Twitter Information Operations Search Engine

Nishanth Nakshatri*
nzn5185@psu.edu
Department of Computer Science and Engineering, Pennsylvania State University
State College, Pennsylvania

Abstract

Search Engines have gained a lot of traction over the last two decades. It acts as a filter for a wealth of available information on the Internet. In this work, we introduce a specialty search engine built on the twitter information operations dataset. We hope to see this tool to be used by researchers, especially people from political science and international affairs background, who are interested in an in-depth understanding of the illicit influential campaigns run on twitter.

Keywords: search engine, information retrieval, google custom search engine, indexing, elasticsearch

ACM Reference Format:
Nishanth Nakshatri. . Twitter Information Operations Search Engine. In . IST 441, Penn State, PA, USA.

1 Introduction

The average person living in a modern industrialized society is exposed to several pieces of information every single day. This information would equal the amount of information that a person living 100 years ago would have experienced in a year. The information is displayed as advertisements, newspaper headlines, websites, text messages, traffic signs, T-shirt slogan, and so forth. With this amount of information that we sift through on a daily basis, it is virtually impossible to remember everything that we need to know, i.e, names, dates, figures, phone numbers, email address and many more. Most of us need tools that remember our information and will retrieve it on demand. For this purpose, we need a search engine.

A Search Engine acts as a filter for a wealth of available information on the Internet. Search Engines allow users to not only quickly, but also easily, find the information that is of interest or value to them. It also eliminates the need to wade through numerous amounts of irrelevant web pages. It is safe to say that the goal of a Search Engine is to provide the user with search results that lead to relevant information on high quality websites. The ideal word here being "relevant". In order for a Search Engine to attain and retain a market, they need to make sure to deliver results that are relevant to what the user's search term.

In this work, we have focused our efforts in building a search engine specific to twitter information operations dataset [3]. Users will be able to search through comprehensive archive of Tweets associated with known state-backed information operations on Twitter. We believe that people and organizations with the advantages of institutional power and which consciously abuse our service are not advancing healthy discourse but are actively working to undermine it. With this search engine accessible to everyone, we seek to empower researchers, journalists, governments, and members of the public to deepen their understanding of critical issues impacting the integrity of public conversation online, particularly around elections.

2 Dataset

In October 2018, Twitter disclosed the first comprehensive archive of state-backed information operations on Twitter. It was launched with an initiative to empower academic and public understanding of these coordinated campaigns around the world, and to empower independent, third-party scrutiny of these tactics on Twitter's platform.

Twitter is actively working with the industry peers about possible malicious activity on their service about any attempted influence campaigns located in any part of the world. In collaboration with the peers, they investigated further to build an understanding of these networks. Law enforcement and the general public were immediately notified as soon as any malicious activities were discovered as the platform manipulation is a violation of the Twitter Rules.

Twitter subsequently added new datasets in January, June and August, while also sharing insights on their internal investigative approach and how these complex, sometimes cross-jurisdictional operations were identified.

The archive, as of now, is the largest of its kind in the industry. Thousands of researchers have made use of these datasets that contain millions of individual Tweets and more than one terabyte of media. Using this archive, the researchers have conducted their own investigations and shared their insights and independent analyses with the world. Some of the prominent work associated with this archive are [2], [1].

*

IST 441, Information Retrieval Course Project, PSU, Spring'20

In this work, we have considered archives related to Iran, Venezuela and Russia. These archives were released in October 2018, January 2019 and June 2019. We have to note that Twitter attribution is not 100% accurate. So researchers are expected to use them out of an abundance of caution.

2.1 Iran

Twitter managed to remove a few set of accounts that originated in Iran. These accounts were found to be associated with or directly backed by the Iranian government. Twitter was able to find specific traits pertaining to each account. For instance, some of these accounts tweeted nearly 2 million times. They tweeted global news content, often with an angle that benefited the diplomatic and geo-strategic views of the Iranian state. Further, some accounts directly engaged with discussions related to Israel specifically. A few set of accounts employed a range of false personas to target conversations about political and social issues in Iran and globally.

2.2 Russia

Twitter investigations and an increased information sharing between industry peers and law enforcement helped them to remove a few IRA-specific accounts around the 2016 US Presidential Election. Through ongoing analyses and investigations, twitter is working towards contextual understanding of these networks of accounts to improve the ability to find and suspend the activities as quickly as possible in the future. This is particularly vital as groups such as the IRA evolve their practices in response to industry-wide suspension efforts.

2.3 Venezuela

Twitter removed some accounts located in Venezuela. These accounts were another example of a foreign campaign of spammy content focused on divisive political themes, and the behavior was similar to that utilized by potential Russian IRA accounts. A few other accounts located in Venezuela appeared to be engaged in a state-backed influence campaign targeting domestic audiences were also removed by Twitter.

3 Search Engine

This section discusses about the various components involved in developing the search engine. Figure 1 shows a pictorial representation of the components involved in building our specialty search engine.

3.1 Raw Data & Pre-processing

As mentioned earlier, dataset was obtained from twitter pertaining to countries - Iran, Venezuela and Russia. We have considered a total of 7,539 user accounts with 19,510,805 million full tweet-texts. The tweets were comprised of 70 different languages. The size of the entire dataset was approximately 11.5 GB. Table 1 shows the schema associated with the raw dataset.

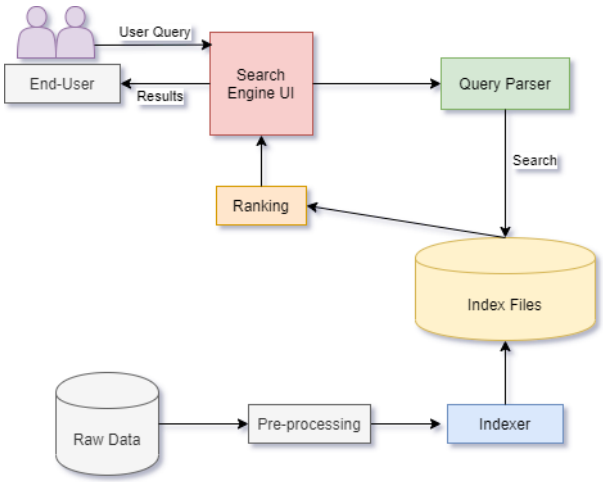


Figure 1. Shows the different components used in building the search engine

We pre-processed the dataset before passing it to the indexer. During pre-processing the raw data, we handled the data-type issues and the null-value issues. Further, we did not consider the latitude, longitude columns associated with the data. The values associated with these columns were *ABSENT* or *PRESENT*. This was not of much use to the end-user. Therefore, we neglected these columns during the pre-processing phase.

Further, we did not perform any form of text analysis or language translation on the tweet text. This was done to retain the tweet emotions and the sentiment in the corresponding language and allow the researchers to be able to build on top of the raw dataset.

3.2 Indexer

Indexing is the process by which search engines organise information before a search to enable super-fast responses to queries. We used ElasticSearch in the backend. Elasticsearch is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents.

Elasticsearch is able to achieve fast search responses because, instead of searching the text directly, it searches an index instead. This is like retrieving pages in a book related to a keyword by scanning the index at the back of a book, as opposed to searching every word of every page of the book. In other words, it uses *inverted index*. An inverted index is a system wherein a database of text elements is compiled along with pointers to the documents which contain those

Columns	Description
tweetid	tweet identification number
userid	user identification number (anonymized for users which had fewer than 5,000 followers at the time of suspension)
user_display_name	the name of the user (same as userid for anonymized users)
user_screen_name	the Twitter handle of the user (same as userid for anonymized users)
user_reported_location	the user's self
user_profile_description	the user's profile description (*)
user_profile_url	the user's profile URL (*)
follower_count	the number of accounts following the user (*)
following_count	the number of accounts followed by the user (*)
account_creation_date	date of user account creation
account_language	the language of the account, as chosen by the user
tweet_language	the language of the tweet
tweet_text	the text of the tweet (mentions of anonymized accounts have been replaced with anonymized userid)
tweet_time	the time when the tweet was published (UTC)
tweet_client_name	the name of the client app used to publish the tweet
in_reply_to_tweetid	the tweetid of the original tweet that this tweet is in reply to (for replies only)
in_reply_to_userid	the userid of the original tweet that this tweet is in reply to (for replies only)
quoted_tweet_tweetid	the tweetid of the original tweet that this tweet is quoting (for quotes only)
is_retweet	True/False, is this tweet a retweet
retweet_userid	for retweets, the userid who authored the original tweet
retweet_tweetid	for retweets, the tweetid of the original tweet
latitude	geo (Absent or Present)
longitude	geo (Absent or Present)
quote_count	the number of tweets quoting this tweet
reply_count	the number of tweets replying to this tweet
like_count	the number of likes that this tweet received (^)
retweet_count	the number of retweets that this tweet received (^)
hashtags	a list of hashtags used in this tweet
urls	a list of urls used in this tweet
user_mentions	a list of userids who are mentioned in this tweet (includes anonymized userids)
poll_choices	if a tweet included a poll, this field displays the poll choices separated by |

Table 1. Shows the data schema associated with twitter information operations dataset.

elements. Then, search engines use a process called *tokenisation* to reduce words to their core meaning, thus reducing the amount of resources needed to store and retrieve data. This is a much faster approach than listing all known documents against all relevant keywords and characters.

In Elasticsearch, a document is the unit of search and index. An index consists of one or more documents, and a document consists of one or more fields. In database terminology, a document corresponds to a table row, and a field corresponds to a table column.

We have used the default tokenizer associated with ElasticSearch. It was able to tokenize the tweet-texts related to almost all the languages. This gives the search engine the ability to have multi-linguistic query terms. End users can search in Arabic, Russian, Persian, Spanish etc. in addition to English. We built **twitter-index** on the entire pre-processed dataset which includes all the data attributes mentioned in Table 1 except *latitude* and *longitude*.

Python's ElasticSearch API, specifically Bulk API was used in conjunction with python's helpers module to build the index. Bulk provides a way to perform multiple index, create, delete, and update actions in a single request.

3.3 IndexFiles

We know that Elasticsearch cluster can contain multiple Indices (databases), which in turn contain multiple Types (tables). These types hold multiple Documents (rows), and

each document has Properties(columns). In our case, we had one index, **twitter-index**, with approximately 19 million documents. These represent the indexed files.

3.4 Ranking

This section provides a detailed understanding of how the ranking scores are determined in Elasticsearch.

Elasticsearch first reduces the candidate documents down by applying a boolean test of whether the document match the query. Once the results that match are retrieved, the score they receive will determine the rank ordered for relevancy.

The scoring of a document is determined based on the field matches from the query specified and any additional configurations applied to the search.

Elasticsearch runs Lucene under the hood so by default it uses Lucene's **Practical Scoring Function**. This is a similarity model based on Term Frequency (tf) and Inverse Document Frequency (idf) that also uses the Vector Space Model (vsm) for multi-term queries.

We have used the default **Practical Scoring Function** offered by Lucene for the search engine. The formula used to calculate the score is as follows,
Let

- q denote a query.
- d represent a document.
- tf(t in d) be the term frequency for term t in document d.
  $tf(\text{t in d}) = \sqrt{(termFreq)}$
- idf(t) is the inverse document frequency for term t.
  $idf(t) = 1 + \ln \frac{maxDocs}{(docFreq+1)}$
- queryNorm(q) is the query normalization factor. It is the sum of squared weights for the terms in the query. It is used so that different queries can be compared.
- coord(q,d) is the coordination factor that counts the number of terms from the query that appear in the document
- t.getBoost() is the boost that has been applied to the query. It is a percentage or absolute number that can be used to boost any query clause at query time.
- norm(t,d) is the field-length norm, combined with the index-time field-level boost, if any. In other words, it is the inverse square root of the number of terms in the field. $norm(t,d) = \frac{1}{\sqrt{(numFieldTerms)}}$

Then, the ranking function can be written as,
$score(q,d) = queryNorm(q) * coord(q,d) *$
$SUM(tf(\text{t in d}), idf(t)^2, t.getBoost(), norm(t,d))(\text{t in q})$

3.5 Query Parser

We have leveraged Lucene's query parser for the search engine. It parses the documents based on a provided query string, using a parser with a strict syntax.

The query string "mini-language" is used by the Query string and by the q *query string* parameter in the *search API*. The query string is parsed into a series of terms and operators. A term can be a single word *quick* or *brown* or a phrase, surrounded by double quotes, *"quick brown"* which searches for all the words in the phrase, in the same order. We can use operators such as AND, OR that allow us to customize the search.

Query Parser uses a syntax to parse and split the provided query string based on operators, such as AND or NOT. The query then analyzes each split text independently before returning matching documents.

We have to note that default query parser provided by Elasticsearch is capable of parsing most of the languages and hence, providing the ability to end-users to type the query in any language of their choice.

3.6 Search Engine User-Interface

This section provides a detailed overview of the tools used to build the user-interface for the search engine.

The UI was built using Django in the backend. This is because Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. It takes care of much of the hassle of Web development, so we can focus on writing our search engine's crucial components without needing to reinvent the wheel. Also, it is free and open source.

We have extensively leveraged Django's concept of "views" to encapsulate the logic responsible for processing a user's request and for returning the response. Django's template layer helped us provide a designer-friendly syntax for rendering the information to be presented to the end-user.

Figure 2 shows the home page for twitter information operations search engine. We provide a simple form that takes a mandatory query parameter that user wants to search for. It takes start tweet time, end tweet time and specific page number that user wants to navigate. However, these parameters are not mandatory from the user. The user is given two options after filling up the form. The user either can view the results on the screen by submitting the query or can download the resutls in a csv format.

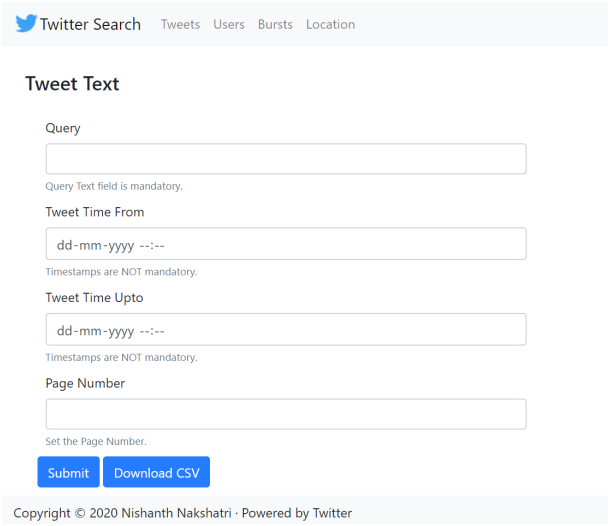


Figure 2. Shows the home page for twitter search engine on information operations dataset.

We have incorporated Bootstrap's styling components. It is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components. It was originally named *Twitter Blueprint*, developed by Mark Otto and Jacob Thornton at Twitter,m as a framework to encourage consistency across internal tools.

4 Custom APIs

We have built four custom APIs to let the user understand the data from different perspectives. This section provides a detailed overview of all the perspectives and shows the user-interface associated with the webpage.

4.1 Tweet Text Search API

The intent of this API is to allow the users' to search through the tweet text associated with the information operations data. It searches the whole of tweet-text to find the relevant tweets to the user-query.

The APIs were built with Python's Django framework. After the user enters the search query, based on the action chosen by the user, SUBMIT or DOWNLOAD CSV, corresponding action is chosen in the backend. Figure 3 shows the primary query associated with the API that delivers the results back to the template layer.

Elasticsearch's **search** API is used to obtain the results. These results are passed on to the template layer, which displays them in a user-friendly manner. We have tried to display the tweets similar to Twitter webpage results. Figure 4 shows the results obtained for the search term **"Obama"**. We can see that each tweet follows a similar design as that

```
body = {
    "from": start,
    "size": size,
    "query": {
        "bool": {
            "must": [
                {
                    "match": {
                        "tweet_text": query_text
                    }
                }
            ],
            "filter": {
                "range": {
                    "tweet_time": {
                        "gte": lower_time,
                        "lte": upper_time
                    }
                }
            }
        }
    }
}
```

Figure 3. Shows the query associated with ***tweet text search API***. Displays top 20 tweet results for a page. *query_text*, *lower_time* and *upper_time* are user query specific parameters.

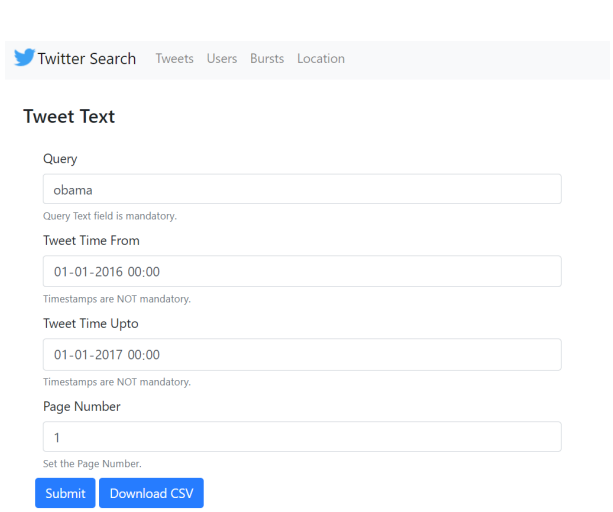
of Twitter. Header of each card/tweet contains the details such as user-name, tweet time, retweet or not, original author (if it is a retweet). The content of the cards contain the tweet text, user mentions, like count, reply count and quote count. We have made user names displayed for each tweet to be clickable. This makes a call to **User Details API** in the backend and displays all the information related to the user.

4.2 User Details API

The intent of this API is to allow the end-users to obtain a holistic information about the users present in the information operations dataset. This API provides details such as user name, user reported location, user profile description, user follower count, user following count, all the hashtags and URLs used in the tweets, count of all the tweets in a specified timeframe. This information not only gives a holistic view of the user but also, can be leverged to assert the user's legitimacy. We could deduce if the user is genuine or a bot.

We use Elasticsearch's **sql** API to obtain the aggregated user related information in conjunction with a DSL query to obtain all the URLs and hashtags related to the user. Figure 5 shows the body of queries used to obtain the results.

The end-users are allowed to search the user of choice based on userid. Figure 6 shows the results obtained for



(a) Query

(b) Results - set1

(c) Results - set2

(d) Results - set3

Figure 4. Shows the results obtained from the tweet search API for the search term **"Obama"** with lower timestamp to be the First of Jan 2016 and upper timestamp to be the First of Jan 2017. We can see that results obtained are relevant to the query terms.

query term, userid **2361340122**. We can also see that results obtained are relevant to the query term.

4.3 Tweet Bursts API

The intent of this API is to allow the end-users to find all the users who have tweeted more than x number of tweets in a specified timeframe. This API is particularly useful to identify all the bots in the dataset. For instance, if the end-user believes that a legitimate user won't tweet more than a hundred tweets per day, then the end-user can search all the users with this threshold value for tweet burst. In addition, the end-users are given an option to search the users pertaining to specific location. This is an optional parameter. Figure 7 shows the query associated with the tweet burst API. We have used SQL API to obtain the aggregated results for tweet bursts.

Figure 8 shows the results obtained for a user query. We can see that the results obtained are relevant to the search term.

4.4 Location Search API

This API is intended to find the total tweet count and total user count pertaining to a specific location of choice. Figure 9 shows the query associated with the location search API.

```
body1 = {
    "query": "SELECT userid, user_display_name, user_reported_location, user_profile_description
account_creation_date, MAX(follower_count) AS follower_count, "
             "MAX(following_count) AS following_count, MIN(tweet_time) as first_tweet_time, "
             "MAX(tweet_time) as last_tweet_time, count(tweetid) as total_tweets "
             "FROM twitter_index "
             "WHERE (tweet_time > CAST(" + "'%s'" % lower_time + " AS DATETIME) and "
                                                                 "tweet_time < CAST(" + "'%s'" %
upper_time + " AS DATETIME)) AND userid LIKE " +
             "'%s'" % query_text +
             "GROUP BY userid, user_display_name, user_reported_location, user_profile_description,
account_creation_date"
}
```

(a) Query - SQL API to obtain aggregated user information

```
body2 = {
    "from": start,
    "size": 9999,
    "query":
        {
            "bool": {
                "must": [
                    {"match": {
                        "userid": query_text
                    }}
                ],
                "filter": {
                    "range": {
                        "tweet_time": {
                            "gte": lower_time,
                            "lte": upper_time
                        }
                    }
                }
            }
        },
    'highlight': {'fields': {'body': {}}}
}
```

(b) DSL query to obtain URLs and Hashtags

Figure 5. Shows the query used to obtain the user details.

We have used bucket aggregations and cardinality to obtain the results.

Figure 10 shows the query and the results obtained for the corresponding query. We can see that relevant results are obtained for the query.

5 Google Custom Search Engine

Google Custom Search Engine is irrelevant for our project. Although we can build a google custom search engine by providing the base url to be **"twitter.com"**, this will still be invalid due to the fact that all the users associated with the archive are removed by Twitter. We are dealing with users who are non-existent from the twitter network. Therefore, we cannot compare this search engine with Google Custom Search Engine.

Further, we have built a niche search engine specifically related to the information operations dataset. Thus far, there are no search engines built on top of this dataset. Hence, there are no competitions associated with our search engine.

6 Evaluation of the Results

Figures 4, 6, 8 and 10 shows the query information and the obtained results. We can see that relevant results are obtained in all the cases.

However, all the queries were in English. In this section, we provide the results obtained for Arabic and Russian search terms. Figure 11 shows the query and results obtained for other languages. We can see that results obtained in this case are also relevant to the search term or query term.

In addition to the queries presented here and during the presentation, the search engine was rigorously tested against 50 different queries on all the APIs and it produced relevant results. **The entire system was built on localhost. Temporarily, this has been made available for a subset of dataset at http://3.133.83.16:8080/home/.** The site currently supports approximately 2 GB of data. This is primarily because Amazon EC2 instance allows only less amount of storage for free tier usage. The entire codebase has been shared to test the code on localhost.

7 Conclusion

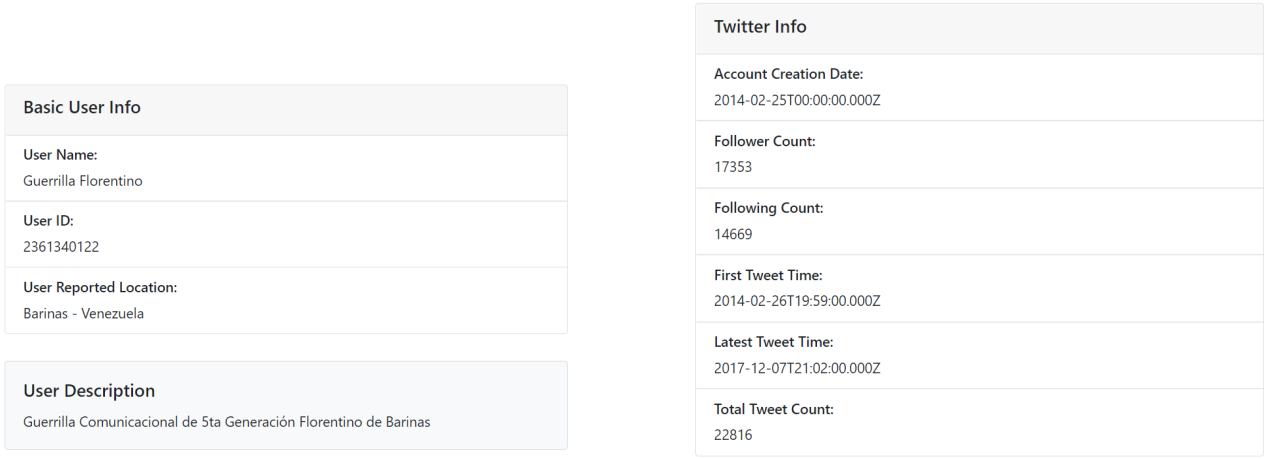
We have seen the building blocks and working of a niche search engine built on top of twitter information operations dataset. We strongly believe that the tool can be very handy for researchers interested in understanding more about the tweets associated with state-backed information operations on Twitter.

References

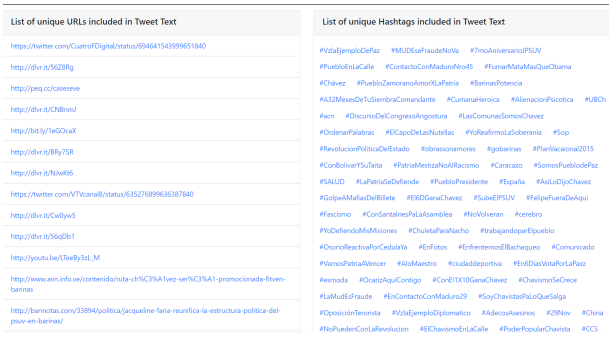
[1] Christopher Griffin and Brady Bickel. [n.d.]. *Unsupervised Machine Learning of Open Source Russian Twitter Data Reveals Global Scope and Operational Characteristics.*

[2] et al. Rajtmajer, Sarah. [n.d.]. *A Dynamical Systems Perspective Reveals Coordination in Russian Twitter Operations.*

[3] Twitter. 2018. *Twitter Information Operations Dataset.* https://transparency.twitter.com/en/information-operations.html



(a) Basic User Info (b) Aggregated User Info

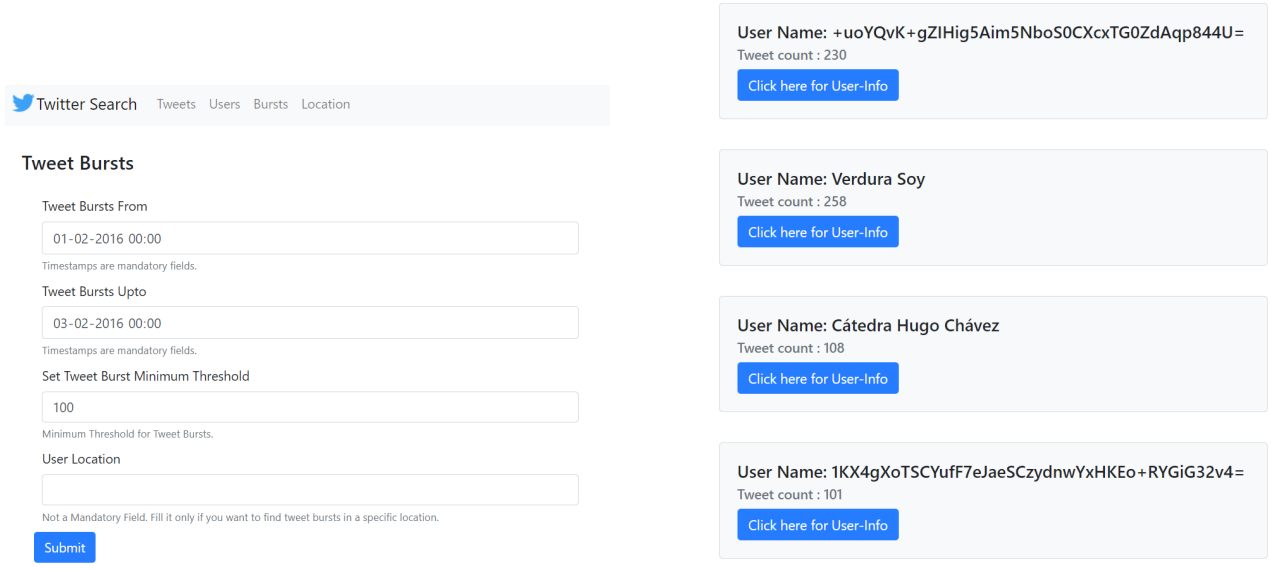


(c) List of URLs and Hashtags

Figure 6. Shows the results obtained from the user details API for the search term **"2361340122"** with lower timestamp and upper timestamp to be blank. We can see that we have obtained the relevant user information.

```
    body = {
        "query": "SELECT userid, user_display_name, count(userid) as total_tweets FROM twitter_index "
                 "WHERE user_reported_location LIKE " + "'%s'" % user_location + " AND  "
                                                                                 "(tweet_time > CAST(" +
"'%s'" % lower_time + "  AS DATETIME) AND "
                 "tweet_time < CAST(" + "'%s'" % upper_time + "  AS DATETIME)) "
                                                              "GROUP BY userid,user_display_name HAVING
total_tweets > " + str(
            tweet_count)
    }
```

Figure 7. Shows the query associated with tweet burst API.

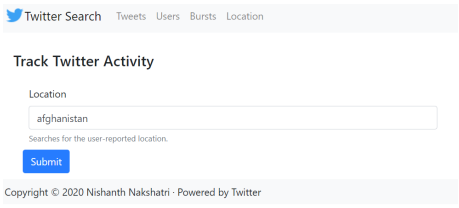


(a) Query (b) Tweet Burst Results

Figure 8. Shows the query and results obtained for tweet bursts. We can see that the obtained results are relevant to the query. The tweet counts are greater than the threshold value 100.

```
body = {
    "size": 0,
    "query": {
        "regexp": {
            "user_reported_location": {
                "value": ".*"+query_location+".*",
                "flags": "ALL"
            }
        }
    },
    "aggs": {
        "total_user_count": {
            "terms": {
                "field": "user_reported_location.keyword",
                "size": 500,
                "order": {
                    "tweet_count": "desc"
                }
            },
            "aggs": {
                "user_count": {
                    "cardinality": {
                        "field": "userid.keyword",
                        "precision_threshold": 100
                    }
                },
                "tweet_count": {
                    "cardinality": {
                        "field": "tweetid",
                        "precision_threshold": 100
                    }
                }
            }
        }
    }
}
```

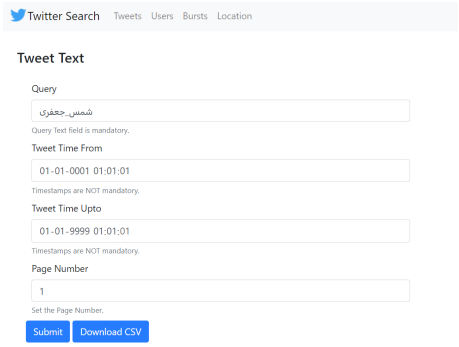
Figure 9. Shows the query associated with location search API.



(a) Query

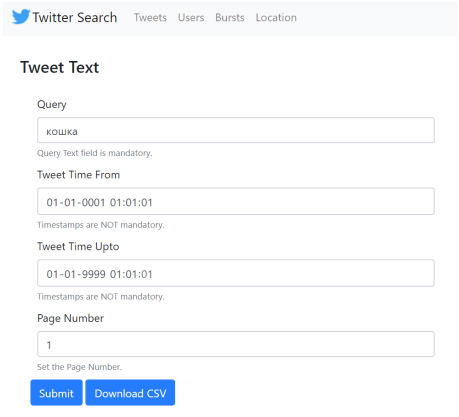
(b) Location API Results

Figure 10. Shows the query and results obtained from location search API. The results are shown for query term *afghanistan*. We can see that results are relevant to the search term.

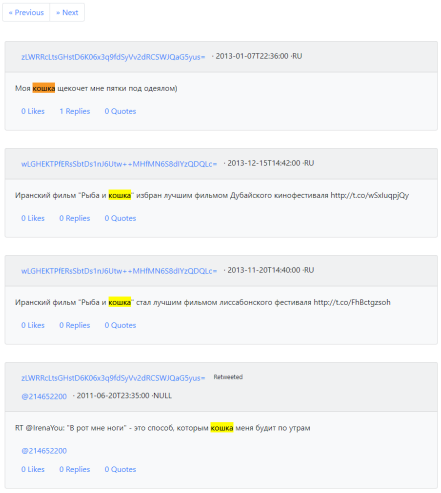


(a) Query - Arabic

(b) Results - Arabic Query



(c) Query - Russian



(d) Results - Russian Query

Figure 11. Shows the results obtained from the tweet search API for the search terms shown in figure (a) and figure (c). We can see that obtained results are relevant even in this case.