UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II

Scuola Politecnica e delle Scienze di Base

Corso di Laurea Magistrale in Ingegneria Informatica

Social Network Analysis – Big Data Engineering

Anno Accademico 2022-2023

Riccio Emanuele – M63001339

Tammaro Ferdinando – M63001380

Prof. Moscato Vincenzo

Sommario

Introduzione

Come Progetto finale per il corso di Big Data Engineering, si è effettuata un'analisi dei sentimenti partendo da alcuni tweet scritti sulla piattaforma Twitter nel periodo delle elezioni americane del 2020.

Lo scopo del lavoro era di poter determinare, preso un qualsiasi utente, quali fossero i suoi interessi principali, il suo allineamento politico generale, i sentimenti espressi tramite i suoi tweet e se fosse un *mobilitatore passivo* o un *mobilitatore attivo* di video di Youtube, ovvero se li condividesse attivamente o soltanto tramite retweet. Per effettuare ciò, si è dovuti ricorrere a tecniche di gestione di Big Data, come il database Spark, tecniche di Natural Language Processing, in particolare la libreria python TweetNLP per effettuare sentiment analysis, e tecniche di Machine Learning per determinare l'allineamento politico dell'utente.

Dataset

Il dataset utilizzato è stato il dataset *YoutubeMobilizers,* fornito durante il corso, che era così strutturato:

```
root
 |-- tweetid: long (nullable = true)
 |-- userid: long (nullable = true)
 |-- screen_name: string (nullable = true)
 |-- date: string (nullable = true)
 |-- lang: string (nullable = true)
 |-- description: string (nullable = true)
 |-- text: string (nullable = true)
 |-- reply_userid: long (nullable = true)
 |-- reply_screen: string (nullable = true)
 |-- reply_statusid: double (nullable = true)
 |-- tweet_type: string (nullable = true)
 |-- friends_count: integer (nullable = true)
 |-- listed_count: integer (nullable = true)
 |-- followers_count: integer (nullable = true)
 |-- favourites_count: integer (nullable = true)
 |-- statuses_count: integer (nullable = true)
 |-- verified: boolean (nullable = true)
 |-- hashtag: string (nullable = true)
 |-- urls_list: string (nullable = true)
 |-- profile_pic_url: string (nullable = true)
 |-- profile_banner_url: string (nullable = true)
 |-- display_name: string (nullable = true)
 |-- date_first_tweet: string (nullable = true)
 |-- account_creation_date: string (nullable = true)
 |-- rt_urls_list: string (nullable = true)
 |-- mentionid: string (nullable = true)
 |-- mentionsn: string (nullable = true)
 |-- rt_screen: string (nullable = true)
 |-- rt_userid: long (nullable = true)
 |-- rt_user_description: string (nullable = true)
 |-- rt_text: string (nullable = true)
 |-- rt_hashtag: string (nullable = true)
 |-- rt_qtd_count: integer (nullable = true)
 |-- rt_rt_count: double (nullable = true)
 |-- rt_reply_count: double (nullable = true)
 |-- rt_fav_count: double (nullable = true)
 |-- rt_tweetid: long (nullable = true)
 |-- qtd_screen: string (nullable = true)
 |-- qtd_userid: long (nullable = true)
 |-- qtd_user_description: string (nullable = true)
 |-- qtd_text: string (nullable = true)
 |-- qtd_hashtag: string (nullable = true)
 |-- qtd_qtd_count: integer (nullable = true)
 |-- qtd_rt_count: double (nullable = true)
 |-- qtd_reply_count: double (nullable = true)
 |-- qtd_fav_count: double (nullable = true)
 |-- qtd_tweetid: long (nullable = true)
 |-- qtd_urls_list: string (nullable = true)
 |-- qtd_location: string (nullable = true)
 |-- media_urls: string (nullable = true)
 |-- rt_media_urls: string (nullable = true)
 |-- q_media_urls: string (nullable = true)
```

Figura 1 - Schema del dataset

Si può notare che questo dataset è a tutti gli effetti separato in più sezioni:

- Metadati del tweet e dell'account
- Contenuto del tweet
- Tipo del tweet
- Eventuali dati di tweet correlati

Per le elaborazioni effettuate, è stato di cruciale importanza il campo tweet_type, che specificava appunto il tipo del tweet. Questo poteva essere di quattro tipi diversi:

- Original, cioè un tweet originale dell'account in esame
- Reply, una risposta ad un altro tweet
- Retweeted without comments, cioè una semplice condivisione di un tweet già esistente, senza alcuna aggiunta di contenuto
- Quoted tweet, dove viene condiviso un tweet già esistente andando ad aggiungere del nuovo contenuto

Su questo campo si basavano alcune query importanti, in particolare riguardanti l'analisi degli *Youtube Mobilizers,* inoltre in base al suo contenuto cambiavano completamente i campi del dataset da analizzare: infatti se un tweet non è di tipo "quoted_tweet", tutti i campi relativi al tweet citato erano vuoti; e così anche per gli altri tipi di tweet.

In totale, il dataset era composto di sette file in formato .csv, ognuno composto da centinaia di migliaia di tweet creati nei mesi da giugno 2020 a dicembre 2020, per un totale di circa 10.7 GB di dati. Oltre a questi, era presente un ulteriore file chiamato userids.csv, che conteneva tutti gli username degli utenti i cui tweet erano presenti nel dataset utilizzato, e per ogni utente era segnato se questi fossero stati moderati o meno.

Research Questions

Le domande che ci si è posti all'inizio di questo elaborato sono state:

- distribuzione dei tweet dell'utente in termini temporali, tipologia di tweet e lingua
- i domini dei link e gli hashtag maggiormente tweettati dall'utente
- i profili con cui l'utente ha maggiormente interagito tramite tweet
- sentimenti, topic ed emozioni espressi dall'utente tramite tweet
    - in relazione a tutti i tweet dell'utente
    - con un focus particolare per gli account con cui l'utente ha interagito di più, analizzando anche l'eventuale presenza di frasi offensive
- quali fossero i comportamenti e le emozioni dell'utente nei confronti dei video di youtube condivisi tramite tweet
    - percentuale di video youtube condivisi dall'utente e poi moderati dalla piattaforma
- si è cercato di capire dai tweet dell'utente la sua inclinazione politica

Elaborazione dati

Spark

Per gestire ed elaborare i dati si è fatto affidamento sul database Spark, utilizzato tramite Python sulla piattaforma Google Colab. Sebbene per la mole di dati in esame non fosse strettamente necessario utilizzare un tool così specifico per i Big Data, vi è stata una particolare attenzione verso la gestione dei dati, in modo che il progetto fosse facilmente scalabile anche per dataset decisamente più grandi.

Il dataset completo è stato quindi importato su spark con il seguente codice:

```
data_paths = ["/content/drive/MyDrive/DS/2020-6.csv","/content/drive/MyDrive/DS/2020-7.csv","/content/drive/MyDrive/DS/2020-8.csv","/content/drive/MyDrive/DS/2020-9.csv","/content/drive/MyDrive/DS/2020-10.csv","/content/drive/MyDrive/DS/2020-11.csv","/content/drive/MyDrive/DS/2020-12.csv",]

tweets = spark.read \
  .option("inferSchema", True) \
  .option("header", True) \
  .option("quote", "\"") \
  .option("escape", "\"") \
  .csv(data_paths, sep=',', multiLine=True)
```

```
userids = spark.read \
  .option("inferSchema", True) \
  .option("header", True) \
  .option("quote", "\"") \
  .option("escape", "\"") \
  .csv("/content/drive/MyDrive/DS/userids.csv", sep=',', multiLine=True)
```

Per questioni di complessità computazionale, ci si è limitati a studiare un utente alla volta, creando un nuovo dataframe dove fossero contenuti soltanto i tweet creati dalla persona selezionata; inoltre, essendo alcuni campi superflui rispetto alle analisi desiderate, ne sono stati mantenuti solamente alcuni, sempre per questioni di complessità computazionale.

Questo dataframe è stato creato in questo modo:

```
name1 = "JimBonz"
path_query = f"/content/drive/MyDrive/DS/Query/{name1}/"
attrib = [
"account_creation_date","date","description","followers_count","hashtag","lang
","mentionsn","profile_pic_url","qtd_screen","qtd_urls_list","reply_screen","r
t_screen","rt_urls_list","screen_name","text","tweet_type",
"tweetid","urls_list","verified"]
# Filtro i tweets, ottengo solo quelli appartenenti all'utente specificato
name1_dataframe = tweets.select(*[attrib]).where(tweets.screen_name == name1)
print(name1_dataframe.count())
```

TweetNLP

Per poter comprendere i sentimenti espressi da ogni utente nei propri tweet, è stata utilizzata una libreria python open source chiamata TweetNLP. La descrizione ufficiale del sito della libreria riporta che in essa è presente una collezione di tool per l'analisi e la comprensione dei tweet, come sentiment analysis, emoji prediction e name-entity recognition.

In particolare, nel caso in esame sono stati istanziati i seguenti modelli:

```
import tweetnlp
model_e = tweetnlp.load_model('emotion')
model_o = tweetnlp.load_model('offensive')
model_s = tweetnlp.load_model('sentiment')
model_i = tweetnlp.load_model('irony')
model_t = tweetnlp.load_model('topic_classification')
```

Keras

Per poter comprendere l'inclinazione politica dell'utente, si è dovuti ricorrere ad un modello di machine learning creato ex-novo per il progetto, andando a sfruttare un dataset disponibile liberamente sulla piattaforma Kaggle.

In particolare, è stata creata una semplice rete neurale con un solo layer che, prendendo in ingresso una versione vettorizzata del testo del tweet, restituisse in output un numero compreso nell'intervallo [0,1], dove 0 è l'etichetta assegnata ai tweet più *democratici* e 1 quella assegnata ai tweet più *repubblicani*.

	Party	Tweet
0	Republican	Today marks the final time that Americans will...
1	Democrat	Spent the morning with Ambassador Mohib and a ...
2	Republican	RT @WMALDC: Listen now as Congressman @RobWitt...
3	Democrat	MUST READ: 7 Gutsy Women To Know For Internati...
4	Democrat	We are proud of our diversity and immigrant co...

Figura 2 - Estratto dal dataset utilizzato per l'addestramento del modello Keras

Lo schema della rete neurale è il seguente:

```
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(2500,)))
model.add(Dense(1, activation='sigmoid'))
```

L'input è stato preprocessato con un oggetto TfidVectorizer, da esso sono stati rimossi tutti i segni di punteggiatura e altre stringhe che potevano causare problemi (come "https://").

Il modello addestrato è poi stato salvato in un file chiamato model.h5, per poterlo importare nel notebook con Spark in esecuzione. Invece per importare il vectorizer è stato necessario creare un file *pickle* contenente l'oggetto vectorizer e i dati su cui era stato precedentemente *fitted*, per poi rieffettuare l'operazione di *fitting* all'interno del nuovo notebook.

Streamlit

Una volta terminata l'elaborazione dei dati, questi sono stati raccolti e visualizzati tramite una dashboard realizzata con il framework Streamlit, tramite Python. Grazie ad esso, è stato possibile creare grafici ad alto impatto visivo per visualizzare l'output delle query eseguite.

La dashboard è interattiva: si può selezionare un utente di cui visualizzare i dati ed è possibile interagire con qualsiasi grafico presente in essa.

Per questioni di complessità computazionali, ci si è limitati ad importare i dati di un numero limitato di utenti.

Query effettuate

In questo paragrafo si osservano le query scritte su Spark e la corrispondente rappresentazione sulla dashboard. Si noti che tutte le immagini prese da Streamlit fanno riferimento all'utente @S_Devenish.

Query 0

Nella query 0 si effettua solamente un salvataggio dei dati dell'utente come si può vedere nel seguente codice:

```
#Query0: estrazioni dati utente base
from pyspark.sql.functions import col, lit

query0 = name1_dataframe.select('screen_name', "verified", "description",
"followers_count", "account_creation_date", "profile_pic_url")\
    .distinct()\
    .orderBy(desc('followers_count')) \
    .limit(1)

#Estraggo categoria utente
user_cat = userids.filter(col("user") == name1).collect()[0]['cat']

# Aggiungi la colonna cat
query0_with_cat = query0.withColumn('cat', lit(user_cat))
```

All'interno della dashboard di Streamlit, questi dati sono visualizzati con una scheda creata in html utilizzando questo codice:

Figura 3 - Rappresentazione Query 0

Query 1

Nella query 1 si inizia a rispondere alla prima domanda di ricerca: la visualizzazione della distribuzione temporale dei tweet dell'utente. Per fare ciò, partendo dal campo *data* sono stati creati i due campi day_of_week e hour, per poi eseguire un'operazione di groupBy e una di count:

```
1. query1 = name1_dataframe.select("date").withColumn('day_of_week',
   date_format('date', 'EEEE')) \
2.         .withColumn('hour', date_format('date',
   'HH')).groupBy("day_of_week","hour").count().sort("day_of_week","hour")
```

In Streamlit è stata visualizzata con un istogramma bidimensionale:

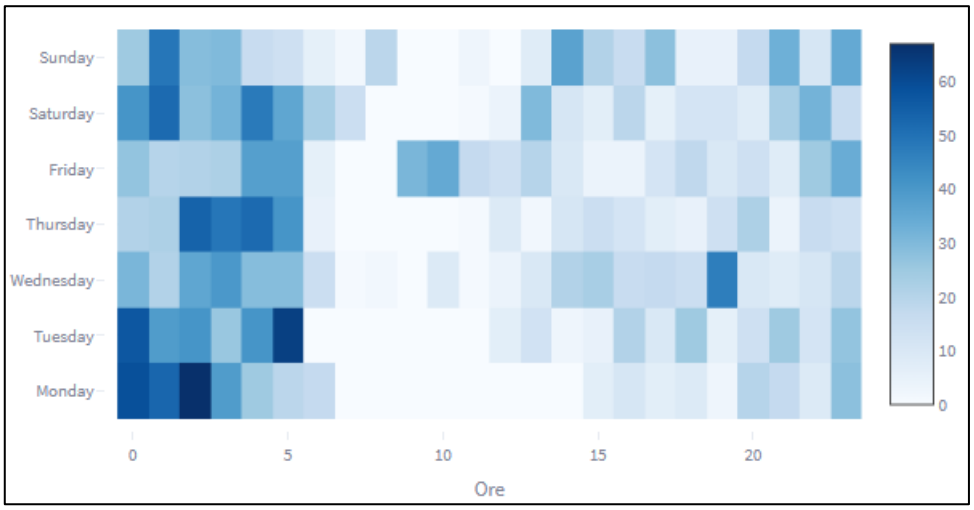

Figura 4 - Rappresentazione Query 1

Query 2

In maniera simile alla query precedente, nella query 2 si effettua un conteggio dei tweet distribuiti secondo i giorni della settimana:

```
1. query1 = name1_dataframe.select("date").withColumn('day_of_week',
   date_format('date', 'EEEE')) \
2.         .withColumn('hour', date_format('date',
   'HH')).groupBy("day_of_week","hour").count().sort("day_of_week","hour")
```

In Streamlit l'output della query è stato rappresentato con un istogramma:

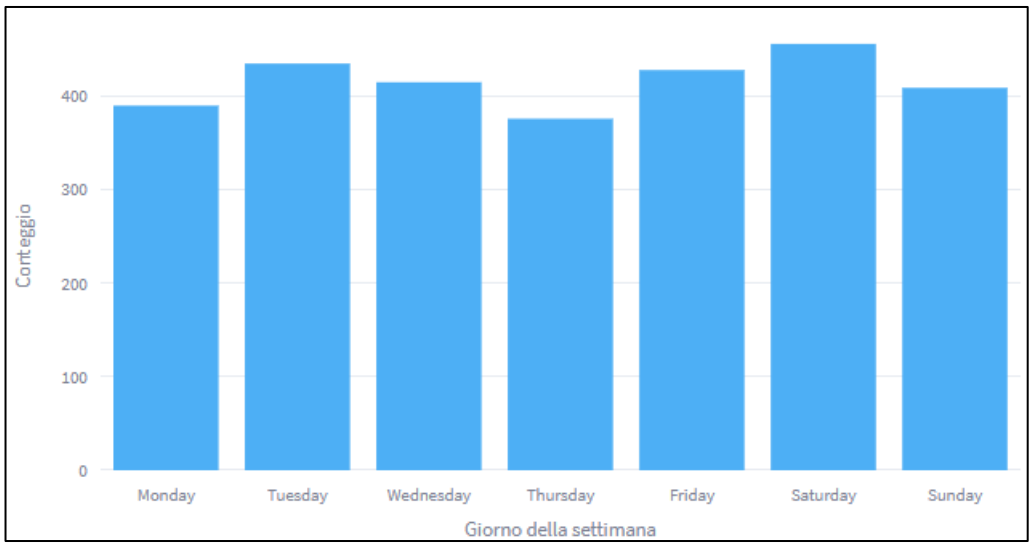

Figura 5 - Rappresentazione Query 2

Query 3

Proseguendo con la visualizzazione della distribuzione temporale dei tweet, nella query 3 si può vedere la distribuzione dei tweet raggruppati per mese:

```
1. query3 = name1_dataframe.select("date").withColumn('month',
   date_format('date', 'MMMM')).groupBy("month",).count().sort("month",)
```

Anche questa query è stata rappresentata con un istogramma in Streamlit:

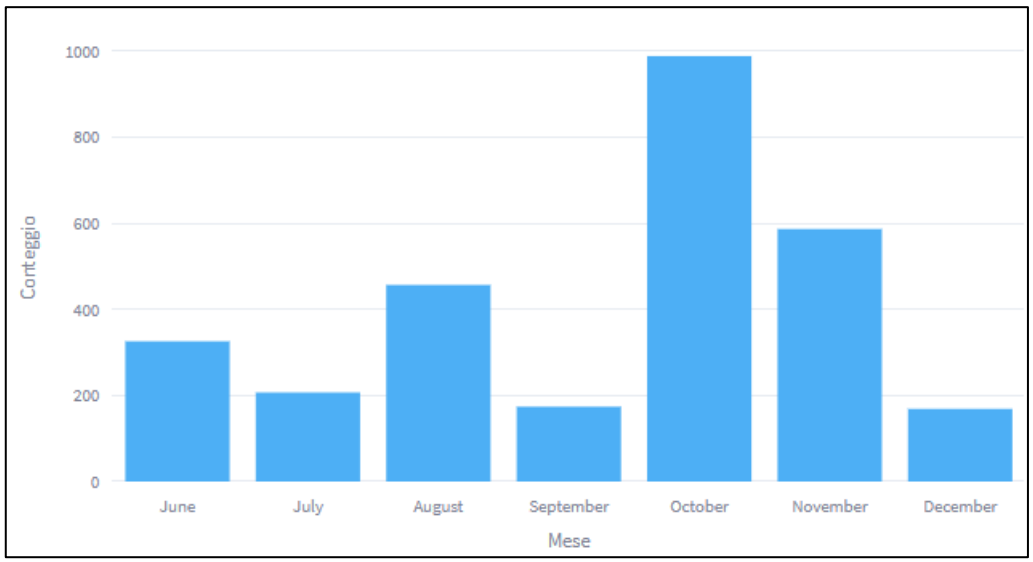

Figura 6 - Rappresentazione Query 3

Query 4

Nella query 4 si è voluto evidenziare la distribuzione dei tipi di tweet, come contenuto nel campo tweet_type. Per fare ciò è bastato eseguire un'operazione di groupBy su questo campo e un conteggio:

```
1. query4 =
   name1_dataframe.select("tweet_type").groupBy("tweet_type").count().sort("count
   ")
```

L'istogramma visualizzato su Streamlit è il seguente:

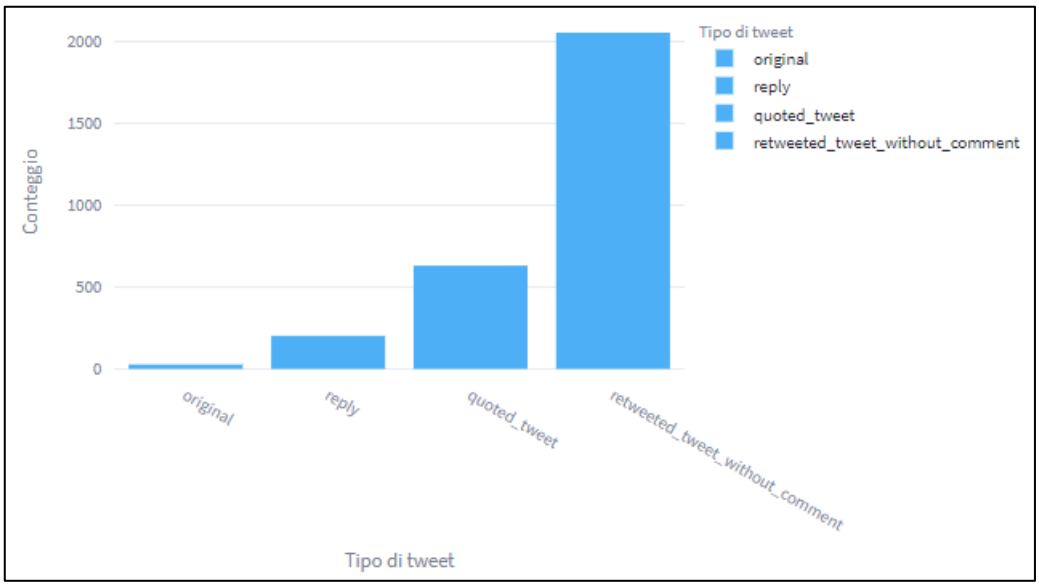

Figura 7 - Rappresentazione Query 4

Query 5

Terminando le operazioni eseguite per la prima domanda di ricerca, nella query 5 si visualizza la distribuzione delle lingue utilizzate nei tweet, eseguita e visualizzata su Streamlit in maniera analoga alle precedenti:

```
1. query5 = name1_dataframe.select("lang").groupBy("lang").count().sort("count")
```

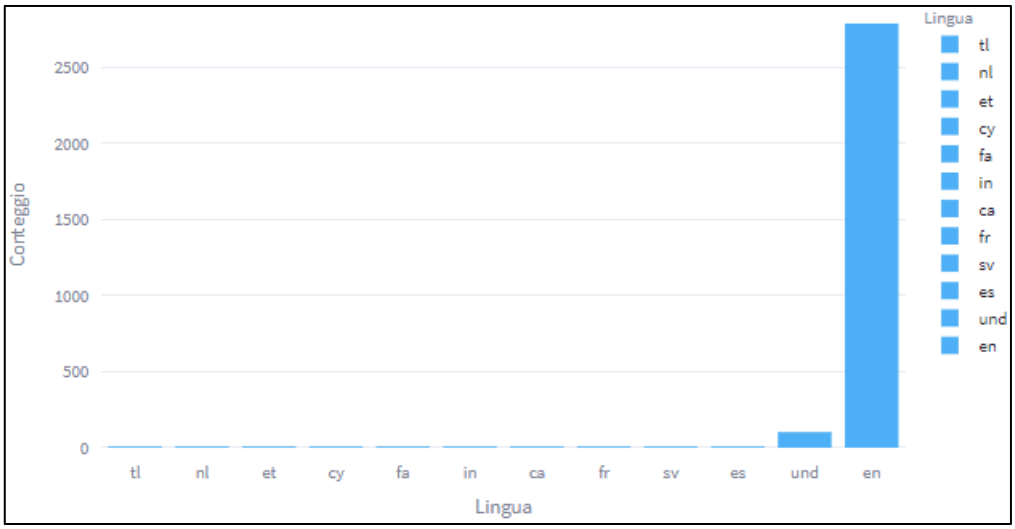

Figura 8 - Rappresentazione Query 5

Query 6

Dalla query 6 si inizia a rispondere alla seconda domanda, cioè quali sono i domini dei link e gli hashtag maggiormente twittati dall'utente. In particolare, in questa query ci si occupa dei domini dei link in questo modo:

```
# Definisci lo schema dei dati JSON
schema = ArrayType(
    StructType([
        StructField('url', StringType(), True),
        StructField('expanded_url', StringType(), True),
        StructField('display_url', StringType(), True),
        StructField('indices', ArrayType(IntegerType()), True)
    ])
)
#I link possono essere presenti in uno dei 3 campi in base al tipo di tweet
# Applica la funzione from_json per analizzare la colonna urls_list
df = name1_dataframe.select("urls_list","rt_urls_list","qtd_urls_list","tweet_type").withColumn('urls_data', from_json('urls_list', schema))
# Applica la funzione from_json per analizzare la colonna rt_urls_list
df = df.withColumn('rt_urls_data', from_json('rt_urls_list', schema))
# Applica la funzione from_json per analizzare la colonna rt_urls_list
df = df.withColumn('qtd_urls_data', from_json('qtd_urls_list', schema))
#Con la funzione from_json si trasforma un campo di tipo string con struttura json
```

```
#in un campo di tipo dizionario
# Estrai gli URL e i domini in base al campo tweet_type
df = df.withColumn('url_info', \
        when(col('tweet_type').isin('original', 'reply'), col('urls_data')) \
       .when(col('tweet_type').isin('quoted_tweet'), col('qtd_urls_data'))
       .otherwise(col('rt_urls_data')) ) \
       .withColumn('url_info', explode('url_info')) \
       .withColumn('url', expr('url_info.expanded_url')) \
       .withColumn('domain', substring_index(substring_index('url', '//', -1),
   '/', 1))
# Esegui una groupby count per contare le occorrenze dei domini
result = df.groupBy('domain').agg(count('*').alias('count'))
result = result.orderBy(col('count').desc())
```

Viene definito uno schema dei dati json, attraverso il quale sono poi recuperate le informazioni necessarie alla nostra elaborazione; in seguito il risultato è convertito in un dizionario python. Una volta estratti gli url e i domini in base al campo tweet_type, facendo attenzione ad esplodere il campo "url_info" in quanto contenente più url, è stata eseguita un'operazione di groupBy e una di conteggio.

Su streamlit, è stata rappresentata così:

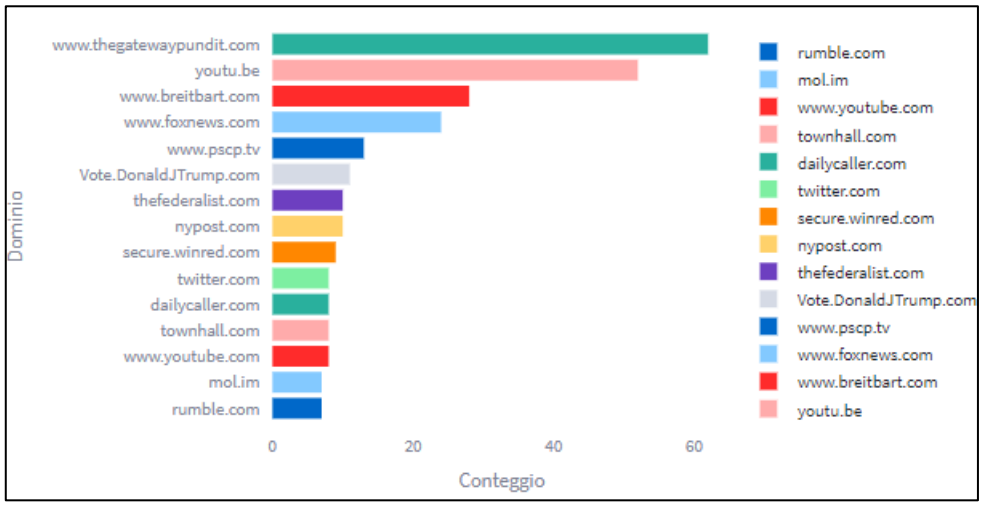

Figura 9 - Rappresentazione Query 6

Query 7

Nella query 7 sono state calcolate le occorrenze di ogni hashtag all'interno dei tweet dell'utente, selezionando il campo hashtag, separando i vari hashtag presenti al suo interno e poi eseguendo un'operazione di groupBy e una di count.

```
# Hashtag è tipo string, rimuove i caratteri '[' e ']'
df = name1_dataframe.select("hashtag").withColumn('hashtag',
expr("substring(hashtag, 3, length(hashtag)-4)"))
# Viene fatto lo split per generare una lista
df = df.withColumn('hashtag_array', split('hashtag', "', '"))
# Viene fatto explode su hashtag_array cosi da replicare ogni riga per quanti
sono
# gli elem della lista hashtag_array
df = df.withColumn('hashtag', explode('hashtag_array'))
# Filtra le righe in cui il campo "hashtag" non è vuoto
df = df.filter(col("hashtag") != "")
# Esegui una groupby count sul singolo elemento della lista "hashtag"
result =
df.groupBy('hashtag').agg(count('*').alias('count')).orderBy(desc('count'))
```

Anche questa in Streamlit è stata visualizzata con un istogramma, analogamente alla query precedente:

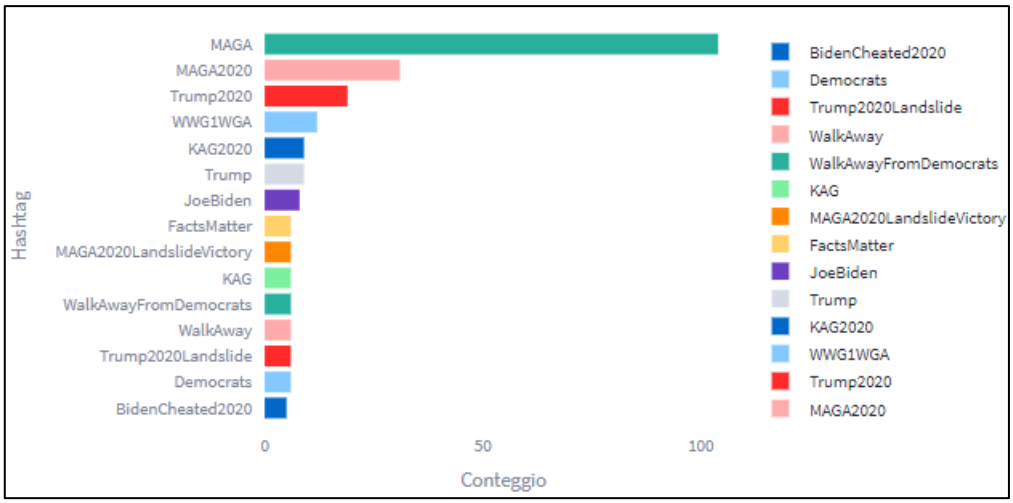

Figura 10 - Rappresentazione Query 7

Query 8

Dalla Query 8 in poi si ricercano quali siano i profili con cui l'utente ha maggiormente interagito tramite tweet. A tal proposito, in questa query sono state contate le volte in cui l'utente ha risposto a tweet prodotti da altri account.

Questo risultato è stato ottenuto filtrando i tweet che contenevano tweet_type = reply, per poi contare quante volte comparisse ogni profilo. Il codice è il seguente:

```
df = name1_dataframe.select("tweet_type","reply_screen").filter(name1_dataframe["tweet_type"] == "reply")
# Esegui una groupby count sul campo reply_screen, il quale contiene l'username
# dell'utente a cui si è risposto
result = df.groupBy('reply_screen').agg(count('*').alias('count')).orderBy(desc('count'))
#join tra la tabella reply_screen-count con la tabella userids, cosi da poter
#associare il campo cat ad ogni utente
joined_df = result.join(userids, col("reply_screen") == col("user"), "left").drop("user")
```

Su Streamlit è stata visualizzata così:

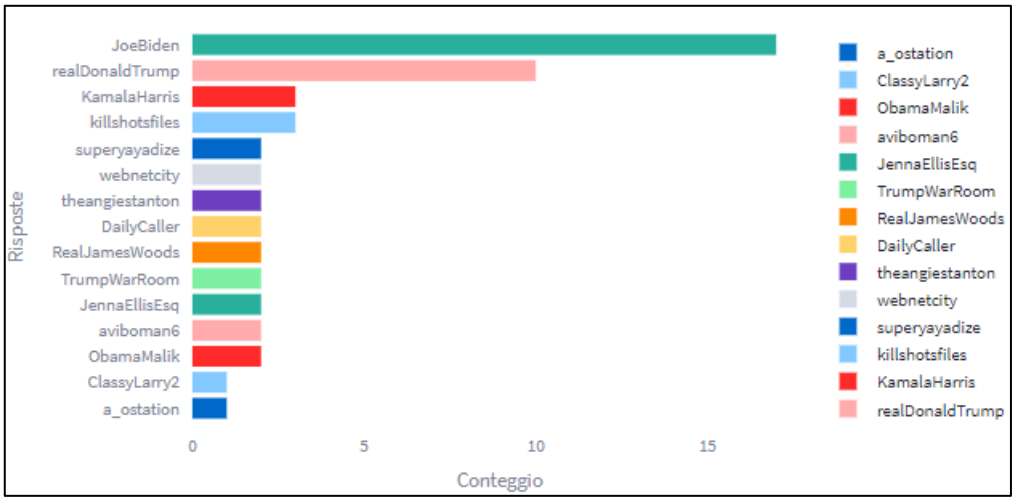

Figura 11 - Rappresentazione Query 8

Query 9

La query 9 è del tutto analoga alla precedente, con la differenza che il conteggio viene fatto sui tweet retwittati senza aggiungere ulteriori commenti.

```
df = name1_dataframe.select("tweet_type","rt_screen").filter(name1_dataframe["tweet_type"] == "retweeted_tweet_without_comment")
# Esegui una groupby count sul campo rt_screen
result = df.groupBy('rt_screen').agg(count('*').alias('count')).orderBy(desc('count'))
joined_df = result.join(userids, col("rt_screen") == col("user"), "left").drop("user")
```

Analoga è anche la visualizzazione sulla dashboard:

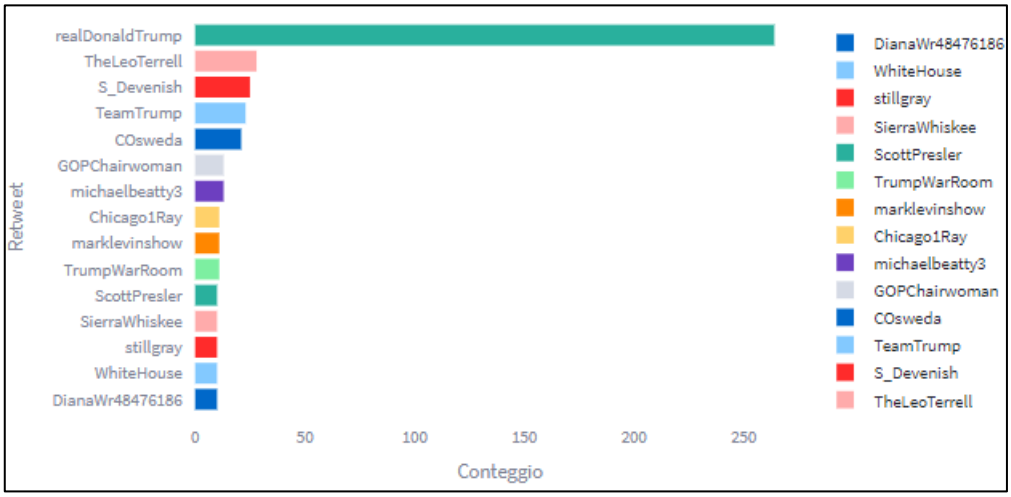

Figura 12 - Rappresentazione Query 9

Query 10

In questa query viene contato il numero di volte in cui un utente ha citato un tweet di un altro utente, analogamente alle query precedenti.

```
1. df =
   name1_dataframe.select("tweet_type","qtd_screen").filter(name1_dataframe["twee
   t_type"] == "quoted_tweet")
2. # Esegui una groupby count sul campo qtd_screen
3. result =
   df.groupBy('qtd_screen').agg(count('*').alias('count')).orderBy(desc('count'))
4. joined_df = result.join(userids, col("qtd_screen") == col("user"),
   "left").drop("user")
```

Anche qui la visualizzazione su Streamlit è fatta tramite istogramma:

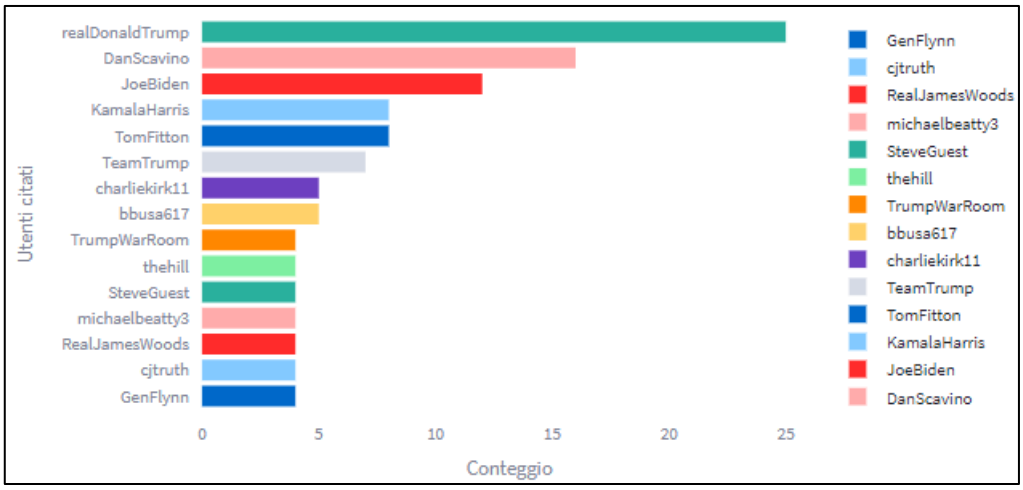

Figura 13 - Rappresentazione Query 10

Query 11

Query 11.1

Nella query 11, è stato calcolato quante volte l'utente in esame menzionasse altri utenti, in maniera analoga a quanto fatto precedentemente:

```
1. df = name1_dataframe.select("mentionsn")
2. # mentionsn è tipo string, rimuove i caratteri '[' e ']'
3. df = df.withColumn('mentionsn_', expr("substring(mentionsn, 3,
   length(mentionsn)-4)"))
4. # Separa la stringa "mentionsn" in un array di elementi
5. df = df.withColumn('mention_array', split('mentionsn_', "', '"))
6. # Viene fatto l'explode sul campo mention_array, cosi da replicare ogni row
7. # per quanti elementi ci sono nella lista mention_array
8. df = df.withColumn('mention', explode('mention_array'))
9. # Filtra le righe in cui il campo "mention" non è vuoto
10.df = df.filter(col("mention") != "")
11.# Viene effettuato il groupby sul campo mention e poi count
12.result =
   df.groupBy('mention').agg(count('*').alias('count')).orderBy(desc('count'))
```

Su Streamlit è stato visualizzato così:

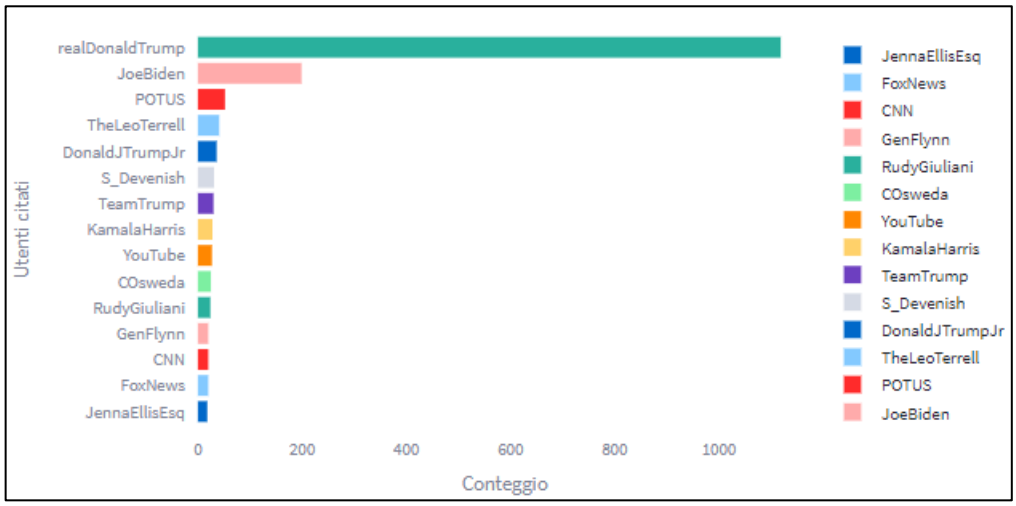

Figura 14 - Rappresentazione Query 11

Query 11.2

Una volta calcolati tutti questi conteggi, è stato poi visto in percentuale con quanti utenti moderati, non moderati oppure non presenti nel dataframe userids l'utente avesse interagito. Su stramlit è stato visualizzato tramite un grafico a torta:

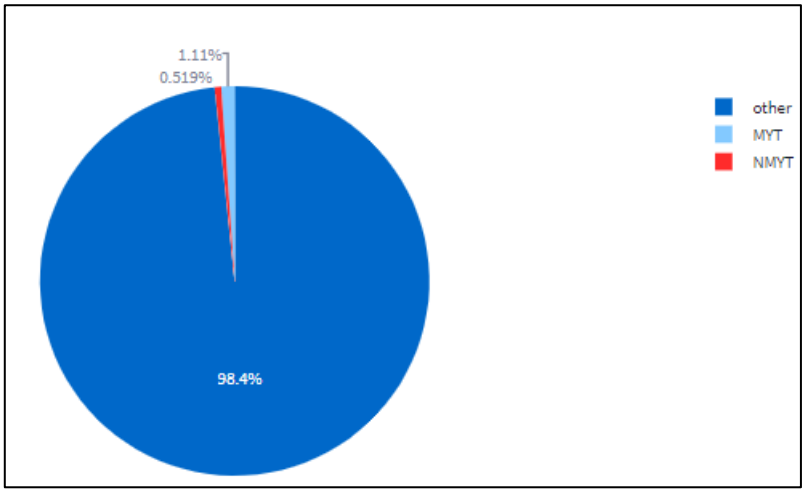

Figura 15 - Rappresentazione Query 12

Sentiment Analysis

Per la sezione di sentiment Analysis, è stato estratto dal dataframe il campo text, contenente il testo del tweet, e questo è poi stato fornito in input ai vari modelli istanziati, per poi produrre un csv per visualizzare i risultati delle query in Streamlit.

```
all_tweets = name1_dataframe.select("tweetid", "text")
with open(f'{path_query}sentiment.csv', 'a', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["tweetid", "sentiment", "irony", "emotion", "topic",
"leaning"])
    #Itero su tutti i tweet dell'utente
    for row in all_tweets.rdd.toLocalIterator():
        s = row["text"]
        sent1 = model_s.sentiment(s)
        sent2 = model_i.irony(s)
        sent3 = model_e.emotion(s)
        sent4 = model_t.topic(s)
        sent6, val = leaning(s)
        writer.writerow([row["tweetid"], sent1, sent2, sent3, sent4, val])
```

I grafici risultanti sono stati i seguenti:

Sentiment

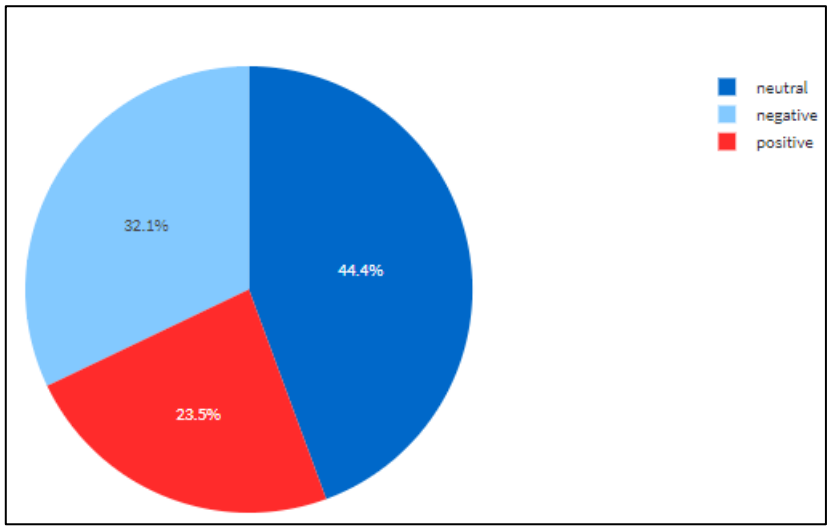

Figura 16 - Rappresentazione Sentiment Analysis (Sentiment)

Topic

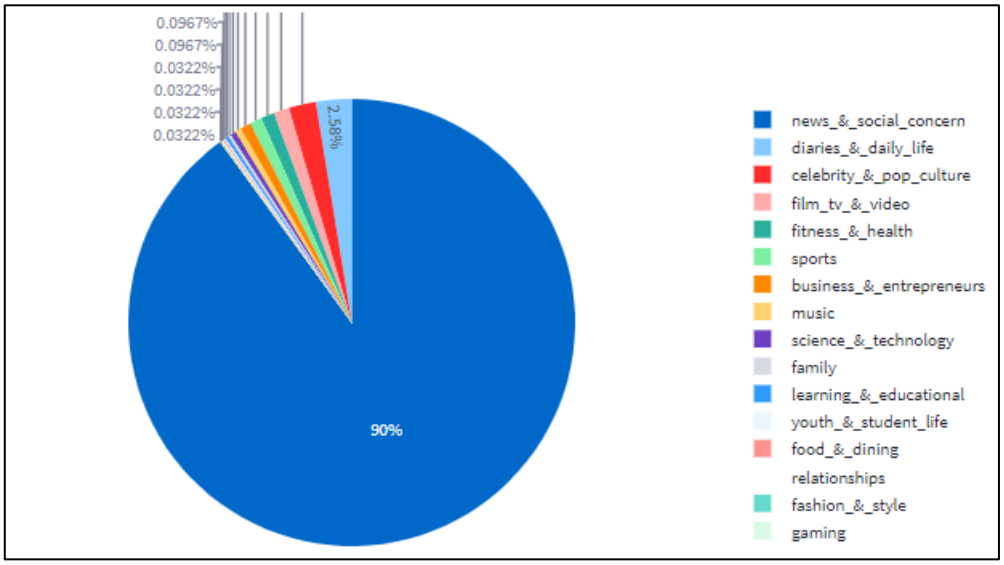

Figura 17 - Rappresentazione Sentiment Analysis (Topic)

Emotion

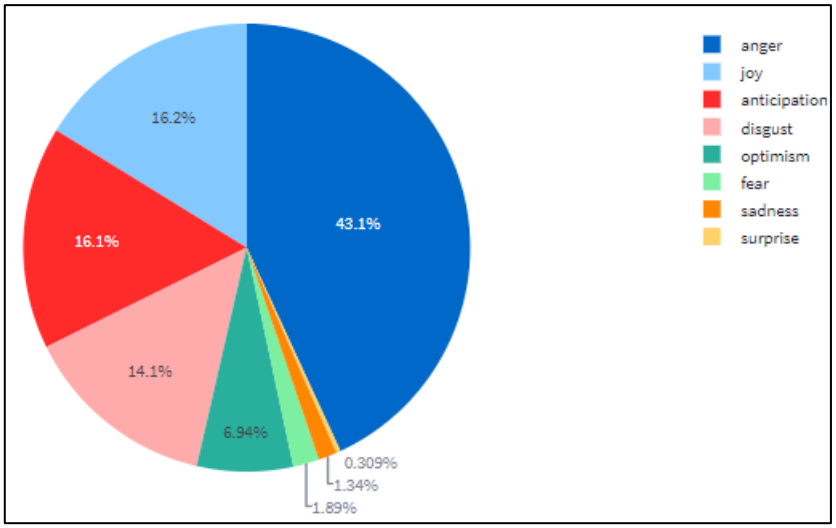

Figura 18 - Rappresentazione Sentiment Analysis (Emotion)

Irony

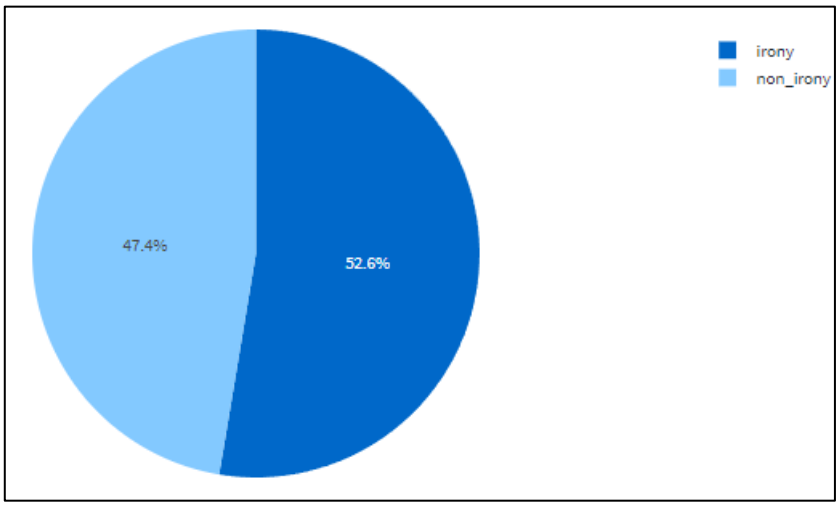

Figura 19 - Rappresentazione Sentiment Analysis (Irony)

Query 12

Nella query 12 ci si è occupati di calcolare le emozioni espresse dall'utente nei tweet che coinvolgono le 2 persone più menzionate, calcolate nella query 11. Il codice eseguito è così fatto:

```
1. @udf
2. def emotion(text):
3.   return model_e.emotion(text)['label']
4. @udf
5. def offensive(text):
6.   return model_o.offensive(text)['label']
7. #NOTA mentionsn è ['a','b'], però è tipo stringa
8. df = name1_dataframe.select("text","mentionsn")
9. most_ment = [e["mention"] for e in query11.head(3)]
10.lis = ["offensive", "non-offensive",
   'anger','anticipation','disgust','fear','joy','love','optimism','pessimism','s
   adness','surprise','trust']
11.dic = {e:{l: 0 for l in lis} for e in most_ment}
12.#Estraggo i tweet dove sono menzionati in mentions almeno uno dei 3 utenti più
   menzionati dall'utente
13.df_mention_filter = name1_dataframe.select("text", "mentionsn") \
14.   .filter(F.greatest(*[F.col("mentionsn").contains(name) for name in
   most_ment]) == True)
15.# Itero sui risultati restituiti dalla query
16.for row in df_mention_filter.rdd.toLocalIterator():
17.   for m in most_ment:
18.     if m in row["mentionsn"]:
```

```
19.         off = model_o.offensive(row["text"])["label"]
20.         emo = model_e.emotion(row["text"])["label"]
21.         dic[m][off] += 1
22.         dic[m][emo] += 1
```

Query 12.1

Su Streamlit, questo è il grafico relativo alla prima persona:

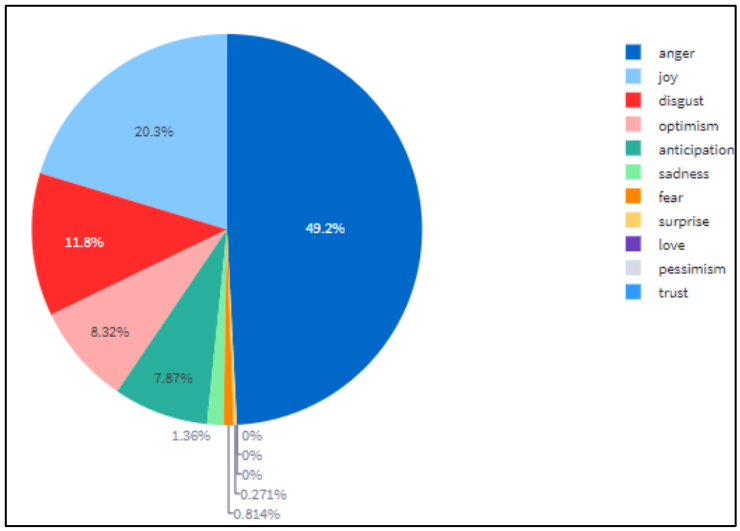

Figura 20 - Rappresentazione query 12.1

Query 12.2

Su Streamlit, questo è il grafico relativo alla seconda persona:

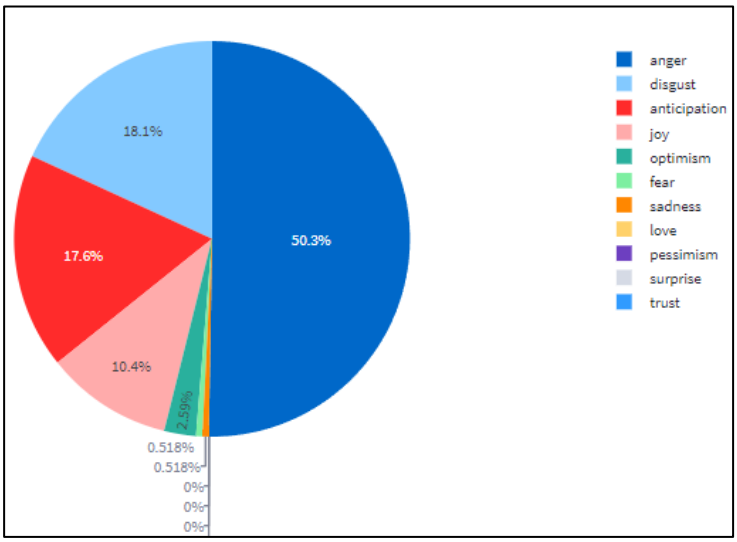

Figura 21 - Rappresentazione query 12.2

Query 12.3

Questa query rappresenta i sentimenti espressi dall'utente nei tweet rivolti ai due profili con cui ha interagito di più, in rosso possiamo vedere la percentuale di tweet offensiva e in blu quella non offensiva.

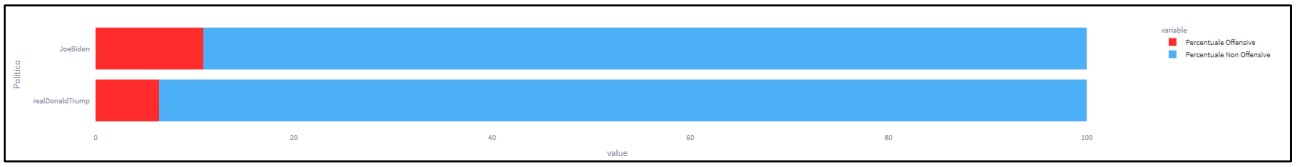

Figura 22 - Rappresentazione query 12.3

Query 13

L'obiettivo dalla query 13 in poi è stata l'osservazione del modo in cui l'utente ha ricondiviso i video di youtube. In particolare, in questa query ci si è soffermati sul verificare se un utente fosse più un mobilitatore passivo o attivo; ciò è stato ottenuto filtrando tutti i tweet contenenti l'url di youtube e poi eseguendo delle operazioni di groupBy e count:

```
1. # Dai tweet vengono selezionati quelli che contengono un url youtube
2. query13 = name1_dataframe.select('urls_list', 'rt_urls_list', 'qtd_urls_list',
   'tweet_type')\
3.     .filter( col('urls_list').contains('youtu') |
   (col('rt_urls_list').contains('youtu') |
   (col('qtd_urls_list').contains('youtu'))  )  )
4. # Group by per tipo di tweet
5. result = query13.groupBy('tweet_type').count()
```

Possiamo vedere il risultato su Streamlit con questo grafico a torta:

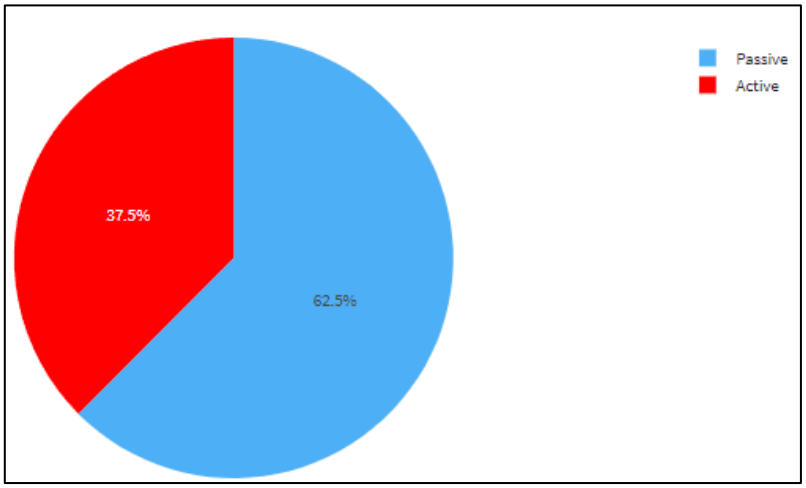

Figura 23 - Rappresentazione query 13

Query 14

Nella query 14 ci si è dedicati a capire quali fossero le opinioni dell'utente riguardo i video condivisi e quanti di questi fossero poi stati eliminati dalla piattaforma.

Per il primo obiettivo, è stato definito uno schema dei dati in formato json attraverso il quale sono stati estratti tutti i tweet contenenti un link ad un video youtube. In seguito, è stato generato un dizionario con degli elementi di tipo tweetid: url ed è stata eseguita un'operazione di groupBy e una di count.

Per il secondo invece obiettivo, sono state utilizzate le API di youtube per capire quali video fossero stati eliminati o no, tramite una funzione custom che prende in input l'url del video. Questi risultati sono poi stati aggregati in un csv e su Streamlit sono stati visualizzati tramite un grafico a torta.

```
# Definisci lo schema dei dati JSON
schema = ArrayType(
    StructType([
        StructField('url', StringType(), True),
        StructField('expanded_url', StringType(), True),
        StructField('display_url', StringType(), True),
        StructField('indices', ArrayType(IntegerType()), True)
    ])
)
# I link possono essere presenti in tre campi diversi a seconda del tweet type
# Applica la funzione from_json per analizzare la colonna urls_list(reply, original)
df = name1_dataframe.select('urls_list', 'rt_urls_list', 'qtd_urls_list', 'tweet_type','tweetid') \
..withColumn('urls_data', from_json('urls_list', schema))
# Applica la funzione from_json per analizzare la colonna rt_urls_list(retweeted)
df = df.withColumn('rt_urls_data', from_json('rt_urls_list', schema))
# funz json per la colonna qtd_urls_list (quoted)
df = df.withColumn('qtd_urls_data', from_json('qtd_urls_list', schema))
df = df.withColumn('url_info', \
        when(col('tweet_type').isin('original', 'reply'), col('urls_data')) \
       .when(col('tweet_type').isin('quoted_tweet'), col('qtd_urls_data'))
       .otherwise(col('rt_urls_data')) ) \
       .withColumn('url_info', explode('url_info')) \
       .withColumn('url', expr('url_info.expanded_url'))
#ottengo solo i tweet che contengono link youtube
df = df.filter( col('url').contains('youtu') )
# genero un dizionario con più elementi di tipo tweetid:url
```

```
yt_video_dict = {row['tweetid']:row['url'] for row in df.collect()}
# Esegui una groupby count per contare le occorrenze dei domini
n_total_video = df.agg(count('*').alias('count')).collect()[0]['count']
```

```
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
def parseYoutubeURL(url):
    data = re.findall(r"(?:v=|\/)([0-9A-Za-z_-]{11}).*", url)
    if data:
        return data[0]
    return ""
def is_video_removed(video_id, api_key):
    try:
        youtube = build('youtube', 'v3', developerKey=api_key)
        #  richiesta API per ottenere i dettagli del video
        response = youtube.videos().list(
            part='status',
            id=video_id
        ).execute()
        l = len(response['items'])
        if(l==0): #rimosso
          return True
        # Verificare lo stato del video
        if response['items'][0]['status']['uploadStatus'] == 'removed':
            return True
        else:
            return False
    except HttpError as e:
        print('Errore durante la richiesta API:', e)
        return False

# La tua chiave API
api_key = '-U'
# L'obbiettivo
n_removed, n_neutral, n_positive, n_negative = 0
# Itero sul dizionario dei video yt estratti in precedenza
for tweetid, video in yt_video_dict.items():
  v_id = parseYoutubeURL(video)
  sent = sentiment_tweet.select("sentiment").filter(col("tweetid")==tweetid).collect()[0]["sentiment"]
  if "neutral" in sent:
    n_neutral += 1
```

```
38.   elif "positive" in sent:
39.     n_positive += 1
40.   else:
41.     n_negative += 1
42.   # Verificare se il video è stato rimosso
43.   removed = is_video_removed(v_id, api_key)
44.   if removed:
45.       n_removed += 1
```

Query 14.1

Questo grafico esprime la percentuale di emozioni espresse dall'utente nei tweet contenenti l'url di un video di youtube

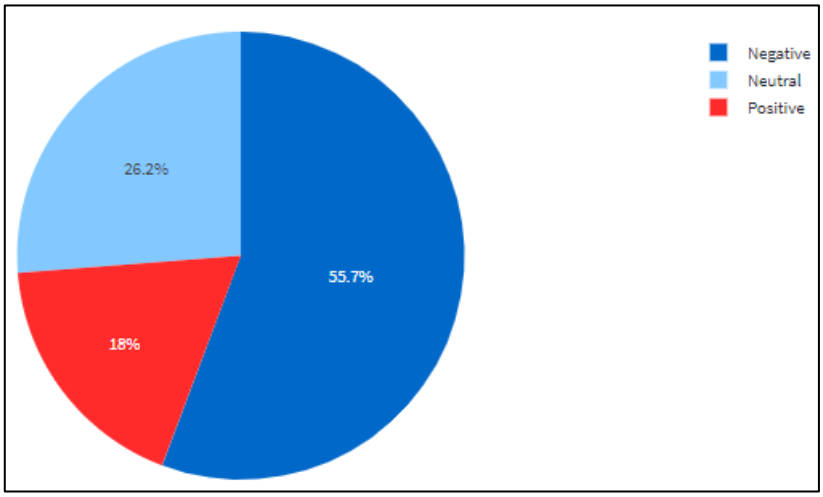

Figura 24 - Rappresentazione query 14.1

Query 14.2

Questa barra invece rappresenta la percentuale (in rosso) dei video di youtube condivisi dall'utente che sono stati eliminati.

Figura 25 - Rappresentazione query 14.2

Inclinazione politica

Con i dati estratti tramite la rete neurale poi, è stata creata tramite HTML una barra che indichi l'inclinazione politica dei tweet dell'utente, che in streamlit viene così rappresentata:

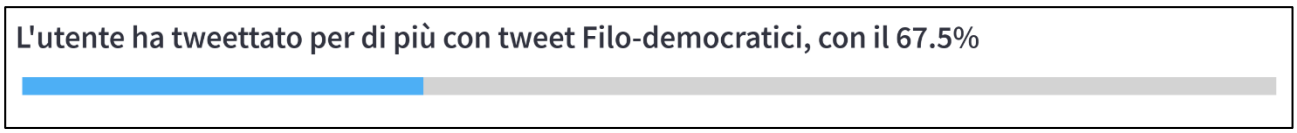

Figura 26 - Rappresentazione Inclinazione Politica

Conclusioni

Lo studio effettuato ha portato alla luce le caratteristiche di ogni utente analizzato, partendo dalle sue abitudini nell'utilizzare la piattaforma e passando per i suoi interessi, fino ad arrivare alle sue preferenze politiche.

Nonostante i dati in esame fossero di dimensioni considerevoli, l'elaborazione degli stessi è stata gestibile anche da una macchina cloud come Google Colab, grazie all'impiego di tool appropriati.