraped tweets;  
        self.n_tweets = n_tweets #limit of tweets you want to scrape calling the object. By default is inf;
```

La classe ha bisogno di tutti i parametri specificati e descritti in precedenza, eccetto lo sleeping_time_external, che agisce esclusivamente nel main.py tra la chiamata di un oggetto e l'altro. Il metodo **dump_tweets()** permette di estrarre effettivamente i tweet servendoci della libreria non ufficiale snscreape, a cui viene passato il profilo Twitter del giornale, e le date di inizio e fine estrazione:

```

def dump_tweets(self):
    """
        return tweets given a specific twitter user and a time interval
    """
    tweets = []
    df = pd.DataFrame()

    try:
        # calls the API to obtain tweets
        # parsing the tweets
        print('scraping tweets...')
        counter = 0
        for i, tw in enumerate(sntwitter.TwitterSearchScraper(
            f'from:{self.newspaper} since:{self.start} until:{self.end}').get_items()):
            if i > self.n_tweets:
                df = pd.DataFrame(tweets, columns=['text', 'date', 'newspaper', 'likes', 'retweets', 'lang', 'period'])
                print(f'the date of last tweet scraped is {df.date.iloc[-1]}')
                self.create_csv(df)
                return df

            else:
                if counter == self.tweets2sleep:
                    print('i am sleeping')
                    time.sleep(self.sleeping_time)
                    print('scraping tweets...') ←
                    counter = 0

                tw_attributes = [tw.content, tw.date, self.newspaper.lower(),
                                 tw.likeCount, tw.retweetCount, self.lang, self.period]
                tw_attributes_names = ['text', 'date', 'newspaper', 'likes', 'retweets', 'lang', 'period']
                dic_tweet = {tw_attributes_names[attr]: tw_attributes[attr] for attr in range(len(tw_attributes))}

                tweets.append(dic_tweet) ←
                counter += 1

        df = pd.DataFrame(tweets, columns=['text', 'date', 'newspaper', 'likes', 'retweets', 'lang', 'period'])
        self.create_csv(df)

    except(Exception, KeyboardInterrupt) as e:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(str(e), fname, exc_tb.tb_lineno)

    try:
        df = pd.DataFrame(tweets, columns=['text', 'date', 'newspaper', 'likes', 'retweets', 'lang', 'period'])
        print(f'the date of last tweet scraped is {df.date.iloc[-1]}')
        self.create_csv(df)
        return df

    except:
        print('No one tweet extracted, dataframe is empty')
        return df

    return df

```

Per ogni tweet estratto, dobbiamo specificare quali attributi di quel tweet ottenere. La libreria snscreape ci permette infatti non solo di ottenere i testi dei tweet, ma anche altre informazioni, come ad esempio la data. Nel nostro caso scegliamo di ottenere le seguenti features (con relativa descrizione):

- **content**: testo del tweet
- **date**: data di pubblicazione del tweet
- **likeCount**: likes ottenuti da quel tweet
- **retweetCount**: numero di volte in cui il tweet è stato retweettato

Ognuno di questi attributi costituirà una colonna del dataframe (e poi del csv) in cui salveremo i nostri tweet estratti. A queste colonne aggiungiamo poi anche tre ulteriori attributi(che conosciamo già al momento della chiamata dell'oggetto in quanto parametri necessari) che sono:

- `self.newspaper`: nome del giornale di cui stiamo estraendo i tweet
- `self.lang`: lingua dei tweet che siamo estraendo
- `self.period`: periodo storico delle date selezionate per l'estrazione dei tweet

Per ogni tweet estratto viene inoltre fatto partire un contatore `counter` che permette di stabilire quando si arriva alla soglia `tweets2sleep` che porta allo stand-by dello scraping; una volta eseguito lo stand-by tramite il metodo `sleep()`, il contatore si riazzera. Di seguito l'output una volta estratti i primi tweet e attivatosi il primo sleeping time interno:

```
Run: main
scraping tweets...
i am sleeping
```

Questo invece è l'output una volta ripreso lo scraping

```
Run: main
scraping tweets...
i am sleeping
scraping tweets...
```

I codice che permette l'estrazione dei tweet è inoltre inglobato in un blocco `try and except`, che permette di gestire le eccezioni in caso di errore e conseguente interruzione del codice.

In realtà l'estrazione può interrompersi anticipatamente (rispetto all'estrazione di tutti i tweet compresi nelle date indicate) in 3 casi:

- nel caso in cui venga superato il limite `n_tweets`;
- nel caso in cui si incorra in errore;
- nel caso in cui il codice venga interrotto dall'utente.

Qualsivoglia sia la causa delle 3 appena elencate, se al momento dell'interruzione sono già stati estratti dei tweet, questi vengono comunque salvati in un dataframe e conseguentemente in un file csv, chiamando la funzione `create_csv` in modo da non perdere il lavoro eseguito.

E' stata inoltre inserita una courtesy per l'utente: una volta interrotto il codice, verrà stampata la `data dell'ultimo tweet estratto`. In questo modo l'utente è in grado di capire da quale punto in poi far ripartire l'estrazione. Se invece l'interruzione avviene senza che venga estratto nessun tweet, allora verrà comunicato all'utente che `nessun tweet è stato estratto`:

```

except(Exception, KeyboardInterrupt) as e:
    exc_type, exc_obj, exc_tb = sys.exc_info()
    fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
    print(str(e), fname, exc_tb.tb_lineno)

    try:
        df = pd.DataFrame(tweets, columns=['text', 'date', 'newspaper', 'likes', 'retweets', 'lang', 'period'])
        print(f'the date of last tweet scraped is {df.date.iloc[-1]}') ← green arrow
        self.create_csv(df)
        return df

    except:
        print('No one tweet extracted, dataframe is empty') ← yellow arrow
        return df

return df

if i > self.n_tweets:

    df = pd.DataFrame(tweets, columns=['text', 'date', 'newspaper', 'likes', 'retweets', 'lang', 'period'])
    print(f'the date of last tweet scraped is {df.date.iloc[-1]}') ← green arrow
    self.create_csv(df)
    return df

```

Questo l'output simulando un errore o un'interruzione dell'utente prima che qualsiasi tweet venga estratto:

```

Run: main ×
▶ ↑ name 'simulazione_errore' is not defined scraping.py 52
No one tweet extracted, dataframe is empty
Empty DataFrame
Columns: [text, date, newspaper, likes, retweets, lang, period]
Index: []

```

Questo l'output simulando un errore o un'interruzione dell'utente quando sono già stati estratti dei tweet:

```

Run: main ×
▶ ↑ name 'simulazione_errore' is not defined scraping.py 80
the date of last tweet scraped is 2019-07-02 23:57:37+00:00
text ... period
0 Sea-Watch, Carola Rackete è libera: "Commissa"... ... precovid
[1 rows x 7 columns]

```

Nel fare questo, abbiamo dovuto gestire in maniera specifica la terza causa di interruzione, quella relativa alla fatispecie in cui sia l'utente ad interrompere il codice. Per far sì che l'interruzione porti all'attivazione del blocco `except` (e quindi alla creazione del csv con i tweet estratti fino a quel momento), abbiamo aggiunto ad hoc l'eccezione `KeyboardInterrupt`, che altrimenti non veniva riconosciuta.

Quest'eccezione si attiva quando il codice viene appunto interrotto dall'utente, ma occorre fare alcune precisazioni. Testando il codice sia su sistema operativo Windows che su sistema operativo Linux abbiamo appurato che:

In linux l'interruzione del codice tramite ctrl+c fa scattare in automatico l'eccezione, e dunque la courtesy per l'utente.

In windows invece, a causa di un conflitto con gli shortcut di sistema, l'utente che voglia sfruttare la courtesy, dovrà necessariamente recarsi nel menù di Pycharm in: Run ->edit configuration->emulate terminal in output console.

Una volta attivata la spunta, per interrompere il codice utilizzare la shortcut ctrl+c posizionando il cursore sul nome del file in esecuzione.

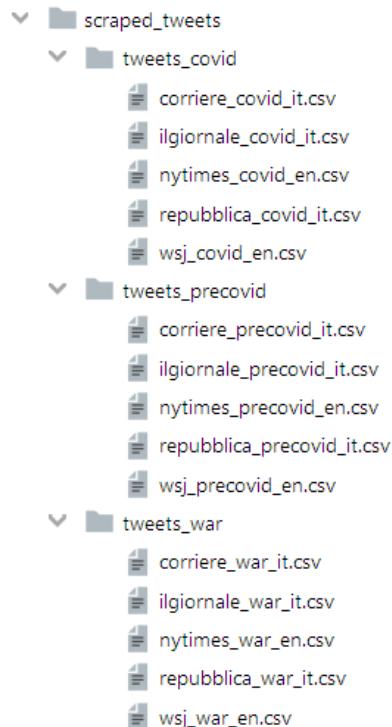
Infine il metodo `create_csv_file` permette di creare un file csv in cui vengono salvati i tweet estratti:

```
def create_csv(self, df):
    """
    creates csv file to save scraped tweets
    """
    path = self.path
    if not os.path.exists(path): os.makedirs(path)
    df.to_csv(f'{path}/{self.newspaper.lower()}_{self.period}_{self.lang}.csv', index=False, encoding='utf-8')
```

Il metodo viene attivato quando l'estrazione dei tweet viene completata, o come già accennato, nel caso in cui l'estrazione abbia già prelevato alcuni tweet e in seguito venga interrotta anticipatamente.

Nel definire questo metodo, è stato fatto in modo che il percorso e il nome del file venissero creati in maniera automatica, sulla base dello specifico giornale che si sta prendendo in considerazione, della sua lingua, e del periodo storico.

Sulla base di quanto appena detto, al termine della chiamata di tutti gli oggetti della classe `Tweetscraper`, avremo una directory `scraped_tweets` che conterrà tutti i csv con i tweet e che avrà la struttura riportata nella figura di destra:



Di seguito una porzione di uno dei file csv creati, nello specifico il file csv relativo ai tweet del giornale `repubblica` nel periodo `precovid`:

```
text,date,newspaper,likes,retweets,lang,period
Dunkirk, aggiornamento https://t.co/lzpXd6Vtwy,2019-12-30 19:53:05+00:00,repubblica,7,1,it,precovid
"Bancarotta Etruria: 14 a giudizio https://t.co/1Hv1bhz3AQ",2019-12-30 19:53:05+00:00,repubblica,10,3,it,precovid
È ufficiale: #Fioramonti lascia il #M5S https://t.co/C1ImiKRUBb,2019-12-30 19:46:42+00:00,repubblica,12,8,it,precovid
"l'ex ministro #Fioramonti lascia il #Movimento https://t.co/99pjNwcsls",2019-12-30 19:41:39+00:00,repubblica,11,8,it
"Dall'Europeo alle Olimpiadi https://t.co/1IrfyuUs3S",2019-12-30 17:14:05+00:00,repubblica,1,0,it,precovid
```

Pre-processamento dei tweet: branch tweets-pre-processing

Una volta estratti e salvati i nostri tweet passiamo alla seconda parte dell'analisi, e di conseguenza al secondo branch: quello relativo al pre processamento del nostro testo.

Abbiamo scelto di adottare i seguenti step di pre-processamento:

- **Pulizia generale** del testo grezzo, con rimozione della punteggiatura e degli altri caratteri speciali.
- **Tokenizzazione** del testo:

La tokenizzazione consiste nel suddividere un testo in entità più piccole, chiamate *token*. Con la tokenizzazione il testo grezzo viene trasformato in una lista di parole.

Essa viene effettuata mediante l'utilizzo di un *tokenizzatore* che prende in ingresso appunto un documento testuale (nel nostro caso il testo di un tweet) e restituisce una lista di parole (tokens) di quel documento.

- Eliminazione delle **stopwords**

Per stopwords si intendono quelle parole che, data la loro elevata frequenza in una lingua, sono ritenute poco significative nell'ambito dell'analisi testuale e di conseguenza poco informative. Di conseguenza la rimozione delle stopwords rappresenta uno dei passaggi fondamentali nel pre-processing del testo. In python, esistono delle librerie, come **NLTK** o **Spacy**, che permettono di individuare tutte le stopwords di una determinata lingua e filtrarle automaticamente. Esempi di stopwords nella lingua italiana sono: "mio", "dagli", "questo", "quello". Esempi di stopwords nella lingua inglese sono: "by", "me", "do", "same".

- **Lematizzazione**

Il quarto ed ultimo step è quello della lemmatizzazione.

La lemmatizzazione è uno step di pre-processing del testo che permette di riportare una parola al suo lemma. Ad esempio, tra le varie trasformazioni:

- i singolari ed i plurali, ma anche i maschili ed i femminili, vengono ricondotti allo stesso lemma:

amico, amica, amici, amiche → *amico*;
friend, friends → *friend*

- i verbi vengono ricondotti all'infinito:

vinci → *vincere*;
wins → *win*

Un'alternativa alla lemmatizzazione è rappresentata dallo **stemming**. Al contrario della lemmatizzazione, che riporta le parole al proprio lemma, lo stemming riporta le parole alla propria radice sintattica, rimuovendone i prefissi e i suffissi. Ad esempio:

doing → *do*;
national → *nation*;
witnesses → *witness*;

La lemmatizzazione è un'operazione più complessa, lenta, ed onerosa in termini di calcolo rispetto allo stemming. È però preferibile in quanto, riportando una parola al proprio lemma, automaticamente la riporta ad un termine esistente nel vocabolario. Al contrario, lo stemming potrebbe portare alla creazione di parole che non esistono effettivamente nel vocabolario di una determinata lingua.

Ci si trova quindi davanti ad un trade-off complessità/performance nello scegliere tra lemmatizzazione e stemming: in questo caso si è optato per l'operazione di lemmatizzazione.

Ciascuno di questi quattro passaggi appena descritti è stato approfondito ed adattato agli specifici obiettivi della nostra analisi e alle specifiche caratteristiche dei testi dei nostri tweet, come descriveremo tra poco quando entreremo nel dettaglio sul codice di questo secondo branch.

Nel secondo branch, oltre i file `requirements.txt` e `README.md`, sono presenti i file:

- `main.py`
- `italian_cleaning.py`
- `english_cleaning.py`

Nelle scelte delle librerie da utilizzare e nella stesura del codice abbiamo dovuto tenere conto del fatto di dover gestire tweet appartenenti a due differenti lingue: l'italiano e l'inglese.

In particolare, per quanto riguarda l'esecuzione della lemmatizzazione, la libreria `NLTK` permette di sfruttare il `WordNetLemmatizer`. Il WordNetLemmatizer è però esclusivamente abilitato per la lingua inglese.

Di conseguenza abbiamo optato per l'utilizzo della libreria `Spacy`, già esaminata durante le lezioni, seppur nell'ambito della sentiment analysis.

La libreria Spacy ci torna molto utile in quanto permette di eseguire tutta una serie di operazioni di pre-processing (tra cui la tokenizzazione, la rimozione della punteggiatura, la rimozione delle stopwords, e la lemmatizzazione) implementabili su un'ampia lista di linguaggi, tra cui quelli necessari per il nostro progetto, ovvero l'italiano e l'inglese. Questo è possibile installando le pipeline di pre-processamento addestrate di Spacy, disponibili appunto per vari linguaggi e scaricabili dal terminale.

Nello specifico, per l'italiano e l'inglese, una volta installato spacy, occorre eseguire da terminale i seguenti comandi:

```
python -m spacy download it_core_news_md
```

```
python -m spacy download en_core_web_md
```

Per gestire al meglio i tweet appartenenti alle due diverse lingue, si è scelto di creare due file separati:

- `italian_cleaning.py` nel quale è definita la classe ItalianCleaner, che si occuperà della pulizia dei tweet italiani.
- `english_cleaning.py` nel quale è definita la classe EnglishCleaner, che si occuperà della pulizia dei tweet inglesi.

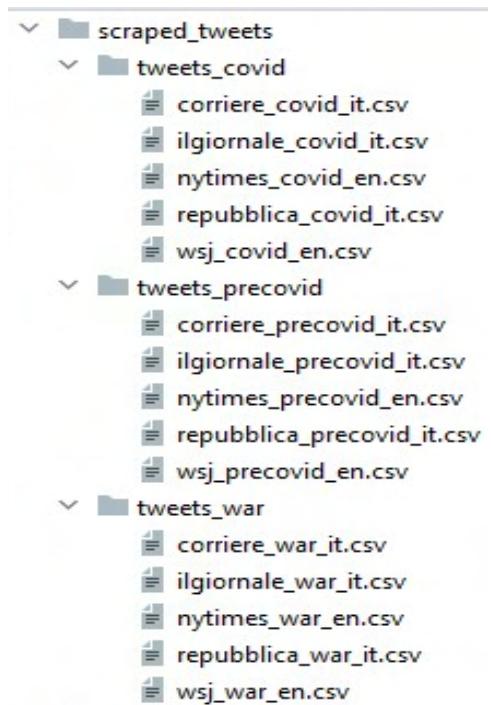
Questo ci permette, per alcuni aspetti e per alcune fasi del pre-processing, di agire in maniera differente e ad hoc per le due diverse lingue.

Entrambe le classi ItalianCleaner e EnglishCleaner vengono chiamate dal file main.py, che verrà a breve illustrato.

Dal branch precedente, tweets-scraping, abbiamo importato la nostra cartella scraped_tweets con tutti i nostri tweets estratti, eseguendo il comando da terminale:

```
git checkout tweets-scraping scraped_tweets
```

Anche in questo branch avremo quindi la cartella scraped_tweets con la seguente struttura:



File main.py

Nella prima parte del file main.py iteriamo su ognuno dei file csv contenuti nella cartella scraped_tweets. Per ogni file salviamo in un dizionario:

- il dataframe ottenuto convertendo il file csv in dataframe
- il percorso del file
- il nome del file

Ciascuno degli elementi salvati è utile in quanto viene passato come parametro al momento della chiamata delle classi ItalianCleaner ed EnglishCleaner.

Le classi adepti alla pulizia vengono chiamate di volta in volta per ciascuno dei file csv della nostra cartella:

```

if __name__ == "__main__":
    dataframes = {}
    roottdir = 'scraped_tweets'

    for subdir, dirs, files in os.walk(roottdir):
        for file in files:
            path=os.path.join(subdir, file)
            print(file,'file')
            print(path,'path')
            print(subdir,'subdir')
            print(roottdir,'roottdir')
            dataframes[file]=(pd.read_csv(path),path)
            dataframes[file]={'dataframe':pd.read_csv(path),'path': subdir,'file_name':file}

    for dframe in dataframes.values():
        df = dframe['dataframe']
        path = dframe['path']
        file_name = dframe['file_name']
        print(f'{path}\\"{file_name}')

        if file_name.endswith('it.csv'):
            Italian_Cleaner(df, path, file_name).pipeline()

        if file_name.endswith('en.csv'):
            English_Cleaner(df, path, file_name).pipeline()

```

Come è possibile notare dalla figura in alto, si è fatto in modo che, sulla base del nome del file, venga automaticamente riconosciuta la lingua dei tweets contenuti al suo interno, e di conseguenza venga chiamata alternativamente la classe ItalianCleaner o la classe EnglishClenaer. Entriamo adesso nel dettaglio della pulizia.

Prima di passare alla descrizione dei due file `italian_cleaning.py` ed `english_cleaning.py` occorre fare una premessa.

Nell'esecuzione delle prove di pulizia dei tweet, sono stati riscontrati per la lingua italiana dei problemi nella gestione delle parole con apostrofo, e abbiamo quindi deciso di approfondire la questione.

Tramite delle ricerche sul web, abbiamo individuato quelle che sono le più frequenti parole apostrofatate (sottoposte ad elisione) nella lingua italiana, eseguendo poi dei test per verificare come queste venissero gestite una volta sottoposte alle operazioni di eliminazione delle stopwords e di lemmatizzazione. E' stato riscontrato che, tra le parole soggette ad elisione:

- alcune sono riconosciute come stopwords solo se comprensive di apostrofo. Ad esempio “nient” viene riconosciuta come stopword ed eliminata solo se comprensiva di apostrofo. Di conseguenza effettuando una pulizia degli apostrofi prima dell'eliminazione delle stopwords, non verrebbe individuata.
- alcune sono riconosciute come stopword solo se NON comprensive di apostrofo. È il caso ad esempio di “cos”, “dov”, “sull”.
- alcune non sono riconosciute come stopwords né se comprensive di apostrofo né se ripulite dall'apostrofo. Se tenute con l'apostrofo però, la lemmatizzazione le riporta al loro lemma, e con un ulteriore ciclo di eliminazione delle stopwords vengono riconosciute ed eliminate. È il caso ad esempio di “quant”
- alcune non sono riconosciute come stopwords né se comprensive di apostrofo né se ripulite dall'apostrofo. Se tenute SENZA apostrofo però, la lemmatizzazione le riporta al loro lemma e con un ulteriore ciclo di eliminazione delle stopwords vengono riconosciute ed eliminate.
- alcune non vengono riconosciute come stopwords in alcun modo. È il caso ad esempio della parola “quand”.

Sulla base di queste considerazioni, abbiamo approfondito la ricerca sul fenomeno dell'elisione nella grammatica italiana, arrivando a conoscenza del fatto che, le parole che nel vocabolario italiano sono sottoponibili ad elisione (e dunque contengono l'apostrofo), fanno parte di una cerchia ristretta e ben definita. Tutte le parole appartenenti a questa cerchia sono riconosciute, nella loro forma senza elisione, come stopwords, eccetto "bell", "sant", "pover", "buon".

Di conseguenza si è arrivati alla scelta di creare una lista contenente tutte le parole apostrofatate della lingua italiana, "salvando" le quattro non stopwords appena indicate: tutte le parole riscontrate nei testi dei tweet che appariranno alla lista delle apostrofatate verranno eliminate.

Per quanto riguarda la lingua inglese invece, effettuando delle prove preliminari, è stato appurato che, la quasi totalità delle parole che più comunemente vengono scritte con l'apostrofo, sono gestite in maniera perfetta dalle operazioni di tokenizzazione e filtraggio delle stopwords attuate da Spacy. Per essere gestite al meglio devono però essere comprensive di apostrofo.

Sia per la lingua italiana che per la lingua inglese decidiamo dunque di "salvare" gli apostrofi dal primo step di pulizia generale, quello relativo all'eliminazione della punteggiatura e dei caratteri speciali, per poi gestirli in maniera migliore nelle fasi successive.

File [italian_cleaning.py](#)

Nel file `italian_cleaning.py` è definita la classe `Italian_Cleaner`, che permette di eseguire tutti i metodi necessari alla trasformazione, di volta in volta, di ciascuno dei vecchi file csv con i tweet sporchi, in nuovi file csv puliti:

```
class Italian_Cleaner:  
  
    def __init__(self, dataframe, path, file_name):  
  
        self.dataframe = dataframe #dataframe obtained by dirty csv file  
        self.path = path #csv file path  
        self.file_name = file_name #csv file name
```

Come visibile dalla figura sopra, la classe Tweetscraper ha bisogno dei seguenti parametri:

- `dataframe`: il dataframe ottenuto (nel `main.py`) dal file csv che si vuole pulire
- `path`: il percorso del file csv che si vuole pulire
- `file_name`: il nome del file csv che si vuole pulire

Dal file `main.py`, per ogni oggetto, la classe Tweetscraper viene chiamata con il metodo `pipeline()`.

Nel metodo `pipeline()` è stata costruita una pipeline nel quale vengono, di volta in volta, chiamati tutti i metodi necessari alla pulizia dei tweet di un determinato file csv ed alla creazione del nuovo file csv pulito. Come si può vedere, l'output di ciascun metodo chiamato costituisce l'input del metodo successivo:

```

def pipeline(self):
    """
        calls step by step the methods that allows to pre-process your tweets
        and to save them first in a dataframe and then in a new csv file
    """
    nlp = spacy.load("it_core_news_md")

    final_tweets_texts = []
    tweets_texts = [tweet.text for tweet in self.dataframe.itertuples()]
    tweets_texts_cleaned = [self.clean_rubbish(tw) for tw in tweets_texts]

    for tweet_text_cleaned in nlp.pipe(tweets_texts_cleaned, disable=['parser', 'ner', 'textcat']): #tokenization
        tweet_text_no_stop = self.stop(tweet_text_cleaned)
        tweet_text_final = self.lemma(tweet_text_no_stop)
        final_tweets_texts.append(tweet_text_final)

    clean1_dataframe = self.update_df(final_tweets_texts)
    clean2_dataframe = self.remove_duplicates(clean1_dataframe)
    clean3_dataframe = self.remove_empty_texts(clean2_dataframe)
    clean4_dataframe = self.datetime_format(clean3_dataframe)

    self.create_clean_csv(clean4_dataframe)

```

Il primo metodo ad essere chiamato dalla pipeline è il metodo `clean_rubbish()`:

```

def clean_rubbish(self, text):
    """
        cleans raw text from: all special characters; alone numbers;
        square brackets, curly brackets and the words inside them
    """
    text = re.sub('!', "", text)
    text_no_brackets = (re.sub("[{\[].*?[{\}]]", "", text))
    text_no_alone_num = (re.sub("\b[0-9]+\b", "", text_no_brackets))
    return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^\u00a9-\u00e1 \t])|(\https?\S+)|(_)", " ", text_no_alone_num).split())

```

Il metodo `clean_rubbish()` prende in ingresso di volta in volta il testo di ciascuno dei tweet di un vecchio file csv, provvedendo ad una prima sostanziale pulizia.

NB: i testi di ciascun tweet del vecchio file csv sono ottenuti iterando, tramite il metodo `itertuples()` della libreria Pandas, le righe del dataframe ricavato dal file csv stesso. Lo stesso metodo `itertuples()` verrà utilizzato anche nelle successive fasi del progetto ogni qualvolta ci sia la necessità di iterare un dataframe. La base della pulizia effettuata in `clean_rubbish` è la seguente regular expression vista a lezione:

```
re.sub("(@[A-Za-z0-9]+)|([^\u00a9-\u00e1 \t])|(\https?\S+)", " ", tweet)
```

Questa regular expression permette di eliminare dal testo tutta la punteggiatura, i caratteri speciali, i links, le email. Questa regex di base è stata modificata in maniera tale da:

- eliminare anche gli underscore
- tenere inizialmente le lettere accentate in quanto, effettuando delle prove, è stato riscontrato che alcune parole una volta rimosso l'accento non sono lemmatizzate correttamente o riconosciute come stopwords
- tenere inizialmente gli apostrofi, sulla base della premessa effettuata in precedenza.

Inoltre, abbiamo aggiunto, oltre alla regex di base, tre ulteriori regular expression:

- la prima permette di convertire il carattere con il classico carattere dell'apostrofo. Il carattere ' viene utilizzato in particolare dal Corriere della Sera e questo crea problemi nella pulizia:

```
1 text,date,newspaper,likes,retweets,lang,period
2 "Influenza, l'allarme contro il fai da te: «Usare meno farmaci» https://t.co/0qsarwfqfN",2019-12-30 22:02:07+00:00,corriere,10,6,it,precovid
```

- la seconda permette di eliminare il testo contenuto nelle parentesi quadre e nelle parentesi graffe, mantenendo invece il testo contenuto nelle parentesi tonde. In particolare Repubblica utilizza molto spesso, nei propri tweet, espressioni come:
→ [aggiornamento delle ore 21:07] certamente poco utile ai fini dell'analisi.

```
1 text,date,newspaper,likes,retweets,lang,period
2 "Colpo della Juventus, ecco Dejan Kulusevski [aggiornamento delle 21:16] https://t.co/Hb4x0rtlGK",2019-12-30 22:10:04+00:00,corriere,10,6,it,aggiornamento
3 "Bancarotta Etruria: 14 a giudizio, c'è anche Pierluigi Boschi [aggiornamento delle 21:07] https://t.co/2xbK17bZP7",2019-12-30 22:10:04+00:00,corriere,10,6,it,aggiornamento
4 La grande nave da crociera Msc urta la banchina del porto https://t.co/MG4zVN5jqq,2019-12-30 21:47:02+00:00,repubblica,10,6,it
5 "M5s, l'ex ministro Fioramonti lascia il Movimento [aggiornamento delle 20:55] https://t.co/HnVSpV4b8p",2019-12-30 20:04+00:00,corriere,10,6,it,aggiornamento
6 "Finisce in carcere Nicoletta Dosio, la passonaria No Tav ha rifiutato misure alternative [aggiornamento delle 20:50] h1
7 "M5s, l'ex ministro Fioramonti lascia il Movimento [aggiornamento delle 20:47] https://t.co/luB0nW2eA6",2019-12-30 19:51+00:00,corriere,10,6,it,aggiornamento
```

- la terza regex infine permette di eliminare tutti quei numeri presenti in un testo e staccati da qualsiasi lettera, ed allo stesso tempo di mantenere invece quei numeri che fanno parte di una parola. Ad esempio:
→ “2019” viene eliminato
→ “m5s” ; “g8” tengono invece i numeri

La funzione `clean_rubbish()` viene chiamata per ognuno dei tweet del file csv che la classe ItalianCleaner sta pulendo.

Una volta ripuliti tutti i tweet, ciascuno di essi viene tokenizzato tramite lo step di tokenizzazione della pipeline addestrata di Spacy. Ciascuno dei tweet, puliti da `clean_rubbish()` e tokenizzati, sarà l'input del secondo metodo, il metodo `stop()`, il cui compito principale è quello di eliminare le stopwords:

```
def stop(self, tweet_text_cleaned):
    """
        cleans tokenized text from: stopwords; apostrophized_words; italian_trash_words
    """
    apostrophized_words = ["qual'", "nell'", "all'", "nient'", "cos'", "com'", "dov'", "quant'", "quand'",
                           "nessun'", "un'", "dell'", "grand'", "quell'", "sull'", "quest'", "mezz'", "senz'",
                           "tutt'", "anch'", "c'", "d'", "s'", "v'", "t'", "m'", "l'", "po'", "di'", "fa'", "va'",
                           "sta'", "da'", "null'"]
    italian_trash_words = ["null", "rep", "o", "si", "de", "mi", "ti", "vi", "ci", "li", "lo", "la", "ne", "si",
                           "repubblica", "corriere", "giornale", "rt", "volere", "potere", "sapere", "arrivare", "dovere"]
    tweet_nostop = [t for t in tweet_text_cleaned if
                    (not t.is_punct and not t.is_stop and not t.text.lower() in apostrophized_words and
                     not t.text.lower() in italian_trash_words)]
    return tweet_nostop
```

Come possibile vedere dalla figura in alto, oltre a provvedere ad eliminare le stopwords italiane catalogate da Spacy, il metodo `stop()`, sulla base della premessa fatta in precedenza, elimina anche tutte le parole apostrofatate contenute nella lista `apostrophized_words` creata da noi.

Viene inoltre creata un'ulteriore lista `trash_words` di parole da eliminare, aggiornata di volta in volta sulla base dei risultati delle analisi successive e della Topic modeling.

L'output restituito da `stop()` diventa l'input del terzo metodo `lemma()` riportato di seguito:

```

def lemma(self, tweet_nostop):
    """
    lemmatizes words, removes: stopwords and italian trash words once more after lemmatization;
    removes words with lenght 1
    """
    italian_trash_words = ["null", "rep", "o", "si", "de", "mi", "ti", "vi", "ci", "li", "lo", "la", "ne", "si",
                           "repubblica", "corriere", "giornale", "rt", "volere", "potere", "sapere",
                           "arrivare", "dovere"]

    tweet_lemmatized_unstopped_ = [t.lemma_.lower() for t in tweet_nostop]
    tweet_text_final_trash = " ".join(tweet_lemmatized_unstopped_)
    tweet_text_final_splitted_trash = tweet_text_final_trash.split()
    tweet_text_final_splitted_no_trash = [word for word in tweet_text_final_splitted_trash if (
        word not in STOP_WORDS and word not in italian_trash_words and len(word) > 1)]
    tweet_text_final = " ".join(tweet_text_final_splitted_no_trash)
    return tweet_text_final

```

Come si può notare, il metodo `lemma()` provvede alla lemmatizzazione delle parole del testo rimaste dopo le precedenti operazioni di pulizia e, successivamente alla lemmatizzazione, ad un ulteriore passaggio di eliminazione delle stopwords e delle trash words.

Una volta applicati i metodi appena descritti su ciascuno dei tweet del file csv sporco, le operazioni di pre-processamento del testo sono terminate.

Ogni volta che ciascuno dei tweet viene ripulito, il suo nuovo testo viene aggiunto di volta in volta ad una lista `final_tweets_texts` che conterrà quindi tutti i testi puliti di tutti i tweet del csv. A questo punto dalla pipeline viene chiamato il metodo `update_df()`:

```

def update_df(self, final_tweets_texts):
    """
    updates with clean text the 'text' column of the dataframe obtained by old and dirty csv file
    """
    self.dataframe.text = final_tweets_texts
    return self.dataframe

```

Il metodo `update_df()` permette di aggiornare la colonna ‘text’ del datafame con i nuovi testi puliti. Otterremo quindi in output il dataframe aggiornato. Questo dataframe costituirà l’input per il successivo metodo `remove_duplicates()` illustrato di seguito:

```

def remove_duplicates(self, dataframe):
    """
    from dataframe removes duplicated tweets
    """
    dataframe.drop_duplicates(subset='text', keep='first', ignore_index=True, inplace=True)
    return dataframe

```

In quasi tutti i giornali infatti, abbiamo riscontrato la tendenza a pubblicare dei tweet identici. Nella quasi totalità dei casi questi tweet identici differiscono per il link a cui rimandano oppure per l’aggiunta di espressioni come: [aggiornamento delle 17:20]

Una volta effettuata la pulizia del testo però, sia i link che quanto contenuto nelle parentesi quadre viene rimosso. Di conseguenza sarà facilmente possibile individuare due tweet duplicati.

Il metodo `remove_duplicates()` permette proprio la rimozione dei duplicati, sfruttando il metodo `drop_duplicates()` della libreria pandas.

Otteniamo in output il dataframe ripulito dai tweet duplicati. Questo dataframe sarà a sua volta input del successivo metodo `remove_empty_texts()` illustrato di seguito:

```

def remove_empty_texts(self, dataframe):
    """
        from dataframe removes tweets left with no text after pre-processing
    """
    dataframe['text'].replace('', np.nan, inplace=True)
    dataframe.dropna(subset=['text'], inplace=True)
    dataframe.reset_index(drop=True, inplace=True)
    return dataframe

```

Nello svolgere le operazioni di pulizia del testo, potrebbe capitare che un determinato tweet, una volta sottoposto ai vari metodi, resti senza alcuna parola. In questo caso è possibile eliminare i tweet vuoti sfruttando il metodo `drop_na()` della libreria pandas.

Infine, ripuliamo la colonna ‘date’ contenente la data dei nostri tweet estratti. Al momento dell’estrazione la data viene resituita nel seguente formato:

2022-01-30 22:54:58+00:00

Di conseguenza conterrà anno, mese, giorno, ora, minuti, secondi, e fuso orario.

Il metodo `datetime_format()` ci permette di mantenere soltanto l’anno ed il mese, uniche due informazioni utili per il resto dell’analisi:

```

def datetime_format(self, dataframe):
    """
        from tweets dates removes: jetlegs, days, hours, minutes, seconds
    """
    dataframe['date'] = pd.to_datetime(dataframe.date).dt.strftime("%Y-%m")
    return dataframe

```

Una volta completate tutte le operazioni necessarie, viene chiamato il metodo `create_clean_csv()` che tramite il metodo `to_csv()` di pandas ci permette di ottenere, dal dataframe aggiornato, il nuovo file csv pulito:

```

def create_clean_csv(self, dataframe):
    """
        from dataframe creates a new csv file with clean tweets
    """
    path = f'clean_{self.path}'
    if not os.path.exists(path): os.makedirs(path)
    dataframe.to_csv(f'{path}\\{self.file_name}', index=False, encoding='utf-8')

```

Come si può notare, si è fatto in modo che, sulla base del percorso del path e del file_name passati come parametri alla classe, il file csv pulito venga creato in automatico con lo stesso nome del file csv vecchio, ma questa volta inserito in una nuova cartella chiamata `clean_scraped_tweets`.

File english_cleaning.py

Per quanto riguarda il file english_cleaning, in esso viene definita la classe EnglishCleaner, che si occupa della pulizia dei file csv contenenti tweet inglesi, e per la quale valgono praticamente le medesime considerazioni ed i medesimi commenti scritti per il file italian_clenaer.py.

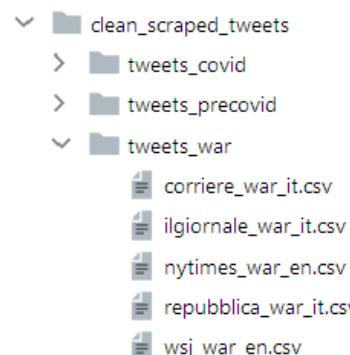
Tuttavia sono presenti alcune differenze opportune da spiegare.

- Innanzitutto al momento del caricamento dei modelli di spacy, andrà caricata la pipeline addestrata relativa all'inglese:

```
nlp = spacy.load("it_core_news_md")  
nlp = spacy.load("en_core_web_md")
```

- Nel metodo *stop()* non viene definita la lista delle parole apostrofatate. Nell'inglese, al contrario dell'italiano, le parole apostrofatate vengono gestite in maniera adeguata.
- Ovviamente le trash words saranno differenti e specifiche per ognuno dei due linguaggi:

```
italian_trash_words = ["null", "rep", "o", "si", "de", "mi", "ti", "vi", "ci", "li", "lo", "la", "ne", "sì",  
"repubblica", "corriere", "giornale", "rt", "volere", "potere", "sapere", "arrivare", "dovere"]  
  
english_trash_words = ["new", "city", "wall", "street", "journal", "post", "wsjwhatsnow",  
"say", "month", "year", "news", "day", "now"]
```



La cartella clean_scraped_tweets, una volta completate le operazioni di pulizia per tutti i file, avrà la struttura riportata sulla figura di destra:

Di seguito, un esempio del file csv riguardante i tweet di Repubblica nel periodo precovid, prima e dopo la pulizia:

```
1 text,date,newspaper,likes,retweets,lang,period  
2 Renault-Nissan: l'ex presidente Ghosn lascia gli arresti domiciliari in Giappone e vola in Libano. Tokyo apre una inchiesta https://t.co/c7mNjre0xz,2019-12-30 22:10:05+01  
3 "Fiori, pianti, biglietti: ora il guardrail di Gaia e Camilla diventa un altare [aggiornamento delle 22:34] https://t.co/oyCYZZRPh",2019-12-30 22:10:05+01  
4 "Colpo della Juventus, ecco Dejan Kulusevski [aggiornamento delle 21:16] https://t.co/Hb4xOrtlGK",2019-12-30 22:10:04+00:00,repubblica,15,1,it,precovid  
5 "M5s, l'ex ministro Fioramonti lascia il Movimento: ""Troppi attacchi dai Cinque stelle, delusione è un sentimento diffuso"" [aggiornamento delle 21:15] https://t.co/A020MzG2jk",2019-12-30 22:10:04+00:00,repubblica,15,1,it,precovid  
6 "M5s, l'ex ministro Fioramonti lascia il Movimento: ""Troppi attacchi dai Cinque stelle, delusione è un sentimento diffuso"" https://t.co/A020MzG2jk",2019-12-30 22:10:04+00:00,repubblica,15,1,it,precovid
```

```
1 text,date,newspaper,likes,retweets,lang,period  
2 renault nissan presidente ghosn lasciare arresto domiciliare giappone volare libano tokyo aprire inchiesta,2019-12,repubblica,9,7,it,precovid  
3 fiori pianta biglietto guardrail gaia camilla altare,2019-12,repubblica,3,2,it,precovid  
4 colpo juventus dejan kulusevski,2019-12,repubblica,15,1,it,precovid  
5 m5s fioramonti lasciare movimento troppi attacco cinque stella delusione sentimento diffuso,2019-12,repubblica,8,1,it,precovid  
6 bancarotta etruria giudizio pierluigi boschi,2019-12,repubblica,6,6,it,precovid
```

Analisi preliminari: branch tweets-preliminar-analyses

Una volta effettuate tutte le operazioni di pre-processing, i tweet sono pronti per poter svolgere alcune prime analisi e poter stilare alcune considerazioni.

Le analisi preliminari permettono infatti di avere un primo flash su quelli che possono essere gli argomenti trattati dai quotidiani oggetto d'analisi, e come questi si evolvono nel tempo.

Per importare i nostri file csv puliti in questo branch, inseriamo da terminale il seguente comando:

```
git checkout tweets-pre-processing clean_scraped_tweets
```

In questo modo sarà presente nel branch tweets-preliminar-analyses la copia di clean_scraped_tweets. L'analisi preliminare si svolge in tre diversi file, che si aggiungono ai file requirements.txt, README.md, main.py, e sono:

- topwords.py
- word_trends.py
- retweet_like_tweet.py

Le classi definite in ciascuno dei tre file vengono chiamate nel file main.py, di seguito descritto.

File main.py

Nel file main.py si itera per ciascun file csv pulito contenuto nella cartella clean_scraped_tweets.

Per ogni file, grazie al metodo `read_csv()` della libreria Pandas, ricaviamo il dataframe corrispondente e, per mezzo del metodo `concat()` della libreria Pandas, lo concateniamo con il dataframe ricavato dal file csv dell'iterazione precedente.

Una volta iterato per ogni file csv della cartella, si ottiene il dataframe_complete, ovvero il dataframe completo dato dalla concatenazione dei dataframe ricavati da ciascun file csv:

```
if __name__ == "__main__":
    rootdir = 'clean_scraped_tweets'

    dataframe_complete=pd.DataFrame(columns=('text', 'date', 'newspaper', 'likes', 'retweets', 'lang', 'period'))

    #iterating through all the csv files in the clean_scraped_tweets directory
    #returning a dataframe
    #dataframe is obtained by concatenating the dataframes derived from each of the csv files of the directory

    for subdir, dirs, files in os.walk(rootdir):
        for file in files:

            path=os.path.join(subdir, file)
            dataframe=pd.read_csv(path)
            dataframe_complete = pd.concat([dataframe,dataframe_complete])
```

Il dataframe completo risulterà utile nella chiamata delle diverse classi che permetteranno l'analisi preliminare. Ci consentirà infatti, nelle singole chiamate degli oggetti delle classi, di poter selezionare di volta in volta solo specifiche righe del dataframe stesso, sulla base di quali giornali e di quali periodi si vogliono di volta in volta considerare nell' analisi.

Una volta ottenuto il complete_dataframe, si può procedere alla chiamata degli oggetti delle varie classi. Di seguito alcune chiamate a titolo esemplificativo per ciascuno dei tre file di analisi:

```
# calling objects of Topwords class
```

```
Topwords(dataframe_complete, 'it', 'precovid').show_topwords()
Topwords(dataframe_complete, 'it', 'precovid', 'corriere').show_topwords()
Topwords(dataframe_complete, 'it', 'precovid', 'repubblica').show_topwords()
Topwords(dataframe_complete, 'it', 'precovid', 'ilgiornale').show_topwords()
Topwords(dataframe_complete, 'it', 'covid').show_topwords()
Topwords(dataframe_complete, 'it', 'covid', 'corriere').show_topwords()
Topwords(dataframe_complete, 'it', 'covid', 'repubblica').show_topwords()
Topwords(dataframe_complete, 'it', 'covid', 'ilgiornale').show_topwords()
Topwords(dataframe_complete, 'it', 'war').show_topwords()
Topwords(dataframe_complete, 'it', 'war', 'corriere').show_topwords()
Topwords(dataframe_complete, 'it', 'war', 'repubblica').show_topwords()
Topwords(dataframe_complete, 'it', 'war', 'ilgiornale').show_topwords()
Topwords(dataframe_complete, 'en', 'precovid').show_topwords()
Topwords(dataframe_complete, 'en', 'precovid', 'wsj').show_topwords()
Topwords(dataframe_complete, 'en', 'precovid', 'nytimes').show_topwords()
Topwords(dataframe_complete, 'en', 'covid').show_topwords()
Topwords(dataframe_complete, 'en', 'covid', 'wsj').show_topwords()
Topwords(dataframe_complete, 'en', 'covid', 'nytimes').show_topwords()
Topwords(dataframe_complete, 'en', 'war').show_topwords()
Topwords(dataframe_complete, 'en', 'war', 'wsj').show_topwords()
Topwords(dataframe_complete, 'en', 'war', 'nytimes').show_topwords()
```

```
#calling objects of WordTrend class
```

```
WordTrend(dataframe_complete, 'it').calculate_cfdist()
WordTrend(dataframe_complete, 'it', 'corriere').calculate_cfdist()
WordTrend(dataframe_complete, 'it', 'repubblica').calculate_cfdist()
WordTrend(dataframe_complete, 'it', 'ilgiornale').calculate_cfdist()
WordTrend(dataframe_complete, 'en').calculate_cfdist()
WordTrend(dataframe_complete, 'en', 'wsj').calculate_cfdist()
WordTrend(dataframe_complete, 'en', 'nytimes').calculate_cfdist()
```

```
#calling objects of CountTweetRetweetLike class
```

```
CountTweetRetweetLike(dataframe_complete, 'it')
CountTweetRetweetLike(dataframe_complete, 'it', 'corriere')
CountTweetRetweetLike(dataframe_complete, 'it', 'repubblica')
CountTweetRetweetLike(dataframe_complete, 'it', 'ilgiornale')
CountTweetRetweetLike(dataframe_complete, 'en')
CountTweetRetweetLike(dataframe_complete, 'en', 'wsj')
CountTweetRetweetLike(dataframe_complete, 'en', 'nytimes')
```

File topwords.py

La prima analisi svolta è contenuta nel file topwords.py nel quale è definita la classe Topwords:

```
class Topwords:  
    """  
        given: a specific newspaper or a group of newspapers; a specific period; a specific lang;  
        returns: the most frequent words for that specific newspaper(s) in that specific period  
    """  
    def __init__(self, dataframe, lang, period, newspaper='all newspapers'):  
        self.dataframe = dataframe #complete dataframe with all tweets of clean_scraped_tweets directory  
        self.lang = lang #language of the newspaper (or newspapers) under analysis  
        self.period = period #historical period under analysis  
        self.newspaper = newspaper #newspaper (or newspapers) under analysis
```

Dati in ingresso: uno specifico giornale o un gruppo di giornali i cui tweet sono scritti nella medesima lingua; uno specifico periodo storico; la lingua in cui sono scritti il giornale/i preso/i in considerazione. La classe restituisce: le parole più frequenti nei tweet pubblicati da quello specifico giornale, o da quello specifico gruppo di giornali, nel periodo storico indicato.

Occorre quindi specificare i seguenti parametri:

- **dataframe**: il dataframe completo ottenuto dalla concatenazione
- **lang**: il linguaggio con cui sono scritti i tweet dei giornali che si stanno prendendo in considerazione per l'analisi
- **period**: il periodo storico che si sta prendendo in considerazione per l'analisi;
- **newspaper**: il giornale (o il gruppo di giornali) che si sta prendendo in considerazione per l'analisi;

Nella classe Topwords è definito il metodo **show_topwords()**:

```
def show_topwords(self):  
    """  
        returns the most frequent words  
    """  
    fdist = nltk.FreqDist([w for tweet in self.dataframe.itertuples() for w in tweet.text.split()  
                          if (tweet.lang == self.lang and tweet.period == self.period  
                              and (self.newspaper == tweet.newspaper or self.newspaper == 'all newspapers'))])  
  
    most_common = fdist.most_common(6)  
    table = PrettyTable(['Word', 'Count'])  
    for t in most_common:  
        table.add_row(t)  
  
    print(table)
```

Il metodo **show_topwords()** viene attivato nel main.py al momento della chiamata di ogni oggetto della classe Topwords.

Esso esegue il conteggio delle parole, andando a costruire la distribuzione di frequenza delle stesse nei tweet presi in considerazione. Questo avviene grazie al metodo **FreqDist()** della libreria NLTK, un metodo apposito per la costruzione delle distribuzioni di frequenza.

Una volta calcolata la distribuzione di frequenza, viene restituita la top6 delle parole più frequenti, accompagnate dall'indicazione numerica delle volte in cui occorrono: il tutto è schematizzato in una tabella con l'ausilio della libreria Prettytable di Python, in modo da servire all'utente una rappresentazione più chiara e leggibile dei risultati.

Di seguito vengono riportate le tabelle ottenute per ciascuna chiamata, suddivise per giornale e periodo storico:

GIORNALI ITALIANI

Word	Count	Word	Count	Word	Count
salvini	1862	italia	12116	ucraina	2983
italia	1657	covid	11802	russo	1945
conte	1531	coronavirus	10785	guerra	1816
roma	1241	italiano	5686	putin	1640
pd	1115	roma	5637	russia	1520
morire	996	conte	5413	italia	1401

CORRIERE DELLA SERA

Word	Count	Word	Count	Word	Count
italia	588	italia	4371	russo	462
salvini	422	covid	3944	putin	460
italiano	417	milano	2389	ucraina	417
conte	415	morire	2081	italia	371
morire	410	vaccino	2069	guerra	345
figlio	394	coronavirus	1826	russia	287

REPUBBLICA

Word	Count	Word	Count	Word	Count
salvini	1122	coronavirus	8241	ucraina	1700
italia	986	covid	6268	scoprire	978
conte	937	italia	5451	guerra	877
roma	862	roma	3469	russo	703
pd	712	italiano	2614	russia	654
morire	559	usa	2515	putin	606

IL GIORNALE

Word	Count	Word	Count	Word	Count
salvini	318	italia	2294	ucraina	866
pd	183	covid	1590	russo	780
conte	179	conte	1409	guerra	594
m5s	138	italiano	1291	russia	579
migrante	124	vaccino	1052	putin	574
dimaio	103	presidente	1031	italia	533

Considerazioni sul trend delle topwords nei giornali italiani:

- Nel periodo **antecedente** allo scoppio della **pandemia**, le topwords risultanti da ciascun giornale indicano come il tema predominante sia la politica. Le parole più frequenti riguardano infatti, per tutti e tre i giornali considerati, quasi esclusivamente nomi di esponenti politici (Salvini, Conte, Di Maio) o di partiti (m5s, pd). Alle parole riguardanti la sfera politica si aggiunge poi, sia per il Corriere che per Repubblica, la parola “morire”, il che potrebbe far pensare che, la seconda sfera più trattata dopo quella politica sia quella relativa alla cronaca nera. Da segnalare infine, la presenza della parola migrante nelle topwords de Il Giornale, selezionato come quotidiano con orientamento politico a destra: il tema dei migranti e della difesa dei confini nazionali è infatti un tema solitamente preponderante nella propaganda elettorale dei partiti di destra.
- Nel periodo **covid** si assiste all’ascesa in classifica delle parole “covid”, “coronavirus”, “vaccino”. La scalata del tema covid comporta la quasi totale estromissione dalla top six delle parole inerenti il tema politico, monopolizzando dunque la classifica.
- Nel periodo **war** si assiste invece ad una monopolizzazione ancora maggiore. Dalla top6 di tutti e tre i giornali italiani scompare qualsiasi parola inerente al coronavirus ed alla pandemia; ascesa netta invece delle parole riguardanti la sfera dell’invasione russa in Ucraina: “ucraina” “russo”, “guerra”, “russia”, “Putin”.

GIORNALI AMERICANI					
Word	Count	Word	Count	Word	Count
trump	3073	coronavirus	12009	ukraine	2786
president	3040	president	10225	russia	2141
write	1839	people	9745	russian	1752
people	1794	trump	8631	president	1485
company	1253	pandemic	7368	people	1285
house	1047	write	7344	war	1050

WALL STREET JOURNAL					
Word	Count	Word	Count	Word	Count
trump	1218	covid	4718	ukraine	1040
president	1002	coronavirus	4573	russia	847
write	794	company	3914	russian	654
company	791	pandemic	3869	company	541
people	666	president	3748	president	505
china	650	people	3581	people	471

NEW YORK TIMES					
Word	Count	Word	Count	Word	Count
president	2038	coronavirus	7436	ukraine	1746
trump	1855	president	6477	russia	1294
people	1128	people	6164	russian	1098
write	1045	trump	5471	president	980
york	709	york	4313	people	814
police	641	write	3852	war	649

Considerazioni sui trend delle topwords nei giornali americani:

- Nel periodo *pre-pandemico*, come per i giornali italiani, dominano le parole inerenti la politica: “*trump*”, “*president*”, “*house*” (casa bianca?). Spiccano anche: “*police*” nel New York Times, probabilmente legata alla forte attenzione mediatica sui poliziotti americani; “*china*” nel Wall Street Journal, probabilmente legata all’attenzione da parte dei media americani sulle tensioni USA-Cina.
- Nel periodo *covid*, come per i quotidiani italiani, si assiste all’ascesa delle parole inerenti alla pandemia: “*covid*”, “*coronavirus*”, “*pandemic*”, seguite dalle parole collegate alla sfera politica.
- Nel periodo *war*, anche stavolta in linea con quanto accade in Italia, la monopolizzazione delle parole attinenti al tema guerra è ancora maggiore rispetto a quella verificatasi nel periodo covid. Nella top six del Wall Street Journal metà delle parole appartengono alla sfera della guerra (“*ukraine*”, “*russia*”, “*russian*”), mentre nella top six del New York Times addirittura 4 parole su 6 richiamano l’argomento (“*ukraine*”, “*russia*”, “*russian*”, “*war*”).

File word_trends.py

La seconda analisi svolta è contenuta nel file word_trends.py nel quale è definita la classe WordTrend:

```
class WordTrend:  
    """  
        given a specific newspaper or a group of newspapers, returns the time trend  
        with which, a set of words chosen by user, occurs in newspaper's tweets  
    """  
  
    def __init__(self, dataframe, lang, newspaper='all newspapers'):  
  
        self.lang = lang          #language of the newspaper (or newspapers) under analysis  
        self.newspaper = newspaper #newspaper (or newspapers) under analysis  
        self.dataframe = dataframe #complete dataframe with all tweets of clean_scraped_tweets directory
```

Dati in ingresso: uno specifico giornale o un gruppo di giornali i cui tweet sono scritti nella medesima lingua; la lingua in cui sono scritti il giornale(i giornali) preso(i) in considerazione

La classe restituisce: una rappresentazione grafica del trend temporale della frequenza(nei tweet pubblicati dai giornali oggetto d'analisi) delle parole appartenenti ad una lista target scelta dall'utente.

Occorre quindi specificare i seguenti parametri:

- **dataframe**: il dataframe completo ottenuto dalla concatenazione
- **lang**: il linguaggio in cui sono scritti i tweet dei giornali che si stanno prendendo in considerazione per l'analisi;
- **newspaper**: il giornale (o il gruppo di giornali) che si sta prendendo in considerazione per l'analisi. NB: il parametro newspaper è impostato di default a “all newspapers”, in questo modo, se l'utente non specifica nessun giornale, verranno presi in considerazione tutti i giornali appartenenti alla lingua selezionata.

Nella classe WordTrend è definito il metodo **calculate_cfdist()**:

```
def calculate_cfdist(self):  
    """  
        plots the time trend month by month of a specific set of words chosen by user  
    """  
  
    word_cfdist = nltk.ConditionalFreqDist(  
        (target, tweet.date)  
        for tweet in self.dataframe.itertuples()  
        for w in tweet.text.split()  
        for target in ['covid', 'vaccine', 'putin', 'ukraine', 'trump']  
        if (w.lower() == target and (self.newspaper == tweet.newspaper or self.newspaper == 'all newspapers') and  
            tweet.lang == self.lang))  
  
    word_cfdist.plot()  
    self.calculate_normalized_cfdist(word_cfdist)
```

Il metodo **calculate_cfdist()** viene attivato nel main.py al momento della chiamata di ogni oggetto della classe WordTrend.

Esso esegue il conteggio delle co-occorrenze mese/parola-target, andando a costruire una distribuzione di frequenza condizionata ricevendo in input il testo e la data dei tweet presi in considerazione.

Questo avviene grazie al metodo **ConditionalFreqDist()** della libreria NLTK, un metodo apposito per la costruzione delle distribuzioni di frequenza condizionate.

Una volta calcolata la distribuzione di frequenza condizionata, viene restituito il grafico del trend temporale, mese per mese, della frequenza delle parole scelte dall'utente come parole target.

Per ciascuna parola target, verrà utilizzato un colore diverso, riscontrabile nella legenda, in modo da migliorare la leggibilità del grafico.

Infine `calculate_cfdist()`, una volta eseguito il suo compito, chiama il metodo `calculate_normalized_cfdist()`:

```
def calculate_normalized_cfdist(self, word_cfdist):
    """
    plots the time trend month by month of a specific set of words chosen by user,
    normalized on number of tweets published per month
    """
    months = sorted(list(set([tweet.date for tweet in self.dataframe.itertuples()])))
    tweets4month = [np.sum([1 for tweet in self.dataframe.itertuples()
                           if (tweet.date == month and tweet.lang == self.lang
                               and (self.newspaper == tweet.newspaper or self.newspaper == 'all newspapers'))])
                    for month in months]

    dic_tweets4month = {months[index_month]: tweets4month[index_month] for index_month in range(0, len(months))}

    for target_word in word_cfdist.values():
        for month in target_word.keys():
            target_word[month] = target_word[month]/dic_tweets4month[month]

    word_cfdist.plot()
```

Il metodo `calculate_normalized_cfdist()` esegue esattamente gli stessi compiti di `calculate_cfdist()`, ma normalizza il conteggio delle co-occorrenze mese per mese in base al numero di tweet pubblicati in ciascun specifico mese.

Di seguito vengono i grafici con i trend temporali ottenuti per ciascuna chiamata, in base ai giornali presi in considerazione:

GIORNALI ITALIANI

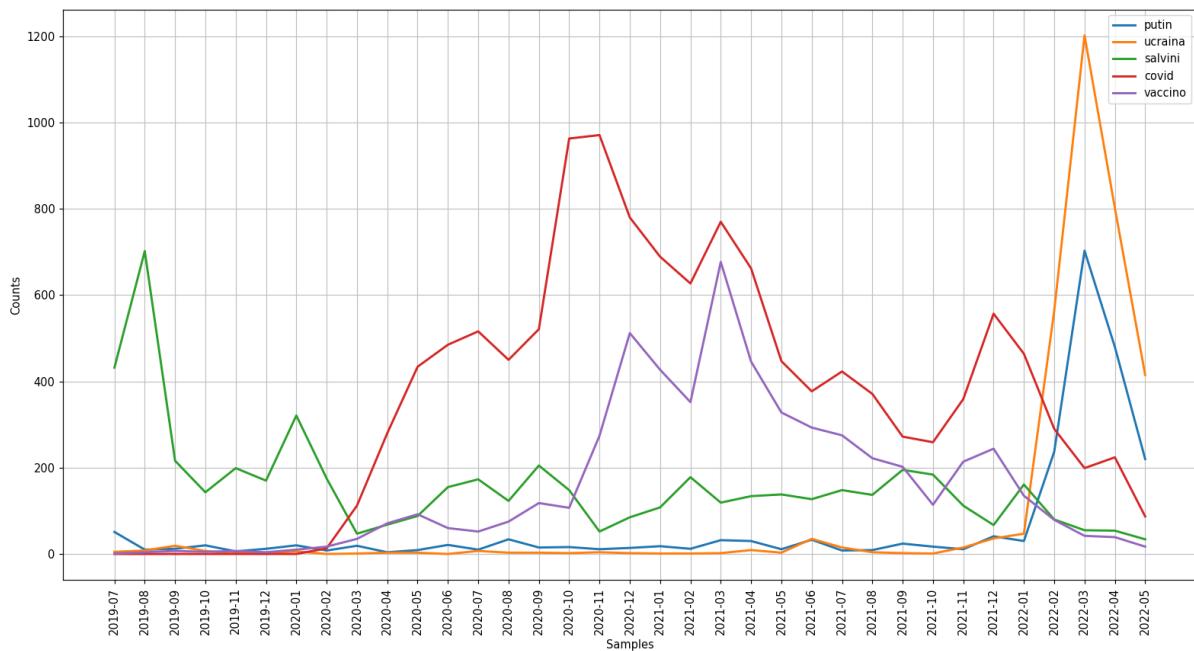
Abbiamo scelto di utilizzare come parole target: “*covid*”, “*vaccino*”, “*putin*”, “*ucraina*”, “*salvini*”.

Le parole “*covid*” e “*vaccino*” sono state scelte come rappresentanti del tema pandemia;

Le parole “*putin*” e “*ucraina*” sono state scelte come rappresentanti del tema guerra;

La parola “*salvini*” è stata scelta come rappresentante del tema politica, che come visto nella precedente analisi, sembra essere il tema dominante prima dell'inizio della pandemia.

Di seguito il grafico risultante dall'analisi effettuata sui tre giornali italiani presi insieme:



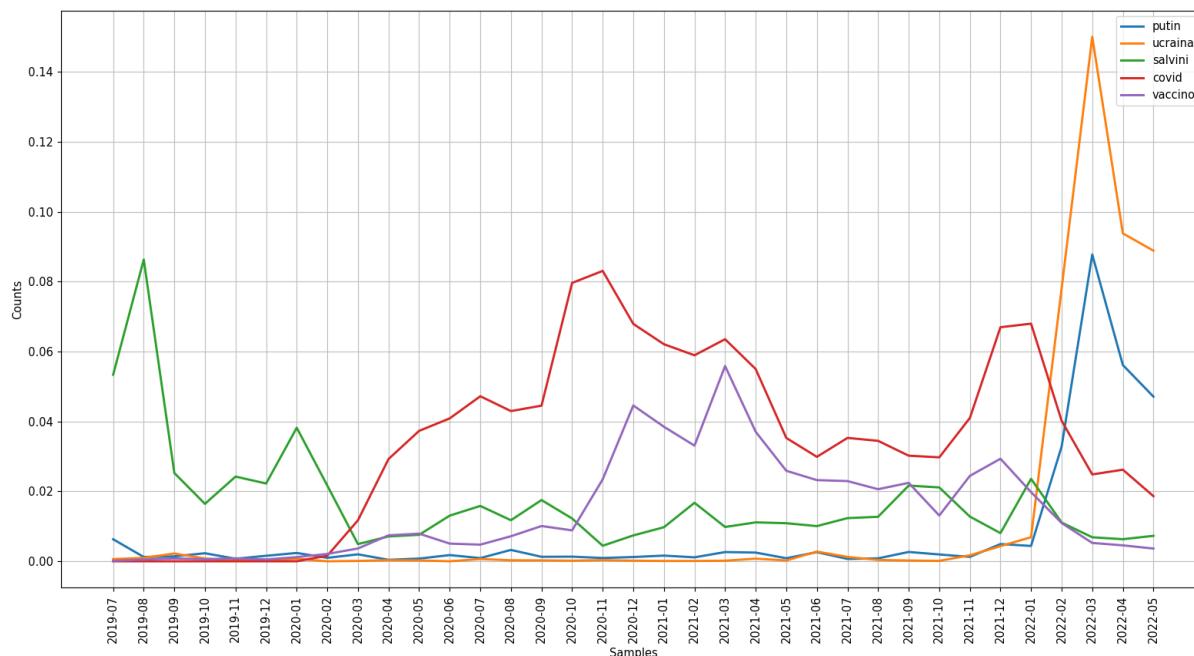
Possiamo notare dal grafico come, la parola "Salvini", seppur già in trend discendente, subisca un evidente e brusco crollo definitivo (da 200 a meno di 50) proprio tra febbraio e marzo, mese in cui scoppia la pandemia in Italia. Tra il mese di febbraio e marzo le due curve, una in totale discesa e l'altra in totale ascesa, si incrociano.

Le occorrenze della parola "salvini" si riprendono poi leggermente nei mesi successivi rimanendo più o meno costanti sulle 150 circa al mese.

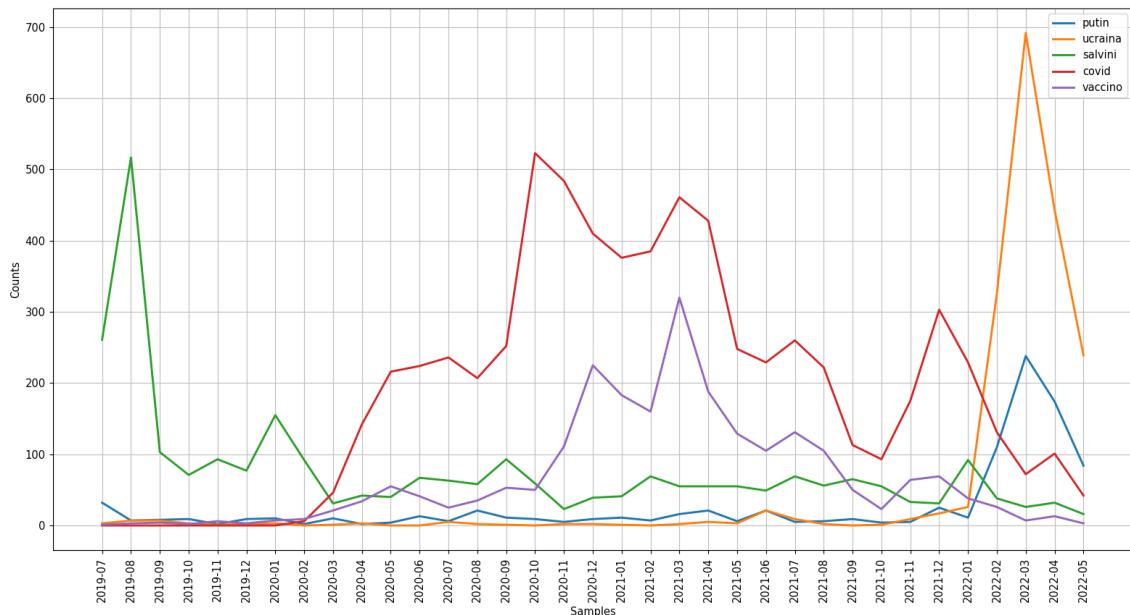
Nei mesi tra marzo 2020 e gennaio 2022 le parole "vaccino" e "covid" monopolizzano totalmente la situazione, dominando sulle altre.

Interessante risulta essere il periodo che va dal mese di gennaio 2022 a seguire: non solo "ucraina" e "putin", rappresentative del tema guerra, conquistano la scena, ma provocano di pari passo un crollo delle parole "vaccino" e "covid", che tra dicembre 2021 e marzo 2022 vengono più che dimezzate.

Il grafico normalizzato segue in maniera quasi identica l'andamento del grafico precedente, di seguito riportato a titolo esemplificativo:



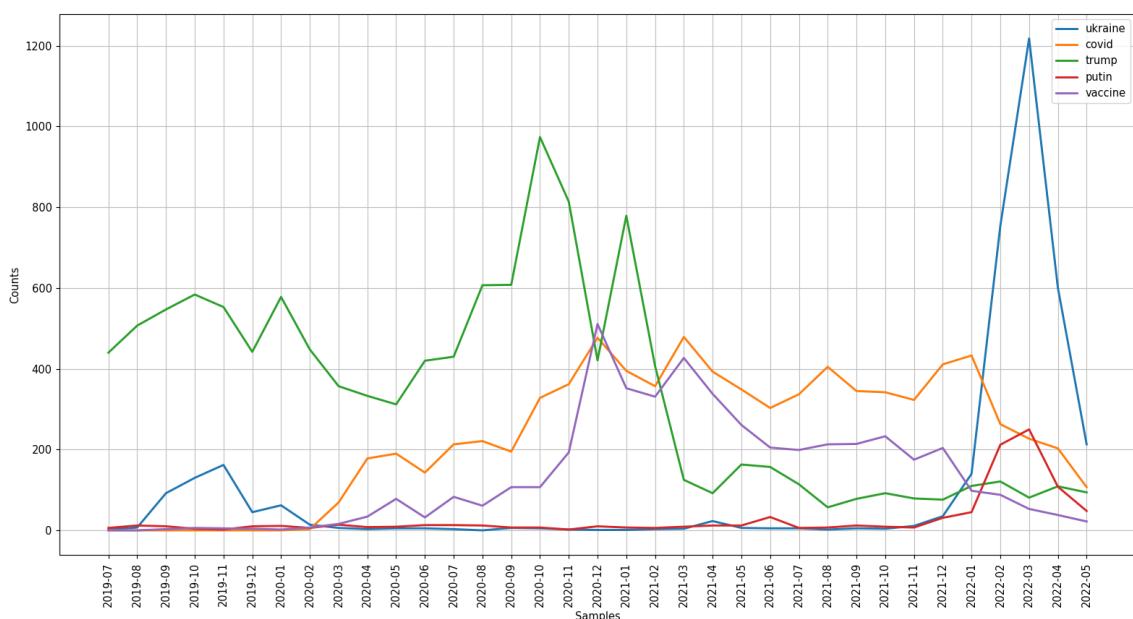
Il codice permette di stampare, come anticipato, anche i grafici giornale per giornale. Nel caso dei giornali italiani, l'andamento è praticamente per tutti il medesimo di quello appena visto nel grafico generale, senza nulla aggiungere al contenuto informativo. A titolo esemplificativo riportiamo quello relativo al quotidiano Repubblica:



GIORNALI AMERICANI

Abbiamo scelto di utilizzare come parole target: "covid", "vaccine", "putin", "ukraine", "trump". Le parole "covid" e "vaccine" sono state scelte come rappresentanti del tema pandemia; Le parole "putin" e "ukraine" sono state scelte come rappresentanti del tema guerra; La parola "trump" è stata scelta come rappresentante del tema politica, che come visto nella precedente analisi, sembra rappresentare il tema dominante prima dell'avvento della pandemia.

Di seguito il grafico risultante dall'analisi effettuata sui due giornali statunitensi presi insieme:



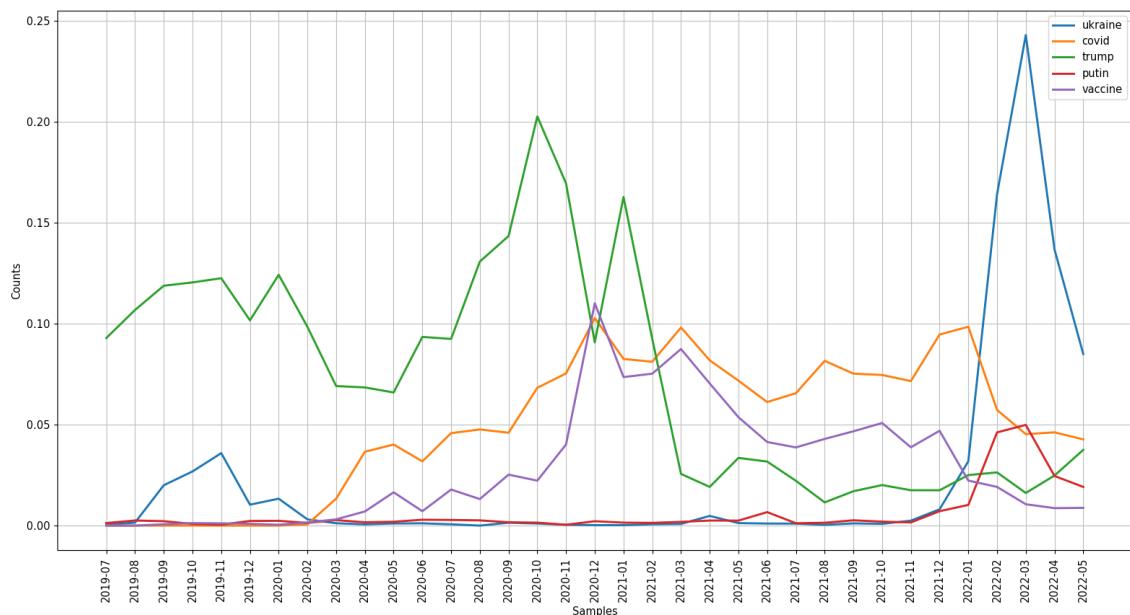
Possiamo notare dal grafico come la situazione sia sotto certi aspetti diversa rispetto a quella dei quotidiani italiani. Allo scoppio della pandemia si assiste sì ad una forte crescita della parola ‘covid’ e della parola ‘vaccine’, ma non c’è, perlomeno a confronto con la parola ‘trump’, una monopolizzazione netta.

Occorre però precisare che, il 2020, è stato per gli USA l’anno delle elezioni presidenziali. Questo potrebbe spiegare come la parola ‘trump’ sia riuscita comunque a dominare su ‘covid’ e ‘vaccine’, in quanto Donald Trump è stato proprio uno dei due candidati nella corsa alla Casa Bianca, conclusasi il 3 novembre 2020.

Da notare anche come, negli USA, il tema Ucraina abbia acquisito una certa rilevanza negli ultimi mesi del 2019, poi crollato definitivamente in concomitanza del diffondersi della pandemia.

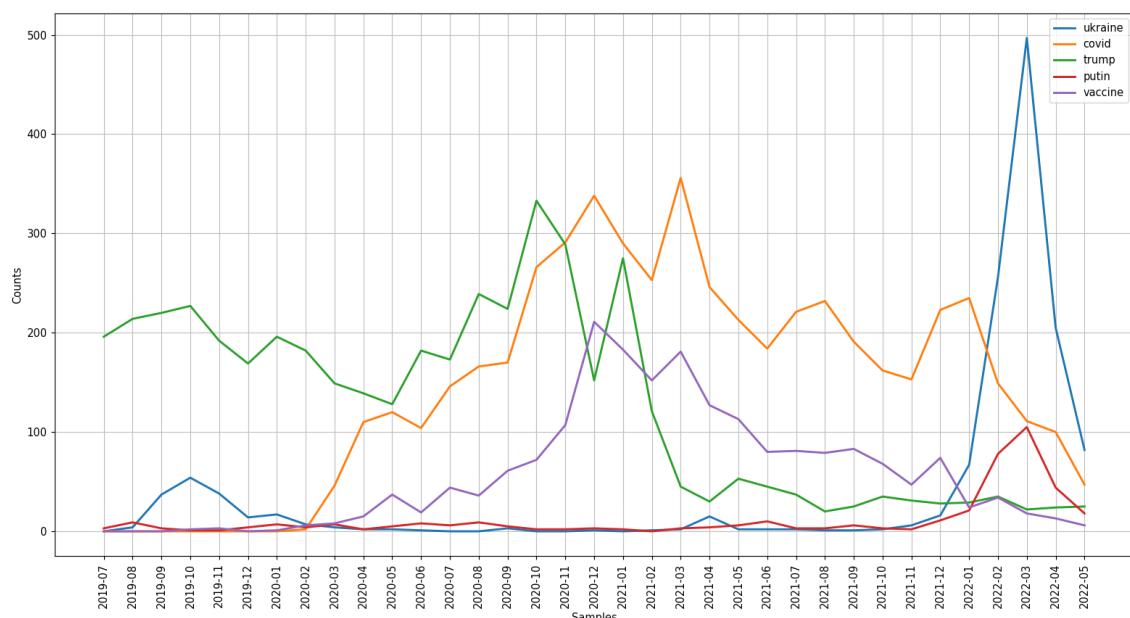
Anche in questo caso, come per i quotidiani italiani, è interessante ciò che succede da gennaio 2022 in poi: le parole ‘putin’ e soprattutto ‘ukraine’ iniziano la loro ascesa, e di contro, si assiste ad un crollo delle parole ‘covid’ e ‘vaccine’ che tra gennaio 2022 ed aprile 2022 dimezzano le loro occorrenze.

Di seguito il medesimo grafico ma normalizzato:

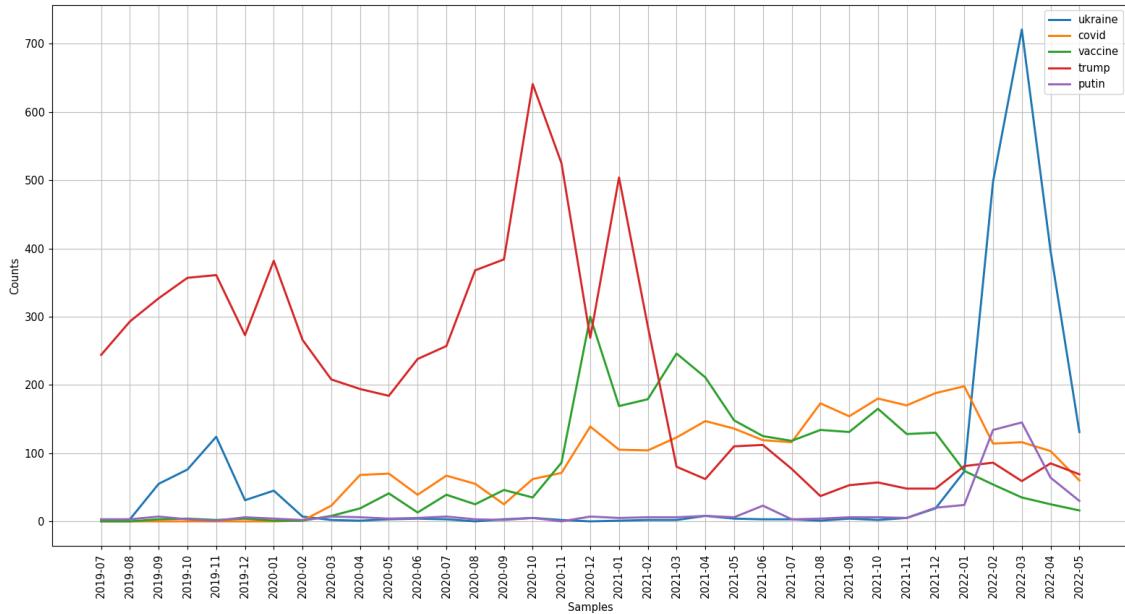


Al contrario di quanto visto per i giornali italiani, in questo caso è interessante l’analisi del Wall Street Journal e del New York Times presi singolarmente. Di seguito i rispettivi grafici.

Wall Street Journal



New York Times:



Si può notare una importante differenza. Relativamente al diffondersi della pandemia, nel New York Times sembrerebbe sia stato dato molto meno spazio al tema covid, con la parola ‘covid’ che fatica a prendere quota, e che per tutto il 2021 resta sempre molto distanziata rispetto alla parola ‘trump’.

[File retweet_like_tweets.py](#)

La terza ed ultima analisi svolta è contenuta nel file retweet_like_tweets.py nel quale è definita la classe CountTweetRetweetLike:

```
class CountTweetRetweetLike:  
    """  
        given a specific newspaper or a group of newspapers, returns the  
        graph with the time trend of tweets, retweets and likes of the givens newspaper(s)  
    """  
    def __init__(self, dataframe, lang, newspaper='all newspapers'): #language of the newspaper (or newspapers) under analysis  
        self.lang = lang  
        self.newspaper = newspaper #newspaper (or newspapers) under analysis  
        self.dataframe = dataframe #complete dataframe with all tweets of clean_scraped_tweets directory  
  
        months = sorted(list(set([tweet.date[0:7] for tweet in self.dataframe.itertuples()]))) #months  
  
        #calling methods  
        retweets4month = self.count_retweets(months)  
        likes4month = self.count_likes(months)  
        tweets4month = self.count_tweets(months)  
  
        self.plot_all(months, retweets4month, likes4month, tweets4month)
```

Dati in ingresso: uno specifico giornale o un gruppo di giornali i cui tweet sono scritti nella medesima lingua; la lingua in cui sono scritti il giornale(i giornali) preso(i) in considerazione

La classe restituisce: una rappresentazione grafica del trend temporale del numero dei tweet pubblicati dai giornali oggetto d'analisi, dei loro like, e dei loro retweet.

Occorre quindi specificare i seguenti parametri:

- **dataframe**: il dataframe completo ottenuto dalla concatenazione
- **lang**: il linguaggio in cui sono scritti i tweet dei giornali che si stanno prendendo in considerazione per l'analisi;
- **newspaper**: il giornale (o il gruppo di giornali) che si sta prendendo in considerazione per l'analisi.

NB: il parametro newspaper è impostato di default a “all newspapers”, in questo modo, se l'utente non specifica nessun giornale, verranno presi in considerazione tutti i giornali appartenenti alla lingua selezionata.

Viene inoltre creata una lista months nella quale vengono inseriti, in ordine cronologico e con annesso anno, i vari mesi del periodo temporale preso in considerazione.

Come visibile dalla figura sopra, nell'init della classe CountRetweetLike, vengono chiamati uno dopo l'altro i seguenti metodi:

- **count_retweets()**
- **count_likes()**
- **count_tweets()**
- **plot_all()**

Il primo metodo ad essere chiamato è **count_retweets()**:

```
def count_retweets(self, months):
    """
    counts month by month the retweets of the tweets published by a newspaper or a group of newspapers
    """
    retweets4month = [np.sum([tweet.retweets for tweet in self.dataframe.itertuples()
                             if (tweet.date == month and tweet.lang == self.lang
                                 and (self.newspaper == tweet.newspaper or self.newspaper == 'all newspapers'))])
                      for month in months]
```

Esso permette di calcolare, mese per mese, il numero di retweet dei tweet pubblicati dai giornali che si stanno prendendo in considerazione. La somma è eseguita tramite il metodo **sum()** della libreria Numpy, sfruttando la lista months definita in precedenza.

Il secondo metodo ad essere chiamato è **count_likes()**:

```
def count_likes(self, months):
    """
    counts month by month the likes of the tweets published by a newspaper or a group of newspapers
    """
    likes4month = [np.sum([tweet.likes for tweet in self.dataframe.itertuples()
                           if (tweet.date == month and tweet.lang == self.lang
                               and (self.newspaper == tweet.newspaper or self.newspaper == 'all newspapers'))])
                  for month in months]

    return likes4month
```

Esso permette di calcolare, mese per mese, il numero di retweet dei tweet pubblicati dai giornali che si stanno prendendo in considerazione. La somma è eseguita tramite il metodo `sum()` della libreria Numpy, andando a contare tweet per tweet iterando nel dataframe completo tramite il metodo `iteruples()` della libreria Pandas.

Il terzo metodo ad essere chiamato è `count_tweets()`:

```
def count_tweets(self, months):
    """
    counts month by month the tweets published by a newspaper or a group of newspapers
    """
    tweets4month = [np.sum([1 for tweet in self.dataframe.iteruples()
                           if (tweet.date == month and tweet.lang == self.lang
                               and (self.newspaper == tweet.newspaper or self.newspaper == 'all newspapers'))])
                    for month in months]

    return tweets4month
```

Esso permette di calcolare, mese per mese, il numero dei tweet pubblicati dai giornali che si stanno prendendo in considerazione. La somma è eseguita tramite il metodo `sum()` della libreria Numpy, sfruttando la lista `months` definita in precedenza, esattamente con la stessa logica dei due metodi sopra.

I tre metodi appena visti restituiscono dunque rispettivamente:

- una lista con il numero di retweet mese per mese
- una lista con il numero di like mese per mese
- una lista con il numero di tweet pubblicati mese per mese

Questi tre output, vanno a costituire l'input del quarto ed ultimo metodo chiamato nella classe `CountRetweetLike`, il metodo `plot_all()`:

```
def plot_all(self, months, retweets4month, likes4month, tweets4month):
    """
    plots the graph
    """
    # Create figure with secondary y-axis
    fig = make_subplots(specs=[[{"secondary_y": True}]])
    # noinspection PyTypeChecker
    fig.add_trace(
        go.Scatter(x=months, y=tweets4month, name="Tweet", mode='lines', line=dict(width=7, color='black'),
                   line_shape='spline'), secondary_y=True)
    # noinspection PyTypeChecker
    fig.add_trace(go.Bar(x=months, y=likes4month, name="Like", marker_color='blue'), secondary_y=False)
    # noinspection PyTypeChecker
    fig.add_trace(go.Bar(x=months, y=retweets4month, name="Retweet", marker_color='orange'),
                  secondary_y=False)

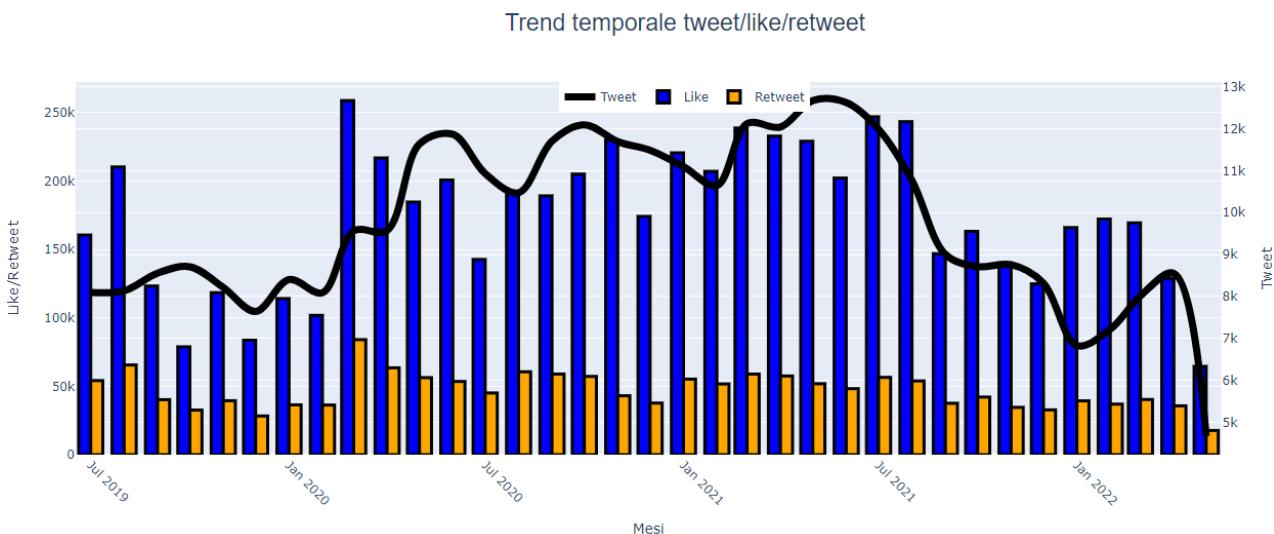
    fig.update_traces(marker_line_color='black', marker_line_width=3)

    # Add figure title
    fig.update_layout(barmode='group', title_text="Trend temporale tweet/like/retweet", title_font_family='Arial',
                      title_x=0.5, title_font_size=23,
                      legend=dict(orientation='h', xanchor="center", x=0.5, y=1))
    # Set x-axis title
    fig.update_xaxes(title_text="Mesi", tickangle=45)
    # Set y-axes titles
    fig.update_yaxes(title_text="Like/Retweet", secondary_y=False) # left yaxes
    fig.update_yaxes(title_text="Tweet", secondary_y=True) # right yaxes
    fig.show()
```

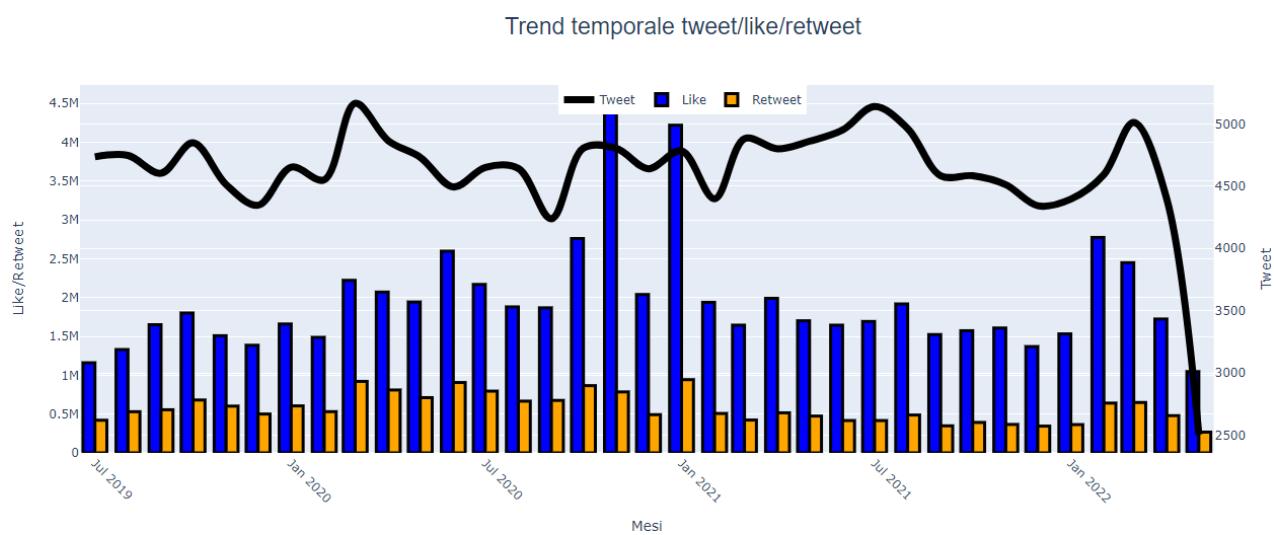
Il metodo consente di plottare, in un unico grafico, l'andamento mese per mese del numero di tweet, retweet, e like.

Il grafico è disegnato sfruttando la libreria grafica plotly, che ci ha consentito di creare un grafico con doppio asse e totalmente interattivo in fase di consultazione html.

La scelta di creare due assi è dettata dalla discrepanza di valori tra il numero di retweet/like e il numero dei tweet stessi. Di seguito è riportato il grafico i cui valori sono stati ottenuti dai dati di tutte le testate giornalistiche italiane:



Di seguito invece il grafico relativo ai quotidiani statunitensi:



Per quanto riguarda i giornali italiani, effettivamente si può notare una crescita nel numero di tweet pubblicati che va da marzo 2020, proprio in corrispondenza del diffondersi della pandemia in Italia, fino al luglio 2021, per poi riassestarsi ai livelli pre-covid.

Analogamente è possibile riscontrare nel medesimo arco temporale, seppur in maniera meno evidente, un aumento dei like, e in maniera molto meno marcata anche un leggero aumento dei retweet.

Per quanto riguarda invece i quotidiani inglesi è difficile riscontrare trend verso l'alto o verso il basso in maniera evidente, sia per quanto riguarda i tweet pubblicati, sia per quanto riguarda like e retweet.

Topic modeling: branch tweets-topic-modeling

Terminata l'analisi preliminare, e ottenuta una prima idea sugli argomenti trattati dai quotidiani italiani e americani e sui loro possibili cambiamenti nel tempo, passiamo alla topic modeling.

Un problema di topic modeling non è nient'altro che un problema relativo a determinare quali sono gli argomenti che vengono trattati all'interno di un corpus di documenti, che nel nostro caso saranno dei tweet.

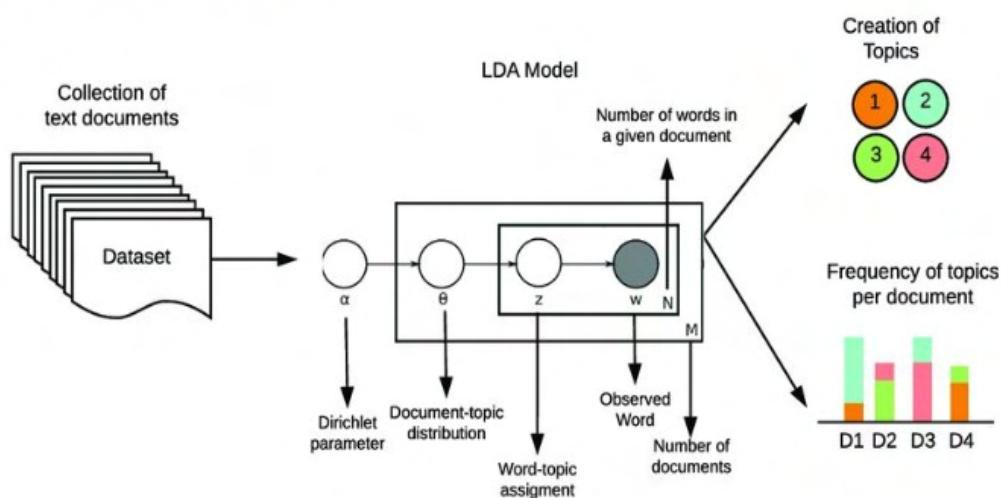
Nel natural language processing, e più in generale nell'apprendimento statistico, gli algoritmi di topic modeling sono modelli statistici non supervisionati che cercano di associare un argomento ad un documento appartenente ad una collezione di documenti. Il concetto base, è che i documenti che trattano lo stesso argomento, hanno una probabilità maggiore di contenere termini simili rispetto a documenti che trattano altri argomenti. La topic modeling consente quindi di trovare i termini che compongono un particolare topic, e di raggruppare in maniera automatica documenti con lo stesso topic.

LDA

Uno degli algoritmi più utilizzati in questo contesto è la *LDA (Latent Dirichlet Allocation)*, ed è l'algoritmo con cui si è scelto di implementare la nostra analisi di topic modeling.

La LDA è un algoritmo probabilistico che estrae, a partire da un corpus di documenti, i vari topic di quei documenti. Il tutto basandosi sulle co-occorrenze dei termini nei documenti, e sulle distribuzioni a priori ed a posteriori delle parole all'interno dei topic e dei topic all'interno dei documenti.

Latent Dirichlet Allocation (LDA)



La ratio che c'è dietro questo modello è che si possono rappresentare i singoli documenti come un mix di topic. Quindi, dato un corpus di documenti e il numero di topics che si vengono chiesti all'LDA, si genererà una lista di parole con un certo peso, ed ogni topic non sarà altro che una combinazione lineare delle parole chiave individuate.

L'obiettivo è quello di determinare la distribuzione di parole all'interno dei topic e la distribuzione dei topic all'interno dei documenti. Una cosa importante, quando lavoriamo con la LDA è che dire che un documento appartiene a un topic non è corretto, si è infatti detto che la LDA come presupposto si basa sul fatto che il documento è un mix di più topics.

Allora, quando si dice che secondo la LDA un documento appartiene ad un determinato topic, si sta in realtà utilizzando un concetto di topic-dominante, cioè stiamo prendendo in considerazione il topic che ha la percentuale più alta di contributo per quel documento.

Si è scelto di applicare l'algoritmo di topic modeling sia mediante l'utilizzo della libreria Scikit-learn, sia mediante l'utilizzo della libreria Gensim, entrambe viste durante il corso.

In particolare, si è cercato di sfruttare al massimo le informazioni date dal codice visto nel corso, adattandolo in modo tale da combinare gli output ottenuti con Scikit-learn con quelli ottenuti con Gensim.

Ad esempio, il codice di Scikit-learn è stato adattato in modo tale da stampare, per ogni topic, non solo le sue 10 parole più rappresentative, ma anche i loro pesi, come visto nel codice di Gensim.

Viceversa, nel codice relativo alla libreria Gensim, si è ad esempio fatto in modo che venissero stampati il numero dei documenti nei quali ciascun topic domina, sfruttando il codice visto con Scikit-learn.

Oltre ai file requirements.txt e README.md, nel branch sono presenti i seguenti file:

- main.py che permette di chiamare le classi che si occuperanno della topic modeling
- topic_skl.py nel quale viene implementata l'LDA con Scikit-learn
- topic_gsm.py nel quale viene implementata l'LDA con Gensim

Anche in questo caso, ci importiamo la nostra cartella di tweet puliti digitando da terminale:

git checkout tweets-pre-processing clean_scraped_tweets

File main.py

Nel file main.py, esattamente come per il branch precedente, si itera per ciascun file csv pulito contenuto nella cartella clean_scraped_tweets.

Per ogni file, grazie al metodo *read_csv()* della libreria Pandas, ricaviamo il dataframe corrispondente e, per mezzo del metodo *concat()* della libreria Pandas, lo concateniamo con il dataframe ricavato dal file csv dell'iterazione precedente.

Una volta iterato per ogni file csv della cartella, si ottiene il dataframe_complete, ovvero il dataframe completo dato dalla concatenazione dei dataframe ricavati da ciascun file csv:

```
if __name__ == "__main__":
    rootdir = 'clean_scraped_tweets'

    dataframe_complete = pd.DataFrame(columns=['text', 'date', 'newspaper', 'likes', 'retweets', 'lang', 'period'])

    # iterating through all the csv files in the clean_scraped_tweets directory
    # returning a dataframe
    # dataframe is obtained by concatenating the dataframes derived from each of the csv files of the directory

    for subdir, dirs, files in os.walk(rootdir):
        for file in files:
            path = os.path.join(subdir, file)
            dataframe = pd.read_csv(path)
            # print(type(dataframe))
            dataframe_complete = pd.concat([dataframe, dataframe_complete])
```

Il dataframe completo risulterà utile nella chiamata delle diverse classi che permetteranno l'esecuzione della topic modeling.

Ci consentirà infatti, nelle singole chiamate degli oggetti delle classi, di poter selezionare di volta in volta solo specifiche righe del dataframe stesso, sulla base di quali giornali e di quali periodi si vogliono di volta in volta considerare nella topic modeling

Una volta ottenuto il complete_dataframe, si può procedere alla chiamata degli oggetti delle varie classi.

Di seguito alcune chiamate a titolo esemplificativo per ciascuno dei due file di topic modeling:

```

#calling objects of class TopicScikit

TopicScikit(dataframe_complete, 'precovid', 'it', "repubblica").topic_modeling_starter()
TopicScikit(dataframe_complete, 'precovid', 'it', "corriere").topic_modeling_starter()
TopicScikit(dataframe_complete, 'precovid', 'it', "ilgiornale").topic_modeling_starter()
TopicScikit(dataframe_complete, 'precovid', 'it').topic_modeling_starter()
TopicScikit(dataframe_complete, 'covid', 'it', "repubblica").topic_modeling_starter()
TopicScikit(dataframe_complete, 'covid', 'it', "corriere").topic_modeling_starter()
TopicScikit(dataframe_complete, 'covid', 'it', "ilgiornale").topic_modeling_starter()
TopicScikit(dataframe_complete, 'covid', 'it').topic_modeling_starter()
TopicScikit(dataframe_complete, 'war', 'it', "repubblica").topic_modeling_starter()
TopicScikit(dataframe_complete, 'war', 'it', "corriere").topic_modeling_starter()
TopicScikit(dataframe_complete, 'war', 'it', "ilgiornale").topic_modeling_starter()
TopicScikit(dataframe_complete, 'war', 'it').topic_modeling_starter()

#Calling objects of class TopicGensim

TopicGensim(dataframe_complete, 'precovid', 'it', "repubblica").topic_modeling_starter()
TopicGensim(dataframe_complete, 'precovid', 'it', "corriere").topic_modeling_starter()
TopicGensim(dataframe_complete, 'precovid', 'it', "ilgiornale").topic_modeling_starter()
TopicGensim(dataframe_complete, 'precovid', 'it').topic_modeling_starter()
TopicGensim(dataframe_complete, 'covid', 'it', "repubblica").topic_modeling_starter()
TopicGensim(dataframe_complete, 'covid', 'it', "corriere").topic_modeling_starter()
TopicGensim(dataframe_complete, 'covid', 'it', "ilgiornale").topic_modeling_starter()
TopicGensim(dataframe_complete, 'covid', 'it').topic_modeling_starter()
TopicGensim(dataframe_complete, 'war', 'it', "repubblica").topic_modeling_starter()
TopicGensim(dataframe_complete, 'war', 'it', "corriere").topic_modeling_starter()
TopicGensim(dataframe_complete, 'war', 'it', "ilgiornale").topic_modeling_starter()
TopicGensim(dataframe_complete, 'war', 'it').topic_modeling_starter()

```

File topic_skl.py

Nel file topic_skl.py definiamo la classe TopicScikit, che permette l'esecuzione della topic modeling con Scikit-learn:

```

class TopicScikit:
    """
        Topic modeling with Scikit-learn library
    """
    def __init__(self, dataframe, period, lang, newspaper='all newspapers'):

        self.dataframe = dataframe      #complete_dataframe with all tweets of clean_scraped_tweets directory
        self.period = period           #historical_period under analysis
        self.lang = lang                #language of the newspaper (or newspapers) under analysis
        self.newspaper = newspaper     #newspaper (or newspapers) under analysis

```

Dati in ingresso: uno specifico giornale o un gruppo di giornali i cui tweet sono scritti nella medesima lingua; uno specifico periodo storico; la lingua in cui sono scritti il giornale(i giornali) preso(i) in considerazione.

La classe restituisce: tutti gli output su schermo, le tabelle, i grafici, relativi alla topic modeling eseguita con Scikit-learn

Occorre quindi specificare i seguenti parametri:

- **dataframe**: il dataframe completo ottenuto dalla concatenazione
- **lang**: il linguaggio in cui sono scritti i tweet dei giornali che si stanno prendendo in considerazione per la topic modeling;
- **period**: il periodo storico che si sta prendendo in considerazione per la topic modeling;
- **newspaper**: il giornale (o il gruppo di giornali) che si sta prendendo in considerazione per la topic modeling .

NB: il parametro newspaper è impostato di default a “all newspapers”, in questo modo, se l'utente non specifica nessun giornale, verranno presi in considerazione tutti i giornali appartenenti alla lingua selezionata.

Premessa:

La libreria Scikit-learn, nell'ambito della topic modeling, mette a disposizione la funzione GridSearchCV. GridSearchCV permette di passare, per ogni parametro relativo alla topic modeling, una serie di valori, per poi effettuare la topic modeling tenendo automaticamente conto della migliore combinazione possibile tra tutti i parametri. È inoltre possibile stampare i migliori parametri ottenuti. Di conseguenza, l'idea è quella di sfruttare GridSearchCV nell'ambito della topic modeling con Scikit-learn, e salvare di volta in volta, per ogni topic modeling eseguita, i parametri migliori restituiti dalla grid search. In questo modo i parametri potranno essere riutilizzati nell'ambito della topic modeling con Gensim.

Nella classe TopicScikit è definito il metodo `topic_modeling_starter()`. Si tratta del metodo che viene chiamato dal file main.py relativamente agli oggetti della classe TopicScikit:

```
def topic_modeling_starter(self):  
    #passing the documents  
    documents = [tweet.text.split() for tweet in self.dataframe.itertuples() if (  
        tweet.period == self.period and tweet.lang == self.lang and (  
            tweet.newspaper == self.newspaper or self.newspaper == 'all newspapers'))][:10]  
  
    tf_vectorizer = CountVectorizer(tokenizer=self.ghost_tokenizer,  
                                    preprocessor=self.ghost_tokenizer,  
                                    max_df=0.8, max_features=1500  
                                    )  
    data_vectorized = tf_vectorizer.fit(documents)  
    data_vectorized = tf_vectorizer.transform(documents)  
    tf_feature_names = tf_vectorizer.get_feature_names_out()  
  
    #LDA  
    lda = LatentDirichletAllocation(learning_method='online')  
    search_params = {'n_components': [n for n in range(3, 10)],  
                    'learning_decay': [.5, .7, .9], 'max_iter': [50, 75, 100]}  
    grid_lda = GridSearchCV(lda, param_grid=search_params)  
  
    # Best Model  
    grid_lda.fit(data_vectorized)  
    ldaBestModel = grid_lda.best_estimator_  
    ldaData = ldaBestModel.transform(data_vectorized)  
    topics = len(ldaBestModel.components_ )  
    print(ldaBestModel.components_, 'best model')  
  
    # Model Parameters  
    print("Best Model's Params: %s" % grid_lda.best_params_)  
    # Log Likelihood Score  
    print("Best Log Likelihood Score: %.3f" % grid_lda.best_score_)  
    # Perplexity  
    print("Model Perplexity: %.3f" % ldaBestModel.perplexity(data_vectorized))  
  
    #calling methods  
    df_document_topics = self.dominant_topic_report(topics,  
                                                    ldaData,  
                                                    len(documents))  
  
    df_topic_distribution = self.topic_distribution(df_document_topics)  
    self.make_topic_distribution_plot(df_topic_distribution)  
    self.report_topic_key_words(ldaBestModel.components_,  
                               tf_vectorizer.get_feature_names_out())  
  
    self.report_tweets_dominant_topic(  
        documents, ldaData, ldaBestModel.components_, tf_feature_names)  
    self.update_parameters(grid_lda)
```

Nel metodo `topic_modeling_starter()` viene costruito il nostro corpus di documenti, viene eseguita la vettorizzazione (tramite il CountVectorizer che conta le occorrenze delle parole) e chiamati i metodi `fit()` e `transform()` dell'algoritmo di LDA.

Inoltre, come anticipato nella premessa, alla GridSearchCV vengono passati i possibili valori dei vari parametri con i quali essa provvederà a restituire la migliore combinazione.

Abbiamo scelto di inserire nella gridsearch i seguenti parametri con le seguenti possibili combinazioni:

- `n_components`: da 3 a 10
- `learning_decay`: 0.5 ; 0.7; 0.9
- `max_iter`: 50, 75, 100

I migliori parametri vengono poi stampati, e con essi, anche le due metriche Likelihood e Perplexity, che offrono una misura numerica della bontà della topic modeling: più la Likelihood è alta più il modello performa bene; più la Perplexity è bassa, più il modello performa bene. Vengono poi chiamati tutti i successivi metodi, che permettono di ottenere in output i risultati ed i grafici relativi alla topic modeling.

Il primo metodo ‘chiamato in causa’ è il metodo `ghost_tokenizer()` che viene attivato dal CountVectorizer:

```
def ghost_tokenizer(self, doc):
    """
    allows to avoid tokenization and pre-processing
    ...
    return doc
```

Di per sé infatti, il CountVectorizer tokenizza e pre-processa i documenti che gli vengono dati in input. In questo caso però, i nostri tweet erano già stati tokenizzati e pre-processati nel secondo branch tramite la libreria Spacy. Proprio per questo abbiamo provveduto alla creazione di un tokenizer ‘fantasma’ che si limita a restituire come output ciò che gli viene passato in input, andando così ad ‘ingannare’ il CountVectorizer.

Il secondo metodo chiamato è `dominant_topic_report()`:

```
def dominant_topic_report(self, topics, ldaTransformedDate, numberOfRowsDocuments):
    """
    prints a table showing for each document: the various topics with the relative weights in the document;
    the dominant topic of the document
    ...
    # column names
    topicnames = ["Topic" + str(i) for i in range(topics)]
    # index names
    docnames = ["Doc" + str(i) for i in range(numberOfDocuments)]
    # Make the pandas dataframe
    df_document_topic = pd.DataFrame(np.round(ldaTransformedDate, 2),
                                       columns=topicnames,
                                       index=docnames)

    dominantTopic = np.argmax(df_document_topic.values,
                             axis=1)
    df_document_topic['dominant_topic'] = dominantTopic
    # Apply Style
    df_document_topics = df_document_topic.head(100).style.applymap(self.color_green).applymap(self.make_bold)
    path = f'./report/report_scikit/{self.period}'
    if not os.path.exists(path): os.makedirs(path)
    with open(f'{path}/ldaReport_{self.newspaper}_{self.lang}.html', 'w') as fileWriter:
        fileWriter.write(df_document_topics.render())
    print(tabulate(df_document_topic.head(15), headers='keys', tablefmt='psql'))
    return df_document_topic
```

Permette di stampare una tabella in cui ogni riga è rappresentata da ciascun documento, ed in cui le varie colonne indicano il peso di ciascun topic all'interno di quel documento. L'ultima colonna restituisce l'indicazione del topic dominante per quel documento.

	Topic0	Topic1	Topic2	Topic3	Topic4	dominant_topic
Doc0	0.1	0.1	0.1	0.6	0.1	3
Doc1	0.03	0.03	0.03	0.73	0.18	3
Doc2	0.2	0.03	0.7	0.03	0.03	2
Doc3	0.03	0.37	0.03	0.37	0.2	1
Doc4	0.8	0.05	0.05	0.05	0.05	0
Doc5	0.04	0.04	0.24	0.64	0.04	3
Doc6	0.05	0.05	0.8	0.05	0.05	2
Doc7	0.05	0.05	0.8	0.05	0.05	2
Doc8	0.07	0.07	0.07	0.73	0.07	3
Doc9	0.07	0.73	0.07	0.07	0.07	1
Doc10	0.55	0.05	0.3	0.05	0.05	0
Doc11	0.07	0.07	0.07	0.73	0.07	3
Doc12	0.4	0.07	0.07	0.07	0.4	0
Doc13	0.07	0.73	0.07	0.07	0.07	1
Doc14	0.07	0.07	0.07	0.07	0.73	4

La tabella è ricavata dopo aver costruito il dataframe df_document_topic, che provvediamo a convertire in tabella tramite la libreria tabulate, che consente di convertire dataframe in tabelle più facilmente leggibili dall'utente. La stessa libreria verrà utilizzata in questo branch ogni qualvolta ci sia l'esigenza di stampare il contenuto di un dataframe.

La stessa tabella appena descritta viene anche salvata in formato html nella cartella report:

	Topic0	Topic1	Topic2	Topic3	Topic4	dominant_topic
Doc0	0.100000	0.100000	0.100000	0.600000	0.100000	3
Doc1	0.030000	0.030000	0.030000	0.730000	0.180000	3
Doc2	0.200000	0.030000	0.700000	0.030000	0.030000	2
Doc3	0.030000	0.370000	0.030000	0.370000	0.200000	1
Doc4	0.800000	0.050000	0.050000	0.050000	0.050000	0
Doc5	0.040000	0.040000	0.240000	0.640000	0.040000	3
Doc6	0.050000	0.050000	0.800000	0.050000	0.050000	2
Doc7	0.050000	0.050000	0.800000	0.050000	0.050000	2
Doc8	0.070000	0.070000	0.070000	0.730000	0.070000	3
Doc9	0.070000	0.730000	0.070000	0.070000	0.070000	1
Doc10	0.550000	0.050000	0.300000	0.050000	0.050000	0

Si è fatto in modo che la cartella report, le sue subdirectories, ed il nome del file html, venissero creati in automatico sulla base di: libreria utilizzata per la topic modeling; giornale preso in considerazione; periodo storico preso in considerazione.

Viene inoltre restituito il dataframe df_document_topic, che sarà utile come input per il successivo metodo.

Il terzo metodo chiamato è `topic_distribution()` che prende in ingresso come input l'output del metodo precedente:

```
def topic_distribution(self, df_document_topic_frame):
    """
        prints a table in which each topic number is associated with
        the number of documents in which it is dominant
    """
    df_topic_distribution = df_document_topic_frame['dominant_topic'].value_counts().reset_index(
        name="Num Documents")
    df_topic_distribution.columns = ['TopicNum', 'NumDocuments']
    print(tabulate(df_topic_distribution, headers='keys', tablefmt='psql', showindex=False))
    return df_topic_distribution
```

Permette di stampare una tabella nella quale ad ogni topic è associato il numero dei documenti in cui è contrassegnato come topic dominante:

TopicNum	NumDocuments
0	2086
1	1910
3	1902
2	1547
4	1087

Anche in questo caso viene inoltre restituito il dataframe `df_topic_distribution` che costituirà l'input del metodo successivo.

Il quarto metodo chiamato è `make_topic_distribution_plot()` che prende in ingresso come input l'output del metodo precedente:

```
def make_topic_distribution_plot(self, df_topic):
    """
        plots a barplot in which each bar is a topic and the height of the bar
        represents the number of documents in which that topic is dominant
    """
    df_topic["Color"] = np.where(df_topic["NumDocuments"] == max(df_topic['NumDocuments']), 'darkblue',
                                 'darkorange')
    topics = [f'TOPIC {topic.TopicNum}' for topic in df_topic.itertuples()]

    fig = make_subplots()
    fig.add_trace(go.Bar(x=topics, y=df_topic['NumDocuments'], marker_color=df_topic['Color']))
    fig.update_layout(title_text="Topic Distribution Scikit", title_font_family='Arial',
                      title_x=0.5, title_font_size=23,
                      legend=dict(orientation='h', xanchor="center", x=0.5, y=1))

    fig.update_traces(marker_line_color='black', marker_line_width=3)

    fig.update_xaxes(title_text="Topic", tickangle=0)

    fig.update_yaxes(title_text="N_documents") # left yaxes
    fig.show()
```

Permette di disegnare, tramite la libreria plotly, un grafico a barre interattivo in cui ogni barra rappresenta il topic, e dove l'altezza di ogni barra è data dal numero di documenti in cui quel topic risulta dominante:

Topic Distribution Scikit



Il quinto metodo chiamato è `report_topic_key_words()`:

```
def report_topic_key_words(self, topics, featureNames, topWords=10):
    """
    prints a table in which for each topic the 10 topwords
    of that topic are printed with the relative weights
    """

    keywords = np.array(featureNames)
    topic_keywords = []
    for topicWeights in topics:
        topKeyWordIndexesAndWeights = []
        topKeywordDocs = topicWeights.argsort()[:-1 - topWords:-1]
        pesi = sorted(topicWeights, reverse=True)
        for i in range(0, len(topKeywordDocs)):
            topKeyWordIndexesAndWeights.append((topKeywordDocs[i], round(pesi[i], 2)))
        topKeywordsAndWeights = [(keywords.take(wordindex_weight[0]), wordindex_weight[1]) for wordindex_weight
                                 in topKeyWordIndexesAndWeights]
        topKeyWordsWithWeights = [f'{word_weight[0]},{word_weight[1]}' for word_weight in
                                  topKeywordsAndWeights]

        topic_keywords.append(topKeyWordsWithWeights)

    df_topic_keywords = pd.DataFrame(topic_keywords)

    df_topic_keywords.columns = ['Word %d' % i for i in range(df_topic_keywords.shape[1])]
    df_topic_keywords.index = ['Topic %d' % i for i in range(df_topic_keywords.shape[0])]
    print(tabulate(df_topic_keywords, headers='keys', tablefmt='psql'))
```

Permette di stampare per ogni topic, la top10 delle parole rappresentative di quel topic, con relativo peso.

Il sesto metodo chiamato è `report_tweets_dominant_topic()`:

```
def report_tweets_dominant_topic(self, documents, ldaData, topics, tf_feature_names):
    """
    creates and saves an html report in which for each document its text and its dominant topic are shown;
    for the dominant are then shown its weight and its 10topwords;
    the content of the html report is also printed
    """

    print('REPORT TWEETS DOMINANT TOPIC')
    list_dicts = []
    for n_doc in range(0, len(documents)):
        doc_text = " ".join(documents[n_doc])
        doc_topics = ldaData[n_doc]

        # the dominant topic is the one with the highest value, argmax return the index of max in a numpy array
        doc_dominant_topic = np.array(doc_topics).argmax()
        weight_dominant_topic = max(doc_topics)
        # again top 10 words for the dominant topic of doc0
        top10_topic_word_indexes_of_doc = topics[doc_dominant_topic].argsort()[:-11:-1]
        top10_words_of_doc = " ".join(
            [tf_feature_names[wordIndex] for wordIndex in top10_topic_word_indexes_of_doc])
        list_dicts.append({
            'text': doc_text, 'topic': doc_dominant_topic, 'weight': weight_dominant_topic,
            'top words': top10_words_of_doc})

    df = pd.DataFrame(list_dicts)
    df.index = ['Doc %d' % i for i in range(len(documents))]
    df_style = df.head(15).style
    path = f'./report/report_scikit/{self.period}'
    if not os.path.exists(path): os.makedirs(path)
    with open(f'{path}/ldaReport2_{self.newspaper}_{self.lang}.html', 'w') as fileWriter:
        fileWriter.write(df_style.render())
    print(tabulate(df.head(15), headers='keys', tablefmt='psql'))
```

Permette di stampare, per ogni documento, il suo testo, il topic dominante, il peso del topic dominante, e la top10 delle parole rappresentative del topic dominante:

	text	topic	weight	top words
Doc 0	congedo mestruale discutere spagna	3	0.599994	putin ucraina guerra russo morire russia kiev figlio zelensky mosca
Doc 1	padre figlio campo olimpiadi sordo buone notizie edicola	3	0.734044	putin ucraina guerra russo morire russia kiev figlio zelensky mosca
Doc 2	mascherina scuola lega pressing speranza decidere scienza	2	0.700081	uccidere diretta donna covid morto italia roma presidente green pass
Doc 3	colonna autobus lasciare acciaieria azovstal video	1	0.367069	milano tornare video gas parlare sanzione segreto rischio gara russia
Doc 4	mariupol irriducibile azovstal assedio dilemma resa	0	0.798245	ucraino auto europa inter mariupol italiano vincere attacco famiglia russo
Doc 5	vergognare padre riconoscere scelta giusto	3	0.640041	putin ucraina guerra russo morire russia kiev figlio zelensky mosca
Doc 6	addio giorgio chiellini stadium abbraccio emozione standing ovation	2	0.799998	uccidere diretta donna covid morto italia roma presidente green pass
Doc 7	coming out calciatore jake daniels gay inglese attività dire	2	0.799998	uccidere diretta donna covid morto italia roma presidente green pass
Doc 8	barricati sotterraneo acciaieria resistente azovstal	3	0.732997	putin ucraina guerra russo morire russia kiev figlio zelensky mosca
Doc 9	spiagge intesa ipotesi aumentare indennizzo	1	0.733332	milano tornare video gas parlare sanzione segreto rischio gara russia
Doc 10	sparatoria california odiare taiwan immigrato cinese uniti	0	0.550317	ucraino auto europa inter mariupol italiano vincere attacco famiglia russo
Doc 11	matteo vincitore certamen pensavo impreciso farcee	3	0.73333	putin ucraina guerra russo morire russia kiev figlio zelensky mosca
Doc 12	torino gallerie italia fotografia tesore	0	0.399961	ucraino auto europa inter mariupol italiano vincere attacco famiglia russo
Doc 13	comandante battaglione azov obbediremo ordine evacuazione	1	0.73333	milano tornare video gas parlare sanzione segreto rischio gara russia
Doc 14	berlusconi sorpresa treviglio convention fi comunismo	4	0.733332	italia cambiare usa draghi euro russo chiedere storia vedere succedere

Lo stesso risultato viene ottenuto anche in formato html e salvato nella cartella report:

text	topic	weight	top words
Doc 0 congedo mestruale discutere spagna	3	0.599994	putin ucraina guerra russo morire russia kiev figlio zelensky mosca
Doc 1 padre figlio campo olimpiadi sordo buone notizie edicola	3	0.734044	putin ucraina guerra russo morire russia kiev figlio zelensky mosca
Doc 2 mascherina scuola lega pressing speranza decidere scienza	2	0.700081	uccidere diretta donna covid morto italia roma presidente green pass
Doc 3 colonna autobus lasciare acciaieria azovstal video	1	0.367069	milano tornare video gas parlare sanzione segreto rischio gara russia
Doc 4 mariupol irriducibile azovstal assedio dilemma resa	0	0.798245	ucraino auto europa inter mariupol italiano vincere attacco famiglia russo
Doc 5 vergognare padre riconoscere scelta giusto	3	0.640041	putin ucraina guerra russo morire russia kiev figlio zelensky mosca
Doc 6 addio giorgio chiellini stadium abbraccio emozione standing ovation	2	0.799998	uccidere diretta donna covid morto italia roma presidente green pass
Doc 7 coming out calciatore jake daniels gay inglese attività dire	2	0.799998	uccidere diretta donna covid morto italia roma presidente green pass
Doc 8 barricati sotterraneo acciaieria resistente azovstal	3	0.732997	putin ucraina guerra russo morire russia kiev figlio zelensky mosca
Doc 9 spiagge intesa ipotesi aumentare indennizzo	1	0.733332	milano tornare video gas parlare sanzione segreto rischio gara russia
Doc 10 sparatoria california odiare taiwan immigrato cinese uniti	0	0.550317	ucraino auto europa inter mariupol italiano vincere attacco famiglia russo
Doc 11 mattheo vincitore certamen pensavo impreciso farcee	3	0.73333	putin ucraina guerra russo morire russia kiev figlio zelensky mosca
Doc 12 torino gallerie italia fotografia tesore	0	0.399961	ucraino auto europa inter mariupol italiano vincere attacco famiglia russo
Doc 13 comandante battaglione azov obbediremo ordine evacuazione	1	0.73333	milano tornare video gas parlare sanzione segreto rischio gara russia
Doc 14 berlusconi sorpresa treviglio convention fi comunismo	4	0.733332	italia cambiare usa draghi euro russo chiedere storia vedere succedere

L'ultimo metodo chiamato è `update_parameters()`:

```
def update_parameters(self, grid_lda):
    """
        writes to a txt file the best parameters returned by the topic modeling
    """
    openfile = open(f'./topic_parameters/{self.period}_{self.newspaper}_parameters.txt', 'w')
    openfile.write(f'best parameters are: {grid_lda.best_params_}')
    openfile.close()
```

Permette di scrivere un file di testo in cui vengono salvati i parametri migliori ottenuti dalla topic modeling, e li salva nella cartella `topic_parameters`.

I parametri saranno utili per lo svolgimento della topic modeling con Gensim.

File `topic_gsm.py`

Nel file `topic_gsm.py` definiamo la classe `TopicGensim`, che permette l'esecuzione della topic modeling con Gensim:

```
class TopicGensim:
    """
        Topic modeling with Gensim library
    """
    def __init__(self, dataframe, period, lang, newspaper='all newspapers'):

        self.dataframe = dataframe #complete dataframe with all tweets of clean_scraped tweets directory
        self.period = period #historical period under analysis
        self.lang = lang #language of the newspaper (or newspapers) under analysis
        self.newspaper = newspaper #newspaper (or newspapers) under analysis
```

Sui dati in ingresso e sui parametri da specificare valgono le stesse considerazioni effettuate per Scikit- learn. Anche nella classe `TopicGensim` è definito il metodo `topic_modeling_starter()`. Si tratta del metodo che viene chiamato dal file `main.py` relativamente agli oggetti della classe `TopicGensim`:

```
def topic_modeling_starter(self):

    documents = [tweet.text.split() for tweet in self.dataframe.itertuples() if (
        tweet.period == self.period and tweet.lang == self.lang and (
            tweet.newspaper == self.newspaper or self.newspaper == 'all newspapers'))]

    id2word = corpora.Dictionary(documents)
    # Create Corpus: Term Document Frequency
    corpus = [id2word.doc2bow(text) for text in documents]
    # Build LDA model
    lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                                id2word=id2word,
                                                num_topics=4,
                                                iterations=100,
                                                decay=0.5,
                                                random_state=100,
                                                update_every=1,
                                                chunksize=10,
                                                passes=10,
                                                alpha='symmetric',
                                                per_word_topics=True)

    pprint(lda_model.print_topics())

    #calling methods
    df_topic_sents_keywords = self.format_topics_sentences(ldamodel=lda_model, corpus=corpus, texts=documents)

    # Format
    df_dominant_topic = df_topic_sents_keywords.reset_index()
    df_dominant_topic.columns = ['Document_No', 'Dominant_Topic', 'Topic_Perc_Contrib', 'Keywords', 'Text']
    print(tabulate(df_dominant_topic.head(15), headers='keys', tablefmt='psql', showindex=False))

    self.count_dominant_topics(df_dominant_topic)

    #Visualize HTML reports of topics and topic clusters
    self.show_topic_clusters(lda_model, corpus, n_topics=4)
    self.visualize_topics(lda_model, corpus)
```

Nel [topic_modeling_starter\(\)](#) vengono definiti i documenti ed i parametri con i quali deve essere eseguito l'algoritmo di topic modeling LDA.

Si ricorda che i parametri num_topics, iterations, e decay vengono definiti in base alla combinazione di migliori parametri risultante dalla topic modeling eseguita con Scikit-learn.

Vengono poi stampate per ogni topic, la top10 delle parole che rappresentano quel topic, con i relativi pesi:

```
[(),  
 '0.034*"votare" + 0.025*"stella" + 0.019*"diretta" + 0.015*"amadeus" + '  
 '0.013*"italia" + 0.012*"salvini" + 0.012*"suv" + 0.010*"indicare" + '  
 '0.009*"giro" + 0.008*"terzo"),  
(1,  
 '0.028*"appena" + 0.026*"morire" + 0.014*"italiano" + 0.014*"palco" + '  
 '0.011*"mascherina" + 0.011*"fuga" + 0.010*"pass" + 0.010*"green" + '  
 '0.010*"europeo" + 0.009*"regola"),  
(2,  
 '0.038*"gara" + 0.023*"covid" + 0.018*"vedere" + 0.010*"mattarella" + '  
 '0.009*"pubblico" + 0.009*"milano" + 0.008*"cambiare" + 0.008*"febbraio" + '  
 '0.008*"terra" + 0.007*"euro"),  
(3,  
 '0.054*"sanremo2022" + 0.026*"piacere" + 0.019*"esibire" + 0.016*"uccidere" + '  
 '+ 0.016*"serata" + 0.016*"valutazione" + 0.014*"indicare" + 0.014*"tornare" + '  
 '+ 0.013*"blanco" + 0.011*"mahmood"),  
(4,  
 '0.038*"sanremo" + 0.030*"canzone" + 0.017*"pandemia" + 0.014*"padre" + '  
 '0.013*"festival" + 0.011*"storia" + 0.011*"voto" + 0.010*"ubriaco" + '  
 '0.008*"lauro" + 0.007*"achille")]
```

Successivamente vengono chiamati i diversi metodi della classe TopicGensim, che permettono di visualizzare i risultati della topic modeling tramite grafici interattivi e tabelle.

Il primo metodo ad essere chiamato è [format_topic_sentences\(\)](#):

```
def format_topics_sentences(self, ldamodel=None, corpus=None, texts=None):  
    """  
        returns a dataframe in which for each topic results the text, the dominant topic,  
        the weight of the dominant topic, the top10 words of the dominant topic  
    """  
    # Init output  
    sent_topics_df = pd.DataFrame()  
  
    # Get main topic in each document  
    for i, row_list in enumerate(ldamodel[corpus]):  
        if len(row_list) == 0:  
            continue  
        row = row_list[0] if ldamodel.per_word_topics else row_list  
        if isinstance(row, tuple):  
            row = [row]  
        row = sorted(row, key=lambda x: (x[1]), reverse=True)  
        # Get the Dominant topic, Perc Contribution and Keywords for each document  
        for j, (topic_num, prop_topic) in enumerate(row):  
            if j == 0: # => dominant topic  
                wp = ldamodel.show_topic(topic_num)  
                topic_keywords = ", ".join([word for word, prop in wp])  
                sent_topics_df = sent_topics_df.append(  
                    pd.Series([int(topic_num), round(prop_topic, 4), topic_keywords]), ignore_index=True)  
            else:  
                break  
    sent_topics_df.columns = ['Dominant_Topic', 'Perc_Contribution', 'Topic_Keywords']  
    contents = pd.Series(texts)  
    sent_topics_df = pd.concat([sent_topics_df, contents], axis=1)  
  
    return sent_topics_df
```

Ci restituisce un dataframe nel quale per ogni documento risultano il suo testo, il suo topic dominante, il peso del topic dominante, e le top10 parole rappresentanti il topic dominante.

Il dataframe restituito viene rinominato df_topic_sents_keywords e successivamente viene sistemata la struttura delle colonne. Sempre grazie alla libreria tabulate stampiamo le prime righe del nostro dataframe in modo tale che siano maggiormente leggibili dall'utente:

```
df_topic_sents_keywords = self.format_topics_sentences(ldamodel=lda_model, corpus=corpus, texts=documents)

# Format
df_dominant_topic = df_topic_sents_keywords.reset_index()
df_dominant_topic.columns = ['Document_No', 'Dominant_Topic', 'Topic_Perc_Contrib', 'Keywords', 'Text']
print(tabulate(df_dominant_topic.head(15), headers='keys', tablefmt='psql', showindex=False))
```

Il risultato è il seguente:

Document_No	Dominant_Topic	Topic_Perc_Contrib	Keywords	Text
0	2	0.4644	[gara, covid, vedere, mattarella, pubblico, milano, cambiare, febbraio, terra, euro	['congedo', 'mestruale', 'discutere', 'spagna']
1	3	0.5254	[sanremo2022, piacere, esibire, uccidere, serata, valutazione, indicare, tornare, bianco, mahmood	['padre', 'figlio', 'campo', 'olimpiadi', 'sordo', 'buone', 'notizie', 'edicola']
2	1	0.5376	[appena, morire, italiano, palco, mascherina, fuga, pass, green, europeo, regola	['appena', 'morire', 'italiano', 'palco', 'mascherina', 'fuga', 'pass', 'green', 'europeo', 'regola']
3	1	0.8583	[appena, morire, italiano, palco, mascherina, fuga, pass, green, europeo, regola	['colonna', 'autobus', 'lasciare', 'acciaieria', 'azovstal', 'decidere', 'scienze']
4	1	0.8578	[appena, morire, italiano, palco, mascherina, fuga, pass, green, europeo, regola	['maripoli', 'inriducibile', 'azovstal', 'assesto', 'dilemma', 'resa']
5	1	0.6547	[appena, morire, italiano, palco, mascherina, fuga, pass, green, europeo, regola	['vergognare', 'padre', 'ricomossera', 'scelta', 'giusto']
6	2	0.4814	[gara, covid, vedere, mattarella, pubblico, milano, cambiare, febbraio, terra, euro	['addio', 'piorgio', 'chiellini', 'stadium', 'abbraccio', 'emozione', 'standing', 'ovation']
7	0	0.982	[votare, stella, diretta, amadeus, italia, salvini, sut, indicare, giro, terzo	['coming', 'out', 'calcio', 'jake', 'daniels', 'gay', 'inglese', 'attività', 'dire']
8	2	0.8188	[gara, covid, vedere, mattarella, pubblico, milano, cambiare, febbraio, terra, euro	['baricati', 'sotterraneo', 'acciaieria', 'resistente', 'azovstal']
9	0	0.4295	[votare, stella, diretta, amadeus, italia, salvini, sut, indicare, giro, terzo	['spiegare', 'intesa', 'iostesi', 'aumentare', 'indennizzo']
10	1	0.6591	[appena, morire, italiano, palco, mascherina, fuga, pass, green, europeo, regola	['sparatoria', 'california', 'odiale', 'taiwan', 'immigrato', 'cinese', 'unite']
11	3	0.582	[sanremo2022, piacere, esibire, uccidere, serata, valutazione, indicare, tornare, bianco, mahmood	['matteo', 'vincitore', 'centauro', 'pensavo', 'impreciso', 'farce']
12	0	0.843	[votare, stella, diretta, amadeus, italia, salvini, sut, indicare, giro, terzo	['torino', 'gallerie', 'italia', 'fotografia', 'tesoro']
13	3	0.8588	[sanremo2022, piacere, esibire, uccidere, serata, valutazione, indicare, tornare, bianco, mahmood	['comandante', 'battaglione', 'azov', 'obbediremo', 'ordine', 'evacuazione']
14	0	0.4734	[votare, stella, diretta, amadeus, italia, salvini, sut, indicare, giro, terzo	['berlusconi', 'sorpresa', 'treviglio', 'convention', 'fi', 'comunismo']

Il secondo metodo ad essere chiamato è `count_dominant_topics()`:

```
def count_dominant_topics(self, df_dominant_topic):
    """
    prints a table in which each topic number is associated with
    the number of documents in which it is dominant;
    plots a barplot in which each bar is a topic and the height of the bar
    represents the number of documents in which that topic is dominant
    """

    n_topics = 4
    topics = [n_topic for n_topic in range(n_topics)]
    topics_with_label = [f'TOPIC{n_topic}' for n_topic in range(n_topics)]
    n_documents = [df_dominant_topic['Dominant_Topic'].value_counts()[topic] for topic in topics]
    dic_count_dominant_topics = {'topics': topics, 'n_documents': n_documents}
    df_count_dominant_topics = pd.DataFrame(dic_count_dominant_topics, columns=['topics', 'n_documents'])

    print(tabulate(df_count_dominant_topics, headers='keys', tablefmt='psql', showindex=False))

    #barplot
    df_count_dominant_topics["Color"] = np.where(df_count_dominant_topics["n_documents"] == max(
        df_count_dominant_topics['n_documents']), 'darkblue', 'darkorange')

    fig = make_subplots()

    fig.add_trace(
        go.Bar(x=topics_with_label, y=df_count_dominant_topics['n_documents'],
               marker_color=df_count_dominant_topics['Color']))
    fig.update_layout(title_text="Topic Distribution Gensim", title_font_family='Arial',
                      title_x=0.5, title_font_size=23,
                      legend=dict(orientation='h', xanchor="center", x=0.5, y=1))

    fig.update_traces(marker_line_color='black', marker_line_width=3)

    fig.update_xaxes(title_text="Topic", tickangle=0)

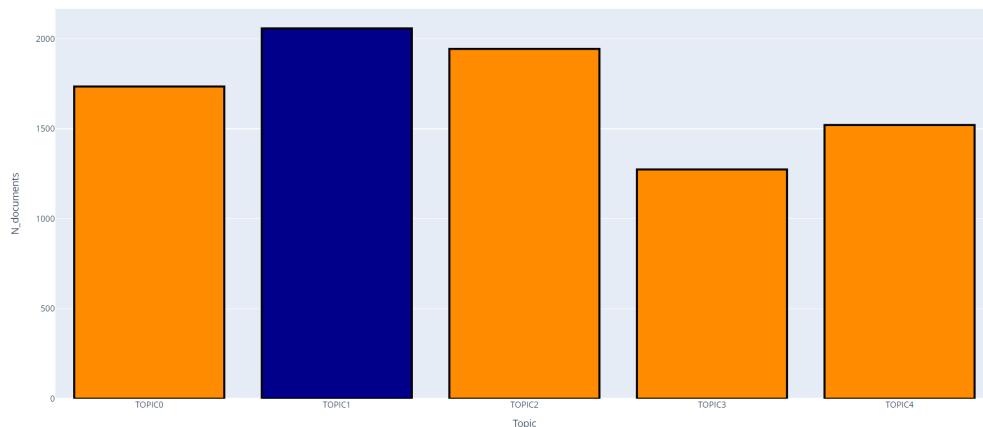
    fig.update_yaxes(title_text="N_documents") # left yaxes
    fig.show()
```

Il metodo stampa una tabella in cui ad ogni topic è associato il numero di documenti in cui esso è dominante:

topics	n_documents
0	1735
1	2058
2	1944
3	1274
4	1521

Lo stesso risultato viene poi rappresentato, grazie alla libreria grafica plotly, in un grafico a barre interattivo, in cui ad ogni barra corrisponde un topic, e l'altezza di ogni barra è data dal numero di documenti in cui domina:

Topic Distribution Gensim



Il terzo metodo chiamato è `show_topic_clusters()`:

```
def show_topic_clusters(self, lda_model, corpus, n_topics=4):
    """
    saves html report topic_clusters in a directory
    """
    topic_weights = []
    for i, row_list in enumerate(lda_model[corpus]):
        topic_weights.append([w for i, w in row_list[0]])

    # Array of topic weights
    arr = pd.DataFrame(topic_weights).fillna(0).values

    # Keep the well separated points (optional)
    arr = arr[np.amax(arr, axis=1) > 0.35]

    # Dominant topic number in each doc
    topic_num = np.argmax(arr, axis=1)

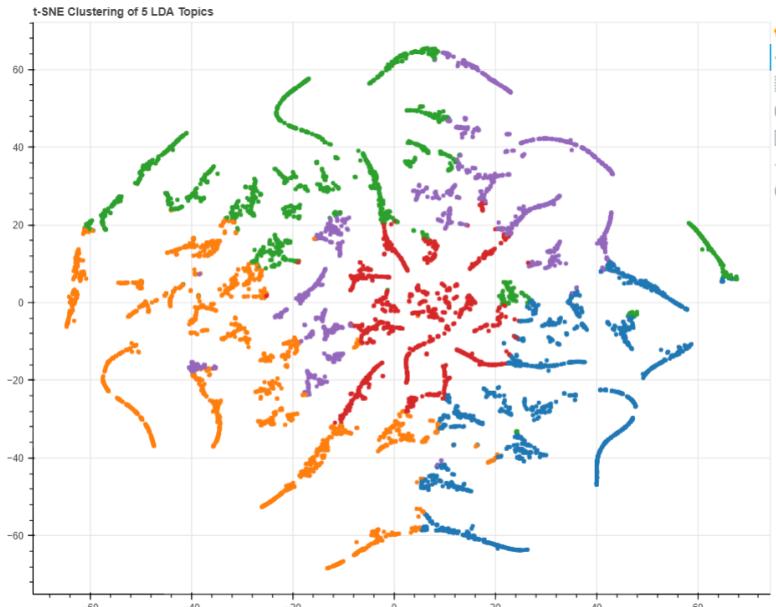
    # tSNE Dimension Reduction
    # t-distributed Stochastic Neighbor Embedding
    tsne_model = TSNE(n_components=2, verbose=1, random_state=0, angle=.99, init='pca')
    tsne_lda = tsne_model.fit_transform(arr)

    # Plot the Topic Clusters using Bokeh
    output_notebook()

    path = f'./report/report_gensim/{self.period}'
    if not os.path.exists(path): os.makedirs(path)
    file_name = f'{path}/topic_clusters_{self.newspaper}_{self.lang}.html'
    output_file(file_name)

    mycolors = np.array([color for name, color in mcolors.TABLEAU_COLORS.items()])
    plot = figure(title=f't-SNE Clustering of {n_topics} LDA Topics',
                  plot_width=900, plot_height=700)
    plot.scatter(x=tsne_lda[:, 0], y=tsne_lda[:, 1], color=mycolors[topic_num])
    save(plot)
```

Ci salva un report topic_clusters in formato html:



Nel report abbiamo una rappresentazione grafica dei clusters in due dimensioni.

Si sta utilizzando un algoritmo di riduzione della dimensionalità che si chiama t-SNE, un po' più complesso rispetto alla PCA, che ci fa vedere in uno spazio a 2 dimensioni la disposizione spaziale dei vari topics (la caratteristica principale di questo algoritmo è che preserva le proporzioni, cioè punti che nello spazio aumentato sono vicini tra loro, sono proiettati nello spazio ridotto vicini tra loro).

Nella rappresentazione grafica ogni punto rappresenta un documento e il colore di ogni punto rappresenta il topic di appartenenza.

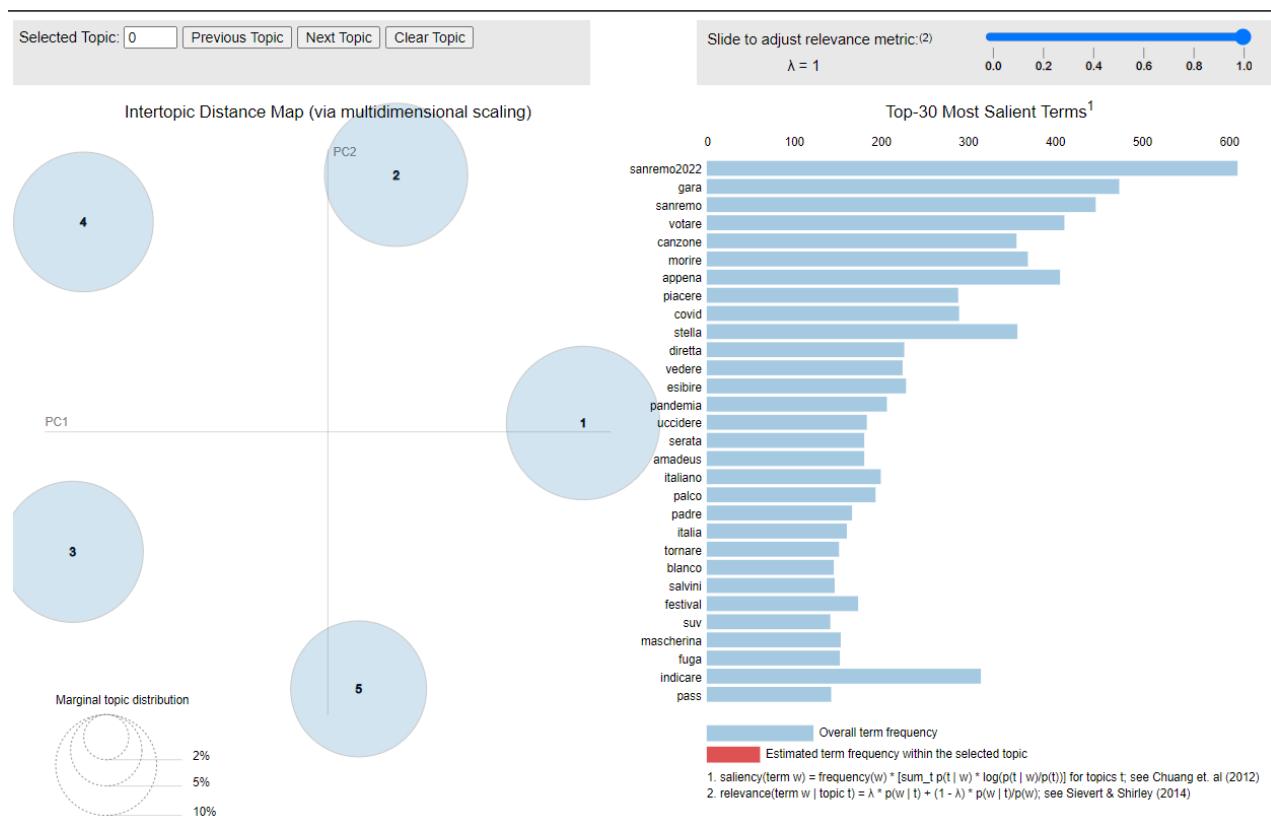
Più i punti dello stesso colore sono aggregati tra loro e separati da quelli dei colori differenti, più la clusterizzazione è ben riuscita.

Il quarto ed ultimo metodo chiamato è `visualize_topics()`:

```
def visualize_topics(self, lda_model, corpus):
    """
    saves the html report topic_visualization to a directory
    """
    vis = prepare(lda_model, corpus, dictionary=lda_model.id2word, mds='mmds')
    path = f'./report/report_gensim/{self.period}'
    if not os.path.exists(path): os.makedirs(path)
    pyLDAvis.save_html(vis, f'{path}/topic_visualization_{self.newspaper}_{self.lang}.html')
```

Viene restituito il report topic_visualization.html che è realizzato con una libreria che si chiama pyLDAvis. Gensim e pyLDAvis come permettono una visualizzazione molto efficace dei topic.

Quello che ci restituisce questo report è infatti una rappresentazione interattiva:



Sulla sinistra possiamo visualizzare una PCA in 2 dimensioni con i vari topic e, per ogni topic, se ci passiamo sopra, vediamo un cerchio che lo rappresenta. Tanto più quel cerchio è grande tanto più conterrà dei termini all'interno, e inoltre tanto più i topic (cerchi) sono distanziati tra loro, tanto più conterranno termini diversi; di conseguenza quando abbiamo due o più topic che si sovrappongono significa che c'è tra loro una sovrapposizione elevata di termini.

Sempre passando interattivamente sopra ciascun topic possiamo poi vedere le top 30 keywords, ordinate per importanza, che rappresentano il topic, e possiamo andare a vedere per ogni parola, passandoci sopra, qual è la sua term frequency globale e la frequenza all'interno dei vari topic.

Conclusioni

Dopo aver mostrato tutti gli output risultanti dalle due topic modeling, nell'esporre le conclusioni ci concentriamo sulle tabelle che mostrano per ogni topic le sue dieci top words. L'obiettivo è quello di individuare per ogni topic, sulla base delle sue top words, l'argomento a cui potrebbe riferirsi. Di seguito, per ogni periodo, riportiamo a titolo esemplificativo ma non esaustivo alcuni degli output da noi commentati

Precovid Scikit

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0	roma,597.65	uccidere,449.64	napoli,330.27	video,294.08	euro,284.45	potere,253.9	domani,252.89	chiedere,243.48	polizia,242.64	fermare,240.19
Topic 1	mori,573.23	m5s,521.34	tornare,434.5	donna,413.08	volere,407.11	renzi,353.85	italiano,338.19	voto,298.64	italia,296.75	bambino,287.19
Topic 2	salvini,1384.49	conte,993.09	pd,761.57	italia,684.68	maio,520.38	usa,491.31	lega,456.29	crisidigoverno,442.18	crisi,353.94	europa,247.41
Topic 3	migrante,417.08	milano,368.16	auto,349.24	arrivare,309.79	figlio,297.33	vincere,277.08	roma,276.26	social,249.95	europa,247.41	presidente,241.61

Topic 0 potrebbe riguardare: cronaca nera

Topic 1 potrebbe riguardare: ?

Topic 2 potrebbe riguardare: politica

Topic 3 potrebbe riguardare: ?

Repubblica

Ilgiornale

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0	salvini,59.71	lega,49.84	ue,46.51	conte,39.69	sardine,36.71	sinistra,31.5	renzi,31.37	sondaggio,28.92	carola,28.72	polizia,28.45
Topic 1	salvini,63.35	pd,59.71	foto,42.52	openarms,35.73	attacco,32.74	italiano,31.95	migrante,31.77	uccidere,28.8	attaccare,26.99	migranti,25.71
Topic 2	trump,54.09	italia,40.06	use,36.32	blog,29.3	choc,27.75	fisco,25.91	francia,22.5	potere,21.35	andrea,20.55	bretxit,19.46
Topic 3	pd,56.44	m5s,49.84	roma,40.06	zingaretti,37.44	macron,34.86	berlusconi,33.52	voto,33.45	tassa,32.07	lega,27.17	mori,26.62
Topic 4	mori,94.09	maio,84.28	salvini,77.19	migrante,76.95	ong,63.35	volere,57.31	renzi,47.6	pronto,40.06	tornare,34.06	porto,32.13

Topic 0 potrebbe riguardare: politica nazionale

Topic 1 potrebbe riguardare: immigrazione/politica

Topic 2 potrebbe riguardare: politica estera

Topic 3 potrebbe riguardare: politica nazionale

Topic 4 potrebbe riguardare : immigrazione

Nytimes

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0	hong,451.84	kong,443.59	people,411.88	police,252.22	price,232.28	woman,226.78	expect,223.87	fall,217.14	fire,197.72	familiar,195.61
Topic 1	company,604.42	government,322.44	york,314.46	big,307.58	world,306.69	talk,272.58	home,269.16	market,268.02	like,266.4	federal,262.92
Topic 2	trump,308.76	china,496.22	trade,440.06	president,339.46	business,316.25	hear,289.05	democratic,269.19	economy,263.77	administration,237.17	candidate,226.82
Topic 3	write,780.01	trump,710.05	president,632.42	house,425.56	official,283.96	country,259.51	american,257.58	political,246.49	report,244.7	global,221.85

Topic 0 potrebbe riguardare: proteste a Hong Kong del 2019/2020

Topic 1 potrebbe riguardare: ?

Topic 2 potrebbe riguardare: affari esteri

Topic 3 potrebbe riguardare: politica interna

Wsj

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0	hong,451.8	kong,443.55	take,323.16	government,322.46	world,273.82	police,252.17	time,238.43	try,237.52	expect,223.85	work,208.84
Topic 1	trump,1218.51	president,971.58	write,779.98	house,425.53	official,361.96	economy,263.74	american,257.56	end,242.14	administration,237.15	bank,219.27
Topic 2	democratic,269.15	federal,262.92	people,258.07	set,233.71	candidate,226.79	global,221.86	court,202.01	plan,196.65	familiar,195.57	presidential,194.2
Topic 3	want,327.73	find,326.29	company,262.56	report,244.7	million,241.83	billion,223.13	know,219.58	people,215.34	fire,197.69	sell,185.34
Topic 4	china,649.94	company,457.75	trade,440.02	market,439.38	investor,306.77	hear,289.03	stock,278.5	home,269.12	deal,263.19	go,245.13

Topic 0 potrebbe riguardare: proteste a Hong Kong 2019/2020

Topic 1 potrebbe riguardare: economia interna

Topic 2 potrebbe riguardare: giustizia?

Topic 3 potrebbe riguardare: economia?

Topic 4 potrebbe riguardare : affari esteri

covid Scikit

Repubblica

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0 italiano, 2729.59	milano, 2164.44	scuola, 2072.3	cina, 1542.1	napoli, 1516.02	mascherina, 1415.62	ue, 1333.57	chiudere, 1329.02	storia, 1210.37	uccidere, 1205.53	
Topic 1 usa, 2663.22	trump, 1943.42	italia, 1851.47	covid, 1438.74	rischio, 1376.84	emergenza, 1160.08	crisi, 1129.89	figlio, 1120.88	paura, 1100.19	salvare, 1091.45	
Topic 2 coronavirus, 14127.19	italia, 3468.14	covid, 2685.37	morts, 1748.3	positivo, 1594.82	europea, 1566.12	tornare, 1478.01	regione, 1310.47	cambiare, 1269.57	vaccino, 1255.17	
Topic 3 roma, 3285.65	conte, 2666.27	virus, 2063.0	salvini, 1597.2	pd, 1478.96	bambino, 1187.86	restare, 1114.83	fermare, 1063.5	strada, 1041.4	piano, 1019.2	
Topic 4 morire, 2483.52	donna, 1825.27	auto, 1422.33	presidente, 1090.49	quarantena, 966.1	famiglia, 925.2	diretta, 890.92	riaprire, 873.0	guerra, 837.8	voto, 835.78	

Topic 0 potrebbe riguardare: inizio diffusione pandemia

Topic 1 potrebbe riguardare: covid?

Topic 2 potrebbe riguardare: covid

Topic 3 potrebbe riguardare: politica?

Topic 4 potrebbe riguardare : ?

Ilgiornale

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0 conte, 1362.02	salvini, 1055.81	chiedere, 969.73	pd, 932.89	parlare, 885.45	italia, 862.2	premier, 772.25	m5s, 693.45	lega, 680.1	leader, 646.4	
Topic 1 italiano, 1243.16	raccontare, 798.29	covid, 767.48	dare, 759.32	rischiare, 577.35	centro, 566.99	prendere, 548.21	giovane, 495.61	morire, 488.51	roma, 471.29	
Topic 2 coronavirus, 1129.08	covid, 920.43	vaccino, 660.64	settimana, 623.4	storia, 614.79	italia, 529.44	studio, 517.55	prossimo, 489.86	numero, 466.36	renzi, 442.19	
Topic 3 migrante, 881.72	italia, 869.91	positivo, 851.45	milano, 538.62	vittima, 476.44	zona, 462.34	portare, 441.94	continuare, 431.08	tornare, 413.61	scuola, 410.01	
Topic 4 social, 1065.56	regione, 774.55	mettere, 726.1	euro, 723.85	polimica, 581.21	cambiare, 573.5	andare, 567.62	sindaco, 473.77	parola, 454.21	annunciare, 441.01	

Topic 0 potrebbe riguardare: politica nazionale

Topic 1 potrebbe riguardare: ?

Topic 2 potrebbe riguardare: covid

Topic 3 potrebbe riguardare: immigrazione

Topic 4 potrebbe riguardare : ?

Nytimes

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0 world, 1934.97	like, 1699.47	look, 1588.94	democratic, 1580.22	people, 1566.39	virus, 1524.61	outbreak, 1523.05	vote, 1477.26	lead, 1395.44	pandemic, 1328.06	
Topic 1 york, 3281.25	police, 2440.06	find, 2177.73	woman, 2121.29	people, 2045.27	man, 1745.87	public, 1739.62	kill, 1668.94	write, 1589.65	school, 1455.81	
Topic 2 president, 7241.61	trump, 6747.26	biden, 3235.71	write, 2478.35	live, 2116.36	election, 1853.71	house, 1830.58	black, 1824.29	test, 1697.72	change, 1631.34	
Topic 3 joe, 1899.7	win, 1386.54	nearly, 1255.0	trial, 1233.31	good, 1160.25	time, 1119.72	result, 1115.87	question, 1001.29	sunday, 997.39	de, 973.87	
Topic 4 coronavirus, 9827.06	country, 3109.97	case, 2952.48	people, 2567.19	pandemic, 2285.01	health, 2178.93	death, 1905.47	face, 1790.17	vaccine, 1777.86	covid, 1744.14	

Topic 0 potrebbe riguardare: ?

Topic 1 potrebbe riguardare: polizia americana

Topic 2 potrebbe riguardare: elezioni presidenziali 2020

Topic 3 potrebbe riguardare: ?

Topic 4 potrebbe riguardare : covid

Wsj

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0 coronavirus, 6867.84	pandemic, 2595.81	china, 2434.06	market, 2057.63	time, 1813.35	world, 1771.93	stock, 1658.05	investor, 1414.18	week, 1288.4	rise, 1245.71	
Topic 1 trump, 3863.67	company, 3581.25	big, 1607.8	billion, 1431.06	people, 1369.5	worker, 1198.86	plan, 1161.65	deal, 1156.21	president, 1155.66	start, 1101.63	
Topic 2 write, 2382.81	million, 2164.22	york, 1916.02	look, 1550.11	like, 1527.47	pandemic, 1494.85	work, 1259.88	joe, 1251.64	help, 1077.85	good, 1059.26	
Topic 3 covid, 3671.62	official, 1788.49	case, 1641.44	vaccine, 1436.17	people, 1304.18	test, 1244.13	include, 1182.59	country, 1113.49	government, 1027.81	spread, 981.74	
Topic 4 president, 2620.81	biden, 2344.02	house, 1439.73	way, 1304.73	administration, 1232.47	report, 1217.19	write, 1121.21	court, 1073.6	pay, 1034.88	lead, 975.18	

Topic 0 potrebbe riguardare: effetti della pandemia sull'economia

Topic 1 potrebbe riguardare: politica economica

Topic 2 potrebbe riguardare: ?

Topic 3 potrebbe riguardare: covid

Topic 4 potrebbe riguardare : politica interna

War Scikit

Repubblica

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0 sanremo,176.2	mariupol,161.3	figlio,128.96	eurovision,115.01	pronto,104.24	milano,96.32	maggio,80.59	ospedale,70.29	sfida,68.68	vincere,67.68	
Topic 1 ucraina,1711.54	guerra,851.61	russia,667.24	putin,604.45	russo,546.54	metropolislive,424.1	metropolis,+20.65	kiev,408.06	draghi,233.06	zelesky,222.65	
Topic 2 mosca,349.77	morire,207.25	usa,189.42	donna,142.91	russo,137.77	attacco,135.47	presidente,126.74	italiano,123.26	tornare,114.84	pagina,110.15	
Topic 3 commento,408.6	covid,372.83	italia,203.7	arma,183.22	gas,148.95	morto,144.92	sanzione,126.32	lega,123.04	m5s,122.25	conte,114.44	

Topic 0 potrebbe riguardare: musica/contest

Topic 1 potrebbe riguardare: guerra

Topic 2 potrebbe riguardare: guerra

Topic 3 potrebbe riguardare: sanzioni/guerra

Ilgiornale

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0 social,207.77	italiano,199.26	venire,190.92	vittima,140.59	figlio,137.63	raccontare,132.16	decidere,130.16	colpire,129.07	centro,126.55	cambiare,125.35	
Topic 1 russo,393.12	ucraino,390.77	leader,204.02	attacco,165.85	settimana,158.55	ucraina,142.45	finire,130.56	possibile,128.8	pubblico,127.34	italiano,126.99	
Topic 2 italia,287.39	storia,200.05	europa,169.64	euro,166.43	covid,164.23	roma,164.21	pandemia,112.89	prezzo,108.65	prendere,102.46	sistema,95.22	
Topic 3 ucraina,687.28	russia,497.29	guerra,462.07	putin,430.07	mosca,328.6	russo,323.38	parlare,216.58	donna,202.09	militare,180.26	crisi,155.34	
Topic 4 presidente,397.83	kiev,262.76	tornare,214.55	conflitto,201.52	andare,136.62	politico,127.87	pd,127.51	portare,126.19	parola,123.3	trovare,120.89	

Topic 0 potrebbe riguardare: ?

Topic 1 potrebbe riguardare: guerra

Topic 2 potrebbe riguardare: covid?

Topic 3 potrebbe riguardare: guerra

Topic 4 potrebbe riguardare : guerra

Nytimes

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0 biden,472.85	york,389.71	president,308.61	million,300.44	leave,297.25	trump,291.74	take,289.56	china,278.96	country,269.98	report,246.93	
Topic 1 win,334.53	people,327.18	country,266.3	covid,258.04	find,253.88	die,252.74	world,232.98	know,228.15	coronavirus,225.04	test,223.82	
Topic 2 man,323.3	company,298.44	beijing2022,297.95	police,273.04	people,250.82	begin,237.4	kill,203.98	olympics,197.28	olympic,189.55	set,188.89	
Topic 3 ukraine,1775.78	russia,1325.5	russian,1117.79	president,930.83	war,622.17	ukrainian,515.22	invasion,477.82	force,431.55	putin,391.59	official,346.55	
Topic 4 live,377.86	court,350.01	update,345.41	de,242.98	woman,201.3	supreme,200.77	follow,169.97	crisis,168.19	expert,155.24	figure,143.94	

Topic 0 potrebbe riguardare: elezioni presidenziali 2020?

Topic 1 potrebbe riguardare: covid?

Topic 2 potrebbe riguardare: olimpiadi invernali 2022

Topic 3 potrebbe riguardare: guerra

Topic 4 potrebbe riguardare : giustizia?

Wsj

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0 company,531.22	business,199.96	billion,172.08	share,147.78	elon,143.61	musk,143.58	plan,139.43	worker,138.55	explain,135.59	late,133.39	
Topic 1 million,581.54	people,501.33	house,178.94	way,178.89	find,153.13	hear,142.91	like,136.56	sell,134.0	home,130.96	win,119.08	
Topic 2 ukraine,1708.8	russia,874.32	russian,668.23	president,523.26	war,384.33	biden,309.67	country,345.21	invasion,267.84	putin,253.47	sanction,230.89	
Topic 3 price,399.02	world,276.4	stock,263.88	ukrainian,255.38	investor,232.95	pandemic,224.96	high,224.64	rise,205.95	inflation,202.86	write,182.95	
Topic 4 covid,318.39	federal,160.6	rate,156.63	cost,130.49	increase,130.21	team,127.59	write,127.27	face,124.41	judge,121.93	begin,119.36	

Topic 0 potrebbe riguardare: economia

Topic 1 potrebbe riguardare: ?

Topic 2 potrebbe riguardare: guerra

Topic 3 potrebbe riguardare: effetti guerra/covid sull'economia?

Topic 4 potrebbe riguardare : effetti covid sull'economia?

precovid gensim

Repubblica

- Topic 0 potrebbe riguardare: politica
- Topic 1 potrebbe riguardare: ?
- Topic 2 potrebbe riguardare: immigrazione
- Topic 3 potrebbe riguardare: immigrazione

```
[(),  
 '0.020*"salvin1" + 0.019*"ue" + 0.012*"conte" + 0.008*"luglio" + '  
 '0.008*"vertice" + 0.008*"nomina" + 0.008*"mercato" + 0.008*"europa" + '  
 '0.007*"rifiuto" + 0.007*"pd"'),  
(1,  
 '0.013*"donna" + 0.013*"italiano" + 0.009*"trump" + 0.009*"auto" + '  
 '0.009*"wimbledon" + 0.008*"vincere" + 0.008*"napoli" + 0.008*"addio" + '  
 '0.008*"lasciare" + 0.008*"putin"),  
(2,  
 '0.030*"italia" + 0.018*"carola" + 0.018*"watch" + 0.018*"sea" + '  
 '0.009*"rackete" + 0.007*"repty" + 0.007*"caldo" + 0.007*"autotrade" + '  
 '0.006*"estate" + 0.006*"europoe"),  
(3,  
 '0.024*"roma" + 0.022*"migrante" + 0.011*"usa" + 0.008*"libia" + '  
 '0.006*"lampedusa" + 0.006*"processo" + 0.006*"finale" + 0.006*"morire" + '  
 '0.006*"social" + 0.006*"domani")]
```

Ilgioriale

- Topic 0 potrebbe riguardare: ?
- Topic 1 potrebbe riguardare: immigrazione
- Topic 2 potrebbe riguardare: ?
- Topic 3 potrebbe riguardare: immigrazione
- Topic 4 potrebbe riguardare : immigrazione

```
[(),  
 '0.021*"trump" + 0.020*"editoriale" + 0.018*"usa" + 0.017*"m5s" + '  
 '0.014*"conte" + 0.013*"sindaco" + 0.010*"settimana" + 0.010*"soldo" + '  
 '0.009*"euro" + 0.008*"figlio"),  
(1,  
 '0.096*"salvini" + 0.044*"migrante" + 0.033*"carola" + 0.021*"italia" + '  
 '0.016*"rackete" + 0.015*"ira" + 0.014*"libia" + 0.011*"putin" + '  
 '0.011*"mare" + 0.009*"tornare"),  
(2,  
 '0.026*"dimaio" + 0.026*"nave" + 0.024*"lega" + 0.023*"ue" + '  
 '0.020*"parlamento" + 0.011*"attaccare" + 0.010*"andare" + 0.009*"germania" + '  
 '+ 0.009*"procedura" + 0.009*"infrazione"),  
(3,  
 '0.053*"ond" + 0.014*"mediterranea" + 0.014*"confine" + 0.014*"est" + '  
 '0.014*"porto" + 0.009*"tedesco" + 0.009*"pronto" + 0.009*"pro" + '  
 '0.009*"centro" + 0.009*"morto"),  
(4,  
 '0.033*"pd" + 0.025*"seawatch" + 0.021*"roma" + 0.018*"piano" + '  
 '0.013*"italiano" + 0.013*"toninelli" + 0.011*"migranti" + 0.011*"fico" + '  
 '0.011*"critico" + 0.011*"sicurezza")]
```

quotidiani statunitensi

- Topic 0 potrebbe riguardare: ?
- Topic 1 potrebbe riguardare: proteste HK 2019/20
- Topic 2 potrebbe riguardare: ?
- Topic 3 potrebbe riguardare: ?
- Topic 4 potrebbe riguardare : ?

```
[(),  
 '0.017*"time" + 0.017*"country" + 0.012*"man" + 0.011*"house" + '  
 '0.009*"federal" + 0.009*"face" + 0.009*"school" + 0.009*"help" + '  
 '0.009*"fire" + 0.009*"home"),  
(1,  
 '0.030*"president" + 0.030*"trump" + 0.019*"hong" + 0.018*"kong" + '  
 '0.014*"woman" + 0.013*"protester" + 0.010*"american" + 0.010*"protest" + '  
 '0.010*"take" + 0.009*"like"),  
(2,  
 '0.023*"people" + 0.018*"police" + 0.018*"week" + 0.018*"york" + '  
 '0.010*"official" + 0.012*"include" + 0.011*"question" + 0.011*"look" + '  
 '0.010*"raise" + 0.010*"today"),  
(3,  
 '0.018*"write" + 0.014*"million" + 0.010*"child" + 0.010*"leave" + '  
 '0.009*"united" + 0.009*"government" + 0.009*"know" + 0.009*"court" + '  
 '0.009*"team" + 0.009*"find"),  
(4,  
 '0.026*"world" + 0.016*"cup" + 0.015*"china" + 0.010*"final" + '  
 '0.010*"democratic" + 0.009*"way" + 0.008*"debate" + 0.008*"national" + '  
 '0.008*"system" + 0.008*"candidate")]
```

covid Gensim

quotidiani italiani

- Topic 0 potrebbe riguardare: covid
- Topic 1 potrebbe riguardare: politica
- Topic 2 potrebbe riguardare: ?
- Topic 3 potrebbe riguardare: sport?
- Topic 4 potrebbe riguardare: ?

```
[(),  
 '0.022*"italia" + 0.018*"italiano" + 0.018*"morire" + 0.014*"euro" + '  
 '0.012*"coronavirus" + 0.012*"morte" + 0.012*"roma" + 0.011*"uccidere" + '  
 '0.010*"aereo" + 0.010*"auto"'),  
(1,  
 '0.026*"salvini" + 0.014*"conte" + 0.013*"m5s" + 0.013*"libia" + '  
 '0.011*"voto" + 0.010*"maio" + 0.009*"lasciare" + 0.009*"parlare" + '  
 '0.008*"migliore" + 0.007*"errore"),  
(2,  
 '0.014*"iran" + 0.010*"storia" + 0.009*"figlio" + 0.008*"tornare" + '  
 '0.007*"meghan" + 0.007*"edicola" + 0.007*"tv" + 0.007*"harry" + '  
 '0.007*"film" + 0.007*"guerra"),  
(3,  
 '0.018*"trump" + 0.016*"donna" + 0.011*"inter" + 0.010*"napoli" + '  
 '0.010*"milan" + 0.010*"morte" + 0.008*"famiglia" + 0.008*"australia" + '  
 '0.008*"scuola" + 0.008*"vedere"),  
(4,  
 '0.025*"usa" + 0.022*"milano" + 0.018*"sanremo" + 0.017*"diretta" + '  
 '0.012*"cina" + 0.010*"andare" + 0.010*"vincere" + 0.009*"craxi" + '  
 '0.009*"attacco" + 0.008*"addio")]
```

quotidiani statunitensi

- Topic 0 potrebbe riguardare: covid
- Topic 1 potrebbe riguardare: cronaca nera
- Topic 2 potrebbe riguardare: ?
- Topic 3 potrebbe riguardare: ?
- Topic 4 potrebbe riguardare : politica interna

```
[(),  
 '0.019*"time" + 0.017*"week" + 0.016*"million" + 0.015*"coronavirus" + '  
 '0.015*"fire" + 0.013*"home" + 0.012*"leave" + 0.011*"death" + 0.010*"try" + '  
 '0.009*"face"),  
(1,  
 '0.027*"people" + 0.018*"kill" + 0.016*"york" + 0.016*"trial" + '  
 '0.012*"american" + 0.012*"general" + 0.010*"live" + 0.009*"strike" + '  
 '0.009*"help" + 0.008*"attack"),  
(2,  
 '0.030*"write" + 0.012*"find" + 0.011*"know" + 0.011*"report" + '  
 '0.010*"change" + 0.010*"like" + 0.010*"campaign" + 0.010*"look" + '  
 '0.009*"work" + 0.008*"good"),  
(3,  
 '0.019*"china" + 0.014*"man" + 0.014*"case" + 0.013*"win" + 0.011*"long" + '  
 '0.011*"take" + 0.011*"way" + 0.011*"australia" + 0.011*"call" + '  
 '0.010*"crash"),  
(4,  
 '0.035*"president" + 0.033*"trump" + 0.019*"iran" + 0.014*"impeachment" + '  
 '0.013*"break" + 0.011*"democratic" + 0.011*"country" + 0.010*"world" + '  
 '0.010*"woman" + 0.009*"military")]
```

guerra Gensim

Il giornale

```
[(),  
 '0.013*"presidente" + 0.012*"palco" + 0.011*"finire" + 0.011*"mascherina" + '  
 '0.010*"politico" + 0.009*"denaro" + 0.008*"truffa" + 0.008*"omicron" + '  
 '0.007*"cambiare" + 0.006*"vedere"),  
(1,  
 '0.018*"italiano" + 0.016*"italia" + 0.015*"parlare" + 0.013*"serata" + '  
 '0.010*"misura" + 0.008*"chiedere" + 0.008*"restare" + 0.007*"maio" + '  
 '0.007*"entrare" + 0.007*"partito"),  
(2,  
 '0.018*"sanremo" + 0.018*"festival" + 0.011*"leader" + 0.010*"trovare" + '  
 '0.007*"vittima" + 0.006*"pronto" + 0.006*"movimento" + 0.006*"numero" + '  
 '0.006*"puntare" + 0.006*"tema"),  
(3,  
 '0.011*"presenza" + 0.011*"ariston" + 0.010*"febbraio" + 0.010*"terzo" + '  
 '0.009*"settimana" + 0.008*"mattarella" + 0.008*"storia" + 0.008*"mettere" + '  
 '0.008*"quirinale" + 0.007*"controllo"),  
(4,  
 '0.019*"covid" + 0.018*"sanremo2022" + 0.016*"tornare" + 0.012*"social" + '  
 '0.010*"venire" + 0.010*"raccontare" + 0.009*"polemica" + 0.008*"media" + '  
 '0.008*"dare" + 0.008*"donna")]
```

quotidiani statunitensi

```
[(),  
 '0.015*"monday" + 0.015*"week" + 0.014*"woman" + 0.012*"government" + '  
 '0.010*"make" + 0.010*"leader" + 0.010*"plan" + 0.010*"american" + '  
 '0.010*"election" + 0.009*"court"),  
(1,  
 '0.020*"president" + 0.017*"write" + 0.017*"people" + 0.013*"china" + '  
 '0.013*"official" + 0.011*"biden" + 0.009*"police" + 0.009*"house" + '  
 '0.009*"beijing" + 0.008*"kill"),  
(2,  
 '0.017*"covid" + 0.016*"york" + 0.012*"test" + 0.012*"coronavirus" + '  
 '0.012*"pandemic" + 0.011*"world" + 0.010*"begin" + 0.010*"good" + '  
 '0.010*"company" + 0.010*"death"),  
(3,  
 '0.021*"ukraine" + 0.020*"russia" + 0.019*"country" + 0.015*"russian" + '  
 '0.012*"break" + 0.012*"leave" + 0.010*"high" + 0.010*"end" + '  
 '0.009*"military" + 0.009*"attack"),  
(4,  
 '0.019*"beijing2022" + 0.019*"olympics" + 0.017*"winter" + 0.014*"olympic" + '  
 '0.012*"time" + 0.011*"win" + 0.011*"trump" + 0.011*"medal" + 0.010*"team" + '  
 '0.010*"gold")]
```

Unendo quanto ottenuto e già commentato nelle analisi preliminari, con i risultati ottenuti dalla topic modeling, possiamo ipotizzare alcune conclusioni.

È evidente come lo scoppio della pandemia abbia monopolizzato le notizie.

Sia dalle analisi preliminari che della topic modeling si può notare come il tema della politica, dominante prima del diffondersi del virus, sia stato sovrastato.

Interessante il fatto che, nelle topic modeling effettuate relativamente al periodo della pandemia, anche quei topic identificabili in maniera abbastanza netta come non relativi alla pandemia, mostrino comunque, nelle 10topwords, seppur in minoranza, parole riconducibili al virus.

A questo proposito, soprattutto nei giornali statunitensi, più volte abbiamo individuato come topic quello riguardante gli effetti della pandemia sull'economia.

Questo potrebbe spiegarsi con il fatto che, durante la pandemia, gran parte delle notizie anche relative ad altri argomenti, venissero in qualche modo comunque ricondotte al virus.

Pensiamo ad esempio all'argomento sportivoed alle notizie sportive relative allo stop dei campionati, alla chiusura degli stadi, alle restrizioni nel pubblico.

Oppure al tema politico stesso, imprescindibile dall'evolversi della pandemia.

Sempre relativamente al periodo pandemico ci si deve soffermare sul caso particolare del Wall Street Journal, sembrato tra tutti il più "impermeabile" alle notizie relative al coronavirus, o comunque quello in cui le notizie sulla pandemia hanno monopolizzato di meno la situazione.

Ancor più interessante quello che accade nei primi mesi del 2022, sia nelle analisi preliminari che nella topic modeling, sembrerebbe che le notizie sulla guerra siano ancora più polarizzanti rispetto a quanto successo per quelle relative alla pandemia.

L'argomento covid infatti, molto forte nel corso degli ultimi due anni, è stato in gran parte rimpiazzato proprio dalle notizie relative alla guerra.

In tal senso occorre dire che la stessa pandemia, potrebbe aver perso quota tra le notizie dei giornali non solo per lo scoppio della guerra in Ucraina, ma per un affievolirsi negli ultimi mesi delle preoccupazioni relative alla pandemia stessa.

Infine menzioniamo, tra gli altri topic che più si sono mostrati (anche se in maniera molto minore rispetto a quelli sopracitati) nell'analisi:

- la cronaca nera;
- la polizia, in particolare nei giornali degli Stati Uniti, dove il dibattito sul comportamento dei poliziotti è molto in voga;
- l'immigrazione, in particolare per quanto riguarda Il Giornale, rispecchiando il suo orientamento politico verso destra;
- le elezioni presidenziali, nei giornali statunitensi;
- la giustizia, nei giornali statunitensi;
- le rivolte di Hong Kong del 2020, nei giornali statunitensi