YouTube Video Classification on Twitter

Final Project Big Data Engineering

University Federico II of Naples

Giuliano Di Giuseppe
Raffaele Russo
Alessandro Vanacore

supervised by
Vincenzo Moscato

Contents

1 Introduction

1.1 Context

Social media platforms play a significant role in shaping the modern digital information ecosystem by allowing users to contribute to discussions on a wide range of topics. However, the freedom of expression offered by these platforms can potentially threaten the integrity of these information ecosystems when harmful content (e.g., fake news, hateful speech) is shared and propagates across the digital population. To overcome this problem, mainstream social media platforms (e.g., Facebook and Twitter) deploy diverse moderation interventions to target both inappropriate content and the users responsible for spreading it. However, these moderation efforts are typically enacted in a siloed fashion, largely overlooking other platforms' interventions on harmful content that has migrated to their spaces [1, 2]. This approach poses risks as any harmful content originating on a source platform can migrate to other target platforms, gaining traction with specific communities and reaching a wider audience [3, 4]. Cooperation among social media platforms is therefore desirable but also practically valuable: knowing what content has been deemed inappropriate on another platform can inform moderation strategies, or help with the early detection of similarly harmful, or related content. The general objective of the challenge is to assess the validity of the above mentioned statement. Specifically, we consider YouTube (YT) and Twitter as the source and target platforms. The aim of this challenge is to predict whether a YouTube video shared on Twitter will be removed by YouTube.

1.2 Dataset

The dataset comprises three files:

- train.csv: a table with three columns, i.e., the video identifier, the video URL(s) (the same video can be shared through different URLs), the video status (the video could either moderated or not moderated).
- test.csv: it shares the same structure of train.csv without the last column.
- df youtube.csv.zip: the archive with all tweets sharing the videos listed in the above-mentioned files.

2 Tools

2.1 MongoDB

One of the key motivations for employing MongoDB in our research was its ability to handle large volumes of data and execute complex querying operations efficiently. We were working with two distinct datasets: one containing tweets referring to YouTube videos and another containing the YouTube videos themselves.

MongoDB's capabilities allowed us to efficiently map each YouTube video to its corresponding list of tweets using flexible querying operations. This approach was preferred over Neo4j, as we needed a solution that could handle not only the graph-like relationships between the datasets but especially the large volume of data and complex queries involved in the mapping process.

After mapping the YouTube videos to their respective tweets, we used this information to create a graph in Python. This graph was then utilized with various models, including those that work with tabular data, as well as graph network models. The incorporation of MongoDB into our research workflow exemplifies our commitment to employing innovative and efficient technologies to enhance the quality and effectiveness of our analysis.

2.2 Pandas

The initial approach employed the pandas library to perform various operations on the dataset. However, as the analysis progressed, it became apparent that performing joins between YouTube videos and tweets using pandas proved to be inefficient.

The inefficiency arose primarily from the size of the dataset and the complexity of the join operations required. Joining large volumes of data from both the YouTube videos and tweets, which involved matching and merging based on specific criteria, posed significant computational challenges.

2.3 Polaris

Another approach used was Polaris, which is known for its powerful data processing capabilities and optimized algorithms, and offered potential advantages for our analysis.

However, despite the promising features of Polaris, we found that computation times remained relatively high. The complexity of the join operations, coupled with the large amount of data to be processed, continued to pose problems in achieving optimal performance. As a result, the gains achieved with the Polaris library were not sufficient to overcome the inherent computational limitations.

To overcome these inefficiencies and improve the join performance, alternative approaches were explored, leveraging more efficient data processing frameworks and algorithms. By utilizing specialized tools and techniques specifically designed for handling large-scale data joins, such as Apache Spark or no relational database, the subsequent analysis could achieve significantly improved computational efficiency and reduced execution times.

2.4 Pytorch geometric

PyG (PyTorch Geometric) is a library built upon PyTorch to easily write and train Graph Neural Networks (GNNs) for a wide range of applications related to structured data. It was so possibile to build a graph which models the relationships among heterogeneus entities of our dataset.

3 Data set analysis

3.1 Youtube Dataset

- id - id of youtube video
- lista_url - list of the youtube video urls
- moderationStatus - moderate or not-moderated

3.2 Twitter Dataset

The columns in the dataset are described in detail in the Twitter API documentation[1] which provides comprehensive information about each column, including their data types, purposes, and possible values. It serves as a valuable resource for understanding the structure and content of the dataset.

Please refer to the documentation for a thorough description of each column and its significance within the context of Twitter data analysis.

1. tweetid - Tweet identifier
2. userid - User identifier
3. screen_name - User's screen name
4. date - Date of the tweet
5. lang - Language of the tweet
6. location - User's location
7. description - User's description
8. place_id - Place identifier
9. place_url - URL associated with the place
10. place_type - Type of place
11. place_name - Name of the place
12. place_full_name - Full name of the place
13. place_country_code - Country code of the place
14. place_country - Country of the place
15. place_bounding_box - Bounding box coordinates of the place
16. text - Text of the tweet

[1]Twitter API v1 Data Dictionary: `https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/tweet`

17. extended - Indicates if the tweet is extended
18. coord - Coordinates associated with the tweet
19. reply_userid - User ID being replied to
20. reply_screen - Screen name being replied to
21. reply_statusid - Status ID being replied to
22. tweet_type - Type of tweet
23. friends_count - Number of friends of the user
24. listed_count - Number of lists the user is in
25. followers_count - Number of followers of the user
26. favourites_count - Number of tweets favorited by the user
27. statuses_count - Number of tweets by the user
28. verified - Indicates if the user is verified
29. hashtag - Hashtags used in the tweet
30. urls_list - List of URLs in the tweet
31. profile_pic_url - URL of the user's profile picture
32. profile_banner_url - URL of the user's profile banner
33. display_name - User's display name
34. date_first_tweet - Date of the user's first tweet
35. account_creation_date - Date of the user's account creation
36. rt_urls_list - List of URLs in retweets
37. mentionid - User ID mentioned in the tweet
38. mentionsn - Screen name mentioned in the tweet
39. rt_screen - Screen name of the retweet
40. rt_userid - User ID of the retweet
41. rt_user_description - Description of the retweet user
42. rt_text - Text of the retweet
43. rt_hashtag - Hashtags in the retweet
44. rt_qtd_count - Retweet count of the retweet

45. rt_rt_count - Retweet count of the retweet
46. rt_reply_count - Reply count of the retweet
47. rt_fav_count - Favorite count of the retweet
48. rt_tweetid - Tweet ID of the retweet
49. rt_location - Location of the retweet
50. qtd_screen - Screen name of the quote tweet
51. qtd_userid - User ID of the quote tweet
52. qtd_user_description - Description of the quote tweet user
53. qtd_text - Text of the quote tweet
54. qtd_hashtag - Hashtags in the quote tweet
55. qtd_qtd_count - Quote tweet count of the quote tweet
56. qtd_rt_count - Retweet count of the quote tweet
57. qtd_reply_count - Reply count of the quote tweet
58. qtd_fav_count - Favorite count of the quote tweet
59. qtd_tweetid - Tweet ID of the quote tweet
60. qtd_urls_list - List of URLs in the quote tweet
61. qtd_location - Location of the quote tweet
62. sent_vader - Sentiment score using VADER
63. token - Tokens in the tweet text
64. media_urls - URLs of media attached to the tweet
65. rt_media_urls - URLs of media in retweets
66. q_media_urls - URLs of media in quote tweets
67. state - State information
68. country - Country information
69. rt_state - State information of the retweet
70. rt_country - Country information of the retweet
71. qtd_state - State information of the quote tweet
72. qtd_country - Country information of the quote tweet

73. norm_country - Normalized country information
74. norm_rt_country - Normalized country information of the retweet
75. norm_qtd_country - Normalized country information of the quote tweet
76. acc_age - Age of the Twitter account
77. domains - Domains in the tweet
78. rt_domains - Domains in retweets

3.3 Dataset Analysis with PySpark

```
dataset = spark.read \
  .option("inferSchema", "true") \
  .option("header", "true") \
  .option("sep", ",") \
  .csv('/content/drive/MyDrive/BDE/tw_collection_w_status.csv')

dataset.printSchema()
```

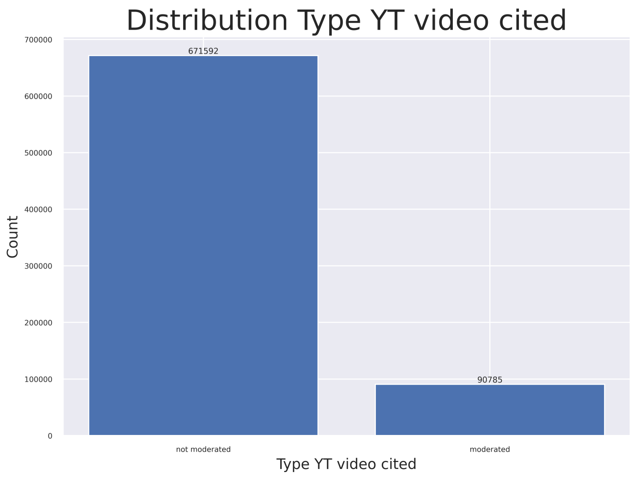


Figure 1: Distribution Youtube videos status

```
grouped_df1 = dataset.groupBy("moderated").agg(count("userid")
                    .alias("count")).orderBy(col("moderated"))
grouped_df1 =
grouped_df1.withColumnRenamed("moderated",
                "moderated_value").reset_index(drop=True)
```

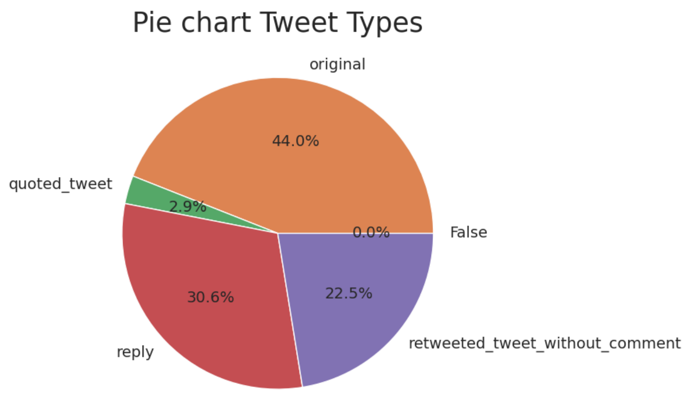
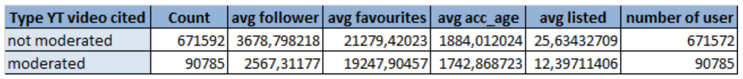


Figure 2: Pie chart tweet type

```
grouped_df2 = dataset.groupBy("tweet_type").agg(count("*").alias("count"))
```



Type YT video cited	Count	avg follower	avg favourites	avg acc_age	avg listed	number of user
not moderated	671592	3678,798218	21279,42023	1884,012024	25,63432709	671572
moderated	90785	2567,31177	19247,90457	1742,868723	12,39711406	90785

Figure 3: Statistics

```
grouped_df = dataset.groupBy(dataset["moderated"]).agg(
    count("_id").alias("count"),
    avg("followers_count").alias("avg_followers_count"),
    avg("favourites_count").alias("avg_favourites_count"),
    avg("acc_age").alias("avg_acc_age"),
    avg("listed_count").alias("avg_listed_count")
```

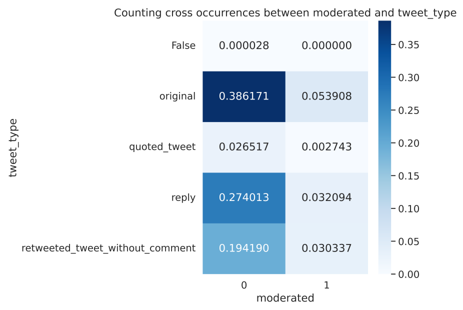


Figure 4: Cross Table

4 Pre-Processing

4.1 Storage Data

We established a connection to MongoDB to store and analyze Twitter and YouTube data. To store the Twitter data, we created a collection specifically designed for this purpose. Each document in the collection represents a single tweet and contains various fields such as text, user information, and metadata. In order to enhance the analysis of the tweets, we introduced a new column called *'expanded_urls'* in the Twitter collection. This column stores a list of URLs referenced by each tweet, extracted from the *'urls_list'* column. The URLs are only populated in cases where the *'urls_list'* field is not null. This additional column provides valuable information for further analysis, allowing us to examine the web resources shared within the tweets. Similar to the Twitter collection, we have a separate collection to store the YouTube data. This collection holds information about id and label. Storing the YouTube data separately enables efficient organization and retrieval of the specific information related to YouTube videos. By leveraging the two collections we created, we executed a query to establish associations between YouTube videos and the tweets that reference them. This association was made possible through the utilization of the *'expanded_urls'* column we introduced earlier. The query resulted in the creation of a new column in the YouTube collection, named *'tweet_ids'*, which contains a list of tweet IDs corresponding to the tweets referencing each video.

4.2 Collections filtering

These processes involve the manipulation and organization of the data to facilitate the generation of a graph for model training. The following modifications were performed:

1. Encoding the *'moderationStatus'* column: The *'moderationStatus'* column values were encoded as follows: 'not moderated' to 0, 'moderated' to 1. This transformation enables quantitative analysis of the moderation status of the tweets and saving the collection as a CSV file named *'update_merged'*.

2. Saving as JSON file: In addition to the CSV format, the processed collection is also saved as a JSON file named *'collection_data'*. This file format preserves the hierarchical structure of the data and allows for future processing and compatibility with other tools and platforms.

3. Filtering tweets based on tweet types: To focus our analysis, we filtered out tweets that do not belong to the following tweet types expressed in the *'tweet_type'* column: [*reply, original, retweeted_tweet_without_comment, quoted tweet*]. This filtering ensures that we consider only relevant and valid tweet types for our research objectives.

4. Dropping insignificant columns: Several columns were deemed non-significant for our analysis and were dropped from the Twitter dataset. The left columns include: [*'tweetid', 'userid', 'date', 'description', 'text', 'reply_statusid', 'tweet_type', 'friends_count', 'listed_count', 'followers_count', 'favourites_count', 'statuses_count', 'verified', 'hashtag', 'date_first_tweet', 'account_creation_date', 'rt_qtd_count', 'rt_rt_count', 'rt_reply_count', 'rt_fav_count', 'rt_tweetid', 'qtd_qtd_count', 'qtd_rt_count', 'qtd_reply_count', 'qtd_fav_count', 'qtd_tweetid'*].

   We retain only the relevant information for our analysis. Saving the processed tweets as a CSV file *'tweets_filtered'*.

1. YouTube collection: The *'update_merged'* collection is used to create the YouTube collection, which contains relevant data related to YouTube videos.

   Example structure of Youtube video:

```
{'_id': ObjectId('6470c440ab1a9c0a9da6d1a6'),
 'id': '2bFLr70bNzA',
 'moderationStatus': '0',
 'tweet_ids': "['1319671748170797062',
 '1319504474956845056', '1319431259194609664',
 '1319537482082492416', '1319452511527460864',
 '1319664735915159553', '1319807337243181062']"}
```

2. Twitter collection: The *'tweets_filtrati'* collection serves as the basis for creating the Twitter collection, which includes essential data related to the filtered tweets.

   Example structure of Tweet:

```
{'_id': ObjectId('6470ce68ab1a9c0a9db37691'),
 'tweetid': '1318295840067186689',
 'userid': '182453941',
 'date': 'Mon Oct 19 20:59:55 +0000 2020',
 'description': 'SGT RET MP VET FEMALE #MAGA,
 #AMERICAFIRST #SOVERITY   #STOPNWO #HATEPC
 #TRUMP2020 #KAG #Q #2A #NATIONALISM #BLOCKPORN
 iraqveteran proud to be followed gen flynn',
 'text': 'RT @kasmouse: Watch LIVE: President Trump Holds Make
 America Great Again Rally in Tu... https://t.co/
 Flzr4GUBi9 via @YouTube',
 'reply_statusid': '',
 'tweet_type': 'retweeted_tweet_without_comment',
 'friends_count': '22144',
 'listed_count': '24',
 'followers_count': '19927',
 'favourites_count': '95842',
 'statuses_count': '319393',
 'verified': 'False',
 'hashtag': '[]',
 'date_first_tweet': 'Mon Oct 19 20:59:55 +0000 2020',
 'account_creation_date': 'Tue Aug 24 16:49:46 +0000 2010',
 'rt_qtd_count': '0.0',
 'rt_rt_count': '1',
 'rt_reply_count': '0.0',
 'rt_fav_count': '1.0',
 'rt_tweetid': 1318295714976342016,
 'qtd_qtd_count': '0.0',
 'qtd_rt_count': '0.0',
 'qtd_reply_count': '0',
 'qtd_fav_count': '0',
 'qtd_tweetid': ''}
```

3. User collection: The user collection is derived from the Twitter collection by leveraging the *'userid'* field. This collection stores user-related information for analysis purposes.

Example structure of User:

```
{'_id': ObjectId('6470d5e1ab1a9c0a9dd7e531'),
 'userid': '182453941',
 'description': 'SGT RET MP VET FEMALE #MAGA, #AMERICAFIRST
 #SOVERITY #STOPNWO #HATEPC #TRUMP2020 #KAG #Q #2A #NATIONALISM
 #BLOCKPORN iraqveteran proud to be followed gen flynn',
 'verified': 'False',
 'friends_count': '22144',
 'listed_count': '24',
 'statuses_count': '319393',
 'followers_count': '19927',
 'favourites_count': '95842',
 'date_first_tweet': 'Mon Oct 19 20:59:55 +0000 2020',
 'account_creation_date': 'Tue Aug 24 16:49:46 +0000 2010'}
```

Saving the collections as JSON files: The three collections (YouTube, Twitter, and User) are saved in JSON format for ease of use and future analysis. The files are named *'youtube_collection.json'*, *'twitter_collection.json'*, and *'user_collection.json'*, respectively.

4.3 Text Retrieval and Processing

In order to analyze the content and sentiments expressed in social media discussions related to YouTube videos, it is necessary to extract and preprocess the textual data from the Twitter platform.

1. Text Extraction from Twitter: To associate each YouTube video with its corresponding tweets, a function is developed to access the tweet IDs stored in the *'tweet_ids'* column of the *'merged_collection'*. This function retrieves the text and hashtags from each tweet related to a specific video, enabling further analysis and processing.

2. Creation of *'merged_youtube_collection_text'*: The extracted text and hashtags are saved in a new collection, *'merged_youtube_collection_text'*. This collection serves as an intermediate step in the data preprocessing pipeline and contains the essential textual information associated with each YouTube video.

3. Preprocessing Text Data: To enhance the quality of the extracted text, a preprocessing function is implemented. This function performs several tasks on the text, including tokenization, punctuation removal, removal of English stop words, and word lemmatization. These steps ensure that the text is prepared for subsequent analysis and modeling.

4. Creation of *'merged_youtube_collection_preprocessed.txt'*:
   The *'merged_ youtube_collection_text'* is read, and the preprocessing function is applied to the column containing the extracted text. The results are stored in a new column, *'preprocessed_text'*, which captures the preprocessed text data for each YouTube video. The updated collection is then saved as *'merged_youtube_collection_ preprocessed.txt'* along with its corresponding CSV file.

4.4 Sentiment analysis on tweets

We conducted sentiment analysis on the text of each tweet. We employed five pretrained models available in the TweetNLP [2]for our sentiment analysis. These models had already undergone extensive training to capture different sentiment categories, including offensive, hate, irony, neutral, and negative sentiments. By utilizing pretrained models, we aimed to leverage the expertise and knowledge embedded within these models to analyze the sentiment expressed in the tweet dataset.

1. Generating distinct txt files: Using the TweetNLP library, we preprocess the text data in the *'merged_youtube_collection_preprocessed.txt'* collection. For each element in the collection, we apply a specific model to extract sentiment-related information from the *preprocessed_text* field. As a result, we generate five different txt files, each containing the output of a specific model.

2. Adding columns for each text file: To facilitate the integration of preprocessed text features, we add a column for each of the previously generated txt files to the dataframe. These columns correspond to different

[2]Doc TweetNLP `https://tweetnlp.org/resources/`

sentiment-related aspects, namely [*"hate," "irony," "offensive," "negative," "neutral"*].

5 Graph modeling and Pipeline

Graph modeling is a fundamental aspect of Graph Neural Networks (GNNs), which are designed to process and analyze data represented as graphs. In this context, the term "graph" refers to a collection of interconnected nodes or vertices, where relationships between nodes are represented by edges. Graph modeling involves constructing an effective representation of the underlying graph structure to capture relevant information and facilitate learning. By leveraging this representation, GNNs can perform powerful computations and make predictions based on the graph's topology, node attributes, and edge connections. The choice of graph modeling techniques plays a crucial role in the success of GNNs, as it determines how effectively the network can understand and reason about complex relational data.

5.1 Graph

In our analysis, we observed a significant relationship between YouTube (YT) nodes and the corresponding tweet nodes discussing them. Each YT node was associated with a collection of tweet nodes that referred to or discussed the specific YouTube video. These tweet nodes provided valuable insights into the conversations, opinions, and reactions surrounding the corresponding YouTube content.

The tweets associated with YT nodes exhibited different structures and types, reflecting the diverse ways users engage with and respond to YouTube videos. We identified three main types of tweets: original tweets, quote tweets, and retweets.

Original tweets were standalone posts created by users directly expressing their thoughts, opinions, or reactions to the YouTube video. Quote tweets, on the other hand, were tweets that quoted and added commentary to an existing tweet referencing the YouTube video. This allowed users to share and amplify specific tweets while adding their own perspectives. Finally, retweets were tweets that re-shared or reposted an existing tweet about the YouTube video, often indicating agreement, endorsement, or the desire to disseminate the content to a wider audience.

To establish the connections between tweet nodes and their respective users, we created edges linking each tweet node to the user who posted it. These edges represented the relationship between the tweet and its creator, allowing us to trace the origin of the tweet and analyze the user's influence and engagement within the network.

By incorporating these edges, we were able to capture the interplay between YouTube videos and the associated tweets, providing a comprehensive understanding of the discussions, sentiments, and user interactions surrounding the YT nodes. The resulting network provided valuable insights into the dynamics of information dissemination, user engagement patterns, and the overall impact of YouTube content on social media platforms.

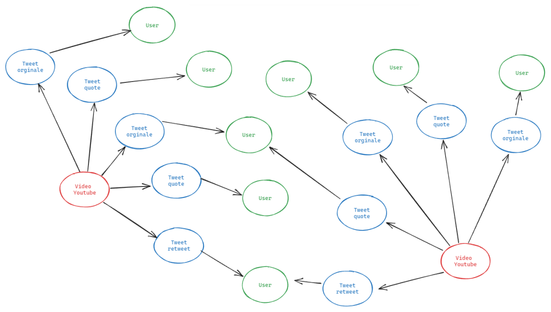


Figure 5: Graph depicting the relationships among YT videos, tweets and users

5.2 Other possible structures of graph

Another innovative solution we pursued involves creating a more intricate graph representation that captures the intricate relationships between original tweets and other types of associated tweets.

In this approach, we construct a graph where each original tweet acts as a central node, serving as the focal point for various types of tweets connected to it. These connected tweets encompass a range of tweet types, including quote tweets, retweets, and other forms of tweets directly referencing or deriving from the original tweet.

By incorporating this heightened level of complexity, we gain a deeper understanding of the intricate web of interactions surrounding an original tweet. This refined graph allows for comprehensive analysis of the dissemination and propagation patterns of the original tweet across diverse tweet types. It enables us to assess the engagement and reactions evoked by the original tweet, while also exploring its substantial influence on subsequent discussions and user interactions.

This more elaborate graph representation enables us to unravel the intricate interconnections and interdependencies among tweets, offering a more holistic understanding of the profound influence and significance of original tweets within the broader social media landscape.

The proposed alternative approach introduces a higher level of complexity by considering the intricate relationships that exist between original tweets and their associated counterparts.

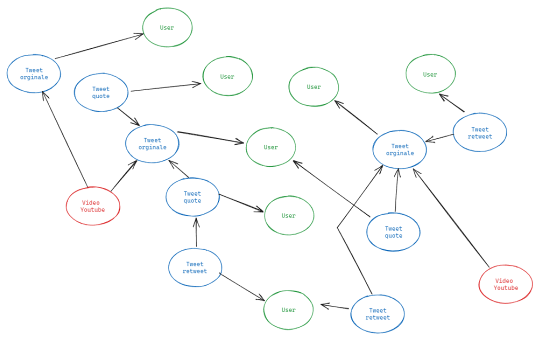


Figure 6: Graph depicting also the relationships among tweets

5.3 Pipeline

In this section, we present the data processing and analysis pipeline employed in our study. Figure 7 provides an overview of the pipeline architecture, showcasing the various steps involved in the analysis.

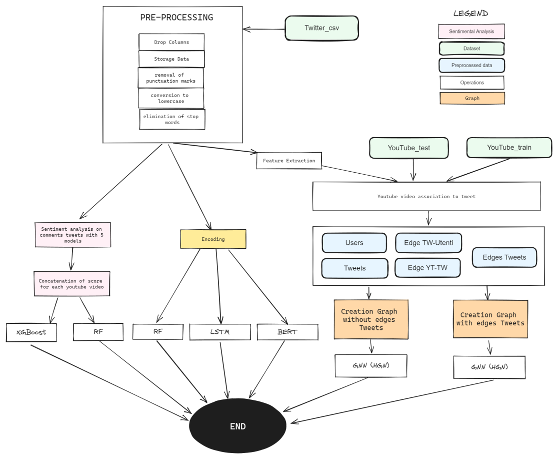


Figure 7: Pipe

The pipeline begins with the pre-processing of the raw data. This involves several operations such as text normalization, punctuation removal, and stop word elimination. After pre-processing, the data is ready for sentiment analysis.

We employ five different sentiment analysis models to capture the sentiment expressed in each tweet. The sentiment analysis results are then concatenated for each YouTube video, providing a comprehensive sentiment profile. Additionally, we utilize a Long Short-Term Memory (LSTM) network to capture temporal dependencies and patterns within the sentiment data.

Simultaneously, another branch of the pipeline focuses on creating the necessary nodes and edges for YouTube videos, tweets, and users. Edges are established between the YouTube video nodes and the associated tweet and user nodes. This network representation enables us to explore the relationships between YouTube videos and the corresponding tweets and users.

Furthermore, we leverage Graph Neural Networks (GNNs) for the detection of moderate and non-moderate YouTube videos. GNNs have proven to be effective in capturing complex network structures and identifying patterns within them, making them suitable for our task.

By integrating sentiment analysis, LSTM modeling, and GNN-based video moderation, our pipeline offers a comprehensive and multi-faceted analysis of the YouTube tweet network, providing valuable insights into the sentiment and moderation aspects of the data.

6 Models

In this section we present different approaches to model our data and to predict the status of a YouTube video. In particular, the figure 7 provides an overview of all the pipeline architecture, including both the models and the various steps involved in the analysis.

As described in the previous section, preprocessing of the raw data is carried out. This involves several operations such as normalization of text, removal of punctuation, and elimination of stop words. After preprocessing, the data are ready for analysis.

6.1 XGBoost

To predict YouTube video labels we first use the previous computed ”hate” ,”irony”, ”offensive” ,”negative” and ”neutral” columns to train a XGBoost model. The results on the test set are shown in table 1.

Code:

```
X_train, X_test, y_train, y_test =
train_test_split(x, y, test_size=0.1, random_state=42,stratify=y)
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
params = {
    'max_depth': 3,'eta': 0.1,
    'objective': 'binary:logistic',
    'eval_metric': 'logloss', 'seed': 42
}
num_rounds = 100
model = xgb.train(params, dtrain, num_rounds)
y_pred = model.predict(dtest)
predictions = [round(value) for value in y_pred]
```

The results:

Metric	**Value**
Accuracy	80.21%
Macro F1-Score	0.4451
Recall	0.0
Precision	0.0
Confusion Matrix	$\begin{matrix} 1374 & 0 \\ 339 & 0 \end{matrix}$

Table 1: Performance Metrics for XGBoost Model

We conclude by also reporting the results for individual classes, as shown in table 2 and 3.

Metric	Value
F1-Score	1.0
Recall	1.0
Precision	1.0

Table 2: Performance Metrics not-moderated class

Metric	Value
F1-Score	0.0
Recall	0.0
Precision	0.0

Table 3: Performance Metrics moderated class

6.2 SVM - Support vector machine

In a similar way can employ the Support Vector Machine (SVM) algorithm. Results are shown in figure 4.

```
svm = SVC(kernel='sigmoid')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
```

Metric	Value
Accuracy	67.94%
Macro F1-Score	0.4883
Recall	0.1797
Precision	0.1718
Confusion Matrix	$\begin{matrix} 2210 & 564 \\ 534 & 117 \end{matrix}$

Table 4: Performance Metrics for SVM

We conclude by also reporting the results for individual classes, as shown in table 5 and 6.

Metric	Value
F1-Score	0.8011
Recall	0.7966
Precision	0.8056

Table 5: Performance Metrics not-moderated class

Metric	**Value**
F1-Score	0.1752
Recall	0.1792
Precision	0.1712

Table 6: Performance Metrics moderated class

6.3 RF - Random forest

In the table 7 are shown the results obtained on the same previous test using a random forest classifier.

```
model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Metric	**Value**
Accuracy	80.41%
Macro F1-Score	0.47347
Recall	0.03072
Precision	0.3333
Confusion Matrix	$\begin{matrix} 2734 & 40 \\ 631 & 20 \end{matrix}$

Table 7: Performance Metrics for RF

We conclude by also reporting the results for individual classes, as shown in table 8 and 9.

Metric	**Value**
F1-Score	0.8912
Recall	0.9855
Precision	0.8125

Table 8: Performance Metrics not-moderated class

Metric	**Value**
F1-Score	0.0562
Recall	0.0309
Precision	0.3333

Table 9: Performance Metrics moderated class

6.4 Improved RF - Random forest

We try to improve the previous performances adding to our mood columns also the preprocessed text. This requires an additional step to encode the text in some numerical features. In table 10 are shown the obtained results.

```
model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Metric	Value
Accuracy	82.22%
Macro F1-Score	0.5233
Recall	0.07987
Precision	0.8387
Confusion Matrix	$\begin{matrix} 2764 & 10 \\ 599 & 52 \end{matrix}$

Table 10: Performance Metrics for RF

We conclude by also reporting the results for individual classes, as shown in table 11 and 12.

Metric	Value
F1-Score	0.9001
Recall	0.9964
Precision	0.8210

Table 11: Performance Metrics not-moderated class

Metric	Value
F1-Score	0.1442
Recall	0.0795
Precision	0.8387

Table 12: Performance Metrics moderated class

6.5 LSTM - Long short term memory

```
import tensorflow as tf

latent_dim = 32

with tf.device('/GPU:0'):
    model = tf.keras.Sequential([
        encoder,
        tf.keras.layers.Embedding(
            input_dim=len_voc,
            output_dim=latent_dim,
            mask_zero=True),
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(latent_dim,
                                      return_sequences=True)),
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(latent_dim)),
        tf.keras.layers.Dense(latent_dim, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(1, activation="sigmoid")
    ])

tf.keras.backend.clear_session()
epochs = 8
batch_size = 16
model.compile(optimizer='adam'
loss=tf.keras.losses.BinaryCrossentropy()
metrics=["accuracy",tf.keras.metrics.Recall()
            ,tf.keras.metrics.Precision()])
model.fit(x=X_train, y=Y_train, epochs=epochs,
           validation_data = (X_test,Y_test),batch_size = batch_size)
```

Metric	**Value**
Accuracy	78.01%
Macro F1-Score	0.64317
Recall	0.4056
Precision	0.44
Confusion Matrix	2397 350 403 275

Table 13: Performance Metrics for LSTM

We conclude by also reporting the results for individual classes, as shown in table 14 and 15.

Metric	**Value**
F1-Score	0.8639
Recall	0.8723
Precision	0.8557

Table 14: Performance Metrics not-moderated class

Metric	**Value**
F1-Score	0.4219
Recall	0.4058
Precision	0.4390

Table 15: Performance Metrics moderated class

6.6 BERT

BERT[3][4] (Bidirectional Encoder Representations from Transformers) is a revolutionary model in natural language processing (NLP). It utilizes a Transformer architecture and is pre-trained on massive amounts of data to learn rich representations of language.

Metric	**Value**
Accuracy	74.34%
Macro F1-Score	0.65324
Recall	0.58997
Precision	0.39960
Confusion Matrix	$\begin{matrix} 2146 & 601 \\ 278 & 400 \end{matrix}$

Table 16: Performance Metrics for BERT

We conclude by also reporting the results for individual classes, as shown in table 17 and 18.

Metric	**Value**
F1-Score	0.8301
Recall	0.7811
Precision	0.8857

Table 17: Performance Metrics not-moderated class

Metric	**Value**
F1-Score	0.4791
Recall	0.5893
Precision	0.3996

Table 18: Performance Metrics moderated class

[3]Link paper `https://arxiv.org/abs/1810.04805`

[4]Documentation : `https://huggingface.co/transformers/v4.8.2/model_doc/bert.html?highlight=berttokenizer`

6.7 Heterogeneous Graph Learning

The purpose is to highlight the relationship between YouTube nodes and the corresponding nodes of tweets about it, , as shown in figure 5. Three main types of tweets were identified: original tweets, quote tweets, and retweets, each of which has different purposes in engaging with YouTube videos.

To establish the connections between tweet nodes and their respective users, we created edges linking each tweet node to the user who posted it. These edges represented the relationship between the tweet and its creator, allowing us to trace the origin of the tweet and analyze the user's influence and engagement within the network.

To build our Graph Neural Network (GNN), it has been used the PyTorch Geometric library in particular Heterogeneous Graphs [5][6]. PyTorch Geometric is a powerful library designed specifically to handle graph-structured data in deep learning tasks. It provides efficient and flexible tools for creating and training GNN models.

PyTorch Geometric offers several advantages to our project. First, it simplifies the process of constructing and manipulating graphs, allowing complex relationships between nodes and edges to be represented efficiently. This aspect is critical to our task, as we propose to model the connections between YouTube nodes and their corresponding tweets. (We did not model the arcs between tweets and between tweets and users.)

```
class HGT(torch.nn.Module):
    def __init__(self, data,hidden_channels, num_heads, num_layers):
        super().__init__()
        self.lin_dict = torch.nn.ModuleDict()
        for node_type in data.node_types:
            self.lin_dict[node_type] = Linear(-1, hidden_channels)

        self.convs = torch.nn.ModuleList()
        for _ in range(num_layers):
            conv = HGTConv(hidden_channels, hidden_channels,
                           data.metadata(),num_heads, group='sum')
            self.convs.append(conv)

        self.lin_label_prediction = nn.Linear(hidden_channels, 1)
        self.sigmoid = nn.Sigmoid()
        self.drop_out = nn.Dropout(p=0.5)
    def forward(self, x_dict, edge_index_dict):
        for node_type, x in x_dict.items():
            x_dict[node_type] = self.lin_dict[node_type](x).relu_()
        for conv in self.convs:
```

[5]Documentation Heterogeneous Graphs : https://pytorch-geometric.readthedocs.io/en/latest/tutorial/heterogeneous.html

[6]Paper : https://arxiv.org/abs/2003.01332

```
        x_dict = conv(x_dict, edge_index_dict)
    yt_x = x_dict['yt']
    yt_x = self.drop_out(yt_x)
    yt_logits = self.lin_label_prediction(yt_x)
    y_pred = self.sigmoid(yt_logits).squeeze()
    return y_pred
```

Metric	**Value**
Accuracy	79.33%
Macro F1-Score	0.67771
Recall	0.47844
Precision	0.49115
Confusion Matrix	$\begin{matrix} 2384 & 345 \\ 363 & 333 \end{matrix}$

Table 19: Performance Metrics for HGN

We conclude by also reporting the results for individual classes, as shown in table 20 and 21.

Metric	**Value**
F1-Score	0.8706
Recall	0.8739
Precision	0.8673

Table 20: Performance Metrics not-moderated class

Metric	**Value**
F1-Score	0.4851
Recall	0.4786
Precision	0.4919

Table 21: Performance Metrics moderated class

6.8 Graph with more edges for tweet

Another innovative solution, as shown in figure 6, we pursued involves creating a more intricate graph representation that captures the intricate relationships between original tweets and other types of associated tweets.

In this approach, we construct a graph where each original tweet acts as a central node, serving as the focal point for various types of tweets connected to it. These connected tweets encompass a range of tweet types, including quote tweets, retweets, and other forms of tweets directly referencing or deriving from the original tweet.

The proposed alternative approach introduces a higher level of complexity by considering the intricate relationships that exist between original tweets and their associated counterparts.

We use the same template as in the previous section, but add a new edges between tweets:

Metric	**Value**
Accuracy	78.48%
Macro F1-Score	0.64
Recall	0.4486
Precision	0.3805
Confusion Matrix	$\begin{matrix} 2430 & 420 \\ 317 & 258 \end{matrix}$

Table 22: Result HGC added tweets edge

We conclude by also reporting the results for individual classes, as shown in table 23 and 24.

Metric	**Value**
F1-Score	0.8685
Recall	0.8529
Precision	0.8847

Table 23: Performance Metrics not-moderated class

Metric	**Value**
F1-Score	0.4115
Recall	0.4486
Precision	0.3808

Table 24: Performance Metrics moderated class

7 Conclusion

The results obtained in the course of our experiments show that the nature of the task we are analyzing is quite challenging. Nevertheless we proved how the use of the graph structure, and consequently the use of graph neural networks brings an improvement to the performance of our model. Despite this, we can observe starting from the first model tested until the last one, how the imbalance in the dataset greatly affects the performance of the algorithm, making life difficult for it in predicting the rarest instances belonging to the class of moderated videos.

References

[1] La Gatta, V., Luceri, L., Fabbri, F., Ferrara, E. *The Interconnected Nature of Online Harm and Moderation: Investigating the Cross-Platform Spread of Harmful Content between YouTube and Twitter.*

[2] Cinelli, M., Quattrociocchi, W., Galeazzi, A., Valensise, C., Brugnoli, E., Schmidt, A., Zola, P., Zollo, F., Scala, A. *The covid-19 social media infodemic.* Scientific Reports, 10, 2020.

[3] Ginossar, T., Cruickshank, I. J., Zheleva, E., Sulskis, J., Berger-Wolf, T. *Cross-platform spread: vaccine-related content, sources, and conspiracy theories in YouTube videos shared in early Twitter COVID-19 conversations. Human Vaccines & Immunotherapeutics*, 18(1), 1–13, 2022.

[4] Mitts, T., Pisharody, N., Shapiro, J. *Removal of anti-vaccine content impacts social media discourse.* In 14th ACM Web Science Conference 2022, WebSci '22, pages 319–326, New York, NY, USA, 2022. Association for Computing Machinery.

[5] Russo, G., Verginer, L., Ribeiro, M. H., Casiraghi, G. *Spillover of antisocial behavior from fringe platforms: The unintended consequences of community banning*, 2022.

[6] Kipf, T. N., Welling, M. *Semi-Supervised Classification with Graph Convolutional Networks.* In *International Conference on Learning Representations (ICLR)*, 2017.

[7] Defferrard, M., Bresson, X., Vandergheynst, P. *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering.* In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[8] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y. *Graph Attention Networks.* In *International Conference on Learning Representations (ICLR)*, 2018.