CS4624
Multimedia, Hypertext, and Information Access

Final Report
May 9, 2024

Tweet Collection

Virginia Tech, Blacksburg VA 24061

Instructor: Dr. Edward A. Fox
Team: Sushen Kolakaleti, Kevin D'Alessandro, Enk Narantsatsralt, Ilya Mruz, Chris Lam
Clients: Andrea Kavanaugh, Mohamed Farag, Satvik Chekuri

Table of Contents

List of Figures

List of Tables

1.0 Abstract

For a series of various Virginia Tech research projects related to Dr. Andrea Kavanaugh, more than six billion tweets between the years 2009-2024 were collected to be used for research purposes. These tweets cover many topics, but primarily focus on trends and important events that occurred during the time period. These tweets were collected in three different formats: Social Feed Manager (SFM), yourTwapperKeeper (YTK), and Digital Methods Initiative Twitter Capture and Analysis Toolset (DMI-TCAT). The original focus of the project was to convert these tweets into a singular format (JSON) in order to make tweet access easier and simplify the research process.

The team in the Fall of 2021 consisting of Yash Bhargava, Daniel Burdisso, Pranav Dhakal, Anna Herms, and Kenneth Powell were the first to take on this project and managed to finish the process of writing the initial Python scripts used to convert the three tweet formats to JSON. They originally provided six different Python scripts, two for each of the three tweet formats, one for the individual schema and the other for the collection level schema. However, large parts of these Python scripts were highly unoptimized and would take an unreasonably long time to run. Thus, the team in Spring of 2022 consisting of Matt Gonley, Ryan Nicholas, Nicole Fitz, Griffin Knock, and Derek Bruce took on the project and managed to optimize a portion of the original Python scripts in addition to implementing a BERT-based machine learning model used to classify the tweets. They adjusted the scripts to better accommodate scale and were able to begin the tweet conversion process, getting through about 800 million of the roughly 6 billion tweets collected.

We took over this project in Spring of 2024, and began by writing additional automation scripts in order to simplify the process and reduce the amount of work that had to be done manually for the SFM conversion process. In addition to writing new scripts, we updated some of the scripts done by the past team, to better suit our uses. We exported 45 collections from the SFM machine and were able to convert 9,744,468 tweets from SFM. Regarding DMI_TCAT and YTK, the raw SQL files needed to be transferred to a new database in order to convert the remaining tweets. This process is currently running for DMI and YTK at the Digital Library Research Laboratory, located in room 2030 at Torgerson Hall. Regarding the machine learning aspects of the project, we implemented a new hate speech classifier, due to the prevalence of hate speech on the internet. We ran a test with both a GloVe model and a BERT model with a Naive Bayes classifier, before ultimately settling on the GloVe model due to it being significantly faster while still providing enough accuracy to be useful.

2.0 Introduction

2.1 Problem Objective

The problem is that we have 3 different tweet collection formats: Social Feed Manager (SFM), Digital Methods Initiative Twitter Capture and Analysis Toolset (DMI-TCAT), and YourTwapperKeeper (YTK). Together, they account for over 6 billion tweets. This tweet data was collected by the Digital Library Research Laboratory (DLRL). Essentially, we needed to convert all of these tweets to a singular JSON format, and organize them into collections so they are grouped by the keywords used to collect them. The motivation behind this project is to be able to give social researchers ease of access when trying to find social media trends based on tweets.

2.2 Deliverables

The previous teams left us in a good spot to manage this project and continue upon the foundation that they built for us [3][8]. The goal is to allow easy access to Virginia Tech faculty and students by taking this variously organized data and converting it into a structured, standardized JSON object format. We set our deliverables as the following:

1. Process all tweets from SFM, YTK, and DMI-TCAT into JSON-formatted objects
2. Process all tweet metadata from SFM, YTK, and DMI-TCAT into JSON-formatted collections
3. (Optional) Run machine learning models on event classification and hate speech classification on a sample set of tweet data

2.3 Clients

Our clients are Dr. Andrea Kavanaugh, Dr. Mohamed Farag, and Satvik Chekuri. Dr. Kavanaugh is the primary client, as she is a researcher with the DLRL, and has guided us throughout the semester on how to approach the challenges of this project.

2.4 Team

Our team is composed of Sushen Kolakaleti, Kevin D'Alessandro, Enk Narantsatsralt, Ilya Mruz, and Chris Lam.

3.0 Requirements

Finish processing the tweets

The previous team who worked on the project was able to finish processing about 800 million of the tweets collected [3]. With the help of the script optimizations done by the Spring 2022 team, the conversion would take roughly two weeks to finish processing the remaining roughly five billion tweets.

Validate data and run analytics

After we finish processing the SFM, YTK, and DMI-TCAT tweets, we planned to run tests in order to validate the data and ensure that everything was processed correctly. In addition to validating the data, we are planning to run analytics and statistics on the finished JSON file. We planned to have this done at the end of April.

Implement machine learning and finish documentation

We have implemented a hate speech detection machine learning model on a smaller scale. Due to the prevalence of hate speech on social media sites, we believe it will provide helpful insights. We were deliberating either a GloVe model [5] or a BERT model [7] with a Naive Bayes classifier [2]. We tested this with a small sample of tweets; more information can be found in Section 4.2.

4.0 Design

The idea of the tweet conversion project is quite simple. The simplicity in the design is primarily because the previous teams created scripts that are efficient and effective. However, the process of wrangling such a huge number of data is what proved difficult. Figure 1 describes the overarching methodology for how we handled this volume of data.

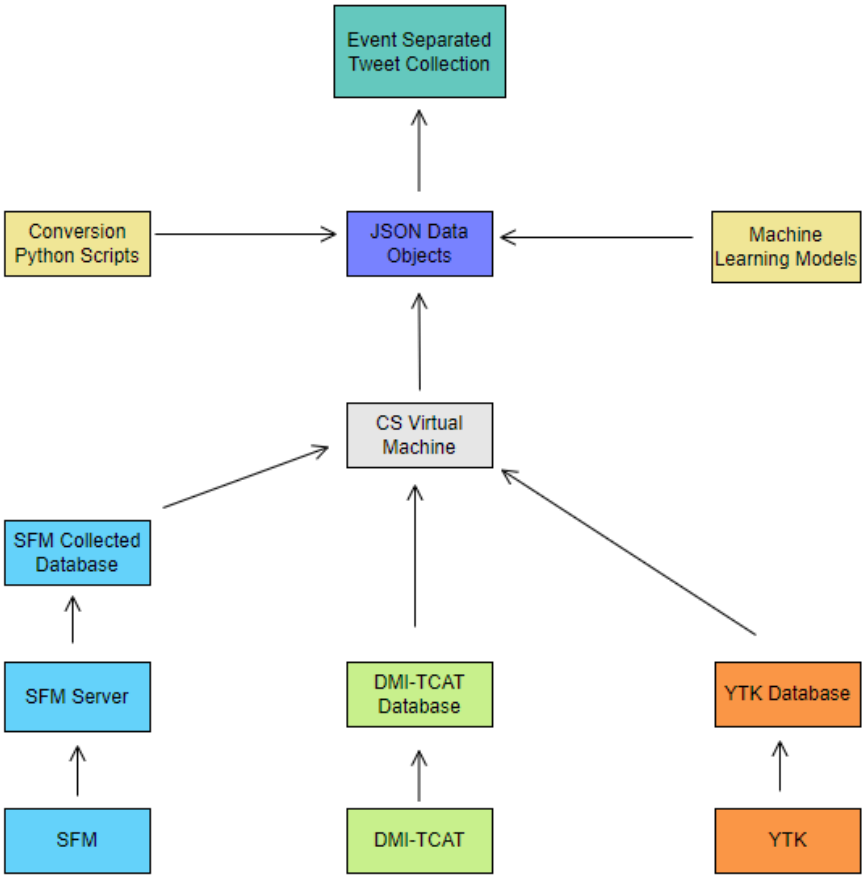

Figure 1: High-Level Data Processing Design [3]

This design consolidated the three different tweet sources into one processing environment. This way, we initiated all the conversion scripts at once, which allowed us to process the 3 different sources simultaneously, and speed up the collection process as a result.

4.1 Schema

The three archive types (SFM, DMI-TCAT, YTK) store different levels of information relating to a tweet. SFM stores the largest amount of data and informs the format of the unified schema. DMI-TCAT and YTK store a much more limited set of data. See Table 1 for an overview of what fields each format stores.

SFM schema	DMI-TCAT schema	YTK schema	Final schema
contributors			contributors
created_at	created_at	created_at	created_at
entities.hashtags			entities.hashtags
entities.media			entities.media
entities.media.url			entities.media.url
entities.media.id_str			entities.media.id
entities.urls			entities.urls
entities.user_mentions			entities.user_mentions
place.country			geo.country
geo.coordinates[0]	geo.lat	geo_coordinates_0	geo.latitude
geo.coordinates[1]	geo.lng	geo_coordinates_1	geo.longitude
geo.type		geo.type	geo.type
id_str	id (BigInteger)	Id	id
in_reply_to_screen_name			in_reply_to_screen_name
in_reply_to_user_id_str			in_reply_to_user_id
in_reply_to_status_id_str			in_reply_to_status_id
is_quote_status			is_quote_status
lang	lang		lang
metadata.iso_language_code		iso_language_code	lang_iso_code

metadata.result_type			result_type
favorite_count (is this for the tweet or user?)	favorite_count		metrics.favorite_count
retweet_count	retweet_count		metrics.retweet_count
source (HTML tag - will have to extract the exact source)	source	source	source
full_text	text	Text	text
user.contributors_enabled			user.contributors_enabled
user.created_at			user.created_at
user.default_profile			user.default_profile
user.default_profile_image			user.default_profile_image
user.description			user.description
user.follow_request_sent			user.follow_request_sent
user.geo_enabled			user.geo_enabled
user.id_str	from_user_id (BigInteger)	from_user_id	user.id
user.favorites_count			user.metrics.favorites_count
user.followers_count			user.metrics.followers_count
user.friends_count			user.metrics.friends_count
user.statuses_count			user.metrics.statuses_count
user.listed_count			user.metrics.listed_count

user.has_extended_profile			user.has_extended_profile
user.is_translation_enabled			user.is_translation_enabled
user.is_translator			user.is_translator
user.lang			user.lang
user.location			user.location
user.name	from_user_real_name		user.real_name
user.notifications			user.notifications
user.screen_name	from_user_name		user.screen_name
user.time_zone			user.time_zone
user.translator_type			user.translator_type
user.url			user.url
user.utc_offset			user.utc_offset
user.verified			user.verified
user.withheld_in_countries			user.withheld_in_countries

Table 1: Schema for Tweets [3]

In the process of converting, if any of the conversion scripts find a missing field, they will set the corresponding field of the unified JSON to NULL. This allows faster validation and manual inspection.

After analyzing the data schema for the tweets and comparing it to the previous group's conversion model, we concluded that the schema should remain the same, and we did not have to change the conversion method in the scripts.

4.2 Machine Learning

The machine learning component of the project was something that was an interesting piece of the project to compile from all the previous work done on it. The previous team had built a BERT classifier [7] that uses the BERT embedding technique to represent the tokenized tweets as sentences. This means that the BERT model is context-aware, which allows for the model to make meaningful connections between keywords in tweets. Our aim was to improve upon the performance of the BERT model built by the previous team. The previous team also used a Naive Bayes classifier [2] to classify tweets under certain events for collection purposes; however, one aim was to consolidate the current classifier and make it more streamlined and easier to use for the end client.

Furthermore, an interesting tool that we thought we could implement is a hate speech classifier. The prevalence of hate speech and hateful spam on the Internet is rampant, and we felt that adding an extra layer of filtration could help us to get a better result of tweet collections that are more original, useful, and insightful than some of the spam on the internet. For this implementation, we decided to use the GloVe model [5], which uses a set of global keywords to vectorize the tweets into a sentence. The reason for using GloVe here instead of BERT is simply the performance benefit. GloVe is much faster than BERT, while producing enough accuracy for the types of tweets that this classifier aims to filter out.

However, for the event classification problem, we decided to use a BERT model because of its better accuracy and ability to allow ease of access for the end user when sorting through tweet collections. A more detailed explanation of why we decided to make these design decisions is in the section under the Machine Learning Testing section. The overall processing plan for the machine learning model is shown in Figure 2.

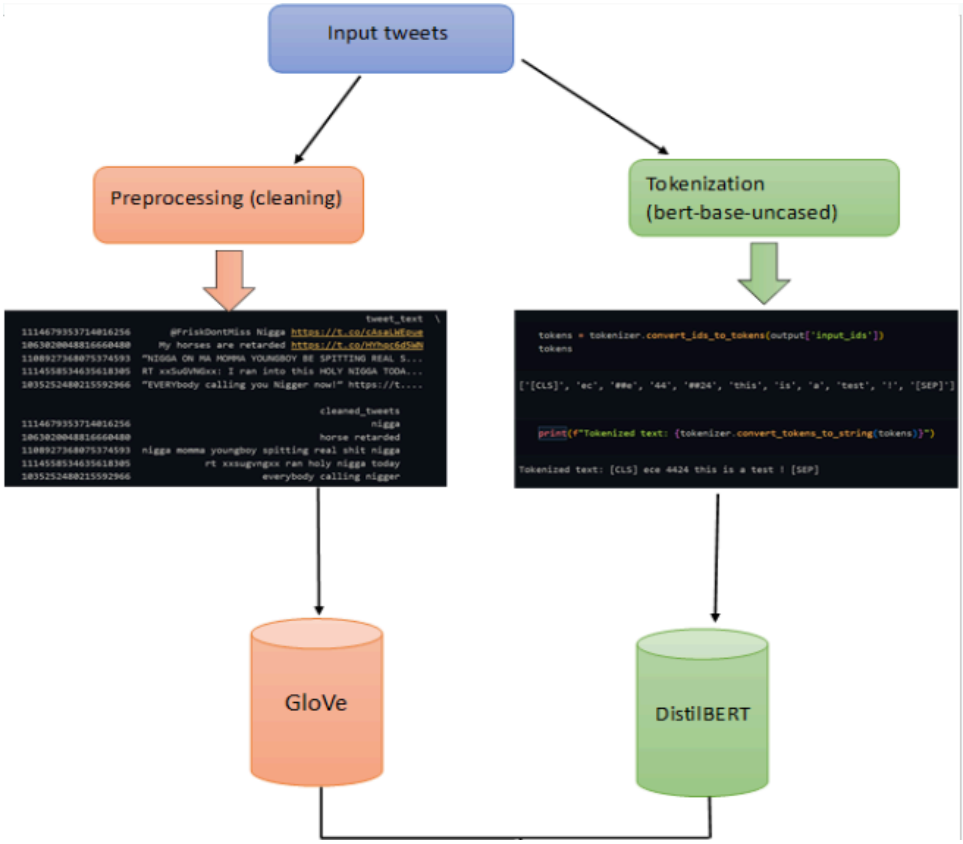

Figure 2: Glove and BERT Models [3]

Essentially, for the GloVe hate speech model, we take the input tweets from a sample database, and we go through some rudimentary cleaning of the JSON tweet object, which

includes filtering out for hashtags, keywords, and the tweet text itself. Then, we vectorize the tweet texts as sentences using the GloVe model, and then feed the output to the classifier for hate speech detection.

The BERT model is quite similar, but has some key differences. Instead of preprocessing the data like the GloVe data, we use the JSON object as a whole and tokenize the tweet text, hashtags, keywords, and more. This acts as a de facto preprocessing method because here we can choose to tokenize only the fields that we need, instead of sifting the whole JSON object like the GloVe method. Next, we feed it into the DistilBERT model [7], which is just a standard BERT model but uses a method called knowledge distillation, which shrinks the size of the model. The whole DistilBERT [7] method is shown in Figure 3.

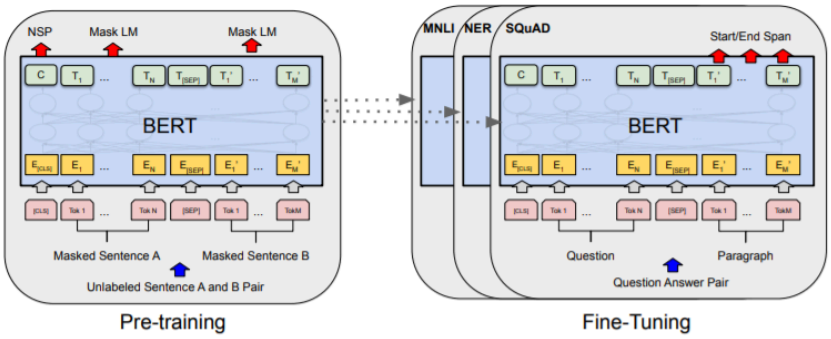

Figure 3: DistilBERT Model [2]

The output of the BERT model is then fed into a classifier, which classifies tweets as hate speech or not hate speech. Both the BERT and GloVe models use a Naive Bayes classifier [2], which is a fast and accurate method for general classification problems.

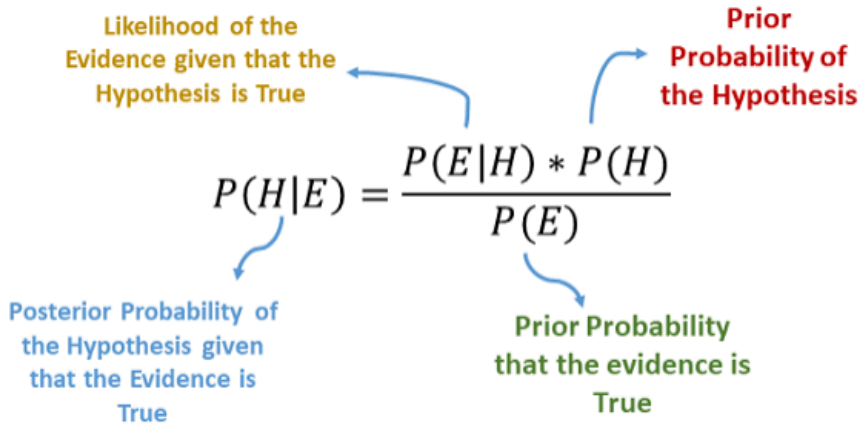

Figure 4: Naive Bayes Model [2]

The Naive Bayes classifier works by multiplying the probability of finding a feature or attribute given a class (P(x | C)) by the probability of just observing that class (P(C)). This value is then normalized by dividing the multiplied value by the probability of observing the given feature or attribute (P(x)). The resulting value is the probability of observing a class given a feature or attribute (P(C | x)). We can use this resulting probability to assign classes to features or attributes. This process is known as classification. Essentially, if the final probability is high, we can confidently assign a class to a feature input. The full details and formula of this machine learning method are in Figure 4.

5.0 Implementation

This project's implementation comprises seven primary components and a supplementary component. Among the primary components, three pertain to individual tweet conversions, while another three handle collection-level tweet conversions. The remaining primary component involves the deployment of a machine learning model aimed at categorizing uncategorized tweets. Table 2 delineates the inputs and outputs for each service, outlines the libraries, functions, and environments utilized, and assigns a unique identifier to each service. We also used a Virtual Machine (VM) to host all the data and conversion scripts for this project on Visual Studio Code (VSC) with the VM name of tweets.cs.vt.edu.

Service ID	Service Name	Input file name(s)	Output file name	Libraries; Environments
1	YTK Individual Conversion	YTK MySQL Data	YTK JSON Formatted Data	Libraries: JSON library, mysql.connector library, regex library Environment: VPN Linux Server
2	DMI Individual Conversion	DMI MySQL Data	DMI JSON Formatted Data	Libraries: JSON, mysql.connect Environment: VPN Linux Server
3	SFM Individual Conversion	SFM JSON data	SFM JSON Formatted Data	Libraries: JSON library Environment: VPN Linux Server
4	YTK Collection Conversion	YTK MySQL Data	YTK JSON Formatted Data	Libraries: JSON library, mysql.connect Environment: VPN Linux Environment
5	DMI Collection Conversion	DMI MySQL Data	DMI JSON Formatted Data	Libraries: JSON, mysql.connect Environment: VPN Linux Environment
6	SFM Collection Conversion	SFM JSON data	SFM JSON Formatted Data	Libraries: JSON library Environment: VPN Linux Server
7	Machine Learning Model	List of Tweet strings	Labels for the corresponding collection for the tweets	Libraries: scikit-learn, mysql.connector, torch, transformers Functions: Preprocessing, Model Prediction, Model Training, Model Analysis, SQL Select Data Environment: VPN Linux Server

Table 2: Service ID's and descriptions [3]

5.1 Events Archive Spreadsheet

The supplementary component is the Events Archive spreadsheet: a running list of all the collections that have been gathered so far. The spreadsheet has a comprehensive listing of the collections in YTK. However, the coverage for SFM and DMI is far from complete. Nevertheless, it serves two purposes: as a reference tool for researchers, and as a tool to help collection level conversions for sources that do not contain that information. More on that is discussed in Section 5.4. This list is hosted on the Collection_Table_for_IA20180620_Labeled5.xlsx spreadsheet and is one of the files available to download. See Table 3 for a description of the columns and their respective purposes.

Column Name	**Role**
ID	Used for easy reference
Source	Where the collection originated
Collection Terms	The term used to gather the collection. It is in the form of a hashtag or a term entered into Twitter's search functionality.
Wikipedia	A link to the Wikipedia article associated with the collection
Description	A quick description of the collection as well as the reason for collecting
Tags	Keywords used in searching as well as describing the collection
Categories (1, 2, 3)	Used to put collections into groups
Event Name	The name of the event that the collection of tweets reflects
Starting Date	When the process of collecting began
Latest Count	The number of tweets currently in the collection

Table 3: Event Archive Spreadsheet Columns and Descriptions

5.2 SFM Individual Conversion Implementation

To transform the Social Feed Manager (SFM) data into a JSON format was the simplest of the other 3 tweet conversion types. Leveraging the past semester's team's work, the Python scripts were designed to take in a JSON file with all the raw SFM tweet data and convert it to a unified JSON format for easy readability.

We have improved the SFM individual script from the previous semester by using the "utf-8" encoding versus the unspecified encoding that the previous team was using. We

found that during the conversion, certain SFM JSON files would run slower or run into conversion errors because of the encoding listed as “unspecified” before. Therefore, changing the encoding to “utf-8”made the encoding clear for the conversion process, which was able to process the JSON files in the right way.

Additionally, we created an automation script process, which after running a Python script, runs the SFM Individual Conversions on all the JSON files within a directory. The script then outputs a converted JSON to an output directory, where the converted JSONs can be validated.

We used the SFM service at the URL sfm1.cs.vt.edu/UI, to be able to see and export all the tweet collections so they could be processed using the scripts from the last team. We converted and hosted all the SFM JSON files on another, bigger storage, on a machine called camelot.cs.vt.edu that has about 24TB of storage for preserving important datasets. Future teams can contact Dr. Fox or someone else in the course for access to either the SFM or Camelot machines. The example for an automation script is shown in Figure 5.

```
import os

def execute_python_file(file_path, input, output):
  try:
    os.system(f'python {file_path} {input} {output}')
  except FileNotFoundError:
    print(f"Error: The file '{file_path}' does not exist.")

def run():
  files = os.listdir()
  for file in files:
    if '.json' in file:
      execute_python_file('sfm_converter.py', file, 'out/' + file)

run()
```

Figure 5: Individual Automation Script Example

5.3 SFM Collection Conversion Implementation

Converting SFM data at the collection level is notably simpler compared to processing individual tweets. The collection conversion script utilizes the same input files as those passed into the individual converter. Similarly, it iterates through the JSON file line by line, tallying the tweet count, compiling lists of tweet IDs and unique hashtags, and aggregating individual metrics such as favorites and retweets for the tweet collection. These aggregated data points are stored in a dictionary, which is then converted into a JSON

object. The process mirrors the approach employed by the individual converter and outputs the results to standard output.

As is explained above, we have improved the SFM collection script from the previous semester by using the “utf-8” encoding versus the unspecified encoding that the previous team was using. Likewise, we created an automation script process, which runs the SFM Collection Conversions on all the JSON files within a directory. We decided to run the individual and collection scripts simultaneously, sequentially, so that the total conversion time is just as long as the longest script runtime. This would speed up the overall conversion process.

5.4 DMI-TCAT Individual Conversion Implementation

The script dmi_converter.py facilitates the conversion of DMI-TCAT data, which comprises tweet information distributed across seven distinct tables within a MySQL database. Each table possesses a unique identifier and tweet IDs. This data organization was necessitated by the initial collection process, which segmented the data into separate tables. The conversion script establishes a connection with the MySQL database, retrieves the pertinent data, and transforms it into the JSON schema.

In cases where a value specified in the schema cannot be located, it defaults to None, representing a null value. The resulting tweet objects are then written to a file named after the originating table. Two output files are generated: one presenting the data inline, while the other encapsulates the data as JSON objects within a JSON array.

For the conversion process, the previous team had started the process but had not properly documented what exactly was done and how to restart the process. Eventually, we identified that the previous team took a set of tables, and made a database called “db1_track.” However, this database only had a partial amount of the data present in the larger DMI-TCAT SQL file.

So, to solve this problem, we made a script that looks through the raw SQL file for the DMI-TCAT tweets and finds all the tables in the raw SQL file. We made a master database of the tables that have not been processed as yet. The example bash script for the table finding is given in Figure 6.

```
#!/bin/bash

EXISTING_DB="db1_track"
IGNORE_COMMAND=""

mysql -u username -p -e "SHOW TABLES FROM $EXISTING_DB" | while read table; do
  IGNORE_COMMAND+=" --ignore-table=$EXISTING_DB.$table"
done

mysql -u username -p 2024_dmi_new < DMI_TCAT.sql $IGNORE_COMMAND
```

Figure 6: Table Finding Script Example

5.5 DMI-TCAT Collection Conversion Implementation

The DMI-TCAT collection level script, dmi_collection_converter.py, shares a similar class structure design with the DMI-TCAT individual tweet converter. However, its implementation is more straightforward since it only parses tweet IDs and any accompanying hashtags. Given that it's tasked with converting only two sets of attributes, it accesses only two relevant tables, extracting a single column from each.

To ensure uniqueness, hashtags are stored as a set. It's important to note that this script is solely capable of identifying tweet IDs and the total count of tweets for each collection. This limitation arises because DMI-TCAT lacks inherent collection-level data storage. Details such as event type, description, and additional attributes are sourced from the Events Archive spreadsheet.

5.6 YTK Individual Conversion Implementation

The YTK converter is responsible for converting each collection of tweets from YTK data, which initially resides in MySQL format, into a JSON file. During this conversion process, every pertinent field is transferred to a new tweet object following the new schema. If any value specified in the new schema cannot be found in the original YTK tweet, it is assigned a null value in the new tweet object.

Given the limited number of fields available from YTK and the presence of some non-essential fields, many fields in the output are null. However, the script ensures that essential information is retained. Each YTK table is characterized as a whole by a collection term that describes the entire table. This term typically appears frequently in the tweets or in similar terms.

Regarding the prior semester's team, they made enhancements to the script since the previous team had not captured hashtags related to each tweet at the individual level.

They incorporated regular expressions (regex) to extract hashtags from each tweet, thus enriching the data captured during the conversion process.

Each hashtag that is found, and the associated frequency, is listed in a “z_#_hashtags.csv” file, where the pound sign is replaced with the table number from the YTK database. These files are used in the later collection conversion.

5.7 YTK Collection Conversion Implementation

The YTK collection-level conversion script demonstrates significantly improved speed compared to the individual-level conversion. This efficiency stems from its reduced complexity. Unlike the individual-level conversion, it only necessitates the inclusion of tweet IDs without the need to handle additional fields.

During the conversion process, each JSON file for a collection incorporates tweet IDs organized as a list. YTK data comprises tables segmented by collection terms, often associated with specific events cataloged in the Events Archive spreadsheet. The script assigns each table a label based on the event, leveraging the "Event Name" column in the spreadsheet. If a table lacks classification in the spreadsheet, a machine learning model is employed. It utilizes the three most common hashtags from the table to classify it as an event. This machine learning model is described more in Section 4.2.

The script ensures each table's information is converted to JSON format and appended to the relevant collection. Previous versions of the file had merged all collections into a single JSON file, but it was updated to segregate collections by events. Collections sharing the same event are aggregated into the same JSON file. Although multiple collections may belong to a single event, each collection retains its distinct JSON object.

For instance, collections on "T-rex" and "velociraptor" both fall under the "Paleontology" event, with each collection representing a unique event within the "Paleontology" category. Information from the Events Archive spreadsheet such as descriptions, collection terms, and start dates from this table are incorporated into the conversion process, alongside utilizing labeled events to tag collection files.

Additionally, hashtags listed in descending order of frequency from individual tables, obtained through the individual converter, are included in CSV files. The collection converter script integrates these hashtags for each collection, further enriching the dataset. Leveraging labeled information from the YTK tables in the spreadsheet file and archive database facilitates seamless mapping, as each table is associated with a corresponding archive number in the spreadsheet file and database. See Figure 7 for more details.

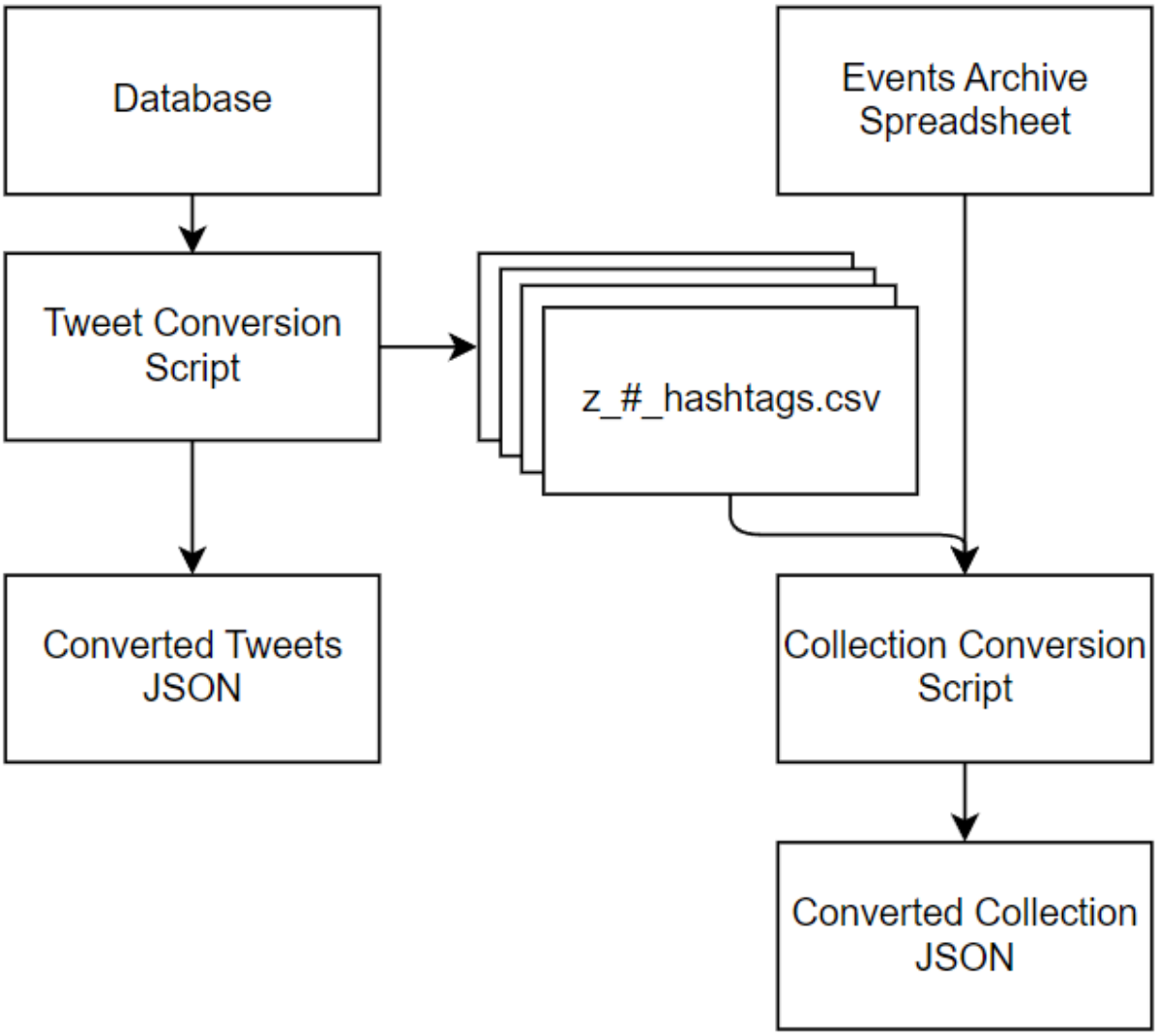

Figure 7: YTK Individual and Collection Conversion Data Flow

5.8 Conversion Runtime for each Tweet Server

Table 4 details the conversion statistics for each of the three tweet formats.

Tweet Format	Number of Tweets Converted	Time Taken (Seconds)	Tweets Per Second	Output File Size (GB)
DMI	21,646,879	9,780	2,213.3	43.13
YTK	791,601,941	157,920	5,012.7	992.98
SFM	200,171	19.24	10,403.9	0.42

Table 4: Previous runtime on the tweet conversions [3]

These are the results taken from previous conversions. We used them to make estimates on the whole dataset population for YTK, DMI-TCAT, and SFM. Our estimates are listed below.

SFM has approx. 4 billion remaining.
Individual Conversion:
10,403.9 tweets per second for 384,471 seconds =
106.8 hours = 4.5 days
Collection Conversion:
14317 tweets per second =
77 hours = 3.2 days

YTK has approx. 1.8 billion remaining.
Individual Conversion:
5,012.7 tweets per second for 359087.7 seconds =
99 hours = 4.1 days
Collection Conversion:
17,858 tweets per second for 101,975 seconds =
28 hours = 1.2 days

DMI-TCAT has approx. 1.2 billion remaining.
Individual Conversion:
2,213 tweets per second for 542,176.8 seconds =
150 hours = 6.28 days
Collection Conversion:
54,029 tweets per second for 22210 seconds =
6.169 hours = 0.257 days

To reduce the overall conversion time, we decided to run some of the collection and individual conversions in parallel. This ensures that, in theory, the total conversion time for all the tweet formats would only be as long as the longest conversion time.

In general, the collection level conversions are faster than their respective individual conversions. This is because the information required is much less, i.e., typically only the name of the collection and the associated tweets.

5.9 End of Semester Data Deliverable Statistics

For the end of the semester we decided to run statistics on all three of the tweet types (DMI-TCAT, YTK, and SFM). We also took those statistics and estimated what the end of the project could look like. Firstly, below is the current statistics and secondly is the estimated statistics.

5.9.1 Current Project Statistics

Overall Statistics:
- SFM Folder: 85,169,248 bytes
- DMI Folder: 45,658,580 bytes
- YTK Folder: 1,058,349,280 bytes

SFM Information:
- Overall SFM folder contains:
  - 3 files:
    - already_imported_json_collections.txt
    - count.py (for finding the total number of tweets)
    - counter.sh (to run the count.py file)
  - 4 subdirectories:
    - done: 66,011,372 bytes (36 files)
    - collections: 211,544 bytes (34 files)
    - individual: 18,946,264 bytes (35 files)
    - imported-jsons: 56 bytes (7 files, 3 subdirectories)

DMI Information:
- For DMI:
  - Info.md file describing the folder and the conversion scripts
  - 2 subdirectories:
    - collection: 429,080 bytes (20 files)
    - individual: 45,229,476 bytes (20 files)

YTK Information:
- For YTK:
  - 2 subdirectories:
    - individual: 1,041,217,300 bytes (425 files)
    - collection: 17,131,980 bytes (639 files)

5.9.2 End of Project Statistics

- Estimated end of summer for SFM folders: 3,577,108,416 in total size
  - Collections, individual, and done: 200–300 files each
  - Imported-jsons: Remains the same

- Estimated end of summer for DMI: 136,975,668 total size

- Estimated end of summer for YTK: 4,233,397,120 total size

5.10 UJSON Usage

To optimize the conversion scripts, the past team incorporated the UJSON (Ultra JSON) library [6], which is implemented in C. To benchmark its performance, they conducted tests by selecting a single tweet from the database. They measured the time it took to convert this tweet to JSON format and then write the resulting object to a JSON file, repeating the process one million times.

Using the standard JSON library, the conversion process took approximately 112.39 seconds. However, the UJSON library achieved the same conversion in just 13.99 seconds.

Although ORJSON is reputed to be faster than UJSON for conversion, their testing revealed limitations. ORJSON exhibited superior speed but was restricted to ASCII characters. Additionally, it lacked a method for directly writing to files, resulting in slower writing times compared to C-based solutions when using pure Python. Given that some of our tweets contain Unicode characters, we opted to utilize UJSON as a faster alternative to the standard JSON library.

6.0 Testing

6.1 Testing Framework

The framework that we used for testing and validation of the effectiveness of the SFM, YTK, and DMI-TCAT JSON data object conversion was Python Unit Testing. The unit tests accurately assess the performance of the tweet conversions. We have a further testing framework that we used for determining machine learning models in the following section. Figure 8 shows an example test case for testing the YTK conversion method, specifically regarding its ability to convert hashtags properly in the tweet formatting.

```
def test_hashtags():
    """
    Test gathering the hashtags for the YTK data
    """
    converter = ytk.ytk_conversion()

    test1 = "testing #california1 random words"
    test2 = "#test #testing"
    test3 = "random no hashtags"
    test4 = "#test start of sentence"
    test5 = "now for the end of the #sentence3"
    test_null = None

    out1 = converter.get_hashtags(test1)
    print(out1)
    out2 = converter.get_hashtags(test2)
    print(out2)
    out3 = converter.get_hashtags(test3)
    print(out3)
    out4 = converter.get_hashtags(test4)
    print(out4)
    out5 = converter.get_hashtags(test5)
    print(out5)
    out6 = converter.get_hashtags(test_null)
    print(out6)


def test_time_format():
    """
```

Figure 8: Example Test Cases [3]

6.2 Machine Learning Testing

Table 5 illustrates the performance to accuracy ratio on a sample set of tweets. The sample set of tweets for testing purposes was the reliable MMHS150K (multimodal hate speech 150 thousand) data set [1]. The reason for using another tweet dataset was to have a test bed setup for the machine learning aspect of the project before we were able to get actual tweet data from the JSON conversion processing.

The way that we tested the machine learning models was that we split the MMHS150k dataset into 2 parts: training and testing data. The training data is 90% of the 150k dataset, and the testing is the remaining 10% of the dataset. After examining the training dataset, we decided to use accuracy as our main metric to measure correctness of the models, because the dataset has balanced classes. Balanced classes means that the training dataset has an equal number of examples for hate speech vs. not hate speech. We then ran the same classifier on both models and produced the results in Table 5.

BERT	GloVe
Running Time: 4m 02s	Running Time: 58s
Accuracy: 83.12%	Accuracy: 70.01%
Ratio: 284.10	Ratio: 82.85

Table 5: Test Accuracy and Runtime

As you can see from Table 5, the BERT model is far more accurate at classifying hate speech, but the run time of the BERT model is far higher than the GloVe model. Both test models used a Naive Bayes Classifier, which is a probabilistic model for determining the most likely case of classification.

For this use case of hate speech classification, we prioritized performance over accuracy. To calculate this, we used a ratio formula. The formula is as follows:

$$R = \frac{RT_{sec}}{Acc} \cdot 100$$

Here *R* is the ratio, *RT_sec* is running time in seconds, and *Acc* is the accuracy of the model in percent. We multiply this by 100, to scale it so that the value is easier to compare. We think of a ratio as a cost value, where the higher the value is, the more cost it is to the end client. Upon analysis, it becomes apparent that the cost ratio strongly favors the GloVe model at 82.85, contrasting with the considerably higher ratio of 284.10 for the BERT approach. This notable difference indicates that employing the BERT model would incur

more than three times the expense compared to utilizing GloVe. Consequently, after careful consideration, we concluded that the GloVe approach aligns better with our budgetary constraints and resource allocation, prompting our decision to adopt it over the BERT model.

7.0 User's Manual

7.1 Environment

As planned by the former teams, this project is implemented using Python scripts intended for command line execution on a VPN-accessible Linux Server. A command line interface is required, which is usually available by default on Windows, Mac, and Linux systems. Another easy to use functional environment is available through Microsoft Visual Studio Code. It allows terminal use and running scripts via the remote development feature. The Visual Studio Code terminal allows SSH into the correct remote server.

Furthermore, Python3 must be installed on the system, as the scripts were written for this specific version. Additionally, the scripts utilize several packages; the packages can be either pre-installed or installed using the provided requirements.txt file:

pip install -r requirements.txt

(To see dependencies, refer to Section 8.1.)

Lastly, MySQL must also be installed and configured on the system with the relevant databases to run the scripts successfully.

7.1.1 Accessing the Environment

This section is used to clarify how to login and access the different virtual machines, MySQL services, and SFM UI for any future work for anyone continuing this project. To obtain the passwords for the following information, contact either Dr. Fox or Dr. Farag. Their contact information is found in Section 11.0.

1. Tweets.cs.vt.edu: you can ssh into the virtual machine using the command
    a. ssh xw0078@tweets.cs.vt.edu
    b. To obtain the password, contact one of the previously mentioned people.
    c. Make sure after you login, to cd ..
    d. cd into the cs4624Spring2022 folder, and there are all the folders, scripts, and original data that we worked on throughout the project

2. camelot.cs.vt.edu: you can use the same thing as above for Camelot
    a. ssh cs4624s24_tweet@camelot.cs.vt.edu
    b. To obtain the password, contact one of the previously mentioned people.
    c. Make sure to "open folder" and click ok. You may need to input the password again. All the SFM conversion scripts and data we worked on are in the "sfm processing" folder on the Camelot machine.

3. MySQL login for tweets.cs.vt.edu
    a. Make sure you are logged in on the tweets.cs.vt.edu VM as listed above.
    b. In any folder, but preferably in the "original data" folder where all the raw .sql files are, enter the command: MySQL -u root -p
    c. To obtain the password, contact one of the previously mentioned people.
    d. From there you are able to see other databases, access the raw SQL files for DMI-TCAT and YTK, create new databases, see table names, etc. for the project.
    e. Make sure to quit after you are finished to leave the MySQL terminal.

4. SFM UI for SFM collection sets and exports
    a. The purpose of this UI server is to export and see all the active / inactive collections for the SFM server.
    b. To login first go to URL sfm1.cs.vt.edu:8084/ui/
    c. Then login with either account:
        i. username: chrislam or ilyamruz
        ii. To obtain the passwords, contact one of the previously mentioned people.
    d. If you click on Export: you will be able to see all successful, in-progress, and running exports of collection sets.
    e. If you go to collection sets on the UI: you can see all the collections sets listed below in a list format
    f. If you click on a specific collection set, and it does not have *tweets: ##* listed on the collection then that specific collection set cannot be exported because it doesn't have any tweets to process.
        i. Example of good set (Figure 9):

Figure 9: SFM Good data set example

        ii. Example of a bad set (Figure 10) and that cannot be exported (see that there are no tweets: ## category in this collection set). The reason why these are "bad sets" is just because we are focusing on collections that have collected tweets and not those that have other information instead. These "bad sets" should be ignored for this

specific project because you cannot export a JSON with no tweets inside of it.

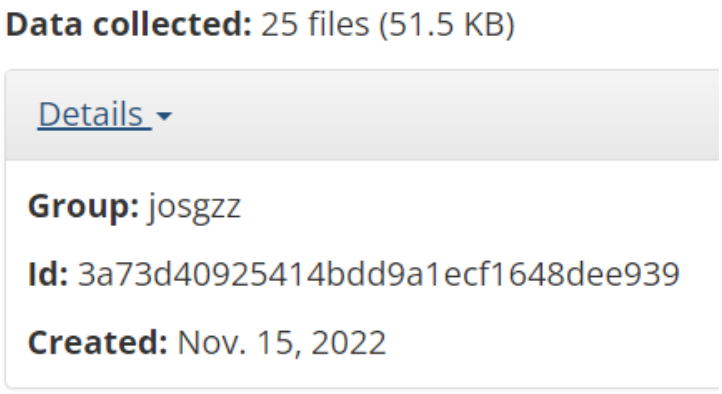

Figure 10: SFM Bad data set example

g. To export a collection set to a JSON format, follow these steps:
   i. select a collection set that has tweets inside
   ii. click on the collection name (Figure 11)

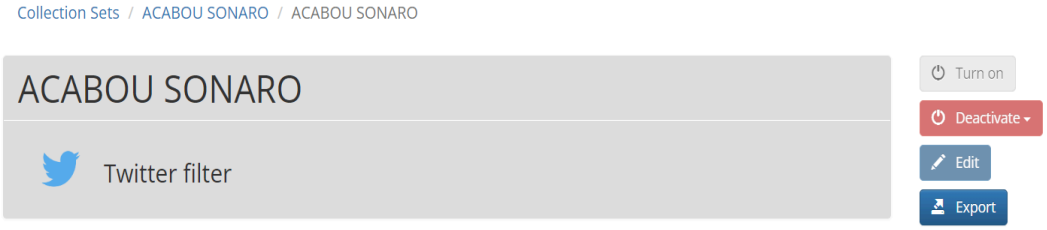

Figure 11: SFM Collection Set Example

   iii. Click the EXPORT button
   iv. Export Format should be : FULL JSON
   v. Maximum number of items per file should be: “single file” option. This is so that all the tweets in the collection set are stored into 1 JSON file and not multiple files when exported off the UI.
   vi. That’s it. Click the Export button and that collection should be added to the exported queue. You can see the queue once you click on the Exports button at the top of the screen.
   vii. Once the collection is marked as “completed success” you can click on the collection from the Exports page and download the # GB JSON file to then import to Camelot for processing. The README is just a temporary file; the one you’re looking for is the .json file for processing. This can be seen in the figure below from the SFM UI, where the first file listed is a README and the second is the .json containing the exported tweets.

Filename	Size
25b267ecc53540c5bb53535aa12b3d83-README.txt	1.5 KB
25b267ecc53540c5bb53535aa12b3d83_001.json	53.4 MB

Figure 12: SFM Collection Set Example Files

viii. You can then repeat these sets for all remaining collection sets that were not either imported to the Camelot or not yet exported from the server itself

7.2 Data Conversion Use Cases/Tasks Supported

The following section gives use cases and instructions about how to use the different JSON conversion methods.

7.2.1 SFM Use Case

The SFM conversion takes place via Python scripts which first take a JSON file as a parameter and then create an individual tweet data or collection level JSON object. In order to collect individual tweet data, an individual conversion script is used. In order to collect collection level conversion, a collection conversion script is used. To make it easier to convert SFM data, we have automated the script running process using a second series of Python and bash scripts.

The simple way to use this script is to upload all of your input JSON files into the base directory of where your conversion scripts are located. Then, run the “runner.sh” file to run both of the automation scripts concurrently. The easiest way to do this is to input “bash runner.sh” into your command line.

The output JSON files are stored in the “out” directory and the collections outputs are stored under the “collections” directory. The output files are under the names of the input JSON file, and the collection outputs are under the name “collection-#inputfile.json”. Thus, it should be easy to distinguish between the JSON files and formats, based on the names and directories.

To find the number of tweets converted in a conversion batch, please use the “counter.sh” script. You can run it by using the command “bash counter.sh”. This script looks through all the files in the “out” directory and gets the line count, to identify how many tweets have been converted.

Also included is a test script that uses Python unit tests to verify the SFM conversions. The simple way to test this is to run "python3 test_sfm.py". The output should show a series of unit tests, which state if the tests have passed or not passed.

The list of SFM scripts the user should utilize are:

1. runner.sh
2. counter.sh
3. test_sfm.py

7.2.2 YTK Use Case

The YTK conversion takes place via Python scripts, which utilize an SQL database to access that data. If the database has not been filled, one should load the ytk1-big_all_MySQL.sql script. In order to collect individual data, the first of the 2 files listed below is used. However, in order to collect collection level conversion data, the first file must be run first because it produces a CSV file with the required collection level hashtags.

YTK scripts used are:

1. ytk_converter.py
2. ytk_collection_converter.py

7.2.3 DMI-TCAT Use Case

The DMI-TCAT conversion takes place via Python scripts which, like YTK, use a SQL database to access the data. Also, similar to YTK, if the database has not been filled, the DMI_TCAT.sql script should be loaded. In order to collect individual tweet data, the first of the 2 files listed below is used. In order to collect collection level conversion, the second of the 2 files is used.

DMI_TCAT scripts used are:

1. dmi_converter.py
2. dmi_collection_converter.py

8.0 Developer's Manual

The SQL data is in the "originaldata" folder of the VT Tweets Virtual Machine for DMI-TCAT and YTK SQL files. The optimizations that the 2022 team made are in the "2022_optimization" folder. Tables 6-9 give the descriptions of the various file types. These came from the 2022 report and are an important reference.

Filename	**Description**	**Format**
sfm_sample_collection_covid_library.JSON	Sample file of SFM data.	Each line contains a JSON object following the SFM schema, representing a single tweet.
ytk2-eom_all_MySQL.sql	Sample file of YTK data. This has a subset of the ytk1-big_all_MySQL.sql data	The file takes the form of a series of SQL commands. These form a YTK database.
ytk1-big_all_MySQL.sql	SQL script that loads in all the YTK data.	The file takes the form of a series of SQL commands. These form a YTK database.
DMI_TCAT.sql	SQL script to import all the DMI data to MySQL	The file takes the form of a series of SQL commands, like a .sql file. These form a DMI-TCAT database.
Collection_Table_for_IA20180620_Labeled4.xlsx	File of known tweet collections found in the tweet data across all three sources. These have all been labeled for their event name.	Comma Separated Values

Table 6: Input Data Files

Filename	Command to use	Description
test_sfm.py	$ python test_sfm.py	Unit test file for the SFM tweet converter.
runner.sh	$ bash runner.sh	Takes all raw JSON files in a directory and converts them to unified JSON
counter.sh	$ bash counter.sh	Takes all JSON files in a directory and counts the total tweets

Table 7: SFM Script files

Filename	Command to use	Description
ytk_converter.py	$ Python3 ytk_converter.py	Python script that connects to a MySQL database and converts given data into the new schema. Asks for username, password, hostname, and database of tweets.
ytk_collection_converter.py	$ Python3 ytk_collection_converter	Python script that is used to collect all the tweet IDs in a collection. It outputs a single JSON object that follows the collection level schema to a file that compiles a list of all the tweets in a collection.

Table 8: YTK Script Files

Filename	Command to use	Description
dmi_converter.py	$ Python3 dmi_converter.py	Python script that connects to a MySQL database and converts given data into the new schema. Asks for username, password, hostname, and database of tweets.
dmi_collection_converter.py	$ Python3 dmi_collection_converter.py	Python script that is used to collect all the tweet IDs in a collection. It outputs a single JSON object that follows the collection level schema to a file that compiles a list of all the tweets in a collection.

Table 9: DMI-TCAT Script Files

8.1 Database Connection

For the tweet servers / converters that are required to use MySQL databases for processing tweets, the access to MySQL on the tweets.cs.vt.edu virtual machine is below:

username : root,

host: localhost

password : to obtain the password, contact Dr. Fox or Dr. Farag (see Section 11.0).

databases: db1-track and twitter

Example command to access MySQL: (The $ is for the command line example from VSC)

$MySQL –u root –p

$(insert password here)

How to see all databases from the MySQL command line (after logging in as a user):

SHOW DATABASES;

How to create a new database from command line and from scratch:

CREATE DATABASE new_database_name;

This command will show you all of the ‘create tables’ in the original .sql file for both DM and YTK

```
grep -oP "CREATE TABLE \`[^\`]+\`" DMI_TCAT.sql | sed 's/CREATE TABLE \`//;s/\`//' | sort | uniq
```

All the table names are in the ‘DMI all tables.txt’ and ‘YTK all tables.txt’ files respectively for DMI-TCAT and YTK tweets in the original data folder on the tweets.cs.vt.edu VM. For DMI-TCAT there is a more detailed description of the db1_track database in the info.md file in the DMI folder in the 2022_optimization folder of the VM.

8.2 Dependencies

The scripts used for the translations were written in Python 3. They depend on Python libraries that need to be installed using pip from the Python Package index [4]. Below are the dependencies required to run the scripts.

- argon2-cffi==21.3.0
- argon2-cffi-bindings==21.2.0
- async-generator==1.10
- attrs==21.4.0
- autopep8==1.6.0
- backcall==0.2.0
- bleach==4.1.0
- certifi==2021.10.8
- cffi==1.15.0
- charset-normalizer==2.0.9
- click==8.0.4
- cycler==0.11.0
- dataclasses==0.8
- decorator==5.1.0
- defusedxml==0.7.1
- entrypoints==0.4
- et-xmlfile==1.1.0
- filelock==3.4.1
- greenlet==1.1.2
- huggingface-hub==0.4.0
- idna==3.3
- importlib-metadata==4.8.3
- importlib-resources==5.4.0
- ipykernel==5.5.6
- ipython==7.16.2
- ipython-genutils==0.2.0

- jedi==0.17.2
- Jinja2==3.0.3
- joblib==1.1.0
- jsonschema==3.2.0
- jupyter-client==7.1.2
- jupyter-contrib-core==0.3.3
- jupyter-contrib-nbextensions==0.5.1
- jupyter-core==4.9.2
- jupyter-highlight-selected-word==0.2.0
- jupyter-latex-envs==1.4.6
- jupyter-nbextensions-configurator==0.4.1
- jupyterlab-pygments==0.1.2
- kiwisolver==1.3.1
- lxml==4.8.0
- MarkupSafe==2.0.1
- matplotlib==3.3.4
- mistune==0.8.4
- MySQL-connector-python==8.0.25
- nbclient==0.5.9
- nbconvert==6.0.7
- nbformat==5.1.3
- nest-asyncio==1.5.4
- notebook==6.4.10
- numpy==1.19.5
- openpyxl==3.0.9
- orjson==3.6.1
- packaging==21.3
- pandas==1.1.5
- pandocfilters==1.5.0
- parse==1.19.0
- parso==0.7.1
- pexpect==4.8.0
- pickleshare==0.7.5
- Pillow==8.4.0
- prometheus-client==0.13.1
- prompt-toolkit==3.0.23
- protobuf==3.17.3
- ptyprocess==0.7.0
- pycodestyle==2.8.0

- pycparser==2.21
- Pygments==2.10.0
- PyMySQL==1.0.2
- pyparsing==3.0.7
- pyrsistent==0.18.0
- python-dateutil==2.8.2
- pytz==2021.3
- PyYAML==6.0
- pyzmq==22.3.0
- regex==2022.3.15
- requests==2.26.0
- sacremoses==0.0.49
- scikit-learn==0.24.2
- scipy==1.5.4
- Send2Trash==1.8.0
- six==1.16.0
- SQLAlchemy==1.4.32
- terminado==0.12.1
- testpath==0.6.0
- threadpoolctl==3.1.0
- tokenizers==0.11.6
- toml==0.10.2
- torch==1.10.2
- tornado==6.1
- tqdm==4.64.0
- traitlets==4.3.3
- transformers==4.18.0
- typing_extensions==4.1.1
- ujson==4.3.0
- urllib3==1.26.7
- wcwidth==0.2.5
- webencodings==0.5.1
- xlrd==2.0.1
- zipp==3.6.0

8.3 Repository Structure

The 2022 team created six folders in the repository: SFM, YTK, DMI, ml_model, testing, and analysis. The SFM, YTK, and DMI folders contain the related individual and collection scripts. The ml_model folder contains the machine learning models that classify the tweets with the appropriate label. Testing contains the benchmarking scripts but also includes the unit tests to verify that each script produces sane results.

Future teams that set up their own environments should reuse the structure established by the 2022 team.

The directory for each type should have an individual and collection folder. Each folder should have their respective conversion script. This is especially important for YTK, because its individual script uses a relative path to create files useful for the collection folder. The YTK folder should also contain the Events Archive spreadsheet, as the scripts refer to the file using a relative path.

All of our files for this semester are located in the TweetScripts.zip file included with the submission. The file structure is broken down into very simple categories based on the aspect of the project: SFM, DMI-TCAT, YTK, and Machine Learning. There is also a requirements.txt file, which lists all the required Python packages in order to run the project scripts. We have also included all the dependencies in a list in Section 8.1.

9.0 File Inventory

9.1 SFM Scripts

Within the SFM folder, we have all of our SFM automation scripts, which were explained before in Section 7.2.1. The main conversion scripts are “sfm_converter.py” and “sfm_collection_converter.py.” The automation scripts are “sequential.py” and “count.py.” The running scripts are bash scripts under the names “runner.sh” and “counter.sh.” The test script is under the name “test_sfm.py.”

9.2 YTK Scripts

The YTK folder contains the collection and individual folder, as well “GetCounts.py,” a Python script that can be used to count the number of tweets in a specified database. The individual folder contains the “ytk_converter.py” and “test_ytk_converter.py” scripts, as well as the “json_size.py” script. These scripts allow individual conversions, testing, and finding JSON file sizes. The collection folder contains the “ytk_collection_converter.py” and “test_ytk_collection_converter.py” scripts, as well as the same “json_size.py” script. These scripts allow for the same functionality on collection YTK conversions.

9.2 DMI-TCAT Scripts

The DMI folder contains the collection and individual folder. There is also an “info.md” file, which contains DMI information about the SQL database that is currently set up. The individual folder contains the “dmi_converter.py” as well as the “json_size.py” script. These scripts allow individual conversions, finding JSON file sizes. The collection folder contains the “dmi_collection_converter.py” as well as the same “json_size.py” script. These scripts allow for the same functionality on collection DMI conversions.

9.3 Excel Files

The Events Archive spreadsheet, described in Section 5.1, is the most recent version of this document, but is still missing many of the collections from SFM and DMI. Some of the YTK information is also missing.

10.0 Lessons Learned

We had serious challenges throughout the semester, but the highlighted challenge was communication. Our initial communication with Satvik Chekuri (GTA), Dr. Fox, and Dr. Farag was segmented. However, we did identify the problem and especially towards the end, we made communication a priority with all the parties involved, which really helped move things along during the later half of the semester. Some of the other challenges and lessons learned are outlined below.

1. Poor Communication – start communication threads with involved parties much sooner
2. Task delegation – start delegated tasks and using task managers (i.e., Trello) to manage team members much sooner
3. Documentation – the previous team left very little documentation about important topics such as continuation for the next team and more. This left us scrambling to find all the segmented items, which took a lot longer than expected. So, one of the key tasks that we wanted to make sure to complete was to provide good documentation for any teams that need to use this project in the future. Mostly that includes clarifying how to access the SFM server UI, and access any MySQL database information (usernames and passwords).

Our initial timeline and plan is outlined below in Figure 13. Unfortunately, we were not able to stick to this timeline due to the aforementioned challenges. We only began processing tweets in April and fell behind schedule from those challenges.

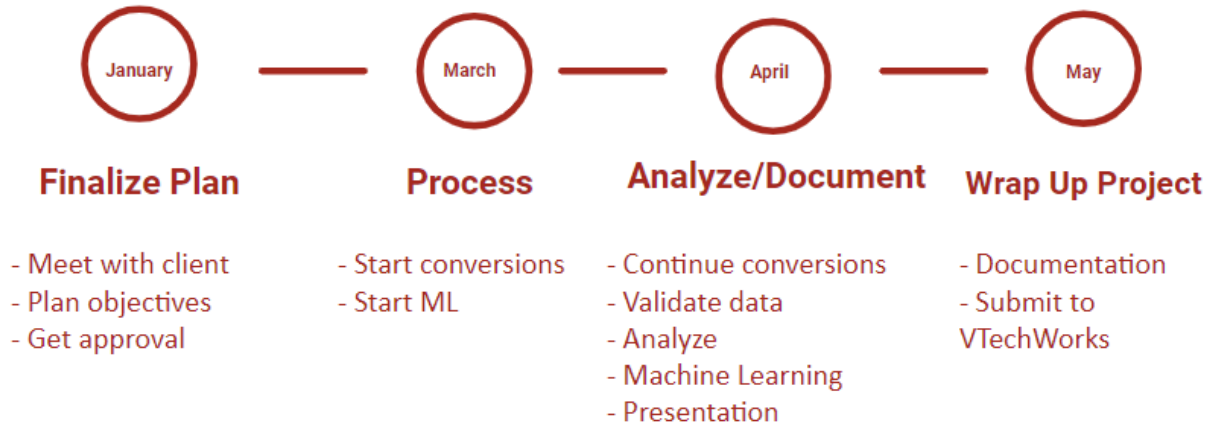

Figure 13: Original Timeline and Outline of Project

11.0 Future Plans

Where we are now in terms of the project is that the DMI-TCAT tweets are being converted to a database hosted on MySQL on the tweets.cs.vt.edu virtual machine. The YTK database is waiting to be run on tweets after the DMI-TCAT processing has been completed, since we don't want to overload MySQL with importing two databases at the same time. For the SFM portion of the project, we have queued most if not all the collection sets to be exported to a JSON format on the SFM server using the UI website. Lastly, we have built and tested a machine learning model to successfully interpret JSON tweets to detect if there is any hate speech written inside the tweet itself.

What was done over the course of the semester for this project was to successfully get access to the different Virginia Tech machines used by the past teams, figure out where the remaining tweets for DMI-TCAT and YTK were located, and start the importing process from a SQL file to a MySQL database. We also gained access and documented the exporting and converting process for SFM collection sets, and successfully converted around 9.7 million tweets from multiple different SFM collection sets.

What is yet to be done for this project is run the conversion scripts for the remaining tweets for DMI-TCAT and YTK because they are in the process of being imported into a MySQL database so that the scripts can connect to them to grab all the tables for the tweets to convert into unified JSONs.

For SFM all that is left to do is to wait on the exporting process to finish processing more collection sets. This is so that they can be imported into camelot.cs.vt.edu, a Virginia Tech server machine. Then, it will be possible to run the conversion scripts to process and unify the JSON files. Additionally, any other raw JSONs that are exported from the SFM UI that our team did not finish by the end of the semester, need to be imported into the Camelot machine and processed into the unified JSON format. For very big files, make sure to use suitable resources to allow for optimal download/upload time for importing those JSONs from the SFM User Interface into Camelot because these files can be upwards of 200 GB.

After all the files have been converted, they will be compressed into a drive and submitted to the Library and to our clients for research and storage. One team member will overlook the remaining conversions over Summer 2024 and submit when the conversions are done.

12.0 Acknowledgements

Dr. Andrea Kavanaugh
Associate Director, Center for Human/Computer Interaction
kavan@vt.edu

Dr. Mohamed Magdy Farag
Research Associate, VTTI-Sustainable Mobility
mmagdy@vt.edu

Dr. Edward Fox
Director, Digital Library Research Laboratory
fox@vt.edu

Satvik Chekuri
Ph.D. student and GTA
satvikchekuri@vt.edu

13.0 References

[1] V. C. Fuentes, “Multimodal Hate Speech.'' 10-Nov-2020. https://www.kaggle.com/datasets/victorcallejasf/multimodal-hate-speech/code. (Accessed February 28, 2024)

[2] B. Gamal, “Naïve Bayes algorithm,” *Analytics Vidhya*, 17-Dec-2020. [Online]. Available: https://medium.com/analytics-vidhya/na%C3%AFve-bayes-algorithm-5bf31e9032a2. (Accessed February 28, 2024)

[3] Gonley, Matt; Nicholas, Ryan; Fitz, Nicole; Knock, Griffin; Bruce, Derek; *Twitter Collections Report*, 05-10-2022. [Online]. Available: http://hdl.handle.net/10919/109988. (Accessed April 10, 2024)

[4] Gedam, Pradyun; Stufft, Donald; Chun, Tzu-ping; et al. pip documentation. (2020). https://pip.pypa.io/en/latest/. (Accessed March 9, 2024)

[5] J. Pennington, “GloVe: Global Vectors for word representation,” *Stanford.edu*, 09-10-2014. [Online]. Available: https://nlp.stanford.edu/projects/glove/. (Accessed April 22, 2024)

[6] J. Tarnstrom, “Ujson,” *PyPI*, 12-10-2023. [Online]. Available: https://pypi.org/project/ujson/. (Accessed April 22, 2024)

[7] D. Sharma, “Introduction to DistilBERT in student model,” *Analytics Vidhya*, 03-Nov-2022. [Online]. Available: https://www.analyticsvidhya.com/blog/2022/11/introduction-to-distilbert-in-student-model/. (Accessed April 22, 2024)

[8] Dhakal, P., Bhargava, Y., Herms, A., Powell, K., & Burdisso, D. (2021, December 16). *Library Tweets Conversion*. VTechWorks Repository. http://hdl.handle.net/10919/107086. (Accessed April 10, 2024)

[9] “Tokenizers,” *PyPI*, 04-17-2024. [Online]. Available: https://pypi.org/project/tokenizers/. (Accessed May 5, 2024)

[10] *Mysql.com*, 05-10-2020. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/. (Accessed May 5, 2024)

[11] “Social Feed Manager,” *Social Feed Manager*, 02-Nov-2021. [Online]. Available: https://gwu-libraries.github.io/sfm-ui/. (Accessed May 5, 2024)

[12] “User guide: contents — scikit-learn 0.21.3 documentation,” *Scikit-learn.org*, 06-15-2007. [Online]. Available: https://scikit-learn.org/0.21/user_guide.html. (Accessed May 5, 2024)

[13] Z. Hira, “Bash scripting tutorial – Linux shell script and command line for beginners,” *freecodecamp.org*, 20-Mar-2023. [Online]. Available: https://www.freecodecamp.org/news/bash-scripting-tutorial-linux-shell-script-and-command-line-for-beginners/. (Accessed May 5, 2024)

[14] G. Goyal, “Twitter Sentiment Analysis using Python,” *Analytics Vidhya*, 11-Jun-2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/twitter-sentiment-analysis-a-nlp-use-case-for-beginners/. (Accessed May 5, 2024)

[15] “Torch,” *PyPI*, 04-24-2024. [Online]. Available: https://pypi.org/project/torch/. (Accessed May 5, 2024)