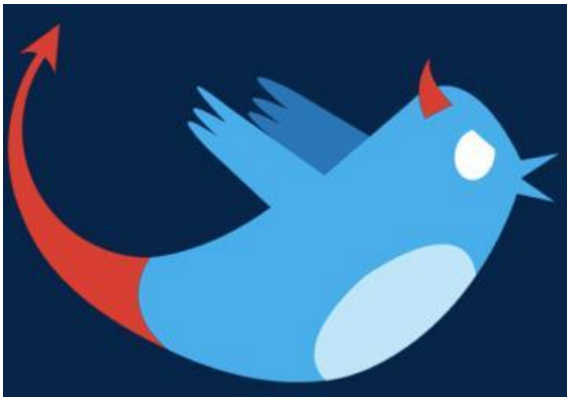
Università degli studi di Salerno

Dipartimento di Informatica

Twitter Hate Speech Analysis



0522500792 **Corso Silvio**
0522500730 **Napoli Giuseppe**
0522500865 **Tramontano Carmine**

Anno Accademico 2020-2021

Abstract

L'utilizzo dei social è largamente diffuso tra le persone di qualsiasi fascia di età, basti pensare che al giorno d'oggi il 58% della popolazione italiana fa utilizzo dei social con una crescita pari al 6,4% rispetto al 2019. [1] Chiunque può connettersi a piattaforme come Twitter, Facebook, Instagram e condividere le proprie idee tramite post, messaggi, foto e altro. È immediato, quindi, che un fenomeno tanto diffuso sia anche difficile da regolamentare sotto il punto di vista del rispetto reciproco tra i vari utenti delle community. Il nostro studio si basa proprio sull'analisi dei commenti offensivi da parte degli utenti, con una particolare attenzione sull' "Hate Speech", ovvero sui commenti discriminatori dal punto di vista razziale, religioso e di orientamento sessuale. Per questo motivo gli utenti sono stati raggruppati per categorie e grazie all'utilizzo di specifici tool, è stata data loro una valutazione in modo da identificare le categorie più affette da ciò ed arginare al meglio il fenomeno. Inoltre, è stata fornita anche una rappresentazione grafica delle informazioni analizzate così da permettere all'utente di visualizzare immediatamente e in maniera chiara quanto richiesto.

Sommario

1. Introduzione ed Obiettivi di Progetto

L'Hate Speech è un fenomeno diffuso non unicamente sulle piattaforme social, ragion per cui è molto difficile da arginare. Il nostro studio non solo si pone come obiettivo quello di confrontare due tool per l'analisi dei commenti ed identificarne il più preciso, ma aspira a poter essere la base, grazie all'utilizzo delle giuste metodologie, per combattere questo tipo di comunicazione offensiva atta a rovinare l'esperienza degli utenti con i social.

La piattaforma analizzata è Twitter in quanto è il social network che permette di lavorare ampiamente e in maniera gratuita sui propri dati grazie alla modalità "sviluppatore" concessa, in seguito ad approvazione, a chi dimostra di voler svolgere studi senza intaccare la privacy di nessun utente. Twitter non è molto usato in Italia, bensì lo è in paesi come Stati Uniti e Regno Unito [2], questo ha portato ad un'analisi mirata proprio a queste aree geografiche così da ottenere grossi quantitativi di dati da analizzare.

Numerosi sono i tool presenti in letteratura che permettono di analizzare i tweet, però la maggior parte degli studi che hanno fatto utilizzo di questi, si sono incentrati sull'analisi dei tweet e non sull'analisi dei commenti. Quest'ultima fa al caso nostro perché l'obiettivo è quello di capire quali siano le categorie maggiormente affette da "hate speech" così da poter imporre delle politiche di controllo più serrate su di esse. Un motivo per cui sono pochi gli studi effettuati sui commenti è che l'API di Twitter non permette l'estrazione precisa di commenti che viene fatta in maniera piuttosto randomica e scarna. Tuttavia, mediante diversi approcci questo paper mostra come sia possibile ottenere un numero significativo di commenti così da poter effettuare diversi tipi di analisi.

Gli strumenti, gli utenti e la rispettiva suddivisione per categorie verranno affrontati nei capitoli 2 e 3 in maggior dettaglio in modo che si possano comprendere il funzionamento dei tool e la motivazione dietro le nostre scelte.

2. Background tecnologico

Per l'implementazione del nostro progetto abbiamo scelto di utilizzare **Python 3.7**, un linguaggio di programmazione di più alto livello rispetto alla maggior parte degli altri linguaggi, orientato a oggetti. Abbiamo fatto uso, inoltre, di **Anaconda** che mira a semplificare la gestione e la distribuzione dei pacchetti. [3] I motivi per cui abbiamo scelto di utilizzare Python sono molteplici:

- Python funziona ovunque, sia Windows, Linux/UNIX, Mac.
- Ha un linguaggio tradizionale, si compone di tutti i costrutti standard come cicli, istruzioni, array eccetera. La sintassi è semplice.
- È un linguaggio completo e molto potente grazie alla filosofia "tutto compreso" relativa alle librerie.

Per i nostri fini abbiamo scelto di utilizzare varie librerie di cui Python disponeva:

- La libreria **re**, che mette a disposizione dei modelli per le espressioni regolari specificando regole per l'insieme di possibili stringhe di cui vogliamo trovare corrispondenze.
- La libreria **csv**, che ci permette sia di leggere sia di scrivere su un file CSV.
- La libreria **Pandas**, che ci permette la manipolazione e l'analisi dei dati offrendo strutture dati e operazioni per la manipolazione dei file CSV che abbiamo a disposizione.
- La libreria **MatPlotLib** per la creazione di visualizzazione statiche, animate e interattive.
- La libreria **NumPy** fornisce diversi strumenti che permettono a NumPy di essere velocemente integrata con una grande varietà di dati quali:
    - un potente oggetto array N-dimensionale,
    - funzioni sofisticate per la gestione dello stesso, tools per l'integrazione di C e C++,
    - funzioni per l'algebra lineare, trasformata di Fourier e molto altro.

Del loro utilizzo parleremo nello specifico una volta analizzate le funzioni che compongono il nostro script.

Per quanto riguarda la cattura dei tweet è stato necessario un account Twitter Developer per accedere ai vantaggi dell'API Twitter e ottenere le credenziali per l'accesso ai servizi; l'API di Twitter viene utilizzata per scaricare real-time i post di Twitter, come nel nostro caso, ma non solo. È utile per ottenere un volume elevato di tweet o per creare un feed live utilizzando un flusso del sito o un flusso utente ovvero prelevando informazione specifiche relative ad un gruppo di utenti o ad un singolo utente. Il portale per sviluppatori Twitter è un insieme di strumenti self-service che gli sviluppatori possono utilizzare per gestire il proprio accesso e per creare e gestire i propri progetti e app. Nel portale si ha la possibilità di:

- Creare e gestire progetti e app Twitter.
- Configurare ambienti di sviluppo.

Per aiutare a gestire l'enorme volume di queste richieste, vengono posti dei limiti al numero di richieste che possono essere effettuate per evitare di danneggiare Twitter, vi è infatti un possibile crash dei Server dovuto all'elevato numero di richieste. Il numero massimo di richieste consentite si

basa su un intervallo di tempo, un periodo o una finestra di tempo specificati, i limiti di velocità vengono applicati in base al metodo di autenticazione utilizzato, nel nostro abbiamo utilizzato **OAuth1.0a** con cui, per recuperare le metriche private dai Tweet, bisogna autenticarsi con i token utente associati all'utente alla creazione dell'app mediante account Developer.
I limiti di velocità dell'account Twitter Developer standard che abbiamo utilizzato:

- Ricerca Tweet recenti: 900 Richieste/15 minuti
- Ricerca User: 900 Richieste/15 minuti
- Limite massimo di Tweet: 500.000 Tweet/1 mese

2.1 Tweepy

Nella prima bozza di progetto avevamo deciso di utilizzare la libreria Tweepy. [4] per la cattura dei tweet e dei commenti fornisce un wrapper per l'API di Twitter che rende il processo di comunicazione con questa davvero rapido. Una volta creato l'account Twitter Developer, per iniziare il processo è necessario registrare l'applicazione client su Twitter; una volta registrata, si ottengono le credenziali, **consumer_key** e **consumer_secret**, che vengono salvate, in Python, nei seguenti campi:

```
consumer_key = ""
consumer_secret = ""
```

Il passo successivo è la creazione di un'istanza **OAuthHandler**, effettuata fornendo le credenziali, ed in seguito lo scambio del token di richiesta con un token di accesso. Gli access tokens forniscono il contesto utente e i permessi che consentono di utilizzare l'API di Twitter; Twitter non fa scadere i tokens, quindi l'unico caso in cui vengono invalidati è se l'utente revoca l'accesso all'applicazione. L'archiviazione del token di accesso dipende dall'applicazione, quindi quel che si fa è creare un client Twitter con le credenziali ottenute.

Fondamentalmente i passi necessari da eseguire sono i seguenti:

1. Salvare i token generati in due stringhe
2. Creare l'oggetto OAuthHandler con le key relative al consumer
3. Passarli all'oggetto OAuthHandler in maniera tale da potersi configurare con l'API twitter

```
access_token = ""
access_token_secret = ""

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
```

L'oggetto *api* ci permette di interagire con le Web API fornite da Twitter e manipolare il volume di dati che ci mettono a disposizione.

Un aspetto molto importante quando si estraggono dati dai social media è la mole dei dati da estrarre. Sarebbe infatti impensabile andare a scaricare centinaia di migliaia di tweet con una solo richiesta alle Web API. Per questi motivi la maggior parte dei social media fornisce le proprie API con un sistema di paginazione basato sui Cursori.

Cursor

Per quanto riguarda Tweepy è necessario considerare manualmente la paginazione, continuando ad iterare finché ci sono nuove pagine. Ad ogni iterazione il client farà una nuova richiesta per ottenere nuovi tweet. Utilizzando un oggetto **Cursor** si può ottenere un oggetto direttamente iterabile:

```
for tweet in tweepy.Cursor(api.user_timeline, id="twitter").items():
    print(tweet)
```

In questo caso il cursore continuerà ad ottenere dati finché questi saranno disponibili, stampando i tweet che vengono generati e gestendo automaticamente la paginazione. Il metodo Cursor() prende in input la timeline dell'utente Twitter ed il suo id Twitter. Per limitare il numero di dati acquisiti, bisogna dare in input al metodo items() il numero massimo di dati da scaricare.

Al fine di rendere completa l'analisi dei tweet, è possibile ottenere da ogni tweet i suoi commenti:

```
for full_tweets in tweepy.Cursor(api.user_timeline, id=username).items(100):
    print("Full_tweet: " + full_tweets.text)
    for comment in tweepy.Cursor(api.search, q='to:' + username,
                                 result_type='recent', timeout=999999).items(100):
        if hasattr(comment, 'in_reply_to_status_id_str'):
            if comment.in_reply_to_status_id_str == full_tweets.id_str:
                print(comment)
```

Una volta iterato sul numero di tweet dati in input, per ogni tweet si cercano tramite query tutte le replies di quel determinato tweet; se la reply trovata ha l'attributo "in_reply_to_status_id_str" che coincide con l'id del tweet, si è trovata una reply a quel tweet.

2.2 Twarc

Twarc è uno strumento, che può essere utilizzato sia a riga di comando sia come libreria Python, per l'archiviazione dei dati in formato JSON di Twitter. Ogni tweet è rappresentato come un oggetto, che è esattamente ciò che viene restituito dall'API di Twitter. [5]
Il punto di forza di Twarc è quello di superare i limiti imposti alle funzionalità dell'API di Twitter che potrebbero risultare restrittive sotto certi aspetti; difatti è in grado di gestire i limiti di velocità sollevando l'utente da tale incarico. Oltre a consentire il raccoglimento dei tweet, Twarc può anche essere utilizzato per raggruppare gli utenti, le tendenze e gli ID dei tweet.
Prima di utilizzarlo è necessario registrare un'applicazione su Twitter. Tale applicazione permette all'utente di utilizzare l'API messa a disposizione da Twitter. Una volta creata l'applicazione,

vengono annotate le consumer key, consumer secret e di conseguenza si va a generare un access token e un access token secret. Con queste quattro variabili si è pronti per utilizzare Twarc. Si ha quindi un procedimento analogo a Tweepy proprio perché di base si fa utilizzo dell'API fornita da Twitter.
Per prima cosa si dovranno comunicare a Twarc le chiavi API dell'applicazione concedendo l'accesso a uno o più account Twitter, attraverso il comando: *twarc configure*. Questo memorizzerà le credenziali in un file chiamato .twarc nella directory home così che non ci sia più bisogno di reinserirle; esse possono anche settate manualmente mediante l'utilizzo delle variabili d'ambiente.

La libreria Twarc può essere vista come un'estensione della libreria Tweepy perché offre un vasto gruppo di funzioni applicabili all'estrazione e all'utilizzo dei tweet, di fatti non tutti questi metodi sono facilmente implementabili con l'API di twitter. Tra le funzioni più importanti ricordiamo:

- **Search**: utilizza la ricerca dell'API di twitter per scaricare tweet pre-esistenti che matchano con una determinata query
- **Filter**: utilizza il filtro dell'API di twitter per ottenere tweet non appena vengono pubblicati
- **Sample**: scarica un campione dei tweet più recenti
- **Dehydrate**: genera una lista di id da un file di tweet
- **Hydrate**: a partire da un file contenente una lista di id, genera i corrispondenti tweet per ciascuno di esso sotto forma di oggetto JSON.
- **Users**: restituisce tutti i metadati relativi all'utente passato in input
- **Followers**: restituisce gli id dei follower relativi ad un utente passato in input
- **Trends**: restituisce informazioni relativi agli hashtag del momento
- **Timeline**: restituisce i tweet più recenti relativi ad uno specifico utente
- **Retweets**: restituisce tutti i retweet relativi ad uno specifico tweet
- **Replies**: funzione per l'ottenimento dei commenti relativi ad uno specifico utente

Search

Dopo aver creato un'istanza Twarc usando le credenziali di accesso ottenute da Twitter, la utilizziamo per iterare attraverso i risultati ottenuto tramite il metodo **Search**, come nel seguente codice:

```
for tweet in t.search(username, max_pages=1, result_type='recent'):
    # Get replies to that tweet
    print(tweet)
```

Replies

Replies è un metodo caratterizzante di twarc che si basa sempre sull'ottenimento delle informazioni relative ai tweet ma in particolar modo delle risposte/commenti. Come detto in precedenza, non è possibile ottenere i commenti relativi ai post unicamente con l'API di twitter ma Replies sfrutta una

serie di metodi per fare ciò. Tuttavia, non è in grado di superare il limite relativo all'ottenimento di tweet più vecchi di una settimana e quindi le risposte ottenute sono solo quelle inviate negli ultimi sette giorni.

2.3 Vader

VADER (Valence Aware Dictionary and sEntiment Reasoner) è uno strumento di *sentiment analysis* basato su regole e lessico che sono specificamente in sintonia con i sentimenti espressi nei social media. [6] È completamente open-source sotto la Licenza MIT. La demo più completa per vaderSentiment.py è strutturata nel seguente modo:

- **Esempi di casi d'uso tipici** per l'analisi del sentiment, inclusa la corretta gestione delle frasi con:
  - Negazioni tipiche (ad es. "not good").
  - Uso delle contrazioni come negazioni (ad esempio, "*wasn't very good*").
  - Uso convenzionale della punteggiatura per segnalare una maggiore intensità del sentiment (ad es. "Good!!!").
  - Uso convenzionale della forma della parola per segnalare l'enfasi (ad esempio, utilizzando TUTTO MAIUSCOLO per parole / frasi).
  - Utilizzo di modificatori di grado per alterare l'intensità del sentimento (ad es. potenziatori di intensità come "very" e smorzatori di intensità come "kind of").
  - Comprensione di molte parole gergali cariche di sentimento (ad esempio, "sux").
  - Comprensione di molte parole gergali cariche di sentimenti come "uber" o "friggin" o "kinda".
  - Comprensione di molte emoticon cariche di sentimento come :) e: D.
  - Traduzione di emoji codificati utf-8 come  e  e  .
  - Comprensione di inizialismi e acronimi carichi di sentimenti (ad esempio: 'lol').
- Altri **esempi di frasi complesse** che confondono altri strumenti di analisi del sentiment.
- Esempio di come VADER può lavorare insieme a **NLTK** (Natural Language Toolkit) per fare analisi del sentiment su testi più lunghi, cioè scomporre paragrafi, articoli/rapporti/pubblicazioni o romanzi in analisi a livello di frase.
- Esempi di un concept per valutare il sentiment di immagini, video o altri contenuti multimediali con **tag**.
- Con un accesso ad Internet, la demo contiene un esempio di come VADER può lavorare con l'analisi del sentiment di **testi in altre lingue** (frasi di testo non in inglese).

Risorse e descrizioni dei set di dati

vader_lexicon.txt:

- Formato: il file è delimitato da tabulazioni con TOKEN, SENTIMENT-RATING MEDIO, DEVIAZIONE STANDARD e RAW-HUMAN-SENTIMENT-RATINGS.

- Descrizione: convalidato empiricamente da molti esperti del settore, VADER incorpora un lessico dei sentimenti "gold standard" che è particolarmente in sintonia con contesti simili a microblog.

Il lessico del sentiment di VADER è sensibile sia alla **polarità**, ovvero un valore che indica se una parola si avvicina più all'essere etichettata come "positiva", "negativa" o "neutrale", sia all'**intensità**, ovvero quanto effettivamente quella parola appartiene a una delle etichette precedenti. Di seguito una piccola parte del dizionario di VADER:

```
nicely  1.9     0.83066 [2, 1, 2, 2, 4, 2, 1, 1, 2, 2]
niceness        1.6     0.66332 [1, 1, 3, 2, 1, 1, 2, 2, 2, 1]
nicenesses      2.1     1.22066 [4, 0, 3, 1, 2, 1, 4, 2, 2, 2]
nicer   1.9     0.53852 [2, 2, 1, 2, 3, 2, 2, 2, 1, 2]
nicest  2.2     0.87178 [1, 4, 1, 2, 2, 2, 3, 3, 2, 2]
niceties        1.5     1.20416 [1, 4, 1, 1, 2, 1, 0, 3, 2, 0]
nicety  1.2     1.07703 [1, 0, 4, 1, 1, 0, 2, 1, 1, 1]
nifty   1.7     0.64031 [2, 2, 1, 1, 1, 2, 3, 2, 2, 1]
niggas  -1.4    2.2     [-4, -3, 2, 1, -4, -2, 0, -1, 1, -4]
nigger  -3.3    1.18743 [-4, -4, -4, -4, -4, -4, -1, -3, -1, -4]
no      -1.2    0.74833 [-1, -1, -1, -1, -1, -1, 0, -1, -2, -3]
noble   2.0     0.89443 [2, 1, 2, 2, 3, 0, 2, 3, 2, 3]
noisy   -0.7    0.64031 [-2, 0, -1, -1, 0, -1, -1, 0, 0, -1]
nonsense        -1.7    0.64031 [-3, -1, -1, -1, -2, -1, -2, -2, -2, -2]
noob    -0.2    1.16619 [-2, 0, -1, 0, -1, 2, 1, 1, -1, -1]
nosey   -0.8    1.16619 [-2, -2, -2, 1, -1, -1, 0, 1, 0, -2]
notorious       -1.9    1.3     [-2, -4, -3, -2, -2, -2, -1, -1, -3, 1]
novel   1.3     0.64031 [2, 0, 1, 1, 1, 1, 2, 2, 2, 1]
numb    -1.4    0.66332 [-1, -1, -1, -1, -2, -1, -1, -3, -2, -1]
numbat  0.2     0.4     [0, 0, 0, 0, 1, 1, 0, 0, 0, 0]
numbed  -0.9    0.53852 [-1, -1, -1, -1, 0, 0, -2, -1, -1, -1]
```

La creazione manuale (molto meno, la convalida) di un lessico dei sentimenti completo è un processo laborioso e talvolta soggetto a errori, quindi non c'è da meravigliarsi che molti ricercatori e professionisti di opinion mining si affidino così tanto ai lessici esistenti come risorse primarie. VADER offre questo lessico come nuova risorsa, iniziando costruendo un elenco ispirato dall'esame di banche di parole di sentiment consolidate esistenti (LIWC, ANEW e GI). A questo, vengono incorporate numerose caratteristiche lessicali comuni all'espressione del sentiment nei microblog, tra cui:

- Un elenco completo di emoticon in stile occidentale(*:-)* che denota una faccina sorridente e generalmente indica un sentimento positivo)

- Acronimi e inizialismi legati al sentimento (*LOL* e *WTF* sono entrambi esempi di inizialismi carichi di sentimento).
- Gergo comunemente usato con valore sentimentale (*nah*, *meh* e *giggly*).

vaderSentiment.py:

Il codice Python per il motore di analisi del sentiment è basato su regole. Esso implementa tali regole grammaticali e sintattiche descritte nel documento, incorporando quantificazioni derivate empiricamente per l'impatto di ciascuna regola sull'intensità percepita del sentiment nel testo a livello di frase. È importante sottolineare che queste euristiche vanno oltre ciò che normalmente sarebbe catturato in un tipico modello di set di parole. Incorporano relazioni sensibili all'ordine delle parole tra i termini. Ad esempio, i modificatori di grado (chiamati anche intensificatori, parole di richiamo o avverbi di grado) influiscono sull'intensità del sentiment aumentando o diminuendo l'intensità, come ad esempio:

i. "The service here is extremely good".
ii. “The service here is good”.
iii. "The service here is marginally good".

Informazioni sul punteggio

Il punteggio composto viene calcolato sommando i punteggi di valenza di ciascuna parola nel lessico, aggiustato secondo le regole e quindi normalizzato per essere compreso tra -1 (estremamente negativo) e +1 (estremo positivo). Questa è la metrica più utile se si desidera una singola misura unidimensionale del sentimento per una data frase. Definirlo un "punteggio composito normalizzato e ponderato" è accurato.
È anche utile per i ricercatori che desiderano impostare soglie standardizzate per classificare le frasi come positive, neutre o negative. I valori di soglia tipici (utilizzati nella letteratura citata in questa pagina) sono:

1. Sentiment positivo: punteggio composto> = 0,05.
2. Sentimento neutro: (punteggio composto> -0,05) e (punteggio composto <0,05).
3. Sentiment negativo: punteggio composto <= -0,05.

```
# --- examples -------
sentences = ["VADER is smart, handsome, and funny.",  # positive sentence example
             "VADER is smart, handsome, and funny!",  # punctuation emphasis handled correctly (sentiment in
             "VADER is very smart, handsome, and funny.", # booster words handled correctly (sentiment inten
             "VADER is VERY SMART, handsome, and FUNNY.",  # emphasis for ALLCAPS handled
             "VADER is VERY SMART, handsome, and FUNNY!!!", # combination of signals - VADER appropriately a
             "VADER is VERY SMART, uber handsome, and FRIGGIN FUNNY!!!", # booster words & punctuation make
             "VADER is not smart, handsome, nor funny.",  # negation sentence example
             "The book was good.",  # positive sentence
             "At least it isn't a horrible book.",  # negated negative sentence with contraction
             "The book was only kind of good.", # qualified positive sentence is handled correctly (intensit
             "The plot was good, but the characters are uncompelling and the dialog is not great.", # mixed
             "Today SUX!",  # negative slang with capitalization emphasis
             "Today only kinda sux! But I'll get by, lol", # mixed sentiment example with slang and constras
             "Make sure you :) or :D today!",  # emoticons handled
             "Catch utf-8 emoji such as such as 💘 and 💋 and 😁",  # emojis handled
             "Not bad at all"  # Capitalized negation
             ]
```

I punteggi *pos*, *neu* e *neg* sono rapporti per le proporzioni di testo che rientrano in ciascuna categoria (quindi questi dovrebbero sommare tutti per essere 1 ... o vicini ad esso con l'operazione float). Queste sono le metriche più utili se si desiderano misure multidimensionali del sentiment per una data frase.

```
handsome, and funny.------------------------------ {'pos': 0.746, 'compound': 0.8316, 'neu': 0.254, 'neg': 0.0}
handsome, and funny!------------------------------ {'pos': 0.752, 'compound': 0.8439, 'neu': 0.248, 'neg': 0.0}
art, handsome, and funny.------------------------- {'pos': 0.701, 'compound': 0.8545, 'neu': 0.299, 'neg': 0.0}
ART, handsome, and FUNNY.------------------------- {'pos': 0.754, 'compound': 0.9227, 'neu': 0.246, 'neg': 0.0}
ART, handsome, and FUNNY!!!----------------------- {'pos': 0.767, 'compound': 0.9342, 'neu': 0.233, 'neg': 0.0}
ART, uber handsome, and FRIGGIN FUNNY!!!---------- {'pos': 0.706, 'compound': 0.9469, 'neu': 0.294, 'neg': 0.0}
rt, handsome, nor funny.-------------------------- {'pos': 0.0, 'compound': -0.7424, 'neu': 0.354, 'neg': 0.646}
d.------------------------------------------------ {'pos': 0.492, 'compound': 0.4404, 'neu': 0.508, 'neg': 0.0}
t a horrible book.-------------------------------- {'pos': 0.363, 'compound': 0.431, 'neu': 0.637, 'neg': 0.0}
y kind of good.----------------------------------- {'pos': 0.303, 'compound': 0.3832, 'neu': 0.697, 'neg': 0.0}
d, but the characters are uncompelling and the dialog is not great. {'pos': 0.094, 'compound': -0.7042, 'neu':
-------------------------------------------------- {'pos': 0.0, 'compound': -0.5461, 'neu': 0.221, 'neg': 0.779}
 sux! But I'll get by, lol------------------------ {'pos': 0.317, 'compound': 0.5249, 'neu': 0.556, 'neg': 0.127}
 or :D today!------------------------------------- {'pos': 0.706, 'compound': 0.8633, 'neu': 0.294, 'neg': 0.0}
i such as 💘 and 💋 and 😁-------------------- {'pos': 0.279, 'compound': 0.7003, 'neu': 0.721, 'neg': 0.0}
-------------------------------------------------- {'pos': 0.487, 'compound': 0.431, 'neu': 0.513, 'neg': 0.0}
```

2.4 Hate Sonar

"Hate Sonar" è un tool Python utilizzato per l'Hate Speech detection. [7] Come Vader, il suo funzionamento è piuttosto semplice ed intuitivo, non c'è alcun bisogno di allenare il modello ma bisogna semplicemente passare ad Hate Sonar delle frasi da analizzare ed avremo in output un punteggio. Il punteggio viene espresso in percentuale e suddiviso secondo tre parametri:

- **Hate Speech**: è il parametro riguardante proprio l'Hate Speech, maggiore è il suo valore, maggiore è la probabilità che la frase possa essere ricollegata ad esso o che almeno contenga dei possibili riferimenti.
- **Offensive language**: Lo strumento fa una differenziazione tra "hate" e "offensive", di fatti la valutazione viene fatta seguendo due dizionari differenti. Questa differenza è necessaria

proprio perché l'hate una "specializzazione" dell'offensive per cui rientrano termini più specifici legati, ad esempio, all'etnia.

- **Neither**: Quando lo strumento non è in grado di valutare al meglio la frase, perlopiù per una mancanza di un numero significativo di vocaboli nel dizionario, allora viene incrementato il punteggio relativo a "neither"

Hate Sonar ha costruito il proprio dizionario basandosi su un altro lavoro, "hate-speech-and-offensive-language"; [8][9] di seguito un estratto del .csv per rendere l'idea delle parole che possono essere ricondotte all' Hate Speech:

	ngram	prophate
2	allah akbar	0.87
3	blacks	0.583
4	chink	0.467
5	chinks	0.542
6	dykes	0.602
7	faggot	0.489
8	faggots	0.675
9	fags	0.543
10	homo	0.667
11	inbred	0.583
12	nigger	0.584
13	niggers	0.672
14	queers	0.5
15	raped	0.717
16	savages	0.778
17	slave	0.667

Come si può notare le parole utilizzate sono appartenenti allo slang "americano/inglese" e tutte riconducibili ad argomenti sensibili. A ciascuna parola/frase viene assegnato un punteggio il quale avrà poi un peso sul rate finale della frase.

3. Progettazione e Implementazione

In questo capitolo tratteremo la scelta dei personaggi pubblici che hanno contraddistinto l'analisi effettuata tramite il nostro tool e le categorie in cui li abbiamo inseriti (Lewis Hamilton→Sportivi). Entreremo nel dettaglio per ciò che riguarda gli script e le librerie che inizialmente hanno influenzato l'approccio iniziale, l'utilizzo di Tweepy per il download dei tweet e dei relativi commenti e quelle che hanno contribuito all'implementazione finale del nostro tool, ovvero Twarc.

3.1 Individuazione categorie e approccio iniziale

Analizzando il problema dell'Hate Speech e del significato che lo circonda, abbiamo concentrato la nostra attenzione sia su personaggi pubblici influenzati in prima persona da questo problema sia su personaggi pubblici che contribuiscono alla diffusione di questo problema, concentrandoci anche sulle categorie alla quale questi personaggi appartengono per avere un'analisi più approfondita. Siamo stati in grado di distinguere 5 categorie principali:

- **Politici**
  - Barack Obama, Michelle Obama, Boris Johnson, Greta Thunberg(attivista per lo sviluppo sostenibile e contro il cambiamento climatico), Joe Biden e Donald Trump(coinvolti in questo periodo nella battaglia politica per le elezioni americane)
- **Sportivi**
  - Lebron James, Marcus Rashford, Shaquille O'Neal, Lewis Hamilton(veste un ruolo importante nel movimento Black Lives Matter)
- **Influencer**
  - Ariana Grande, Nicki Minaj, Dwayne Johnson(The Rock)
- **Attori**
  - Kevin Hart, Samuel L.Jackson, Channing Tatum, Chris Pratt
- **YouTuber**
  - Casey Neistat, Ellen DeGeneres, Daniel Middleton, Gary Vaynerchuk, Richard Tyler Blevins

Ognuno dei personaggi scelti per ogni categoria è soggetto al problema dell'Hate Speech o è causa del problema. In media sono stati scelti 5 personaggi per ogni categoria in modo da ottenere più precisione nel calcolo dei dati. Oltre ad aver preso in considerazione personaggi pubblici, ci siamo concentrati anche su 3 pagine che trattano diversi temi:

- **Testate Giornalistiche**
  - CBS News, France24, The Independent, The Guardian, Sky News, The Times
- **Pagine Sportive**
  - BBC, Champions League, England, Lakers
- **Pagine di Divulgazione Scientifica**
  - CERN, National Geographic, WIRED, New Scientist

Testate Giornalistiche e Pagine Sportive sono coinvolte in prima persona nel problema dell'Hate Speech, le pagine di divulgazione scientifica non sono soggette a questo problema ma sono state inserite per analizzare la sostanziale differenza tra le pagine.

L'approccio iniziale per l'analisi dei tweet di ogni personaggio e dei commenti ad ogni tweet si è basato sull'utilizzo della libreria **Tweepy**. Una volta ottenuto l'account Twitter Developer ed aver creato l'app nell'ambiente di sviluppo ottenendo le chiavi di autenticazione, abbiamo inizialmente creato la funzione ***get_tweets****(username, category)* che prende in input l'username Twitter del personaggio di cui vogliamo analizzare i tweet e la categoria a cui questo personaggio appartiene. Il metodo inizialmente imposta il numero di tweet che sono di nostro interesse, in questo caso 100, creava ogni volta un dizionario con i campi relativi ai dati del tweet che ci interessavano ed iterava tramite il metodo Cursor() tra i tweet restituendo i 100 tweet richiesti. A tale metodo passavamo la timeline del personaggio e il suo username Twitter e venivano inseriti progressivamente i tweet nel dizionario creato in precedenza. Una volta ottenuto il numero di tweet richiesto, creavamo un file CSV "*category_username_tweets.csv*" ed ogni volta sovrascrivevamo questo file con il nuovo dizionario composto dai tweet come nel seguente codice:

```
def get_tweets(username, category):
    # Number of tweets to download
    number_of_tweets = 100
    # get tweets randomly between a number of 1 to 100
    tweets_for_csv = [["Username", "Data", "Tweet"]]
    for tweet in tweepy.Cursor(api.user_timeline, screen_name=username).items(number_of_tweets):
        # create array of tweet information: username, date/time, text
        tweets_for_csv.append([username, tweet.created_at, tweet.text.encode("ascii", "ignore")])

    # Writing tweet obtained in a csv
    header = ['Username', 'Data', 'Tweet']
    outfile = category + "_" + username + "_tweets.csv"
    print("writing to " + outfile)
    # Adding tweets
    with open(outfile, 'w', newline='') as file:
        writer = csv.writer(file, delimiter=',')
        writer.writerows(tweets_for_csv)
```

Per ogni tweet eravamo interessati all'username del personaggio in questione, alla data di pubblicazione del tweet e al testo del tweet, dati che ci sarebbero serviti per l'analisi. La cattura dei tweet andava a buon fine, il numero di tweet che ottenevamo non era precisamente 100 per via dei limiti dovuti all'API di Twitter ma circa il 60% dei tweet richiesti veniva ricavato. L'output che derivava dall'esecuzione della funzione era il seguente:

```
b'@MercedesAMGF1 '
b'#92  https://t.co/4jcrZTTfUb'
b'And to you #TeamLH, I cant express how grateful I am for all of you. All I can say is thank you, from the bottom o https://t.co/83QmquM
b'Its been such a privilege to work with you all. Im thinking of Michael today, I will forever have the utmost admi https://t.co/waOyLeDE
b'NINETY TWO WINS  Today is beyond my wildest dreams. I couldnt be here today without my team, continuously innovat https://t.co/8NWZXwaX
b'We all have a responsibility to educate ourselves and raise awareness of the tragedies happening in the world aroun https://t.co/9v5Tvs
b'97th pole  This track is tricky! Really blown away with all the support here this weekend. So nice to see the fa https://t.co/nUD2JUfde
b'@Charles_Leclerc @MercedesAMGF1 Naughty Roscoe! Ill have a word with him '
b'@F1Tricky @MercedesAMGF1 @LauraMcdonoug20 '
b'Made it to Portugal! Hope you are all staying safe  https://t.co/mKWllT24VT'
b'We had a tyre blow out on the motorway and had to stop at the station. Took a minute to change the wheel but once d https://t.co/zpKYd1
b'Still I Rise. https://t.co/6Dj6GarNpw'
b'Weekend vibes with Roscoe https://t.co/QIZbyqIqlm'
b'There is no I in team. Youve all heard this saying before but last weekend was truly an example of that. When a https://t.co/XoHXVmLCrv
b'@sou_draws @DivyaHeartsLew @MercedesAMGF1 @F1 Wow '
b'Roscoe and I are staying safe, I hope you are too  https://t.co/FZClB79dqJ'
b'Outdoor training is the best, fresh air and all. I love to hike. Wishing everyone an amazing day https://t.co/opSS8dUdBl'
writing to Sportivi_LewisHamilton_tweets.csv
```

Una volta eseguita la funzione, i testi di tutti i tweet venivano poi salvati nel file CSV.

Per una completa analisi, oltre alla cattura dei tweet, era necessario anche la cattura dei commenti in quanto era questo l'obiettivo stabilito inizialmente. Per questo motivo avevamo definito la funzione ***get_comments_from_tweets****(username, category)* a cui passavamo gli stessi parametri della funzione precedente. Una volta iterato sulla timeline del personaggio in questione tramite il metodo Cursor(), iteravamo sui tweet di questo personaggio passandogli la query "*to: username*" che indica la risposta ad un tweet; in seguito si controllava se il tweet era in possesso dell'attributo "*in_reply_to_statut_id_str*", in caso positivo bisognava accertarsi se quest'attributo ed il campo *id_str* del tweet coincidevano; infine, se il controllo dava esito positivo, salvavamo il testo del commento in un array il cui contenuto sarebbe stato salvato in un file denominato "*category_username_comments.csv*". Il codice che rappresenta la corrispondente funzione è il seguente:

```python
def get_comments_from_tweets(username, category):
    replies = []
    for full_tweets in tweepy.Cursor(api.user_timeline, screen_name=username, timeout=999999).items(2):
        #replies.append([username, full_tweets.id_str, full_tweets.created_at, full_tweets.text.encode("ascii", "ignore")])
        print("Full_tweet: " + full_tweets.text)
        for tweet in tweepy.Cursor(api.search, q='to:' + username, result_type='recent', timeout=999999).items(100):
            if hasattr(tweet, 'in_reply_to_status_id_str'):
                if tweet.in_reply_to_status_id_str == full_tweets.id_str:
                    replies.append([tweet.text.encode("ascii", "ignore")])
                    print("Tweet :", tweet.text)
    outfile = category + "_" + username + "_comments.csv"
    print("writing to " + outfile)
    with open(outfile, 'w+') as file:
        writer = csv.writer(file, delimiter=',')
        writer.writerows(replies)
```

Per ogni tweet cercavamo di ottenere 100 commenti salvati in un nuovo file CSV che poi avremmo analizzato. Una volta che il programma terminava, nel file CSV contenente i commenti non erano presenti i 100 commenti richiesti ma in media 10 commenti per tweet; il motivo per cui non ottenevamo 100 commenti è nei limiti dell'account Twitter Developer Standard che non permette

il salvataggio di un gran numero di commenti per ogni tweet. L'output della funzione era il seguente:

```
Tweet : @LewisHamilton CONGRATULATIONS Sir Lewis Hamilton  @LewisHamilton and everyone   Please support @Iam_Claude25 with
Tweet : @LewisHamilton Awwww, y'all look good. Nice pic. Happy new year man❤
Tweet : @LewisHamilton Happy New year please sir we have been sending alot of this message to people but all to no avail s
Tweet : @LewisHamilton We are waiting for your knighthood acceptance statement.  Hopefully, you will not turn it down.
Tweet : @LewisHamilton Happy New year Lewis and wish you all the best and look forward to seeing you racing God bless
Tweet : @LewisHamilton Love this!
Tweet : @LewisHamilton Wait to feel the love for your children... then you will know what really love is... Happy new year
Tweet : @LewisHamilton Beautiful
Tweet : @LewisHamilton Incredible uncle!!
Tweet : @LewisHamilton ❤❤❤Happy New Year.
Tweet : @LewisHamilton Happy New Year Lewis. Well Done My Friend on Your Knighthood.
Sir Lewis Hamilton. I love it, plus H… https://t.co/q0BA5SN5nW
Tweet : @LewisHamilton Wonderful to see you with your family. Congratulations on your knighthood, it's overdue. Thank you
Tweet : @LewisHamilton SPENDING TIME WITH NEAR AND DEAR IS GREAT
Tweet : @LewisHamilton Feliz Ano Novo.
Tweet : @LewisHamilton They are beautiful Lewis. Blesses to you and your family!!🙏
Tweet : @LewisHamilton You're right Sir
Full_tweet: https://t.co/JKKxdPyCkJ
Tweet : @LewisHamilton Please go destroy this guy team LH44 https://t.co/GALbhMs4dV
Tweet : @LewisHamilton Pay your tax
Tweet : @LewisHamilton CONGRATULATIONS Sir Lewis Hamilton  @LewisHamilton and everyone   Please support @Iam_Claude25 with
writing to Sportivi_LewisHamilton_comments.csv
```

Full tweet rappresentava il tweet ottenuto, mentre tweet rappresentava il commento al full tweet; il numero di commenti ricevuto per ogni tweet non era mai lo stesso, alcuni commenti venivano inoltre ripetuti tra i vari full tweet. A causa di questi limiti abbiamo deciso di non usufruire della libreria Tweepy che quindi non faceva al caso nostro e di utilizzare la libreria Twarc per la loro cattura.

3.2 Script "extract.py"

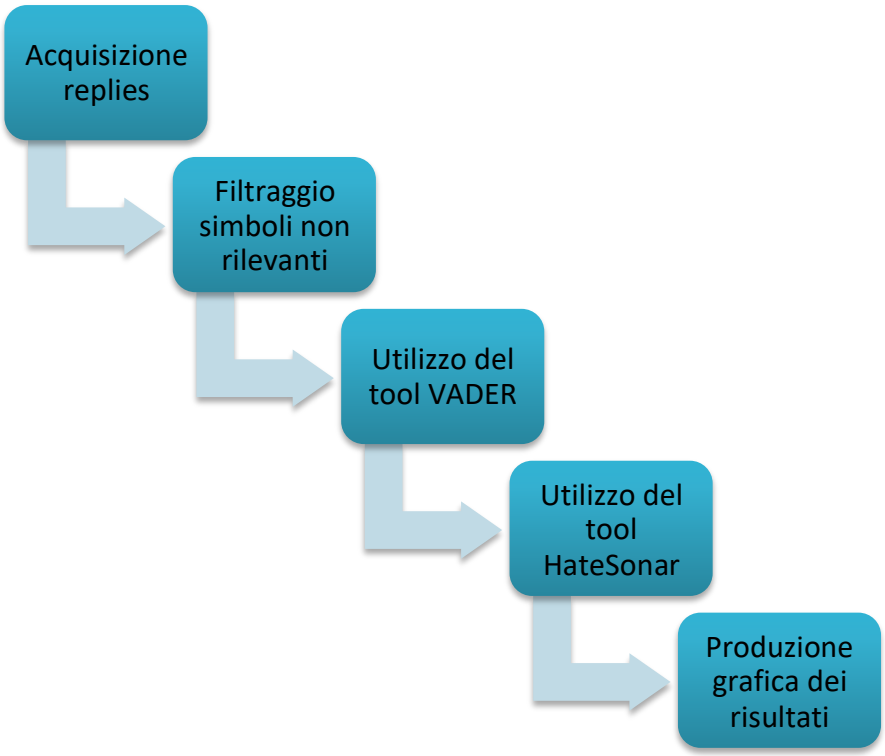
Lo script python che verrà presentato in questo paragrafo, è da considerarsi come la parte fondamentale di tutto il progetto in questione. Esso è deputato all'acquisizione delle informazioni e all'analisi delle stesse, tramite opportuno input, del profilo Twitter (*username*) che si intende analizzare e la categoria (*category*) a cui tale profilo appartiene. Di seguito vi è l'immagine che mostra il *main* dello script:

```python
if __name__ == '__main__':
    # Fornisco l'username e categoria
    username = "joebiden"
    category = "Politici"
    get_replies(username, category)
    csv_cleaning(username, category)
    calculate_vader_score(username, category)
    calculate_hatespeech_score(username, category)
    create_score_csv(username, category)
```

Quindi lo script dopo aver acquisito i parametri enunciati in precedenza, si occupa di:

1. Fornirli alla funzione ***get_replies()*** la quale si occuperà, come vedremo in seguito, dell'acquisizione dei tweet per quel dato utente, e dopo aver scaricato tali tweet, la funzione si preoccuperà di creare un file CSV con essi.
2. Successivamente viene richiamata la funzione ***csv_cleaning()***, dove anch'essa prenderà come parametri i suddetti username e category. Questa funzione, esegue un passaggio che si è reso fondamentale per la successiva fase di sentiment analysis, poiché essa va ad effettuare una rimozione di tutto ciò che è superfluo per l'analisi, rendendo i tweet molto più leggibili e soprattutto alleggerendo molto il lavoro, in termini computazionali, del tool di analisi.
3. A questo punto viene richiamata una delle funzioni cardine dello script, ovvero ***calculate_vader_score()***, che anche in questo caso lavora con i parametri username e category. Questa funzione istanzia un modulo che richiamerà il primo tool utilizzato per la sentiment analysis, ovvero VADER. Come si vedrà successivamente con maggior dettaglio, essa effettuerà tutte le operazioni del caso in merito all'analisi, restituendo lo stesso CSV ma con annessi altri tre parametri: *positive rate*, *negative rate*, *neutral rate*.
4. Dopo aver ultimato l'analisi con il tool VADER, e quindi avendo un nuovo file con questi tre valori, viene aumentata la specifica di analisi, con un ulteriore tool di analisi deputato della ricerca di particolari keywords atte a specificare se nel tweet, preso nella sua interezza, siano presenti frasi che inneggiano all'**hate speech** oppure, frasi offensive o nessuna delle due. La funzione in questione ***calculate_hatespeech_score()***, istanzia un oggetto di tipo Sonar ed effettua le analisi del caso e, come con VADER, va a concatenare nel file csv preso in input altri parametri: *hate rate*, *offensive rate*, *neither rate*.
5. Infine attraverso ***create_score_csv()*** vengono prodotti tutti i grafici corrispondenti alle analisi fatte affinché l'utente possa avere una visione globale di quanto ottenuto e quindi andare a rilevare con elevata precisione dove sono presenti o, nel caso positivo, dove non sono presenti commenti negativi.

Il work-flow dello script può essere descritto semplicemente attraverso il seguente grafico:



3.2.1 Ottenimento dei commenti

In questa sezione del documento, viene affrontato il discorso inerente all'acquisizione dei tweet e la pulizia che viene effettuata su di essi per la futura analisi effettuata dai tools. Facendo riferimento a ciò che è stato detto nella descrizione generica, la funzione ***get_replies()*** che prende come parametri l'username e la categoria del profilo in questione, inizia con la costruzione del file CSV, che salverà i tweet secondo l'ordine: Username, Data e Reply. Il ciclo for più esterno si collega attraverso l'oggetto istanziato con Twarc, per iterare in modo randomico sul profilo dell'utente, fornito come parametro, alla ricerca di un tweet. Una volta individuato il tweet, un secondo ciclo for viene eseguito sul tweet, per la cattura dei *replies*. Questi ultimi verranno catturati e salvati nel CSV costruito all'inizio seguendo l'ordine dato; tutto ciò finché non vengono catturati un numero valido di replies precedentemente stabilito. Una volta effettuate le operazioni appena descritte, esaurendo il numero di replies catturati, viene salvato il file CSV mediante la dicitura: *username_replies.csv*. Ed ovviamente questo file andrà nella cartella rinominata con la categoria apposita.

```python
# Function that create the .csv of replies
def get_replies(username, category):
    i = 0
    replies = [["Username", "Data", "Reply"]]
    # Select randomly a tweet from the selected user (username)
    for tweet in t.search(username, max_pages=1, result_type='recent'):
        # Get replies to that tweet
        for reply in t.replies(tweet):
            replies.append([username, reply["created_at"], reply["full_text"].encode("ascii", "ignore")])
            if i == 10:
                break
            i += 1

    outfile = "./" + category + "/" + username + "_replies.csv"
    print("writing to " + outfile)
    with open(outfile, 'w', newline='') as file:
        writer = csv.writer(file, delimiter=',')
        writer.writerows(replies)
```

Successivamente, al termine di questa funzione, quindi alla creazione del file CSV, viene richiamata una seconda procedura, la quale permette di attuare una sorta di filtraggio sui caratteri che risultano essere inutili per la sentiment analysis. La funzione in questione è ***csv_cleaning()***:

```python
def csv_cleaning(username, category):
    df = pd.read_csv("./" + category + "/" + username + "_replies.csv")
    space_pattern = '\s+'
    # Reply cleaning
    for i, row in df.iterrows():
        row["Reply"] = re.sub(r"https?://[A-Za-z0-9./]*", '', row["Reply"])
        row["Reply"] = re.sub(r"@[\w]*", '', row["Reply"])
        row["Reply"] = re.sub(r"RT @[\w]*:", '', row["Reply"])
        row["Reply"] = re.sub(r"RT :", '', row["Reply"])
        row["Reply"] = re.sub(space_pattern, ' ', row["Reply"])
        row["Reply"] = row["Reply"].replace("RT", '')

        # Cleaning quotes from row Reply
        row["Reply"] = row["Reply"].replace("  ", '')
        row["Reply"] = row["Reply"].replace("   ", '')
        row["Reply"] = row["Reply"].replace("    ", '')

        # Deleting 'b' char at the beginning of the text
        if row["Reply"][0] == 'b':
            row["Reply"] = row["Reply"][1:]

        # Update the dataframe with the new rows
        df.loc[i, "Reply"] = row["Reply"].lower()

        # Deleting rows that have no replies
        df = df[df.Reply != "' '"]
        df = df[df.Reply != "'  '"]
        df = df[df.Reply != "'   '"]
    df.to_csv("./" + category + "/" + username + "_replies.csv", sep=',', index=False)
```

L'idea di creare una funzione simile per la scrematura di questi possibili simboli nasce dal fatto che i tool di sentiment analysis, producono delle performance migliori laddove il tweet è chiaro nella

sua forma sia in termini di testo ma anche di emoji e similari. Infatti, questa funzione non va a modificare l'integrità del tweet bensì riformatta quest'ultimo, in ottica di una complessità computazionale inferiore e una precisione maggiore dell'analisi.
La funzione parte con la lettura, attraverso username e categoria, del file CSV di interesse e subito dopo viene lanciato un ciclo for dove vi sono istruzioni che individuano e rimuovono i determinati simboli come:

- spazi superflui o addirittura ripetuti anche per una digitazione errata da parte di un utente,
- link che portavano su pagine terze che potevano inquinare l'analisi,
- simboli ripetuti che venivano concatenati nel momento dell'estrazione attraverso il flusso di download,
- carattere "b" che veniva inserito in maniera automatica dalla funzione di estrazione per indicare tale stringa come "reply",
- stringa "RT" per rimuovere i replies relativi a dei Retweet,
- "@" per rimuovere i tag verso altri utenti

La funzione si è rivelata molto efficiente in termini di performance, infatti essa dopo aver analizzato il file CSV e aver effettuato il filtraggio, effettua una operazione di sovrascrittura del CSV precedente, producendo quindi un file CSV molto più pulito, pronto per l'analisi da parte dei tool.

3.2.2 Utilizzo dei tool di analisi

Dopo aver ottenuto tutte le informazioni di nostro interesse e averle raggruppate, come descritto in precedenza, in file CSV, ciò che viene fatto è applicare i tool di analisi "Vader" e "Hate Sonar" che sono stati implementati rispettivamente attraverso i metodi ***calculate_vader_score()*** e ***calculate_hatesonar_score()***. Di seguito verranno descritti in dettaglio come operano e ciò che si ottiene dai metodi.

Il metodo **calculate_vader_score()** prende in input due parametri che sono username e category, questi due parametri sono necessari per poter scegliere il CSV di cui si vuole analizzare il contenuto. Dopo aver scelto la colonna del CSV di nostro interesse, ovvero quella dei commenti, si andrà ad utilizzare il tool Vader per poter ottenere i valori relativi a quanto quel commento possa essere categorizzato come "positivo", "neutro" o "negativo". Dopo aver calcolato i punteggi essi verranno scritti nel file CSV aperto in precedenza sotto nuove colonne **"Positive"** per il valore positivo, **"Negative"** per il valore negativo e **"Neutral"** per il valore neutrale. Inoltre, verranno cancellate le righe contenenti valori calcolati in maniera errata da Vader, ovvero contenenti 1.0 nella colonna "Neutral" e 0.0 negli altri campi. Questo calcolo errato è dovuto al fatto che alcuni commenti hanno contenuto non facilmente interpretabile dal tool, ad esempio non sono in lingua inglese. Il metodo descritto è mostrato nel seguente frammento di codice:

```python
# Function to calculate vader score and add it to the .csv
def calculate_vader_score(username, category):
    analyser = SentimentIntensityAnalyzer()
    vader_score = []
    df = pd.read_csv("./" + category + "/" + username + "_replies.csv")

    # Calculating vader score for each reply
    for i in range(df['Reply'].shape[0]):
        # print(analyser.polarity_scores(sentiments_pd['text'][i]))
        pos = analyser.polarity_scores(df['Reply'][i])["pos"]
        neu = analyser.polarity_scores(df['Reply'][i])["neu"]
        neg = analyser.polarity_scores(df['Reply'][i])["neg"]

        vader_score.append({"Positive": round(pos, 2),
                            "Negative": round(neg, 2),
                            "Neutral": round(neu, 2)
                            })

    # Adding vader scores as columns to .csv file
    sentiments_score = pd.DataFrame.from_dict(vader_score)
    df = df.join(sentiments_score)

    # Dropping rows that aren't calculated well by vader
    for i, row in df.iterrows():
        if row["Neutral"] == 1.0:
            df.drop(i, inplace=True)
    # Adding an index column
    id = range(1, len(df) + 1)
    df.insert(0, "ID", id)

    df.to_csv("./" + category + "/" + username + "_replies.csv", sep=',', index=False)
```

Il metodo **calculate_hatesonar_score()** prende i parametri “username” e “category” ed opera esattamente come il metodo calculate_vader_score(), la differenza sostanziale è nel come viene calcolato il punteggio. Infatti, come si può intuire dal nome questa volta la colonna relativa ai commenti verrà analizzata con il tool “Hate Sonar” anziché con “Vader”. Si otterranno tre informazioni che verranno anch’esse scritte in aggiunta al CSV precedentemente aperto: **“Hate”** che indica la valutazione del commento relativa all’hate speech presente, **“Offensive”** per i commenti offensivi ma non propriamente accostabili all’hate speech e **“Neither”** se il commento non è attribuibile a nessuno delle due categorie. Il seguente screen mostra la sua implementazione in dettaglio, come si può notare non è stato necessario alcuna ulteriore cancellazione delle colonne perché hate sonar è stato eseguito sequenzialmente a Vader il quale si era già occupato di eliminare i replies non valutabili.

```python
# Function to calculate hate speech score and append to the .csv
def calculate_hatespeech_score(username, category):
    sonar = Sonar()
    hatespeech_score = []

    df = pd.read_csv("./" + category + "/" + username + "_replies.csv")
    for i in range(df['Reply'].shape[0]):
        score = sonar.ping(df['Reply'][i])
        hate = round(score["classes"][0]["confidence"],2)
        offensive = round(score["classes"][1]["confidence"],2)
        neither = round(score["classes"][2]["confidence"],2)

        hatespeech_score.append({
            "Hate": hate,
            "Offensive": offensive,
            "Neither": neither
        })

    hate_score = pd.DataFrame.from_dict(hatespeech_score)
    df = df.join(hate_score)

    df.to_csv("./" + category + "/" + username + "_replies.csv", sep=',', index=False)
```

Dopo aver eseguito entrambi i metodi si otterrà quindi un nuovo file CSV contenente 6 nuovi attributi necessari per il nostro confronto finale tra i tool, di seguito è mostrata una riga d'esempio del CSV finale.

```
ID,Username,Data,Reply,Positive,Negative,Neutral,Hate,Offensive,Neither
1,channingtatum,Sat Nov 28 14:23:09 +0000 2020,' compassion. smiling with you. ',0.6,0.0,0.4,0.04,0.4,0.55
```

3.2.3 *Calcolo media rate*

La fase finale dello script *extract.py* si occupa di creare un nuovo file CSV denominato "*username_rates.csv*" contenente la media delle valutazioni di tutte le righe per ogni singolo campo e per ogni utente. Così facendo si avrà un file formato nel seguente modo:

Username	Category	Positive Rate	Negative Rate	Neutral Rate	Hate Rate	Offensive rate	Neither rate
KevinHart4real	Attori	19.28	12.99	67.8	7.1	45.48	47.37

Ad ogni nuova esecuzione dello script extract.py verrà aggiunta una nuova riga a questo file CSV contenente la media delle nuove valutazioni relative allo stesso utente. Infatti, per i nostri esperimenti è stata effettuata più volte l'esecuzione affinché si potessero avere dei dati validi.

Il codice della funzione è riportato di seguito:

```python
# Function that creates a .csv file containing for each user the average scores obtained
def create_score_csv(username, category):
    df = pd.read_csv("./"+category+"/"+username+"_replies.csv")
    pos_sum = 0
    neg_sum = 0
    neut_sum = 0
    hatespeech_sum = 0
    offensive_sum = 0
    neither_sum = 0
    rates = []

    for i,row in df.iterrows():...

    pos_sum = str(round((pos_sum / len(df.index))*100,2))
    neg_sum = str(round((neg_sum / len(df.index))*100,2))
    neut_sum = str(round((neut_sum / len(df.index))*100,2))
    hatespeech_sum = str(round((hatespeech_sum / len(df.index))*100,2))
    offensive_sum = str(round((offensive_sum / len(df.index))*100,2))
    neither_sum = str(round((neither_sum / len(df.index))*100,2))

    rateFile = "./"+category+"/"+username+"_rates.csv"

    if(os.path.isfile(rateFile)):
        rates.append([username, category, pos_sum, neg_sum, neut_sum, hatespeech_sum, offensive_sum, neither_sum])
        with open(rateFile, 'a', newline='') as file:
            writer = csv.writer(file, delimiter=',')
            writer.writerows(rates)
    else:
        rates = [["Username","Category","Positive Rate","Negative Rate","Neutral Rate","Hate Rate","Offensive rate", "Neither rate"]]
        rates.append([username, category, pos_sum, neg_sum, neut_sum, hatespeech_sum, offensive_sum, neither_sum])
        with open(rateFile, 'w', newline='') as file:
            writer = csv.writer(file, delimiter=',')
            writer.writerows(rates)
```

Si può quindi notare che vengono prese tutte le righe del file "*username_replies.csv*" ma unicamente i campi aggiunti in precedenza ad esso, ovvero quelli inerenti alle valutazioni dei tool. A questo punto, viene effettuata la media su ciascun campo così da poter riscrivere la nuova riga nel nuovo file "*username_rates.csv*". Questo file è assolutamente necessario per l'utilizzo del prossimo script in quanto sarà questo ad essere utilizzato come base di partenza per poter costruire i grafici.

3.3 Script "make_graph.py"

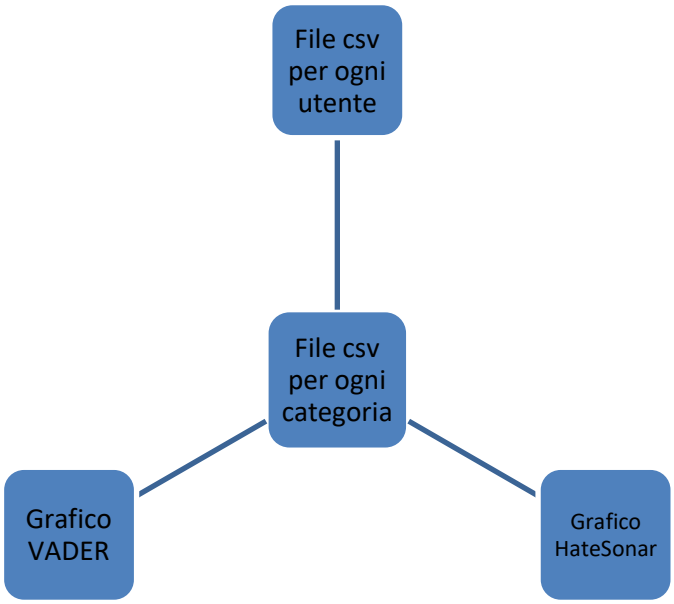
Successivamente all'ottenimento e analisi dei risultati con i tool precedentemente descritti, *make_graph.py* è lo script che si occupa di formalizzare i risultati ottenuti e quindi produrre delle rappresentazioni grafiche chiare affinché esaltino la problematica presa in considerazione in questo progetto. Quest'ultimo racchiude in sole tre funzioni tutto ciò che è possibile ottenere a livello grafico. Gli schemi che verranno prodotti si baseranno, per ovvie ragioni, sui file CSV ottenuti in precedenza.

```python
#Calculate average for each user of the category taken by argument
def user_average(category):...


#Save vader graph of the category taken by input
def vader_graph(category):...


#Save sonar graph of the category taken by input
def sonar_graph(category):...
```

Lo script opera nelle seguenti modalità:

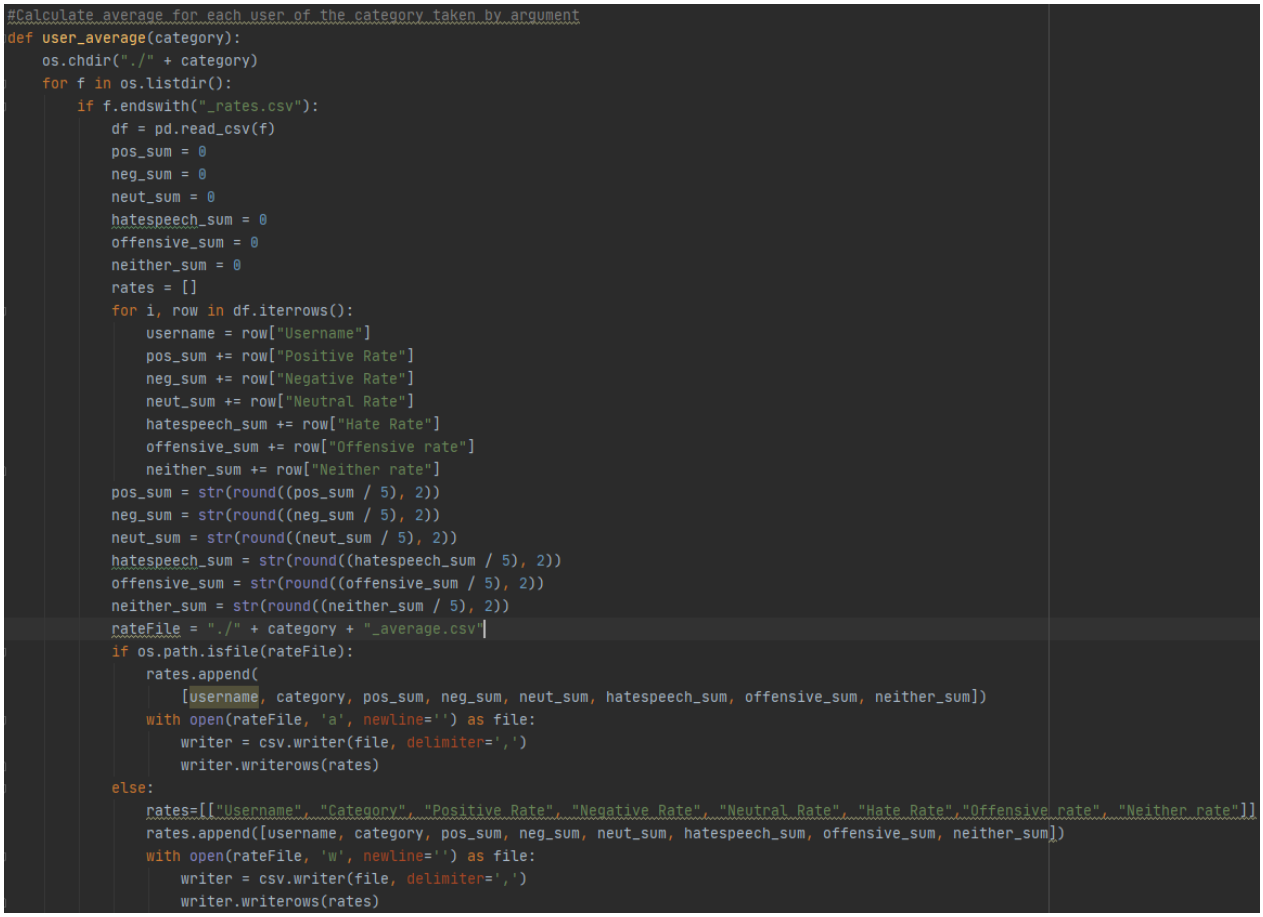
1. La funzione ***user_average()*** si preoccupa di produrre un singolo CSV per tutti gli utenti della categoria presa in questione, sia per quanto riguarda i dati prodotti da VADER e sia quelli da Hate Sonar.
2. La funzione ***vader_graph()***, ovviamente scende nello specifico andando a produrre un grafico che racchiude tutti i profili analizzati per una data categoria, al fine di poter effettuare un confronto tra categorie. Il grafico prodotto è inerente alle valutazioni VADER.
3. La funzione ***sonar_graph()*** opera analogamente alla precedente ma effettua, ovviamente, una produzione grafica dei valori inerenti prodotti da Hate Sonar.

Il funzionamento di tale script è descritto dal seguente schema:



3.3.1 *Visualizzazione dei risultati per categoria*

Preliminarmente si è pensato ad una creazione, in termini grafici, dei risultati prodotti dall'analisi dei tweet per ogni utente preso in considerazione in precedenza. Quindi è stato realizzato una funzione ***user_average()*** la quale viene costruita nel seguente modo:



```
#Calculate average for each user of the category taken by argument
def user_average(category):
    os.chdir("./" + category)
    for f in os.listdir():
        if f.endswith("_rates.csv"):
            df = pd.read_csv(f)
            pos_sum = 0
            neg_sum = 0
            neut_sum = 0
            hatespeech_sum = 0
            offensive_sum = 0
            neither_sum = 0
            rates = []
            for i, row in df.iterrows():
                username = row["Username"]
                pos_sum += row["Positive Rate"]
                neg_sum += row["Negative Rate"]
                neut_sum += row["Neutral Rate"]
                hatespeech_sum += row["Hate Rate"]
                offensive_sum += row["Offensive rate"]
                neither_sum += row["Neither rate"]
            pos_sum = str(round((pos_sum / 5), 2))
            neg_sum = str(round((neg_sum / 5), 2))
            neut_sum = str(round((neut_sum / 5), 2))
            hatespeech_sum = str(round((hatespeech_sum / 5), 2))
            offensive_sum = str(round((offensive_sum / 5), 2))
            neither_sum = str(round((neither_sum / 5), 2))
            rateFile = "./" + category + "_average.csv"
            if os.path.isfile(rateFile):
                rates.append(
                    [username, category, pos_sum, neg_sum, neut_sum, hatespeech_sum, offensive_sum, neither_sum])
                with open(rateFile, 'a', newline='') as file:
                    writer = csv.writer(file, delimiter=',')
                    writer.writerows(rates)
            else:
                rates=[["Username", "Category", "Positive Rate", "Negative Rate", "Neutral Rate", "Hate Rate","Offensive rate", "Neither rate"]]
                rates.append([username, category, pos_sum, neg_sum, neut_sum, hatespeech_sum, offensive_sum, neither_sum])
                with open(rateFile, 'w', newline='') as file:
                    writer = csv.writer(file, delimiter=',')
                    writer.writerows(rates)
```

La funzione si preoccuperà di aprire tutte le directory create in precedenza con lo scopo di catturare tutti gli script "*rates*" prodotti nella fase di cattura e analisi. La funzione andrà ad aprire ogni file CSV contenente tutti i dati relativi a VADER ed Hate Sonar andandoli a salvare ed effettuare le normali operazioni di media, ovviamente siccome gli esperimenti sono stati condotti su numerose ripetizioni, si andrà a calcolare una media in percentuale, arrotondata al secondo decimale, per non andare incontro a problemi di diversificazione dei risultati. La funzione ovviamente andrà a creare un nuovo file CSV, se non esiste, col nome della categoria, ad esempio *Politici_average.csv* al fine di poter compattare queste media in una sola tabella lineare. Se il file esiste i valori verranno aggiunti ad esso onde evitare ridondanze.

Username	Category	Positive Rate	Negative Rate	Neutral Rate	Hate Rate	Offensive rate	Neither rate
barackobama	Politici	18.03	8.52	73.48	7.59	32.74	59.56
borisjohnson	Politici	11.64	10.16	78.23	5.35	35.1	59.62
GretaThunberg	Politici	14.65	12.38	72.99	5.21	38.01	56.78
joebiden	Politici	18.14	7.45	74.38	5.57	34.02	60.46
MichelleObama	Politici	26.31	5.87	67.83	11.2	35.31	53.45
realDonaldTrump	Politici	14.72	14.43	70.87	6.41	35.81	57.85

Infatti, da come si evince, vi sono i vari profili, analizzati per una data categoria, con i valori discussi in fase di analisi. Questo file prodotto sarà poi il file CSV su cui poi si baserà la creazione dei grafici inerenti alla categoria in modo più generico, che verrà discussa nel prossimo sottoparagrafo.

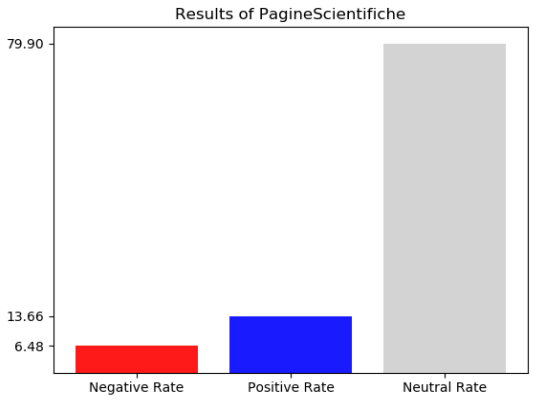
3.3.2 Visualizzazione dei risultati per utenti

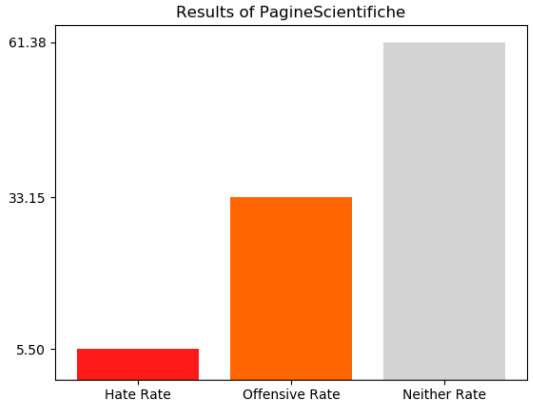
L'idea alla base della visualizzazione è stata quella di poter fornire all'end-user la possibilità di effettuare più confronti, partendo dal presupposto che vi sono alcune categorie più colpite di altre. Ragion per cui i grafici prodotti verranno salvati separatamente con questa dicitura "category_sonarGraph.png" oppure "category_vaderGraph.png" a seconda di quale script venga eseguito.

```python
#Save vader graph of the category taken by input
def vader_graph(category):
    os.chdir("./" + category)
    for f in os.listdir():
        if f.endswith("_average.csv"):
            df = pd.read_csv(f)
            positive_rate = df["Positive Rate"]
            negative_rate = df["Negative Rate"]
            neutral_rate = df["Neutral Rate"]
            plt.title("Results of " + category)
            rate = ["Negative Rate", "Positive Rate", "Neutral Rate"]
            positives = 0
            negatives = 0
            neutrals = 0
            for positive in positive_rate:
                positives += positive
            positives = positives/len(positive_rate)
            for negative in negative_rate:
                negatives += negative
            negatives = negatives/len(negative_rate)
            for neutral in neutral_rate:
                neutrals += neutral
            neutrals = neutrals / len(neutral_rate)
            results = [negatives, positives, neutrals]
            plt.yticks(results)
            plt.bar(rate, results,color=['#ff1a1a','#1a1aff','lightgrey'])
            plt.show(block=False)
            plt.pause(2)
            plt.savefig("../" + category + "/" + category + "_vaderGraph.png")
            plt.close()
```

```python
#Save sonar graph of the category taken by input
def sonar_graph(category):
    os.chdir("./" +category)
    for f in os.listdir():
        if f.endswith("_average.csv"):
            df = pd.read_csv(f)
            hate_rate = df["Hate Rate"]
            offensive_rate = df["Offensive rate"]
            neither_rate = df["Neither rate"]
            plt.title("Results of " + category)
            rate = ["Hate Rate", "Offensive Rate", "Neither Rate"]
            hates = 0
            offensives = 0
            neithers = 0
            for hate in hate_rate:
                hates += hate
            hates = hates/len(hate_rate)
            for offensive in offensive_rate:
                offensives += offensive
            offensives = offensives/len(offensive_rate)
            for neither in neither_rate:
                neithers += neither
            neithers = neithers/len(neither_rate)
            results = [hates, offensives, neithers]
            plt.yticks(results)
            plt.bar(rate, results,color=['#ff1a1a','#ff6600','lightgrey'])
            plt.show(block=False)
            plt.pause(2)
            plt.savefig("../"+category+"/"+category+"_sonarGraph.png")
            plt.close()
```

Come è possibile osservare, i due script lavorano in modalità molto simile, andando a selezionare volta per volta la directory rappresentativa della categoria presa in esame. Dopo aver individuato tale directory, viene catturato il contenuto del file CSV precedentemente calcolato, effettuata la media tra i vari utenti e creato un grafo avente sull'asse delle ascisse i valori inerenti al tool VADER che corrispondono a: Negative rate, Positive rate e Neutral Rate e sull'asse delle ordinate i valori corrispondenti. Oppure, allo stesso modo, sull'asse delle ascisse i valori inerenti ad Hate Sonar che vengono riconosciuti da: Hate Rate, Offensive Rate e Neither Rate.





I grafici mostrati, sono solo alcuni di quelli che vengono creati dal tool sviluppato, infatti tutti gli esperimenti verranno mostrati nel capitolo successivo. Come già anticipato per dare un risalto maggiore ai dati, si è scelto un colore grigio per quanto riguarda i Neutral Rate e Neither Rate, cercando di far risaltare con colori più accesi le valutazioni inerenti agli altri valori.

4. Esperimenti e analisi dei risultati

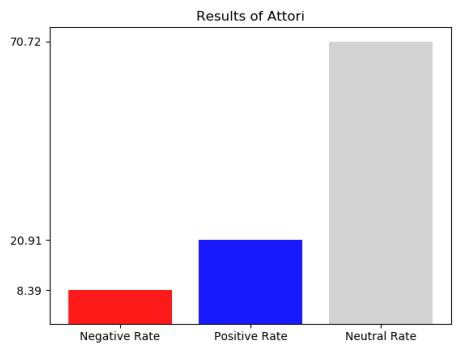
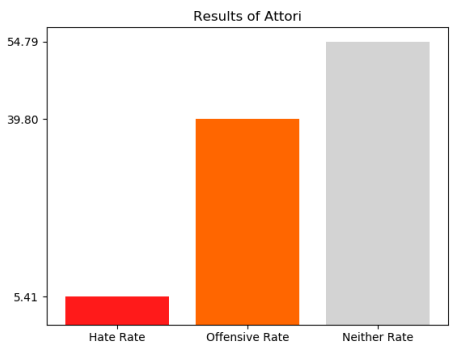
In questo paragrafo analizzeremo i grafici che sono stati elaborati da *make_graph()* e metteremo a confronto i risultati ricavati da ogni categoria. I dati sono stati ottenuti dall'esecuzione della funzione *create_score_csv()* per ogni personaggio presente in quella determinata categoria; l'esecuzione di tale funzione è stata effettuata cinque volte per ottenere una maggiore accuratezza dei dati e una media precisa. Dopo la prima esecuzione della funzione, che prende in input l'username Twitter del personaggio e la categoria di appartenenza, viene creato un file CSV per ogni personaggio con la percentuale di:

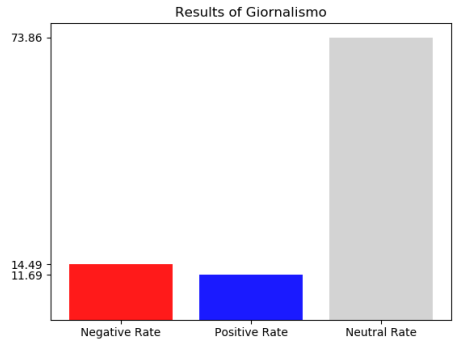
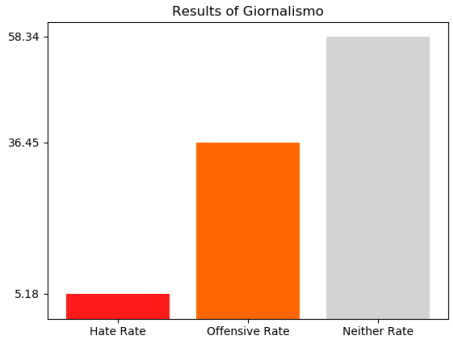
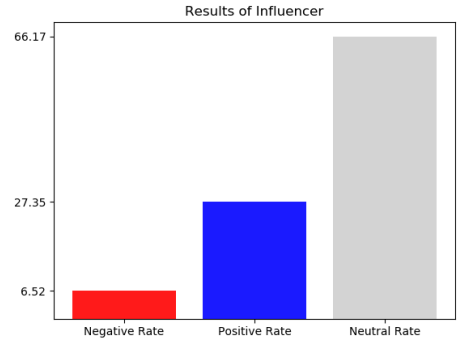
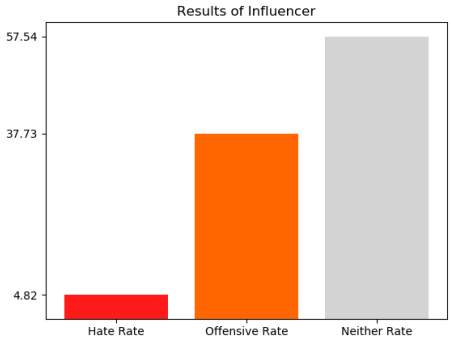
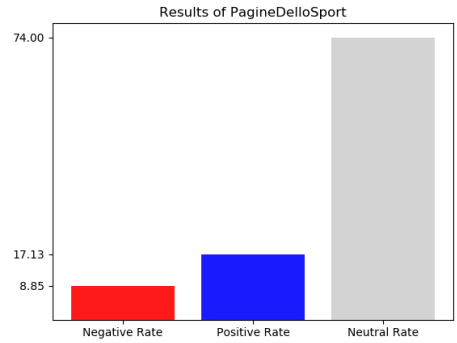
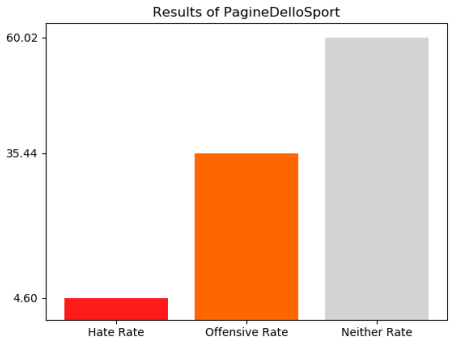
- Positive Rate
- Negative Rate
- Neutral Rate
- Hate Rate
- Offensive Rate
- Neither Rate

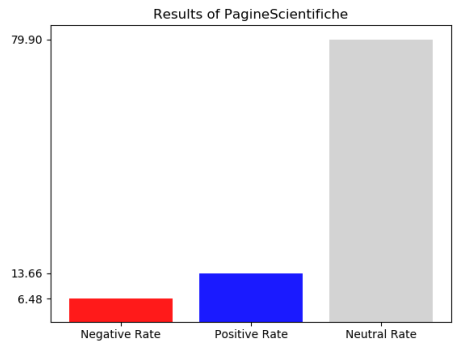
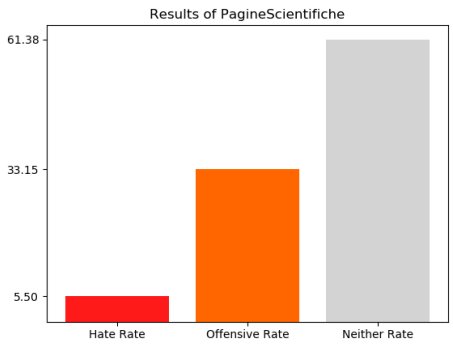
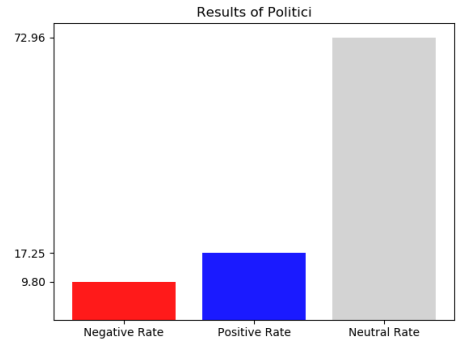
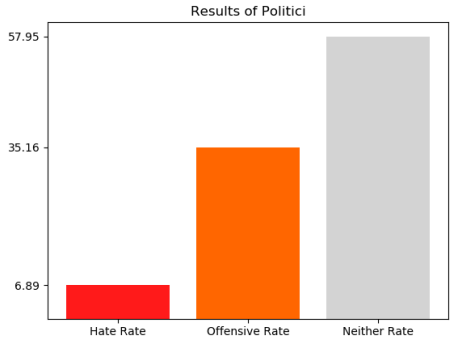
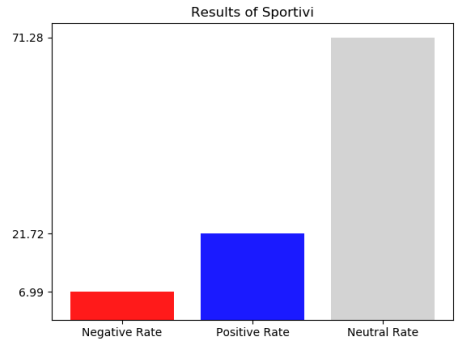
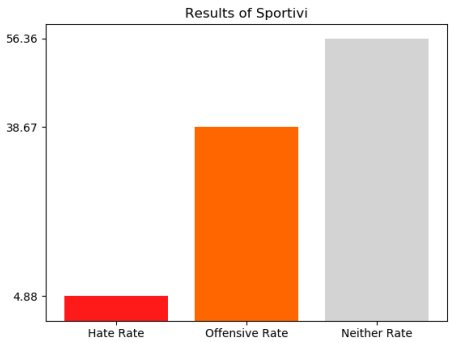
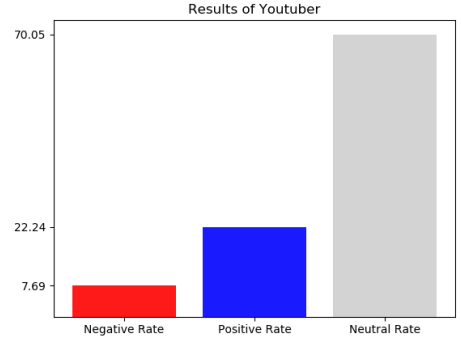
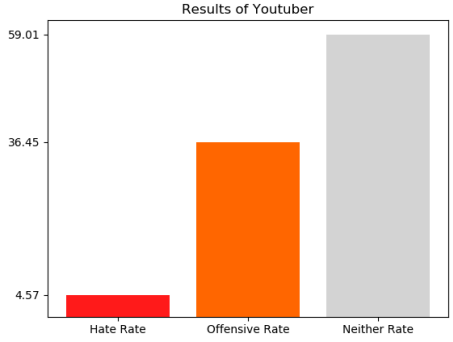
Dopo cinque esecuzioni consecutive il file CSV conteneva i dati relativi ai campi sopra citati da cui bisognava ottenere la media. La funzione *user_average()* che prende in input la categoria cui il personaggio prende parte, calcolava per ogni file ottenuto tramite la funzione create_score_csv() la media aritmetica. Con il file di output ottenuto generiamo i grafici relativi ai tool VADER ed Hate Sonar che analizzeremo e confronteremo nel seguente sottoparagrafo.

4.1 Esperimenti ed analisi per categoria

In questo sottoparagrafo vengono mostrati in formato tabellare tutti i grafici che vengono creati utilizzando lo script *make_graph()*. La tabella seguente mostra quindi tutti i grafici relativi all'analisi delle categorie effettuata con i due tools utilizzati.

Categoria	**VaderGraph**	**HateSonarGraph**
Attori		

Giornalismo		
Influencer		
Pagine dello sport		

Pagine scientifiche		
Politici		
Sportivi		
Youtuber		

Partendo da un'analisi sui grafici in questione, focalizzando l'attenzione primariamente su quelli prodotti da VADER si possono trarre varie conclusioni:

1. La categoria che mostra il **maggior numero di commenti negativi**, risulta essere il *Giornalismo*, con il 14.49%.
2. La categoria, invece, che mostra il **minor numero di commenti negativi** è quella delle P*agine scientifiche* con il 6.48%.

Le altre categorie invece presentano valori che sono compresi tra questi due estremi. Sommariamente la presenza di questi due intervalli fa notare una percentuale rilevante di tweet negativi in tutte le categorie analizzate, quindi un primo segnale di possibile fenomeno di hate speech. Ora si prendono in considerazione le percentuali inerenti ai commenti positivi, infatti:

1. La categoria che mostra il **maggior numero di commenti positivi** risulta essere quella che riguarda gli *Influencer*, con il 27.35%.
2. La categoria che possiede il **minor numero di commenti positivi** la si riscontra in *Giornalismo* con il 11.69%.

In questo caso le percentuali di commenti positivi sono sensibilmente più alte. L'analisi ha mostrato che vi sono molti commenti che non inneggiano alla negatività, ma soprattutto la categoria del *Giornalismo* risulta essere quella cardine avendo molti commenti negativi e meno commenti positivi rispetto alle altre categorie prese in esame. Da ciò si può constatare che i tweet postati dalle varie testate giornalistiche trovano molti contrasti tra i commenti degli utenti, i quali portano a questi risultati appena descritti. Inoltre, la categoria degli *Influencer* presenta il tasso più alto, in percentuale, di commenti positivi; ciò fa pensare a tanti replies relativi ai fan di quest'ultimi. Ed infine ciò che riguarda le *pagine scientifiche*; in esse si può riscontrare il tasso minore di tutte le altre categorie per quanto riguarda i commenti negativi, ciò fa dedurre che gli utenti in questo ramo sono molto volti ad un utilizzo corretto rispetto alla semplice critica negativa di ciò che viene condiviso.

Riassumendo viene mostrata una tabella con tutti i valori delle categorie per VADER.

Categoria	**Negative rate**	**Positive Rate**
Attori	8.39%	20.91%
Giornalismo	**14.49%**	**11.69%**
Influencer	6.52%	**27.35%**
Pagine dello sport	8.85%	17.13%

Pagine scientifiche	**6.48%**	13.66%
Politici	9.80%	17.25%
Sportivi	6.99%	21.72%
Youtuber	7.69%	22.24%

Avendo riportato i risultati ottenuti dall'analisi con il tool VADER, si passa ora all'analisi attraverso il tool Hate Sonar, il quale come ampiamente descritto ha valutato i replies, raccogliendoli secondo i criteri: Hate Speech, Offensive Language e Neither Rate. Con questo strumento è stato possibile fare un'analisi ancor più specifica sul linguaggio utilizzato nei replies, affinché sia possibile ricercare dove risalta il fenomeno dell'Hate Speech. Infatti, si può evincere che:

1. La categoria che mostra la **maggior percentuale inerente all'Hate Speech** si è rivelata esser quella dei *Politici*.
2. Viceversa, la categoria che mostra la **minor percentuale di Hate Speech** è quella dei *Youtuber*.

Ciò fa intendere che una categoria come quella dei politici, soprattutto i profili presi in causa per l'analisi del progetto, è molto colpita dal fenomeno dell'Hate Speech. Infatti, sono documentati tanti casi simili sia per quanto riguarda i replies contro questa specifica categoria di utenti, ma soprattutto perché alcuni di questi utenti sono implicitamente a favore di determinate ideologie pro odio. Passando all'altro parametro di analisi, ovvero quello riguardante l'Offensive Rate, si nota che:

1. La categoria degli *Attori* è quella che presenta la più **alta percentuale in termini di commenti offensivi**.
2. Invece, per quanto riguarda la **percentuale più bassa di commenti offensivi** la si riscontra nella categoria delle *Pagine scientifiche*.

Per ciò che riguarda l'Offensive rate si può intuire che questo fenomeno è diffuso praticamente ovunque, infatti i dati risultanti fanno emergere delle statistiche piuttosto omogenee in questo senso. Non è da escludere che il fenomeno dell'Hate Speech sia annidato anche nei commenti non valutati come tali, in quanto nascosto dall'ironia e/o sarcasmo dei commenti.

Categoria	**Hate Rate**	**Offensive Rate**
Attori	5.41%	**39.80%**
Giornalismo	5.18%	36.45%
Influencer	4.82%	37.73%
Pagine dello sport	4.60%	35.44%
Pagine scientifiche	5.50%	**33.15%**
Politici	**6.89%**	35.16%
Sportivi	4.88%	38.67%
Youtuber	**4.57%**	36.45%

Riassumendo viene mostrata una tabella con i valori delle categorie per Hate Sonar.

5. Conclusioni

In conclusione, il progetto ha raggiunto gli scopi prefissati. Infatti, sono stati applicati i due tool VADER ed Hate Sonar, non solo per confrontarli tra loro ma anche per utilizzarli in maniera complementare così da avere dei risultati quanto più precisi possibile.

I risultati ottenuti rispecchiano quanto si era previsto, le pagine a carattere scientifico/culturale sono quelle con il minor numero di commenti negativi/hate speech e questo è chiaramente dovuto dalla tipologia di utenti che le frequentano.

Un fattore da tenere in considerazione è quello temporale, ovvero il risultato di quanto ottenuto dall'analisi può variare a seconda del periodo storico. In questo caso le pagine dei politici sono risultate particolarmente frequentate da diverse tipologie di utenti proprio perché vi era il periodo delle elezioni negli Stati Uniti. Infatti, utenti di fazioni opposte commentavano post non di loro interesse unicamente per alimentare odio.

Un ulteriore fattore è quello della lingua: entrambi gli strumenti operano su commenti in lingua inglese, per cui una possibile evoluzione dello studio potrebbe basarsi sull'implementazione di tool che possano analizzare anche commenti in altre lingue. Questa possibilità garantirebbe infatti di mostrare a livello geografico dove sia più diffuso il fenomeno dell'Hate Speech. Di fatti analizzare unicamente commenti in lingua inglese ci ha limitati nello scegliere pagine e utenti accomunati dalle stesse aree linguistiche quali USA e Regno Unito.

Come detto precedentemente nell'introduzione, i nostri dati potranno essere utilizzati da chi di dovere per poter arginare il fenomeno nelle community dove è più presente; ad esempio, mediante la specifica di nuove restrizioni sui commenti. Il tool può essere anche riutilizzato per compiere nuovi studi su diverse categorie o ampliare quanto ottenuto su quelle già stabilite.

Infine, si può affermare che nonostante il problema dell'Hate Speech cresca proporzionalmente all'utilizzo dei social e che quindi sia destinato ad aumentare, grazie ad analisi come quelle effettuate in questo progetto si potrà eliminare o quantomeno limitare il più possibile, attraverso apposite tecniche, la diffusione di questo fenomeno.

Bibliografia

[1] Utenti social network: tutte le statistiche 2020 - https://cultadv.com/utenti-social-network-2020

[2] Leading countries based on numbers of Twitter users as of October 2020 (in millions) - https://www.statista.com/statistics/242606/number-of-active-twitter-users-in-selected-countries/

[3] Anaconda - https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)

[4] Tweepy - http://docs.tweepy.org/en/latest/

[5] Twarc - https://www.github.com/DocNow/twarc/

[6] VADER - https://github.com/cjhutto/vaderSentiment#citation-information/

[7] Hate Sonar - https://github.com/Hironsan/HateSonar

[8] Thomas Davidson, Dana Warmsley, Michael Macy, Ingmar Weber, "Automated Hate Speech Detection and the Problem of Offensive Language" - https://arxiv.org/pdf/1703.04009.pdf

[9] hate-speech-and-offensive-language - https://www.github.com/t-davidson/hate-speech-and-offensive-language