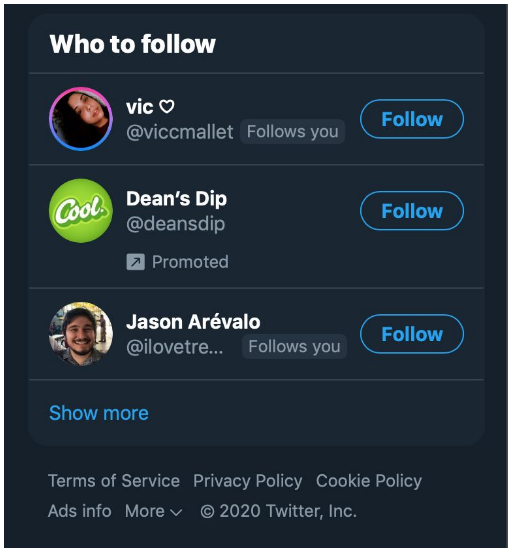
Twitter Recommendations based on Text Report

Team Members:
Rohan Khanna (Team Captain) - rohank2@illinois.edu
Tyler Wong - tylercw3@illinois.edu
Cesia Bulnes - cbulnes2@illinois.edu

Introduction

Twitter is a social media platform allowing users to connect and share thoughts and information. A notable example of Twitter and content shared was seen in 2020, with the presidency and election. There are currently recommendations on who to follow in general for Twitter users. However, when a tweet is made, that tweet does not have suggested/similar tweets that a user can react to or retweet. The purpose of this project is to give users the ability to get recommendations based on their tweets. Suppose you write “love hamburgers and fries”, you should expect to get back a topic, and if you run that same query with our ranker, you will get a list of 10 tweets that are closest in similarity, along with the 10 users that have tweets that are closest to the content of this query. We would also get the topic of this tweet for classification purposes.

Currently on twitter:



Who to follow

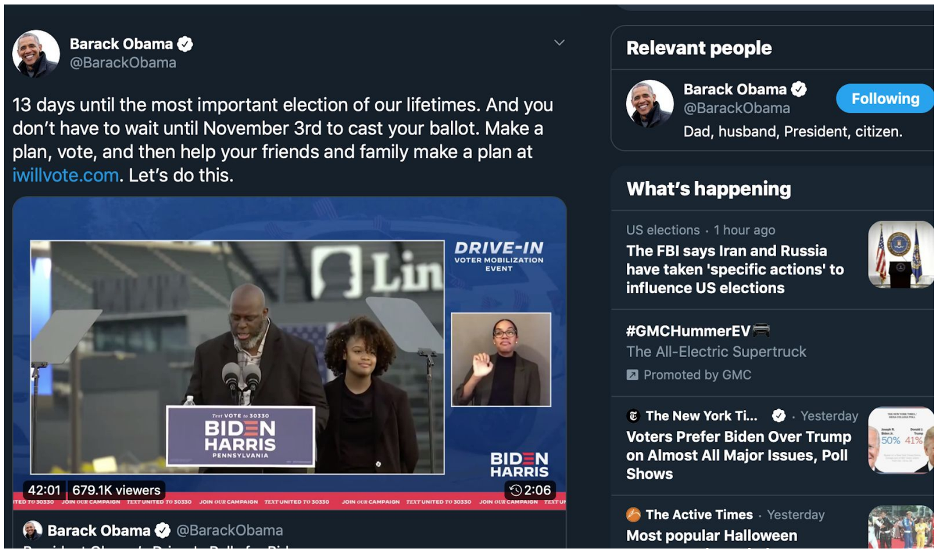
vic ♡
@viccmallet Follows you
Follow

Dean’s Dip
@deansdip
Promoted
Follow

Jason Arévalo
@ilovetre... Follows you
Follow

Show more

Terms of Service Privacy Policy Cookie Policy
Ads info More © 2020 Twitter, Inc.



Barack Obama
@BarackObama

13 days until the most important election of our lifetimes. And you don’t have to wait until November 3rd to cast your ballot. Make a plan, vote, and then help your friends and family make a plan at iwillvote.com. Let’s do this.

Barack Obama @BarackObama

Relevant people

Barack Obama
@BarackObama
Following
Dad, husband, President, citizen.

What’s happening

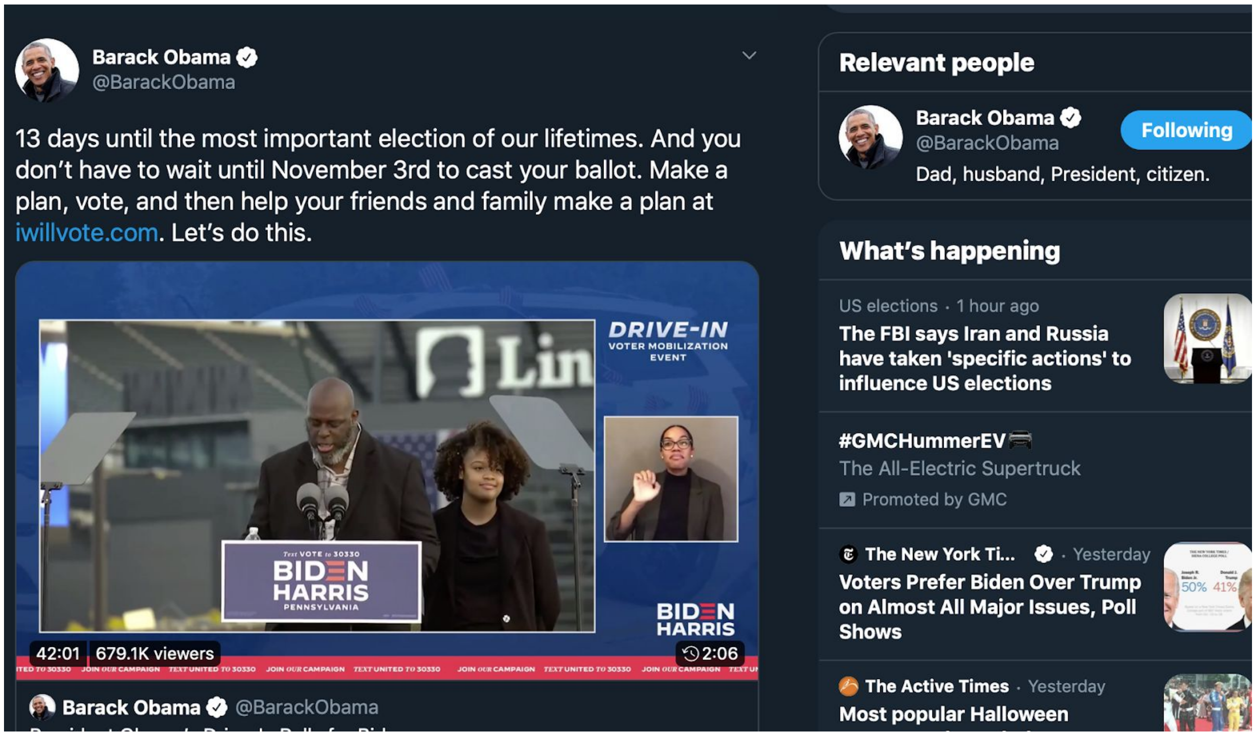
US elections · 1 hour ago
The FBI says Iran and Russia have taken 'specific actions' to influence US elections

#GMCHummerEV
The All-Electric Supertruck
Promoted by GMC

The New York Ti... · Yesterday
Voters Prefer Biden Over Trump on Almost All Major Issues, Poll Shows

The Active Times · Yesterday
Most popular Halloween

What this project accomplishes:



Barack Obama @BarackObama

13 days until the most important election of our lifetimes. And you don't have to wait until November 3rd to cast your ballot. Make a plan, vote, and then help your friends and family make a plan at iwillvote.com. Let's do this.

Relevant people

Barack Obama @BarackObama Following

Dad, husband, President, citizen.

What's happening

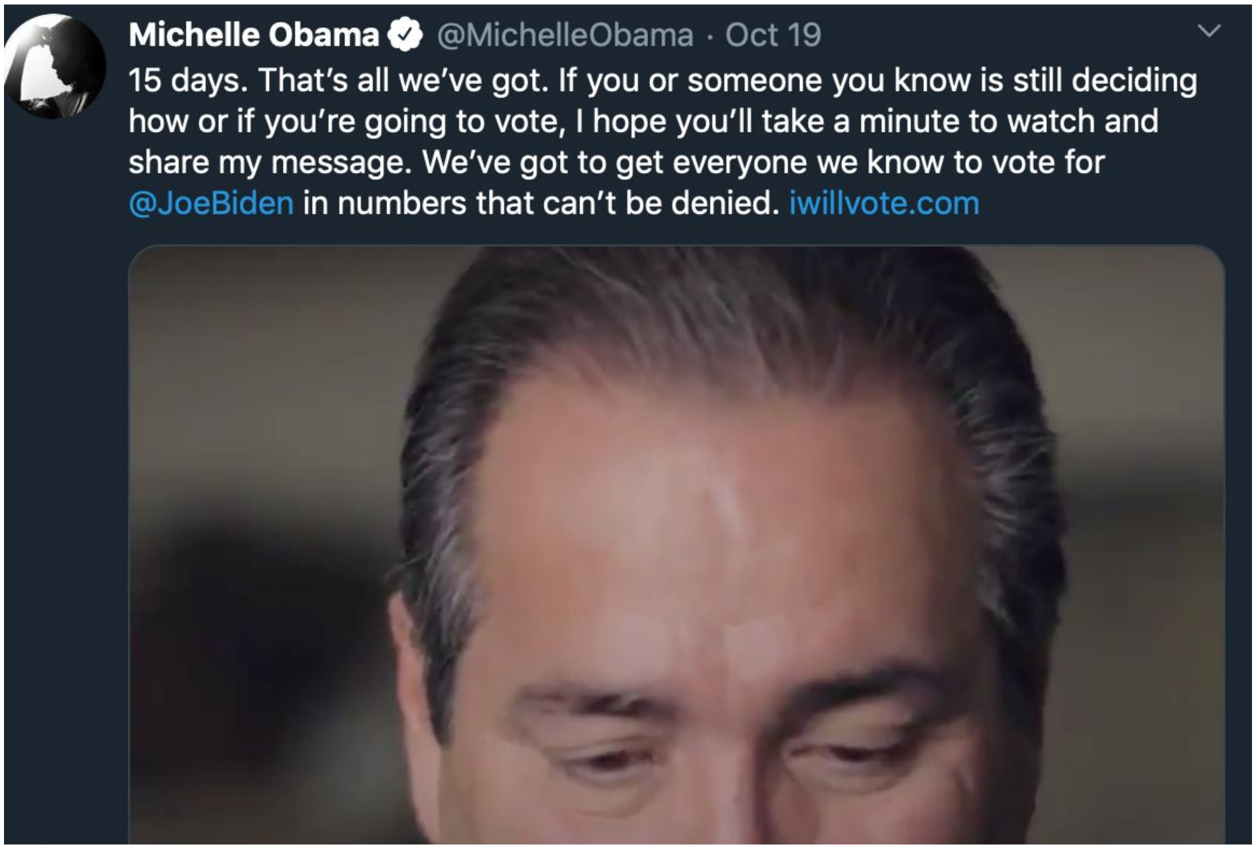
US elections · 1 hour ago
The FBI says Iran and Russia have taken 'specific actions' to influence US elections

#GMCHummerEV
The All-Electric Supertruck
Promoted by GMC

The New York Ti... · Yesterday
Voters Prefer Biden Over Trump on Almost All Major Issues, Poll Shows

The Active Times · Yesterday
Most popular Halloween

The following tweet will have a .89% of similarity.



Michelle Obama @MichelleObama · Oct 19

15 days. That's all we've got. If you or someone you know is still deciding how or if you're going to vote, I hope you'll take a minute to watch and share my message. We've got to get everyone we know to vote for @JoeBiden in numbers that can't be denied. iwillvote.com

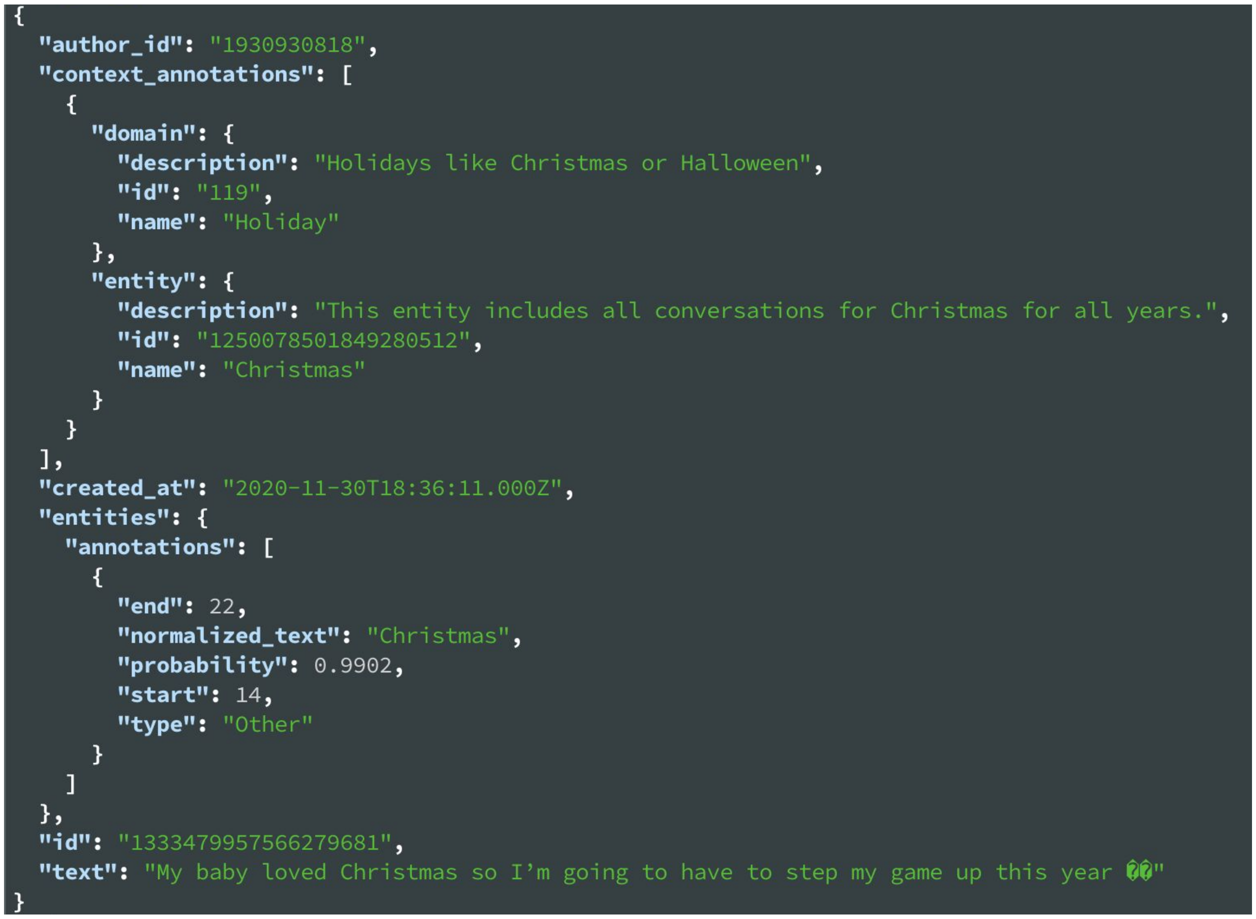
Twitter Data

In order to get data from Twitter, we had to get approved for a developer key from Twitter. This is an application that we submitted which outlined our use case and was promptly approved. Even though we now have this API key, Twitter still limits your use of the platform through API. Most notably, they limit how much request you can do in one hour and they limit how much data you can get per request.

Because of these request limits from the Twitter API, we decided to use an open source Python library to handle requesting data from Twitter's v1.1 API called python-twitter. For querying data from Twitter's v2 API, we directly used Twitter's HTTP API since it's so new, there aren't many open source tools to use.

To get the users we decided to get 3000 random active Twitter users. In order to do this, we queried Twitter's sample stream which provides a subset of active tweets coming in as a stream. Once we obtained a tweet from the stream, we queried the language of the tweet and to check if the user has a public profile or not. If they were english speaking and had a public profile, we obtained their user information.

Once we have our random users, we got the last 7 days of tweets from them. We did this by querying for the tweet IDs and then getting each associated tweet from the v2 API. This was a very expensive operation since twitter limits how many tweets you can get, but given enough time we were able to get all this data. We ended up with around 120,000 tweets as our full data set which is included in the "tweets.tar.gz" file in the "data" folder of our GitHub repository.



```json
{
  "author_id": "1930930818",
  "context_annotations": [
    {
      "domain": {
        "description": "Holidays like Christmas or Halloween",
        "id": "119",
        "name": "Holiday"
      },
      "entity": {
        "description": "This entity includes all conversations for Christmas for all years.",
        "id": "1250078501849280512",
        "name": "Christmas"
      }
    }
  ],
  "created_at": "2020-11-30T18:36:11.000Z",
  "entities": {
    "annotations": [
      {
        "end": 22,
        "normalized_text": "Christmas",
        "probability": 0.9902,
        "start": 14,
        "type": "Other"
      }
    ]
  },
  "id": "1333479957566279681",
  "text": "My baby loved Christmas so I’m going to have to step my game up this year ��"
}
```

Example of a single Tweet data with annotations and entities

How it works

1) You will need Python installed and all the required packages installed.
   `pip install -r requirements.txt`

2) Obtain a Twitter developer account through the Twitter Developer Portal if you haven’t already. Add the consumer key, consumer secret, access token, access token secret, and bearer token to the “twitter_utils.py” main method’s variables. These will be used to interact with the Twitter API.

3) Run the code to get the users and tweets. The code will check to make sure not to regenerate these, so if you run it multiple times you will need to delete the files in the “data” directory of the repository.
   `python src/twitter_utils.py`

Database

SQLite was used to extract the tweets and user information from the Twitter API’s response. We used the Standard v1.1 API to extract information on the users. This was mainly important to later map the user id’s to the screen name, location, etc. In the future, this information can be used to build recommendations of tweets based on the location of a user. Say a user lives in Miami, FL, and they tweet about an upcoming event with a celebrity. There can be a recommendation of tweets with that same geographical location. Not only could Twitter use this for marketing, but also increasing the connectivity that other social media platforms like Facebook have taken as an approach.

In addition to the information on the users, we also have the tweet data from the early access Twitter v2 API. This version of the API was used because it contains the entity and context information that provides data on the subject and relations of the tweet. Some examples of entities are: Barack Obama, IBM, Mountain Dew, and San Francisco. Some examples of context are: TV Shows, TV Episodes, Podcast, Holiday, Politicians, and Video Game.

SQLite stored this information in the following tables:
userTable and originalTweets

Topic Extraction

For topic extraction we use the nltk toolkits and gensim. As we learnt in class LDA is an unsupervised machine learning algorithm that uses a mixture model to generate documents. Each topic can be assigned some prior probability and each topic consists of probabilities for generating a particular word from the vocabulary.

DATA RETRIEVAL/CLEANING:
We developed a few different functions to perform data retrieval and cleaning of the tweets:

- `remove_emoji(text)`
- `remove_links(text)`
- `remove_users(text)`
- `clean_tweet(tweet)`

All of these functions are meant to clean up the data so that we can perform better analysis of the data with better accuracy. We process all of our original tweets from the SQLite database through these functions.

The `remove_emoji(text)` method removes any emojis that are found in the tweet because we found that emojis didn't provide much meaningful information. The `remove_links(text)` method removes any HTTP or HTTPs links that are found in the tweet text since that data isn't useful when determining category. The `remove_users(text)` method removes any "@" mentions for any other user. The `clean_tweet(tweet)` method makes the tweet all lowercase, removes punctuation, removes any stopwords, and removes any words that are 2 characters or less.

CORE ALGORITHM:

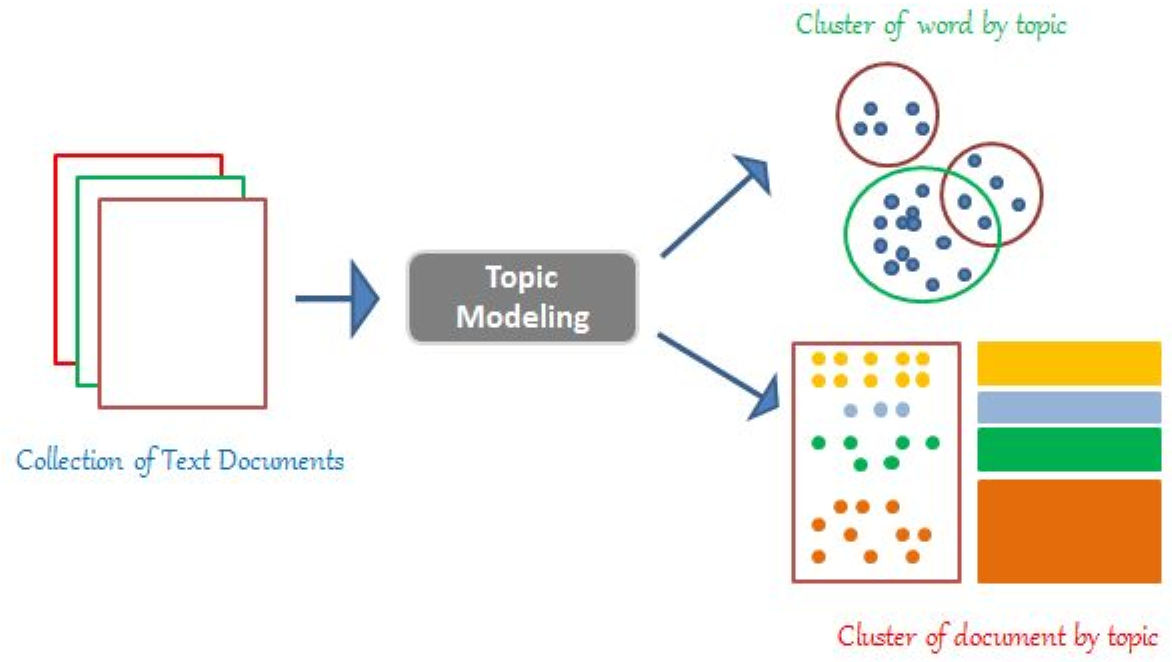


Figure:1

Picture reference: https://medium.com/@osas.usen/topic-extraction-from-tweets-using-lda-a997e4eb0985

We decided to go with 5 topics as this was a proof of concept and our test data was relatively small and could be described with 5 topics. After cleaning our data, we essentially create a document term matrix where the rows are the cleaned tweets and the columns are words. The matrix entries hold the count of those words in that particular tweet. For example matrix[i][j] will denote the count of that word in the tweet. This is essentially following the core concept of creating a bag of words model for our project. We then send this document count into the gensim lda model and it runs the LDA model to generate the topics. We run the lda with 20 passes for convergence and we decided to use gensim because it gives us the ability to store this model on the disk and hence reuse it for later applications. Moreover, we use the multicore lda as it can process the data parallely on multiple threads leading to better overall performance. After this for a new tweet we use this lda model and try to assign probabilities for which topic this new tweet could belong to.
As shown in the demo video, we ran our topic classifier on the test tweet :

```
Twitch is so cool! #twitch #twitchstreamer #digital
```

Our classifier did a good job of classifying this to topic 4 which was :
Topic: 4
Words: 0.020*"marketing" + 0.013*"#digital" + 0.008*"media" + 0.007*"#youtube" + 0.007*"#twitchde" + 0.007*"#germanmediart" + 0.007*"#twitch" + 0.007*"#seo" + 0.007*"#email" + 0.007*"marketer"

Overall, this was a good approach as a proof of concept however, we definitely have a lot of room to develop on this project.

How to run it:
To run the code a user can put test tweets in the test tweets.json file and just execute the python3.8 topicdeterminant.py this will classify the tweet into the most likely topic. Using this topic modelling you can draw a relationship between people who have similar tweets.

Execute `python3.8 topicdeterminant.py` on the terminal

IMPROVEMENTS:

1. We could have worked more on cleaning the data to account for duplicate tweets in the form of retweets. Moreover, a lot of our accuracy issues were centered around bad data. We had tweets like : **`oo I know what I\u2019m going to do.`** These tweets are extremely hard to classify into any particular topic. There is no broad common topic that the above tweet could be classified into and hence such a tweet just corrupts our training data.
2. We could have worked to put a special emphasis on features of tweets like hashtags.

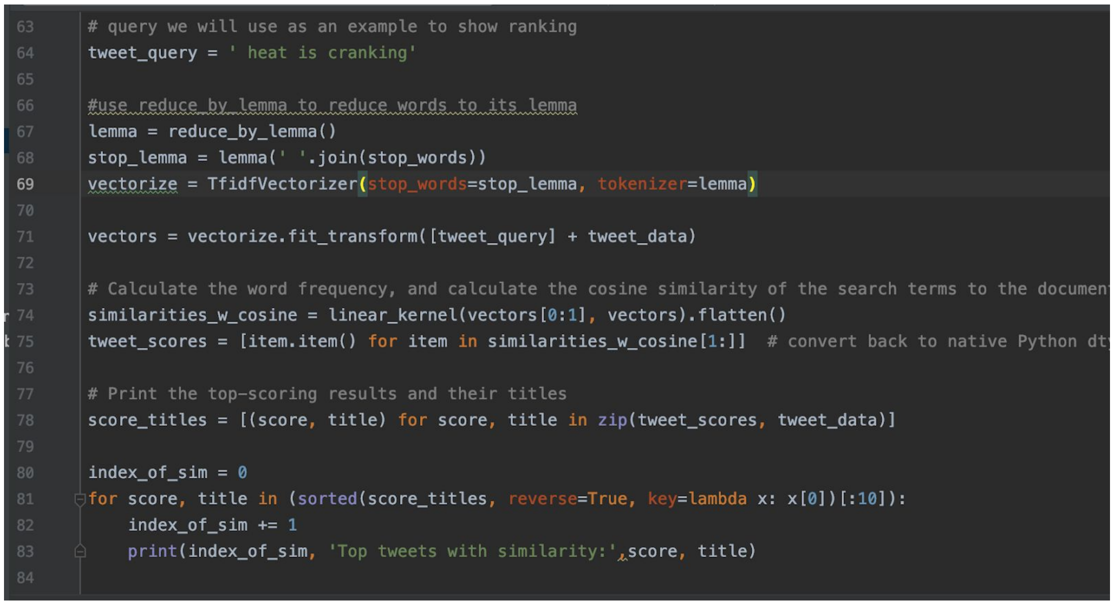
Sources:

1. https://medium.com/@osas.usen/topic-extraction-from-tweets-using-lda-a997e4eb0985
2. http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/
3. https://github.com/enoreese/topic-modelling/blob/master/preprocessor.py

Ranker of tweets and users

After loading the tweet data and creating a map of the author id’s to the author’s screen name, the tweets are then using tf-idf weights per word to score shared words.

Given the following tweet_query “heat is cranking” I want to return a recommendation of tweets that are similar to the tweet_query, along with the % of their similarity.



```python
# query we will use as an example to show ranking
tweet_query = ' heat is cranking'

#use reduce by lemma to reduce words to its lemma
lemma = reduce_by_lemma()
stop_lemma = lemma(' '.join(stop_words))
vectorize = TfidfVectorizer(stop_words=stop_lemma, tokenizer=lemma)

vectors = vectorize.fit_transform([tweet_query] + tweet_data)

# Calculate the word frequency, and calculate the cosine similarity of the search terms to the documen
similarities_w_cosine = linear_kernel(vectors[0:1], vectors).flatten()
tweet_scores = [item.item() for item in similarities_w_cosine[1:]]  # convert back to native Python dt

# Print the top-scoring results and their titles
score_titles = [(score, title) for score, title in zip(tweet_scores, tweet_data)]

index_of_sim = 0
for score, title in (sorted(score_titles, reverse=True, key=lambda x: x[0])[:10]):
    index_of_sim += 1
    print(index_of_sim, 'Top tweets with similarity:',score, title)
```

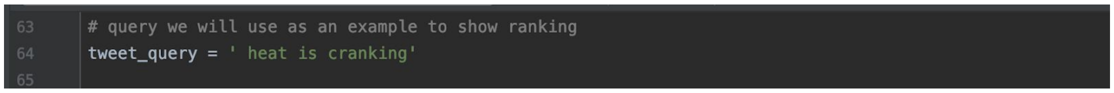
We introduced a reduce_by_lemma function to reduce words to their lemma. Stop words from english nltk are the following: {‘ourselves’, ‘hers’, ‘between’, ‘yourself’, ‘but’, ‘again’, ‘there’, ‘about’, ‘once’, ‘during’, ‘out’, ‘very’, ‘having’, ‘with’, ‘they’, ‘own’, ‘an’, ‘be’, ‘some’, ‘for’, ‘do’, ‘its’, ‘yours’, ‘such’, ‘into’, ‘of’, ‘most’, ‘itself’, ‘other’, ‘off’, ‘is’, ‘s’, ‘am’, ‘or’, ‘who’, ‘as’, ‘from’, ‘him’, ‘each’, ‘the’, ‘themselves’, ‘until’, ‘below’, ‘are’, ‘we’, ‘these’, ‘your’, ‘his’, ‘through’, ‘don’, ‘nor’, ‘me’, ‘were’, ‘her’, ‘more’, ‘himself’, ‘this’, ‘down’, ‘should’, ‘our’, ‘their’, ‘while’, ‘above’, ‘both’, ‘up’, ‘to’, ‘ours’, ‘had’, ‘she’, ‘all’, ‘no’, ‘when’, ‘at’, ‘any’, ‘before’, ‘them’, ‘same’, ‘and’, ‘been’, ‘have’, ‘in’, ‘will’, ‘on’, ‘does’, ‘yourselves’, ‘then’, ‘that’, ‘because’, ‘what’, ‘over’, ‘why’, ‘so’, ‘can’, ‘did’, ‘not’, ‘now’, ‘under’, ‘he’, ‘you’, ‘herself’, ‘has’, ‘just’, ‘where’, ‘too’, ‘only’, ‘myself’, ‘which’, ‘those’, ‘i’, ‘after’, ‘few’, ‘whom’, ‘t’, ‘being’, ‘if’, ‘theirs’, ‘my’, ‘against’, ‘a’, ‘by’, ‘doing’, ‘it’, ‘how’, ‘further’, ‘was’, ‘here’, ‘than’}.

We also lemma the stop words with stop_lemma. We then initialize TF-IDF Vector with the Lemma function under vectorize. The next thing to do here is create a vector between the tweet_query that we are looking for with the tweet data we collected. These vectors will be examined by creator cosine similarities between the vectors. Then we go through the flattened vectors to obtain the scores by the vectors.

For score_titles, we want to get the score per tweet. Here we view tweet_data as the title since it’s hard to give a title to one tweet. After, we print the authors associated with the recommended tweets based on similarity. These authors are printed for the user to know which authors may produce similar content as them.

How it works

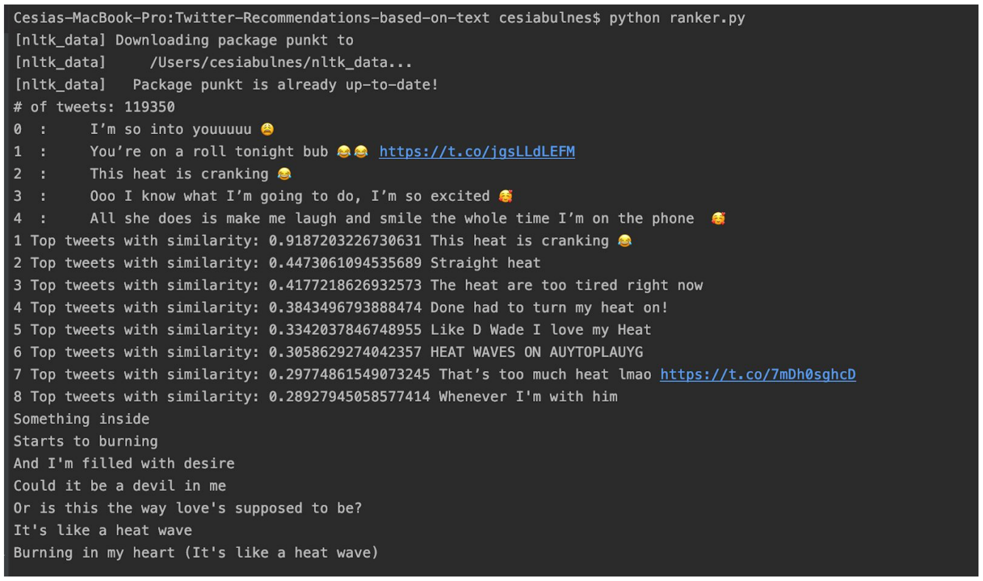
1) To begin with, once git cloned, go to data and unzip tweets.tar.gz, this will unzip tweets.json.
   `tar -xzvf tweets.tar.gz`
2) Like shown above, you can see that as a user, you can choose a query to substitute into tweet_query in the ranker.py code. **You may replace the writing within this query to obtain similarities with different queries.**



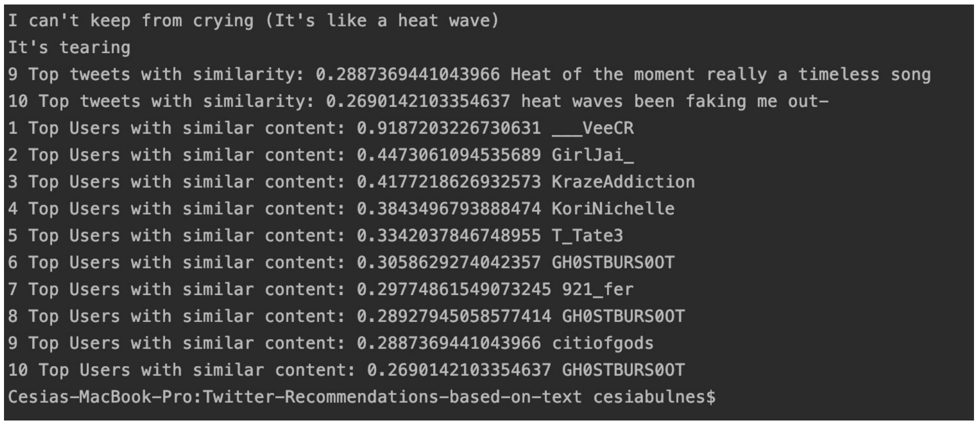
```
# query we will use as an example to show ranking
tweet_query = ' heat is cranking'

```

3) **To execute simply write the following in the terminal: python ranker.py**

Depending on your python version, you may have to pip install some nltk libraries.
You would get a message saying whether you have to pip install nltk for example. Follow the prompt in the terminal and that should be resolved.
Upon executing this you will receive the following output:



```
Cesias-MacBook-Pro:Twitter-Recommendations-based-on-text cesiabulnes$ python ranker.py
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/cesiabulnes/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
# of tweets: 119350
0  :     I'm so into youuuuu 😩
1  :     You're on a roll tonight bub 😂😂 https://t.co/jgsLLdLEFM
2  :     This heat is cranking 😂
3  :     Ooo I know what I'm going to do, I'm so excited 🥰
4  :     All she does is make me laugh and smile the whole time I'm on the phone  🥰
1 Top tweets with similarity: 0.9187203226730631 This heat is cranking 😂
2 Top tweets with similarity: 0.4473061094535689 Straight heat
3 Top tweets with similarity: 0.4177218626932573 The heat are too tired right now
4 Top tweets with similarity: 0.3843496793888474 Done had to turn my heat on!
5 Top tweets with similarity: 0.3342037846748955 Like D Wade I love my Heat
6 Top tweets with similarity: 0.3058629274042357 HEAT WAVES ON AUYTOPLAUYG
7 Top tweets with similarity: 0.29774861549073245 That's too much heat lmao https://t.co/7mDh0sghcD
8 Top tweets with similarity: 0.28927945058577414 Whenever I'm with him
Something inside
Starts to burning
And I'm filled with desire
Could it be a devil in me
Or is this the way love's supposed to be?
It's like a heat wave
Burning in my heart (It's like a heat wave)
```



```
I can't keep from crying (It's like a heat wave)
It's tearing
9 Top tweets with similarity: 0.2887369441043966 Heat of the moment really a timeless song
10 Top tweets with similarity: 0.2690142103354637 heat waves been faking me out-
1 Top Users with similar content: 0.9187203226730631 ___VeeCR
2 Top Users with similar content: 0.4473061094535689 GirlJai_
3 Top Users with similar content: 0.4177218626932573 KrazeAddiction
4 Top Users with similar content: 0.3843496793888474 KoriNichelle
5 Top Users with similar content: 0.3342037846748955 T_Tate3
6 Top Users with similar content: 0.3058629274042357 GH0STBURS0OT
7 Top Users with similar content: 0.29774861549073245 921_fer
8 Top Users with similar content: 0.28927945058577414 GH0STBURS0OT
9 Top Users with similar content: 0.2887369441043966 citiofgods
10 Top Users with similar content: 0.2690142103354637 GH0STBURS0OT
Cesias-MacBook-Pro:Twitter-Recommendations-based-on-text cesiabulnes$
```

Like discussed above, the results result in the number of tweets, a sample of five tweets from the total number of tweets, the top 10 tweets with similarity of content to the original tweet_query, and the top 10 users who made the tweets with most similarity.

For testers: edit the query to anything random that you may think of as a tweet. Run the ranker.py to get recommendations on similar tweets.

Improvements:
The performance of the ranker was where we missed the mark. We could have optimized the code to perform better. We originally wanted results in 2 seconds, but when doing a lemma, the results take longer than without the lemma. With a lemma it takes approximately 30 seconds longer to obtain the results.

Sources: This work was done by following the following github tutorial which explained using TF-IDF with cosine similarities.
https://github.com/4OH4/doc-similarity/blob/master/examples.ipynb

Contributions of Team Members

Team Member	Hours Worked	Contributions to Project
Rohan Khanna (Team Captain)	20 hours	1. Performed partial cleaning of the tweets data, ie, removing emoticons, punctuations etc. 2. Wrote the code for the main lda algorithm used to classify a tweet into a topic 3. Worked on software documentation and software usage presentation.
Tyler Wong	20 hours	1. Set up Twitter API for project and familiarized myself with both versions of the Twitter API. 2. Gathered 3000 active random Twitter users and their last 7 days of claned categorized tweets using the Twitter API. 3. Worked on software documentation and software usage presentation.

Cesia Bulnes	20 hours	1. Set up the skeleton of the database 2. Worked on the ranking of tweet content, to recommend top tweets a user would like based on their query/tweet, and returned the top users 3. Worked on software documentation and software usage presentation