

# **Analisi dell' orientamento politico, del sentimento e dell' emotions delle principali testate giornalistiche italiane**

Progetto di esame a cura di Alessia Lusci e  
Nicholas Garau

Web Analytics e Analisi Testuale - A.A. 2020/2021

Docente: Prof. Marco Ortù  
Università degli Studi di Cagliari



# 1. Introduzione

Il progetto descritto dal report che segue, presenta diversi obiettivi che ci siamo posti di raggiungere, cercando di sfruttare a pieno le metodologie e le tecniche introdotte dal docente durante il corso di studi, partendo dal *social-mining* e il *web-scraping* passando per il *Natural Language Processing* e il *machine learning*.

In particolare abbiamo deciso di incentrare gli sforzi di *social-mining* tramite Twitter, andando a estrarre i dati testuali di diverse testate giornalistiche e di diversi partiti politici, rigorosamente in italiano, andando a cercare di vincere una sfida data dal fatto che gli strumenti di *Natural Language Processing* per la nostra lingua sono quantitativamente meno, rendendo il corso del progetto più stimolante e suggestivo.

## 1.1. Obiettivi del progetto.

L' obiettivo finale è quello di costruire un modello capace di capire tramite l' analisi testuale dei tweets e degli articoli presenti all' interno delle testate giornalistiche o di qualsiasi media che contenga dati testuali al proprio interno, l' orientamento politico dell' autore, il *sentiment* a esso collegato e contestualmente anche l' *emotions*.

Utilizzeremo i tweet dei partiti politici appartenenti a diverse aree dello spettro politico al fine di ottenere un corpus di classificazione che ci permetta di avere dei tokens di riferimento per ciascun orientamento, in particolare destra, sinistra e neutro. Analizzeremo contestualmente anche il *sentiment* e l' *emotion*, presentando i dati attraverso l' utilizzo di uno *spiderplot*. Successivamente utilizzeremo questi corpus per costruire una *frequency distribution* per avere una sorta di sunto sui termini più presenti nel panorama politico del web, costruiremo un classificatore che ci possa aiutare a effettuare delle previsioni sull' orientamento politico dei media e,

analizzeremo anche per questi ultimi il *sentiment* e l' *emotions* di ognuno di essi.

### 1.1. Articolazione del progetto.

Facendo versioning del codice con git sulla piattaforma BitBucket, il progetto è organizzato in 3 branch:

- il branch 'Scraping' che comprende tutti i file addetti al mining dei dati sul web, con la cartella data che andrà a contenere tutti i dataset in formato .txt o .csv estratti attraverso il codice.
- Il branch 'data\_cleaning' che andrà a contenere tutti i file di scripting addetti alla pulizia del codice, al filtraggio, alla tokenizzazione del testo e al codice al fine di ottenere solamente dei dataset che saranno pronti all'utilizzo, alla modellizzazione e alle analisi finali.
- Il branch 'Analysis' che conterrà tutti gli script e i file che produrranno i modelli e i test di questi ultimi, da cui si trarranno le conclusioni del progetto.

### 1.2 Strumenti utilizzati

L' intero progetto è stato svolto sull' IDE 'Pycharm', con il supporto di diverse librerie che richiameremo man mano nello svolgimento del report.

## 2. Svolgimento e descrizione

### **2.1 Scraping**

#### 2.1.1 Estrazione dei tweets dei partiti politici

Il primo task del branch 'Scraping' è l' estrazione dei tweet dei partiti politici italiani. A tal fine a rappresentare lo spettro completo della politica italiana, si è scelto di optare per l' estrazione dei tweet di 9 differenti users:

- per l' ala sinistra abbiamo scelto: il Partito Democratico, Rifondazione Comunista e Liberi e Uguali
- per i neutrali, o ala moderata si è optato per: il Movimento 5 Stelle, Forza Italia e PiùEuropa
- Per l'ala destra: Lega, Fratelli d'Italia e Casapound.

Per la stesura del codice abbiamo optato per una soluzione object oriented, come possiamo notare nel file parties\_tweets\_scraper.py , con una classe *ScraperPartiesTweets* a supporto, che al suo interno dopo l’inizializzazione della classe stessa, che comprende gli slot start, until, user e max\_tweet a parametrizzare quelli che son i dati di input che ci chiede la query della libreria *snsscraper*

```

def __init__(self, start, until, user, max_tweet):
    self.start = start
    self.until = until
    self.user = user
    self.max_tweet = max_tweet

def scraping_tweet_to_df(self):
    tweets_list = []
    try:
        for i, tweet in enumerate(
            sntwitter.TwitterSearchScraper(
                f'from:{self.user} since:{self.start} until:{self.until}').get_items()):
            if i > self.max_tweet:
                break
            else:
                tweets_list.append([tweet.date, tweet.content, tweet.username])
    return pd.DataFrame(tweets_list, columns=['DateTime', 'Text', 'Username'])
    except Exception 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)
    return pd.DataFrame(tweets_list, columns=['DateTime', 'Text', 'Username'])

```

*Figura1: la classe ScraperPartiesTweets*

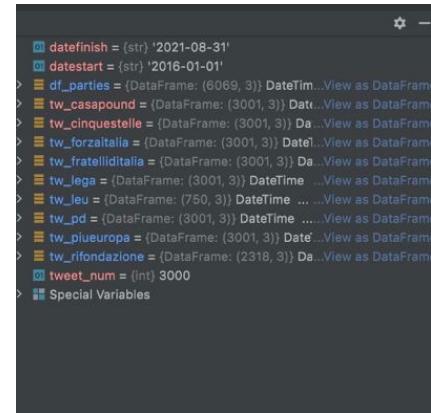
Come possiamo notare dal codice in figura 1, abbiamo optato per un pattern comprendente un metodo chiamato *scraping\_tweet\_to\_df* che non fa altro che estrarre i dati e restituire un dataframe di *pandas*. Abbiamo scelto di fissare un tetto massimo di 3000 tweet per user, unica eccezione per 2 profili, Rifondazione Comunista e LeU che non raggiungono nell’ intervallo di tempo fissato (1 Gennaio 2016 - 31 Agosto 2021) i 3000 tweets, causata dall’ inattività dei profili stessi (figura 2):

Al momento dell’ estrazione utilizziamo una lista di supporto per ogni request che lo scraper effettua, effettuiamo l’ appenò all’ interno di un ciclo for e al momento in cui la condizione dell’ if è soddisfatta, il break termina la

funzione e ritorna un dataframe di *pandas*, metodologia scelta per gestire meglio eventuali analisi che dovessero esser necessarie in fasi intermedie dello sviluppo, utilizzando per esempio metodi come il *.tail()* o il *.head()* che visualizzano velocemente la coda o le prime righe del dataframe. Il corpo del metodo sta all' interno di un blocco *try except* che nel caso in cui dovesse entrare nell' *except* ci fornirebbe informazioni circa l' errore che affronta il codice.

*Figura 2: dataframes Pandas dopo lo scraping*

Per ogni oggetto che si va a creare nel *\_\_main\_\_* chiamando la classe *ScaperPartiesTweets*, dopo che assegno a *datestart* e *datefinish* e a *max\_tweet* i parametri che voglio passare all' oggetto creato, ci sarà un assegnazione a una variabile che sarà di tipo dataframe, come vediamo in figura 2. Infine per poter creare un dataset unico, utiliziamo il metodo *.concat()* di *pandas* per creare un unico dataframe che viene convertito in csv tramite il metodo *.to\_csv()* di *pandas*, e salvato nella cartella 'data' all' interno del branch *Scraping* (figura 4).



```
▶ if __name__ == '__main__':
    datestart = '2016-01-01'
    datefinish = '2021-08-31'
    tweet_num = 3000

    tw_pd = ScaperPartiesTweets(datestart, datefinish, tweet_num).scraping_pd()
    print('pd done...pause')
    time.sleep(1800)
```

*Figura 3: prima parte del main in parties\_tweets\_scaper.py*

È importante notare che per ogni user, abbiamo fissato un *time.sleep* che nell' esempio sovrastante (figura 3) è di mezz'ora (1800 secondi), ma in

fase di estrazione è stato spesso fissato a 900 secondi per poter non superare i limiti delle requests ed evitare eventuali negazioni di accesso.

```
df_parties = pd.concat([tw_pd, tw_leu, tw_rifondazione, tw_forzaitalia,  
                       tw_piueuropa, tw_cinquestelle, tw_lega,  
                       tw_fratelliditalia, tw_casapound], keys=['DateTime', 'Text', 'Username'])  
  
df_parties.to_csv('./data/tweets_partiti.csv', index=False)
```

Figura 4: il concatenamento dei dataframes e il salvataggio dei dati estratti.

### 2.1.2 Estrazione dei tweets delle testate giornalistiche

In secondo luogo, il secondo script del branch *Scraping*, si occupa di estrarre sempre utilizzando *snsscraper*, i tweet relativi di qualsivoglia testata giornalistica, che porta con se i relativi link che in secondo luogo verranno filtrati e utilizzati, laddove possibile, per estrarre le sole notizie di carattere politico ed economico, isolando le stesse urls da una parte e i tweet della notizia dall’ altra così da avere in un dataset unico sia i tweet che le relative notizie e i relativi link in features separate.

Il nostro approccio, in questa fase è stato quello di utilizzare Twitter, come archivio di appoggio così da non utilizzare dei *crawler* da “lanciare” nei siti web delle testate giornalistiche, ma di avere un link diretto frutto dell’ estrazione dei tweet stessi. Questo ci potrebbe permettere addirittura di paragonare il sentiment di un tweet da quello del relativo articolo, così da valutare, per esempio, come viene effettuata la scrittura di un determinato tweet in confronto all’ articolo stesso così da risultare più accattivante.

Il file in cui si sviluppa l’ estrazione è *scraping\_tweets\_newspaper.py*, in cui troviamo la classe *ScraperNewsTweets* (figura 5) che rappresenta lo scraper stesso, e prende come attributi: la data di inizio, la data di fine e l’ username del profilo da ‘scrapare’.

È stato creato un metodo opportuno per i giornali che utilizzano *URL-shortening services*, dato che successivamente avrebbero potuto creare

```

class ScraperNewsTweets(object):
    """
        A class to represent a tweets scraper.

        Attributes
        -----
        start : str
            Scraping start date
        until : str
            Scraping end date
        user : str
            Twitter username

        Methods
        -----
        dataframe_tweets():
            A method to scrape tweets with the attributes established and creating a pd.DataFrame
            with DateTime, Id, Text, Link, Username of the single tweet
    """

    def __init__(self, start, until, user):
        """
        Args:
        start (str): the established start for scraping
        until (str): the established end for scraping
        user (str): twitter username of newspaper

        """
        self.start = start
        self.until = until
        self.user = user

```

Figura 5: la classe ScraperNewsTweets

problematiche per poter recuperare il testo della notizia, si è optato per ottenere l'url adeguato.

Si è preso come attributo i link dei tweet e, tramite la libreria *requests* che mi consente di richiedere informazioni verso il Web Server, facciamo una richiesta head e ottenere l'un-short url.

A differenza di GET non viene restituito come risposta il corpo contenente il messaggio ma le sole intestazioni del messaggio.

Successivamente si è creato un metodo per poter fare lo scraper dei tweets, un metodo per i giornali che non utilizza dei short url e uno che consente di utilizzare il metodo creato in precedenza per correggere gli url associati (figura 6):

```

def longurl(self, links):
    """
    Unshortening link
    :param link: tweet.outlinks
    :return: unshorted link or empty list if link is invalid
    """
    long_url = []
    try:
        for link in links:
            print(link)
            url = requests.head(link, allow_redirects=True, timeout=30).url
            long_url.append(url)
    return long_url

    except Exception as e:
        print(e)
        return long_url

def scraping_news(self):
    """
    Scraping newspapers' tweets and saving the date, the text, the link associated with the news and username of newspaper
    :return: dataframe of the information saved
    """
    tweets_list = []
    try:
        for i, tweet in enumerate(sntwitter.TwitterSearchScrapper(
                f'from:{self.user} since:{self.start} until:{self.until}').get_items()):
            tweets_list.append([_, tweet.date, tweet.content, tweet.outlinks, self.user_])
    return pd.DataFrame(tweets_list, columns=['DateTime', 'Text', 'Link', 'Username'])

    except Exception 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)
        return pd.DataFrame(tweets_list, columns=['DateTime', 'Text', 'Link', 'Username'])

def scraping_news_unshort(self):
    """
    Scraping newspapers' tweets and saving the date, the text,
    the link associated with the news and username of newspaper
    :return: dataframe of the information saved
    """
    tweets_list = []
    try:
        for i, tweet in enumerate(sntwitter.TwitterSearchScrapper(
                f'from:{self.user} since:{self.start} until:{self.until}').get_items()):
            if tweet.outlinks:
                long_url = self.longurl(tweet.outlinks)
                tweets_list.append([_, tweet.date, tweet.content, long_url, self.user_])
    return pd.DataFrame(tweets_list, columns=['DateTime', 'Text', 'Link', 'Username'])

    except Exception 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)
        return pd.DataFrame(tweets_list, columns=['DateTime', 'Text', 'Link', 'Username'])

```

*Figura 6: i metodi della classe ScraperNewsTweets*

Infine nel main (figura 7) possiamo notare che per ogni particolare quotidiano si è utilizzato il metodo adeguato in base alla presenza o meno degli short-url, e, dopo aver estratto i tweet, con anche qui un time sleep di 600 secondi, che possiamo far variare a seconda del funzionamento o meno degli scraper al momento del lancio del codice, non si è fatto altro che concatenare i dataframes con pandas.concat(), ottenuti dalla chiamata dei metodi e assegnati a una variabile per ciascun quotidiano. In ultima istanza andremo a salvare, ogni volta che estrarremo dei dati con lo scraper, i dataframes concatenati in dei csv contenuti nella cartella data.

```

if __name__ == '__main__':
    datestart = '2021-01-01'
    datefinish = '2021-03-31'
    tw_Repubblica = ScraperNewsTweets(datestart, datefinish, 'repubblica').scraping_news()
    time.sleep(600)
    tw_FattoQuotidiano = ScraperNewsTweets(datestart, datefinish, 'fattoquotidiano').scraping_news_unshort()
    time.sleep(600)
    tw_Ilgiornale = ScraperNewsTweets(datestart, datefinish, 'ilgiornale').scraping_news()
    time.sleep(600)
    tw_UnioneSarda = ScraperNewsTweets(datestart, datefinish, 'UnioneSarda').scraping_news_unshort()
    time.sleep(600)
    tw_Ilmessaggero = ScraperNewsTweets(datestart, datefinish, 'ilmessaggeroit').scraping_news()
    time.sleep(600)
    tw.Libero = ScraperNewsTweets(datestart, datefinish, 'Libero_official').scraping_news()
    time.sleep(600)
    tw_IlPost = ScraperNewsTweets(datestart, datefinish, 'ilpost').scraping_news()
    time.sleep(600)
    tw_Corriere = ScraperNewsTweets(datestart, datefinish, 'Corriere').scraping_news_unshort()
    time.sleep(600)
    tw_LaStampa = ScraperNewsTweets(datestart, datefinish, 'LaStampa').scraping_news_unshort()
    df = pd.concat([tw_IlPost, tw.Libero, tw_Ilgiornale, tw_Ilmessaggero,
                    tw_FattoQuotidiano, tw_UnioneSarda, tw_Corriere, tw_LaStampa])

    df.to_csv('./data/tweets_giornali_2021_first_semester.csv', index=False)

```

Figura 7: il main di scraping\_tweets\_newspaper.py

## 2.2 Data cleaning e filtraggio

### 2.2.1 Creazione corpus per orientamento politico

Nel secondo branch del progetto troviamo le operazioni di filtraggio e di data cleaning. Per prima cosa nel file *parties\_clean.py* abbiamo creato, una classe che lasciasse nella feature *datetime* del csv con i tweets dei partiti politici estratti in precedenza, solamente la data in formato ‘aaaa-mm-gg’ e che rimuovesse gli urls dal testo del tweet. Come possiamo notare in figura 8, la classe è stata chiamata *DataFrameCleaner* e prende come attributo solamente il dataframe che viene importato. I metodi che effettuano le operazioni descritte in precedenza sono *datetime\_cleaner()* e *remove\_urls()*.

La seconda classe del file è *CreatingCorpus* (figura9) che al suo interno racchiude i metodi per poter estrarre da ciascuna riga che rispetti la condizione *.isin()* di *pandas* in relazione alla lista che viene assegnata alla variabile *usernames*, una lista con tutti i tweets per orientamento politico (quest’ ultimo viene aggiunto come feature nel main che vediamo in figura 10, utilizzando la sintassi di *numpy* che vede l’ utilizzo del *where()* e del *select()*.

```

class DataFrameCleaner:
    """
        A class to clean the dataframe from urls and a part of the Datetime
        Attributes
        -----
        dataframe : pandas.DataFrame
    """

    def __init__(self, dataframe):
        self.df = dataframe

    def datetime_cleaner(self):
        """
        ...
        dates = []
        for item in self.df.iloc[:, 0]:
            dates.append(item[:9])
        self.df.iloc[:, 0] = pd.Series(dates)

    def remove_urls(self):
        """
        ...
        tweet_no_url = [re.sub(r"http\S+", '', item) for item in self.df.iloc[:, 1]]
        self.df.iloc[:, 1] = tweet_no_url

```

Figura 8: la classe DataFrameCleaner

A questo punto implementiamo una funzione esterna alla classe, *remove\_stopwords()*, che non fa altro che prendere un testo (in questo caso nel main andrà a prendere ciò che abbiamo avuto in output dai metodi della classe *CreatingCorpus()* e attraverso l’ utilizzo della libreria *NLTK* e dei suoi preziosi metodi, andrà con un doppio for a tokenizzare e filtrare dalle stopwords i tweets passati come parametro. Per eliminare maggiormente il rumore abbiamo fatto che si che vi siano solo lettere minuscole con *.lower()* e abbiamo eliminato la punteggiatura attraverso la condizione che ciò che andrà nella lista di appoggio che verrà ritornata dalla funzione, non sia in *string.punctuation*, parte del modulo *string*, e sia strettamente solo alfabetico con *.isalpha()*. Questa funzione verrà utilizzata e riprodotta di frequente lungo lo svolgimento del progetto.

Nel main, non facciamo altro che importare il csv estratto nel branch precedente, per poi, come accennato in precedenza, crearne uno che porti con se una feature aggiuntiva con l’ orientamento politico da utilizzare in fase di analisi, e poi creare un istanza di classe per ogni corpus che vogliamo

creare, destra, sinistra e neutro. Tramite il modulo json salviamo i corpus prodotti dai metodi di *CreatingCorpus*, filtrati per la funzione *remove\_stopwords()*, in dei file di testo che verranno successivamente salvati nella cartella data.

```
class CreatingCorpus:  
    """ ... """  
  
    def __init__(self, dataframe):  
        self.df = dataframe  
        pass  
  
    def corpus_neutro(self):  
        """ ... """  
        usernames = ['Piu_Europa', 'forza_italia', 'Mov5Stelle']  
        df_filtered = self.df[self.df.Username.isin(usernames)]  
        return list(pd.Series(df_filtered['Text']))  
  
    def corpus_right(self):  
        """ ... """  
        usernames = ['LegaSalvini', 'FratelliItalia', 'CasaPoundItalia']  
        df_filtered = self.df[self.df.Username.isin(usernames)]  
        return list(pd.Series(df_filtered['Text']))  
  
    def corpus_left(self):  
        """ ... """  
        usernames = ['pdnetwork', 'liberi_uqualsi', 'direzioneprc']  
        df_filtered = self.df[self.df.Username.isin(usernames)]  
        return list(pd.Series(df_filtered['Text']))
```

Figura 9: la classe *CreatingCorpus* e i suoi metodi

Nel main, non facciamo altro che importare il csv estratto nel branch precedente, per poi, come accennato in precedenza, crearne uno che porti con se una feature aggiuntiva con l'orientamento politico da utilizzare in fase di analisi, e poi creare un istanza di classe per ogni corpus che vogliamo creare, destra, sinistra e neutro. Tramite il modulo json salviamo i corpus prodotti dai metodi di *CreatingCorpus*, filtrati per la funzione *remove\_stopwords()*, in dei file di testo che verranno successivamente salvati nella cartella data.

```

if __name__ == '__main__':
    df1 = pd.read_csv('./data/tweets_partiti.csv', sep=',')
    df_test = DataFrameCleaner(df1)
    df_test.datetime_cleaner()
    df_test.remove_urls()
    conditions = [
        ((df1['Username'] == 'Piu_Europa') | (df1['Username'] == 'forza_italia') | (df1['Username'] == 'Mov5Stelle')),
        ((df1['Username'] == 'pdnetwork') | (df1['Username'] == 'liberi_eguali') | (df1['Username'] == 'direzioneprc')),
        ((df1['Username'] == 'LegaSalvini') | (df1['Username'] == 'FratellidItalia') | (
            df1['Username'] == 'CasaPoundItalia'))
    ]
    values = ['neutro', 'sinistra', 'destra']
    df1['Orientamento'] = np.select(conditions, values)

    df1.to_csv('./data/tweets_partiti_orient.csv', index=False) # salvo il dataframe con l' orientamento

    corpus_to_filter = CreatingCorpus(df1)
    left_to_filter = corpus_to_filter.corpus_left()
    right_to_filter = corpus_to_filter.corpus_right()
    neutro_to_filter = corpus_to_filter.corpus_neutro()

    tweets_left_tokens = remove_stopwords(left_to_filter)
    tweets_right_tokens = remove_stopwords(right_to_filter)
    tweets_neutro_tokens = remove_stopwords(neutro_to_filter)

    with open('./data/corpus_sinistra.txt', 'w') as f:
        f.write(json.dumps(tweets_left_tokens))

    with open('./data/corpus_destra.txt', 'w') as f:
        f.write(json.dumps(tweets_right_tokens))

    with open('./data/corpus_neutro.txt', 'w') as f:
        f.write(json.dumps(tweets_neutro_tokens))

```

Figura 10: il main del file parties\_clean.py

## 2.2.2 Filtraggio delle news

In fase di estrazione, osserviamo che come ci sarebbe da aspettarsi, non tutte le notizie potrebbero essere utili al fine di classificare l' orientamento politico della notizia stessa, perciò abbiamo optato per un opzione che ci consentisse di filtrare solo le notizie categorizzate in politica ed economia.

Per fare ciò, si è creata, come osserviamo nel file *filtering\_news.py*, la classe *CategorizedNews*, che ha come attributo semplicemente un link, che in fase di istanziamento non faremo altro che prenderlo dalla feature che abbiamo isolato precedentemente in fase di scraping delle news e dell'accumulo dei dati nei csv divisi a periodi. Come osserviamo in figura 11, la classe *CategorizedNews* porta con se 3 diversi metodi, uno stratagemma che ci permette di trattare la varietà dei diversi link che utilizzano le diverse

```

class CategorizedNews(object):
    """
    ...
    """

    def __init__(self, links):
        self.links = links

    def filtering_tweets(self):
        """
        ...
        political_economic_link = []
        try:
            for link in self.links:
                parsed_url = urlparse(link)
                if ('politica' in parsed_url.path.lower() or ('economia' in parsed_url.path.lower()) or ('interni' in parsed_url.path.lower())):
                    political_economic_link.append(link)

            return political_economic_link

        except Exception as e:
            print(e)
            return political_economic_link

    def filtering_tweets_query(self):
        """
        ...
        political_economic_link = []
        try:
            for link in self.links:
                parsed_url = urlparse(link)
                parsed_query = parse_qs(parsed_url.query)
                for value in parsed_query.values():
                    if...('politica' in value[0]) or ('economia' in value[0].lower()) or ('palazzi&potere' in value[0].lower()):
                        political_economic_link.append(link)

            return political_economic_link

        except Exception as e:
            print(e)
            return political_economic_link

    def filtering_tweets_parsed(self):
        """
        ...
        political_economic_link = []
        try:
            for link in self.links:
                if link != '[']:
                    page = requests.get(link.replace("'", "")).replace("[", "").replace("]", "").text
                    soup = BeautifulSoup(page, "html.parser")
                    articles = soup.find_all("article")
                    for item in articles:
                        if item.select_one("li") != None:
                            category = item.select_one("li").text
                            if (category.strip() == 'Politica') or (category.strip() == "Economia"):
                                political_economic_link.append(link)

            return political_economic_link

        except Exception as e:
            print(e)
            return political_economic_link

```

testate giornalistiche, a seconda di ciò che incontriamo nella colonna ‘Link’ che andiamo a importare nel main.

*Figura 11: la classe CategorizedNews e i suoi metodi*

Il primo metodo è *filtering\_tweets()*, che utilizziamo laddove la categoria giornalistica è presente nel path. Facciamo questo dopo che effettuiamo un

parsing utilizzando la libreria *urllib*, inserendo tutto in un blocco try except. Questo metodo utilizza il modulo *urllib.parse* per suddividere l' URL nelle sue 6 componenti: "scheme://netloc/path;parameters?query#fragment". Abbiamo utilizzato la funzione *urlparse* per filtrare la notizia della componente 'path'.

Il second metodo (*filtering\_tweets\_query()*) è quello che utilizziamo nel momento in cui la categoria da noi ricercata si trova all' interno della query stessa, per accedere alla componente 'query string' successivo al parsing (come accade nel Fattoquotidiano). È stata sempre utilizzata la funzione *urlparse()* nel modulo precedentemente citato, per in aggiunta a essa si è optato per l' utilizzo di 'parse\_qs' che fa il parsing della striga query e restituisce un dizionario il cui primo valore indica la categoria della notizia.

Il terzo metodo invece (*filtering\_tweets\_parsed()*) è stato creato per il giornale 'il Post' che non presenta la categoria della notizia nell'url quindi si è deciso di creare un metodo apposito. In questo caso è stata utilizzando la libreria *requests* per inviare una richiesta HTTP ottenendo un oggetto di classe *Response* e con "text" siamo in grado di leggere il contenuto della risposta del server.

Successivamente è stata utilizzata la libreria *BeautifulSoup* per poter fare il parsing del documento html di ogni pagina presa in considerazione e andando a filtrare nel tag "article" il contenuto presente nel tag "li", tenendo conto del fatto che non sempre è presente la categoria (nel caso delle vignette di Peanuts il contenuto del tag sarà "None") e si filtrerà per notizie di politica ed economia.

```
class ExtractArticle(object):
    """
    A class to extract newspapers' articles and his tags
    """

    def __init__(self, links):
        """
        Initializing class
        :param links: a dataframe's column
        """
        self.links = links
```

Figura 12: la classe Extract Article

Dopo aver filtrato le notizie per categoria si procede all'estrazione del testo dell'articolo creando una classe apposita: *ExtractArticle* (figura 12). All'interno della classe è stato creato un metodo *extract\_text()*(figura 14) per estrarre il testo utilizzando la libreria *Newspapers3k*. In prima battuta si sono filtrati gli url in modo da ottenere solo gli url delle notizie dei giornali presi in considerazione così da poter escludere notizie da pagine o giornali esterni a quelli presi in input. Considerato il caso in cui la connessione alla pagina vada in timeout dato che si utilizza la libreria request si è importato l'oggetto *Config* (figura13) che supporta un parametro di timeout che previene futuri problemi di “read timed out”.

```
user_agent = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:78.0) Gecko/20100101 Firefox/78.0'
config = Config()
config.browser_user_agent = user_agent
config.request_timeout = 30
```

Figura 13: istanziamento oggetto Config()

Per poter gestire le problematiche legate all'estrazione del testo è stato impostato un *try except* importando *ArticleException*.

```
def extract_text(self):
    """
        A method to extract news's text
    :return: text
    """
    text = []
    for link in self.links:
        link = link.replace(" ", "").replace("[", "").replace("]", "")
        if ('repubblica' in urlparse(link).netloc) or ('ilpost' in urlparse(link).netloc) or (
                'liberquotidiano' in urlparse(link).netloc) or ('ilgiornale' in urlparse(link).netloc) or (
                'ilmessaggero' in urlparse(link).netloc) or ('corriere' in urlparse(link).netloc) or (
                'lastampa' in urlparse(link).netloc) or ('unionesarda' in urlparse(link).netloc) or (
                'ilfattoquotidiano' in urlparse(link).netloc):
            article = Article(link, config=config)
            try:
                article.download()
                article.parse()
            except ArticleException:
                pass
            text.append(article.text)
        else:
            text.append(None)
```

*Figura 14: il metodo extract\_text()*

Successivamente è stata utilizzata una funzione già precedentemente sfruttata nel corso del progetto in modo da poter eliminare punteggiatura e le stopwords e restituire per ogni tweet una lista di tokens.

Si è ritenuto opportuno tenere conto esclusivamente della data del tweet e si è pulito il testo del tweet da eventuali links.

Di conseguenza importiamo il csv con i tweets e i link relativi agli articoli di riferimento, e creiamo un’istanza di classe chiamando *CategorizedNews* utilizzando il metodo *isin()* di *pandas* filtrando le osservazioni del dataframe in una sola riga. Per ogni circostanza, creiamo una variabile con il metodo relativo che abbiamo creato all’ interno della classe *CategorizedNews*, e successivamente andremo a creare un dataframe di *pandas* con le osservazioni filtrate con il metodo *concat()*, e salvate successivamente in un csv che collichiamo nella solita cartella ‘data’.

## **2.3 Analisi e modellizzazione**

### 2.3.1 Sentiment analysis ed emotions analysis

Per calcolare il sentiment dei tweets e del testo degli articoli, sono stati utilizzati i moduli *textblob* ed *nltk.sentiment*.

Per quanto riguarda il riconoscimento delle emotions è stato utilizzato *text2emotion* (package, a collection of modules) che è in grado di elaborare ogni tipo di testo e restituisce 5 categorie di emozioni: Happy, Angry, Sad, Surprise e Fear.

Come output otteniamo un dizionario le cui chiavi sono le emozioni riconosciute dal package e il relativo valore associato che va da 0.0 a 1.0.

Per poter calcolare il sentiment dei tweets è stato utilizzata la libreria *textblob* che ha il metodo *sentiment* che restituisce “polarity” un valore float nel range [-1.0 to 1.0] dove 0 indica neutralità, +1 indica un sentiment positivo e -1 rappresenta un sentiment negativo.

Per quanto riguarda il sentiment dei testi si è optata per una suddivisione del testo in frasi utilizzando `nltk.tokenize.sent_tokenize` ed effettuando il calcolo del sentiment ad ogni frase utilizzando VADER (Valence Aware Dictionary and Sentiment Reasoner) built-in, sentiment analyzer pre-addestrato che restituisce un dizionario con i valori del sentiment negativo, neutrale e positivo.

Creo delle liste all' interno della funzione `evaluating_sentiment` (figura 15) in cui andrò a salvare il sentiment individuato da ciascuna frase a seconda del risultato e successivamente calcolo la percentuale di frasi con ciascun sentimento rispetto al numero di frasi totali e restituirò il sentimento predominante per ciascun testo a seconda della percentuale più elevata.

La percentuale viene calcolata utilizzata una funzione apposita (figura16).

```
def evaluating_sentiment(list_sentences):
    """
    ...
    positive = 0
    negative = 0
    neutral = 0
    neutral_list = []
    negative_list = []
    positive_list = []
    for element in list_sentences:
        analyzer = SentimentIntensityAnalyzer().polarity_scores(element)
        neg = analyzer['neg']
        neu = analyzer['neu']
        pos = analyzer['pos']

        if neg > pos:
            negative_list.append(element)
            negative += 1
        elif pos > neg:
            positive_list.append(element)
            positive += 1
        elif pos == neg:
            neutral_list.append(element)
            neutral += 1

    positive = percentage(positive, len(list_sentences))
    negative = percentage(negative, len(list_sentences))
    neutral = percentage(neutral, len(list_sentences))

    if (positive > neutral) and (positive > negative):
        return 'positive'
    elif (negative > neutral) and (negative > positive):
        return 'negative'
    else:
        return 'neutral'
```

Figura 15: la funzione `evaluating_sentiment`

```

def percentage(part, whole):
    """
    :param part: a part of a quantity
    :param whole: the total amount of a quantity
    :return: percentage of a quantity as a rate per 100
    """
    return 100 * float(part) / float(whole)

```

Figura 16: la funzione percentage()

La funzione `evaluating_sentiment` viene richiamata all’ interno `get_article_sentiment()` all’ interno della classe `SentimentAnalysisEmotionDetection` (figura17) che abbiamo appositamente creato per attribuire a ciascun articolo di giornale un sentimento.

```

class SentimentAnalysisEmotionDetection(object):
    """
    Calculating sentiment and detecting emotion of an article and tweet
    """

    def __init__(self, dataframe):
        """
        Initializing class
        :param dataframe: a pandas's dataframe
        """
        self.dataframe = dataframe

    def get_article_sentiment(self):
        """
        Calculating articles' sentiment dividing text in sentences
        :return: articles'sentiment
        """
        articles = self.dataframe[['Article']]
        sentences_list = [nltk.tokenize.sent_tokenize(article, language="italian") for article in articles if article]
        sentiment_text = []
        for sent in sentences_list:
            sentiment_text.append(evaluating_sentiment(sent))
        return sentiment_text

```

Figura 17: la classe SentimentAnalysisEmotionsDetection()

Per quanto riguarda il riconoscimento delle emotions sono stati presi in considerazione i seguenti casi:

- per ogni emozione si ottiene un valore pari a 0.0 quindi si è deciso di etichettare il suddetto risultato con “neutral emotion”;
- si ottiene un’emozione con valore pari a 1.0, essendo il massimo si restituisce la emozione che presenta questo risultato;
- si ottengono più di due emozioni predominanti (individuate con un’apposita funzione (figura 18) in grado di restituire la lista di emozioni

che presentano il medesimo valore, ossia il massimo dei valori del dizionario) allora si è deciso di restituire “no particular emotion”

- caso in cui sono predominanti due emozioni e si restituisce la lista con le suddette emozioni.

```
def get_key(diz):
    """
    Finding keys' name of a list of values
    :param diz: it's a dictionary
    :return: the corresponding keys' name of the max values
    """
    keys = []
    for key, value in diz.items():
        if value == max(diz.values()):
            keys.append(key)
    return keys
```

Figura 18: la funzione `get_key()`

Per poter visualizzare graficamente i dati raccolti precedentemente sono state utilizzati:

- per i diagrammi a barre: la libreria `seaborn`
- Per gli spider plot si sono utilizzate due librerie

### 2.3.2 Data visualization per sentiment ed emotions

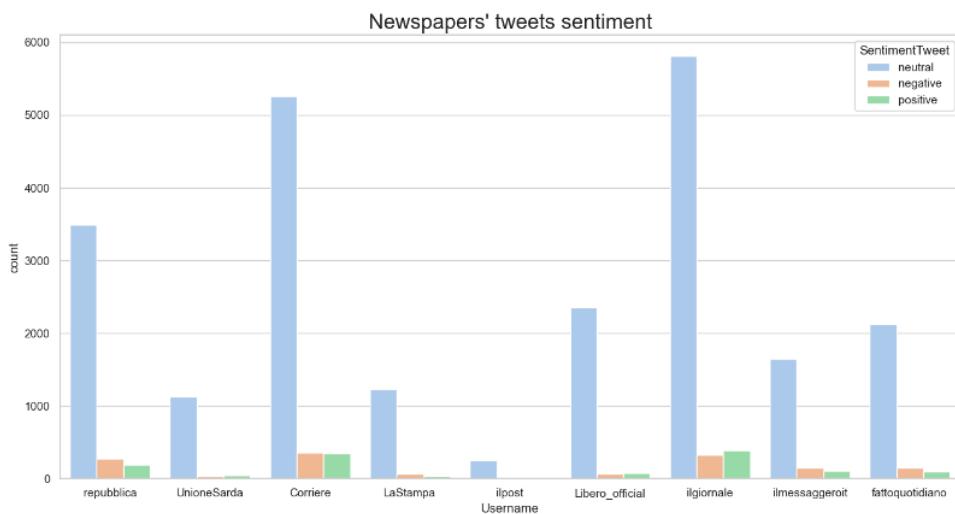
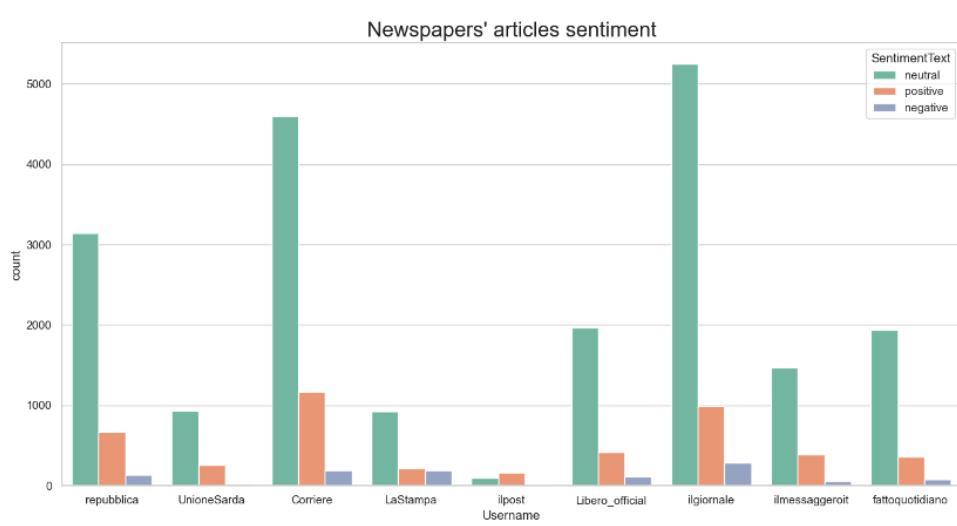


Figura 19: barplot per il sentiment dei tweets delle testate giornalistiche

Quello che possiamo osservare in figura 19 è che tendenzialmente i tweets dei giornali presentano un sentimento in gran parte neutrale, un pattern che si osserva anche in figura 20 nel barplot relativo al sentimento per quanto riguarda gli articoli. In questi ultimi si presenta un leggero trend in aumento per quanto concerne il sentimento positivo.



*Figura 20: barplot per il sentimento degli articoli delle testate giornalistiche*

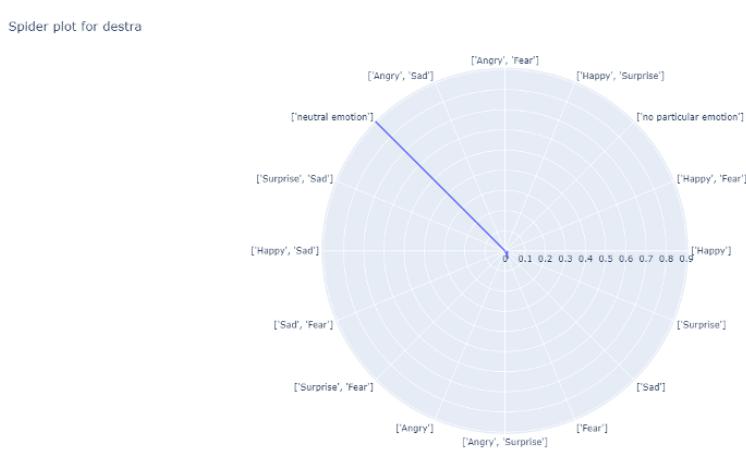
Per quanto riguarda l' analisi delle emotions effettuata utilizzando gli spider plot, seguiamo generalmente lo stesso trend, dato che, come possiamo osservare nelle figure 21, 22, 23 e 24, in prevalenza abbiamo un emotion neutrale. Per l' esattezza le percentuali sono: sinistra con 88%, orientamento neutrale con 85% e destra con 91%.



*Figura 21:spider plot per le emotions dell' orientamento di sinistra*

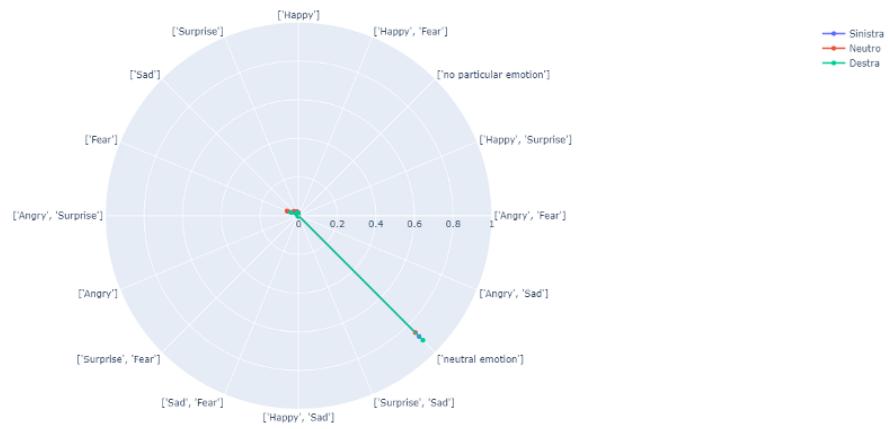


*Figura 22:spider plot per le emotions dell' orientamento di neutro*



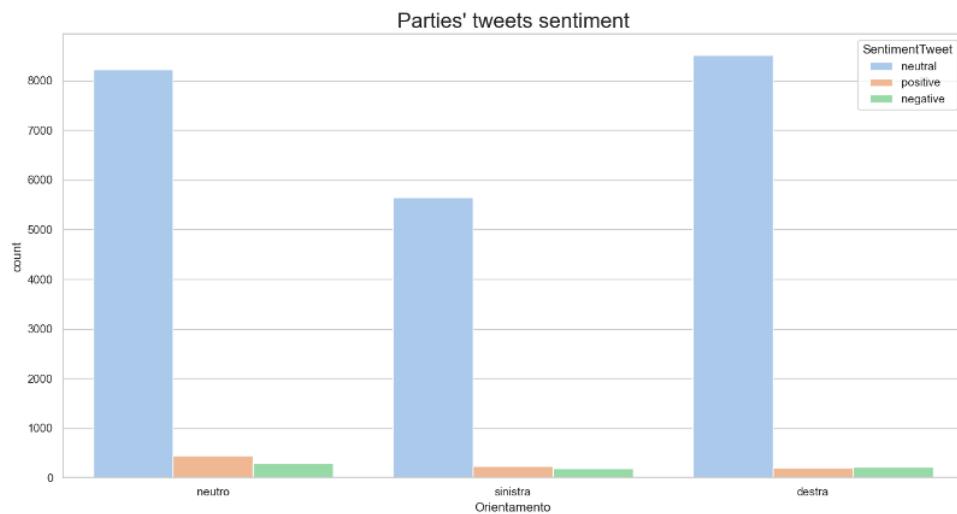
*Figura 23:spider plot per le emotions dell' orientamento di destra*

In ultima istanza si è provato a sovrapporre i risultati di tutti e 3 gli orientamenti politici in un unico spider plot, ottenendo però risultati che dal punto di vista grafico ci confermano il fatto che tutti e 3 sono neutrali in questa fase di analisi, infatti i raggi si sovrappongono e non ci danno un'informazione grafica accattivante (figura 24)



*Figura 24: spider plot per le emotions di tutti gli orientamenti politici*

L' ultimo grafico realizzato per questa parte riguarda i sentiment dei tweets dei partiti, raggruppati per orientamento politico. Anche qui in prevalenza osserviamo una predominanza dei sentiment neutrali (figura 25)



*Figura 25: barplot dei sentiment dei tweets dei partiti spartiti per orientamento*

### 2.3.3 Validazione del modello per sentiment ed emotions

È stato preso in considerazione un campione di 100 tweets relativi all'anno 2020 di 3 partiti, uno per ciascuna fazione:

- PD per la sinistra
- Movimento 5 stelle per il neutro
- Lega per la destra

Per poter validare l'attribuzione del sentiment effettuata dalla libreria *textblob* si sono letti i testi senza tenere conto del sentiment attribuito da quest'ultima ottenendo come risultato un 0,25 di errore. L'errore rilevato si presenta prevalentemente nei tweets che classificheremmo come negativi ma rilevati come positivi o neutri. Arriviamo alla conclusione che la libreria non avendo conoscenza dell'uso di determinati termini nella lingua italiana (vedi "pagano TUTTI"), classificato come positivo ma in realtà era in accezione negativa, può presentare un margine di errore non elevato dato che è difficile individuare il sentiment in queste situazioni.

Per la parte della validazione abbiamo usato i file già esistenti nel branch, per quanto riguarda lo scraping, il filtraggio e l' analisi del sentiment. Possiamo osservarlo perchè all' interno del repository i dataset frutto del seguente lavoro sono contrassegnati dalla presenza di 'analyzed' all' interno del nome.

### 2.3.4 Random forest

Per quanto riguarda le predizioni sull' orientamento politico degli articoli del news abbiamo optato su una *Random Forest* addestrata su 6000 tweet per orientamento politico, per un totale di 18000 osservazioni. Dopo aver effettuato una pulizia del testo attraverso l' utilizzo di *remove\_stopwords()*, si è modificato il dataframe in ingresso per poter dare in pasto al classificatore il testo tokenizzato pronto per essere vettorizzato con un *TfidfVectorizer*. All' istanzamento di quest' ultimo all' interno della funzione *random\_forest()* (figura 26) si è optato per fissare a FALSE il parametro *lowercase*, dato che la

funzione `remove_stopword()` effettua questo per noi. Questo è stato uno dei problemi da aggirare a cause delle osservazioni in lingua italiana, perchè attualmente *Scikit-learn* supporta solo la stringa ‘english’ nella tokenizzazione e rimozione delle stopword tramite i parametri della classe stessa.

```
def random_forest():
    """
    training and testing of the random forest model on party tweets with the dataset of 6000 observations per orientation
    """

    X = df_to_train['Text']
    y = df_to_train['Orientamento']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
    vectorizer = TfidfVectorizer(
        strip_accents='unicode',
        decode_error='ignore',
        lowercase=False,
        norm='l2'
    )
    pipeline = Pipeline([
        ('vect', vectorizer),
        ('clf', RandomForestClassifier(
            n_estimators=500,
            max_features=None,
            n_jobs=-1,
            random_state=910,
            oob_score=True,
        )))
    ])

    # fitting our model and save it in a pickle for later use
    model = pipeline.fit(X_train, y_train)
    with open('RandomForest.pickle', 'wb') as f:
        pickle.dump(model, f)
    ytest = np.array(y_test)
    print('matrice di confusione:\n{}\n'.format(confusion_matrix(ytest, model.predict(X_test))))
    print('report :\n{}\n'.format(classification_report(ytest, model.predict(X_test))))
```

Figura 26:implementazione della random forest

Dopo la vettorizzazione delle osservazioni testuali si è passato alla creazione della *pipeline*, che prende in ingresso la variabile `vectorizer` creata in precedenza e il `RandomForestClassifier` con 500 alberi. Abbiamo inoltre fissato il test sulle osservazioni *out of bag* attraverso il parametro `oob_score`.

Possiamo inoltre osservare che al momento dell’assegnazione della variabile target e delle features, si è spartito il dataset in training e test set in proporzione 80%/20% con la funzione `train_test_split()`.

Dopo aver settato il modello e addestrato il modello, passiamo al salvataggio dello stesso in un file `.pickle`, per poterlo utilizzare in secondo luogo per le predizioni sugli articoli delle testate giornalistiche.

report :	precision	recall	f1-score	support
destra	0.83	0.85	0.84	1191
neutro	0.78	0.78	0.78	1213
sinistra	0.78	0.76	0.77	1196
accuracy			0.80	3600
macro avg	0.80	0.80	0.80	3600
weighted avg	0.80	0.80	0.80	3600

Figura 27:report sul test set del modello random forest

Come possiamo osservare in figura 27, il modello ha un accuracy dell' 80%, una precision dell' 83% per le osservazioni di destra, e qualche punto percentuale in meno per le osservazioni neutro e di sinistra con un 78%, valori presi sulle osservazioni parte del test set.

```
def random_forest_on_medias():
    """
    testing of model saved on .pickle file on newspaper articles
    """
    loaded_model = pickle.load(open('RandomForest.pickle', 'rb'))

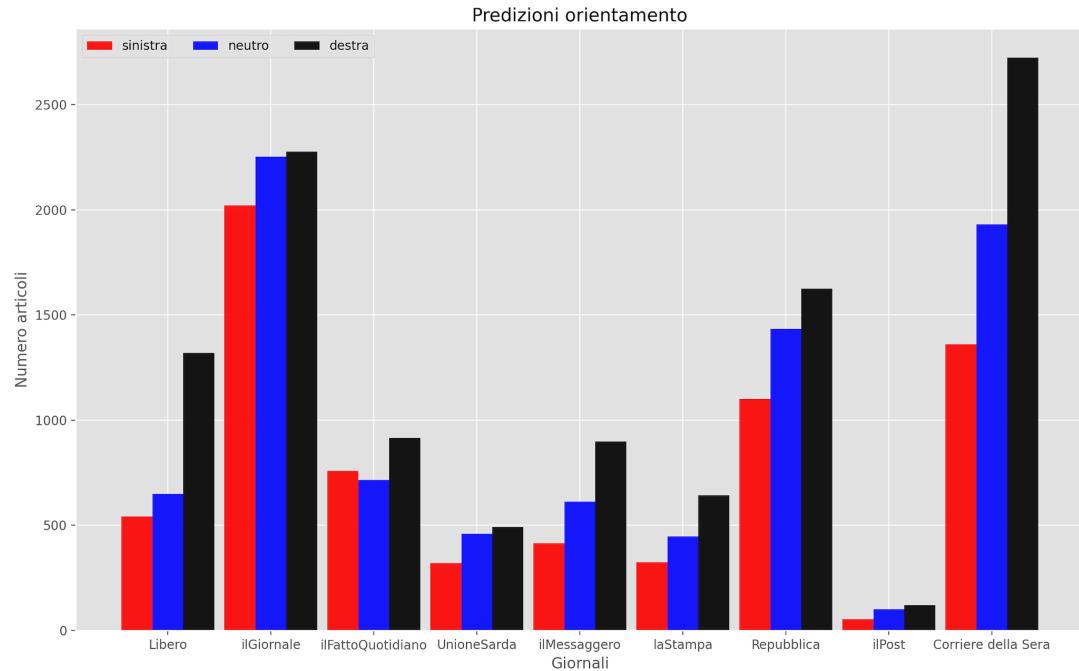
    articles_libero = [remove_stopwords(str(tweet)).replace('[', '').replace(']', '') for tweet in
                       df_medias['Text'].loc[df_medias['Username'].isin(['Libero_official'])]]
    res_libero = loaded_model.predict(articles_libero)
```

Figura 28:predizioni sulle testate giornalistiche

Come accennato in introduzione al paragrafo, dopo aver salvato il modello in un file `.pickle`, per ogni testata giornalistica abbiamo ricavato le predizioni, dopo averle pulite con `remove_stopwords()` e inserite in una lista con una list comprehension, e salvandole in una variabile per ogni testata giornalistica.

Dopo aver ricavato le predizioni, i risultati che abbiamo ottenuto denotano, secondo il nostro modello, una predominanza di articoli ritenuti dallo stesso lievemente un po' più destrorsi piuttosto che neutrali o di sinistra, eccezion fatta per Libero, che notoriamente è considerato un giornale che strizza l' occhio maggiormente ai partiti di destra, e il modello

ce lo conferma come possiamo osservare in figura 29, con circa il doppio delle osservazioni classificate come destrorse rispetto alle neutrali.



*Figura 25: barplot dei sentiment dei tweets dei partiti spartiti per orientamento*

Il Fatto Quotidiano possiamo osservare che, secondo il modello, è l'unica che presenta le osservazioni classificate di sinistra in maggioranza rispetto alle neutrali, ma comunque vede la predominanza delle osservazioni di destra.

L'unica testata che, come Libero, vede un gap ampio di circa 700 osservazioni, seppur non doppio come quest'ultimo, è il Corriere della Sera che classifica oltre 2600 articoli come destrorsi.

### 2.3.5 Frequency distribution per orientamento politico

L'ultima analisi del progetto prevede la creazione di 3 oggetti *Frequency Distribution* per cercare di dare un summary dei principali temi affrontati per ciascuna area politica. Per fare ciò, abbiamo utilizzato i corpus di tokens precedentemente creati nel branch *data\_cleaning*, e, dopo un ulteriore pulizia con la funzione *maketrans()* del modulo *string* per eliminare della

punteggiatura extra, creiamo i 3 oggetti grazie al supporto della classe *FreqDist()* di *NLTk* (figura 26).

Dopo di che passiamo alla realizzazione dei cumulative plot con i 50 termini più utilizzati per ciascuna area politica.

```
import string
import nltk

translator = str.maketrans(' ', ' ', string.punctuation)

with open('./data/corpus_sinistra.txt', 'r') as r:
    corpus_sinistra = r.read().replace('[', '').replace(']', '')

corpus_sinistra = [word.translate(translator) for word in corpus_sinistra.split(',')]

print(corpus_sinistra[:20])

with open('./data/corpus_neutro.txt', 'r') as r:
    corpus_neutro = r.read().replace('[', '').replace(']', '')

corpus_neutro = [word.translate(translator) for word in corpus_neutro.split(',')]

print(corpus_neutro[:20])

with open('./data/corpus_destra.txt', 'r') as r:
    corpus_destra = r.read().replace('[', '').replace(']', '')

corpus_destra = [word.translate(translator) for word in corpus_destra.split(',')]

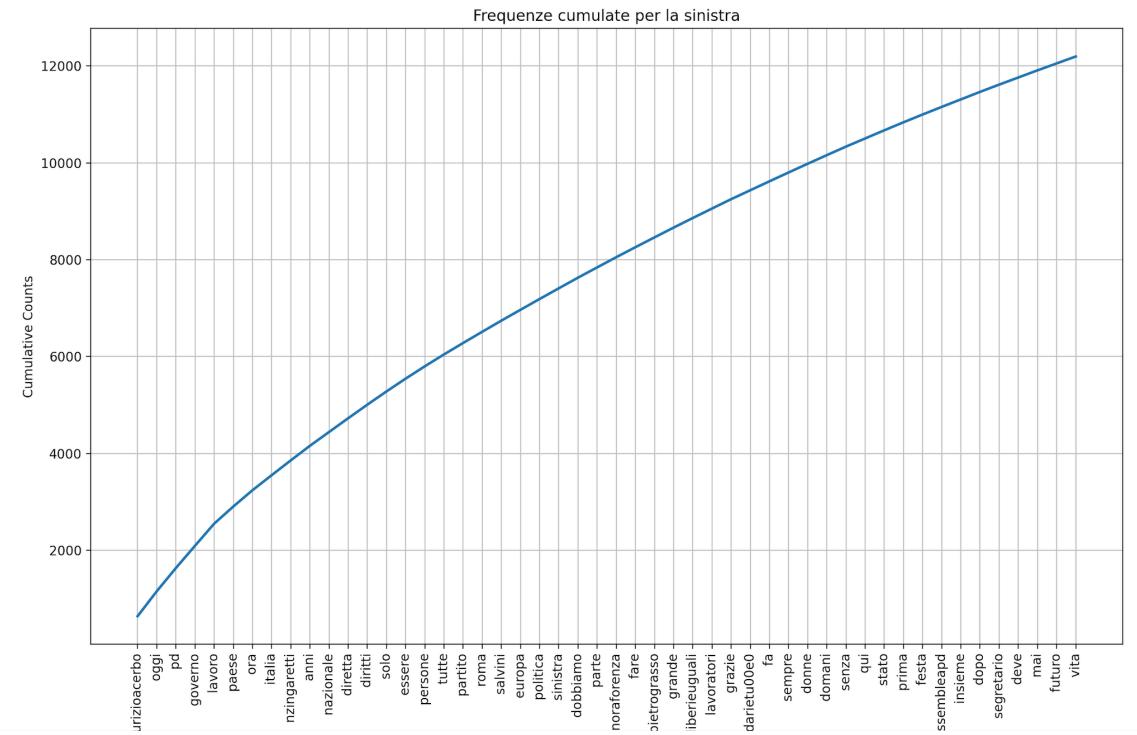
print(corpus_destra[:20])

words_sinistra = nltk.FreqDist(corpus_sinistra)
words_neutro = nltk.FreqDist(corpus_neutro)
words_destra = nltk.FreqDist(corpus_destra)

# words_sinistra.plot(50, cumulative=True, title='Frequenze cumulate per la sinistra')
# words_neutro.plot(50, cumulative=True, title='Frequenze cumulate per i tweets neutrali')
# words_destra.plot(50, cumulative=True, title='Frequenze cumulate per la destra')
```

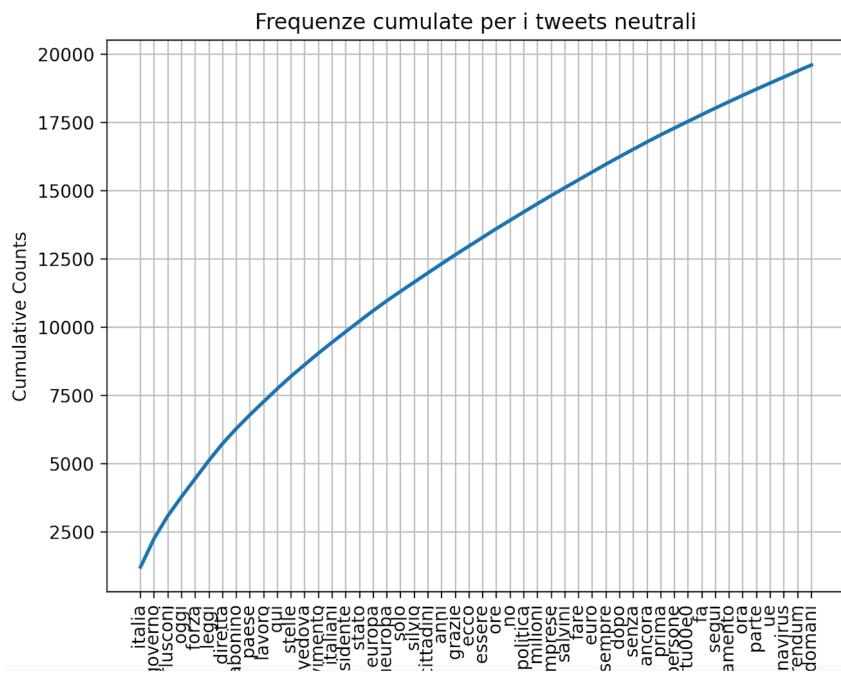
Figura 26: il file freq\_distr\_on\_corpus.py

Per quanto riguarda la sinistra (figura 27) possiamo notare informazioni rilevanti sui tokens più utilizzati sulla parte destra del grafico, ‘vita’ e ‘futuro’ sono i tokens più frequenti con circa 12000 utilizzi, ma tra gli 8000 e i 10000 vediamo che termini quali ‘donne’, ‘stato’, ‘domani’ e ‘futuro’ ci danno una generica accezione delle modalità comunicative con cui quest’area politica affronta la comunicazione web, lasciando intravedere quello che potrebbe



*Figura 27: cumulative plot per la sinistra.*

essere un attacco contro gli avversari politici solamente poco sopra i 6000 termini con il token ‘salvini’.



*Figura 28: cumulative plot per i partiti neutrali.*

Per quanto riguarda i partiti neutrali (figura 28) osserviamo la presenza del token ‘domani’ come più frequente, ma osserviamo anche come ‘coronavirus’, ‘ue’ e ‘euro’ si posizionino nella parte più a destra del grafico, che vede token più moderati al suo interno, con ‘salvini’ però presente nuovamente attorno ai 15000 utilizzi.

Per quanto riguarda l’ area destra (figura 29) notiamo come c’è un forte richiamo ai simboli o nomi che si identificano nei partiti stessi, ‘fdi’ è infatti il secondo token più utilizzato con una frequenza di 24000, ma vediamo anche il token ‘giorgiameloni’ sopra i 20000. È molto rilevante anche l’ aspetto che vede questi partiti più frequentemente attaccare gli avversari, infatti il termine ‘sinistra’ sta oltre la metà destra del grafico con una frequenza di circa 21000. Sono presenti in questa top 50 termini come ‘piazza’, ‘italiani’, forti richiami all’ identità nazionalista, tipici segni distintivi della destra sovranista che copre gran parte del panorama politico di quest area di questi tempi.

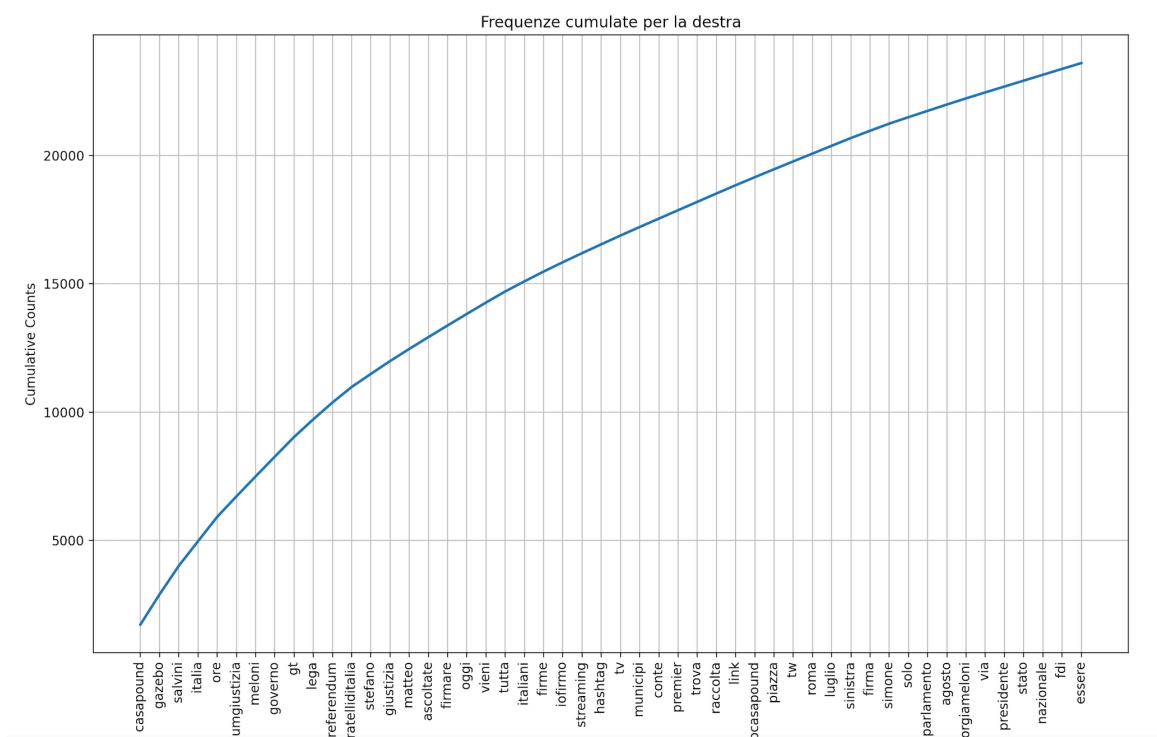


Figura 29: cumulative plot per i partiti di destra.

### 3. Conclusioni

In conclusione possiamo affermare che per quanto riguarda l' analisi dei sentiment e delle emotion i risultati hanno confermato, secondo i nostri modelli, che prevale una neutralità generale sui profili Twitter dei partiti politici e sia sulla produzione giornalistica, sia lato articoli che lato tweets.

Per quanto riguarda invece la predizione dell' orientamento politico, possiamo affermare che, dall 'apprendimento dei Tweet dati in pasto all' algoritmo, è emersa una tendenza destrorsa, potenzialmente derivante dal fatto che la produzione social dei partiti di destra è tendenzialmente più ricercata dal punto di vista della polarizzazione degli utenti, una produzione più 'catchy' che ritroviamo anche nelle testate giornalistiche italiane alla ricerca di contenuti sempre più cliccabili.