

UNIVERSITÀ DELLA CALABRIA

Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e  
Sistemistica



UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI  
**INGEGNERIA INFORMATICA,  
MODELLISTICA, ELETTRONICA  
E SISTEMISTICA**

DIMES

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Relazione del Progetto di Modelli e Tecniche per Big Data

## Disastri Naturali

**Professori:**

Paolo Trunfio  
Fabrizio Marozzo

**Studenti:**

Mattia Presta 239051  
Silvio Raso 235219

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
1.1	Obiettivi . . . . .	2
1.2	Interfaccia Web . . . . .	3
<b>2</b>	<b>Descrizione Del Dataset</b>	<b>3</b>
<b>3</b>	<b>Tecnologie Utilizzate</b>	<b>3</b>
<b>4</b>	<b>Fasi preliminari</b>	<b>4</b>
4.1	Configurazione Apache Spark . . . . .	4
4.2	Lettura dei file JSON . . . . .	4
<b>5</b>	<b>Operazioni di Pre-Processing</b>	<b>5</b>
5.1	Selezione dei campi più significativi . . . . .	5
5.2	Join Dataset Tweet - Dataset Tweet Classificati . . . . .	6
<b>6</b>	<b>Implementazione</b>	<b>6</b>
6.1	Query con Visualizzazione . . . . .	7
6.1.1	Conteggio dei tweet per data . . . . .	7
6.1.2	Numero medio dei tweet per ogni ora del giorno . . . . .	7
6.1.3	Conteggio dei tweet da parte di utenti verificati . . . . .	9
6.1.4	Parole più frequenti nei tweet . . . . .	9
6.1.5	Hashtag più frequenti . . . . .	11
6.1.6	Conteggio dei tweet per ogni tipologia . . . . .	11
6.1.7	Parole più frequenti per ogni tipologia di tweet . . . . .	12
6.1.8	Conteggio dei tweet nei Stati più colpiti (Texas, Louisiana) . . . . .	16
6.1.9	Tweet Geolocalizzati . . . . .	16
<b>7</b>	<b>Query con l'utilizzo del Machine Learning</b>	<b>19</b>
7.1	Sentiment Analisys . . . . .	19
7.2	Sentiment Analisys dei tweet degli Stati più colpiti . . . . .	21
7.3	Matrice di correlazione . . . . .	22
7.4	Classificazione con Random Forest . . . . .	24
7.4.1	Valutazione delle prestazioni . . . . .	27
7.5	Classificazione con Regressione Logistica . . . . .	28
7.5.1	Valutazione delle prestazioni . . . . .	28
<b>8</b>	<b>Conclusioni</b>	<b>29</b>

# 1 Introduzione

Nel 2017, tre disastri naturali devastanti, ovvero l'uragano Harvey, l'uragano Irma e l'uragano Maria, hanno causato danni catastrofici per miliardi di dollari e numerose vittime, lasciando migliaia di persone colpite. Durante queste emergenze potenzialmente letali, le persone colpite e vulnerabili, le organizzazioni umanitarie e le autorità competenti cercano informazioni utili per prevenire la crisi o per aiutare gli altri. Le organizzazioni di risposta formale, come le agenzie governative, le autorità sanitarie pubbliche e militari, hanno bisogno di informazioni tempestive e credibili per prendere decisioni rapide e avviare gli sforzi di soccorso. Le esigenze informative di questi soggetti variano in base al loro ruolo, alle responsabilità e alla situazione che stanno affrontando, ma in situazioni critiche l'importanza di informazioni tempestive e accurate aumenta, specialmente quando altre fonti tradizionali come la TV o la radio non sono disponibili.

L'uso crescente delle tecnologie dell'informazione e della comunicazione (ICT), delle tecnologie mobili e delle piattaforme dei social media come Twitter e Facebook ha fornito al pubblico in generale opportunità facili da usare ed efficaci per diffondere e ricevere informazioni. Milioni di persone utilizzano sempre più i social media durante disastri naturali e causati dall'uomo. Studi di ricerca hanno dimostrato l'utilità delle informazioni sui social media per una serie di compiti umanitari come la "consapevolezza della situazione". Tuttavia, fare senso di queste informazioni in situazioni critiche è una sfida, a causa del volume e della velocità elevati dei dati dei social media, l'analisi manuale di migliaia di messaggi è impossibile.

La comprensione dei dati dei social media per aiutare i soccorritori comporta la risoluzione di molte sfide, tra cui l'analisi di contenuti non strutturati e brevi, la filtrazione di contenuti irrilevanti e rumorosi, la gestione dell'overload informativo, tra le altre.

Questa lavoro fornisce una panoramica dettagliata dell'analisi dei dati Twitter utilizzando il linguaggio di programmazione Python. In particolare, viene effettuata un'analisi del sentiment per determinare come pensieri e sentimenti delle persone cambino nel tempo durante l'evolversi degli eventi di crisi. Vengono utilizzate tecniche di modellizzazione dei temi per comprendere i diversi argomenti discussi durante ciascuna giornata.

## 1.1 Obiettivi

L'obiettivo del progetto è realizzare un'applicazione che permetta di effettuare interrogazioni aggregate sfruttando le informazioni dai social media. In particolare, il focus è comprendere la reazione delle persone al disastro naturale andando a studiare alcuni aspetti come:

- **Analisi del Sentiment:** Il progetto utilizza l'analisi del sentiment per valutare le emozioni e le opinioni espresse nei tweet. Questo può aiutare a comprendere come le persone reagiscono emotivamente all'evento di emergenza, se sono preoccupate, spaventate, ottimiste o neutre.
- **Classificazione dei Tweet:** I tweet vengono classificati in diverse categorie o tipologie, come tweet di solidarietà, informazioni rilevanti, richieste di aiuto, consigli sulla sicurezza, ecc. Questa classificazione aiuta a organizzare e comprendere meglio il contenuto dei tweet.
- **Valutazione dei Modelli di Machine Learning:** Il progetto include l'addestramento di modelli di machine learning, come il Random Forest e la Regressione Logistica, per classificare i tweet. L'obiettivo è valutare l'efficacia di questi modelli nel categorizzare i tweet in base alle categorie di interesse.
- **Analisi delle Parole Chiave:** Viene effettuata un'analisi delle parole chiave o dei termini più frequenti nei tweet. Questo può rivelare quali sono le principali preoccupazioni dagli utenti durante l'evento di emergenza.
- **Analisi Geografica:** I tweet geolocalizzati vengono visualizzati su una mappa per identificare le aree più colpite dall'evento di emergenza e per comprendere meglio come le persone nelle diverse regioni rispondono all'evento.
- **Analisi delle Etichette e delle Metriche:** Viene condotta un'analisi delle etichette associate ai tweet, inclusa la fiducia attribuita a ciascuna etichetta. Questa analisi può aiutare a determinare quanto i tweet siano considerati affidabili o meno.
- **Confronto tra Regioni Colpite:** Il progetto mira anche a confrontare le reazioni e il comportamento degli utenti nelle regioni più colpite dall'evento di emergenza.

I risultati possono essere utilizzati per migliorare la risposta alle emergenze e la comunicazione pubblica durante tali situazioni critiche.

## 1.2 Interfaccia Web

Per poter rendere più comprensibile l'output delle interrogazioni è stata realizzata un'interfaccia web user-friendly. In questo modo le operazioni effettuate sui dataset vengono visualizzate tramite grafici interattivi. Per avviare l'applicazione è necessario posizionarsi all'interno della cartella di lavoro digitando:

```
1 streamlit run application.py
```

## 2 Descrizione Del Dataset

Il Dataset (fornito dai docenti, al seguente link: Dataset) contiene informazioni di dettaglio su dati su disastri naturali, concentrando la sua attenzione sul disastro provocato dall'uragano Harvey nel 2017. Include una vasta raccolta di tweet pubblicati dagli utenti durante l'uragano, fornendo un'ampia varietà di informazioni come commenti sulla situazione, segnalazioni di danni, richieste di aiuto, aggiornamenti sul meteo e il sentimento delle persone riguardo all'evento. Questi dati sono preziosi per comprendere la percezione pubblica durante una crisi e possono essere utilizzati per migliorare la risposta e il coordinamento dei soccorsi in futuro. Il dataset rappresenta una risorsa fondamentale per la ricerca e l'analisi delle dinamiche sociali durante i disastri naturali. Il periodo di campionamento dei tweet corrisponde alla finestra temporale 27 agosto-19settembre 2017. Certamente, basandoci sulle informazioni fornite sui tre elementi dati, possiamo fornire una descrizione più dettagliata del contenuto dei dataset:

### 1. dataset001 e dataset002 (Cartelle contenenti file JSON):

- Entrambe le cartelle contengono numerosi file JSON.
- Ogni file JSON all'interno di queste cartelle rappresenta un insieme di tweet.
- I dati nei file JSON includono dettagli sui tweet pubblicati durante l'uragano Harvey nel 2017, come il testo dei tweet, l'utente che li ha pubblicati, l'orario di pubblicazione, e altre informazioni pertinenti.
- La size è di circa 3 milioni di tweet.
- dataset001 e dataset002 contengono rispettivamente 21 e 9 file JSON.

### 2. harvey data 2017 aidr classification.txt (File di testo):

- Si tratta di un file di testo in cui sono presenti informazioni relative all'identificazione e alla classificazione dei tweet nei dataset001 e dataset002. In particolare, oltre a possedere informazioni riguardanti l'uragano Harvey possiede anche informazioni relative ai tweet di altri uragani, Irma e Maria.
- Contiene 12 etichette uniche che classificano i tweet.
- La size è di circa 9 milioni di tweet.

Questi dati sono preziosi per condurre analisi approfondite sulla risposta dei social media durante l'uragano Harvey e per comprendere meglio la natura e la rilevanza dei tweet in base a criteri specifici.

## 3 Tecnologie Utilizzate

Prima di passare all'implementazione vera e propria dell'applicazione, in fase di progettazione si è scelto di utilizzare le seguenti tecnologie:

- **Apache Spark:** è preferito per la sua capacità di elaborare dati in parallelo accelerando notevolmente l'analisi su dataset di grandi dimensioni. Esistono diversi vantaggi nell'utilizzo di Apache Spark per l'elaborazione dei dati:
  - **Scalabilità:** Spark è progettato per lavorare su cluster di computer, rendendolo ideale per elaborare grandi quantità di dati in modo efficiente.

- **Velocità:** Spark è molto più veloce rispetto ad altre soluzioni di elaborazione dei dati poiché utilizza la memoria del sistema invece di scrivere su disco rigido.
- **Linguaggio Python:** è ampiamente utilizzato nell'ambito del big data per la sua flessibilità e vasta libreria di strumenti specializzati. La sua semplicità e facilità d'uso consentono di scrivere rapidamente codice per l'analisi dei dati e l'applicazione di algoritmi di machine learning su dataset di grandi dimensioni, semplificando notevolmente lo il lavoro.
- **Dataframe Spark e RDD.**
- **Streamlit:** è una libreria Python che consente di creare applicazioni web interattive per la visualizzazione e l'analisi dei dati in modo rapido e semplice. È ampiamente utilizzato nella comunità di data science per condividere risultati ed esplorare dati in modo interattivo.

I dettagli verranno esplicitati nella sezione riguardante l'implementazione.

## 4 Fasi preliminari

### 4.1 Configurazione Apache Spark

È necessario inizializzare Spark tramite SparkSession per configurare e creare un ambiente di esecuzione appropriato per le applicazioni Spark, questo consente di utilizzare efficacemente le risorse di calcolo e di memoria disponibili e di eseguire operazioni distribuite su cluster di computer. Di seguito il codice:

```

1 spark = SparkSession.builder.config("spark.executor.cores", "8") \
2     .config("spark.executor.memory", "8g") \
3     .config("spark.driver.memory", "4g") \
4     .config("spark.memory.offHeap.enabled", "true") \
5     .config("spark.memory.offHeap.size", "2g") \
6     .getOrCreate()

```

- **spark.executor.cores:** imposta il numero di core CPU per ciascun executor Spark: 8 core per ogni executor.
- **spark.executor.memory:** imposta la quantità di memoria da assegnare a ciascun executor: 8 gigabyte di memoria per ogni executor.
- **spark.driver.memory:** imposta la quantità di memoria da assegnare al driver Spark. Il driver è responsabile della gestione complessiva dell'applicazione Spark: 4 gigabyte di memoria per il driver.
- **spark.memory.offHeap.enabled:** abilita l'utilizzo di memoria off-heap, che è una memoria gestita direttamente dal sistema Spark al di fuori della JVM (Java Virtual Machine). È utile per gestire in modo più efficiente la memoria Spark.
- **spark.memory.offHeap.size:** imposta la dimensione della memoria off-heap: 2 gigabyte di memoria off-heap.

### 4.2 Lettura dei file JSON

```

1 directory="Data Set/001"
2 file_paths = [os.path.join(directory, file) for file in os.listdir(
3     directory) if file.endswith(".json")]
4 df1 = spark.read.json(file_paths)
5 df1= df1.drop("_corrupt_record")
6 directory2="Data Set/002"
7 file_paths2 = [os.path.join(directory2, file) for file in os.listdir(
8     directory2) if file.endswith(".json")]
9 df2 = spark.read.json(file_paths2)

```

Il codice sopra esegue le seguenti azioni:

- Imposta la variabile **directory** con il percorso della directory **Data Set/001**.

- Crea una lista **file\_paths** che contiene i percorsi completi a tutti i file con estensione **.json** presenti nella directory **Data Set/001**.
- Utilizza **spark.read.json** per leggere i dati dai file JSON elencati in **file\_paths** e crea un DataFrame chiamato **df1** che rappresenta i dati estratti dai file JSON.
- Rimuove eventuali colonne chiamate **\_corrupt\_record** dal DataFrame **df1**. Questa operazione elimina eventuali dati corrotti o non validi.
- Successivamente, come si può vedere, si eseguono operazioni simili per la directory **Data Set/002**.

Infine, con la seguente riga di codice:

```
1 df_start = df1.union(df2)
```

si ottiene **df\_start** che contiene tutte le righe presenti sia in **df1** che in **df2**, senza duplicati. Questo è stato fatto in quanto si desidera trattare i dati provenienti da entrambi i DataFrame come se provenissero dalla stessa fonte.

## 5 Operazioni di Pre-Processing

### 5.1 Selezione dei campi più significativi

Nel contesto dell'elaborazione dei dati provenienti dai file JSON, è essenziale selezionare i campi più significativi e rilevanti per l'analisi e l'elaborazione successiva. La selezione dei campi è una pratica comune nell'ambito dell'analisi dei dati e presenta diverse ragioni:

- **Rilevanza dei dati:** I campi selezionati sono quelli che contengono informazioni fondamentali per comprendere e analizzare i tweet. Ad esempio, il campo **tweet\_id** identifica in modo univoco ogni tweet, mentre il campo **tweet\_date\_full** fornisce la data e l'orario di creazione del tweet, informazioni cruciali per l'analisi temporale.
- **Contesto geografico:** I campi **tweet\_pos** e **tweet\_place** forniscono dati sulla posizione geografica associata al tweet, il che può essere rilevante per l'analisi basata sulla geolocalizzazione o per la comprensione del contesto in cui è stato pubblicato il tweet.
- **Informazioni sull'utente:** I campi come **user\_id**, **user\_name**, **user\_location**, **user\_followers** e **user\_verified** offrono dettagli sull'utente autore del tweet. Questi dati sono importanti per comprendere il profilo dell'utente, la sua autenticità (verificato o meno) e il suo impatto (numero di follower).
- **Testo del tweet:** Il campo **text** contiene il testo effettivo del tweet, che rappresenta il contenuto principale e le informazioni comunicate dal messaggio.
- **Hashtag:** Il campo **hashtags** cattura gli hashtag presenti nel tweet, che possono rivelare argomenti rilevanti o tendenze associate ai messaggi.
- **Crisi associata:** Il campo **crisis\_name** può essere utile per associare il tweet a una specifica crisi o evento, facilitando l'analisi delle discussioni relative a eventi specifici.

```
1 df_start = df_start.select(
2     col("id").alias("tweet_id"),
3     col("created_at").alias("tweet_date_full"),
4     col("geo.coordinates").alias("tweet_pos"),
5     col("place.full_name").alias("tweet_place"),
6     col("aidr.crisis_name").alias("crisis_name"),
7     col("user.id").alias("user_id"),
8     col("user.name").alias("user_name"),
9     col("user.location").alias("user_location"),
10    col("user.followers_count").alias("user_followers"),
11    col("user.verified").alias("user_verified"),
12    "text",
13    col("entities.hashtags.text").alias("hashtags")
14 )
```

Inoltre, è da notare l'operazione di ridenominazione dei campi selezionati. Questa è stata fatta per poter rendere più comprensibile e leggibile il DataFrame `df_start`.

Per agevolare alcune interrogazioni si è resa necessaria la modifica del campo **Date** nel seguente modo:

```

1 # Estraggo la parte della data (i primi 10 caratteri)
2 df_start = df_start.withColumn("tweet_date", substring("tweet_date_full",
3   1, 10))
4
5 # Estraggo la parte dell'ora (dopo il 10    carattere)
6 df_start = df_start.withColumn("tweet_time", substring("tweet_date_full",
7   12, 8))

```

## 5.2 Join Dataset Tweet - Dataset Tweet Classificati

Si effettua un join tra `df_start` e `df3` in base alla chiave `tweet_id` per creare un nuovo DataFrame chiamato `df_combined` che contiene dati combinati da entrambe le fonti.

```

1 df_combined = df_start.join(df3, on="tweet_id", how="inner")

```

L'operazione di join è di tipo `inner`, il che significa che vengono mantenute solo le righe con chiavi corrispondenti in entrambi i DataFrame.

## 6 Implementazione

Dopo le fasi preliminari di pre-processing, possiamo specificare i passi dell'implementazione vera e propria dell'applicazione che si suddivide in **Interrogazioni** e **Visualizzazione del loro output**. In particolare, questo ha richiesto lo sviluppo di due file separati `queries.py` e `application.py`.

Per la realizzazione delle Interrogazioni si è reso necessario l'utilizzo di determinate librerie:

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import plotly.express as px
4 import pyspark
5 from pyspark.sql import SparkSession
6 from pyspark.sql.window import Window
7 from pyspark.sql.functions import hour, avg, col, trim, to_date,
8   date_format, when, substring, explode, lower, udf, split
9 from pyspark.sql.types import StringType, StructType, StructField,
10  DoubleType
11 from pyspark.ml.feature import StringIndexer, VectorAssembler,
12  IndexToString
13 from pyspark.ml.stat import Correlation
14 from pyspark.ml.classification import RandomForestClassifier,
15  LogisticRegression
16 from pyspark.ml.evaluation import MulticlassClassificationEvaluator
17 from pyspark.mllib.evaluation import MulticlassMetrics
18 from pyspark.ml import Pipeline
19 import nltk
20 from nltk.corpus import stopwords
21 from nltk.sentiment.vader import SentimentIntensityAnalyzer

```

Invece, per la visualizzazione dell'output delle queries si è reso necessario l'utilizzo di determinate librerie:

```

1 from queries import *
2 import streamlit as st
3 import plotly.express as px
4 import plotly.graph_objects as go
5 import seaborn as sns
6 import pydeck as pdk
7 import numpy as np

```

## 6.1 Query con Visualizzazione

In questa sezione verranno visualizzati gli output delle interrogazioni e il loro di codice.

### 6.1.1 Conteggio dei tweet per data

```
1 def dateTweet():
2     date_tweet_count = df_start.filter(col("tweet_date").isNotNull())
3     date_tweet_count = date_tweet_count.withColumn("tweet_date", substring
4         ("tweet_date", 5, 10))
5     date_tweet_count = date_tweet_count.withColumn("tweet_date", to_date(
6         date_tweet_count["tweet_date"], "MMM dd"))
7     date_tweet_count = date_tweet_count.withColumn("tweet_date",
8         date_format(date_tweet_count["tweet_date"], "MM-dd"))
9     date_tweet_count = date_tweet_count.groupBy("tweet_date").count().
10        orderBy("tweet_date")
11
12     return date_tweet_count
```

L'obiettivo della funzione dateTweet() è eseguire un conteggio dei tweet in base alla data. Ciò è utile per analizzare e visualizzare le tendenze temporali nell'attività dei tweet.

L'output è il seguente:

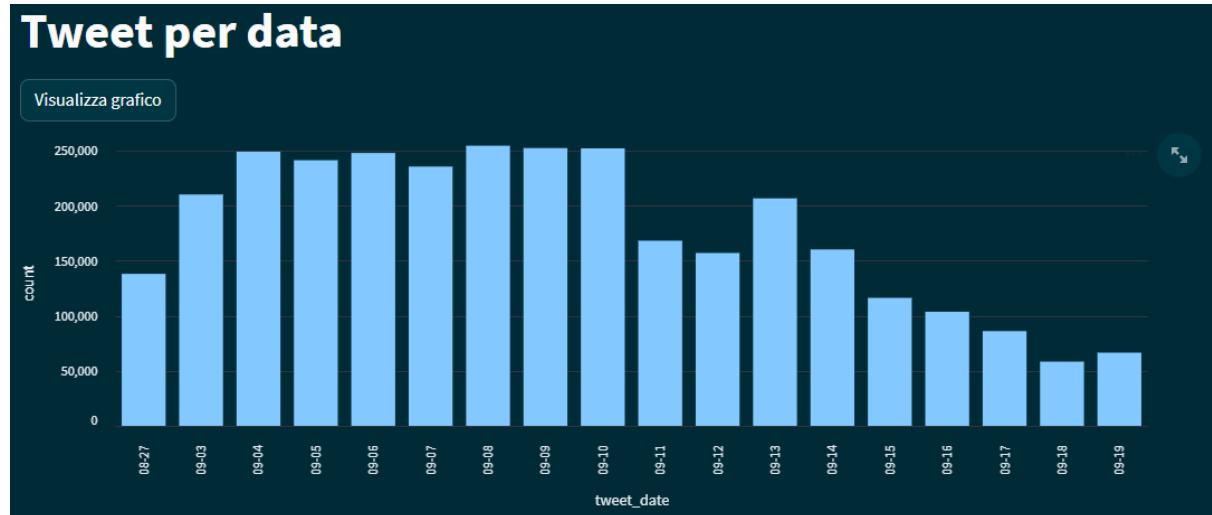


Figure 1: Tweet per Data

Nel contesto di un disastro causato da un uragano, dall'output della funzione dateTweet() si è visto che nella finestra temporale che va dal 3 settembre al 13 settembre il numero di tweet è arrivato a superare significativamente la soglia dei 200mila al giorno.

In generale, questa funzione fornisce un quadro temporale dell'attività sui social media in relazione all'uragano e alle sue conseguenze. Ciò consente di avere una visione più chiara dell'andamento della situazione e di prendere decisioni informate per la gestione dell'emergenza e il soccorso alla popolazione colpita.

### 6.1.2 Numero medio dei tweet per ogni ora del giorno

La funzione mediaTweetPerOra() esegue un'analisi dei tweet per calcolare la media dei tweet per ogni ora del giorno.

```
1 def mediaTweetPerOra():
2     # Aggiungi una colonna "tweet_hour" per l'ora del giorno
3     df_with_hour = df_start.withColumn("tweet_hour", hour("tweet_time"))
4     # Calcola il conteggio medio dei tweet per ogni ora utilizzando una
5         finestra
6     window_spec = Window.partitionBy("tweet_date").orderBy("tweet_hour")
```

```

1   hourly_tweet_avg = df_with_hour.groupBy("tweet_date", "tweet_hour") .
2       count() \
3           .withColumn("avg_tweet_count", avg("count").over(window_spec))
4   hourly_tweet_avg = hourly_tweet_avg.filter(hourly_tweet_avg(tweet_hour
5       .isNotNull())
6   hourly_tweet_avg = hourly_tweet_avg.groupBy("tweet_hour").agg(avg("c
7       count").alias("avg_tweet_count")).orderBy("tweet_hour")
8   hourly_tweet_avg = hourly_tweet_avg.toPandas()
9
10  return hourly_tweet_avg

```

Ecco una spiegazione dettagliata di ciò che fa:

- `window_spec = Window.partitionBy("tweet_date").orderBy("tweet_hour")`:

Viene definita una specifica di finestra (*window specification*) chiamata "window\_spec". Questa specifica di finestra viene utilizzata per organizzare i dati in base alla data del tweet e all'ora del giorno. In pratica, i dati verranno suddivisi in partizioni basate sulla data e ordinati all'interno di ciascuna partizione in base all'ora del giorno.

- `hourly_tweet_avg = df_with_hour
 .groupBy("tweet_date", "tweet_hour")
 .count()
 .withColumn("avg_tweet_count", avg("count")
 .over(window_spec))`:

In questa riga, il DataFrame `df_with_hour` viene raggruppato per "tweet\_date" e "tweet\_hour", cioè per data e ora del giorno. Viene quindi calcolato il conteggio dei tweet in ciascun gruppo utilizzando la funzione `count()`. Successivamente, viene utilizzata la funzione `withColumn()` per calcolare la media mobile (rolling average) del conteggio dei tweet all'interno della finestra specificata da "window\_spec" utilizzando la funzione `avg("count").over(window_spec)`. Il risultato viene memorizzato nella colonna "avg\_tweet\_count".

L'output è il seguente:

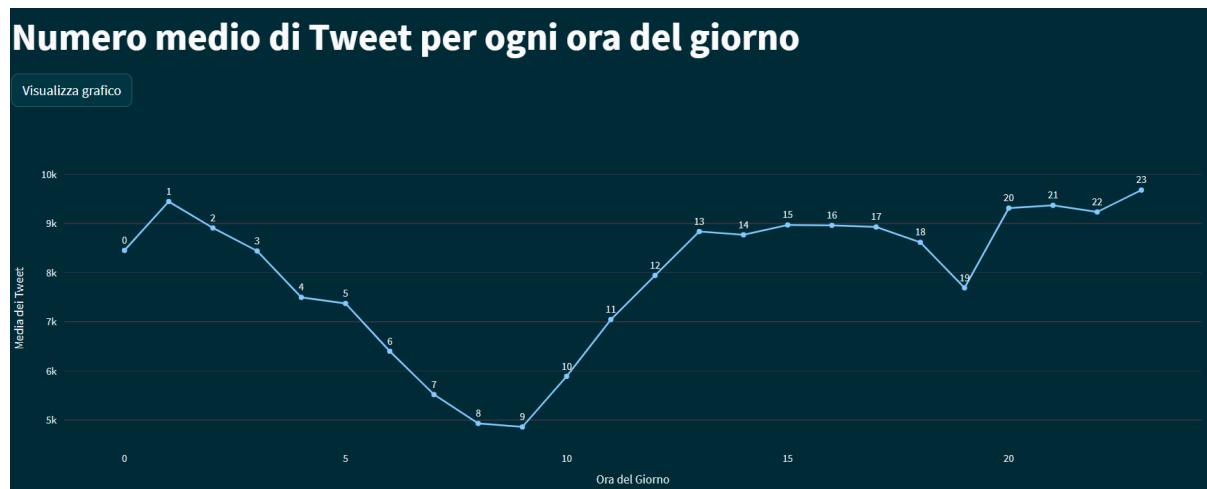


Figure 2: Numero medio dei tweet per ogni ora del giorno

L'output della query è utile per analizzare le fluttuazioni orarie nell'attività dei tweet e identificare le ore del giorno in cui si verifica un aumento o una diminuzione significativa dei tweet. Si può notare un calo significativo della pubblicazione dei tweet dalla mezzanotte in poi fino alle 10 del mattino, orario in cui la quest'ultima riprende. Si stabilizza sui 9mila tweet dalle 13 in poi, fino alle 23.

### 6.1.3 Conteggio dei tweet da parte di utenti verificati

```
1 def verifTweet():
2     verified_tweet_count = df_start.groupBy("user_verified").count() .
3         toPandas()
4     verified_tweet_count = verified_tweet_count.dropna(subset=["
5         user_verified"])
6
7     return verified_tweet_count
```

La funzione `verifTweet()` ha lo scopo di calcolare il numero di tweet pubblicati da utenti verificati e non verificati. Ecco una spiegazione della parte più significativa di essa:

- `verified_tweet_count = df_start.groupBy("user_verified").count():`

I dati vengono raggruppati in base alla colonna "user\_verified", che indica se un utente su Twitter è verificato o no. La funzione `count()` viene quindi applicata a ciascun gruppo per contare il numero di tweet in ciascun gruppo. Questo restituirà un DataFrame Spark con due colonne: "user\_verified" e "count" che indica il numero di tweet verificati e non verificati.

L'output è il seguente:

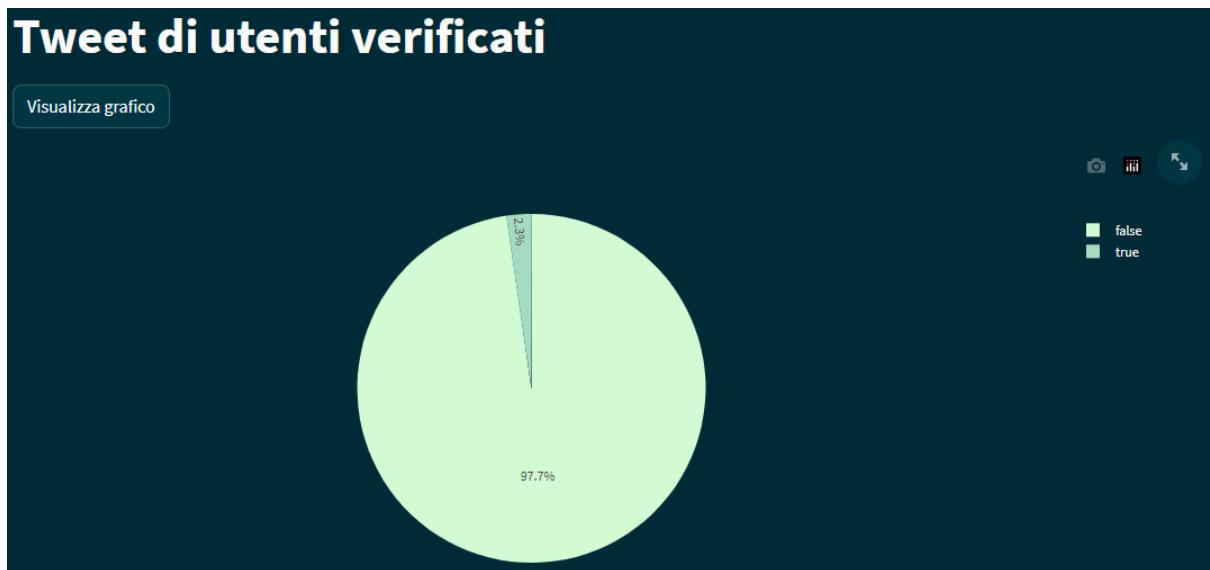


Figure 3: Tweet da parte di utenti verificati e non verificati

Come si può vedere, poco più del 2% dei tweet sono stati pubblicati da utenti verificati (circa 73 mila).

### 6.1.4 Parole più frequenti nei tweet

La funzione `parolePiùFrequenti()` ha lo scopo di identificare e calcolare le parole più frequenti. Questa analisi aiuta a capire quali parole sono più comuni nei tweet.

```
1 def parolePiùFrequenti():
2     # stop words in inglese da NLTK
3     nltk.download("stopwords")
4     default_stop_words = set(stopwords.words("english"))
5     # elenco personalizzato di parole da eliminare
6     custom_stop_words = ["rt", "https", "co", " ", "amp", "like", "1", "go"
7             , "2", "3", "5", "4", "edt", "fl", "mph",
8                 "d6vj7", "ap", "60", "xkcqz4s2ra", "2qoqloeg2", "
9                     kqsptnr4ox"]
9
10    all_stop_words = default_stop_words.union(custom_stop_words)
```

```

1 # testo in minuscolo
2 words_df = df_start.select("text").withColumn("words", split(lower("text"), r'\W+'))
3 # Espande l'array di parole in singole righe ed escludere le stop
4 # words
5 word_df = words_df.select(explode("words").alias("word")).filter(~col("word").isin(all_stop_words))
6 word_df = word_df.filter(col("word") != "")
7 # Calcolo le parole più frequenti
8 word_counts = word_df.groupBy("word").count().orderBy(col("count").desc()).limit(20)
9 word_counts_pandas = word_counts.toPandas()
10 word_counts_pandas = word_counts_pandas.iloc[::-1]
11 return word_counts_pandas

```

Segue una spiegazione dettagliata di alcune parti del codice:

- `nltk.download("stopwords")`:

In questa parte, si scaricano le "stop words" in inglese da NLTK (Natural Language Toolkit), che sono parole comuni che vengono spesso rimosse durante l'analisi del testo poiché non portano informazioni significative.

- `default_stop_words = set(stopwords.words("english"))`:

Creazione di un insieme di "stop words" predefinite in inglese utilizzando NLTK.

- `custom_stop_words = ["rt", "https", "co", " ", "amp", "like", "1", "go", "2", "3", "5", "4", "edt", "f1", "mph", "d6vj7", "ap", "60", "xkczqz4s2ra", "2qoqlhoege2", "kqsptnr4ox"]` :

Creazione di un elenco personalizzato di parole che desideriamo eliminare dall'analisi dei tweet. Questo elenco contiene parole come "rt," "https," "co," ecc.

L'output conterrà le 20 parole più frequenti nei tweet, insieme al loro conteggio. Questo può essere utilizzato per ulteriori analisi o visualizzazioni delle parole più comuni nei tweet.

L'output visualizzato è il seguente:

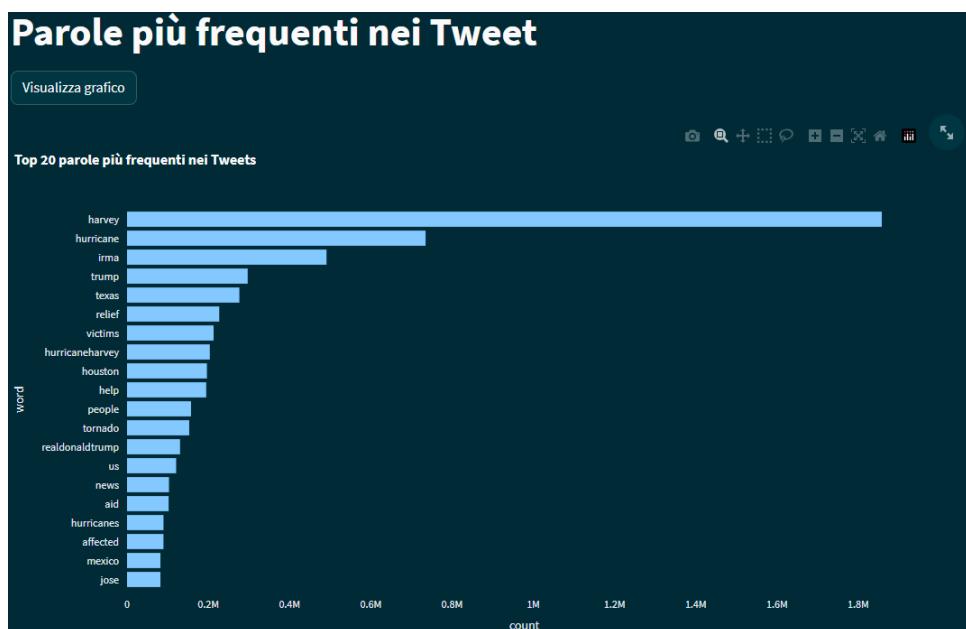


Figure 4: Parole più frequenti nei tweet

Dalla figura è possibile notare come le parole più frequenti effettivamente siano legate al contesto del disastro, infatti tra le più utilizzate nei tweet risultano esserci **"harvey"** con una frequenza di utilizzo che supera i 1.8 milioni e **"hurricane"** con una frequenza di utilizzo che si aggira intorno a 800 mila.

### 6.1.5 Hashtag più frequenti

La funzione `hashtagPiuFrequenti()` ha lo scopo di identificare e calcolare gli hashtag più frequenti all'interno dei tweet, oggetto di studio. Gli hashtag sono parole o frasi precedute dal simbolo "#" e vengono spesso utilizzati per etichettare o categorizzare i contenuti sui social media.

```
1 def hashtagPiuFrequenti():
2     hash_df = df_start.select(explode("hashtags").alias("hashtag"))
3     hash_df = hash_df.withColumn("words", split(lower(col("hashtag")), ","))
4
5     hash_df = hash_df.select(explode("words").alias("word"))
6     word_counts_h = hash_df.groupBy("word").count().orderBy(col("count").desc()).limit(15)
7     word_counts_pandas_h = word_counts_h.toPandas()
8     word_counts_pandas_h = word_counts_pandas_h.iloc[::-1]
9
10    return word_counts_pandas_h
```

Segue una spiegazione della parte più significativa del codice:

- `hash_df = hash_df.withColumn("words", split(lower(col("hashtag")), ","))`:  
Suddivide il contenuto della colonna "hashtag" in parole minuscole e separa le parole utilizzando la virgola come delimitatore. Questo passaggio è utile per separare gli hashtag multipli presenti in una singola cella.

L'output visualizzato è il seguente:

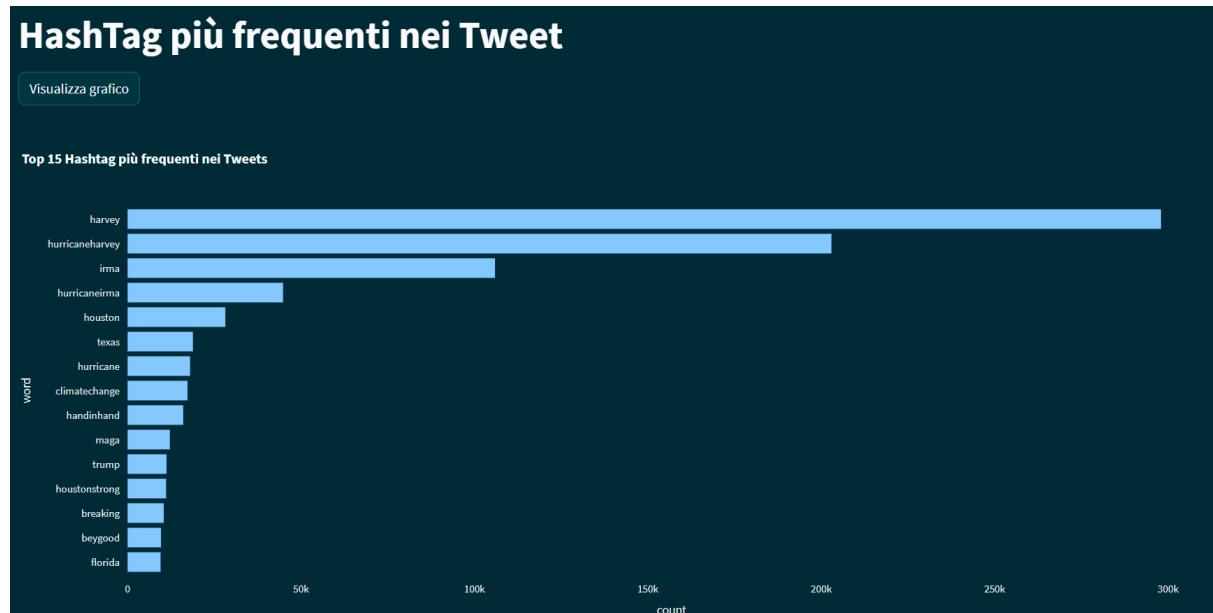


Figure 5: Hashtag più frequenti nei tweet

Dalla figura è possibile notare come gli hashtag più frequenti effettivamente siano **"harvey"** con una frequenza di utilizzo che supera i 300mila e **"hurricaneharvey"** con una frequenza di utilizzo che si aggira intorno a 200 mila.

### 6.1.6 Conteggio dei tweet per ogni tipologia

La funzione `labelTweet()` ha lo scopo di conteggiare le etichette AIDRLabel assegnate ai tweet.

```
1 def labelTweet():
2     label_counts = df_combined.groupBy("AIDRLabel").count().orderBy(col("count").desc())
3     label_count_pandas= label_counts.toPandas()
4
5     return label_count_pandas
```

```

• label_counts = df_combined
    .groupBy("AIDRLabel")
    .count()
    .orderBy(col("count").desc()):

```

Si usa il metodo `groupBy("AIDRLabel").count()` per raggruppare i dati in base alle etichette (`AIDRLabel`) e quindi si procede al conteggio di ciascuna etichetta. I risultati sono ordinati in ordine discendente.

La funzione restituirà le etichette presenti nei tweet e il numero di volte in cui ciascuna etichetta è stata assegnata. Segue l'output:

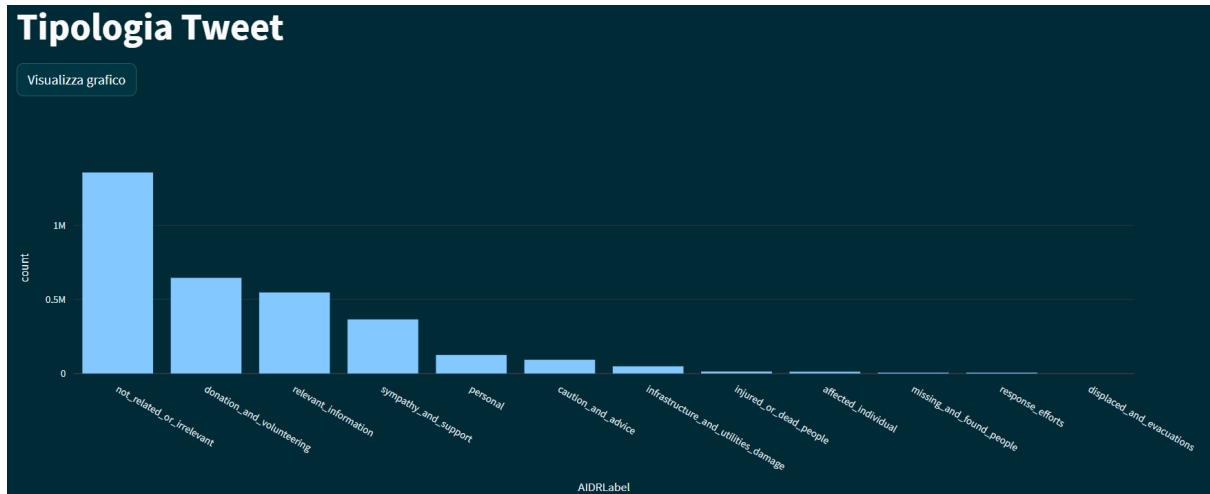


Figure 6: Conteggio dei tweet per ogni tipologia

Dalla figura è possibile notare che il maggior numero di tweet analizzati risulta appartenere alle seguenti tipologie: `not_related_or_irrelevant` con un conteggio pari a circa 1.3 milioni e `donating_and_volunteering` con un conteggio pari a 645 mila.

#### 6.1.7 Parole più frequenti per ogni tipologia di tweet

La funzione `paroleFrequentiPerTipologia()` ha lo scopo di estrarre le parole più frequenti per ciascuna tipologia di tweet, basandosi sulle etichette (`AIDRLabel`) assegnate ai tweet. Le diverse tipologie di tweet includono, ad esempio, "not\_related\_or\_irrelevant", "donation\_and\_volunteering", "relevant\_information" e altre. Segue il codice con spiegazione:

```

1 def paroleFrequentiPerTipologia():
2
3     nltk.download("stopwords")
4
5     # estraggo text per ogni tipologia
6     label_not_related_or_irrelevant = df_combined.filter(col("AIDRLabel")
7                 == "not_related_or_irrelevant").select("text")
8     label_donation_and_volunteering = df_combined.filter(col("AIDRLabel")
9                 == "donation_and_volunteering").select("text")
10    label_relevant_information = df_combined.filter(col("AIDRLabel") ==
11                 "relevant_information").select("text")
12    ...

```

```

• label_not_related_or_irrelevant = df_combined
    .filter(col("AIDRLabel")=="not_related_or_irrelevant")
    .select("text"):

```

Estrae i tweet associati alla tipologia "not\_related\_or\_irrelevant". Quindi, seleziona solo la colonna "text" da questi tweet.

- Si ripete questo processo per ciascuna delle altre tipologie di tweet elencate. Ad esempio, per la tipologia "donation\_and\_volunteering", si estraggono i tweet corrispondenti e si seleziona solo la colonna "text".

La funzione **parolePiuFrequenti(data\_frame)** ha lo scopo di calcolare le parole più frequenti all'interno di **data\_frame**, passato come argomento. Il suo funzionamento è simile alla funzione utilizzata nella sezione 6.1.4.

```

1 def parolePiuFrequenti(df):
2     default_stop_words = set(stopwords.words("english"))
3     custom_stop_words = ["rt", "https", "co", " ", "amp", "like",
4     "1", "go", "2", "3", "5", "4", "25", "21", "25th", "21st", "26th",
5     "com", "26", "54", "190", "b", "000", "u", "edt", "fl", "mph", "d6vj7",
6     "daca", "ap", "60", "xkcqz4s2ra", "2qoqloge2", "kqsptnr4ox", "rumqxwyfkb"]
7     all_stop_words = default_stop_words.union(custom_stop_words)
8     words_df = df.select("text").withColumn("words", split(lower("text"),
9     r'\W+'))
10    word_df = words_df.select(explode("words").alias("word")).filter(~col(
11        "word").isin(all_stop_words))
12    word_df = word_df.filter(col("word") != "")
13    word_counts = word_df.groupBy("word").count().orderBy(col("count").
14        desc()).limit(20)
15    word_counts_pandas = word_counts.toPandas()
16    word_counts_pandas = word_counts_pandas.iloc[::-1]
17
18    return word_counts_pandas

```

La seguente porzione di codice utilizza la funzione **parolePiuFrequenti()** per calcolare le parole più frequenti per ciascuna tipologia di tweet.

```

1 # calcolo parole più frequenti
2 label_not_related_or_irrelevant= parolePiuFrequenti(
3     label_not_related_or_irrelevant)
4 label_donation_and_volunteering= parolePiuFrequenti(
5     label_donation_and_volunteering)
6 label_relevant_information = parolePiuFrequenti(
7     label_relevant_information)
8 ...
9
10 return label_not_related_or_irrelevant,label_donation_and_volunteering
11 ...

```

L'output visualizzato è il seguente e prevede per ogni gruppo la top 20 di parole più frequenti per ogni tipologia:

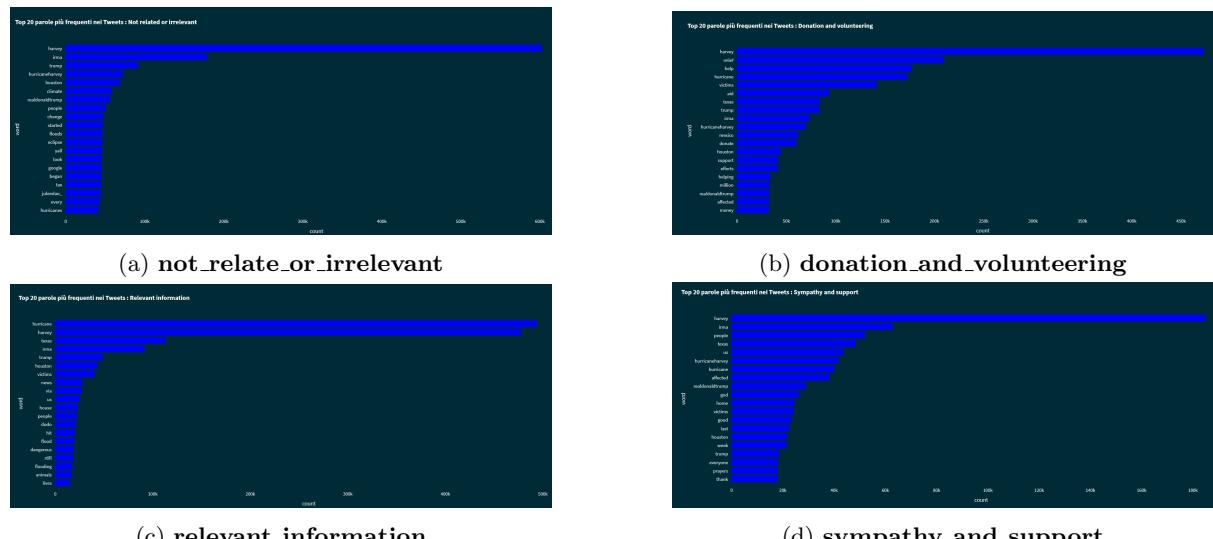


Figure 7: Primo gruppo

Per semplicità di descrizione riportiamo le tre parole più frequenti per ogni tipologia:

- **not\_relate\_or\_irrelevant:**

- **hurricane:** 602.823
- **irma:** 179.247
- **trump:** 92.079

- **donation\_and\_volunteering:**

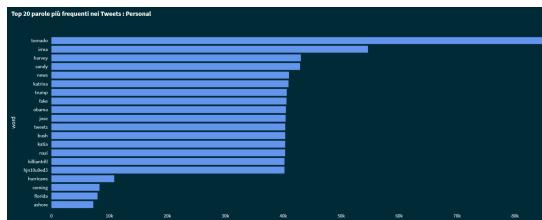
- **harvey:** 473.276
- **relief:** 209.614
- **help:** 177.221

- **relevant\_information:**

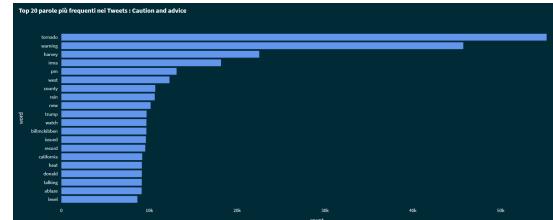
- **hurricane:** 494.118
- **harvey:** 477.969
- **texas:** 113.757

- **sympathy\_and\_support:**

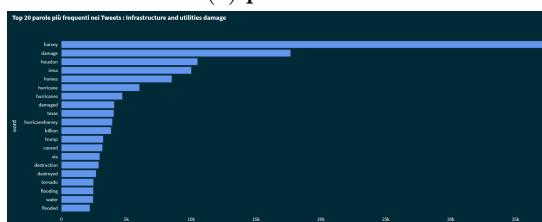
- **harvey:** 185.031
- **irma:** 63.033
- **people:** 52.128



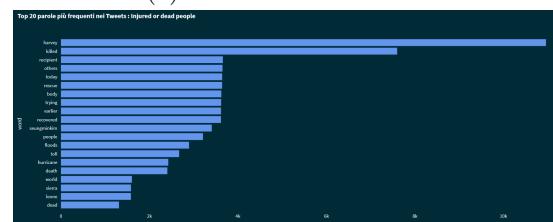
(a) personal



(b) caution\_and\_advice



(c) infrastructure\_and\_utilities\_damage



(d) injured\_or\_dead\_people

Figure 8: Secondo gruppo

- **personal:**

- **tornado:** 85.118
- **irma:** 54.630
- **harvey:** 43.049

- **caution\_and\_advice:**

- **tornado:** 55.222
- **warning:** 45.738
- **harvey:** 22.522

- **infrastructure\_and\_utilities\_damage:**
  - **harvey:** 37.078
  - **damage:** 17.659
  - **houston:** 10.488
- **injured\_or\_dead\_people:**
  - **harvey:** 10.961
  - **killed:** 7598
  - **recipient:** 3659

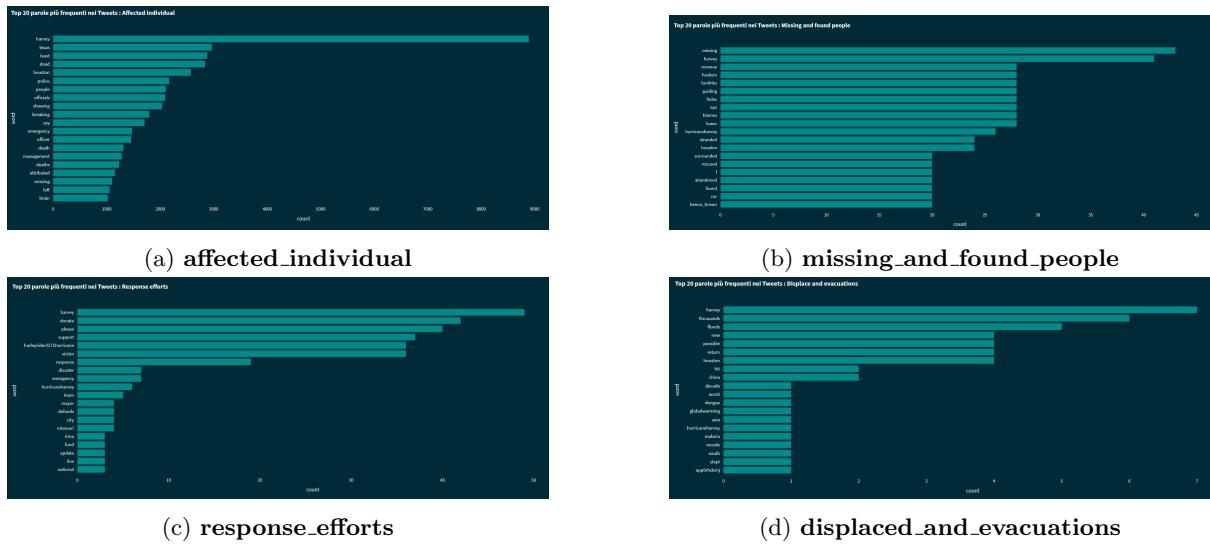


Figure 9: Terzo gruppo

- **affected\_individual:**
  - **harvey:** 8889
  - **texas:** 2965
  - **least:** 2878
- **missing\_and\_found\_people:**
  - **missing:** 43
  - **harvey:** 41
  - **revenue:** 28
- **response\_efforts:**
  - **harvey:** 49
  - **donate:** 42
  - **please:** 40
- **displaced\_and\_evacuations:**
  - **harvey:** 7
  - **thousands:** 6
  - **floods:** 5

### 6.1.8 Conteggio dei tweet nei Stati più colpiti (Texas, Louisiana)

La funzione `posColpiti()` è stata progettata per analizzare i dati al fine di quantificare il numero di tweet pubblicati dagli Stati più colpiti.

```
1 def posColpiti():
2     most = df_combined.filter(trim(col("tweet_place")).isNotNull())
3     most = most.withColumn("nation", split(most["tweet_place"], ",") [1])
4     filter_state = most.filter((col("nation").contains("TX")) | (col("nation").contains("LA")))
5
6     filter_state = filter_state.groupBy("nation").count().orderBy("count")
7     filter_state = filter_state.toPandas()
8
9     return filter_state
```

Segue l'output:

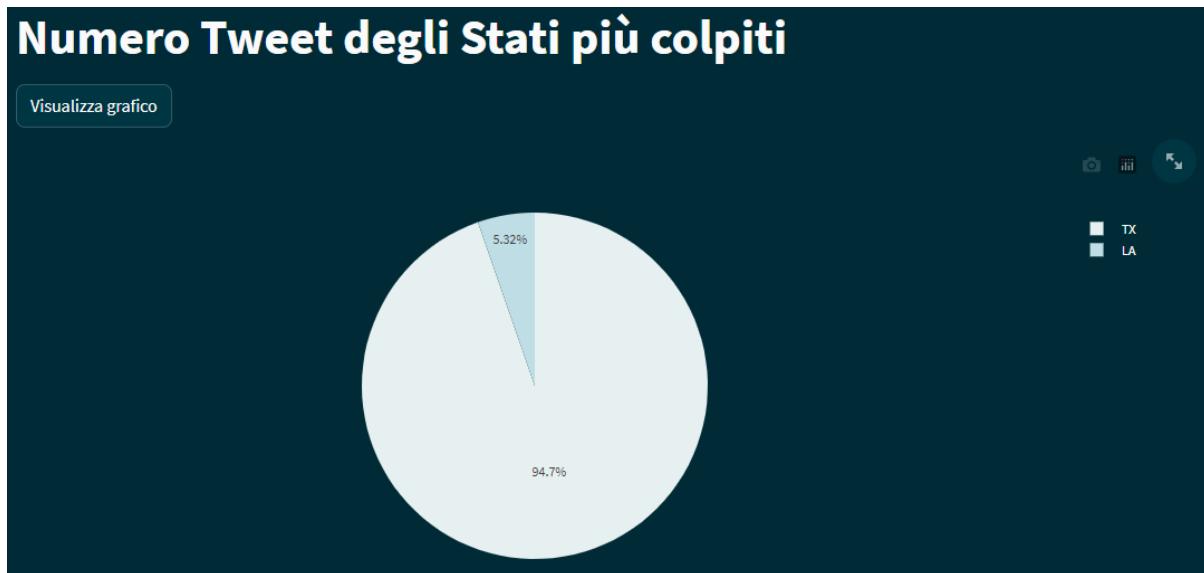


Figure 10: Conteggio dei tweet negli Stati più colpiti (Texas, Louisiana)

Dalla figura si nota che è dallo stato del Texas che sono stati pubblicati la quasi totalità, circa il 94% (7841), dei tweet riguardanti i due stati più colpiti dall'uragano.

### 6.1.9 Tweet Geolocalizzati

Questa funzione è progettata per elaborare dati relativi a tweet geolocalizzati e punti di danno causati dall'uragano, al fine di prepararli per la visualizzazione su una mappa.

```
1 def geoTweet():
2     dfLocalizzati = df_start.filter(col("tweet_pos").isNotNull())
3     # Conversione delle coordinate dei tweet in coordinate Basemap
4     tweet_lons = dfLocalizzati.select(col("tweet_pos")[1].alias("longitude"))
5     .toPandas()["longitude"].tolist()
6     tweet_lats = dfLocalizzati.select(col("tweet_pos")[0].alias("latitude"))
7     .toPandas()["latitude"].tolist()
8     # Posizioni di danni causati dall'uragano
9     damage_locations = [
10         {"latitude": 24.4, "longitude": -93.6, "description": "Damage 1"},
11         {"latitude": 25.0, "longitude": -94.4, "description": "Damage 2"},
12         {"latitude": 25.6, "longitude": -95.1, "description": "Damage 3"},  
13         ...
14     ]
```

```

1  tweet_data = pd.DataFrame({
2      "lat": tweet_lats,
3      "lon": tweet_lons,
4      "type": ["Tweet"] * len(tweet_lats),
5      "color": [(0, 0, 255)] * len(tweet_lats) # Blu per i tweet
6  })
7  damage_data = pd.DataFrame({
8      "lat": [location["latitude"] for location in damage_locations],
9      "lon": [location["longitude"] for location in damage_locations],
10     "type": ["Damage"] * len(damage_locations),
11     "color": [(255, 0, 0)] * len(damage_locations) # Rosso per i
12         punti di danno
13  })
14  # Unire i due DataFrames in uno solo
15  combined_data = pd.concat([tweet_data, damage_data], ignore_index=True
16  )
17
18  return combined_data

```

Di seguito sono riportati i passaggi chiave della funzione:

- **Filtraggio dei dati:** La funzione inizia filtrando il dataframe `df_start` per selezionare solo le righe contenenti informazioni sulla posizione dei tweet.
- **Conversione delle coordinate:** Le coordinate di latitudine e longitudine vengono estratte dai tweet geolocalizzati e memorizzate in due elenchi separati, `tweet_lats` e `tweet_lons`.
- **Definizione dei punti di danno:** Viene creato un elenco di dizionari denominato `damage_locations`, contenente informazioni sulla latitudine, longitudine e una descrizione di diversi punti di danno causati da un uragano.
- **Creazione dei dataframe:** I dati dei tweet e dei punti di danno vengono convertiti in dataframe separati. Ciascun dataframe contiene colonne per latitudine (`lat`), longitudine (`lon`), tipo (`type`) e colore (`color`). I tweet sono contrassegnati come "Tweet" e colorati in blu, mentre i punti di danno sono contrassegnati come "Damage" e colorati in rosso.
- **Unione:** Infine, i due dataframe vengono uniti e forniti in output dalla funzione.

Dai risultati ottenuti è bene specificare che non tutti i tweet a disposizione sono geo-localizzati, ma solo alcuni di essi. In particolare, sono pari a 2787 che risulta essere circa il **0.001%** del totale. Segue l'output:

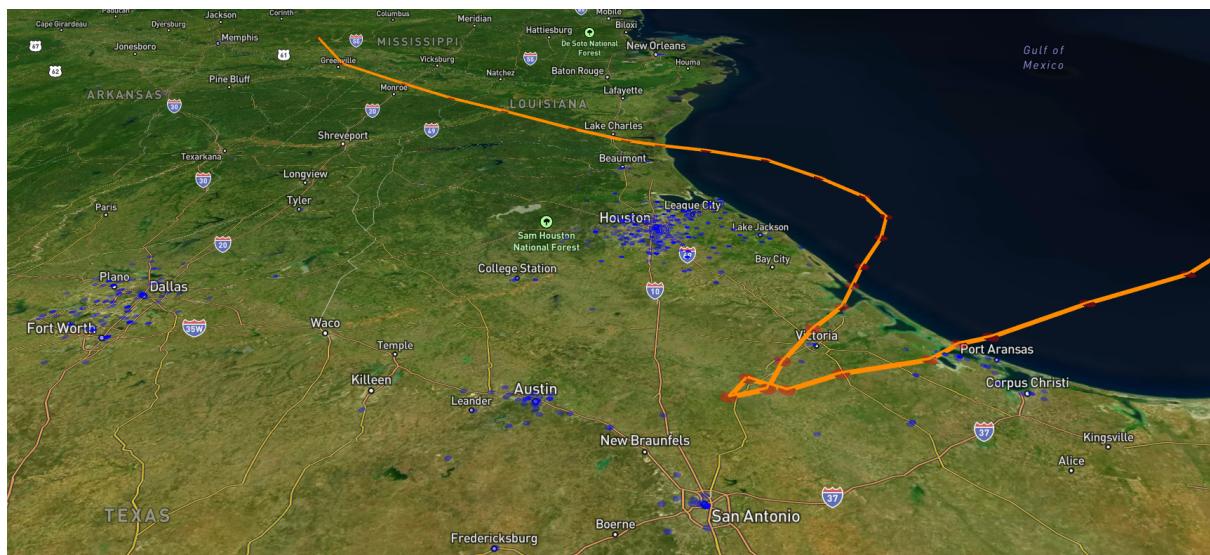


Figure 11: Vista laterale

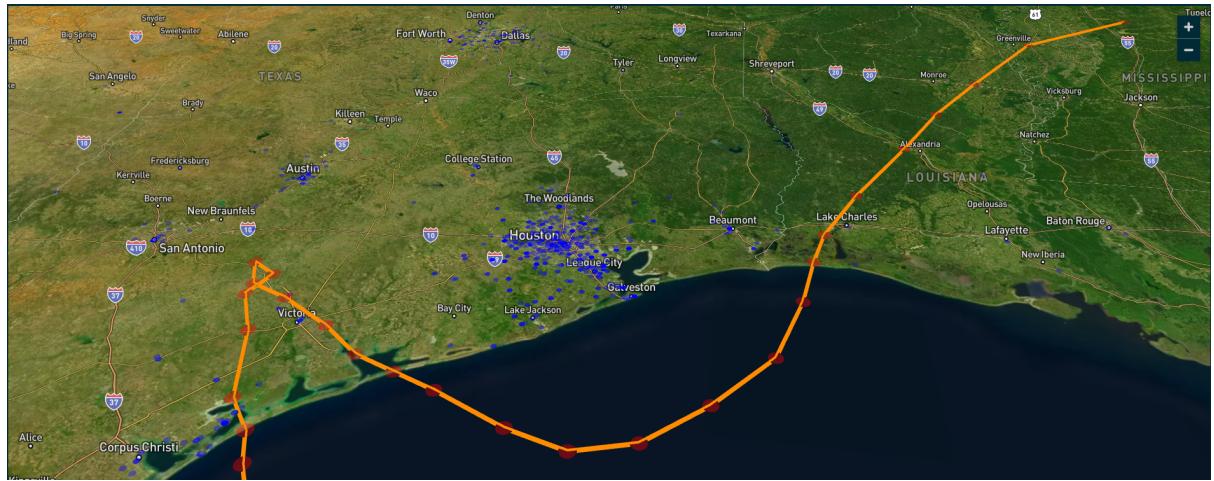


Figure 12: Vista dall'alto

Si è deciso di sovrapporre anche la traiettoria seguita da **Harvey** (consultabile al seguente link: Hurricane Harvey 2017 Data Archive) alla mappa con i tweet geolocalizzati, in questo modo è possibile osservare come le zone in cui sono stati registrati danni da parte dell'uragano siano per la maggior parte le stesse in cui si sono concentrati il maggior numero dei tweet pubblicati. Questo risultato, abbastanza rappresentativo delle zone colpite, è stato ottenuto nonostante il numero relativamente basso di tweet geolocalizzati a disposizione.

Porzione del codice, più significativo, del frontend che ha reso possibile la visualizzazione della query in esame, è il seguente:

```

1 # Layer tweet
2 tweet_layer = pdk.Layer(
3     'ScatterplotLayer',
4     data=tweet_data,
5     get_position='[lon, lat]',
6     get_color='[0, 0, 255, 100]', # Tweet in blu
7     get_radius=2000,
8     pickable=True,
9     auto_highlight=True,
10    filled=True,
11    filled_color="[0, 0, 255, 255]",
12    outline=True,
13    outline_color="[0, 0, 255, 255]",
14    get_tooltip='Tweet'
15 )
16 layers.append(tweet_layer)
17
18 # Layer punti di danno
19 damage_layer = pdk.Layer(
20     'ScatterplotLayer',
21     data=damage_data,
22     get_position='[lon, lat]',
23     get_color='[255, 0, 0, 100]',
24     get_radius=5000,
25     pickable=True,
26     auto_highlight=True,
27     filled=True,
28     filled_color="[255, 0, 0, 255]",
29     outline=True,
30     outline_color="[255, 0, 0, 255]",
31     get_tooltip='Demage'
32 )
33 layers.append(damage_layer)

```

```

1   # Layer traiettoria
2   line_layer = pdk.Layer(
3       "LineLayer",
4       paths,
5       get_source_position="start",
6       get_target_position="end",
7       get_color=[255, 140, 0],
8       get_width=5,
9       highlight_color=[255, 255, 0],
10      picking_radius=10,
11      auto_highlight=True,
12      pickable=True,
13      )
14  layers.append(line_layer)

```

Il codice prevede l'utilizzo della libreria **PyDeck** che consente di creare layer in cui inserire oggetti eterogenei. In questo caso sono stati creati tre layer:

- **Layer Tweet**: dedicato alla visualizzazione dei tweet.
- **Layer Punti di Danno**: dedicato alla visualizzazione dei punti di danno causati dall'uragano.
- **Layer traiettoria**: dedicato alla visualizzazione della traiettoria compiuta da Harvey.

## 7 Query con l'utilizzo del Machine Learning

Dato l'elevato numero di tweet a disposizione si è pensato di utilizzare alcuni algoritmi di machine learning.

### 7.1 Sentiment Analisys

La funzione `sentiment_Vader()` utilizza l'analizzatore di sentiment VADER (Valence Aware Dictionary and sEntiment Reasoner) per calcolare il sentiment dei tweet presenti nel DataFrame `df_start`.

Segue la spiegazione di alcuni passaggi più importanti del codice:

- Viene scaricato il lexicon di VADER utilizzando `nltk.download("vader_lexicon")` per preparare l'analizzatore di sentiment, ovvero `SentimentIntensityAnalyzer()`.

```

1 def sentiment_Vader():
2     nltk.download("vader_lexicon")
3     # Inizializza il SentimentIntensityAnalyzer di NLTK
4     sia = SentimentIntensityAnalyzer()
5     tweets = df_start.select("text")
6     tweets = tweets.filter(col("text").isNotNull())

```

- Viene definita la funzione chiamata `analyze_sentiment_vader()` che prende in input il testo di un tweet, calcola i punteggi di sentiment utilizzando VADER e restituisce una stringa di "sentiment" (positivo, negativo o neutro) e un valore "compound" che rappresenta il punteggio complessivo di sentiment.

```

1 # Definizione della funzione per analizzare il sentiment dei tweet
2 def analyze_sentiment_vader(text):
3     sentiment_scores = sia.polarity_scores(text)
4     # Assegna il sentiment in base al compound score di VADER
5     if sentiment_scores['compound'] >= 0.05:
6         sentiment = 'positive'
7     elif sentiment_scores['compound'] <= -0.05:
8         sentiment = 'negative'
9     else:
10        sentiment = 'neutral'
11    return sentiment, sentiment_scores['compound']

```

- `analyze_sentiment_vader()` viene quindi applicata ai tweet, presenti nel DataFrame `tweets` utilizzando

```
.withColumn("sentiment-score", analyze_sentiment_vader_udf(col("text")))
```

generando una nuova colonna chiamata "sentiment-score" che contiene sia il "sentiment" che il "compound score".

```

1  analyze_sentiment_vader_udf = udf(analyze_sentiment_vader, StructType([
2      StructField("sentiment", StringType()),
3      StructField("compound", DoubleType())
4 ]))
5  # Applicazione della funzione ai tweet per ottenere la colonna del
6  # sentiment
7  tweets_with_sentiment_vader = tweets.withColumn("sentiment-score",
8      analyze_sentiment_vader_udf(col("text")))
9  tweets_with_sent_final = tweets_with_sentiment_vader.withColumn("sentiment",
10     col("sentiment-score.sentiment"))
11 tweets_with_sent_final = tweets_with_sent_final.withColumn("score", col(
12     "sentiment-score.compound"))
13 tweets_with_sent_final = tweets_with_sent_final.drop("sentiment-score")

```

- Viene calcolata la distribuzione dei sentiment utilizzando

```
.groupBy("sentiment").count().orderBy(col("count").desc()).collect().
```

- Vengono selezionati campioni di tweet positivi, negativi e neutri utilizzando

```
t_pos= tweets_with_sent_final.filter(col("sentiment") == "positive"),
t_neg= tweets_with_sent_final.filter(col("sentiment") == "negative"),
t_neu= tweets_with_sent_final.filter(col("sentiment") == "neutral"),
```

e vengono restituiti i primi 5 tweet di ciascuna categoria.

```

1  # Calcola la distribuzione dei sentiment
2  sentiment_counts = tweets_with_sent_final.groupBy("sentiment").count() .
3      orderBy(col("count").desc()).collect()
4  # Sample per ogni tipo di sentiment
5  t_pos= tweets_with_sent_final.filter(col("sentiment") == "positive").limit
6      (5)
7  t_neg= tweets_with_sent_final.filter(col("sentiment") == "negative").limit
8      (5)
9  t_neu= tweets_with_sent_final.filter(col("sentiment") == "neutral").limit
10     (5)
11 t_pos = t_pos.toPandas()
12 t_neg = t_neg.toPandas()
13 t_neu = t_neu.toPandas()
14
15 return sentiment_counts, t_pos, t_neg, t_neu

```

La query **restituisce** la distribuzione dei sentiment e i campioni di tweet positivi, negativi e neutri. Segue l'output:

Sentiment Analysis dei Tweet			
		Visualizza grafico	
	Positive	Negative	Neutral
0	text		
0	RT @fema: Texas: If you're under a tornado warning, seek shelter right away. Be safe. #Harvey #houwx #txwx https://t.co/SbCErgTHfv	positive	0.1280
1	RT @RepPeteKing: Ted Cruz & Texas cohorts voted vs NY/NJ aid after Sandy but I'll vote 4 Harvey aid. NY wont abandon Texas. 1 bad turn does...	positive	0.7835
2	RT @AlexandriaKHOU: This is Allen Parkway in front of our studios. Everyone please don't drive. #KHOU11 #Houston #HurricaneHarvey https://t...	positive	0.3182
3	RT @dallasnews: How you can help: Cribs, clothes and blood donations needed #HurricaneHarvey https://t.co/8bjERY7Di8 https://t.co/ZQm2bCyP...	positive	0.4019
4	RT @laryssa_ellia: we really need to be thankful 4 all the doctors, nurses, cops, firefighters, and any other first responders @ times...	positive	0.5719

Figure 13: Tabella con Tweet del Sentiment Analisys

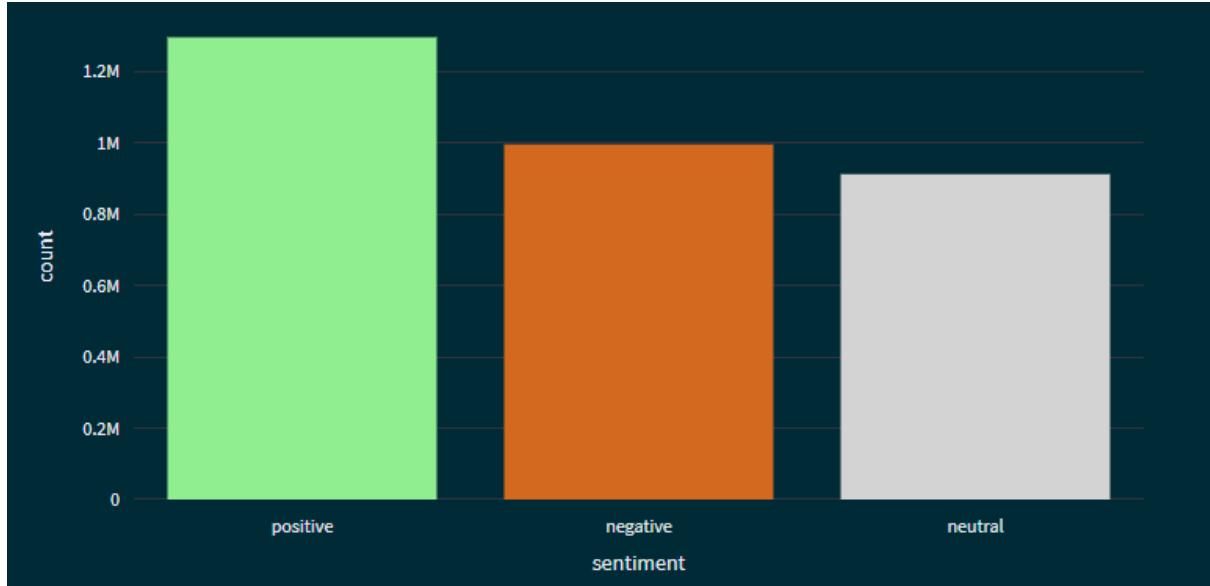


Figure 14: Distribuzione dei Sentiment

Si può osservare che la distribuzione dei **sentiment** risulta essere abbastanza bilanciata. Tuttavia è possibile notare che i tweet **"positive"** presentano uno scarto di circa 200mila rispetto agli altri.

## 7.2 Sentiment Analisys dei tweet degli Stati più colpiti

La funzione **sentimentPosColpiti()** utilizza l'analizzatore di sentiment VADER per calcolare il sentiment dei tweet relativi ai paesi più colpiti. Il processo avviene attraverso gli stessi passaggi descritti per la query precedente. La differenza sta nella porzione di tweet analizzati, ovvero quelli provenienti dagli stati del Texas e della Louisiana.

Segue l'output:

### Sentiment Analysis dei Tweet dei Paesi più colpiti

[Visualizza grafico](#)

Positive   Negative   Neutral

	text	sentiment	score
0	Y'all my auntie is having a Harvey Party , she got strobe lights y'all and everything ó2	positive	0.4019
1	Ok so we all were like "lol ok hurricane whatever" and then Harvey got mad. #rainraingoaway	positive	0.6705
2	A number of our friends (longtime Houstonians) are having water enter their homes for the first time ever. #HurricaneHarvey	positive	0.5267
3	1st time reporting in my hometown. Proud to see army of volunteers helping #Harvey flood victims #Houstonstrong <a href="https://t.co/O79QRwF5SG">https://t.co/O79QRwF5SG</a>	positive	0.4588
4	Don't forget to donate to the surrounding cities affected by #Harvey !!!! <a href="https://t.co/ap0eTIMPxz">https://t.co/ap0eTIMPxz</a>	positive	0.3036

Figure 15: Tabella con Tweet del Sentiment Analisys negli Stati più colpiti

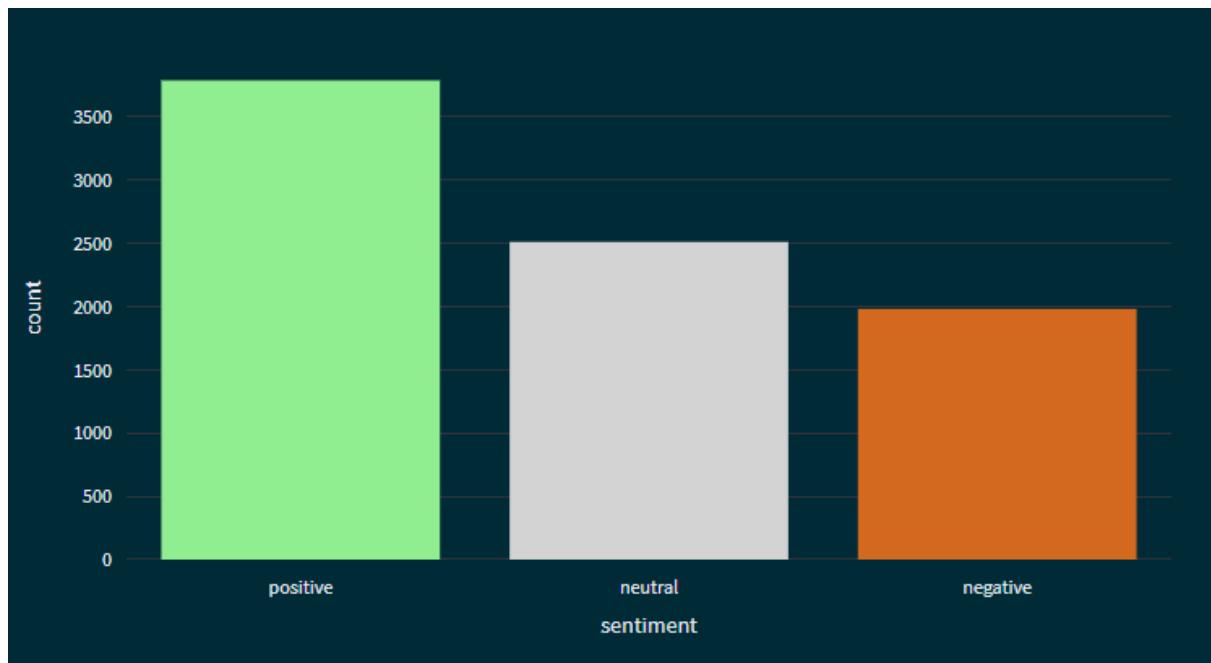


Figure 16: Distribuzione dei Sentiment negli Stati più colpiti

Si può osservare che la distribuzione dei `sentiment` in questo caso risulta essere sbilanciata. In particolare, è possibile notare che tra la categoria "neutral" e "negative" si ha uno scarto di circa 500, mentre tra "positive" e "neutral" più di 1000.

### 7.3 Matrice di correlazione

La funzione `correlazione()` è responsabile per il calcolo della matrice di correlazione tra le feature del DataFrame `df_combined`. Il processo avviene attraverso i seguenti passaggi:

- Vengono selezionate le colonne del DataFrame `df_combined` contenenti le feature rilevanti per il calcolo della correlazione. In questo caso, le colonne selezionate includono "user\_followers," "user\_verified," "AIDRLabel," e "AIDRConfidence."

```

1 def correlazione():
2     pre_enc= df_combined. select("user_followers","user_verified",
3         "AIDRLabel", "AIDRConfidence")
4     pre_enc = pre_enc.toPandas()

```

- Viene eseguito l'encoding delle feature per consentire il calcolo della correlazione. La colonna "user\_verified" viene convertita in una nuova colonna "user\_verified\_index," in cui il valore "true" viene mappato a 1 e tutti gli altri valori a 0. La colonna "AIDRConfidence" viene convertita in una nuova colonna "AIDRConfidence\_numeric" di tipo DoubleType.
- Viene utilizzato StringIndexer per mappare i valori della colonna "AIDRLabel" in una nuova colonna "AIDRLabel\_index" di tipo numerico.

```

1 # Encoding delle feature
2 indexed_df = df_combined.withColumn("user_verified_index", when(col("user_verified") == "true", 1).otherwise(0))
3 indexed_df = indexed_df.withColumn("AIDRConfidence_numeric", col("AIDRConfidence").cast("double"))
4 indexer = StringIndexer(inputCol="AIDRLabel", outputCol="AIDRLabel_index")
5 indexed_df = indexer.fit(indexed_df).transform(indexed_df)

```

- Vengono selezionate le colonne rilevanti per il calcolo della correlazione, inclusi i nuovi campi creati: "user\_followers," "AIDRLabel\_index," "AIDRConfidence\_numeric," e "user\_verified\_index."
- Viene utilizzato VectorAssembler per creare un vettore di features chiamato "features" che contiene le colonne selezionate.
- Il DataFrame risultante, chiamato assembled\_df, contiene ora il vettore di features "features" oltre alle colonne originali.

```

1 # Seleziona le colonne rilevanti per il calcolo della correlazione
2 selected_cols = ["user_followers", "AIDRLabel_index", "AIDRConfidence_numeric", "user_verified_index"]
3
4 # VectorAssembler per creare un vettore di features
5 assembler = VectorAssembler(inputCols=selected_cols, outputCol="features")
6 assembled_df = assembler.transform(indexed_df)
7
8 post_enc= assembled_df. select("user_followers", "user_verified_index", "AIDRLabel_index", "AIDRConfidence_numeric")
9 post_enc = post_enc.toPandas()

```

- Viene calcolata la matrice di correlazione tra le feature utilizzando la funzione Correlation.corr() e specificando il vettore di features "features" come input. Il risultato è una matrice di correlazione.
- La matrice di correlazione viene quindi estratta come un array e convertita in un DataFrame Spark chiamato corr\_matrix\_df.
- Il DataFrame corr\_matrix\_df viene convertito, utilizzando corr\_matrix\_df.corr(), in un DataFrame Pandas chiamato corr\_matrix\_pearson che contiene la matrice di correlazione tra le feature. Questo passaggio è utile per poter visualizzare meglio la matrice.

```

1 # Calcola la matrice di correlazione
2 correlation_matrix = Correlation.corr(assembled_df, "features").head()
3 corr_matrix = correlation_matrix[0].toArray()
4 corr_matrix_df = spark.createDataFrame(corr_matrix.tolist(), selected_cols)
5 corr_matrix_df = pd.DataFrame(corr_matrix, columns=selected_cols)
6 corr_matrix_pearson = corr_matrix_df.corr()
7
8 return corr_matrix_pearson, pre_enc, post_enc

```

La funzione fornisce la matrice di correlazione tra le features di interesse, consentendo l'analisi delle relazioni tra le variabili. Questo è utile per comprendere come le feature influenzano l'obiettivo della analisi. Segue l'output della funzione:

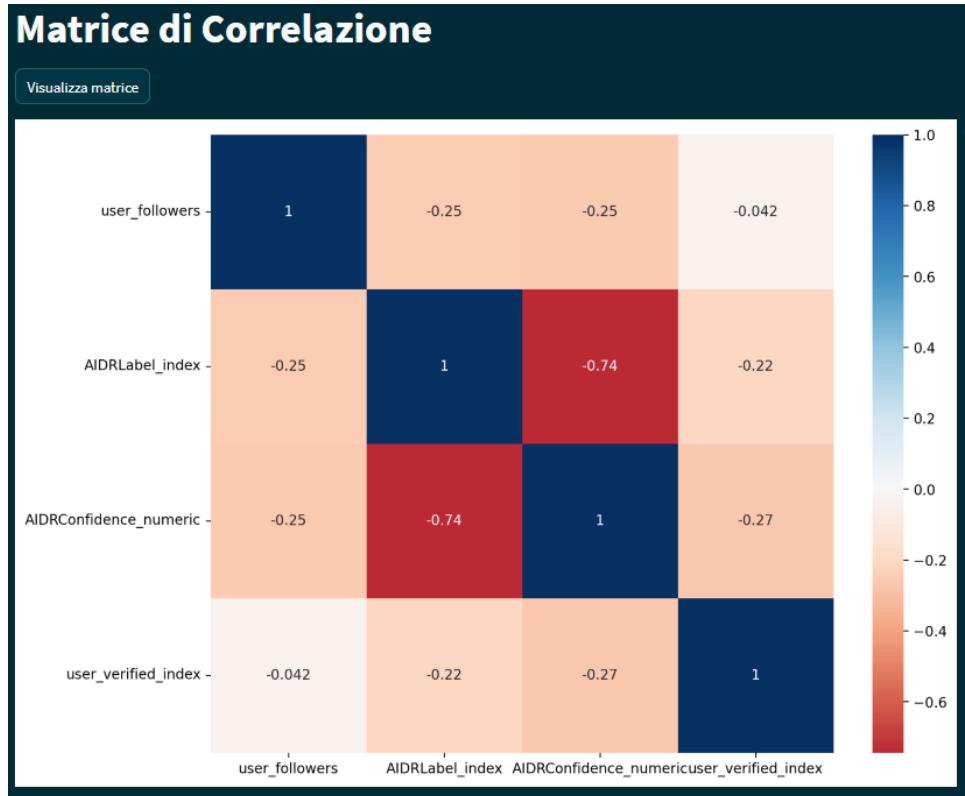


Figure 17: Matrice di Correlazione

Dall'analisi dei risultati ottenuti dalla matrice di correlazione si può notare come le feature abbiano una moderata correlazione con la variabile target. Per questo motivo si è deciso di utilizzare tali attributi per l'addestramento dei modelli di machine learning. In particolare si può vedere come la variabile target **"AIDR\_label\_index"** abbia una correlazione inversa con le altre feature.

## 7.4 Classificazione con Random Forest

Con il seguente codice, viene eseguita la classificazione multiclassa su **"AIDR\_label\_index"** utilizzando l'algoritmo Random Forest. Di seguito sono riportati i passaggi specifici:

- La prima parte è di encoding e segue la stessa procedura vista nella sezione dedicata alla creazione della matrice di correlazione.

```

1 def classification_RandomForest():
2     # Encoding delle feature
3     indexed_df = df_combined.withColumn("user_verified_index", when(col("user_verified") == "true", 1).otherwise(0))
4     indexed_df = indexed_df.withColumn("AIDRConfidence_numeric", col("AIDRConfidence").cast("double"))
5     indexer = StringIndexer(inputCol="AIDRLabel", outputCol="AIDRLabel_index")
6     indexed_df = indexer.fit(indexed_df).transform(indexed_df)
7
8     # Seleziona le colonne rilevanti
9     selected_cols = ["user_followers", "AIDRLabel_index", "AIDRConfidence_numeric", "user_verified_index"]
10    df_class= indexed_df.select(selected_cols)

```

- Viene utilizzato `VectorAssembler` per creare un vettore di features chiamato "features" a partire dalle colonne selezionate. Questo vettore sarà utilizzato come input per l'addestramento del modello.
- Il dataset viene suddiviso in set di addestramento (60%), di validazione (20%) e di test (20%) utilizzando `randomSplit()`.
- Viene calcolata la distribuzione delle etichette di classificazione utilizzando `groupBy()` e `count()`. Questo fornisce un quadro della distribuzione delle etichette nel dataset.
- Viene inizializzato il classificatore Random Forest con `RandomForestClassifier`, specificando il numero di alberi (`numTrees`) come 100 e il seme (`seed`) come 123.
- Il modello Random Forest viene addestrato sui dati di addestramento utilizzando `fit()`.

```

1 #VectorAssembler per creare un vettore di features
2 assembler = VectorAssembler(inputCols=selected_cols, outputCol="features")
3 assembled_df = assembler.transform(df_class)
4
5 # Split dataset in training, validation, test sets
6 train_data, val_data, test_data = assembled_df.randomSplit([0.6, 0.2,
7     0.2], seed=123)
8
9 # Calcolo distribuzione etichette
10 assembled_df_labels = assembled_df.groupBy("AIDRLLabel_index").count().
11     orderBy("AIDRLLabel_index")
12 assembled_df_labels = assembled_df_labels.toPandas()
13
14 # Initializzaione RandomForestClassifier
15 rf_classifier = RandomForestClassifier(featuresCol="features", labelCol="AIDRLLabel_index", numTrees=100, seed=123)
16
17 # Train su training data
18 rf_model = rf_classifier.fit(train_data)

```

- Le predizioni vengono effettuate sui set di validazione e di test e vengono calcolate le rispettive accuratezze (`val_accuracy` e `test_accuracy`) utilizzando `MulticlassClassificationEvaluator`.
- Viene creato un RDD contenente le predizioni e le etichette reali per ulteriori calcoli delle metriche.

```

1 # Predizione su validation set
2 val_predictions = rf_model.transform(val_data)
3 # Valutazione
4 evaluator = MulticlassClassificationEvaluator(labelCol="AIDRLLabel_index",
5     predictionCol="prediction", metricName="accuracy")
6 # Accuracy
7 val_accuracy = evaluator.evaluate(val_predictions)
8 # Predizione su test set
9 test_predictions = rf_model.transform(test_data)
10 # Accuracy su test set
11 test_accuracy = evaluator.evaluate(test_predictions)
12 # DataFrame con Predizioni
13 test_predictions_and_labels_rdd = test_predictions.select("prediction",
14     "AIDRLLabel_index").rdd
15 metrics = MulticlassMetrics(test_predictions_and_labels_rdd)
16 labels = test_predictions_and_labels_rdd.map(lambda x: x[1]).distinct()
17 .collect()

```

- Vengono calcolate le metriche di classificazione come la precision, la recall e l'F1-score per ciascuna etichetta di classe. Queste metriche vengono raccolte in un DataFrame Pandas chiamato `metrics_df`.

- Vengono calcolate le metriche generali come la precision media pesata (`overall_precision`), la recall media pesata (`overall_recall`) e l'accuratezza generale (`overall_accuracy`). Queste metriche vengono raccolte in un DataFrame Pandas chiamato `overall_metrics_df`.

```

1  metrics_data = []
2  label_to_index = {
3      'not_related_or_irrelevant': 0,
4      'donation_and_volunteering':1,
5      'relevant_information':2,
6      'sympathy_and_support': 3,
7      'personal':4,
8      'caution_and_advice':5,
9      'infrastructure_and_utilities_damage':6,
10     'injured_or_dead_people':7,
11     'affected_individual':8,
12     'missing_and_found_people':9,
13     'response_efforts':10
14 }
15    for label in labels:
16        label_precision = metrics.precision(label)
17        label_recall = metrics.recall(label)
18        label_f1_score = metrics.fMeasure(label)
19
20        label_name = [key for key, value in label_to_index.items() if
21                      value == label][0]
22
23        metrics_data.append({"Label": label_name, "Precision":label_precision, "Recall": label_recall, "F1-Score":label_f1_score})
24
25    metrics_df = pd.DataFrame(metrics_data, columns=["Label", "Precision",
26                                              "Recall", "F1-Score"])
27    # Calcolo delle metriche generali
28    overall_precision = metrics.weightedPrecision
29    overall_recall = metrics.weightedRecall
30    overall_accuracy = metrics.accuracy
31    # Creazione del DataFrame per le metriche generali
32    overall_metrics_df = pd.DataFrame({
33        "Metrica": ["Precision", "Recall", "Accuracy"],
34        "Valore": [overall_precision, overall_recall, overall_accuracy]
35    })
36
37    return val_accuracy, test_accuracy,overall_metrics_df,
38          assembled_df_labels , metrics_df

```

Questo codice esegue la classificazione utilizzando l'algoritmo Random Forest e fornisce una valutazione dettagliata delle prestazioni del modello.

#### 7.4.1 Valutazione delle prestazioni

Segue la distribuzione delle etichette, costruita con "AIDRLabel\_index":

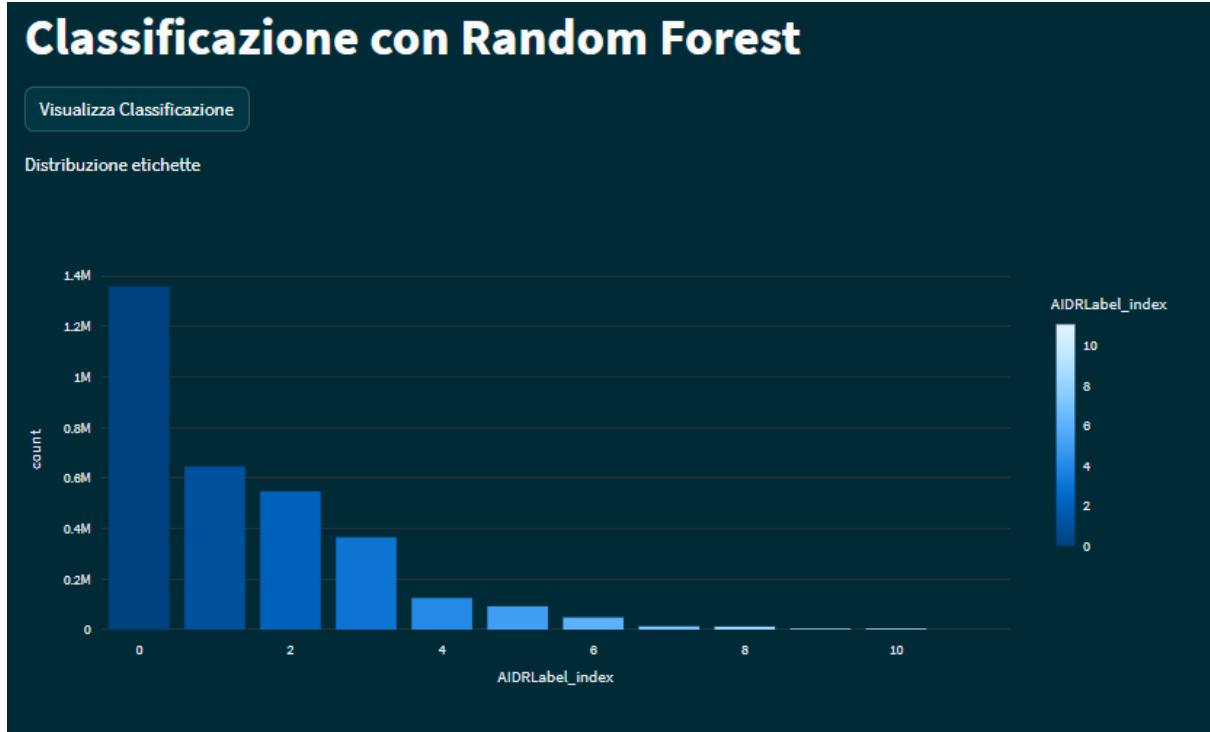


Figure 18: Distribuzione etichette

Segue l'output della classificazione:

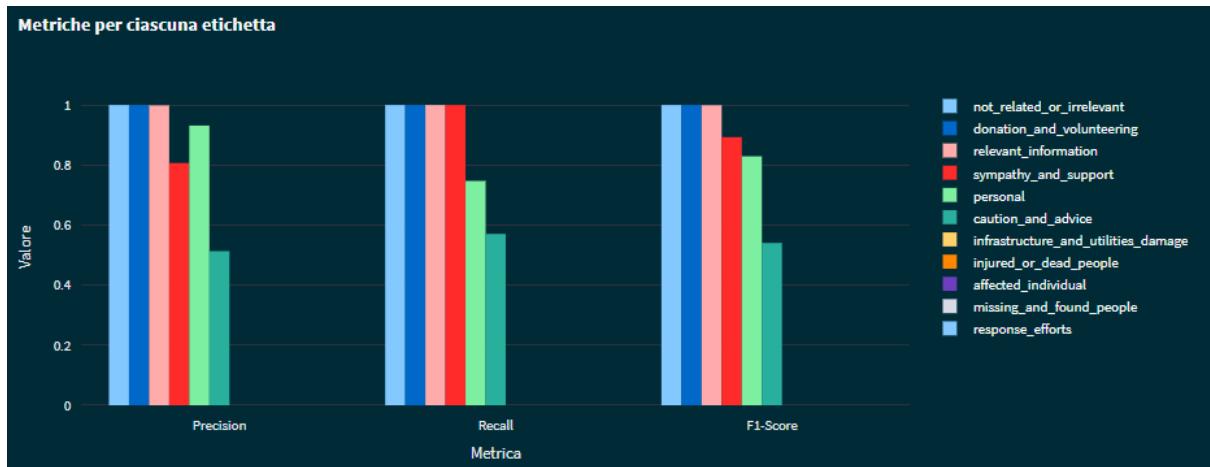


Figure 19: Metriche con Random Forest

Analizzando i risultati si può notare come l'algoritmo del Random Forest non sia stato in grado di classificare alcune etichette. In particolare, queste risultano essere le seguenti:

- infrastructure\_and\_utilities\_damage
- injured\_or\_dead\_people
- affected\_individual

- `missing_and_found_people`
- `response_efforts`

La spiegazione può essere individuata nel fatto che i tweet etichettati con le suddette non erano presenti in una quantità adeguata da consentire un corretto addestramento da parte dell'algoritmo. Di conseguenza, il modello risultante non è riuscito a performare. Per le restanti etichette si può osservare che il modello riesce a classificare abbastanza bene ad eccezione delle etichette **"personal"** e **"caution\_and\_advice"**.

## 7.5 Classificazione con Regressione Logistica

Si è deciso di effettuare il processo di classificazione utilizzando anche l'algoritmo di Regressione Logistica. In questo contesto, viene applicata come tecnica alternativa e complementare rispetto all'algoritmo Random Forest per la classificazione dei tweet in base alle categorie di interesse.

Il codice utilizzato per questo approccio è molto simile, pertanto non verrà riportato completamente l'intera funzione. La differenza sostanziale è da vedere nella porzione di codice la seguente:

```

1 # Inizializzazione LogisticRegression
2 lr_classifier = LogisticRegression(featuresCol="features", labelCol="AIDRLabel_index")
3 # Creazione del pipeline
4 pipeline = Pipeline(stages=[lr_classifier])
5 # Addestramento del modello su training data
6 lr_model = pipeline.fit(train_data)

```

### 7.5.1 Valutazione delle prestazioni

Segue la distribuzione delle etichette, costruita con **"AIDRLabel\_index"**:

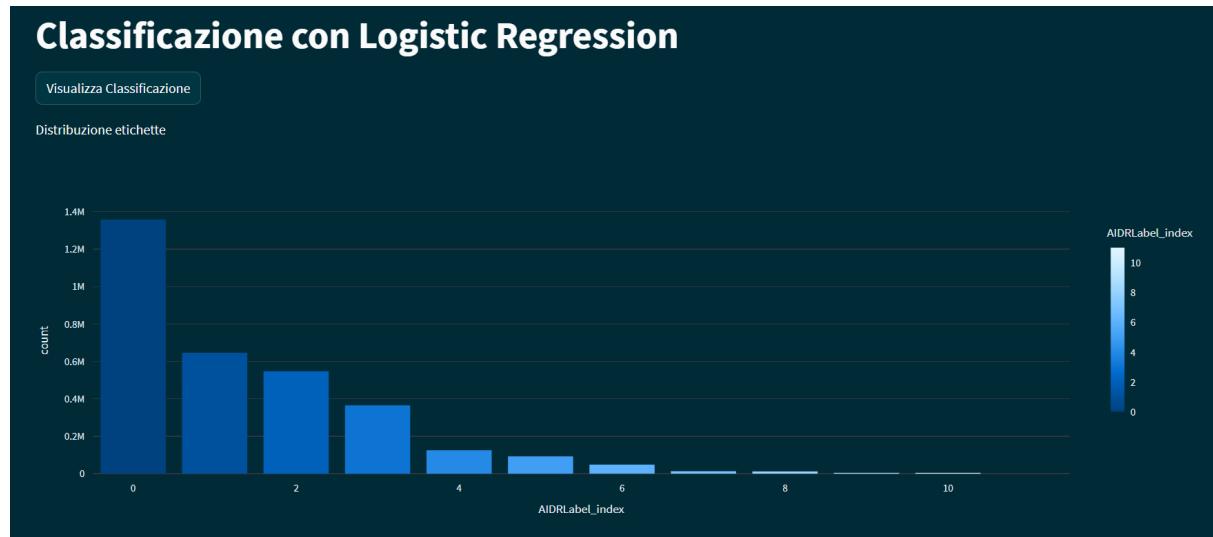


Figure 20: Distribuzione etichette

Segue l'output, utile per la valutazione delle valutazioni:

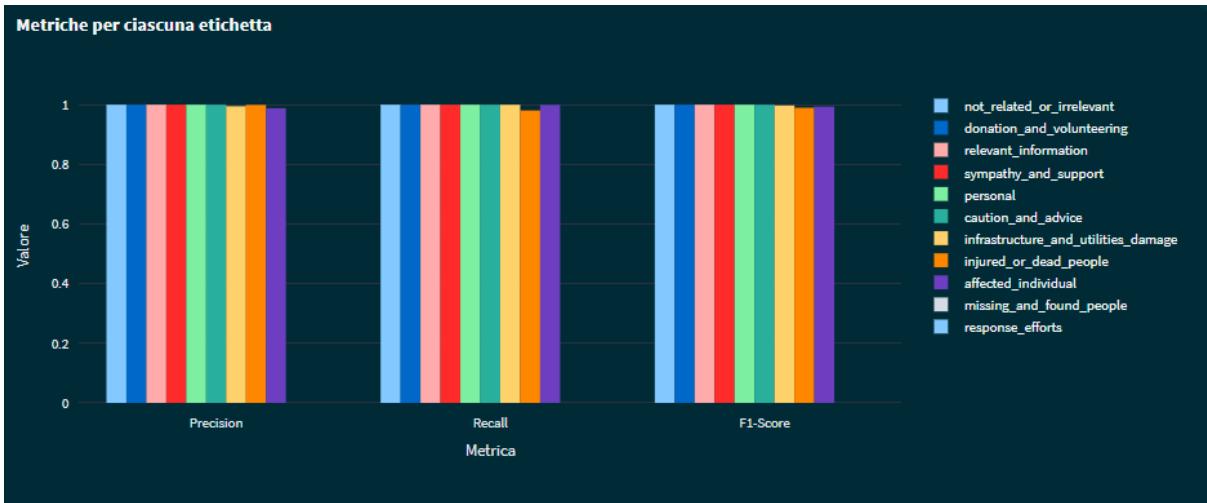


Figure 21: Metriches con Logistic Regression

Analizzando i risultati si può notare come l'algoritmo del Logistic Regression non sia stato in grado di classificare, anche in questo caso (in misura minore) alcune etichette. In particolare, queste risultano essere le seguenti:

- **missing\_and\_found\_people**
- **response\_efforts**

La spiegazione può essere individuata nel fatto che i tweet etichettati con le suddette non erano presenti in una quantità adeguata da consentire un corretto addestramento da parte dell'algoritmo. Di conseguenza, il modello risultante non è riuscito a performare su queste due etichette.

Per le restanti etichette si può osservare come il modello sia riuscito a classificare molto bene.

## 8 Conclusioni

L'analisi dei tweet e dei risultati ottenuti dalle nostre query ha portato alla luce una serie di conclusioni sostanziali sull'uso di Twitter come piattaforma di monitoraggio delle tendenze, delle discussioni pubbliche e delle opinioni degli utenti. Un aspetto che risalta con chiarezza è il modo in cui eventi ad alto impatto sociale, come quello oggetto della nostra analisi, possono suscitare reazioni significative all'interno della comunità online.

Questo fenomeno riveste particolare importanza, poiché offre un'opportunità unica per comprendere le dinamiche dell'opinione pubblica e i comportamenti di una vasta porzione della popolazione. Utilizzando queste informazioni, è possibile sviluppare strumenti di risposta più efficaci e implementare soluzioni mirate. In sostanza, l'analisi dei tweet ci consente di mettere a frutto la potenza delle piattaforme sociali per migliorare la nostra capacità di comprensione e reazione di fronte a eventi di rilevanza sociale.

Questi risultati sottolineano il ruolo cruciale di Twitter e di altre piattaforme simili come fonti di dati preziosi per l'analisi delle tendenze sociali, non solo per scopi accademici ma anche per applicazioni pratiche, come la gestione delle emergenze, la definizione delle strategie di comunicazione e la creazione di politiche pubbliche più informate.

Inoltre, la sentiment analysis dei tweet ha dimostrato la sua utilità nell'identificare come il pubblico reagisce a eventi specifici o argomenti di emergenza pubblica, anche se le opinioni possono variare ampiamente. Gli hashtag si sono rivelati uno strumento cruciale per categorizzare e organizzare i tweet, contribuendo a un maggiore coinvolgimento e amplificando la visibilità di un tweet.

L'analisi geografica ha rivelato tendenze regionali nelle discussioni, evidenziando come determinati argomenti possano interessare diverse aree piuttosto che altre.

In conclusione, Twitter si è dimostrato estremamente sensibile agli eventi di attualità, con discussioni che si sviluppano rapidamente in risposta a crisi globali o notizie di rilievo.