Università della Calabria

Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica

---

Corso di Laurea in Ingegneria Informatica

Progetto Big Data

Analisi del dataset di tweet relativi all'uragano Harvey

Studenti

Francesco Arci [252372]

Niccolò Bossio [257091]

Anno Accademico 2023-2024

Indice

1 Introduzione **2**
1.1 Uragano Harvey . . . 2
1.2 Dataset . . . 2
1.3 Struttura dell'applicazione . . . 3

2 Descrizione delle query **5**
2.1 Query 1: Numero di tweet per categoria e distribuzione dei tweet giornaliera . . . 5
2.2 Query 2: Hashtags più ricorrenti . . . 6
2.3 Query 3: Tweet con media . . . 7
2.4 Query 4: Parola più usata per ogni categoria . . . 8
2.5 Query 5: Le parole più usate nei tweet e la loro occorrenza nelle categorie . . . 11
2.6 Query 6: Gli utenti che hanno postato più tweet per categorie . . . 12
2.7 Query 7: Gli utenti più menzionati nei tweet . . . 13
2.8 Query 8: Gli utenti più influenti che hanno postato almeno un tweet . . . 14
2.9 Query 9: Andamento giornaliero degli Hashtags . . . 15

3 Descrizione del modello di classificazione **17**
3.1 Pre-processing dei dati . . . 17
3.2 Addestramento e test del modello . . . 18
3.3 Confronto tra modelli . . . 20

1 Introduzione

In questo lavoro, dopo una breve introduzione sull'uragano Harvey e sulla composizione del dataset di interesse, si analizzano i tweet relativi all'uragano descrivendo le diverse queries fatte sul dataset stesso.
Infine, si presenta il modello di classificazione binaria creato ed i procedimenti di pre-processing ed esecuzione.

1.1 Uragano Harvey

L'uragano Harvey è stato uno dei più devastanti uragani ad aver colpito gli Stati Uniti negli ultimi decenni, causando danni estesi, inondazioni catastrofiche e perdite umane. Harvey si forma il 17 agosto 2017 come una depressione tropicale nell'Atlantico orientale, inizialmente indebolito, si riorganizza nel Golfo del Messico e inizia a intensificarsi rapidamente. Il 25 agosto, diventa un uragano di categoria 4 sulla scala Saffir-Simpson, con venti sostenuti fino a 215 km/h. Tocca terra vicino a Rockport, Texas, il 25 agosto, portando con sé venti forti e piogge torrenziali. Staziona sul Texas per diversi giorni, scaricando quantità eccezionali di pioggia, soprattutto nell'area di Houston e nelle contee circostanti. Alcune aree hanno registrato oltre 1000 mm di pioggia, un record per il Texas. Dopo circa 4 giorni l'uragano comincia a perdere potenza fino ad esaurirsi completamente facendo segnare un bilancio di almeno 68 morti e una stima complessiva di circa 125 miliardi di dollari di danni.

1.2 Dataset

Il dataset di interesse, proveniente dallo studio di ricerca *"A twitter Tale of Three Hurricanes: Harvey, Irma e Maria"*, è composto da tweets raccolti negli Stati Uniti tra il 25 agosto 2017 e il 3 ottobre 2017, raccolti principalmente nella zona colpita dalla catastrofe. Del dataset completo si è deciso, per motivi di risorse hardware, di lavorare con una porzione significativa di questo, considerando circa 1.200.000 tuple, tra il 4 settembre 2017 e l'8 settembre 2017.
Il dataset è costituito da 37 colonne, tuttavia di queste solo 20 sono state ritenute significative all'analisi dei tweet. Infatti, molte colonne rappresentano informazioni generali di un tweet, perciò queste sono state interpretate come poco utili per un'analisi della catasfrofe avvenuta, tramite i tweet. In particolare, le colonne più significative sulle quali è stato svolto il lavoro sono:

- **Text**: contiene il testo del tweet;
- **User**: è la colonna relativa a tutte le informazioni dell'utente di un tweet;
- **Entities**: è una struttura contenente al suo interno diverse informazioni relativi ad un tweet, come gli hashtags, le menzioni un utente, i media ecc...
- **Created_at**: è la data di creazione del tweet.

Oltre alle colonne caratteristiche dell'oggetto tweet, è presente un ulteriore dataset relativo al lavoro svolto dal paper citato precedentemente. Tale dataset contiene una classificazione dei messaggi in base ai loro contenuti tematici. In particolare, la colonna "AIDRLabel", che associa ad ogni tweet una delle seguenti categorie :

- **injured or dead people** : se il tweet parla di persone ferite o rimaste uccise;
- **relevant information** : se il tweet contiene informazioni rilevanti;
- **caution and advice** : se il tweet contiene avvertimenti o avvisi riguardanti il disastro che possono essere utili;
- **displaced and evacuations** : se il tweet contiene informazioni su evacuazioni o persone sfollate;

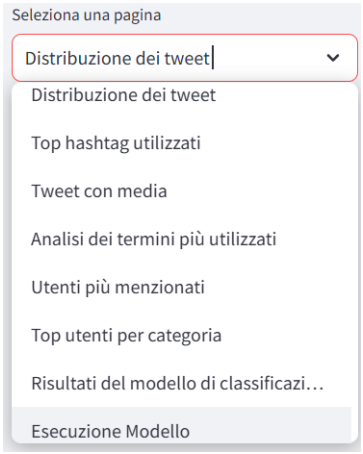
- **sympathy and support** : se il tweet mostra supporto per le vittime;
- **response efforts** : se il tweet è correlato alle risposte di aiuto;
- **infrastructure and utilities damage** : se il tweet riporta danni ad infrastrutture;
- **personal** : se il tweet riporta aggiornamenti personali, soprattutto della situazione e della salute;
- **affected individual** : se il tweet è scritto da persone colpite dalla catastrofe;
- **not related or irrelevant** : se il tweet è poco o per niente rilevante con l'uragano;
- **missing and found people** : se il tweet riporta di persone scomparse o ritrovate;
- **donation and volunteering** : se il tweet contiene richieste o invii di donazioni e volontariato.

1.3 Struttura dell'applicazione

L'intera applicazione è stata sviluppata in Python, usando la libreria Pyspark, che risulta essere una potente soluzione per l'elaborazione distribuita dei dati, consentendo agli sviluppatori di scrivere codice Python ad alte prestazioni per analizzare grandi datataset. Pyspark è una libreria Python per l'integrazione con Apache Spark, un framework di elaborazione progettato per l'analisi di dati su larga scala. Difatti uno dei suoi vantaggi è l'elaborazione dei dati in memoria centrale, in quanto il dataset viene caricato in RAM in strutture quali Dataframe, RDD o Dataset, strutture dati resilienti sulla quale vengono eseguite varie interrogazioni. Utilizzare la RAM, a patto di averne abbastanza, fa sì che si possa effettuare una sola lettura dei dati e successivamente fare qualsiasi tipo di interrogazione direttamente in memoria centrale.

Il frontend dell'applicazione è stato sviluppato sempre in Python, utilizzando il framework open-source Streamlit, per consentire una rappresentazione dei dati leggibile e un'interfaccia user-fiendly tra l'utente e l'applicazione. La scelta di usare Streamlit risiede nella sua semplicità di sviluppo, in quanto consente di realizzare web app senza l'uso di codice CSS, Html e JavaScript. Questo ha permesso di concentrare la nostra attenzione maggiormente sulla manipolazione dei dati e sulla logica delle query. Inoltre fornisce diversi strumenti utili per la visualizzazione dei dati e per tale motivo si è rilevato adatto alla nostra applicazione.
Quest'ultima presenta diversi componenti:

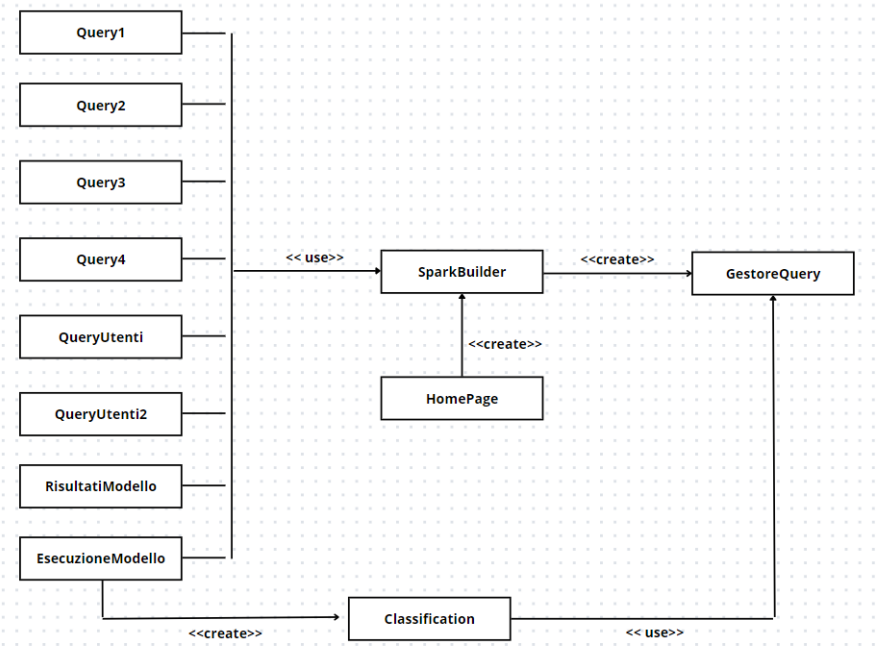
- **SparkBuilder**: è la classe che si occupa di creare una sessione Spark, di inizializzare il dataset, e di crare un oggetto *GestoreQuery*.
- **GestoreQuery**: è la classe che contiene le query usate per la manipolazione del tweet.
- **Classification**: è la classe che si occupa di eseguire la classificazione binaria sul testo.
- **HomePage**: crea un oggetto SparkBuilder e rappresenta il collegamento tra le query e le pagine del frontend. Infatti, le query possono eseguite direttamente dal frontend, e sono state categorizzate in pagine distinte, a seconda del contenuto che restituiscono.



Riassumendo, quindi, la nostra applicazione si suddivide in 3 parti:

- *Analisi dei tweet*: svolta attraverso le query, le quali verranno affrontate nel dettaglio nella sezione 2 della relazione;
- *Classificazione dei tweet*: svolta tramite diverse tecniche di Machine Learning, a seguito di una fase di pre-processing del dataset, descritta nella sezione 3 della relazione;
- *Visualizzazione dei risultati*: realizzato nel Front-End, tramite Streamlit.

Di seguito è riportato uno schema riassuntivo dell'applicazione:



2 Descrizione delle query

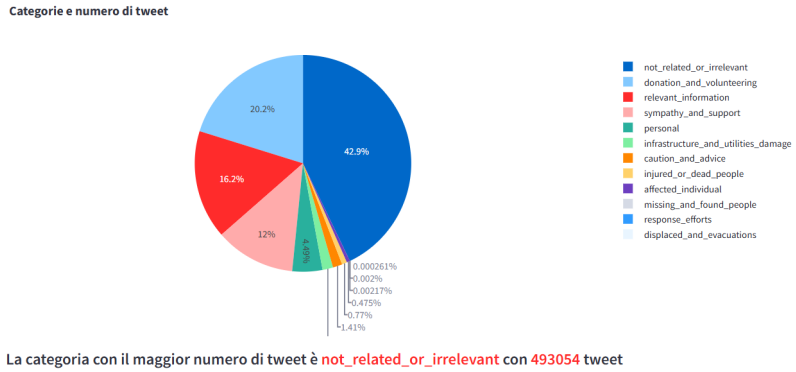
In questa sezione verrà affrontata l'analisi delle query che sono state eseguite sul dataset. In particolare, l'analisi ha voluto evidenziare risultati numerici riguardanti le principali informazioni estraibili dai tweet, più in generale, dai post di un social network. La nostra attenzione, perciò, si è focalizzata principalmente sull'analisi del testo, degli hashtags, degli utenti.

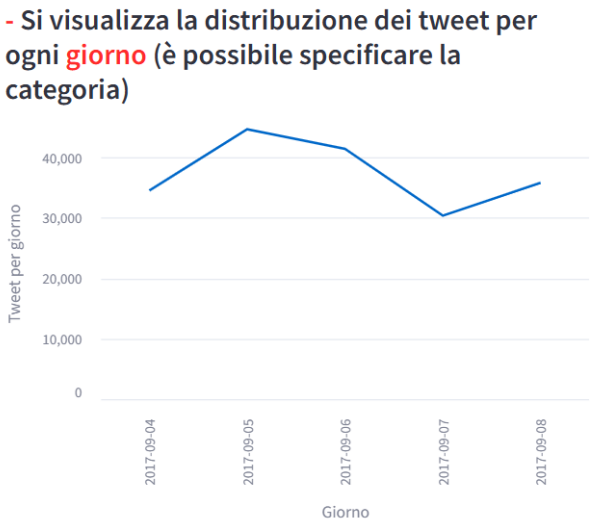
2.1 Query 1: Numero di tweet per categoria e distribuzione dei tweet giornaliera

In questa sezione vengono raggruppate due queries, entrambe relative al numero di tweet. La query *number_of_tweet_for_category(self)* restituisce, per ogni categoria, il numero totale di tweet. La query *tweets_distribution_for_category(self, category)* restituisce, per la categoria passata come argomento, l'andamento temporale del numero di tweet.

```
def number_of_tweet_for_category(self):
        tweets_for_category = self.dataset.groupBy("AIDRLabel")\
        .agg(count("text").alias("Numero di tweet"))
        return tweets_for_category

def tweets_distribution_for_category(self, category):
        ds_data_convertita = self.dataset\
        .withColumn("timestamp", from_unixtime(unix_timestamp(col("
           created_at"), "EEE MMM dd HH:mm:ss ZZZ yyyy")))
        ds_data_AMG = ds_data_convertita.withColumn("Giorno",
           date_format(col("timestamp"), "yyyy-MM-dd"))
        if (category != None):
            risultato = ds_data_AMG.groupBy("AIDRLabel", "Giorno").agg(
               count("*").alias("Tweet per giorno"))\
                                    .filter(col("AIDRLabel") == category
                                       )
        else:
            risultato = ds_data_AMG.groupBy("AIDRLabel", "Giorno").agg(
               count("*").alias("Tweet per giorno"))
        return risultato
```





Nella prima immagine è riportato un grafico a torta (costruito tramite la prima query) che esprime, in percentuale, i tweet relativi ad una determinata categoria nel nostro dataset. Si noti come la categoria maggiormente presente nel dataset sia *not related or irrelevant information*, cioè la categoria che fa riferimento a tweet poco rilevanti.
Nella seconda immagine è riportato l'andamento giornaliero dei tweet relativi alla categoria *relevant information*. Si è scelto di rappresentare la seconda query con un line chart proprio per mostrare l'andamento del risultato.

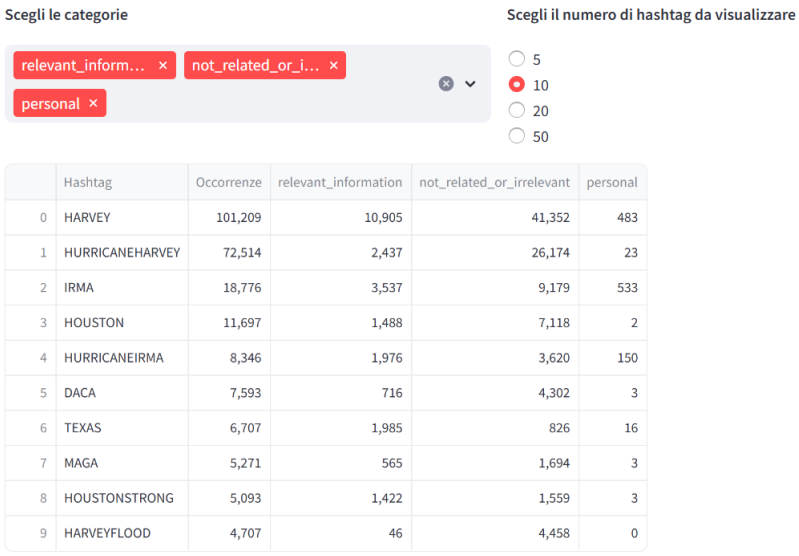
2.2 Query 2: Hashtags più ricorrenti

La seconda query ha lo scopo di restituire i top n hashtags più usati all'interno dei tweet, con il parametro n che può essere selezionato tramite un radio-button nel frontend. Inoltre, per le categorie selezionate, viene riportato il numero di occorrenze nelle categorie di ognuno degli n hashtags. Per ottenere gli hashtags sfruttiamo la colonna entities, che contiene, tra le diverse informazioni di un tweet, anche il numero di hashtags. Contiamo le occorrenze totali di un hashtag, e per ognuno di essi contiamo quante volte occorre nelle categorie passate come parametro.

```
                                                                    1
def top_n_hashtag(self, categories, n=10):                         2
    hashtag = self.dataset.selectExpr("AIDRLabel", "transform(      3
        entities.hashtags,x -> upper(x.text)) as vettore_hashtags")
                                                                    4
    result = hashtag.select(col("AIDRLabel"),explode(col("          5
        vettore_hashtags")).alias("Hashtag"))\
```

```
                .groupBy("Hashtag")\
                .agg(count("Hashtag")\
                .alias("Occorrenze"),*[count(when(col("AIDRLabel") ==
                    label,1)).alias(label) for label in categories])\
                .orderBy(desc("Occorrenze"))\
                .limit(n)

        return  result
```



	Hashtag	Occorrenze	relevant_information	not_related_or_irrelevant	personal
0	HARVEY	101,209	10,905	41,352	483
1	HURRICANEHARVEY	72,514	2,437	26,174	23
2	IRMA	18,776	3,537	9,179	533
3	HOUSTON	11,697	1,488	7,118	2
4	HURRICANEIRMA	8,346	1,976	3,620	150
5	DACA	7,593	716	4,302	3
6	TEXAS	6,707	1,985	826	16
7	MAGA	5,271	565	1,694	3
8	HOUSTONSTRONG	5,093	1,422	1,559	3
9	HARVEYFLOOD	4,707	46	4,458	0

In figura viene mostrata un'applicazione della seguente query. In particolare fissiamo il parametro n a 10 e selezioniamo 3 categorie di interesse. Il risultato ci mostra gli hashtag più usati nell'intero dataset e per ognuno di essi viene restituito quante volte occorre nei tweet etichettati con le categorie scelte. Viene rappresentato in forma tabellare per consentire una visione più chiara del risultato.

2.3 Query 3: Tweet con media

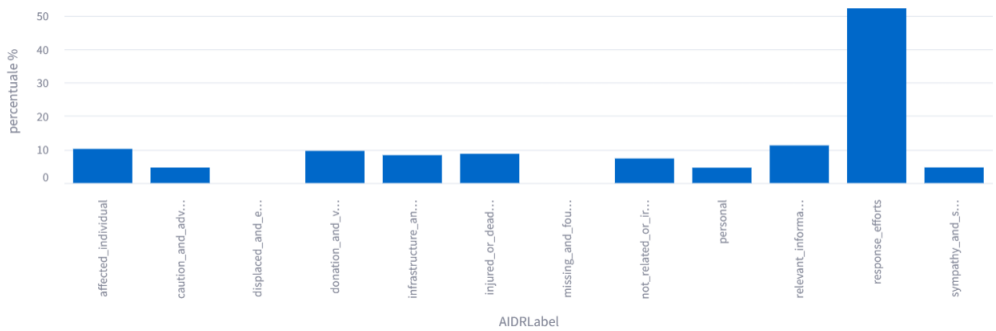
Nella terza query è stata affrontata un'analisi di tipo statistica dei tweet. In particolare, per ogni categoria, viene riportato il numero di tweet totali, il numero di tweet con allegato un contenuto multimediale (foto, video, gif, ecc.) e la sua percentuale rispetto ai tweet totali . Anche in questo caso si è sfruttata la colonna *entities* in modo da ricavare gli utenti che hanno pubblicato un tweet con allegato un contenuto multimediale.

```

def media_for_category(self):
        dataset_tot = self.dataset\
            .groupBy("AIDRLabel")\
            .agg(count("*").alias("Occorrenze totali"))
        dataset_filtrato = self.dataset.filter(col("entities.media").
            isNotNull())\
            .groupBy("AIDRLabel")\
            .agg(count("*").alias("Occorrenze con media"))
        dataset_combinato = dataset_tot.join(dataset_filtrato,
            dataset_filtrato.AIDRLabel == dataset_tot.AIDRLabel, "left")
```

```
        risultato = dataset_combinato.withColumn("percentuale %",  round
            ((col("Occorrenze con media") / col("Occorrenze totali")) *
            100, 2))\
             .na.fill(0, "Occorrenze con media").na.fill(0, "percentuale
                %")
        return risultato.select(dataset_tot["AIDRLabel"], "Occorrenze
            con media", "Occorrenze totali", "percentuale %")
```

	AIDRLabel	Occorrenze con media	Occorrenze totali	percentuale %
0	injured_or_dead_people	774	8,863	8.73
1	relevant_information	20,977	186,770	11.23
2	caution_and_advice	748	16,185	4.62
3	sympathy_and_support	6,401	137,721	4.65
4	infrastructure_and_utilities_damage	1,512	18,175	8.32
5	personal	2,368	51,688	4.58
6	affected_individual	557	5,465	10.19
7	not_related_or_irrelevant	36,160	493,054	7.33
8	donation_and_volunteering	22,294	232,386	9.59
9	response_efforts	12	23	52.17
10	missing_and_found_people	0	25	0
11	displaced_and_evacuations	0	3	0



I risultati ottenuti mostrano la categoria, il numero di tweet totali, il numero di tweet con media e la percentuale dei tweet con media rispetto al totale. E' stata fornita una rappresentazione sia tabellare che con grafico a barre per una visualizzazione più intuitiva.

2.4 Query 4: Parola più usata per ogni categoria

La quarta query fornisce il singolo termine più frequente (nel corpo dei tweets) per ogni categoria. Per poter realizzare questa query è stato necessario creare un metodo preliminare che ha consentito di ripulire il testo del tweet da simboli, caratteri di controllo e non ASCII, punteggiatura, stop words, menzioni di altri utenti, hashtags ed urls:

```

def pre_processing_text(self):
    #Parole che iniziano con @ (nomi utente),parole che iniziano con # (
        hashtag), URL (che iniziano con http:// o https://).
    #Simboli, caratteri di controllo, nuove linee e punteggiatura,la
        stringa RT (indicativo di un retweet),
    #caratteri non ASCII (come emoji e caratteri di altre lingue).
```

```
        regex = r"\B@\w+\b|\B#\w+\b|https?://\S+|[\p{So}\p{Cntrl}\n\p{
            Punct}]+|RT|[^\x00-\x7F]+"
        #Sostituiamo tutte le occorrenze che rispettano la regex con uno
             spazio e rendiamo tutte le parola lower case
        cleaned = self.dataset.withColumn("clean_text", regexp_replace(
            col("text"), regex, " "))
        cleaned_text_df = cleaned.withColumn("clean_text_lower", lower(
            col("clean_text")))

        #crea, per ogni tupla, una lista di parole, il cui separatore
             lo spazio
        tokenizer = RegexTokenizer(inputCol="clean_text", outputCol="
            words", pattern="\\W+")
        dataset_tokenized = tokenizer.transform(cleaned_text_df)

        #rimuove le stop words dalla colonna tokenizzata ( articoli,
            congiunzioni ecc..)
        remover = StopWordsRemover(inputCol="words", outputCol="
            words_filtered")
        result = remover.transform(dataset_tokenized)
        return result
```

Per ogni tweet sono stati eseguiti i sequenti passaggi:

- ogni sottostringa di non interesse è stata sostituita con uno spazio e l'intero corpo del tweet è stato reso lower case;
- dalla colonna *clean text*, dopo essere stata tokenizzata, sono state rimosse le stop words, ovvero quelle parole che potrebbero risultare inutili ai fini di un analisi testuale, come congiunzioni, articoli, preposizioni, ecc...

Si fa uso della query appena descritta per creare la seguente :

```

def max_word_for_category(self):

        dataset_processato = self.pre_processing_text()

        #dal dataset processato prendiamo la colonna AIDRLabel e la
            colonna words_filtered esplosa, rinominata come words
        text_with_label = dataset_processato\
                        .select(col("AIDRLabel"), explode(col("
                            words_filtered"))\
                        .alias("Parola")) #\

        #per ogni coppia label-parola, contiamo quante volte occorre la
            coppia e inseriamo il valore in una nuova colonna words_count
        count_words = text_with_label\
                        .groupBy(col("AIDRLabel"),col("Parola"))\
                        .agg(count("*")\
```

```
                    .alias("Occorrenze"))

    #per ogni label prendiamo il massimo della colonna words count
    max_word = count_words\
                .groupBy(col("AIDRLabel")\
                .alias("labels"))\
                .agg(F.max("Occorrenze")\
                .alias("max_counted"))

    #facciamo la join con la tabella precedente per prendere la
       parola associata alla coppia label-words_count
    #eliminiamo eventuali parole con conteggio uguale, nella stessa
       label, e ordiniamo in ordine decrescente
    result = max_word\
            .join(count_words,(max_word.labels == count_words.
               AIDRLabel) & (max_word.max_counted == count_words.
               Occorrenze) , "inner")\
            .drop_duplicates(["labels"])\
            .orderBy(desc("max_counted"))

    #rimuoviamo le colonne supeflue
    result = result.drop("labels").drop("max_counted")

    return result
```

Dopo aver processato il testo e ottenuto il nuovo dataset, andiamo a selezionare la colonna *AIDRLabel* e ad esplodere la colonna **words filtered**. Successivamente, vengono contate le occorrenze di ogni coppia (label-parola) e inserite in una nuova colonna denominata *Occorrenze*. Per ogni categoria, andiamo a calcolare e selezionare il massimo valore della colonna *Occorrenze*. Infine, viene eseguita una join con il dataset definito al passo precedente in modo da ottenere il risultato espresso come categoria - parola - conteggio.

	AIDRLabel	Parola	Occorrenze
0	not_related_or_irrelevant	harvey	182,895
1	relevant_information	hurricane	171,745
2	donation_and_volunteering	harvey	150,005
3	sympathy_and_support	harvey	56,114
4	personal	irma	40,373
5	infrastructure_and_utilities_damage	harvey	11,821
6	caution_and_advice	harvey	10,975
7	injured_or_dead_people	harvey	7,083
8	affected_individual	harvey	4,184
9	response_efforts	harvey	19
10	missing_and_found_people	houston	17
11	displaced_and_evacuations	hit	2

La figura mostra il risultato, riportato sotto forma di tabella, visualizzabile da frontend. Si noti come, nella maggior parte della categorie, la parola che occorre di più sia **Harvey**.

2.5 Query 5: Le parole più usate nei tweet e la loro occorrenza nelle categorie

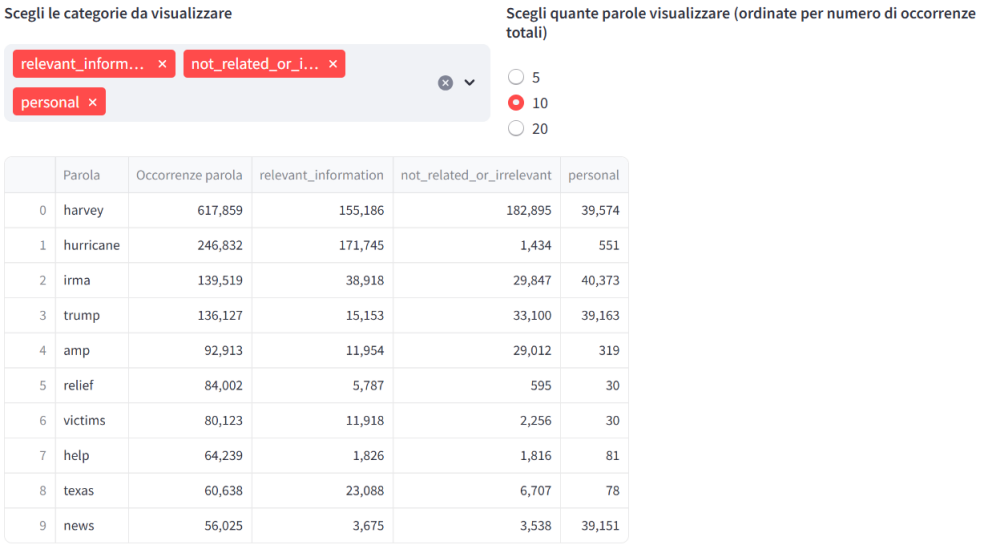
La quinta query, oltre a restituire le parole più utilizzate nei tweet, mostra come occorrono questi termini nelle categorie selezionate nel frontend. Il testo è stato pre-processato con il metodo descritto precedentemente. La query è riportata di seguito:

```
def process_text_data(self, labels, n=10):
        dataset_processato = self.pre_processing_text()

        prova = dataset_processato.select(col("AIDRLabel"), explode(col(
            "words_filtered")).alias("Parola"))\
                .groupBy("Parola")\
                .agg(count("Parola")\
                .alias("Occorrenze parola"),*[count(when(col("AIDRLabel"
                    ) == label,1)).alias(label) for label in labels])\
                .orderBy(desc("Occorrenze parola"))\
                .limit(n)

        return prova
```

Sul dataset processato, la query opera un raggruppamento ed un'aggregazione per ottenere le occorrenze di tutti i termini. Inoltre, per le categorie passate come parametro, viene creata una nuova colonna che riporta il numero di occorrenze di una parola per quella categoria. Del dataset risultante, si selezionano i primi n termini con più occorrenze, in base alla colonna *Occorrenza parola*. Il valore di n è selezionabile nel Frontend.



	Parola	Occorrenze parola	relevant_information	not_related_or_irrelevant	personal
0	harvey	617,859	155,186	182,895	39,574
1	hurricane	246,832	171,745	1,434	551
2	irma	139,519	38,918	29,847	40,373
3	trump	136,127	15,153	33,100	39,163
4	amp	92,913	11,954	29,012	319
5	relief	84,002	5,787	595	30
6	victims	80,123	11,918	2,256	30
7	help	64,239	1,826	1,816	81
8	texas	60,638	23,088	6,707	78
9	news	56,025	3,675	3,538	39,151

In figura è mostrato un esempio di applicazione della query, per un gruppo di categorie scelte e un valore di $n = 10$. Il risultato viene mostrato in forma tabellare. Si noti come, in accordo con la query precedente, il termine più usato sia **Harvey**.

2.6 Query 6: Gli utenti che hanno postato più tweet per categorie

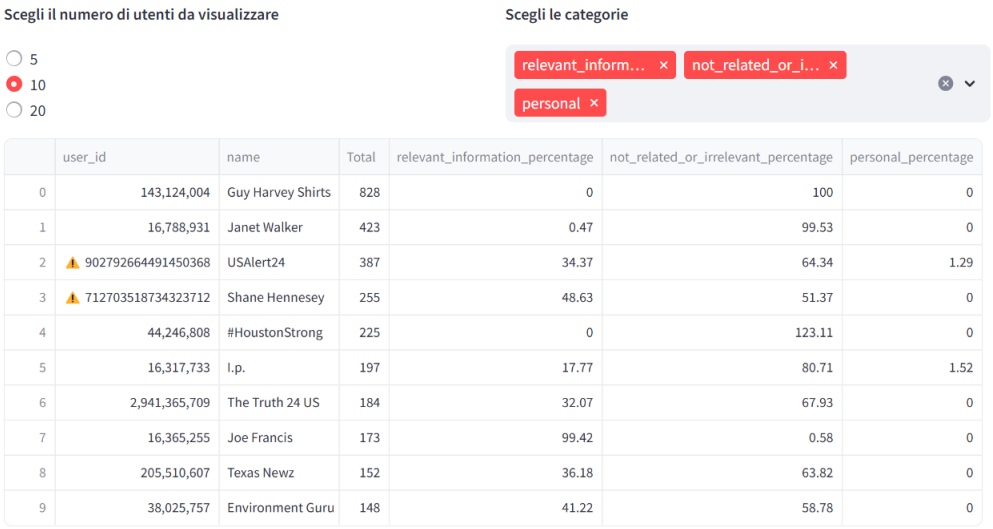
La query mostra gli n utenti che hanno postato di più nel dataset, mostrando per ogni categoria passata come parametro, la percentuale di tweet di ogni utente appartenenti a quella categoria. Anche in questo caso il valore n e le categorie sono selezionabili da Frontend.

```
                                                                    1
                                                                    2
    def top_n_user(self, categories, n=10):                        3
        if categories == []:                                        4
            return self.dataset.groupBy("user.id", "user.name").agg(5
                count("*").alias("Numero di tweet")).orderBy(desc("Numero
                 di tweet")).limit(n)
        else:                                                       6
            filtered_df = self.dataset.filter(col("AIDRLabel").isin(7
                categories))
                                                                    8
        tmp_user = filtered_df.select(col("user.id").alias("user_id"),9
            col("user.name").alias("name"), col("AIDRLabel"))
                                                                    10
        tmp2_user = tmp_user.groupBy("user_id", "name")\            11
            .agg(count("user_id").alias("Total"))\                  12
            .orderBy(desc("Total")).limit(n)                        13
                                                                    14
        tmp_categories = tmp_user.groupBy("user_id")\               15
            .agg(*[                                                 16
                count(when(col("AIDRLabel") == label, 1)).alias(label)17
                for label in categories                             18
            ])                                                      19
                                                                    20
        tmp_percentage = tmp_categories.join(tmp2_user, "user_id")  21
        result = tmp_percentage.select(                             22
            "user_id","name","Total",                               23
            *[round((col(label) / col("Total"))*100, 2).alias(f"{label}24
                _percentage") for label in categories]
        ).orderBy(desc("Total"))                                    25
                                                                    26
        return result                                               27
```

Se la variabile categories è vuota, si fa un'aggregazione sulla base dell'id e del nome dell'utente, e ne vengono contate le occorrenze. Se sono state selezionate le categorie di interesse, la query procede nel seguente modo:

- filtra il dataset in modo da considerare solo le tuple relative alle categorie selezionate;
- conta le occorrenze dei tweet per ogni utente e salva il risultato in un dataframe temporaneo;
- conta, per ogni utente, i tweet relativi ad ogni categoria, e salva il risultato in un altro dataframe temporaneo;

- effettua la join tra i due dataframe e calcola, sul totale delle categorie selezionate, la percentuale di tweet postati associati a ogni categoria.



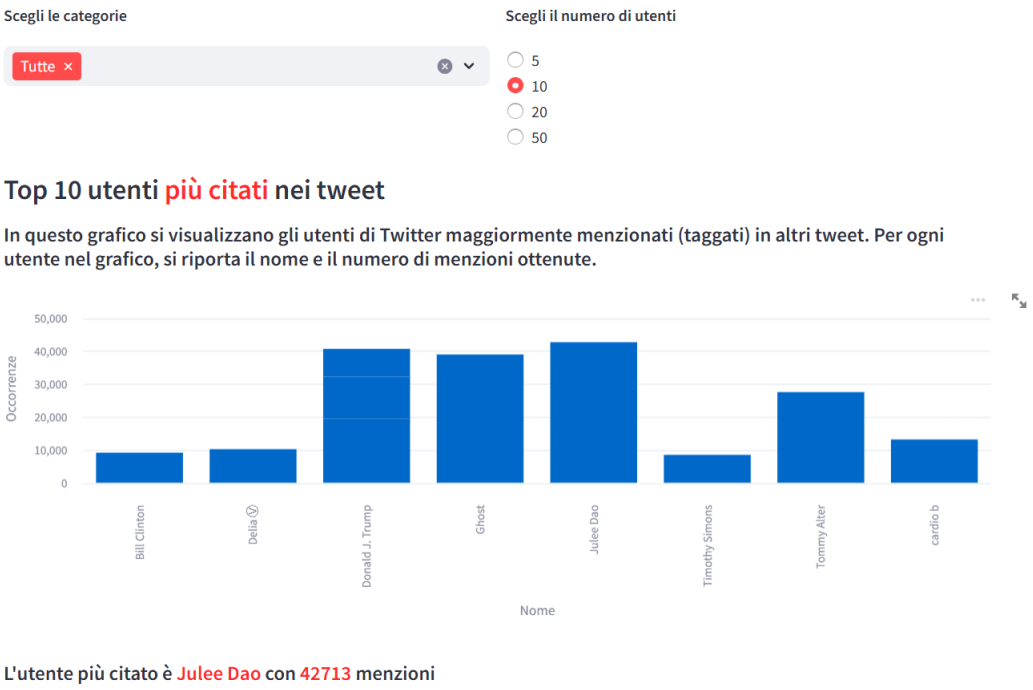
	user_id	name	Total	relevant_information_percentage	not_related_or_irrelevant_percentage	personal_percentage
0	143,124,004	Guy Harvey Shirts	828	0	100	0
1	16,788,931	Janet Walker	423	0.47	99.53	0
2	902792664491450368	USAlert24	387	34.37	64.34	1.29
3	712703518734323712	Shane Hennesey	255	48.63	51.37	0
4	44,246,808	#HoustonStrong	225	0	123.11	0
5	16,317,733	l.p.	197	17.77	80.71	1.52
6	2,941,365,709	The Truth 24 US	184	32.07	67.93	0
7	16,365,255	Joe Francis	173	99.42	0.58	0
8	205,510,607	Texas Newz	152	36.18	63.82	0
9	38,025,757	Environment Guru	148	41.22	58.78	0

La figura mostra un esempio di applicazione della query, scegliendo il parametro $n = 10$ e selezionando tre categorie. Il risultato anche in questo caso è riportato in un formato tabellare.

2.7 Query 7: Gli utenti più menzionati nei tweet

La seguente query ha il compito di mostrare le menzioni più ricorrenti nei tweet per ogni categoria passata come parametro. Per ottenere le menzioni di un tweet si usa la colonna *entities*. Si filtra il dataset, per ottenere i tweet relativi ai topic passati come parametro e si seleziona la colonna *AIDRLabel* e *user_mentions*. In particolare quest'ultima è il risultato dell'esplosione della colonna *entities_mentions*. Successivamente si raggruppa per *AIDRLabel* e *user_mentions.name* e si contano le occorrenze delle menzioni. Il risultato sarà il nome e il numero di citazioni degli n utenti più menzionati.

```
def top_mentioned_for_categories(self, categories, n=5):
        #per ogni tupla, prendere il campo entities.user_mention (che
            restituisce un array di account menzionati)
        #scandire l'array ed estrarne il nome, infine contare il numero
            di volte che appare ogni nome
        ds_esploso = self.dataset.select("AIDRLabel", explode(col("
            entities.user_mentions"))\
                                .alias("user_mentions"))\
                                .filter(col("AIDRLabel")\
                                .isin(*categories)) #ogni riga possiede
                                    la categoria e un utente mezionato
        ds_users = ds_esploso.select("AIDRLabel", "user_mentions.name")
        risultato = ds_users.groupBy("AIDRLabel", "name")\
                            .agg(count("*")\
                            .alias("Occorrenze"))\
                            .orderBy(desc("Occorrenze"))\
                            .limit(n)
        return risultato.withColumnRenamed("name", "Nome")
```



Nella figura è stato riportato un esempio di applicazione della query. Il parametro n è fissato a 10 e vengono scelte tutte le categorie disponibili. Il risultato viene mostrato tramite un grafico a barre dove, sulle ascisse, si riportano gli utenti più menzionati e, sulle ordinate, il numero di occorrenze degli utenti.

2.8 Query 8: Gli utenti più influenti che hanno postato almeno un tweet

La query ha lo scopo di restituire gli utenti più influenti che hanno postato almeno un tweet relativo all'uragano Harvey. Per realizzare questa query si è fatto uso della colonna *user*. Come nelle query precedenti, è possibile selezionare le categorie di interesse nel Frontend. Un utente viene considerato influente se:

- il suo profilo risulta verificato, controllando se il flag *verified* fosse posto a True;
- ha un alto numero di followers (sulla base di questo valore è stato realizzato un ordinamento degli utenti);
- il suo profilo risulta attivo, controllando il numero di tweet totali postati.

```
def most_influents_users(self, categories, n=10):
      filtered_df = self.dataset.filter(col("AIDRLabel").isin(*
         categories))
      dataset_user = filtered_df.select("user","AIDRLabel").filter(col
         ("user.verified") == True)
      tmp1= dataset_user.select("user.name","user.id","user.
         followers_count","user.statuses_count")\
                  .dropDuplicates(["id"])\

      tmp2 = dataset_user.select("user.id").groupBy("id")\
          .agg(count("id").alias("Tweet sull'uragano"))
      tmp2 = tmp2\
          .select(col("id")\
```

```
        .alias("userID"), col("Tweet sull'uragano"))
    result = tmp1.join(tmp2, tmp1["id"] == tmp2["userID"], "inner")\
        .drop("userID")\
        .dropDuplicates(["name"])\
        .orderBy(desc("followers_count"))\
        .limit(n)
    return result.withColumnRenamed("name", "Nome").
       withColumnRenamed("followers_count", "Followers").
       withColumnRenamed("statuses_count", "Tweet totali")
```

Top 10 utenti più influenti

	Nome	id	Followers	Tweet totali	Tweet sull'uragano
0	CNN Breaking News	428,333	52,346,965	57,685	1
1	The New York Times	807,095	39,263,062	290,555	13
2	Donald J. Trump	25,073,877	37,822,925	35,727	1
3	CNN	759,251	37,394,256	147,114	16
4	NFL	19,426,551	23,984,392	149,280	2
5	The Economist	5,988,062	21,860,630	117,350	1
6	BBC News (World)	742,143	21,090,310	263,340	1
7	Reuters Top News	1,652,541	18,630,647	205,099	16
8	Hillary Clinton	1,339,835,893	17,992,641	9,911	1
9	Ryan Seacrest	16,190,898	16,612,416	13,076	1

L'utente più influente che ha twittato almeno una volta sull'uragano Harvey è CNN Breaking News con 52346965 followers

In figura viene mostrato, con un formato tabellare, un esempio applicativo della query. In particolare, ogni tupla è composta da:

- nome dell'utente;
- id dell'utente;
- numero di followers;
- tweet totali pubblicati da quel profilo;
- tweet pubblicati dal profilo relativi all'uragano Harvey.

2.9 Query 9: Andamento giornaliero degli Hashtags

La query mostra la distribuzione degli hashtags nei vari giorni. Si ha la possibilità di selezionare una categoria di interesse. Presi gli hashtags dalla colonna entities, viene creata una colonna "Giorno" con un formato data "yyyy-MM-dd". Se la categoria è diversa da None, allora si filtra il dataset in base ad essa. In entrambi i casi, raggruppiamo le istanze in base al giorno e contiamo le occorrenze. Il risultato viene mostrato sul Frontend tramite un line chart, per evidenziare meglio l'andamento degli hashtags nei diversi giorni.

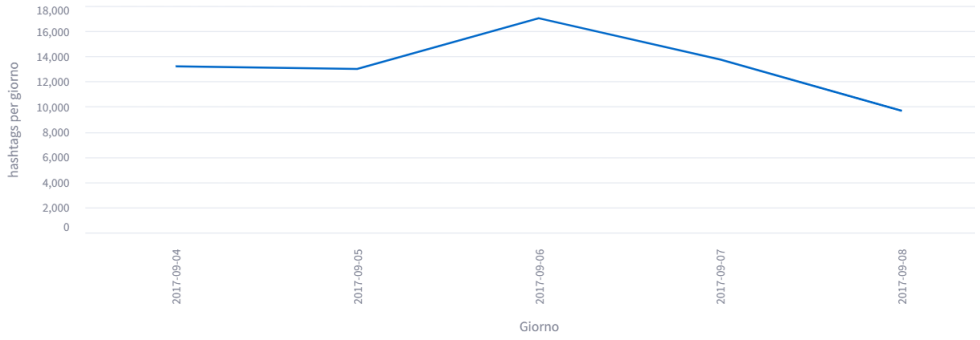
```
#QUERY 10
def hashtags_distribution_for_category(self, category):
    hashtag = self.dataset.selectExpr("AIDRLabel","created_at", "
       transform(entities.hashtags,x -> upper(x.text)) as
       vettore_hashtags")
    hashtag_exploded = hashtag.select("AIDRLabel","created_at",
       explode(col("vettore_hashtags")))
```

```
ds_data_convertita = hashtag_exploded.withColumn("timestamp",
    from_unixtime(unix_timestamp(col("created_at"), "EEE MMM dd
    HH:mm:ss ZZZ yyyy")))
ds_data_AMG = ds_data_convertita.withColumn("Giorno",
    date_format(col("timestamp"), "yyyy-MM-dd"))
if (category != None):
    risultato = ds_data_AMG.filter(col("AIDRLabel") == category)

    risultato = risultato.groupBy("Giorno")\
        .agg(count("*").alias("hashtags per giorno"))\

else:
    risultato = ds_data_AMG.groupBy("Giorno")\
        .agg(count("*").alias("hashtags per giorno"))
return risultato
```

- Si visualizza la distribuzione degli hashtags per ogni giorno (è possibile specificare la categoria)



3 Descrizione del modello di classificazione

All'interno dell'applicazione è stato integrato un modello predittivo di classificazione binaria. Lo scopo del modello è quello di identificare il contenuto di un tweet e assegnarlo ad una delle due categorie riportate:

- *relevant information*: è la classe dei tweet che contengono informazioni rilevanti riguardanti l'uragano Harvey;
- *not related or irrelevant*: è la classe dei tweet che non contengono informazioni rilevanti.

La classe che implementa il modello di classificazione è Classification.py, all'interno della quale vengono usati vari metodi della classe Apache Spark MLLib, sia nella fase di preprocessing dei dati, che nella fase di addestramento e test del modello. Il modello viene addestrato con i metodi Naive Bayes, Logistic Regression e Random Forest al fine di mettere a confronto le diverse tecniche applicate ad una classificazione testuale. Per valutare l'accuratezza di un modello, il dataset è stato suddiviso in training set e test set, consentendo al modello di essere addestrato sui dati di addestramento e valutarne le prestazioni sui dati di test.

3.1 Pre-processing dei dati

Prima dell'addestramento del modello è stata eseguita una fase di elaborazione dei dati, al fine di renderli più puliti e adatti ad essere passati in fase di training. In particolare i passaggi svolti sono i seguenti:

- **Lavoro sul dataset**: un primo passo è stato filtrare il dataset originario in modo da considerare soltanto le tuple relative alle due categorie scelte per la classificazione binaria. Inoltre, sono stati sostituiti i testi troncati dei tweet con i testi completi ed etichettati con una label (1 per *relevant information*, 0 per *not related or irrelevant*).

```
def select_column(self):
    dataset = self.dataset
    categories = self.categories
    new_dataset = dataset.withColumn("text",when(col("truncated
        ") == True,"extended_tweet.text").otherwise(col("text"))
        )
    df_filtered = new_dataset.select("text","AIDRLabel").filter
        (col("AIDRLabel").isin(categories))
    #avendo una colonna label non ho il bisogno di comunicare
        quale    la colonna target quando definisco il modello
    result = df_filtered.withColumn("label", when(col("
        AIDRLabel")== categories[0],1).otherwise(0))
    return result
```

- **Pulizia del testo**: Successivamente è stata effettuata la pulizia del testo. Si è fatto uso di un **RegexTokenizer** e di una **espressione regolare** per individuare i separatori di parole. In questo modo, il testo è stato suddiviso in una sequenza di parole. E' stato inoltre usato un oggetto **StopWordsRemover** per rimuovere le stop words, ovvero parole che non risultano significative alla comprensione del contesto e che possono essere eliminate. Successivamente è stato usato un oggetto **HashingTF**, che si occupa di estrarre le features dal testo usando la tecnica di term frequency : converte una sequenza di parole in un vettore di conteggio, in cui ogni parola viene mappata ad un indice nel vettore. Questo vettore consente di rappresentare il testo in forma numerica che può essere usata come input per il modello (il cosiddetto embedding). Infine è stato usato un oggetto **IDF** che viene calcolato

usando un vettore di conteggio delle frequenze dei termini, come per l'oggetto HashingTF. L'IDF assegna pesi più alti ai termini più rari e informativi, aiutando a distinguere parole chiave che potrebbero essere indicative della classe di appartenenza di un tweet.

```
def get_pipeline(self,model):

    dataset = self.dataset
    regex = r"\B@\w+\b|\B#\w+\b|https?://\S+|[\p{So}\p{Cntrl}\n
        \p{Punct}]+|RT|[^\x00-\x7F]+"
    cleaned = dataset.withColumn("text", regexp_replace(col("
        text"), regex, " "))
    cleaned_text_df = cleaned.withColumn("text", lower(col("
        text")))

    tokenizer = RegexTokenizer(inputCol="text", outputCol="
        words", pattern="\\W+")

    remover = StopWordsRemover(inputCol="words", outputCol="
        filtered_words")

    hashingTF = HashingTF(inputCol="filtered_words", outputCol=
        "raw_features", numFeatures=4000)

    idf = IDF(inputCol="raw_features", outputCol="features")
```

- **Creazione del modello e della pipeline**: Infine, è stato creato il modello con il metodo getModel() che restituisce la tecnica di classificazione che si vuole usare, a seconda di un parametro passato in input, ed è stata costruita la pipeline.

```
    model = self.get_model()

    pipeline = Pipeline(stages=[tokenizer, remover, hashingTF,
        idf, model])

    return pipeline
```

3.2 Addestramento e test del modello

Terminata la fase di pre-processing, il dataset risultante è stato diviso, in maniera casuale, in training set e test set. In particolare l' 80% delle tuple sono state inserite nel training set e le rimanenti nel test set. Successivamente è stato chiamato il metodo getModel(), che restituisce la tecnica da utilizzare per il modello di classificazione testuale:

- il valore 0 indica la tecnica Logistic Regression;
- il valore 1 indica la tecnica Naive Bayes;
- il valore 2 indica la tecnica Random Forest.

Infine, il modello è stato addestrato e successivamente applicato al test set. I valori ottenuti nel test set rappresentano i risultati del modello.

```
        model = self.get_model()
    def main(self):
        dataset = self.select_column()
        (trainingData, testData) = dataset.randomSplit([0.8, 0.2], seed
            =1234)
        pipeline = self.get_pipeline(self.model)
        model = self.train_model(pipeline, trainingData)
        accuracy,precision,recall,f1,auc, predictions = self.
            evaluate_model(model, testData)
        return accuracy,precision,recall,f1,auc, predictions
```

Nello specifico, sono state calcolate diverse metriche di valutazione, ovvero:

- **Accuracy**: in un classificatore binario, l'accuracy è una misura che indica quante volte il nostro modello ha classificato correttamente una tupla rispetto al totale del dataset:

$$\frac{(TP + TN)}{(TP + TN + FP + FN)}$$

- **Precision**: simile all'accuracy ma calcolata solo sulle classi positive:

$$\frac{TP}{TP + FP}$$

- **Recall**: utilizzata quando si vogliono riconoscere quanti più oggetti possibili come appartenenti alla classe positiva:

$$\frac{TP}{TP + FN}$$

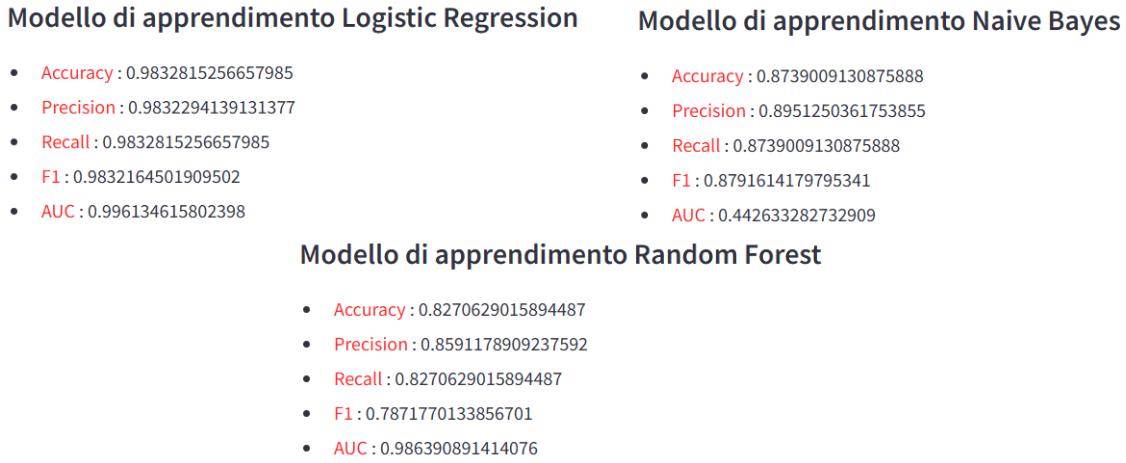
  E' una misura che non risulta particolarmente utile ai fini del nostro dataset, ma che è stata inserita per completezza. Infatti, la recall trova la sua utilità là dove si hanno tuple che, al minimo sospetto di appartenenza alla classe positiva, dovranno essere etichettate come positive (usata spesso in campo medico, per classificare tumori).

- **F1 score**: combina precision e recall in una sola metrica:

$$2 \cdot \frac{precision \cdot recall}{precision + recall}$$

- **AUC**: è l'area sottesa alla curva ROC.

Di seguito sono riportati i risultati per le tre tecniche:



3.3 Confronto tra modelli

Tra le tre tecniche usate, quella che performa meglio è la Logistic Regression, presentando valori elevati, per tutte le metriche usate. Il Naive Bayes e il Random Forest sembrano funzionare allo stesso modo, presentando valori intorno allo 0.8 per la maggior parte delle metriche. L'unica criticità viene riscontrata nel parametro di AUC della tecnica di Naive Bayes. Non è chiaro cosa porti ad avere un valore di AUC al di sotto dello 0.5, in tale tecnica, si può dire in generale che alcuni modelli si adattano meglio a determinati problemi, alcuni ad altri. Tuttavia è possibile ipotizzare quali possano essere le cause:

- **Assunzioni del Modello Naive Bayes**: Naive Bayes assume che le caratteristiche siano condizionatamente indipendenti data l'etichetta di classe. Se questa assunzione non tiene per i nostri dati, il modello potrebbe non essere adatto, influenzando in particolare la calibrazione delle probabilità che impatta sull'AUC.
- **Distribuzione delle Classi**: L'AUC tiene conto di come le probabilità predette separino la classe positiva dalla classe negativa su una scala continua. Se le probabilità predette dal modello Naive Bayes sono mal calibrate, potrebbe significare che il modello sta predicendo con alta confidenza anche quando non dovrebbe. Generalmente Naive Bayes tende a produrre probabilità estreme (molto vicine allo 0 o molto vicine ad 1). Questo può influire negativamente sull'AUC.
- **Sbilanciamento delle classi**: Se le classi sono sbilanciate, Naive Bayes potrebbe sovrastimare la probabilità della classe dominante, il che influenzerebbe negativamente l'AUC anche se altre metriche come la precisione e il recall rimangono alte.