TV Shows Muster and Analysis

Abhimanshu Mishra
Binghamton University
amishr11@binghamton.edu

Aditya Bhagwat
Binghamton University
abhagwa1@binghamton.edu

Sharvari Joshi
Binghamton University
sjoshi14@binghamton.edu

Vinit Bhosale
Binghamton University
vbhosal1@binghamton.edu

ABSTRACT

Online entertainment has become a crucial part of life in the modern age. In recent times, the amount of content and the number of content producers has increased exponentially. With such an increased consumption of content, everyone has opinions about different TV shows and water cooler discussions have become commonplace. People take to different social media forums to express what they think and connect with the world at large. This provides data scientists and researchers a unique opportunity to study the pulse of the public about a specific show easily. We analyse reviews and tweets talking about TV shows by performing sentiment analysis, topic modeling and social network analysis. We see that shows with positive reviews and chatter also have a higher rating. We designed a smooth web interface to demonstrate some of the visualisations for the data we collected as part of this project. We allow users to check out reviews based on a range of sentiment scores. These scores have been calculated by us earlier in the project. Additionally, users can also see a list of TV shows rated higher than a threshold. Our publicly available link is https://team-fire-viz.herokuapp.com/.

1 INTRODUCTION

Our project aimed to collect data about TV Shows from different social media platforms like Twitter and IMDb [1] like tweets and ratings on a large scale and draw patterns about the popularity of a show, it's influence on users from different age groups, the amount of engagement it receives based on retweets, likes and replies as well as how the show trends in different parts of the world. As the process continues we now have abundant data collected to analyze the shows and have also successfully analyzed each TV show depending on certain factors.

Based on the collected data we created a User Interface, which is essentially a replica of the results and insights we derived (based on the collected data). The User Interface consists of a dashboard which shows the depth of the collected insights.

One of the insights we derived was the sentiment analysis of each TV show we analyzed upon which shows how the review sentiments vary over a period of time and how the performance of a TV show can be measured using these review sentiments. We used a scale between 0-100 to let the user choose a range of ratings. Upon choosing a value, the results get displayed along with the sentiment analysis based on the chosen value.

One of the most common things users do before starting to watch a TV show is to check its ratings. Our dashboard also has the capability of displaying all shows above a certain threshold, which the user can pick by providing a value inside the value bar.

In order to create an interactive dashboard for displaying these analyses, we have used Flask [4], Jinja2.

2 DATA SOURCES

2.1 Twitter

Twitter [2] is a micro-blogging and social networking service where users post and interact with messages, "tweets," restricted to 280 characters. Information can be shared using photos, videos and links as and when they're happening, enabling insightful and real-time search results. Tweets are fetched using the Official Twitter API which allows up to 50 requests per hour when the data fetched is from a percent of sampled stream of the total tweets collected in real-time. We use the bearer token for authentication, provided by twitter API when requested using the developer account. We filter and scale the search request according to the fields necessary to perform analysis and the user related data. Tweets collected are provided in JSON format which is collected and stored on our remote storage. Tweets collected may contain some noise or unnecessary data which can be easily removed before storage. We do so by storing the tweets locally in a python list where they're stored in a dictionary format thus creating a list of dictionaries where each entry in the list (a dictionary) represents an individual tweet received. The JSON response (tweet) consists of various fields like text field containing text of status update, user field containing information of user who posted the information, coordinates field containing Geographic information of the user, retweet count field containing number of times tweet has been retweeted, favorite count field containing number of times tweet has been liked by any twitter user. Based on the extracted field values, analysis can be made to decide whether a TV show has a good review or not based on the keywords in the tweet, the number of times it has been retweeted, the likes and replies it received and also the number of times it trended at a particular location in the world. This information also helps in determining how negative or positive tweet can easily influence other social media users in reviewing a TV show. Since the python script to perform this data collection is to be performed everyday at regular intervals of time. We use cron-job to achieve this task. This cron-job runs every hour with an interval of 10 minutes everyday. As the cron-job schedules the script to run after every 10 minutes, the stream is again refreshed so no duplicates arrive while fetching. At an approximate , it can be seen that around 250-350 meaningful tweets arrive after every call. So on an hourly basis we achieve up-to 1500-1600 tweets per hour, which approximates to be about 30,000+ tweet per day. The arrival of such amount Of tweets per day was monitored using

the ROBO 3T application made for keeping a tab on the mongo Databases and also to query and retrieve data from it. As the data stored in the database was separated by TV shows into collections, every meaningful tweet of a show ended up getting into the correct collection in JSON format. We gathered around 229,000+ tweets up-to 26th November, 2020 and we stopped the data collection as the data collected was enough to derive meaningful insights. The major contributors to the twitter data collected were the TV Shows like The Mandalorian , The Queen's Gambit, ,The Crown , The BlackList and The Schitt's Creek. The data collection saw an upward trend during the afternoon and night where the approximate data collection per hour would almost be increased by twice. The total data size of each collected tweet was 0.02 MB and each tweet collected varied in the incoming number of fields from about 14 to 16. We required a total VB is a stupid fucker storage of 1.5 GB for storing this data. After the collection process ended, the data was exported using the Studio 3T application to get the data stored on Drive storage. We consider this to be meaningful data since the data and noise cleansing has already been done.

2.2 IMDb

IMDb (Internet Movie Database) [1] offers an extensive database of information related to films, television programs, video games, and streaming content online – including crew details, plot summaries, trivia, fan and critical reviews. IMDb currently contains approximately 6.5 million titles (which includes movie and TV show titles),10.4 million personalities as well as 83 million registered users. Based on the launch of TV shows on worldwide streaming platforms, the popularity of these shows may vary with time which is one major factor of consideration for the task at hand [3]. After analysis we create a text file of popular Tv shows we then read each Tv show one by one and start collecting TV shows data by using OMDb API and scrapping IMDB website. Each call of OMDb API depends on either TV show title or the imdbID, we took the approach of using title of Tv shows to call the api and as a response we got all the attributes like "Title", "Year", "Rated", "Released", "Genre", "Writer", "Plot", "Language", "Ratings", "imdbId" and others. For user review of Tv show we scrape the imdb website. In scraping process, we encounter Load More button which loads all user reviews for the Tv show, so to get all the user reviews we used Selenium. Selenium automation was implemented to first click all the Load More button of the Tv show review webpage to load all the reviews. Once the automation is done, we then get the outerHTML of the webpage and convert it into lxml tree using lxml.etree.parse() then, with the help of xpath queries we parse the tree to fetch the exact node that will provide the user review data of the webpage and start storing those reviews. After storing all the reviews we append those with the OMDB api results and create a JSON collection object of each TV show which then get inserted into IMDb collection of MongoDB database on remote server and every TV shows associated data can be retrieved on demand for analysis.

3 DATA COLLECTION

We start by running a Daemon script on the virtual machine Written using cronjob which runs the python script every 15 minutes of an hour. This python script uses Twitter API to fetch the tweets of various TV shows in JSON format. The hashtags provided by us to the API results in the tweets to be fetched and will be stored in a list which creates list of dictionaries. Certain fields are extracted from this twitter data and those fields are stored into MongoDB into a single database with multiple collections. Each collection represents a different TV show in the database.

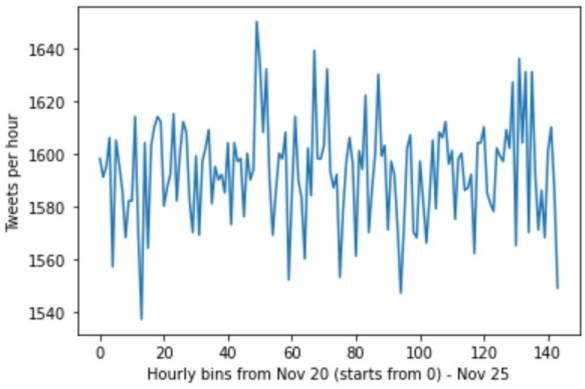


Figure 1: Hourly tweet collection over Nov 20-Nov 25

Figure 1 shows the number of tweets collected each hour in the given time period of November 20-25. It sums up to 229,000 tweets over this period of time.

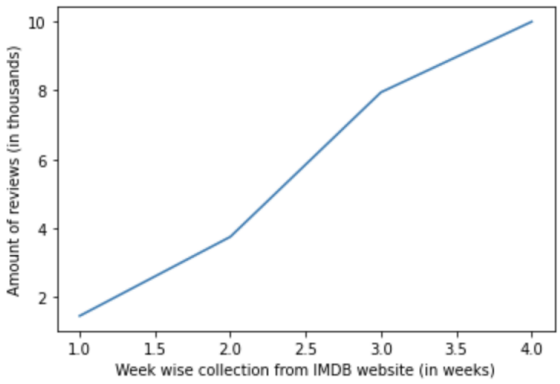
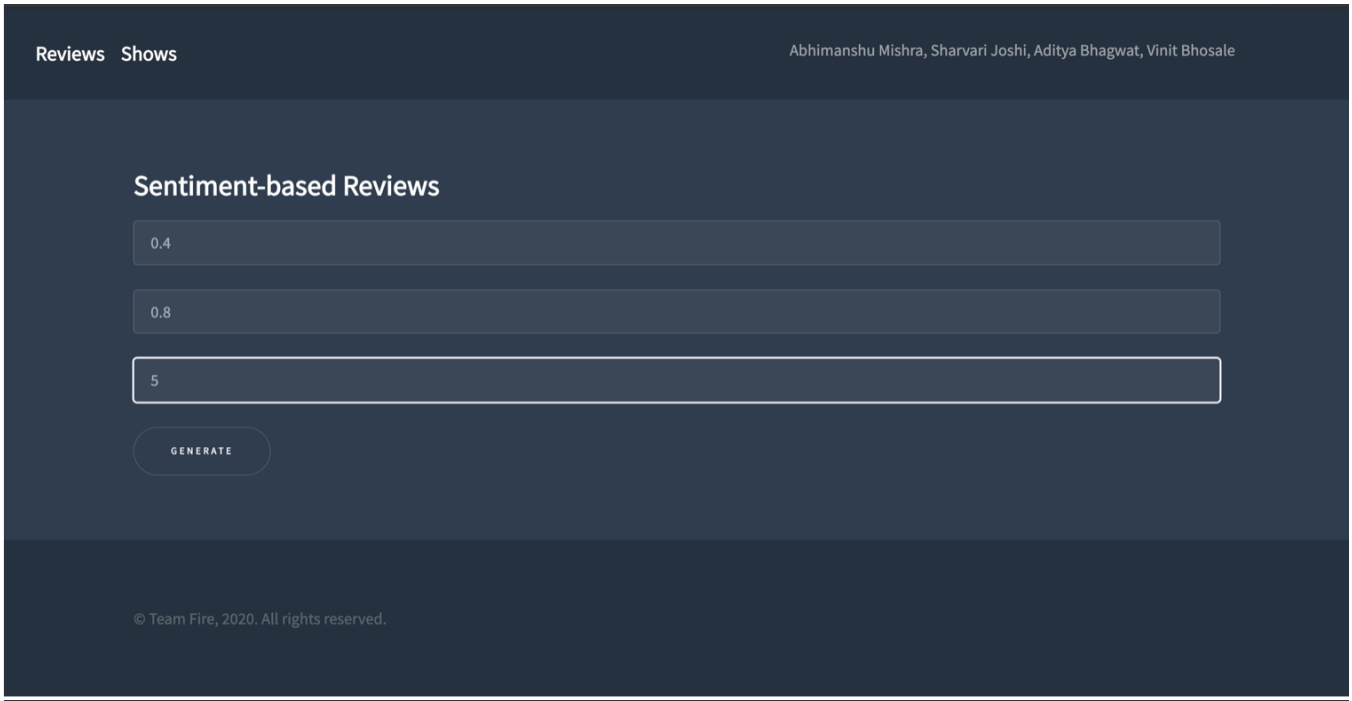


Figure 2: A graph showing the collection of data weekly from the IMDB website.

The above graph plots the number of reviews we collected from IMDb against a particular week. We expect to collect approximately 10,000 reviews from the IMDb website from which we estimate around 5k-6k to be meaningful. We would be able to obtain a fair graph from sample based on these estimations

4 FLASK

A micro-framework is a code library that lets one build reliable, scalable, and maintainable web applications by providing reusable code or extensions for common operations. There are a number of frameworks for Python, including Flask, Tornado, Pyramid, and Django.



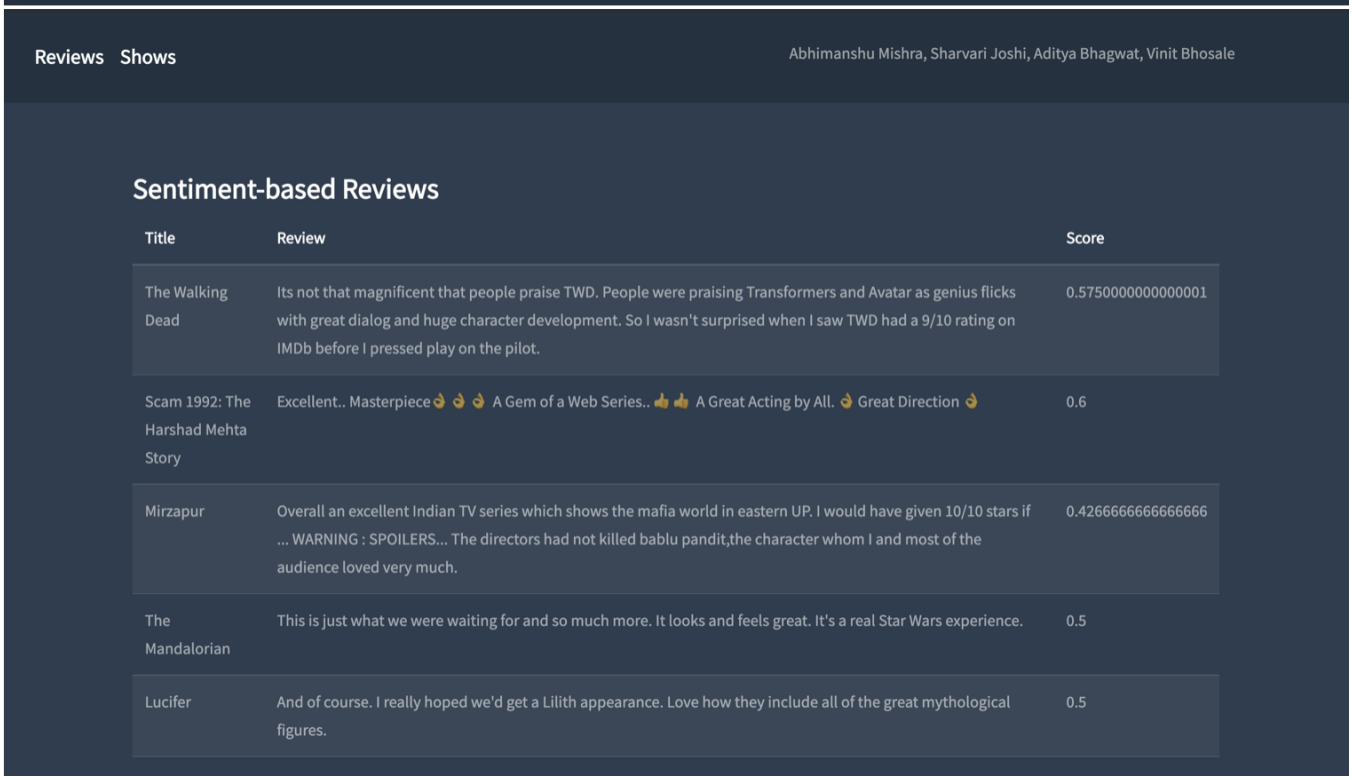


Figure 3: Screenshot of examples being sampled based on the provided values for sentiments

Flask is a Python Framework for Back End Web Development. Flask is a lightweight WSGI(The Web Server Gateway Interface is a simple calling convention for web servers to forward requests to web applications) web application framework. It is designed to make getting

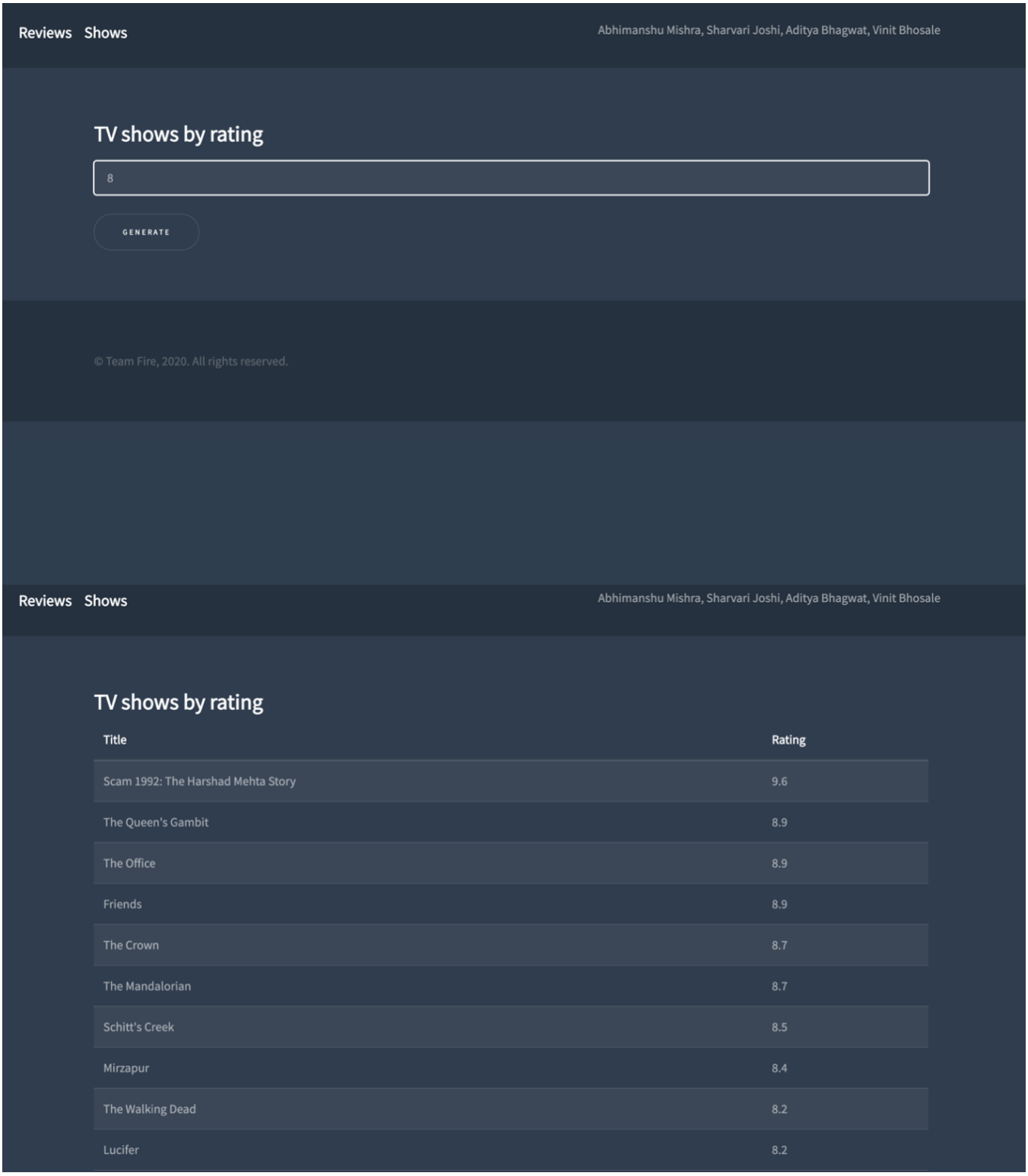


Figure 4: Screenshot of TV shows being displayed in descending order above a given value of rating

started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug(Werkzeug is a comprehensive WSGI web application library) and Jinja(referred to as Jinja2 is a Python template engine used to create HTML, XML or other markup formats that are returned to the user via an HTTP response) and has become one of the most popular Python web application frameworks. Flask is said to be more simple and flexible as compared to Django and provides the user with enough control over the work.We have to first set up the flask server on the local host and later deploy it on python. In order to use Flask with python we first need to install flask library from the FLASK Module. This allows us to now make use of the flask methods and classes to be used in your python script.The python script starts the flask server on localhost and default port (5000) making the URL: http://127.0.0.1:5000/ (http://127.0.0.1:5000/). After initializing the python file, we have to tell the Flask what we want to do after the web page loads. The python code @app.route("/", methods = ["GET"," POST"]) tells the Flask what to do when we load the home page of the website. In methods: GET method is the type of request our web browser will send to the website when it accesses the URL of the web page.

5 WEB APPLICATION

We built a web app using Flask and Jinja2 templating engines. The use of Flask and Python also allowed us to leverage Pandas, which is very effective in dealing with data.

We built a RESTful API which interfaces with our frontend to create a seamless, responsive interface. We present two interactive visualizations from our previous analysis: (1) users can view a list of TV shows higher than a threshold rating they specify, and (2) users can view any number of reviews that fall in a range of sentiment scores. They can provide their own values for these ranges and the number of reviews to display.

The first API endpoint serves the purpose of returning the TV shows above a given rating. It is situated at the /rating route. The input to this endpoint is a form with a rating field. It returns a list of dictionaries which is used to populate the frontend. The information is displayed in descending order of IMDb ratings.

The second API endpoint is located at /sentiment. Its inputs are: (1) minimum sentiment score, (2) maximum sentiment score, and (3) number of reviews to display. The server creates a dataframe of all reviews in this specified range, and randomly sample the desired number of reviews. This is done in order to avoid monotony of the same reviews (out of a very large set) being shown every time.

6 CONCLUSION

We designed and built a seamless web app using Flask and Jinja2. This application serves as a platform for interactive visualisations of the various analyses we performed earlier in this project.

REFERENCES

[1] [n.d.]. https://www.imdb.com/
[2] [n.d.]. https://developer.twitter.com/en/docs/twitter-api
[3] Adam Nyberg. 2018. Classifying movie genres by analyzing text reviews. *CoRR* abs/1802.05322 (2018). arXiv:1802.05322 http://arxiv.org/abs/1802.05322
[4] Armin Ronacher. [n.d.]. https://flask.palletsprojects.com/en/1.1.x/#