Searching for Hate Speech

(Final Report)

1st David Jamriska
University of Illinois Urbana Champaign
Graduate College
Irvine, USA
djj3@illinois.edu

2nd Ayan Putatunda
University of Illinois Urbana Champaign
Graduate College
San Francisco, USA
ayanp2@illinois.edu

3rd Manojit Saha Sardar
University of Illinois Urbana Champaign
Graduate College
Chicago, USA
manojit2@illinois.edu

4th Ganesh Iyer
University of Illinois Urbana Champaign)
Graduate College
San Francisco, USA
gniyer2@illinois.edu

***Abstract*—The project will scrap content from Twitter via a configuration setting and compare that against predefined queries tuned to discover hate speech. Each search term is assigned a weight according to the needs of the user to be used in calculating the ranking.**

Keywords—Search Automation, Hate Speech, Cloud Computing, Search Ranking, BM25 Okapi Customization, Twitter Hate speech

I. INTRODUCTION

Hate speech and the associated hate crime is growing in the United States and law enforcement is devoting increased resources to track and investigate hate crimes. Recent experience has shown that hate crime actors use social media and websites to promote their brand of hate. Currently law enforcement organizations (LEOs) manually search and review the web for these comments and attempt to identify individuals or organizations involved in the coordinated promotion of hate. LEOs could re-task resources from manual searches to investigation with the assistance of an automated tool to identify content of interest. An open source tool would help organization with strained budgets better serve the community. Two members of this team constructed a custom version of the Okapi BM25 [1] ranking function used by search engines to rank and identify content based on keywords. This project expands and extends that ranking function to operate in the cloud at a larger scale. The project will achieve the following goals:

(1) Migrate the solution from end user desktop to a cloud solution.

(2) Modify the datastore away from file based JSON to a scalable data store solution

(3) Implement a scraping function to extract Tweets from Twitter real time based on query topics and user provided inputs

(4) Gather Trump's tweets over the last 10 years for analysis

(5) Analyze the data and publish ranked results for the analyzed tweets

II. DATA SOURCES

A. *Original Data Sources*

The original application read a list of 4000 comments that were scraped from newspaper websites such as OC Register, CNN, Huffington Post, and Fox News. This static set of data was used to tune the custom version of the Okapi BM25 function and validate the search results. For this migration to the cloud the team altered the function to search topics and users that generate hate filled responses to tweets. Using a series of predefined topics the tool issues queries to consisting of

B. *Data Source(s) for the project*

This phase of the project will involve analyzing data from Twitter and hosting it on a cloud-based data store. The application will capture and analyze the twitter data using a corpus of keywords to generate the rankings. The current application will be modified from the existing JSON data store to use an online relational database - SQLite.

For this implementation the project our data set consists of tweets collected over a course of 2019. Leveraging the twitter API and our application, the user can retrieve top tweets that match the topic/theme chosen by the user. We will also be leveraging public twitter search API to collect a corpus based on commonly identified terms which have been known to signify hate-speech. Using the twitter API and the scraping function, last 10 years of Donald Trump's tweets were gathered for analysis

Reference:

1. https://developer.twitter.com/en/docs/tweets/filter-realtime/api-reference/post-statuses-filter.html
2. Donald trump tweet handle: @realDonaldTrump

III. IINTELLECTUAL MERIT

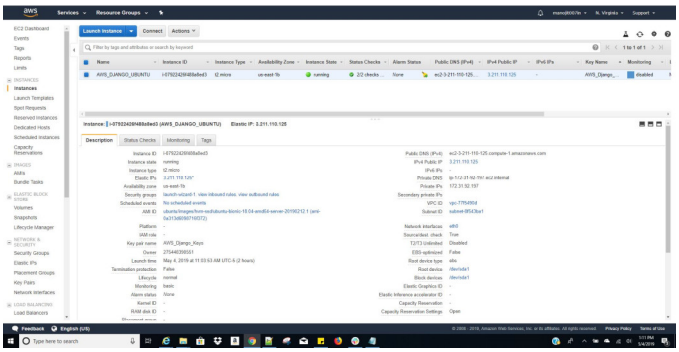
The impact of identifying and monitoring hate speech has gained significant prominence in a world where social media is increasingly pervasive. Twitter in particular invests significant amount of time and resources in detecting, monitoring, and combating hate speech through subjective manual reviews of

questionable content. By actively applying the custom version of the Okapi BM25 algorithm on twitter data we will help actively identify and score tweets. This will help detect and aid the investigation of hate speech.

IV. Cloud Infrastructure and Approach

A. Our cloud computing application on top of Django, and the open source Python Web Framework. Django goes equally well with AWS EC2 (IAAS) as well Elastic Beanstalk (SAAS). In fact with Elastic Beanstalk your clouding computing platform comes with some attractive built-in features like capacity provisioning, load balancing, auto-scaling, and application health monitoring

The team launched and configured a EC2 Ubuntu micro instance, specifically **t2.micro** instance type based out in the **us-east-1b** availability zone with an elastic IP, a service provided by Amazon.



B. Details of the technologies/tools/features used to set up the Django App and running EC2 ubuntu 18.4 platform provided below:

Python 3.6.7	*Pre-installed with the ubuntu OS*
Venv	*Module provides support for 'virtual environments. Installed from CLI*
Django 2.2	*Python based open-source framework facilitating rapid deployment and high scalability*
Nginx	*High-performance HTTP server with reverse proxy server. rich feature set and simple configuration*
Gunicorn	*Python based web gateway interface which is compatible with web frameworks*
Supervisor	*Client/server system for per-process application management*
Elastic IP	*AWS feature which allows our EC2 system to use a unique fully-qualified domain name which is persistent.*

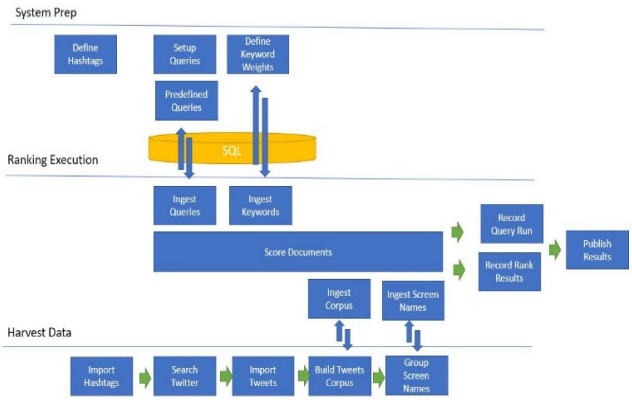
The project team will perform the following:

1) Migrate the current solution to run as a cloud based service.
2) Modify the datastore away from file based JSON to a scalable data store solution, for example SQL Server, SQLite, MongoDB, Hadoop, etc. Update: *For this effort the team has selected SQLite.*
3) Procure a large corpus of tweets from 2019 for research and analysis. We expanded the scope of the data to collect 10 years or corpus of Donald Trump's tweets for analysis
4) Implement data scraping to analyze and score and rankdata from Twitter .
5) Publish analysis on the said data that reach a threshold defined as part of the query.

V. Preliminary Evaluation/Results

The customizations performed on the Okapi BM25 ranking function are believed to be unique and provide a targeted solution to running a set query against a list of articles. Each user can tune the query according to their priorities. The project team is working with a local LEO organization to productionize this solution. A manual implementation of this search has previously resulted in the arrest and conviction for hate crimes in the area of operations (AO) of this law enforcement organization which made national news. The project team looks forward to applying the skills and knowledge gained in this class to enable LEOs to take action against the growing scourge of hate in America. The conversion from text-based data to SQLite database is still on-going, initial results from the preliminary conversion is showing significant performance improvements. The application now reads keywords and corpus (content) from a local database with about a 25% performance increase. Writing results is currently still writing to text files which once complete it is expected this will result in further improvements.

VI. Method/Design



The conversion from the text-based data sources to a normalized database was straight forward and included accommodation for features that will be included in a future release. The database selected is SQLite3, the retrofitting of the existing application proved to be more extensive than expected and has resulted in a slight overrun on the schedule, but does not impact the overall delivery plan.

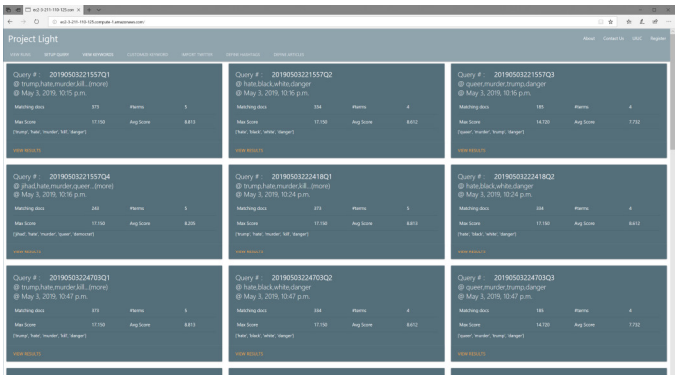
(1) **articles** – In the event the corpus was pulled from the comments section of an article this contains information leading back to the original article, such as title, URL, date, source.
(2) **corpus** – Contains the contents pulled from sources including relationship with other content on the from the same extract. Examples include tweets, comments on web pages, postings on reddit. Contains a like back to the original article if from a web page
(3) **keyword_categories** – Meta categories that link keywords back to subjects, in the case of hate content it groups keywords into categories such as racial, gender, religious, nationality, political party.
(4) **keywords_master** – Contains a list of all keywords used in evaluation and a default weight and keyword category.
(5) **profiles** – For corpus entries contains a link to the profile/user name that was used to generate the corpus. Used to link comments to specific profiles, future enhancement will allow ranking of profiles based on production of hate related content, not in scope for this deliverable.
(6) **queries** – Pre-defined queries to be run against corpus content. Contains a list of words and the owner of the query. Allows for automated re-running of queries as new content is gathered.
(7) **query_runs** – Each time a query is run the contents of the query run are recorded in the table and linked the results. Allows users to identify what parameters were used when the query was run as the query can be modified by the user.
(8) **query_words** – Future enhancement allows the words in the query to have run specific weights, by default the keyword_master contains the weight to be used for all queries.
(9) **Results** – each run of a query is recorded in the results table along with ranking and link back to query run.
(10) **Sources** – definition of the source of the data such as twitter, Redditt, web pages.
(11) Tweets – Tweet corpus pulled based on the hashtags and at (@) defined in the hashtags table that are currently active.
(12) Hashtags – Hashtags and at (@) entered by the user in the Admin section of Django and used to retrieve tweets from Twitter via the Search API.

PROJECT FLOW AND UI DESIGN:

The user experience is divided into 2 areas, the first for general public and users, the second restricted to logged in users with administrative privileges. General users are restricted to read only displays of information consisting of administrator defined queries, query runs (execution of a set of defined queries), the keywords and associated weights, and the ranked documents (tweets) which contained keywords and their relative score to other tweets in the population of tweets. Behind the scenes are two additional features, the first being the process that periodically imports tweets and the second being query execution after the tweets are imported.

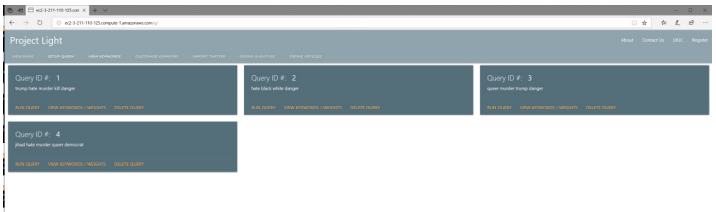
HOMEPAGE:

Displays the most recent runs of queries. Displayed information includes query Id, execution time, number of documents (tweets) containing a search term, the max score, average score and a full list of all terms used. Below is a screenshot from the AWS deployment. The view results displays the documents (tweets) that were discovered in this query run. The last character of the Query # is the order of the query.



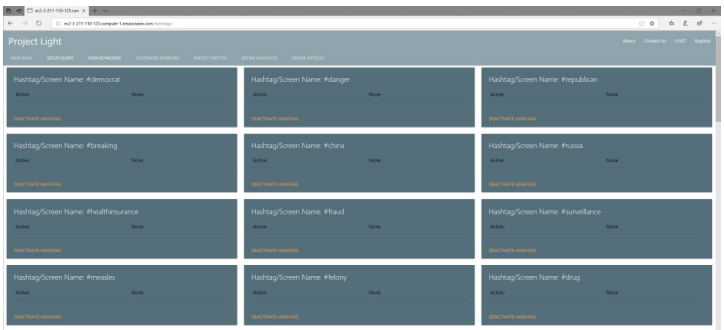
QUERY MODULE:

Displays a list of the currently configured and active queries. All active queries are executed when the query function is run.



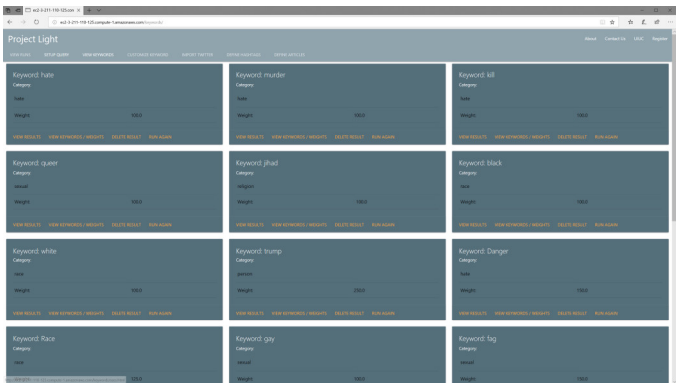
HASHTAGS MODULE USER DISPLAY:

Displays a list of the currently configured hashtags which will be used as search criteria for the next twitter importport.



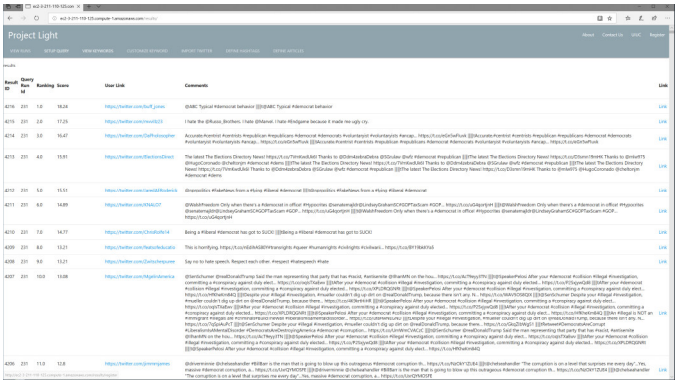
KEYWORD MODULE USER DISPLAY:

displays a list of the currently configured keywords and administrator defined weights, these values can be managed in the admin pages.



RESULTS DISPLAY:

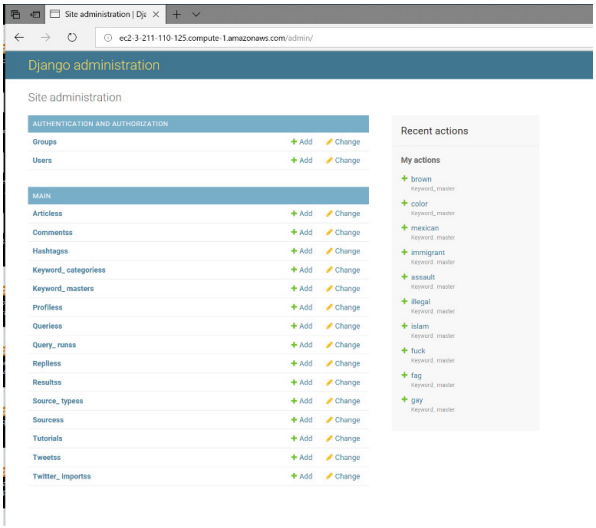
Displayed the results of ranked documents (tweets) related to the query. Ranking is relative to the other documents that matched the search criteria in the query after weights are applied. The calculated score is also displayed and a link to the screen name associated with the tweets. Important note is that the comments section is a consolidation of all tweets for this screen name. Each tweet is separated by a '|||' character.



ADMIN MODULE:

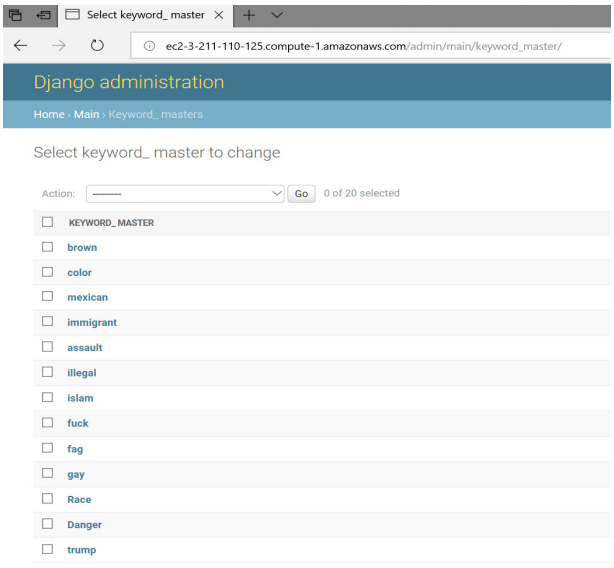
when a validated user logs into the system they have the ability to access the admin pages, which permit the authorized user to add / update / delete the following areas:

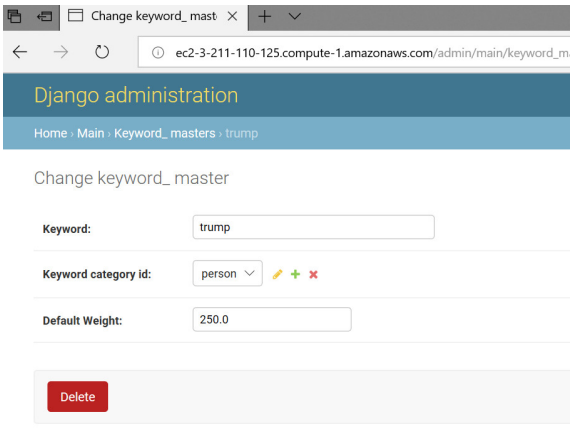
- Queries
- keywords / weights
- hashtags
- articles / screen names
- comments / (consolidated tweets)
- Tweets (raw imported tweets)



KEYWORD MASTER:

Screen allows crud operations on keywords.





VII. Outcome of Research

The team evaluated 3 different choices on data sources, SQLite, MongoDB, SQL Server and elected to go with SQLite for this project due to its integration with the web tool, Django. The team understands and accepts that SQLite will not produce the greatest performance, but Django is new to the team and we elected to focus on completing the project and a future effort to convert to a different data source may be undertaken in the Data Mining Capstone. The next phase of research is selecting a cloud service provider, preliminary options include Amazon Web Services (AWS), Microsoft Azure, Pythonanywhere, Digital Ocean. Each vendor has strengths and weaknesses, however detailed material on how to deploy Django/Python applications, while available, appears in many cases to be outdated or a version behind. After investigation and internal discussion the team elected to use Amazon Web Services to host the application. The documentation on deployment and management of the site was deemed the most straight-forward.

VIII. Technologies and Tools

The application consists of several technologies, outlined below:

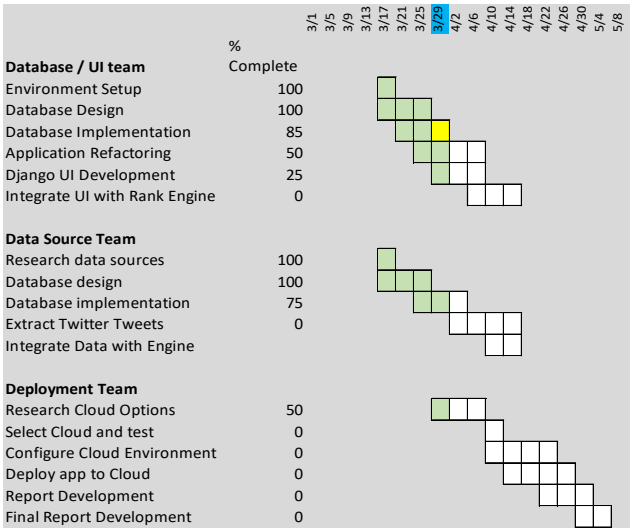
1) PYTHON – The core application is written in Python 3.7 using common editiors such as Pycharm and VS Code.
2) GITHUB – The project team is working in a private github repository, this repository is used for version contol, code, documentation, task management.
3) SQLite – This lightweight database tool is provided as part of the base Python environment. A 3rd party tools by Jetbrains, DataGrip is used to design and implement the database.
4) Django – This framework is currently being used to develop the web based user interface.
5) Tweepy – For managing twitter streaming API (https://developer.twitter.com/en/docs)
6) Twitter Search API – https://developer.twitter.com/en/docs/tweets/search/overview/standard

additional details (as referenced above):

Python 3.6.7	*Pre-installed with the ubuntu OS*
Venv	*Module provides support for 'virtual environments. Installed from CLI*
Django 2.2	*Python based open-source framework facilitating rapid deployment and high scalability*
Nginx	*High-performance HTTP server with rich feature set and simple configuration*
Gunicorn	*Python based web gateway interface which is compatible with web frameworks*
Supervisor	*Client/server system for per-process application management*
Elastic IP	*AWS feature*

IX. Project Schedule and Performance

The project was split into two concurrent work steams, each stream consisting of 2 team members. The first team was the application team focused on retrofitting the application and core database design. The second team is the data source team responsible for extracting and populating the corpus and article tables. The final phase of the project will be integration and deployment to the cloud and producing the final report. The project team dropped slightly behind on a few tasks due to unexpected complications with the refactoring of the application to pull data from a normalized database. Currently the team is working hard to meet the new schedule an is optimistic we will complete well before the final due date.



X. Evaluation of Remaining Work

The project is approximately 1/3 complete with a fairly significant amount of work remaining, including several areas which could be considered risks. The team is currently learning Django and making acceptable progress.

We have also started the development on the twitter scraping code-set and have also made significant progress in collecting a large corpus of tweets

The next few weeks the team will select a cloud provider and the Django interface should start to take shape. Once the team has successful deployed a working solution to a cloud it may look at refactoring the calculation engine to see additional improvements.

XI. Process Evaluation

Although the conversion from text-based data to SQLite database is still on-going the preliminary conversion is showing significant performance improvements. One of the more interesting experiences has been performing research on cloud provider options, as this space is moving quickly viewing the date of published articles is key to making a smooth transition.

XII. Evaluation/Results

The team was successful in deploying the application on the cloud environment. The application extracted tweets real-time

based on topics/keywords/queries defined by the user. The team set out to accomplish the following 5 goals, which produced the expected results, but not without unexpected complications. Recapping the objectives, challenges, and outcome.

1) Migrate the current solution to run as a cloud based service.

Outcome: Migration to the cloud was the last step in our process and unsurprisingly it was the smoothest component of the project. Sources of information on moving the application are abundant and generally straightforward to follow. We did encounter issues related to static files with Django, but were able to obtain assistance to help identify the cause. An important component is due to the dyanamic allocation of IP address is it necessary to configure the server with an Elastic IP. That service was provided via AWS. Performance of t2.micro instance was faster than expected and comparable to local machine performance. The Django site provided acceptable level of responsivness.

2) Modify the datastore away from file based JSON to a scalable data store solution. for example SQL Server, SQLite, MongoDB, Hadoop, etc. Update: *For this effort the team has selected SQLite*.

Outcome: Movement of the application from file based to a relational database, SQLite, was slightly more difficult than expected, but within a reasonable. Part of the difficulty was discovering the ORM components in Django limit the manner in which data is stored. Part of this may be to the lack of familiarity the team had with Django. When initial converted the performance of the application jumped a noticeable amount, approximately 20% faster. The SQL engine requires non-standard syntax in order to conduct some multi-table operations such as updates based on two other tables.

3) Procure a large corpus of tweets from 2019 for research and analysis. We expanded the scope of the data to collect 10 years or corpus of Donald Trump's tweets for analysis.

Outcome: Two efforts were conducted in order to analyze tweets. The first was to replace the article/comments structure in the original implementation of BM25 with screen names / tweets. The application implements a custom version of the

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot\left(1-b+b\cdot\frac{|D|}{\text{avgdl}}\right)},$$

BM25

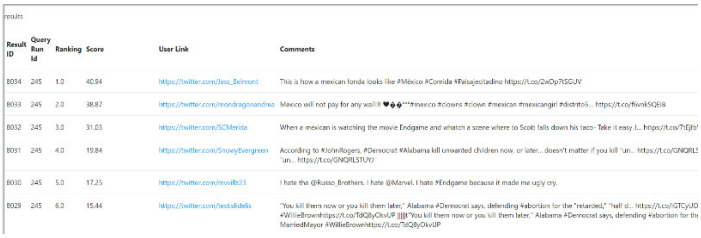
with an additional component to increase or decrease the value of a search term based on a pre-determined weight. A new feature added to support the extraction of tweets was the ability to feed the Tweepy import process a list of hashtags '#' and at @ terms to perform searches. The free developer account limited the ability to ingest so the team had to build a long running background process to pull in groups of tweets on a continuous basis, examples of hashtags include #democrat , "#danger" , #republican , #breaking , #china , #russia , #healthinsurance , #fraud , #surveillance , #measles , #felony , #drug , #mueller , #arrest , #criminal , #police , @CNN , #Foxnews , shooting, @realdonaldtrump, #terrorism, #fishing, #python, #UIUC, "#buildawall", "@OCSheriff". The search terms were then feed into the Tweepy python app and allowed to run over a period of 10 hours. These terms were pulled based on news events occurring at the time. Additional terms, not expected to generate hate speech were also selected for investigation. Some terms such as "fishing" generated a lot of angry tweets, but limited hate speech. In general terms used in queries were avoided to prevent bias in the search, but in a real-world deployment these terms would be included in the search criteria. As a second phase the team imported President Trump's tweets from 2009 to 2017 and performed an analysis of the sentiment changes occurring over that time.

4) Implement data scraping to analyze and score and rankdata from Twitter .

Outcome: Originally designed to rank documents, for this project we consolidated all tweets by screen name into a single corpus ranked those consolidated tweets against others tweeting in the same time frame. This resulted in ranking of screen names on tweeter based on the search terms and term weights. Review of the output demonstrates this function is working as expected. Currently the application is configured to re-run these queries on the population of tweets every day. A key discovery is the abundance of hashtag terms and at symbols reduces the effectiveness as those terms drive down the ranking result for some users. The BM25 algorithm includes a factor to consider the length of a comment. A single tweet with the world "hate" will rank high than a tweet with hate and half a dozen hashtags.

5) Publish analysis on the said data that reach a threshold defined as part of the query.

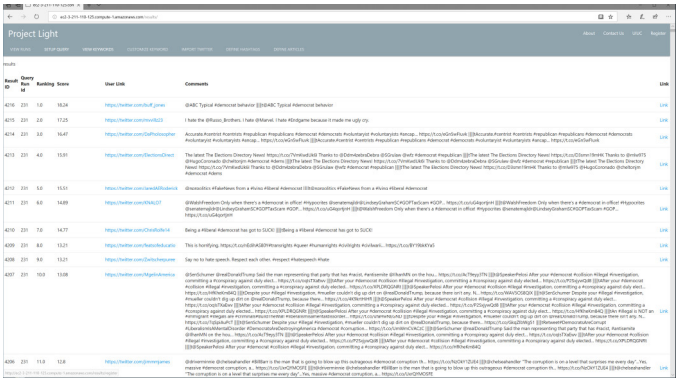
Outcome: The Django site includes a section to display the ranked results of a query, for example.



Result ID	Query Run Id	Ranking	Score	User Link	Comments
8034	245	1.0	40.94	https://twitter.com/Jess_Belmont	This is how a mexican fonda looks like #México #Corrida #Paisajecitadino https://t.co/2wOp7tSCUV
8033	245	2.0	38.87	https://twitter.com/mondragonandrea	Mexico will not pay for any wall!!! ♥️🙌🏽***#mexico #citizens #clove #mexican #mexicangirl #dilatrito5... https://t.co/fkvnkSQEI6
8032	245	3.0	31.03	https://twitter.com/SCMerida	When a mexican is watching the movie Endgame and whatch a scene where to Scott falls down his taco- Take it easy J... https://t.co/7tEjfz
8031	245	4.0	19.84	https://twitter.com/SnowyEvergreen	According to #JohnRogers, #Democrat #Alabama kill unwanted children now, or later... doesn't matter if you kill "un... https://t.co/GNQRLSTUYJ
8030	245	5.0	17.25	https://twitter.com/mvvilb23	I hate the @Russo_Brothers. I hate @Marvel. I hate #Endgame because it made me ugly cry.
8029	245	6.0	15.44	https://twitter.com/JoeJoBells	"You kill them now or you kill them later," Alabama #Democrat says, defending #abortion for the "retarded," "half d... https://t.co/GTCyUO #WillieBrownhttps://t.co/fdQBjyOkvUP [][][]"You kill them now or you kill them later," Alabama #Democrat says, defending #abortion for th MarriedMeyer #WillieBrownhttps://t.co/fdQBjyOkvUP

The volume of tweets and screen names makes this screen difficult to tie back to a specific query. A new or improved version of this interface should be developed to productionize this application.

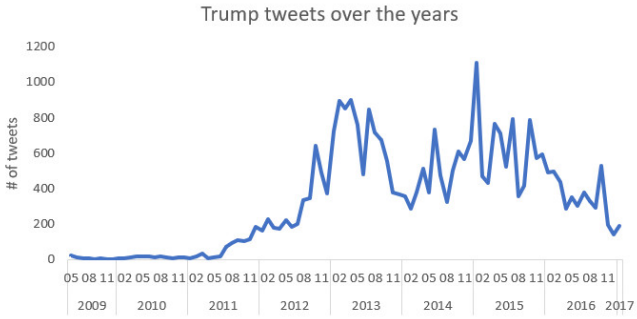
Sample results:



Evaluation of Donald trump's tweets:

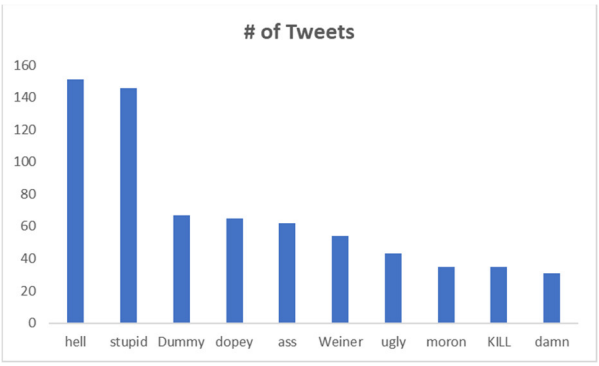
We leveraged the python scraping function to gather the tweets published by the incumbent American President (Donald Trump) for analysis. Details:

1. Data time period: April 2009 to Jan 2017
2. # of tweets: 30K

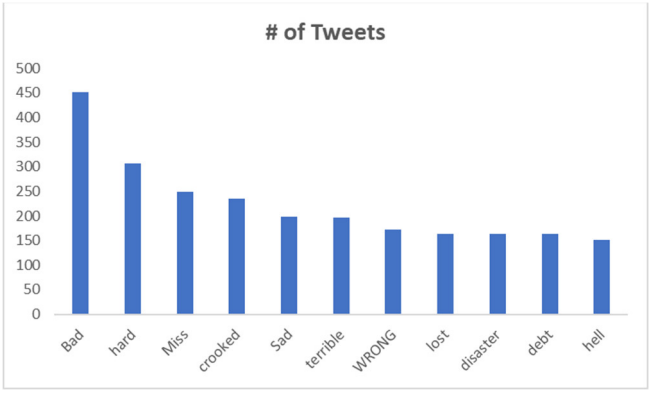


Using the deployed application, we analyzed the tweets to understand the sentiment and topics that Trump tweeted about.

1. Top conventionally bad keywords used in tweets:



2. Top words indicating negative sentiment:



XIII. DISCUSSION

In summary the project was rewarding and everyone on the team was stretched learning new skills and technologies. Key discoveries.

SQLite – This appeared to be a easy win for the team, but the volume of data proves that this is not the best tool for this type of application. The next version would be on a larger scale database in order to improve performance.

Django – For building a quick application this appears to be a good choice, we would not use it for a production version of the application, the framework based on our experience drives the design more than it should. As my Department is interested in using this for articles / comments we will be using our in-house application framework. Django may provide more features than we are aware.

Twitter Data – The use of Twitter data was an interesting use for the tool, given that as mentioned above it does not produce results as reliable as when used against articles and comments.

Accuracy – The users that ranked high scores certainly produced tweets that matched the query terms, however we have come to the conclusion that a sentiment analysis tool should also be incorporated into this application.

This tweet ranked 9th, from a pure search terms we can see the cause, but the purpose of the tweet it should not be considered hate speech by the application.

XIV. Related Work

This project delved deeper into using custom algorithms to detect/identify hate speech on twitter. The methodology adopted treated each tweet as an independent event and scored it based on the customer algorithm developed by the team

1. Incorporating user information (like profile, demographics, historical posts etc.) is likely to prove useful for detecting hate speech. There are studies of the people that post hateful content online, including characteristics and behavioral traits that are typical of the authors behind aggressive behavior. Example: https://dl.acm.org/citation.cfm?id=2411739
2. There have been several studies which have explored the role of anonymity in increasing the likelihood of hateful speech. This paper https://arxiv.org/pdf/1610.08914.pdf posited that hateful speech clusters in time – i.e. one hateful comment follows another.
3. Another useful avenue of research has revolved around using network-based features (ex. # of followers, network depth etc.) for classifying aggressive and hateful behavior.

   https://dl.acm.org/citation.cfm?id=3091487

XV. Conclusion

Detecting harmful and hateful speech in online behavior at scale has proved to be a challenging endeavor for several organizations. Our work through this project has aimed to build and deploy capabilities which can be leveraged for identifying such behavior

This paper investigated deploying an application which leverages a custom version of the BM-25 algorithm to detect hate speech by recognizing offensive words and phrases in tweets based on pre-defined queries set by the user.

The team was able to successful migrate the desktop application to the cloud infrastructure and moving the database to SQLite. When initial converted the performance of the application jumped a noticeable amount, approximately 20% faster. We also developed capabilities to collect and collate tweets. The tool can now consolidate all tweets by screen name into a single corpus ranked those consolidated tweets against others tweeting in the same time frame. This resulted in ranking of screen names on tweeter based on the search terms and term weights.

We expanded the scope of our project to research deeper into the tweeting patterns of @RealDonaldTrump and uncovered some interesting insights on the common parlance used by the tweeter.

There are several opportunities to enhance the breadth of capabilities of this tool. A great amount of information related to the users of Twitter can potentially be retrieved or derived from user behavior. Example: network/follows, communication with other tweeters, group behavior patterns etc. We would also emphasize here that there may be opportunities to build additional capabilities to detect and separate the false positives. There may be cases where what we're classifying as hate-speech in one context may not actually be one in another.

XVI. Division of Work

	Owner
Database / UI team	
Environment Setup	
Database Design	**David Jamriska**
Database Implementation	**Manojit Saha**
Application Refactoring	
Django UI Development	
Integrate UI with Rank Engine	
Data Source Team	
Research data sources	**David Jamriska**
Database design	**Ganesh Iyer**
Database implementation	**Ayan Putatunda**
Extract Twitter Tweets	
Integrate Data with Engine	
Deployment Team	
Research Cloud Options	**David Jamriska**
Select Cloud and test	**Ganesh Iyer**
Configure Cloud Environment	**Ayan Putatunda**
Deploy app to Cloud	**Manojit Saha**
Report Development	
Final Report Development	

XV. Appendix - Installation Recap

1. Set up an Ubuntu system under t2.micro free-tier infrastructure. Here we also provisioned the elastic IP which we will be used as a hostname for our configurations

2. Configure AWS security groups to allow incoming connections from anywhere on (0.0.0.0/0) TCP ports 80 (HTTP), 8000 & 22 (SSH)

3. Download and import the public key into our ssh terminals for remote shell access.

4. Login in to a shell and install the necessary components/ packages highlighted above

5. Spawn and activate a python virtual environment inside our EC2 system.

6. Host our local code base in Github and pulled in our repository in EC2 using git clone.

7. Bind gunicorn to wsgi.py file present in our application folder.

8. Edit the supervisor configuration file for gunicorn by informing supervisor daemon which commands to invoke in case of OS bootup or auto-process restarts following a failure.

9. Configure NGINX to read from the unix socket and tells NIGINX from where to read and write the web requests/ responses appropriately

10. Have supervisor re-read the fresh setup, startup gunicorn interface in background and then restart NIGNX service to source the updated.

11. Our Django based application should now be accessible from browser. We have tested it successfully from Google Chrome, Internet Explorer & Safari

TROUBLESHOOTING

Like every project we had some learning opportunities during deployment which have been listed below alongside their solution for future references:

After initial deployment when we tried to access the cloud app it came back with disallowed host error:

DisallowedHost at /

Invalid HTTP_HOST header: 'XXX.XXX.XXX.XXX:8000'. You may need to add u'XXX.XXX.XXX.XXX' to ALLOWED_HOSTS.

To fix this issue we needed to update our application settings.py file and add AWS EC2 hostname to the list of Allowed host. A generic but less secure solution will be to include all IP addresses:

ALLOWED_HOSTS = ['XX.XX.XX.XX']

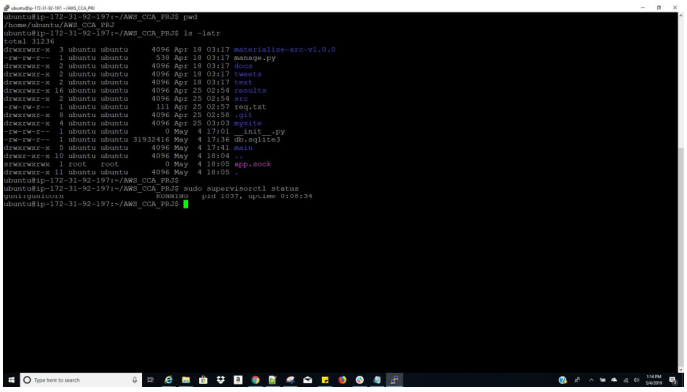
Cloud application wasn't able to locate the inbuilt sqllite DB when we were browsing across the tabs:

Django Version: 2.1.4 Exception Type: OperationalError Exception Value: no such table: main.auth_user Exception Location

Upon review we noted that the git clone didn't pull in our sqllite db file to our EC2 project folder. We had to an additional git pull to get that file added

Cloud Django app wasn't able to find the static CSS files for the admin portal. These needed an additional configuration in our settings.py to add conditions for static file handlers:

Final screenshot of supervisor service running.



References

[1] Wikipedia contributors. "Okapi BM25." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 20 Feb. 2019. Web.

[2] Jamriska, D., Putatunda, A., Chen, C., "Project Proposal: Hate Speech Detector, CS410 Fall 2018. University of Illinois Urbana Champaign, Urbana, Illinois. 2018, pp. 1–2.