IMT 547 Project Part I: Data Collection

Chesie Yu

02/17/2024

*This notebook outlines the **data collection** process for the **YouTube Gaming Comment Toxicity** project.*

Components

1. **Authentication & Configuration**: Library setup, logging configuration, and API client initialization.
2. **Utility Functions**: A series of functions designed to streamline the data collection workflow.
3. **Data Collection**: Channel- and keyword-based data collection producing a DataFrame containing video and comment data.

Functions

- `get_uploads_id(channel_id)` : Fetch the uploads playlist ID for a given YouTube channel.
- `get_video_ids(uploads_id, max_videos=30, keywords="")` : Fetch video IDs (default up to 30) based on given keywords from a upload playlist.
- `get_video_info(video_ids)` : Fetch video info from a list of YouTube videos.
- `get_video_comments(video_ids, max_comments=100)` : Fetch comment info (default up to 100) for a list of YouTube videos.
- `get_youtube_data(channel_ids, max_videos=30, max_comments=100, keywords="")` : Main function. Fetch videos and comments for a list of channels.

Data Collection Procedures

To support our examination of the impact of game genres on comment toxicity across YouTube gaming channels, we have devised the following data collection approach:

Step 1: Keyword Selection

*To **differentiate** action and non-action gaming videos on YouTube, we identified **two sets of keywords** representing popular games in each category.*

The keyword sets are as follows:

- **Action Games**: {"call of duty", "gta", "the last of us", "god of war", "batman", "red dead redemption", "assassin's creed", "star wars jedi", "resident evil", "cyberpunk", "fallout", "tomb raider", "elden ring"}

- **Non-Action Games**: {"minecraft", "pokemon go", "just dance", "it takes two", "uncharted", "brawl stars"}

Step 2: Channel Selection

*From SocialBook's Top 100 Gaming YouTubers, we curated a list of **33 channels** that predominantly create content in **English**. For each channel, we **manually assigned** the binary labels `english` and `gamer` in `gamer-100.csv`, ensuring our focus on **English-speaking gaming community**.*

Step 3: Data Collection

*Leveraging the **YouTube Data API**, we gathered data from **30 videos per category for each channel**, using pre-defined keywords for action and non-action games. We then collected the **100 most relevant top-level comments for each video**.*

The sets of features include:

- **Comment Features**: `[“video_id”, “comment_id”, “comment_author_id”, “comment_text”, “comment_time”, “comment_likecount”, "comment_replycount"]`
- **Video Features**: `[“channel_id”, “channel_name”, “video_id”, “video_title”, “video_creation_time”, “video_description”, “video_tags”, “video_viewcount”, “video_likecount”, "video_commentcount"]`

*The final dataset consists of **140,637 comments** encompassing **17 video and comment features**. Through analyzing this data, we aim to uncover insights into the dynamics of toxic commenting behaviors within the YouTube gaming communities. `02-preprocessing.ipynb` will focus on **data cleaning, text preprocessing, and feature labeling** for subsequent analysis.*

1. Authentication & Configuration

In [1]:
```
# The YouTube API key
API_KEY = "AIzaSyAZoK_8LGGGeTh21WBqDxa94zUztIPGwQM"
```

In [2]:
```
# Install libraries
!pip install --upgrade google-api-python-client --quiet
```

In [3]:
```
# Import libraries
import json
import logging
import time
import pandas as pd
import googleapiclient
from googleapiclient import discovery, errors
```

In [4]:
```
# Configure logging to file
logging.basicConfig(
    filename="../logs/data.log",
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    filemode="w"
)
```

In [5]:
```
# Initialize the YouTube API
youtube = googleapiclient.discovery.build("youtube", "v3", developerKey=API_KEY)
```

2. Utility Functions

In [6]:
```python
def get_uploads_id(channel_id):
    """
    Fetch the uploads playlist ID for a given YouTube channel.
    """
    # Call the API to find uploads channel id
    # Documentation: https://developers.google.com/youtube/v3/docs/channels/list
    request = youtube.channels().list(
        part="contentDetails",
        id=channel_id
    )
    res = request.execute()

    # Extract the uploads playlist id
    uploads_id = res["items"][0]["contentDetails"]["relatedPlaylists"]["uploads"]

    return uploads_id
```

In [7]:
```python
def get_video_ids(uploads_id, max_videos=30, keywords=""):
    """
    Fetch video IDs from a YouTube playlist.
    """
    # Empty list to store video_ids
    video_ids = []
    page_token = None

    # Loop until we collect enough videos
    while len(video_ids) < max_videos:
        # Call the API to extract video IDs from playlist
        # Documentation: https://developers.google.com/youtube/v3/docs/playlistItems
        request = youtube.playlistItems().list(
            part="snippet",
            playlistId=uploads_id,
            pageToken=page_token,
            maxResults=50
        )
        res = request.execute()

        # Store the video ids
        for v in res["items"]:
            # Check if title contains keywords
            # Maybe try stemming/lemmatization if I have the time?
            title = v["snippet"]["title"].lower()
            if any(k.lower() in title for k in keywords):
                video_ids.append(v["snippet"]["resourceId"]["videoId"])

            # Exit the loop once the required number of videos is reached
            if len(video_ids) >= max_videos:
                break

        # Set the token
        page_token = res.get("nextPageToken")

        # Exit the loop if no token is found
        if not page_token:
            break

    return video_ids
```

In [8]:
```python
def get_video_info(video_ids):
    """
    Fetch video info from a list of YouTube videos.
    """
    # Empty list to store video info
    video_info = []
    for vid in video_ids:
        # Call the API to extract video info from ids
        # Documentation: https://developers.google.com/youtube/v3/docs/videos#resource
        request = youtube.videos().list(
            part="snippet, statistics",
            id=vid
        )
        res = request.execute()

        for v in res["items"]:
            # Extract relevant video info
            video_info.append({
                "channel_id": v["snippet"]["channelId"],
                "channel_name": v["snippet"]["channelTitle"],
                "video_id": v["id"],
                "video_title": v["snippet"]["title"],
                "video_creation_time": v["snippet"]["publishedAt"],
                "video_description": v["snippet"]["description"],
                "video_tags": v["snippet"].get("tags", []),
                "video_viewcount": v["statistics"].get("viewCount", "0"),
                "video_likecount": v["statistics"].get("likeCount", "0"),
                "video_commentcount": v["statistics"].get("commentCount", "0"),
            })

    return video_info
```

In [9]:
```
def get_video_comments(video_ids, max_comments=100):
    """
    Fetch comments (up to max_comments) for a list of videos.
    """
    # Empty list to store the comments
    comment_info = []

    # Loop through the video ids
    for vid in video_ids:
        page_token = None

        # Empty list to store individual video comments
        video_comment_info = []

        while len(video_comment_info) < max_comments:
            try:
                # Call the API to extract comments for videos
                # Documentation: https://developers.google.com/youtube/v3/docs/commentThr
                request = youtube.commentThreads().list(
                    videoId=vid,
                    part="id, snippet, replies",
                    textFormat="plainText",
                    order="relevance",
                    maxResults=100,
                    pageToken=page_token
                )
                res = request.execute()

                # Extract relevant comment info
                for c in res["items"]:
                    video_comment_info.append({
                        "video_id": c["snippet"]["videoId"],
                        "comment_id": c["snippet"]["topLevelComment"]["id"],
                        "comment_author_id": c["snippet"]["topLevelComment"]["snippet"]["
                        "comment_text": c["snippet"]["topLevelComment"]["snippet"]["textO
                        "comment_time": c["snippet"]["topLevelComment"]["snippet"]["updat
                        "comment_likecount": c["snippet"]["topLevelComment"]["snippet"]["
                        "comment_replycount": c["snippet"]["totalReplyCount"]
                    })
                    # Exit the loop once the required number of comments is reached
                    if len(video_comment_info) >= max_comments:
                        break

                # Set the token
                page_token = res.get("nextPageToken")

                # Exit the loop if no token is found
                if not page_token:
                    break

            # Error handling for commentsDisabled
            except errors.HttpError as e:
                if e.resp.status == 403 and "commentsDisabled" in str(e):
                    logging.warning(f"Comments are disabled for video {vid}.")
                else:
                    logging.warning(f"An error occurred for video {vid}: {e}")
                break

        # Extend the comment info
        comment_info.extend(video_comment_info)

    return comment_info
```

Main Function

In [10]:
```
def get_youtube_data(channel_ids, max_videos=30, max_comments=100, keywords=""):
    """
    Fetch videos and comments for a list of channels
    """
    # Start timing
    start_time = time.time()

    all_video_info = []
    all_comment_info = []

    for channel_name, channel_id in channel_ids_dict.items():
        logging.info(f"Processing channel: {channel_name}")

        # Get uploads playlist id for channel
        uploads_id = get_uploads_id(channel_id)

        # Get video ids from uploads playlist
        video_ids = get_video_ids(uploads_id, max_videos, keywords)

        # Get video info from videos ids
        video_info = get_video_info(video_ids)
        all_video_info.extend(video_info)
        logging.info(f"Number of Videos Extracted: {len(video_info)}")

        # Fetch comments for each video
        comment_info = get_video_comments(video_ids, max_comments)
        all_comment_info.extend(comment_info)
        logging.info(f"Number of Comments Extracted: {len(comment_info)}\n")

    # Convert to DataFrames
    video_info_df = pd.DataFrame(all_video_info)
    comment_info_df = pd.DataFrame(all_comment_info)

    # Merge video information with comments
    yt_comments = pd.merge(video_info_df, comment_info_df, on="video_id", how="inner")

    # End timing
    print(f"Runtime: {time.time() - start_time:.4f} seconds")

    return yt_comments
```

3. Data Collection

In [11]:
```
# Set the parameters
max_videos = 30
max_comments = 100

# Select the keywords
action_keywords = [
    "call of duty", "gta", "the last of us", "god of war", "batman",
    "red dead redemption", "assassin's creed", "star wars jedi",
    "resident evil", "cyberpunk", "fallout", "tomb raider", "elden ring"
]

nonaction_keywords = [
    "minecraft", "pokemon go", "just dance", "it takes two", "uncharted",
    "brawl stars"
]
```

In [12]:
```
# Load the data
channels = pd.read_csv("../data/gamer-100.csv")
channels.head()
```

Out[12]:

	channel	channel_id	english	gamer	influence_score	followers	avg_views	post
0	PewDiePie	UC-lHJZR3Gqxm24_Vd_AJ5Yw	1.0	1	88	111.0m	7.6m	4.8
1	A4	UC2tsySbe9TNrl-xh2lximHA	0.0	1	61	51.3m	20.8m	86
2	JuegaGerman	UCYiGq8XF7YQD00x7wAd62Zg	0.0	1	82	49.3m	5.3m	2.1
3	Mikecrack	UCqJ5zFEED1hWs0KNQCQuYdQ	0.0	1	59	47.7m	9.3m	2.0
4	Fernanfloo	UCV4xOVpbcV8SdueDCOxLXtQ	0.0	1	82	46.9m	30.8m	54

In [13]:
```
# Filter the English-speaking gamer channels
filtered_channels = channels[(channels["english"] == 1) & (channels["gamer"] == 1)]

# Select the channels
channel_ids_dict = pd.Series(filtered_channels["channel_id"].values,
                             index=filtered_channels["channel"]).to_dict()
len(channel_ids_dict)
```

Out[13]:
```
33
```

Action Gaming Videos

In [14]:
```
# Get YouTube videos and comments for action videos
yt_action = get_youtube_data(channel_ids_dict, max_videos, max_comments, action_keywords)
yt_action["genre"] = "action"
yt_action.head(3)
```

```
Runtime: 647.1408 seconds
```

Out[14]:

	channel_id	channel_name	video_id	video_title	video_creation_time	video_description
0	UC-lHJZR3Gqxm24_Vd_AJ5Yw	PewDiePie	F-yEoHL7MYY	I ~~tried to~~ beat Elden Ring Without Dyi...	2022-04-30T16:40:18Z	🌏 Get exclusive NordVPN deal here ➵ https://N...
1	UC-lHJZR3Gqxm24_Vd_AJ5Yw	PewDiePie	F-yEoHL7MYY	I ~~tried to~~ beat Elden Ring Without Dyi...	2022-04-30T16:40:18Z	🌏 Get exclusive NordVPN deal here ➵ https://N...
2	UC-lHJZR3Gqxm24_Vd_AJ5Yw	PewDiePie	F-yEoHL7MYY	I ~~tried to~~ beat Elden Ring Without Dyi...	2022-04-30T16:40:18Z	🌏 Get exclusive NordVPN deal here ➵ https://N...

In [15]:
```
# Check the dimension
yt_action.shape
```

Out[15]:
```
(64195, 17)
```

In [16]:
```
# Write to CSV
yt_action.to_csv("../data/yt_action.csv", index=False)
```

Non-Action Gaming Videos

In [17]:
```
# Get YouTube videos and comments for non-action videos
yt_nonaction = get_youtube_data(channel_ids_dict, max_videos, max_comments, nonaction_key
yt_nonaction["genre"] = "non-action"
yt_nonaction.head(3)
```

```
Runtime: 812.4069 seconds
```

Out[17]:

	channel_id	channel_name	video_id	video_title	video_creation_time	video_descripti
0	UC-lHJZR3Gqxm24_Vd_AJ5Yw	PewDiePie	KeeeLsAa30M	$39,000,000 Minecraft House..	2023-01-17T17:45:00Z	#AD - Pre-Orde FUEL’s New PA MAN Flavor!
1	UC-lHJZR3Gqxm24_Vd_AJ5Yw	PewDiePie	KeeeLsAa30M	$39,000,000 Minecraft House..	2023-01-17T17:45:00Z	#AD - Pre-Orde FUEL’s New PA MAN Flavor!
2	UC-lHJZR3Gqxm24_Vd_AJ5Yw	PewDiePie	KeeeLsAa30M	$39,000,000 Minecraft House..	2023-01-17T17:45:00Z	#AD - Pre-Orde FUEL’s New PA MAN Flavor!

In [18]:
```
# Check the dimension
yt_nonaction.shape
```

Out[18]:
```
(76437, 17)
```

In [19]:
```
# Write to CSV
yt_nonaction.to_csv("../data/yt_nonaction.csv", index=False, escapechar="\\")
```

Complete Dataset

In [20]:
```python
# Combine into one DataFrame
yt = pd.concat([yt_action, yt_nonaction], ignore_index=True)
yt.head()
```

Out[20]:

	channel_id	channel_name	video_id	video_title	video_creation_time	video_description
0	UC-lHJZR3Gqxm24_Vd_AJ5Yw	PewDiePie	F-yEoHL7MYY	I ~~tried to~~ beat Elden Ring Without Dyi...	2022-04-30T16:40:18Z	🌏 Get exclusive NordVPN deal here ➼ https://N...
1	UC-lHJZR3Gqxm24_Vd_AJ5Yw	PewDiePie	F-yEoHL7MYY	I ~~tried to~~ beat Elden Ring Without Dyi...	2022-04-30T16:40:18Z	🌏 Get exclusive NordVPN deal here ➼ https://N...
2	UC-lHJZR3Gqxm24_Vd_AJ5Yw	PewDiePie	F-yEoHL7MYY	I ~~tried to~~ beat Elden Ring Without Dyi...	2022-04-30T16:40:18Z	🌏 Get exclusive NordVPN deal here ➼ https://N...
3	UC-lHJZR3Gqxm24_Vd_AJ5Yw	PewDiePie	F-yEoHL7MYY	I ~~tried to~~ beat Elden Ring Without Dyi...	2022-04-30T16:40:18Z	🌏 Get exclusive NordVPN deal here ➼ https://N...
4	UC-lHJZR3Gqxm24_Vd_AJ5Yw	PewDiePie	F-yEoHL7MYY	I ~~tried to~~ beat Elden Ring Without Dyi...	2022-04-30T16:40:18Z	🌏 Get exclusive NordVPN deal here ➼ https://N...

In [21]:
```python
# Check the dimension
yt.shape
```

Out[21]:
```
(140632, 17)
```

In [22]:
```python
# Write to CSV
yt.to_csv("../data/yt.csv", index=False, escapechar="\\")
```