Tweet Faker

Venisa Sinha (vs2)

Mike Peretz (peretz2)

Overview

We created a Python tool that can generate believable tweets based on the style and substance of someone's Twitter history. The user can input a Twitter username, and our app will scrape their tweet history, analyze the content, and generate new tweets that are logically plausible and that resemble the style of the text samples. The app also shows a word cloud which is generated based off of all words mined from the previous tweets, minus the stop words and other irrelevant parts. The primary use of our application is entertainment. We've had a lot of fun ourselves reading our fake tweets, such as:

"Make america great again, despite the kentucky derby decision!" - @realDonaldTrump

"Lift sanctions on maryland, elections company bought by police omg." -@cher

It's also been a useful tool to understand how a public figure writes and what topics they are most interested in. For example, Donald Trump's most common word at the time of writing was "great", with "billions" not far behind, and we found that our app commonly places ellipses at the beginning of fake Trump tweets. This is reflective of our preconceptions about his tweeting style, in which he often goes on multi-tweet rants to bypass the character limit (hence the starting ellipses).

Implementation Details

We have two components: API and UI. The API is built in Python using Flask, and the UI is in HTML, CSS, and JQuery. When the user input a Twitter ID into the UI's search bar, it will make a request to our API containing that ID. The API will then scrape their tweets in the **get_tweets** function. This takes about 10 seconds due to having to rely on scraping rather than the Twitter API. We currently pull up to 200 tweets for the sake of demonstration and testing, but that number can be increased to form a stronger model. The function will replace problematic characters--such as slanted quotation marks--and return a list of Tweets. Then, it will send that information to the **split_tweets** function, which will break all of the tweets into a list of words. Additionally, it collects information on word frequencies, common start words, and common end words and calculates weights for use in constructing the Markov Chain. Then, we call the **make_word_cloud** function which forms a dictionary of word frequencies, excluding stop words, and generates a word cloud to visualize on the UI. Then, we pass our list of words to the **generate_transition_matrix** function and start building our Markov model. To do this, we create a dataframe with 2 columns, one being our list of words, and the other our list of words shifted backwards by 1 such that each first word precedes the second word as in the original tweet. Finally, we use our transition matrix to generate a list of fake tweets in **generate_tweets**, which picks a starting term and follows the flow of the model until it reaches an end condition. After receiving the list of fake tweets and word frequencies, the UI programmatically generates a word cloud and stores the fake tweets for the user to cycle

through. We chose this approach so that we would only need to make one API call per search, minimizing the waiting time for the user.

Documentation

API:

First, make sure both **Python 3** and **pip** are installed. *pip* should come with *Python 3* by default, but instructions can otherwise be found here:

https://pip.pypa.io/en/stable/installing/

Then, from the **api** folder, run the following command. Note, this simply installs all of the required packages, so it only needs to be done once after cloning the repo.

```
pip install -r requirements.txt
```

Finally, run the following command to start the server:

```
python main.py
```

If you have an abnormal python setup, run main.py (with no arguments) as you typically would.

UI:

Simply open **index.html** in your preferred web browser (preferably Google Chrome).

Other:

The API can be called from outside of the UI as well (e.g. in Postman or other applications). Simply make a **get** request with a Twitter username as the query parameter. Note, this can also be done by typing the request URL directly into your browser. For example:

http://127.0.0.1:5000/ArianaGrande

Contribution

Venisa was in charge of the entire user experience, user interaction, and front-end for which she used Javascript, HTML, and CSS. She created an input field for users to enter a specific twitter handle and with every new search, the code will call the api to pull all fake tweets which will be stored in an array. If the user wants to generate more fake tweets of the same twitter handle, the tweets can be retrieved in constant time instead of calling the api each time, which would have been severely inefficient. With every search query, Venisa was also in charge of generating a word cloud which would showcase at what frequency each individual word (excluding stop words) is tweeted amongst all their tweets. Mike created most of the back-end, including setting up the API, scraping Twitter data, and creating the model to generate fake tweets.