

Dipartimento di Scienze Aziendali - Management & Innovation Systems
Corso di Laurea in Data Science & Gestione dell'Innovazione

Telegram e la disinformazione: studio delle connessioni informative

Esiste una relazione tra la chiusura delle community Telegram e la disinformazione? Il caso studio tramite Open Measures e Newsguard

INFORMATION DISORDER AND GLOBAL SECURITY

Telegram e la disinformazione: studio delle connessioni informative

Esiste una relazione tra la chiusura delle community Telegram e la disinformazione? Il caso studio tramite Open Measures e Newsguard

by

Denise Brancaccio - MAT. 0222800163
Lucia Brando - MAT. 0222800162
Bruno Maria Di Maio - MAT. 0222800149

Coordinatore: G. Fenza

Anno: 2025

Corso di Laurea: Data Science & Gestione dell'Innovazione



Prefazione

Il mondo delle community online ha assunto un ruolo sempre più centrale nella formazione delle opinioni, nella condivisione di informazioni e nella costruzione di identità collettive. Con la crescente diffusione di notizie false e la polarizzazione delle opinioni, è cruciale comprendere le dinamiche di queste community e il loro rapporto con la qualità delle fonti informative.

Questo lavoro si propone di esplorare il livello di apertura delle community su Telegram e il legame tra tale apertura e l'attendibilità dei siti di notizie condivisi al loro interno.

L'obiettivo è non solo mappare le connessioni tra le community e le fonti, ma anche fornire uno strumento di analisi visiva che evidenzi le dinamiche di interazione attraverso l'utilizzo di grafi generati con Gephi.

Questo studio si inserisce in un contesto di ricerca multidisciplinare che combina data science, analisi delle reti sociali e valutazione delle fonti informative, offrendo nuove prospettive sulla comprensione dell'ecosistema informativo digitale.

Indice

1 Introduzione	5
1.1 Obiettivo principale	5
2 Metodi e strumenti di lavoro	6
2.1 Glossario	6
2.2 Telegram	8
2.3 NewsGuard	8
2.4 Gephi	9
2.5 Open Measures	9
3 Sviluppo del lavoro	11
3.1 Raccolta e pre-analisi dei dati	11
3.1.1 File Newsguard.csv	11
3.1.2 File open_measures.py	12
3.1.3 File elaborazione.py	13
3.1.4 File gephi.py	16
3.1.5 File node.csv:	17
3.1.6 File archs.csv:	18
3.1.7 File connections.py	18
3.1.8 File gephi_canali.py	20
3.1.9 File community.py	21
3.1.10 File analisi_canali_per_domini.py	26
3.1.11 File istogrammi.py	26
3.1.12 Distribuzione degli Score	28
4 Grafi	31
4.1 Creazione dei grafi	31
4.1.1 Tentativi precedenti	31
4.2 Calcolo delle Metriche	32
4.3 Grafo finale	36
4.3.1 Grafo filtrato	39
4.3.2 Grafo dei Canali	40
5 Clustering	42
5.1 Clustering grafo finale	43
5.2 Clustering grafo filtrato	43
6 Tasso di chiusura	45
6.1 Metodo basato sui grafi	45
6.2 Metodo matematico	46
6.3 Analisi dei risultati	48
7 Conclusione e valutazione finale del team	49

1 Introduzione

Negli ultimi anni, la diffusione di informazioni sui social media e sulle piattaforme di messaggistica istantanea ha radicalmente trasformato il modo in cui le persone accedono alle notizie.

Telegram, in particolare, è emerso come uno degli strumenti più utilizzati per la condivisione di contenuti informativi, grazie alla sua flessibilità e alla capacità di ospitare gruppi e canali con migliaia di membri. Tuttavia, questa libertà d'uso si accompagna a rischi significativi, tra cui la proliferazione di notizie false e l'amplificazione di narrazioni manipolatorie. Le **community online** rappresentano un microcosmo di dinamiche sociali e informative, dove l'apertura – intesa come la capacità di interagire con fonti e utenti esterni – gioca un ruolo cruciale nel determinare la qualità e l'affidabilità delle informazioni condivise.

Questo studio si propone di investigare il rapporto tra il livello di apertura delle community e l'attendibilità delle fonti giornalistiche che vi circolano, offrendo una panoramica approfondita delle interazioni tra utenti, contenuti e fonti informative.

1.1 Obiettivo principale

L'obiettivo principale di questo lavoro è analizzare il livello di apertura delle community presenti su Telegram e correlare tale apertura all'affidabilità dei siti di notizie condivisi.

Attraverso l'utilizzo di dati raccolti direttamente da **Telegram**, incrociati con valutazioni fornite da **NewsGuard**, si mira a costruire un grafo rappresentativo delle connessioni e delle interazioni tra community e fonti informative.

Il grafo, realizzato tramite **Gephi**, consente una visualizzazione chiara e intuitiva delle dinamiche emerse, facilitando l'identificazione di pattern significativi e relazioni critiche.

2 Metodi e strumenti di lavoro

Per condurre l'analisi, è stato sviluppato uno **script Python** progettato per raccogliere dati da Telegram.

Questo script ha permesso di filtrare i messaggi contenenti link a siti di notizie.

I dati raccolti sono stati elaborati confrontandoli con i punteggi di **NewsGuard**, generando così un file .csv che includeva nodi (le community) e archi (le connessioni tra essi).

Il file è stato poi importato in **Gephi** per creare un grafo che rappresentasse visivamente le dinamiche informative.

2.1 Glossario

Di seguito si riportano alcuni dei 107 termini di ricerca utilizzati nel contesto della comunicazione e della diffusione delle informazioni.

Sono stati utilizzati termini che identificano contenuti di attualità e aggiornamenti di rilevante importanza:

- News
- Breaking
- Alert
- Headline
- Update
- Broadcast

Termini che si riferiscono a diverse tipologie di documenti informativi e giornalistici:

- Report
- Article
- Scoop
- Flash
- Bulletin
- Interview

Termini che indicano fenomeni informativi trasmessi in tempo reale o caratterizzati da una rapida diffusione:

- Live
- Trending
- Viral
- Insight
- Timeline
- Profile

- Community

Termini che descrivono situazioni di urgenza o emergenza:

- Emergency
- Incident
- Accident
- Disaster
- Crisis

Sono stati utilizzati, inoltre, temi di rilevanza sociale e politica:

- Conflict
- War
- Pandemic
- Health
- Technology
- Economy
- Politics
- Elections
- Debate
- Protest

Ma anche termini che delineano strumenti, processi e dinamiche legati alla raccolta, elaborazione e diffusione dell'informazione:

- Survey
- Poll
- Results
- Statistics
- Scandal
- Investigation
- Inquiry
- Revelation
- Discovery
- Press
- Conference
- On Air

2.2 Telegram



Telegram è una piattaforma di messaggistica molto utilizzata per la creazione di gruppi e canali tematici in cui gli utenti possono condividere e discutere contenuti.

A differenza di altre piattaforme, Telegram permette la creazione di canali pubblici e gruppi con un numero potenzialmente illimitato di membri, favorendo una rapida diffusione delle informazioni.

L'architettura di Telegram, infatti, consente agli utenti di condividere link a siti web, creando un ecosistema in cui notizie, articoli e contenuti virali possono diffondersi rapidamente. Questo lo rende un campo di studio ideale per analizzare la diffusione delle notizie, soprattutto in relazione alla qualità delle informazioni condivise.

Inoltre, la disponibilità di API permette di raccogliere dati in modo automatizzato e strutturato, semplificando l'analisi dei messaggi e dei contenuti condivisi all'interno dei gruppi e canali.

Per il nostro studio, Telegram è stato scelto come piattaforma principale per monitorare e raccogliere i dati, in quanto fornisce un ampio spettro di community diverse e informazioni su un vasto numero di argomenti, spesso trattati in modo non filtrato e senza verifica.

2.3 NewsGuard



NewsGuard è uno strumento che fornisce valutazioni affidabili sui siti web di notizie, assegnando loro un punteggio in base a criteri come la trasparenza, la correttezza e l'imparzialità dei contenuti.

Ogni sito analizzato viene esaminato da un team di giornalisti professionisti, che valuta i

parametri di qualità delle notizie pubblicate e fornisce una valutazione che aiuta gli utenti a discernere tra fonti affidabili e quelle che potrebbero diffondere disinformazione.

L'integrazione di NewsGuard nel nostro studio è stata essenziale, in quanto ha permesso di incrociare i link raccolti da Telegram con una valutazione professionale e oggettiva della loro affidabilità.

Grazie a questo strumento, è stato possibile determinare se le fonti di notizie condivise all'interno delle community fossero credibili o se provenissero da siti tendenti alla disinformazione. L'utilizzo di NewsGuard ha aggiunto un ulteriore livello di analisi alla nostra ricerca, consentendo di classificare i dati in modo oggettivo e basato su criteri di qualità giornalistica.

2.4 Gephi



Gephi è un software open-source utilizzato per l'analisi e la visualizzazione di grafi e reti complesse.

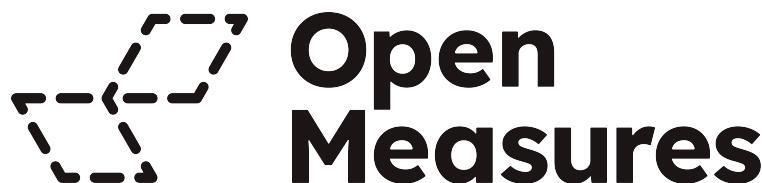
È particolarmente utile per studiare le relazioni tra entità attraverso la rappresentazione visiva di nodi e archi, dove ogni nodo rappresenta un'entità (in questo caso, una community), mentre gli archi rappresentano le connessioni tra di esse.

In questo studio, Gephi è stato utilizzato per visualizzare le connessioni tra i diversi nodi, permettendo una rappresentazione grafica che mostra la struttura e l'interconnessione di queste reti.

L'importazione dei dati raccolti e strutturati in formato CSV ha consentito di generare un grafo interattivo che rende facilmente visibili le relazioni tra le community e le fonti di notizie, offrendo al contempo una panoramica immediata delle dinamiche di diffusione delle informazioni.

Grazie alla sua capacità di gestire grandi quantità di dati e generare visualizzazioni dinamiche e interattive, Gephi ha svolto un ruolo cruciale nell'interpretazione e nella presentazione dei risultati, consentendo di osservare pattern, cluster e altre caratteristiche significative nelle connessioni tra le community e i siti di notizie.

2.5 Open Measures



Open Measures è una piattaforma avanzata progettata per supportare l'analisi e il monitoraggio delle interazioni sui social media, con un focus particolare sui contenuti generati dagli utenti.

La piattaforma offre API flessibili e potenti, che consentono di raccogliere dati strutturati su vasta scala.

Questi dati includono messaggi, metadati (ad esempio, autore, data e ora), contenuti multi-mediali, e informazioni sulle relazioni tra utenti o entità.

Le API di Open Measures sono progettate per essere altamente configurabili, permettendo di personalizzare le query attraverso parametri come parole chiave, intervalli temporali, e piattaforme social specifiche. Ciò consente agli utenti di accedere a dataset precisi e pertinenti per le loro analisi.

Ad esempio, è possibile eseguire ricerche avanzate per identificare discussioni che menzionano URL, frasi chiave o hashtag, oppure per analizzare contenuti pubblicati su piattaforme come Telegram, Twitter o Reddit.

Grazie alla sua architettura scalabile, Open Measures è particolarmente utile per progetti accademici, aziendali o governativi che richiedono l'analisi di grandi quantità di dati social.

Le sue applicazioni spaziano dal monitoraggio della disinformazione e dell'hate speech, all'analisi dei trend, fino agli studi sull'engagement e sulla polarizzazione delle opinioni.

Inoltre, la possibilità di esportare i dati in formati standard, come CSV, facilita l'integrazione con strumenti di analisi esterni come Python, R o software di visualizzazione come Gephi.

3 Sviluppo del lavoro

Il seguente capitolo descriverà in dettaglio la metodologia adottata, illustrando le tecniche di raccolta, filtraggio e analisi dei dati.

Successivamente verranno presentati i risultati ottenuti, con particolare attenzione alla mappatura delle community e alla visualizzazione dei grafi.

Infine, verranno discussi i risultati evidenziando le implicazioni pratiche e teoriche, oltre a considerare i limiti dello studio e le potenziali direzioni future.

3.1 Raccolta e pre-analisi dei dati

La raccolta e la preparazione dei dati costituiscono la base fondamentale di questo studio, che mira ad analizzare il comportamento delle community su Telegram.

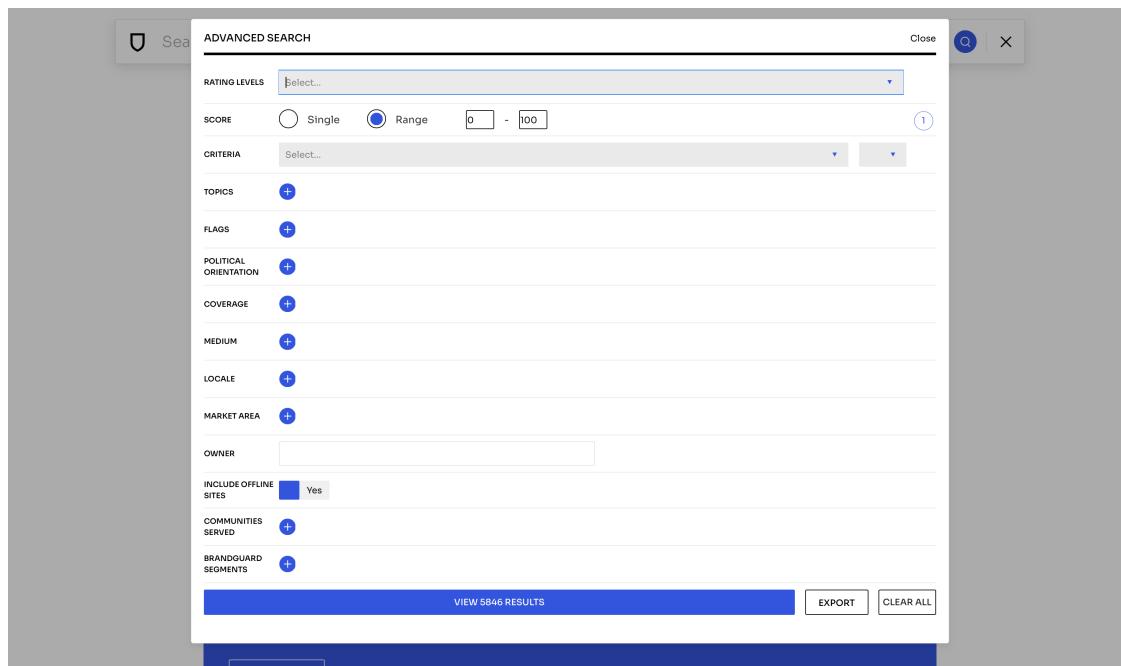
In questa fase, è stata utilizzata una combinazione di strumenti di programmazione e dataset per strutturare le informazioni raccolte, filtrare i dati rilevanti e prepararli per l'analisi dei grafi.

3.1.1 File Newsguard.csv

Per l'avvio del lavoro è stato generato il file "**Newsguard.csv**", contenente diversi dati relativi a fonti di notizie.

L'unico criterio applicato per filtrare i dati in fase iniziale è stato rappresentato dagli score assegnati alle fonti, variabili su una scala da 0 a 100.

Questo approccio ha permesso di selezionare le fonti in base al loro punteggio cercando di evitare bias dovuti all'orientamento politico o i topic trattati dal sito.



3.1.2 File open_measures.py

Per questo progetto è risultato fondamentale l'utilizzo dell'API di Open Measures. Grazie a quest'ultima, infatti, è stato possibile accedere a grandi quantità di dati strutturati relativi a Telegram, includendo dati e metadati come, ad esempio, contenuti dei messaggi, reactions e risposte.

Per configurare il sistema, è necessario:

- Ottenerne un token di autorizzazione (salvato nel file **open-measures-key.txt**).
- Definire i termini di ricerca attraverso file JSON (**parametri/terms.json**). Ogni termine viene verificato **in automatico**: se è già stato processato, non viene ricercato nuovamente.

```
1 def fetch_results(term, social, start_date, end_date,
2                   attempt_count=None):
3
4     params = {
5         'term': '(message:http OR message:https) AND message:%s',
6         '% term',
7         'limit': 10000,
8         'site': social,
9         'since': start_date,
10        'until': end_date,
11        'esquery': 'true'
12    }
```

La funzione genera query dinamiche per l'API, suddivide automaticamente gli intervalli temporali in caso di limiti di risultati, e converte le risposte in un DataFrame.

Quando il limite massimo di 10.000 risultati viene raggiunto, l'intervallo temporale della query viene automaticamente suddiviso a metà.

Questo processo si ripete ricorsivamente fino a quando ogni intervallo produce meno di 10.000 risultati.

```
1 ...
2 if len(hits) == 10000:
3     mid_date = pd.Timestamp(start_date) + (pd.Timestamp(end_date)
4                                     - pd.Timestamp(start_date)) / 2
5     second_half = mid_date + pd.Timedelta(days=1)
6     mid_date = mid_date.strftime("%Y-%m-%d")
7     second_half = second_half.strftime("%Y-%m-%d")
8     ...
9     first_half_df = fetch_results(term, social, start_date,
10                                    mid_date, attempt_count)
11     second_half_df = fetch_results(term, social, second_half,
12                                    end_date, attempt_count)
13 ...
14 else:
15     df = pd.DataFrame([hit['_source'] for hit in hits])
```

```
13     return df
```

Infine, i risultati sono stati salvati in file CSV per ogni termine analizzato, come mostrato nel seguente frammento di codice:

```
1     output_file = "data/%s_%s_2024.csv" % (social, term.replace(" ", "_"))
2     total_df.to_csv(output_file, index=False)
```

Questa metodologia ha garantito la scalabilità del processo di raccolta e la creazione di una base dati robusta per le successive fasi di analisi.

I file finali generati sono in formato CSV e includono:

- **Colonne strutturate:** dati e metadati come contenuto dei messaggi, autori, ecc.
- **Dati consistenti:** grazie alla conversione automatica dei tipi di dati specificati nei file **parametri/dtypes.json** e **parametri/required_columns.json**.

Un termine come "news", ricercato su Telegram nel 2024, produce dati relativi a messaggi contenenti URL (message:http OR message:https), che verranno salvati nel file CSV corrispondente.

Grazie all'automazione, ogni intervallo temporale è stato gestito in modo autonomo, senza intervento manuale.

Tramite questo processo è stato possibile raccogliere materiale corrispondente a **10.518.501 messaggi totali** in tutto l'anno 2024, utilizzando **107 termini di ricerca** totali.

3.1.3 File elaborazione.py

Si vanno ad analizzare i messaggi provenienti da Telegram per individuare e filtrare URL che puntano a domini di interesse, forniti dal file **Newsguard.csv**.

Il processo include la lettura, il filtraggio e l'aggregazione dei dati in due output principali: **output.csv**, che raccoglie i messaggi rilevanti, e **occorrenze.csv**, che riassume statistiche di dominio.

Gli URL vengono estratti dai messaggi tramite la funzione **extract_urls**, che normalizza i formati non standard (es. **[DOT]** → **.**) e identifica gli URL.

```
1 def extract_urls(text):
2     url_pattern = r"https?://(?:[-\w.]+(?:%[\da-fA-F]{2})|"
3     if isinstance(text, str):
4         cleaned_message = re.sub(r"\[(DOT|dot|\.)\]", " . ", str(
5             text))
6         cleaned_message = re.sub(r"\](?=\\s|$)", " ", 
7             cleaned_message)
8         return re.findall(url_pattern, cleaned_message)
9     return []
```

I domini vengono estratti dagli URL utilizzando la libreria **urllib.parse** per separare la parte rilevante.

```

1 def extract_domain(url):
2     try:
3         parsed_url = urlparse(url)
4         domain = parsed_url.netloc
5         if domain.startswith("www."):
6             domain = domain[4:]
7             return domain.lower()
8     except:
9         return ""

```

La funzione **process_csvs** analizza i file CSV nella directory **csv/** e li confronta con i domini di interesse presenti nel file **Newsguard.csv**.

```

1 def process_csvs(output_filename, occorrenze_filename,
2                   newsguard_filename="Newsguard.csv", folder_path="csv"):
3     newsguard_df = pd.read_csv(newsguard_filename).dropna(subset
4                               =[["Score"]]).drop_duplicates()
5
6     original_domains = {domain.lower(): domain for domain in
7                          newsguard_df["Domain"] if isinstance(domain, str)}
8     domain_scores = {domain.lower(): score for domain, score in
9                       zip(newsguard_df["Domain"], newsguard_df["Score"]) if
10                      isinstance(domain, str)}
11     domain_orientations = {domain.lower(): orientation for domain
12                            , orientation in zip(newsguard_df["Domain"], newsguard_df[
13                                "Orientation"]) if isinstance(domain, str)}
14
15     newsguard_domains = set(original_domains.keys())
16     columns_to_keep = ["message", "channelusername", "reactions",
17                         "forwards", "postauthor", "replies", "views"]
18
19     matching_rows = []
20     domain_stats = defaultdict(lambda: {"occorrenze": 0, "views": 0,
21                                 "forwards": 0})
22
23     for filename in os.listdir(folder_path):
24         if filename.endswith(".csv"):
25             print("Elaborazione file: %s" % filename)
26             df = pd.read_csv(os.path.join(folder_path, filename),
27                             low_memory=False).drop_duplicates()
28             for _, row in df.iterrows():
29                 if isinstance(row["message"], str):
30                     urls = extract_urls(row["message"])
31                     domains = {extract_domain(url) for url in
32                               urls}
33                     matching_domains = domains &
34                     newsguard_domains

```

```

23             if matching_domains:
24                 matching_rows.append(row[
25                     columns_to_keep])
26                 views = int(row["views"]) if pd.notna(
27                     row["views"]) else 0
28                 forwards = int(row["forwards"]) if pd.
29                     .notna(row["forwards"]) else 0
30                 for domain in matching_domains:
31                     domain_stats[domain]["occorrenze"]
32                         ] += 1
33                     domain_stats[domain]["views"] +=
34                         views
35                     domain_stats[domain]["forwards"]
36                         += forwards
37
38
39             if matching_rows:
40                 output_df = pd.DataFrame(matching_rows)
41                 output_df.to_csv(output_filename, index=False)
42                 print("\nAnalisi\completata.\u2022 File\%s\creato\con\%s\"
43                     corrispondenze." % (output_filename, len(output_df)))
44             else:
45                 print("\nNessuna\corrispondenza\trovata.\n")
46
47
48             occorrenze_data = []
49             for domain_lower, stats in domain_stats.items():
50                 if stats["occorrenze"] > 0:
51                     occorrenze_data.append({
52                         "Domain": original_domains[domain_lower],
53                         "Score": domain_scores[domain_lower],
54                         "Orientation": domain_orientations[domain_lower],
55                         "Occorrenze": stats["occorrenze"],
56                         "Views": stats["views"],
57                         "Forwards": stats["forwards"]
58                     })
59
60             if occorrenze_data:
61                 occorrenze_df = pd.DataFrame(occorrenze_data)
62                 occorrenze_df = occorrenze_df.sort_values("Occorrenze",
63                     ascending=False)
64                 occorrenze_df.to_csv(occorrenze_filename, index=False)
65                 print("File\%s\creato\con\%s\domini\attivi.\n" %
66                     (occorrenze_filename, len(occorrenze_df)))
67             else:
68                 print("Nessun\dominio\con\occorrenze\trovato.\n")

```

Le tipologie di output che si ottengono sono le seguenti:

O output.csv: Fornisce una visione dettagliata dei messaggi corrispondenti, utile per

analisi qualitative e per approfondire il contesto dei domini rilevati.

- **occorrenze.csv:** Riassume i dati quantitativi, ordinando i domini per numero di occorrenze in ordine decrescente, evidenziando i domini più rilevanti.

Domain	Score	Orientation	Occhorrenze	Views	Forwards
dominio1.com	60		120	1500	300
dominio2.com	90	Left	75	800	120
dominio3.com	40	Right	45	500	100

Tramite questo processo il file **output.csv** è stato popolato con **2.470.949 messaggi** utili alle successive analisi ed elaborazioni.

Il file **occorrenze.csv** è invece popolato da **3.768 domini** di Newsguard con un numero totale di **2.669.367 occorrenze** distribuite tra i diversi siti.

3.1.4 File geph.py

Il seguente file va a processare i dati presenti nel file output.csv per generare due file output:

- **nodes.csv:** contiene l'elenco dei nodi ed il loro tipo
- **archs.csv:** contiene le relazioni tra i nodi

Lo script ha lo scopo di:

- **Identificare i nodi:** creare un elenco di utenti e canali
- **Generare le relazioni:** costruire gli archi che rappresentano interazioni, dati dalle risposte degli utenti ai canali
- **Salvare i risultati**

La prima azione che andiamo a compiere è la lettura del file **output.csv**:

```
1 # Lettura del file CSV
2 data = pd.read_csv("output.csv")
3 df = pd.DataFrame(data)
```

Per ogni riga del file, lo script analizza la **colonna replies**, che contiene le informazioni sugli utenti che hanno risposto a un determinato canale.

Se la colonna **replies** è un dizionario valido, vengono estratti gli identificatori univoci degli utenti (**user_id**) e i canali a cui hanno risposto.

Questi dati vengono salvati in un dizionario.

```
1 dizionario = {}
2
3     for index, replies in enumerate(df["replies"]):
4         if isinstance(replies, str):
5             replies = ast.literal_eval(replies)
```

```

6         if isinstance(replies, dict) and replies["recent_repliers"]:
7             for user, _ in enumerate(replies["recent_repliers"]):
8                 if "user_id" in replies["recent_repliers"][user]:
9                     user_id = replies["recent_repliers"][user]["user_id"]
10                channel = df["channelusername"][index]
11                if user_id not in dizionario:
12                    dizionario[user_id] = ("User", [channel])
13                else:
14                    dizionario[user_id][1].append(channel)

```

I dati elaborati, come già detto in precedenza vengono salvati in due file csv.

3.1.5 File node.csv:

```

1 nodes = "nodes.csv"
2 target_set = set()
3
4 with open(nodes, mode="w", newline="", encoding="utf-8") as file:
5     writer = csv.writer(file)
6     writer.writerow(["id", "type"])
7
8     for key, (value, target) in dizionario.items():
9         writer.writerow([key, value])
10        for element in target:
11            if element not in target_set:
12                target_set.add(element)
13                writer.writerow([element, "Channel"]) # Nodo
14                  canale

```

ID	Type
12345678	User
channel_1	Channel

3.1.6 File archs.csv:

```
1     archs = "archs.csv"
2
3     with open(archs, mode="w", newline="", encoding="utf-8") as file:
4         writer = csv.writer(file)
5         writer.writerow(["source", "target", "type"])
6
7         for key, (_, target) in dizionario.items():
8             for element in target:
9                 writer.writerow([key, element, "Reply"])
```

Source	Target	Type
12345678	channel_1	Reply

3.1.7 File connections.py

Il file **connections.py** estende le relazioni presenti nel dataset **archs.csv**.

Il suo scopo è generare nuove connessioni tra i nodi basandosi su combinazioni esistenti e consolidare tali relazioni in un nuovo file csv denominato **archs_new.csv**.

Lo script esegue le seguenti operazioni:

- **Caricamento del dataset originale:** utilizza il file archs.csv come input per leggere le connessioni esistenti
- **Raggruppamento per origine:** per ogni nodo sorgente, individua i nodi destinazione collegati (source->target)
- **Generazione di combinazioni:** crea una sola coppia tra i nodi di destinazione collegati a una stessa sorgente, ripetuta nel caso di più canali in comune
- **Consolidamento delle connessioni:** concatena le connessioni generate con quelle originali e salva il risultato in **archs_new.csv**

La prima parte del codice legge il dataset per manipolare i dati tabulari:

```
1 df = pd.read_csv("archs.csv")
```

Il file **archs.csv** contiene le connessioni esistenti, strutturate con colonne **source**, **target** e **type**, dove:

- **Source:** rappresenta il nodo sorgente
- **Target:** rappresenta il nodo destinazione
- **Type:** specifica la natura della relazione

Successivamente, si va ad inizializzare le strutture per le nuove connessioni:

```
1 df_total_channels = pd.DataFrame(columns=["source", "target", "type"])
```

Successivamente, utilizza la funzione **combinations** della libreria **itertools** per generare le coppie possibili tra questi nodi di destinazione:

```

1  for _, group in df.groupby("source"):
2      targets = set(group["target"])
3
4      pairs = list(combinations(targets, 2))
5      df_new = pd.DataFrame(pairs, columns=["source", "target"]).
6          drop_duplicates()
7
8      df_new["type"] = "Connection"
9
df_total_channels = pd.concat([df_total_channels, df_new],
ignore_index=True)

```

In questo processo i nodi di destinazione vengono memorizzati nel set **targets** per garantire l'unicità e le combinazioni di coppie vengono generate per creare nuove connessioni tra i nodi. Le connessioni aggiuntive vengono unite a quelle originali presenti nel dataset:

```

1  df_final = pd.concat([df, df_total_channels], ignore_index=True).
2      dropna()
df_final.to_csv(output_file, index=False)

```

Il risultato è salvato in un nuovo file **archs_new.csv** che contiene sia le connessioni originali sia quelle derivate.

Source	Target	Type
user123	canale_1	Reply
user123	canale_2	Reply
user456	canale_3	Reply

Tabella 1: Esempio di file ‘archs.csv’.

Source	Target	Type
user123	canale_1	Reply
user123	canale_2	Reply
user456	canale_3	Reply
canale_1	canale_2	Connection

Tabella 2: Esempio di file ‘archs_new.csv’ con connessioni derivate.

Tramite questo processo è stato possibile ottenere **39.077 nodi**, salvati nel file **nodes.csv**, e **440.672 archi da pesare (51.499 archi pesati)**, salvati nel file **archs_new.csv**.

3.1.8 File gephi_canali.py

Basandosi su un filtro crea archi e nodi solo tra canali, andando ad escludere tutte le connessioni e i nodi utente.

Il filtro può essere attivato o disattivato per selezionare solo le community presenti nel file "community_chiusura.csv".

```
1 filter = True
2
3 if filter:
4     output_archs = "csv/archs_canali_filtrati.csv"
5     output_nodes = "csv/nodes_canali_filtrati.csv"
6 else:
7     output_archs = "csv/archs_canali.csv"
8     output_nodes = "csv/nodes_canali.csv"
9
10 df = pd.read_csv("csv/archs.csv").astype(str)
11
12 df_filter = pd.read_csv("csv/community_chiusura.csv").astype(str)
13 name_set = set(df_filter["Community"].unique())
14
15 if filter:
16     df = df[df["target"].isin(name_set)]
17
18 [...]
19
20 if filter:
21     df_nodes = df_nodes[df_nodes["id"].isin(name_set)]
```

Vengono mappati sia i source ai target che viceversa in modo da poter vedere successivamente, nella funzione calculate_weight, le connessioni in comune e calcolare il peso degli archi tra le community con la formula:

$$\frac{\text{utenti condivisi}}{\text{utenti totali dei canali}}$$

```
1 source_to_targets = df.groupby("source")["target"].apply(set).
2     to_dict()
3
4 target_alle_sue_source
5 target_to_sources = df.groupby("target")["source"].apply(set).
6     to_dict()
7
8 connections = []
9 for source, targets in source_to_targets.items():
10     if len(targets) > 1:
11         for target_pair in combinations(targets, 2):
12             connections.append(target_pair)
```

```

11 df_new = pd.DataFrame(connections, columns=["Source", "Target"])
12
13 df_new = df_new.apply(lambda x: tuple(sorted(x)), axis=1).
14     drop_duplicates().apply(pd.Series)
15 df_new.columns = ["Source", "Target"]
16
17 def calculate_weight(row):
18     t1_sources = target_to_sources[row["Source"]]
19     t2_sources = target_to_sources[row["Target"]]
20
21     shared_sources = len(t1_sources & t2_sources)
22
23     total_sources = len(t1_sources | t2_sources)
24
25     weight = shared_sources / total_sources if total_sources > 0
26         else 0
27     return weight
28
29 df_new["weight"] = df_new.apply(calculate_weight, axis=1)
df_new["type"] = "Connection"

```

3.1.9 File community.py

In questo paragrafo si va ad analizzare in dettaglio il codice sviluppato per l'elaborazione e l'analisi dei dati delle community, esaminando ogni componente e la sua funzione specifica nel processo complessivo.

Il codice definisce anche alcuni pattern regex ottimizzati per il parsing degli URL e la pulizia del testo:

```

1 URL_PATTERN = re.compile(r"https?://(?:[-\w.]+|(?::%[\da-fA-F]{2}))")
2 DOT_PATTERN = re.compile(r"\[(DOT|dot|\.)\] ")
3 BRACKET_PATTERN = re.compile(r"\](?=\\s|$)")

```

La funzione **preprocess_newsguard_data** si occupa di preparare i dati di Newsguard per l'analisi successiva:

```

1 def preprocess_newsguard_data(newsguard_df):
2     clean_df = newsguard_df.dropna(subset=["Score"]).
3         drop_duplicates()
4
5     return {
6         'original_domains': {d.lower(): d for d in clean_df["Domain"]}
7             if isinstance(d, str)},
8         'orientations': {d.lower(): o for d, o in zip(clean_df["Domain"], clean_df["Orientation"])}

```

```

8         if isinstance(d, str)},
9     'scores': {d.lower(): s for d, s in zip(clean_df["Domain"],
10        clean_df["Score"])}
11        if isinstance(d, str)},
12     'topics': {d.lower(): t for d, t in zip(clean_df["Domain"],
13        clean_df["Topics"])}
14        if isinstance(d, str)}
}

```

Questa funzione:

- Rimuove le righe con score mancanti e duplicate
- Crea quattro dizionari normalizzati per domini, orientamenti, score e topic
- Converte tutti i domini in minuscolo per garantire confronti case-insensitive
- Filtra eventuali valori non stringa per evitare errori

Le funzioni `extract_urls` e `extract_domain` lavorano in tandem per processare gli URL nei messaggi:

```

1 def extract_urls(text):
2     if not isinstance(text, str):
3         return []
4     cleaned = DOT_PATTERN.sub(".", text)
5     cleaned = BRACKET_PATTERN.sub("", cleaned)
6     return URL_PATTERN.findall(cleaned)
7
8 def extract_domain(url):
9     try:
10         parsed = urlparse(url)
11         domain = parsed.netloc
12         return domain[4:].lower() if domain.startswith("www.")
13             else domain.lower()
14     except:
15         return ""

```

Queste funzioni hanno gli scopi di:

- Normalizzare il testo sostituendo varianti di "dot" con punti
- Estrarre tutti gli URL validi dal testo
- Eseguire il parsing gli URL per estrarre domini
- Rimuovere il prefisso "www." e convertire in minuscolo le stringhe

Il vero cuore dell'analisi è la funzione `process_community_stats`:

```

1 def process_community_stats(df, newsguard_data):
2     community_stats = defaultdict(lambda: {"messaggi": 0, "views":
3         0, "forwards": 0, "domains": []})

```

```

3     newsguard_domains = set(newsguard_data['original_domains'].
4                               keys())
5
6     df['views'] = pd.to_numeric(df['views'], errors='coerce').
7         fillna(0).astype(np.int32)
8     df['forwards'] = pd.to_numeric(df['forwards'], errors='coerce'.
9         ) .fillna(0).astype(np.int32)
10
11    for community, group in df.groupby('channelusername'):
12        valid_messages = group[group['message'].notna()]
13
14        for _, row in valid_messages.iterrows():
15            urls = extract_urls(row['message'])
16            if not urls:
17                continue
18
19            domains = {extract_domain(url) for url in urls}
20            matching_domains = domains & newsguard_domains
21
22            if matching_domains:
23                stats = community_stats[community]
24                stats["messaggi"] += 1
25                stats["views"] += row['views']
26                stats["forwards"] += row['forwards']
27
28                for domain in matching_domains:
29                    if domain in community_stats[community][
30                        "domains"]:
31                        community_stats[community]["domains"][
32                            domain] += 1
33                    else:
34                        community_stats[community]["domains"][
35                            domain] = 1
36
37
38    return community_stats

```

Questa funzione:

- Inizializza un dizionario per memorizzare le statistiche delle community
- Converte e pulisce i dati numerici (views e forwards)
- Raggruppa i dati per community
- Analizza ogni messaggio per estrarre e contare i domini
- Aggiorna le statistiche per ogni dominio trovato che corrisponde ai dati di Newsguard

Le funzioni `calculate_topic_percentages` e `calculate_orientation_percentages` elaborano le distribuzioni dei topic e degli orientamenti politici:

```

1 def calculate_topic_percentages(domains, topics_data):
2     all_topics = []
3     for domain in domains:
4         topic_str = topics_data.get(domain, '')
5         if isinstance(topic_str, str):
6             topics = [t.strip() for t in topic_str.split(',') if
7                     t.strip()]
8             all_topics.extend(topics)
9
10    if not all_topics:
11        return {}
12
13    topic_counts = {}
14    total_topics = len(all_topics)
15    for topic in set(all_topics):
16        count = all_topics.count(topic)
17        topic_counts[topic] = (count / total_topics) * 100
18
19    return dict(sorted(topic_counts.items(), key=lambda x: x[1],
20                      reverse=True))

```

Le due funzioni vanno a raccogliere tutti i topic e orientamenti politici dai domini rilevanti, calcolano le percentuali di occorrenza ed ordinano i risultati in ordine decrescente.

La funzione `calculate_community_data` aggrega tutti i dati elaborati:

```

1 def calculate_community_data(community_stats, newsguard_data):
2     community_data = []
3
4     for community, stats in community_stats.items():
5         if not stats['domains']:
6             continue
7
8         domains = stats['domains']
9
10        topic_percentages = calculate_topic_percentages(domains,
11                newsguard_data['topics'])
12        orientation_percentages =
13            calculate_orientation_percentages(domains,
14                newsguard_data['orientations'])
15
16        valid_scores = [newsguard_data['scores'].get(domain)
17                        for domain in domains
18                        if domain in newsguard_data['scores']]
19        avg_score = sum(valid_scores) / len(valid_scores) if
20                    valid_scores else 0
21
22        community_data.append({
23            'community': community,
24            'topic_percentages': topic_percentages,
25            'orientation_percentages': orientation_percentages,
26            'avg_score': avg_score})
27
28    return community_data

```

```

19         "Community": community,
20         "Messaggi": stats["messaggi"],
21         "Views": stats["views"],
22         "Forwards": stats["forwards"],
23         "Score_medio": avg_score,
24         "Top_topic": max(topic_percentages.items(), key=
25             lambda x: x[1])[0] if topic_percentages else '',
26         "Top_domain": max(domains.items(), key=lambda x: x
27             [1])[0],
28         "Orientamento_percentuale": orientation_percentages,
29         "Topic_percentuale": topic_percentages,
30         "Domini": domains,
31         "Frequent_domains": dict(sorted(domains.items(),
32             key=lambda x: x[1],
33             reverse=True)[:5]))
34     }
35
36     return community_data

```

Questa funzione va ad elaborare i dati per ogni community, calcola statistiche aggregate come score medio e top topic ed, infine, organizza i dati in un formato strutturato per l'output finale.

Community	Messages	Views	Forwards	Score
realKarliBonne	14,334	260,614,542	1,667,483	73.77
Qrashthematrix	10,619	135,192,562	1,227,388	83.80
Jack_Posobiec	6,326	95,749,128	358,291	64.12
BannonWarRoom	17,581	70,309,397	314,505	74.94
CharlieKirk	1,872	55,537,030	347,258	82.62

Top Topic	Top Domain	Political Orient.
Political news	dailymail.co.uk	Right: 84.0%
Local News	thegatewaypundit.com	Right: 71.4%
Political news	theepochmillennial.com	Right: 90.7%
Political news	thenationalpulse.com	Right: 78.9%
Political news	theepochmillennial.com	Right: 89.3%

Tramite questa analisi è stato possibile verificare su **3.978 community**: il numero di messaggi inviati, le loro visualizzazioni, quante volte questi sono stati inoltrati, lo score medio delle notizie inviate, i topic più trattati dal canale, i domini più condivisi sul canale e l'orientamento politico in percentuale basato sui siti condivisi.

Sono emersi diversi dati che ci permettono di dire che le community più analizzate hanno un orientamento politico di destra, che il topic più discusso (secondo la classificazione di NewsGuard) è "Political news or commentary" e che lo score medio di tutti i siti utilizzati dalle community (dettato dalle metriche di NewsGuard) è 72,00192412015461.

3.1.10 File analisi_canali_per_domini.py

Lo script prende in input "Newsguard.csv", "community_chiusura.csv" e "clustering_canali_filtrati.csv" per generare il file "analisi_canali_per_domini.csv".

Inizialmente vengono mappati i domini agli score e le community ai cluster in modo da poterli recuperare facilmente in un secondo momento.

```
1 domain_score_map = dict(zip(newsguard_df["Domain"], newsguard_df[  
2   "Score"]))  
  
3 clustering_map = dict(zip(clustering_df["node_id"], clustering_df[  
4   "clustering_coefficient"]))
```

Vengono iterate le righe del dataframe contenente le community per poter associare a queste ultime i domini che hanno condiviso singolarmente, le occorrenze di ogni dominio, lo score di ogni dominio ed il clustering della community.

```
1 for _, row in community_df.iterrows():  
2     community = row["Community"]  
3     domain_dict_str = row["Domini"]  
  
4  
5     try:  
6         domain_dict = ast.literal_eval(domain_dict_str)  
7     except Exception as e:  
8         print("Errore nella conversione della colonna Domini per  
9             %s: %s" % (community, e))  
10        continue  
  
11    for domain, occurrences in domain_dict.items():  
12        score = domain_score_map.get(domain, None)  
13        clustering = clustering_map.get(community, None)  
  
14  
15        records.append({  
16            "Community": community,  
17            "Dominio": domain,  
18            "Occorrenze": occurrences,  
19            "Score": score,  
20            "Clustering": clustering  
21        })
```

Questo ci consente di elaborare i domini inviati da ogni community in maniera singola, in modo da poter creare grafici da utilizzare per l'analisi.

3.1.11 File istogrammi.py

Lo script si occupa di prendere l'analisi appena effettuata per generare degli istogrammi di ogni community che riportano gli score dei domini condivisi da queste ultime. Le colonne generate per questi grafici vanno da 0 a 100, con uno step di 5, quindi 0-5, 5-10, etc. (da

notare che, nonostante compaia due volte il valore 5, questo viene contato una sola volta perché il parametro right=True è presente nella definizione dei range).

Inoltre all'interno degli istogrammi vengono calcolati la media degli score, la varianza degli score, il totale delle occorrenze dei domini e viene riportato il coefficiente di clustering.

```
1 bins = list(range(0, 101, 5)) # Range da 0 a 100 con step di 5
2 labels = [f"{bins[i]}-{bins[i+1]}" for i in range(len(bins)-1)]
3
4 for community, group in df.groupby("Community"):
5     group["ScoreRange"] = pd.cut(group["Score"], bins=bins,
6                                   labels=labels, right=True)
7
8 score_counts = group.groupby("ScoreRange", observed=False)[
9     "Occorrenze"].sum().fillna(0)
10
11 mean_score = group["Score"].mean()
12 var_score = group["Score"].var()
13 total_occurrences = group["Occorrenze"].sum()
14
15 clustering_coeff = group["Clustering"].iloc[0] if "Clustering"
16     " in group.columns else "N/A"
17
18 clustering_coeff = "N/A" if pd.isna(clustering_coeff) else
19     clustering_coeff
20
21 fig, ax = plt.subplots(figsize=(14, 6))
22 ax.bar(score_counts.index, score_counts.values, color='skyblue',
23         edgecolor='black')
24
25 ax.set_xlabel("Score_Range")
26 ax.set_ylabel("Somma_delle_Occorrenze")
27 ax.set_title(f"Distribuzione_Score_per_{community}")
28
29 ax.set_xticks(range(len(score_counts)))
30 ax.set_xticklabels(score_counts.index, rotation=45)
31
32 ax.grid(axis='y', linestyle='--', alpha=0.7)
33
34 table_data = [
35     ["Media_Score", f"{mean_score:.2f}"],
36     ["Varianza_Score", f"{var_score:.2f}"],
37     ["Totale_Occorrenze", f"{total_occurrences}"],
38     ["Coefficiente_di_Clustering", f"{clustering_coeff}"]
39 ]
40
41 table = ax.table(cellText=table_data, colLabels=["Parametro",
42         "Valore"],
43                     cellLoc="center", loc="center_right",
44                     )
45
```

```

38         bbox=[1.05 , 0.2 , 0.4 , 0.5])

39
40     table.auto_set_font_size(False)
41     table.set_fontsize(10)
42     table.scale(1.2 , 1.4)

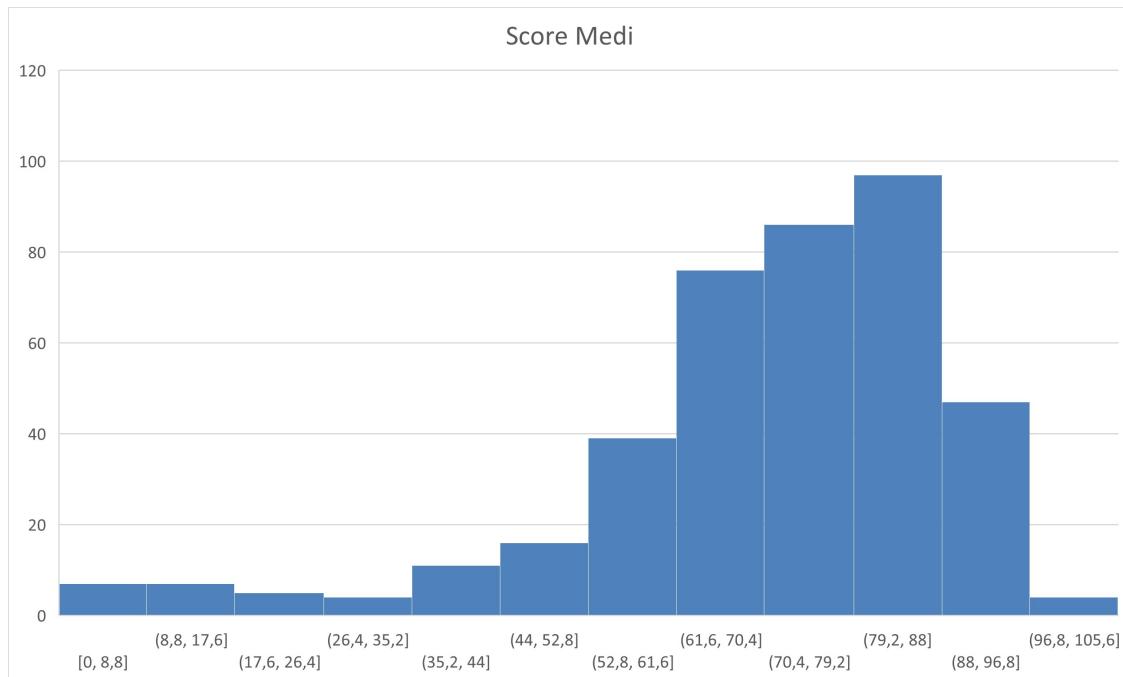
43
44     plt.savefig(f"istogrammi/{community}.png" , bbox_inches="tight"
45             )
        plt.close()

```

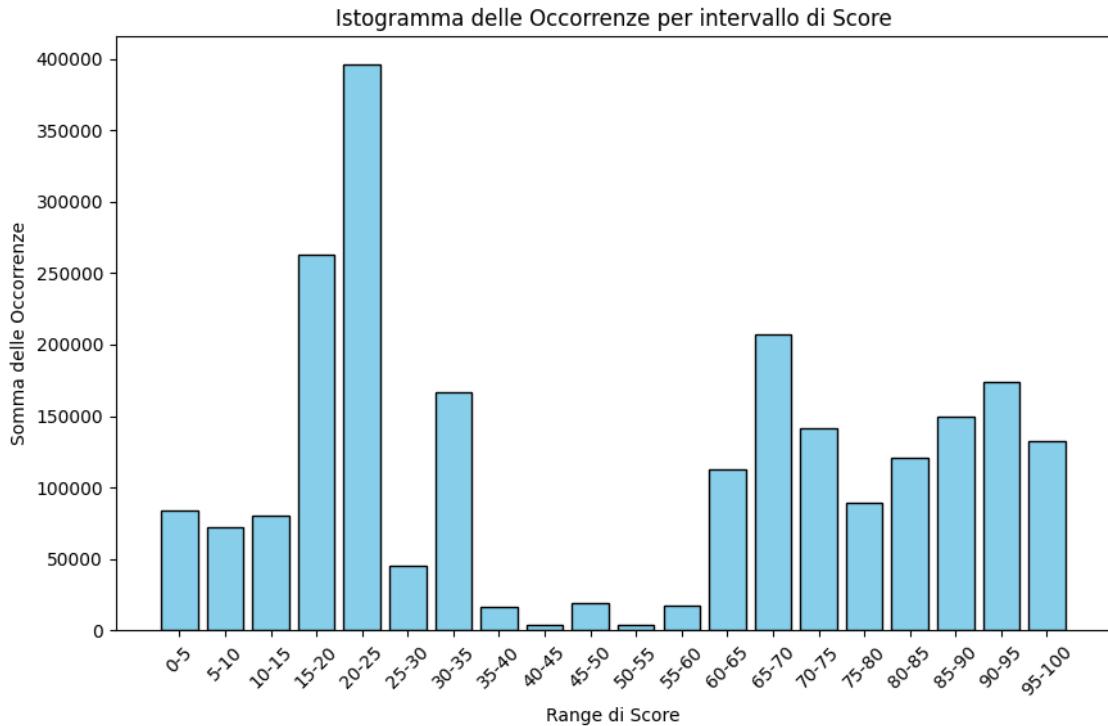
3.1.12 Distribuzione degli Score

A seguito inseriamo un istogramma riguardante la distribuzione generale degli score di Newsguard rispetto ai domini trovati nei canali che abbiamo filtrato.

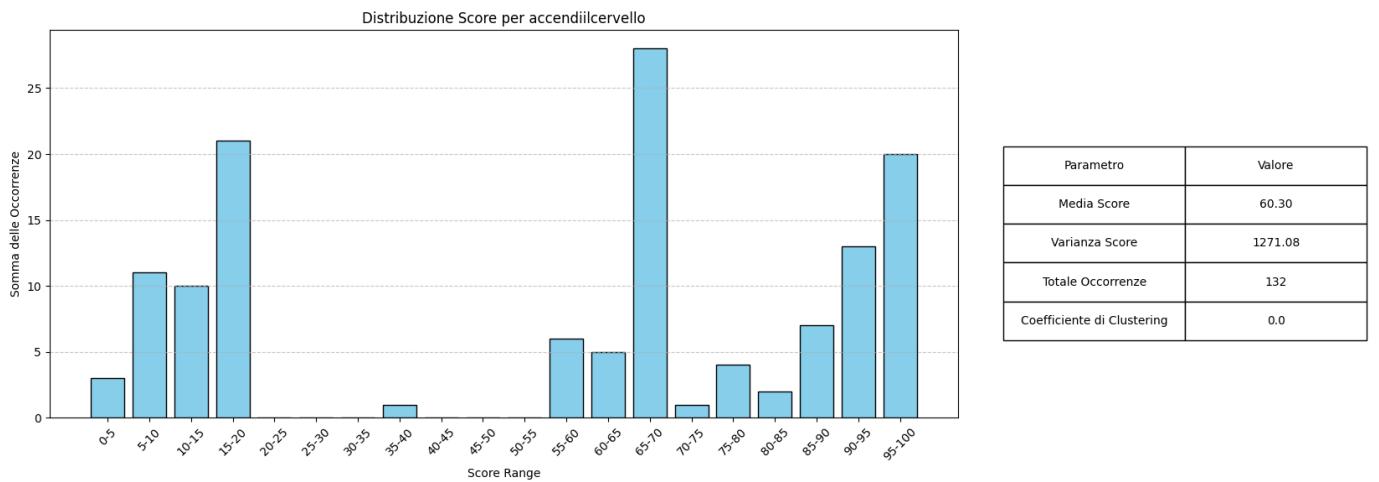
È possibile notare come la maggior parte delle notizie ricadano nell'intervallo di score tra 52,8 e 96,8, delineando una qualità delle notizie medio-alta.

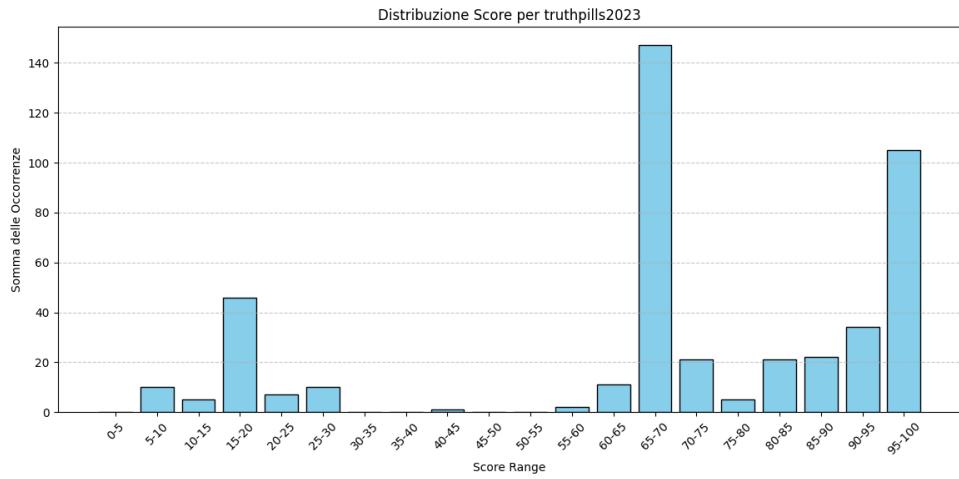


Mentre il precedente istogramma rappresentava gli score medi delle community filtrate, il seguente invece indica gli score di tutti i domini analizzati presenti nel file "occorrenze.csv".

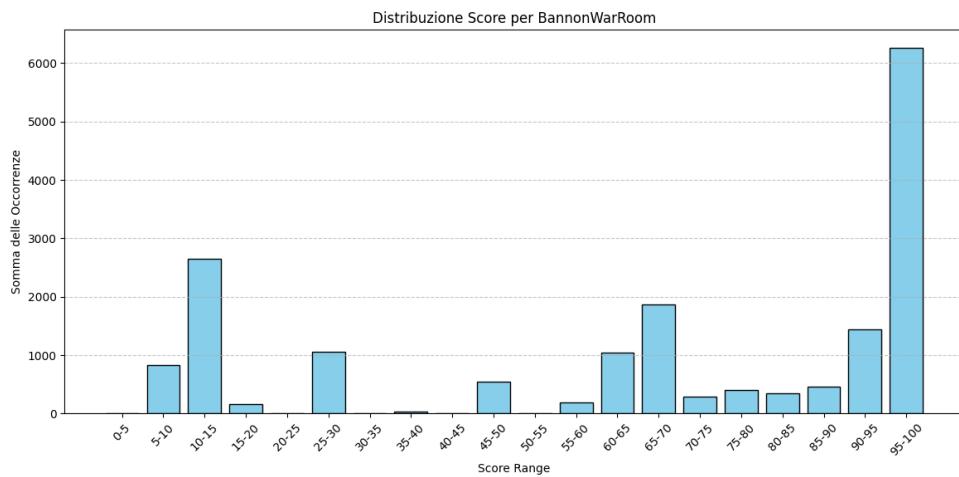


Per approfondire, abbiamo creato ulteriori grafici che mostrano gli score per i domini presenti in ogni singolo canale tenendo conto anche delle occorrenze, le quali erano state escluse nel grafico precedente. Riportiamo qui solo alcuni degli istogrammi creati tramite il file `istogrammi.py`.





Parametro	Valore
Media Score	73.13
Varianza Score	1076.39
Totale Occorrenze	449
Coefficiente di Clustering	0.3



Parametro	Valore
Media Score	74.94
Varianza Score	869.81
Totale Occorrenze	17594
Coefficiente di Clustering	0.1371124031007752

4 Grafi

4.1 Creazione dei grafi

Inizialmente, l'analisi era stata concepita per essere effettuata su grafi orientati, con l'obiettivo di sfruttare la direzionalità degli archi per rappresentare relazioni.

Tuttavia, durante lo sviluppo del lavoro, si sono presentate alcune limitazioni tecniche e metodologiche che hanno reso necessario spostare l'attenzione verso l'utilizzo di grafi non orientati, privilegiando una rappresentazione più semplice e simmetrica delle relazioni.

Per garantire una documentazione completa e trasparente, abbiamo comunque deciso di includere entrambe le versioni dell'analisi, offrendo un confronto tra i due approcci.

4.1.1 Tentativi precedenti

Questa rappresentazione è stata ottenuta utilizzando il software Gephi, attraverso il caricamento dei dataset **nodes.csv** e **archs.csv**, contenenti rispettivamente le informazioni relative ai nodi (utenti o canali) e agli archi (connessioni o interazioni tra i nodi).

I nodi, visualizzati con colori distinti, identificano cluster di utenti che condividono connessioni più dense, suggerendo l'esistenza di comunità tematiche o gruppi coesi.

Gli archi, che collegano i nodi, rappresentano le interazioni tra i diversi utenti o gruppi, evidenziando la struttura globale della rete.

La distribuzione spaziale del grafo evidenzia una centralità maggiore in alcune aree, dove i nodi risultano più grandi e prominenti.

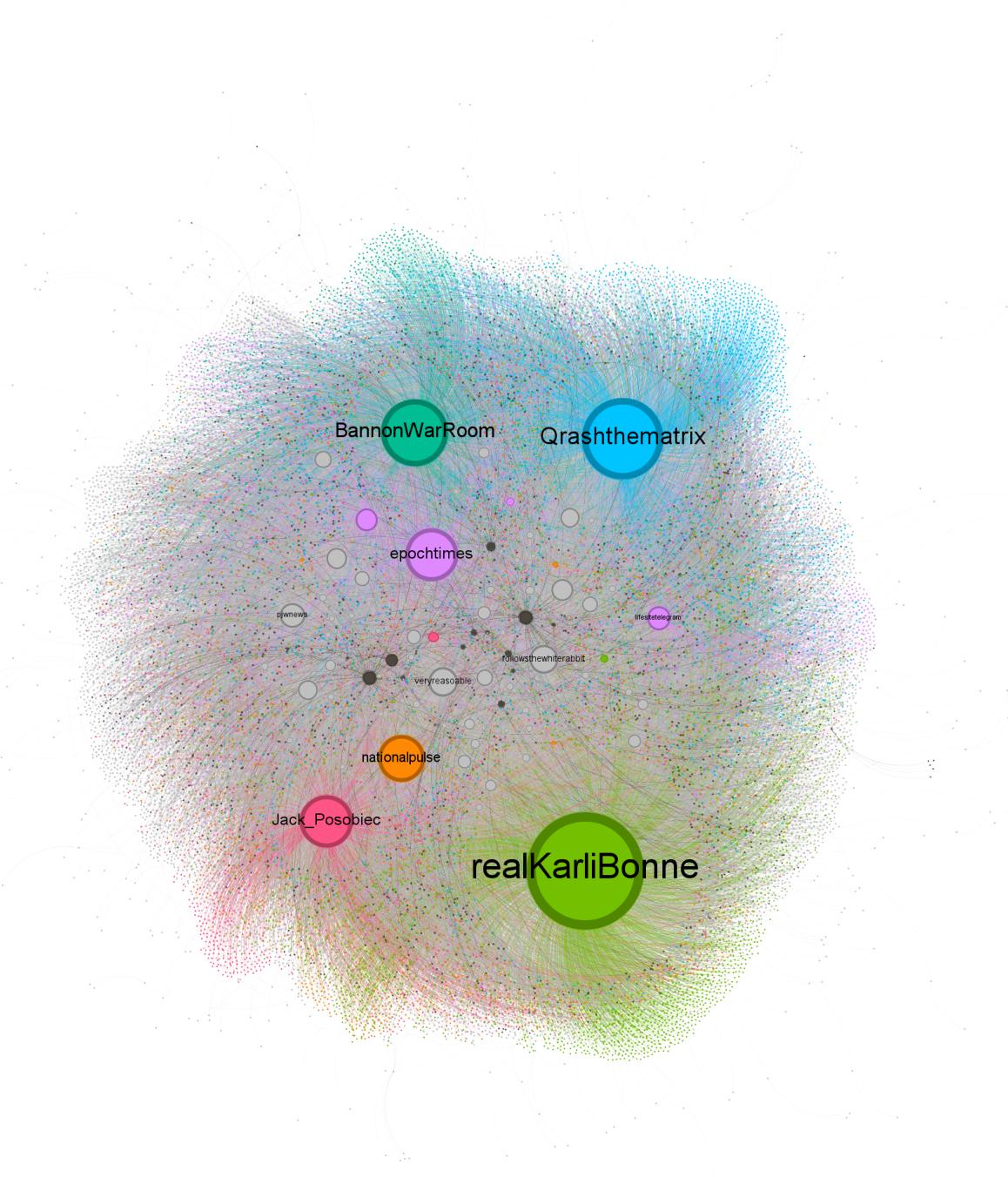
Questi possono essere interpretati come utenti o gruppi particolarmente influenti all'interno della rete, il cui ruolo potrebbe essere cruciale nella diffusione di informazioni o nel consolidamento di opinioni.

Al contrario, i nodi periferici, distanziati dal nucleo centrale, possono rappresentare utenti marginali o comunità meno integrate, contribuendo comunque alla diversità della rete complessiva.

La colorazione differenziale dei cluster tramite la Modularity permette una chiara identificazione delle comunità, facilitando l'analisi di fenomeni come la polarizzazione, l'influenza e la diffusione di informazioni all'interno del network.

Il seguente grafo è frutto di uno degli ultimi tentativi di rappresentazione del network, successivamente scartato. I nodi canali sono connessi ai nodi utente che fanno parte della loro community in base alle interazioni (reply) che hanno effettuato. I nodi utente hanno solo archi uscenti, mentre i nodi canali hanno solo archi entranti, motivo per la quale, a causa della rappresentazione troppo semplificata del network, il grafo è stato scartato.

Ciò che è osservabile in tutti i grafi prodotti è la classifica dei 10 canali con il grado maggiore, la quale è mediamente rimasta tale in tutte le versioni del grafo. I nodi etichettati rappresentano proprio questa classifica e sembrano essere tutti di origine statunitense.



4.2 Calcolo delle Metriche

Successivamente, il file è stato utilizzato per la creazione delle metriche, confrontate con quelle fornite da Gephi.

Il seguente codice implementa un'analisi strutturale sul grafo orientato, esportato in formato **.graphml**.

Esso utilizza librerie Python come **NetworkX** per calcoli di metriche topologiche avanzate e **Pandas** per l'organizzazione e l'esportazione dei risultati.

Il grafo viene analizzato attraverso misure di centralità e metriche di rete.

Di seguito è riportata una spiegazione dettagliata:

○ Caricamento del grafo:

Il grafo viene caricato utilizzando il metodo **nx.read_graphml**. Una volta caricato, vengono stampate informazioni generali sulla rete, come il numero di nodi e di archi:

```
1   print("Grafo caricato con successo. Numero di nodi:", graph.  
      number_of_nodes(), "Numero di archi:", graph.  
      number_of_edges())
```

Questo consente una verifica preliminare per assicurarsi che i dati siano stati importati correttamente.

○ Filtraggio dei nodi "channel":

Il codice filtra i nodi con un **attributo "type"** pari a "Channel", utilizzando una lista comprensione:

```
1   channels = [node for node in graph.nodes if graph.nodes[node]  
      .get("type") == "Channel"]
```

Questa operazione seleziona un sottoinsieme specifico del grafo, così da poter focalizzare l'analisi su canali Telegram piuttosto che su altri tipi di nodi.

Il numero totale di nodi selezionati viene stampato per trasparenza.

○ Calcolo delle metriche di centralità:

– Degree:

Il grado rappresenta il numero di connessioni di un nodo ed è calcolato senza considerare eventuali pesi sugli archi:

```
1   degree_dict = dict(graph.degree(channels))
```

Questa metrica è utile per identificare nodi altamente connessi all'interno della rete.

– Betweenness Centrality:

Questo indica l'importanza del nodo come intermediario:

```
1   betweenness_dict = nx.betweenness_centrality(graph,  
      normalized=True, endpoints=False)
```

L'opzione **normalized=True** assicura che i valori siano scalati rispetto alla dimensione del grafo.

– Closeness Centrality:

Questa metrica misura quanto un nodo sia vicino a tutti gli altri nodi della rete, in termini di distanza geodetica:

```
1   closeness_dict = nx.closeness_centrality(graph)
```

- **Authority Score:**

Se il grafo è orientato, viene calcolata l'authority di ciascun nodo utilizzando l'algoritmo HITS, che assegna punteggi in base alla qualità delle connessioni in entrata:

```
1     _, authority_dict = nx.hits(graph, normalized=True)
```

Nel caso di grafi non orientati, l'authority viene impostata a zero, poiché Gephi non supporta il calcolo di questa metrica per grafi non diretti.

- **Eigenvector Centrality:**

Questa metrica valuta l'influenza di un nodo in base all'importanza dei suoi vicini, calcolata iterativamente:

```
1     eigenvector_dict = nx.eigenvector_centrality(graph,
2                                                 max_iter=1000, tol=1e-6)
```

Parametri come **max_iter** e **tol** sono scelti per garantire la convergenza.

O Creazione dei risultati:

I risultati delle metriche vengono raccolti in una lista di dizionari, dove ciascun dizionario rappresenta un nodo e le relative metriche calcolate:

```
1     results = []
2     for node in channels:
3         results.append({
4             "Node": node,
5             "Degree": degree_dict.get(node, 0),
6             "Betweenness": betweenness_dict.get(node, 0),
7             "Closeness": closeness_dict.get(node, 0),
8             "Authority": authority_dict.get(node, 0),
9             "Eigenvector": eigenvector_dict.get(node, 0),
10            })
```

O Esportazione risultati:

I risultati vengono convertiti in un DataFrame di Pandas, una struttura dati tabellare, e salvati in **formato .csv**:

```
1     df = pd.DataFrame(results)
2     output_file = "apertura_gephi.csv"
3     df.to_csv(output_file, index=False)
```

A seguito del confronto tra le metriche fornite da Gephi e quelle ottenute tramite Python, risultate diverse tra loro, abbiamo deciso di utilizzare le metriche di Gephi in fase finale.

La fase successiva dell'analisi del grafi prevede che i dati derivati vengano elaborati ulteriormente per calcolare una nuova metrica chiamata "**tasso di apertura**", elaborata a scopo ipotetico e successivamente eliminata a favore del tasso di apertura.

Questo processo consente di arricchire ulteriormente le informazioni estratte, aggiungendo

una dimensione sintetica che combina diverse metriche di centralità per fornire una visione complessiva dell'importanza e della posizione strategica dei nodi della rete.

Lo script si basa su due file: il primo, **gephi_data.csv**, contiene i dati di base del grafo già processati e arricchiti con metriche di centralità; il secondo, **apertura_gephi.csv**, viene generato dallo script ed include la nuova colonna "Tasso di apertura" accanto alle metriche originali, ampliando così le possibilità di analisi successive.

Il file **gephi_data.csv** rappresenta una tabella con informazioni dettagliate per ogni nodo del grafo. Le colonne principali includono:

- **Node:** identificativo univoco del nodo nella rete.
- **Degree:** numero di connessioni dirette che un nodo ha nella rete.
- **Betweenness:** misura dell'importanza di un nodo come intermediario nei percorsi brevi tra altri nodi, evidenziando il suo ruolo nel facilitare il flusso di informazioni.
- **Closeness:** rappresenta la vicinanza di un nodo a tutti gli altri nodi della rete, calcolata come l'inverso della somma delle distanze, una metrica utile per identificare nodi centrali.
- **Authority:** valore che indica l'autorevolezza del nodo basato sull'algoritmo HITS.
- **Eigenvector:** centralità che misura l'influenza del nodo in base all'importanza dei nodi a cui è connesso.

Un esempio delle prime cinque righe di questo file è il seguente:

```
1 Node, Degree, Betweenness, Closeness, Authority, Eigenvector
2 1, 10, 0.02, 0.4, 0.3, 0.8
3 2, 5, 0.01, 0.35, 0.2, 0.7
4 3, 8, 0.03, 0.38, 0.25, 0.75
5 4, 6, 0.015, 0.37, 0.22, 0.72
6 5, 7, 0.02, 0.36, 0.24, 0.73
```

Lo script inizia caricando i dati da **gephi_data.csv** utilizzando la libreria **pandas**, un potente strumento per la manipolazione di dati strutturati.

Dopo aver caricato i dati, verifica che le colonne necessarie siano presenti, garantendo che il dataset sia completo e conforme ai requisiti dell'analisi:

```
1 file_path = 'gephi_data.csv'
2 df = pd.read_csv(file_path)
3 required_columns = ["Node", "Degree", "Betweenness", "Closeness",
4                     "Authority", "Eigenvector"]
5 if not all(col in df.columns for col in required_columns):
6     raise ValueError(f"Il file CSV deve contenere le seguenti colonne: {', '.join(required_columns)})")
```

Se una delle colonne mancasse, lo script interromperebbe l'esecuzione con un messaggio di errore chiaro, assicurando un controllo di qualità sui dati in ingresso.

Per rendere comparabili le diverse metriche, lo script utilizza il metodo **MinMaxScaler** di **sklearn** per scalare i valori nel range [0,1].

Questa normalizzazione è essenziale per eliminare eventuali disparità dovute a scale diverse nelle metriche originali e garantire che ogni metrica contribuisca equamente al calcolo successivo:

```
1 metrics = ["Degree", "Betweenness", "Closeness", "Authority", "Eigenvector"]
2 scaler = MinMaxScaler()
3 normalized_metrics = scaler.fit_transform(df[metrics])
```

Il risultato è una matrice in cui ogni valore rappresenta una versione scalata del corrispondente valore originale, uniformando l'importanza relativa di ciascuna metrica.

Il **"Tasso di apertura"** viene calcolato attraverso la media aritmetica delle metriche normalizzate per ogni nodo.

Questo approccio aggrega informazioni complesse in un unico valore, semplificando l'identificazione di nodi che mostrano una combinazione equilibrata di centralità e influenza:

```
1 df[["Tasso_di_apertura"]] = normalized_metrics.mean(axis=1)
```

Il risultato è una nuova colonna aggiunta al dataset, che fornisce un'indicazione sintetica ma potente del ruolo globale del nodo nella rete analizzata.

Il file aggiornato, contenente la nuova colonna "Tasso di apertura", viene salvato in un nuovo file CSV. Le prime cinque righe del file generato appaiono così:

```
1 Node, Degree, Betweenness, Closeness, Authority, Eigenvector,
2 Openness
3 1, 10, 0.02, 0.4, 0.3, 0.8, 0.508
4 2, 5, 0.01, 0.35, 0.2, 0.7, 0.432
5 3, 8, 0.03, 0.38, 0.25, 0.75, 0.488
6 4, 6, 0.015, 0.37, 0.22, 0.72, 0.465
7 5, 7, 0.02, 0.36, 0.24, 0.73, 0.482
```

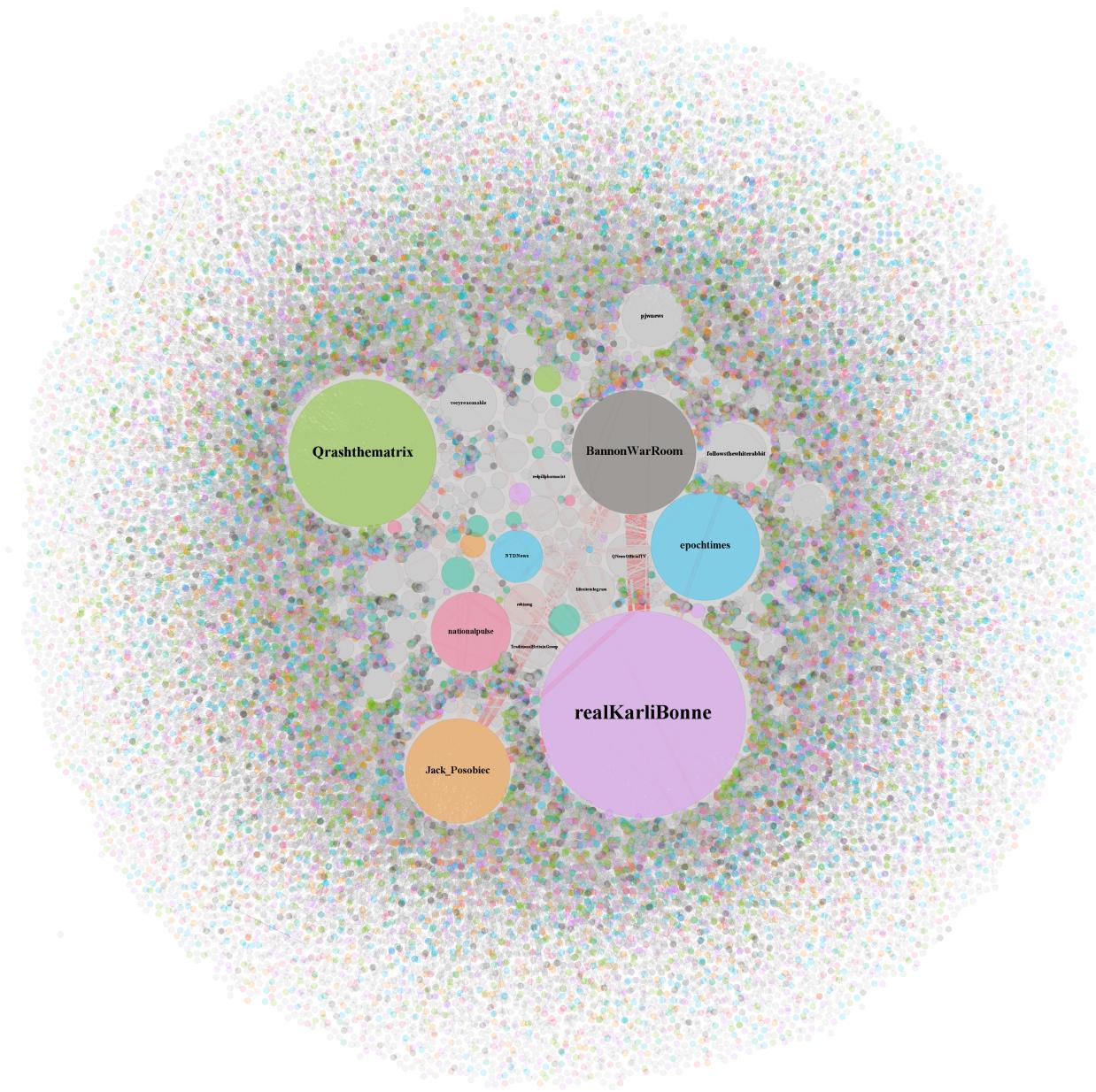
Questo approccio permette di integrare informazioni complesse in un formato più facilmente interpretabile, utile per identificare nodi particolarmente influenti o strategici nella rete.

Inoltre, la combinazione dei due file consente una pipeline analitica ben definita, in cui ogni fase aggiunge un livello di approfondimento, migliorando la comprensione della struttura e delle dinamiche della rete.

4.3 Grafo finale

Segue il grafo finale utilizzato per la raccolta delle metriche conclusive.

Si è passati da un grafo orientato a uno non orientato per semplificare l'analisi e ottenere una visione complessiva più chiara del risultato finale. Il grafo, a differenza dei precedenti, presenta degli archi tra canali che contengono lo stesso numero di utenti. L'obiettivo iniziale era di creare anche archi tra utenti che condividono le stesse community, ma non è stato possibile a causa della mole di dati presenti.



Il grafo presenta il canale "RealKarliBonne" come nodo principale, con "Qrashthematrix" e "BannonWarRoom" a seguire. Questi si trovano al centro e mostrano un elevato grado di interconnessione, indicando un ruolo predominante nella rete. I cluster periferici, invece, rappresentano comunità più specifiche o isolate, che interagiscono principalmente con pochi nodi di ponte.

Gli archi con minore peso sono stati rappresentati con il colore grigio, mentre gli archi con grado mediamente pesante in rosso e i più pesanti in blu. Come è possibile osservare, non esistono archi blu, mostrando come le community non siano totalmente uguali. Per analizzare la rete sono state calcolate le seguenti metriche:

- **Degree (Indegree e Outdegree):** Misura rispettivamente il numero di connessioni

in ingresso e in uscita per ciascun nodo; **realKarliBonne**, con un degree totale pari a 4.230, si distingue come il nodo più connesso.

- **Authority:** Valuta l'importanza di un nodo all'interno del network. È interessante notare come **BannonWarRoom** presenti un valore maggiore di **Qrashthematrix**, nonostante il secondo sia di grado più elevato rispetto al primo.
- **Clustering Coefficient:** Quantifica la densità delle connessioni locali. Nodi periferici con coefficiente di clustering pari a 1 suggeriscono triadi completamente connesse.
- **Eigenvector Centrality:** Misura l'importanza di un nodo considerando anche il prestigio dei suoi vicini. **realKarliBonne** emerge con un'eigenvector centrality massima, indicando una posizione strategica per la diffusione dell'informazione.
- **Betweenness Centrality:** Valuta il ruolo di ponte dei nodi tra diverse comunità. Nodi come **NevsChannel** hanno un'alta betweenness, evidenziando la loro funzione di collegamento.
- **Modularità:** Calcolata con l'**algoritmo di Louvain**, ha permesso di dividere il grafo in diverse comunità. Ad esempio, i nodi **realKarliBonne** e **Qrashthematrix** appartengono a cluster distinti ma fortemente connessi come è possibile comprendere dalle colorazioni differenti e dall'arco che li connette.

Il grafo è stato reso leggermente opaco, permettendo così agli archi rossi di essere maggiormente visibili.

Id	type	indegree	outdegree	degree	weight	weighted	weight	Eccentricity
1534624995	User	0	2	2	0	17	17	7
NevsChannel	Channel	235	53	288	3855	213	4068	6
realKarliBonne	Channel	4156	74	4230	39114	1328	40442	7
1758896015	User	0	3	3	0	35	35	7
Blazenzherb	Channel	77	39	116	865	63	928	6

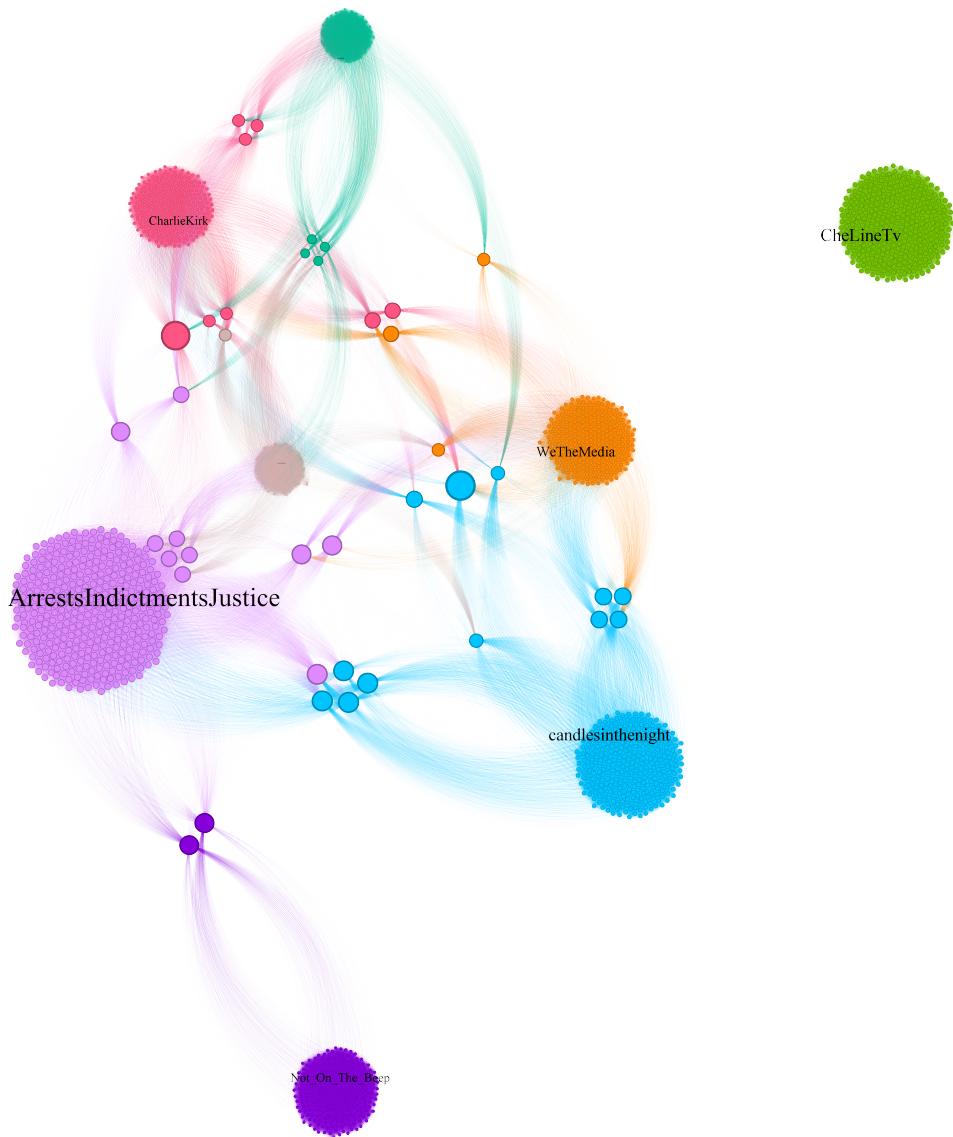
closeness	harmonic	cllo	betweenes	Authority	Hub	pagerank	modularit	stat_inf	cl	clustering	triangles	eigence
325598	338881	0	6565	6565	0,0002	85	96	10	1	1	12264	
411994	435684	5,168E+12	68618	68618	1989	85	33	3936	1440	145456		
474769	526512	1,093E+14	439484	439484	35512	138	96	596	5211	10		
322378	334252	0	4226	4226	0,0003	66	1	10	3	3	10475	
392562	416982	2,051E+12	49439	49439	843	66	1	132122	851	100069		

4.3.1 Grafo filtrato

All'interno del seguente grafo, oltre alle interazioni di tipo utente-canale e canale-canale, troviamo anche quelle che sono le interazioni utente-utente.

Abbiamo estratto 8 community casuali per evitare problematiche legate alle dimensioni eccessive del file, che avrebbero potuto compromettere la fluidità dell'analisi e della visualizzazione tramite Gephi.

Le metriche calcolate su di esso risultano essere le stesse precedentemente citate.



Ogni nodo rappresenta un'entità all'interno del grafo, come un utente o un canale.

I nodi canale principale sono fortemente connessi alle proprie community, rendendo difficile la loro visualizzazione. I nodi utente risultano avere anche un grado maggiore rispetto a quelli canale, viste le loro connessioni con più nodi e con altri utenti.

"ArrestsIndictmentsJustice", "WeTheMedia" e "candlesinthenight" sono visiva-

mente dominanti come community, suggerendo che abbiano un ruolo di rilievo nelle interazioni.

Un nodo isolato visibile in alto a destra ("recl.lineTV") suggerisce una community o un'entità meno connessa con il resto della rete.

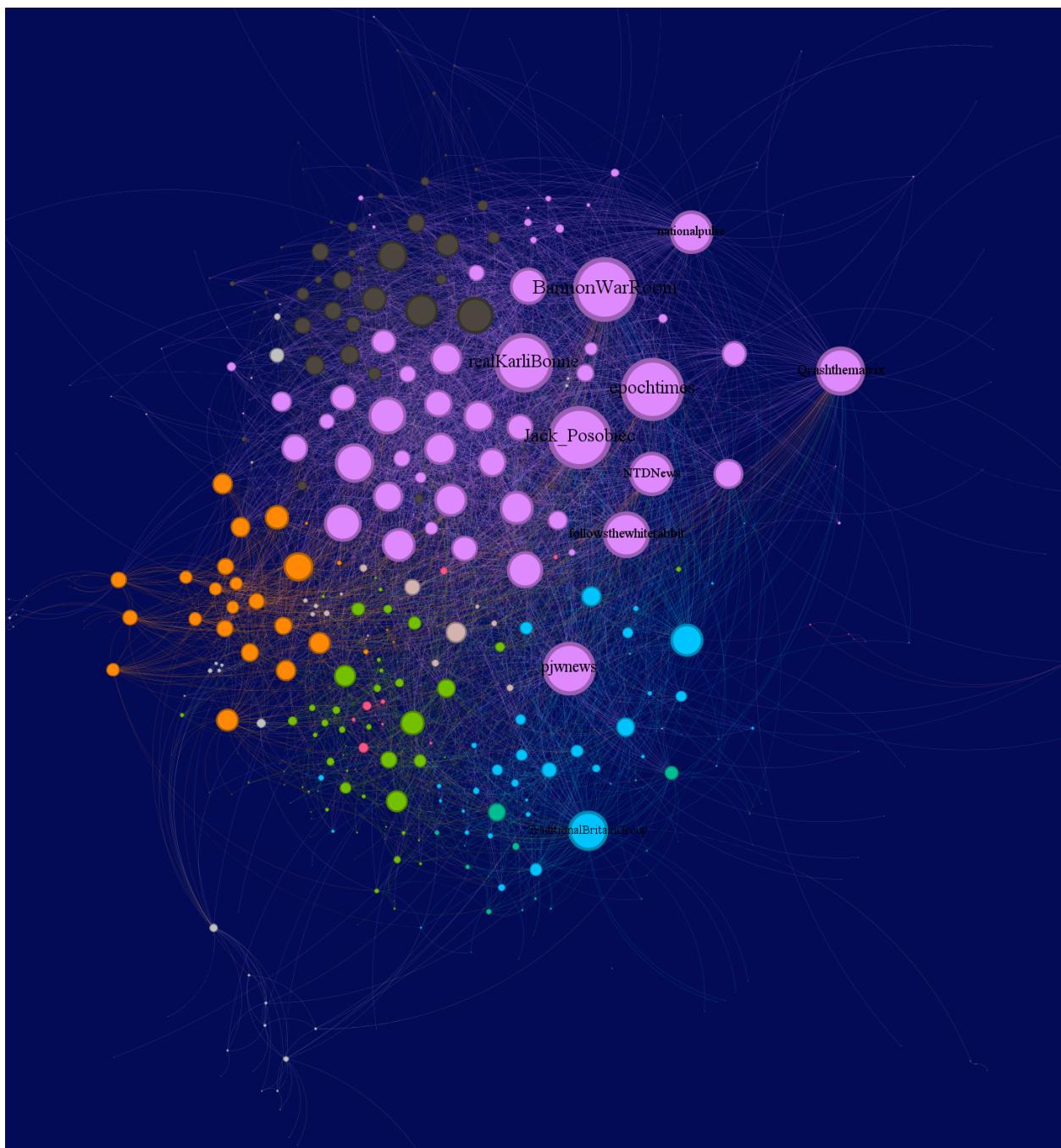
Le community mostrano una chiara separazione visiva, con alcuni nodi utente che interagiscono con non più di 3 community uguali, indicando una forte modularità. Ciò suggerisce che i nodi all'interno di ogni cluster interagiscono più frequentemente tra loro rispetto ai nodi di altri cluster.

La separazione in community è netta, indicando che le interazioni sono fortemente centralizzate in gruppi con interessi o connessioni comuni.

4.3.2 Grafo dei Canali

Il seguente grafo è formato esclusivamente dai nodi di tipo canale, le cui connessioni sono rappresentate da archi creati dalla quantità di utenti che hanno in comune. Ai fini di una migliore visualizzazione è stato impostato uno sfondo blu.

I nodi principali, ovvero quelli con grado maggiore, osservati precedentemente vengono ora riportati quasi tutti nella stessa community in base all'algoritmo di Modularity.

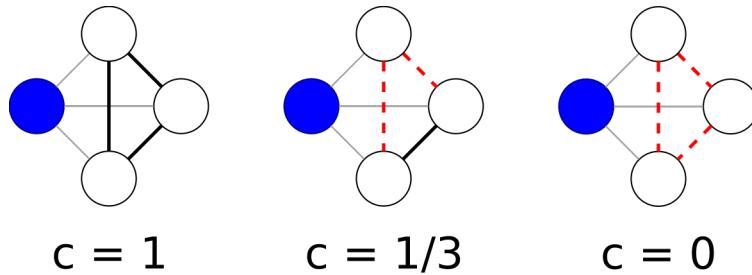


5 Clustering

Quando si parla di clustering, si intende una misura che indica quanto i vicini di un nodo siano connessi tra loro, rappresentando quindi il grado di "*interconnessione*" locale attorno a un nodo.

In particolare, il coefficiente di clustering di un nodo varia tra 0 (**nessuna connessione tra i vicini**) e 1 (**i vicini formano un grafo completamente connesso**).

Il seguente codice calcola il coefficiente di clustering per ogni nodo del grafo e il valore medio di clustering del grafo complessivo.



Questo è particolarmente utile quando il grafo rappresenta comunità online, come nel caso di gruppi Telegram, perché consente di identificare la coesione interna delle reti di utenti.

Ad esempio, un alto coefficiente di clustering può indicare comunità molto affiatate, mentre un basso coefficiente suggerisce un'interazione più dispersa o eterogenea.

I risultati vengono salvati in un file CSV per ulteriori analisi, permettendo di identificare i nodi con il più alto e il più basso clustering, utili per studiare il ruolo di specifici utenti nella rete.

Lo script sviluppato per poter calcolare il coefficiente di clustering viene utilizzato su tutte le casistiche precedentemente analizzate, andando solamente a modificare il file.csv di input.

```
1 def analyze_graph_clustering(graphml_path):
2     try:
3         G = nx.read_graphml(graphml_path)
4     except Exception as e:
5         print(f"Errore durante la lettura del file GraphML: {e}")
6         return None, None
7
8     node_clustering = nx.clustering(G)
9
10    avg_clustering = nx.average_clustering(G)
11
12    clustering_df = pd.DataFrame.from_dict(node_clustering,
13        orient='index',
14        columns=['clustering_coefficient'])
15
16    clustering_df.index.name = 'Canali'
17
18    clustering_df = clustering_df.sort_values('
        clustering_coefficient',
```

```

19     ascending=False)
20
21     print("\nAnalisi del Clustering:")
22     print(f"Coefficiente di clustering medio del grafo:{avg_clustering:.4f}")
23
24     save_clustering_results(clustering_df, avg_clustering)
25
26     return node_clustering, avg_clustering
27
28 def save_clustering_results(clustering_df, avg_clustering):
29     try:
30         avg_row = pd.DataFrame({'clustering_coefficient': [
31             avg_clustering]}, index=[average_clustering])
32         final_df = pd.concat([clustering_df, avg_row])
33
34         final_df.to_csv("csv/clustering.csv", index_label="node_id")
35         print("\nRisultati salvati nel file.")
36     except Exception as e:
37         print(f"Errore durante il salvataggio dei risultati:{e}")
38
39 if __name__ == "__main__":
40     file_path = "grafo.graphml"
41     node_clustering, avg_clustering = analyze_graph_clustering(
        file_path)

```

5.1 Clustering grafo finale

Questo clustering è stato eseguito sul grafo finale e dimostra la varietà di valori di coefficienti di clustering presenti, mostrando sia utenti molto connessi (con valore 1) sia utenti isolati (con valore 0).

Source	Clustering_coefficient
1186198	1
72269003	0.5
121649098	0.83
109389246	0

5.2 Clustering grafo filtrato

Questo clustering è stato eseguito sul grafo filtrato rappresentante solo 8 community casuali e i loro utenti.

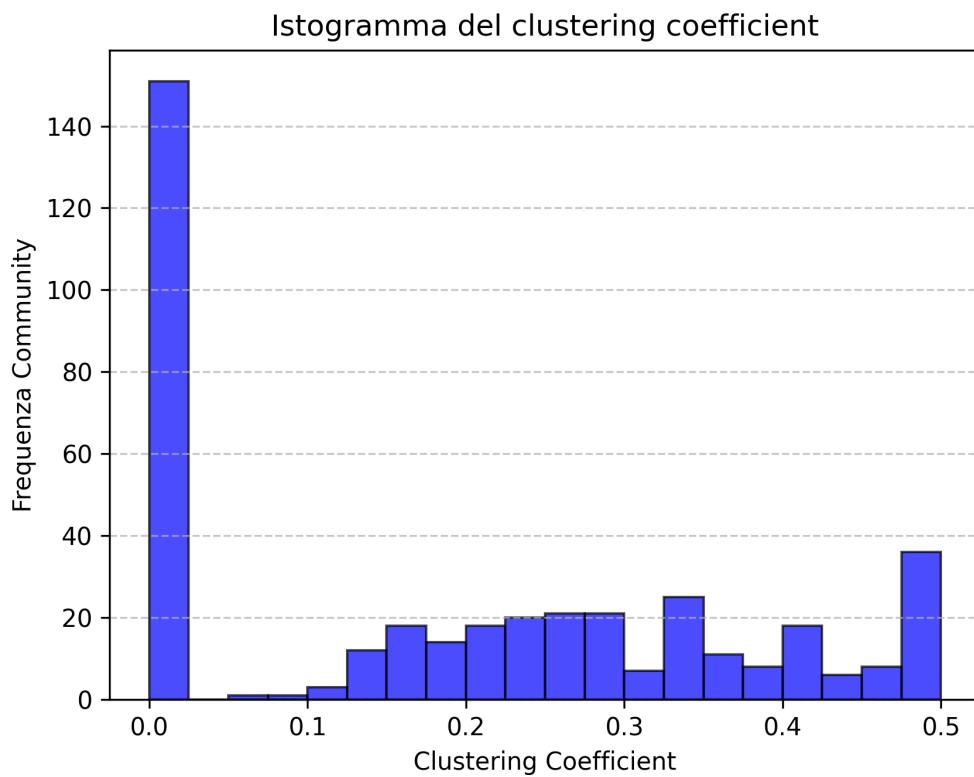
Diversamente dal grafo precedente, i valori, sempre compresi tra 0 e 1, non arrivano in realtà mai ad 1.

Esistono solo utenti poco connessi o quasi fortemente connessi.

Source	Clustering_coefficient
1618281880	0.5
7179197662	0.0
1550520627	0.25
609517172	0.17

Il coefficiente di clustering è stato, successivamente, calcolato anche sulle community del file "community_chiusura.py".

I risultati ottenuti sono facilmente visibili nel seguente istogramma:



6 Tasso di chiusura

In questa sezione verrà analizzato e descritto nel dettaglio il codice sviluppato per calcolare un indice denominato "**tasso di chiusura**" per diverse community, andandosi a basare sulle interazioni tra utenti.

Il tasso di chiusura su un grafo di community misura il grado di connessione interna di una community rispetto alle connessioni esterne, indicando quanto una community sia "chiusa" rispetto al resto del grafo.

6.1 Metodo basato sui grafi

Si va, dunque, a descrivere il funzionamento di uno script Python sviluppato per calcolare il "tasso di chiusura" dei nodi di tipo "Channel" presenti nel dataset.

Questo processo include la lettura di un file CSV, la normalizzazione delle metriche rilevanti e l'integrazione dei risultati nel dataset originale.

Si inizia con il caricamento dei dati:

```
1  file_path = "csv/gephi_data.csv"
2  df = pd.read_csv(file_path)
```

Si filtra il DataFrame per includere solo i nodi con valore "Channel" nella colonna type.

Si utilizza .copy() per evitare avvisi di tipo "*SettingWithCopyWarning*":

```
1  channel_nodes = df[df["type"] == "Channel"].copy()
```

Successivamente, lo script verifica la presenza delle colonne necessarie.

In caso contrario, genera un errore informativo per l'utente.

```
1  required_columns = ["degree", "closenesscentrality", "
2    betweenesscentrality", "Authority", "eigencentrality"]
3  if not all(col in channel_nodes.columns for col in
4    required_columns):
5    raise ValueError("Il file CSV deve contenere le seguenti
6      colonne: %s" % ', '.join(required_columns))
```

Le metriche elencate vengono normalizzate nell'intervallo [0, 1] utilizzando **MinMaxScaler**.

La normalizzazione garantisce che tutte le metriche abbiano lo stesso peso durante il calcolo:

```
1  metrics = ["degree", "closenesscentrality", "betweenesscentrality"
2    , "Authority", "eigencentrality"]
3  scaler = MinMaxScaler()
4  normalized_metrics = scaler.fit_transform(channel_nodes[metrics])
```

Il tasso di chiusura viene calcolato come complemento alla media aritmetica delle metriche normalizzate per ogni nodo:

```

1   channel_nodes["Tasso_di_chiusura"] = (1 - normalized_metrics.mean
                                         (axis=1))

```

La nuova colonna **Tasso** di chiusura viene aggiunta al DataFrame originale attraverso un'operazione di merge, basata sull'identificativo del nodo (**Id**):

```

1   df = df.merge(channel_nodes[["Id", "Tasso_di_chiusura"]], on="Id"
                  , how="left")

```

6.2 Metodo matematico

Data la definizione $U_c = \{u \in Source \mid Target = c\}$ che va ad identificare il set U_c contenente tutti gli utenti u appartenenti alla colonna Source solo se questi hanno interagito almeno una volta con il canale C della colonna Target.

Canali totali di $u = |\{c \in Target \mid u \text{ ha interagito con } c\}|$ conta il numero di canali distinti di ciascun utente u tali che u ha interagito almeno una volta con il canale c .

$$\text{Tasso di chiusura di un canale } C = \frac{|U_C|}{\sum_{u \in U_c} \text{Canali totali di } u}$$

- $|U_C|$: cardinalità del set di utenti di C
- $\sum_{u \in U_c}$: somma totale di un valore per tutti gli utenti del set U_C

Iniziamo con il caricare i dati:

```

1   df = pd.read_csv("archs_new.csv")
2   df = df[df.type == "Reply"]
3   df_community = pd.read_csv("community.csv")

```

- **archs_new.csv**: contiene informazioni sulle connessioni tra utenti e canali, di queste solo quelle di tipo "Reply" vengono considerate per le successive analisi
- **community.csv**: viene utilizzato per generare un nuovo file contenente informazioni aggiuntive sulle community

I file csv vengono caricati in due dataframe denominati **df** e **df_community**.

Si prosegue, poi, con il calcolo degli utenti per canale:

```

1   users_per_channel = df.groupby("target")["source"].apply(set).
                           to_dict()

```

Questo passaggio raggruppa i dati in base ai canali destinatari (**target**) e crea un dizionario in cui ogni chiave rappresenta un canale, il cui valore è l'insieme degli utenti (**source**) che hanno interagito con quel canale.

Si passa al calcolo dei canali per utente:

```

1   channels_per_user = df.groupby("source")["target"].apply(set).
                           to_dict()

```

In modo simile, questo codice calcola per ogni utente l'insieme dei canali (**target**) con cui ha interagito. Il risultato è un dizionario in cui la chiave è un utente e il valore è un insieme di canali.

Si calcola, poi, il tasso di chiusura:

```
1 closure_scores = {}
2     for channel, users in users_per_channel.items():
3         total_channels = sum(len(channels_per_user[user]) for
4             user in users)
5         closure_scores[channel] = len(users) / total_channels
```

Quest'ultima è una metrica che quantifica la concentrazione degli utenti in un canale rispetto al numero totale di canali con cui interagiscono.

- Si somma il numero di canali associati a ciascun utente del canale corrente
- Il tasso di chiusura è definito come il rapporto tra il numero di utenti unici del canale e il numero totale di canali con cui questi utenti interagiscono

Ultimo passo è l'integrazione del tasso di chiusura precedentemente calcolato:

```
1 df_community["Tasso_di_chiusura"] = df_community["Community"].map
    (closure_scores)
```

I valori calcolati vengono mappati al DataFrame **df_community**, aggiungendo una nuova colonna denominata "Tasso di chiusura".

Solo le community che hanno almeno 45 messaggi vengono considerate:

```
1 df_community = df_community[df_community["Messaggi"] >= 45].
    dropna(subset=["Tasso_di_chiusura"])
```

I dati filtrati e arricchiti con il tasso di chiusura vengono salvati in un nuovo file csv denominato **community_chiusura.csv**.

6.3 Analisi dei risultati

Il **Coefficiente di Pearson** tra due variabili statistiche è un indice che esprime un'eventuale relazione lineare tra esse.

Secondo la **disuguaglianza di Cauchy-Schwarz** ha un valore compreso tra +1 e -1, dove **+1** corrisponde alla **perfetta correlazione lineare positiva**, **0** corrisponde a un'**assenza di correlazione lineare** e **-1** corrisponde alla **perfetta correlazione lineare negativa**.

Date due variabili statistiche X e Y , l'indice di correlazione di Pearson è definito come la loro covarianza divisa per il prodotto delle deviazioni standard delle due variabili:

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$$

Se:

- $\rho_{XY} > 0$:** le variabili X e Y si dicono direttamente correlate, oppure correlate positivamente;
- $\rho_{XY} < 0$:** le variabili X e Y si dicono inversamente correlate, oppure correlate negativamente
- $\rho_{XY} = 0$:** le variabili X e Y si dicono incorrelate

E' stato condotto uno studio approfondito volto a determinare la *correlazione tra il tasso di apertura e due diverse metodologie di chiusura: quella matematica e quella basata sui grafi*. Il calcolo è stato implementato mediante un codice sviluppato appositamente per l'elaborazione dei dati e l'applicazione di metodi statistici.

```
1 correlazione_matematica = np.corrcoef(df_community["Score_medio"]
2                               , df_community["Tasso_di_chiusura"])[0, 1]
3 correlazione_gephi = np.corrcoef(df_community["Score_medio"],
4                                 df_community["Chiusura_Gephi"])[0, 1]
```

I risultati ottenuti, salvati nel file **correlazione.txt**, evidenziano una correlazione quasi inesistente, calcolata mediante il coefficiente di Pearson, pari a:

Descrizione	Valore
Correlazione con chiusura matematica	0.060985
Correlazione con chiusura basata sui grafi	0.056714

Si può concludere dicendo che con i dati a nostra disposizione, è impossibile sottolineare la presenza di un effettivo collegamento tra chiusura delle community e basso score Newsguard dei siti condivisi al loro interno.

7 Conclusione e valutazione finale del team

Di seguito è giusto riportare i risultati ottenuti tramite l'elaborazione dei dati a nostra disposizione:

- **10.518.501 messaggi** (36.078 GB di materiale) raccolti tramite Open Measures
- **2.470.949 messaggi** elaborati grazie a NewsGuard
- **3.768 domini** di NewsGuard analizzati raccolti in **2.669.367 occorrenze**
- **3.978 community** analizzate
- **39.077 nodi** di Gephi
- **440.672 archi non pesati**, che diventano **51.499 se pesati** tramite Gephi

L'elevata mole di dati raccolti su tutto il 2024 ci permette di trarre una prima conclusione, che potrebbe essere approfondita meglio in un eventuale lavoro di ricerca effettivo, riguardo l'esistenza o meno di una effettiva correlazione tra la chiusura delle community Telegram e la disinformazione al loro interno.

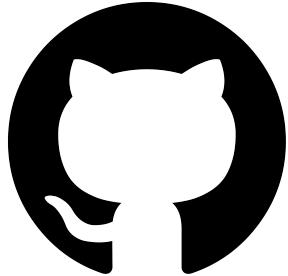
La nostra tesi, infatti, ipotizzava l'esistenza di uno stretto rapporto che vedeva, all'aumentare della chiusura interna di una community Telegram, il diminuire della veridicità delle notizie medie che circolano al suo interno.

Le evidenze raccolte e l'analisi che n'è stata fatta sono state fondamentali per il calcolo del Coefficiente di correlazione di Pearson tra i valori in esame (score del sito e tasso di chiusura della community), andando ad evidenziare una correlazione quasi inesistente, e di conseguenza trascurabile, che va a confutare la tesi di partenza.

Tale risultato potrebbe maggiormente essere espanso mediante la raccolta di più dati che vede l'espansione del set di termini di ricerca iniziale. Tuttavia, vista la considerevole quantità di materiale raccolto, ci viene naturale considerare con sicurezza il risultato ottenuto valido.

Questo studio invita, quindi, a riflettere sulle dinamiche della disinformazione nelle community chiuse, aprendo nuove strade per futuri studi e analisi più approfondate.

8 Collegamenti esterni



Codice completo:

<https://github.com/BurgOwO/information-disorder>