

Report for BigData Project:

Real time Twitter sentiment analysis system based on user-specified topic

Anlan He (he000193@umn.edu)
Ruoyan Kong (kong0135@umn.edu)
William Batu (batux002@umn.edu)
Yuanli Wang (wang8662@umn.edu)

0. Table of Contents

Proposal for BigData Project:	1
Real time Twitter sentiment analysis system based on user-specified topic	1
0. Table of Contents	2
1. Overview	3
1.1 Problem Formulation	3
1.2 System Architecture	3
1.3 Data Source	4
1.4 NoSQL Storage Technology	5
2. Analyzing Data	5
2.1 Data Properties	5
Crawl Related Data	5
2.2 Analytical Questions	6
2.2.1 Easy Questions	6
2.2.2 Difficult Questions	9
2.2.3 Challenging Questions	9
2.3 Data analysis	10
Split Data into Different topics	10
Query and Analyze Data	11
3. Potential Issues and Challenges	14
4. Lessons learned	14
5. References	14
6. Appendix	15
Crawl and Split Data	15
Sentiment Analysis	16
Compute Statistics	16
Sample Data	16
Druid JSON Ingestion Spec	20

1. Overview

1.1 Problem Formulation

Over the last few years, there is a boom in the growth of microblogging's popularity all over the world.

- Hundred millions of users are engaging in microblogging platforms.
- When it comes to breaking news or any heated topics, Twitter becomes a preferred platform for public discussion.
- Due to the rapid growth and the enormous message traffic, people from various industries are increasingly seeking ways to mine Twitter for potential information about what the public think and feel about specific issues.
- Along with the rise of sentiment analysis being applied to both formal articles and informal language, it is reasonable to consider sentiment analysis as a potential powerful tool to extract information from contents of microblogging.

While there has been a fair amount of work about opinion finding in Twitter for brand improvement or financial trend prediction, the choice of conducting sentiment analysis on which topic is in the hand of the system designer. In other words, people are more used to being exposed to the results of sentiment analysis, rather than directly interact with the analysis system. We believe there is a need for a more flexible real-time sentiment analysis system based on user-specified keywords, where clients could look for current public attitudes towards any topic that they are interested in, by entering a group of keywords related to that topic. In this project, we begin to investigate the possibility of such conception.

Following are the steps of this project:

1. Clients send requests to the analysis system towards any specific topics. In particular, clients could request for multiple topics at the same time, whether related or not. Say clients enter keywords "Joker Movie" and "iPhone 11 pro" since they are interested in the opinions toward both the topics.
2. The system will grab streaming data from Tweets to perform topic partition and sentiment analysis.
3. When the results come out, clients will find the results in either a simple opinion classification form: the proportion of positive, neutral and negative, or the results with more detailed information by conducting some NLP analysis, for example a list of the high-frequency related words with "Joker Movie" are "The laugh's on us", "Oscar", "Gun violence". Clients could decide which form they want the analysis results to be displayed in.
4. The points that matter here are this real-time sentiment analysis system could work based on multiple user-specified topics, and is expected to perform the topic partition and sentiment analysis in a relatively short time.

1.2 System Architecture

In Figure 1 we show our real-time sentiment analysis system architecture.

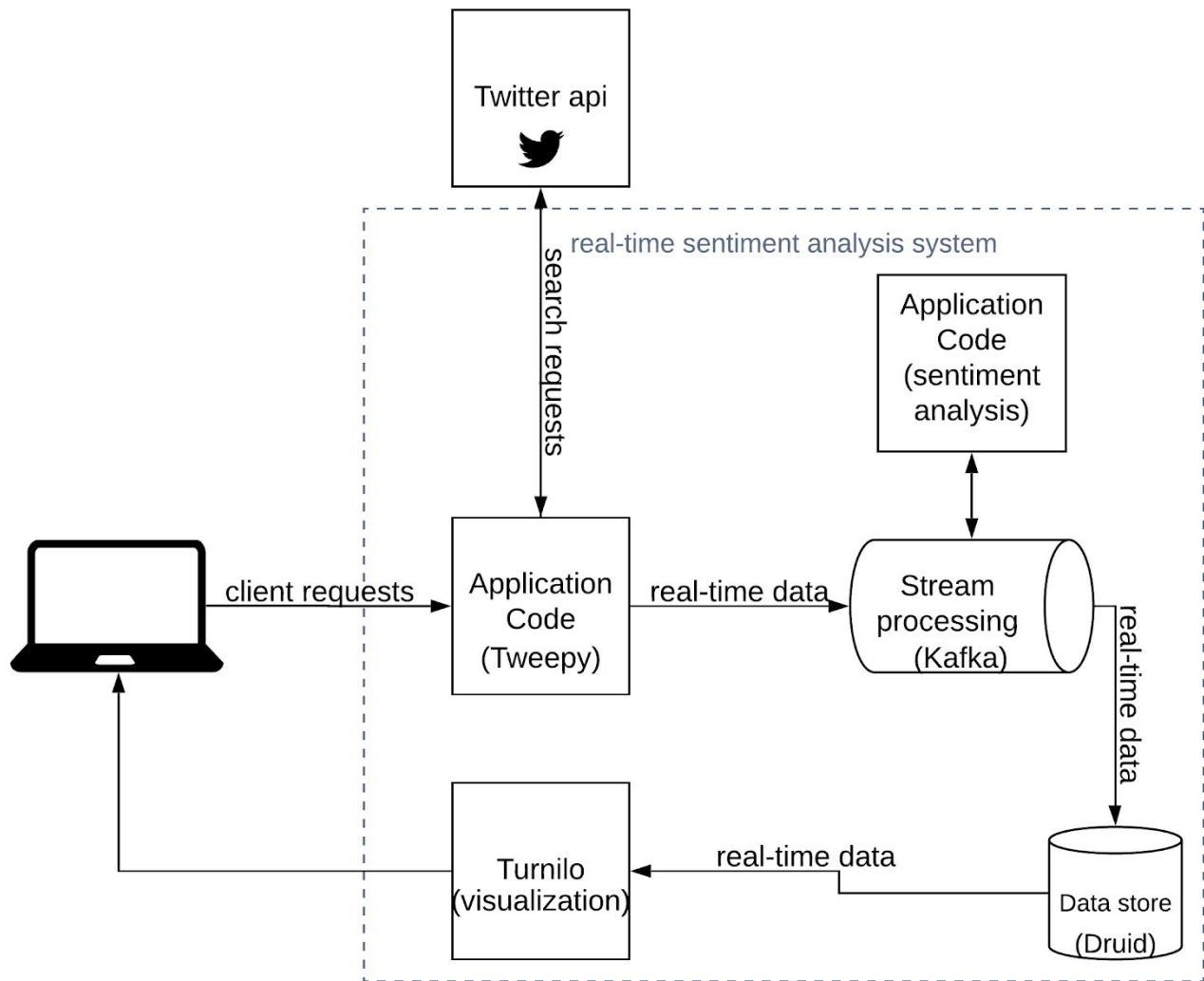


Figure 1: Real-time sentiment analysis system architecture

When a client gives requests to this application, it will search requests via Twitter api and give back the result. Then, the real-time data will be transferred to stream processing part using Kafka. In this part, it also goes through the sentiment analysis. After that, real-time data will be stored in data store through Druid and then the application uses Turnilo doing visualization, giving the result to the client.

1.3 Data Source

In this project, we decided to choose twitter as our project data source. In order to get streaming twitter data, we choose to use Tweepy which is a python library to listen to it. Tweepy is very useful to provide convenient for people to use twitter streaming API which is for downloading tweets in real time. In this way, people can download a high volume of tweets. In Tweepy, the StreamListener has on_status method to receive tweets and on_error method to stop grabbing data when it reaches the rate limits. For data we gathered, the field called text will give us the whole content of every tweet, the field called created_at will give us the date of creating this tweet, and the field called user.screen_name and user.location will give us the users' information about their name and location.

Twitter is a social media which has a large amount of social data. Unlike other social media, every tweet in users' account is accessible and can be gathered. This is really helpful for big data analysis. In the meantime, twitter's data is very specific. People can do very complex query such as requesting all tweets related to one specific topic in the latest 30 minutes. Under this situation, we can use data from twitter to analyze and achieve the goal of real time sentiment analysis.

In addition, twitter has its own API for people to obtain the source and use different functions without knowing the detail of its working mechanism. Thus, we can use twitter API to gather enough social data which is needed for this sentiment analysis task.

Twitter API has a really powerful query ability. In our project, we need to gain lots of tweets which are all related to the specific topic. We can use this API to complete this task. Also, twitter API can gather tweets based on accounts and timelines. It is convenient for us to add more limitations on social data so that we can have more accurate data to build the sentiment analysis system. Besides, Twitter API has many functions to satisfy different needs from developers. It supports people to get batch historical tweets or filter real-time tweets.

Following is the data flow of this project:

1. The streaming API endpoints will return a variety of messages and it will also handle errors.
2. In Twitter API, Tweepy, we can use it to get the entire time line or get specific tweets for sentiment analysis. We can use parameters such as “count” to set the number of tweets we want or use query to obtain specific topic tweets.
3. After obtaining data from twitter, we need to prepare data for next step including cleaning data doing some simplicity for it, and storing the data.
4. In this project, we will use Druid to record data which we gained in JSON format.

1.4 NoSQL Storage Technology

We choose Kafka and Druid to be our technology stack.

1. Why do we use Kafka here? Because Kafka is designed to be a distributed streaming platform for publishing and subscribing to streams of records. By Kafka, we can process data and split them into topics as they occur.
2. Here we choose Druid, why? Because Druid is good at dealing with real-time data with timestamp, has sql-like query mechanism, and can also decode json data and split them into different fields automatically. So we choose Druid to exploit the topics deeply.

2. Analyzing Data

2.1 Pipeline

We do 3 steps: 1) crawl related data (tweepy, keyword:“iphone”); 2) do sentiment analysis then split data into topics, like “iphone”/“iphone_positive”/“iphone_negative”/“iphone_neutral”/“iphone_sentiment”(Kafka); 3) do basic statistics about retweeted_status field of the tweet then push result to “iphone_stats” topic(Kafka) 4) query and analyze the data in each topic (Druid). 5) visualize data (Turnilo)

After we catch the messages from tweepy (section 2.1). We use Kafka to split the messages we receive into topic, the messages containing “iphone” will be classified into topic “iphone”(see code in Appendix: Crawl and Split Data).

Why do we use Kafka here? Because Kafka is designed to be a distributed streaming platform for publishing and subscribing to streams of records. By Kafka, we can process data and split them into topics as they occur.

Basically, Kafka is structured by Producers and Consumers. Producers can receive messages and send them to different topics. Consumers, who may be splitted into different groups, read the messages of a set of topics of their choice at their own pace. By this mechanism, we can classify messages quickly while dealing with them at different pace under different needs.

In our case, we have 1 producer (get message from tweepy), who send messages to one topic ("iphone")(see code in Appendix: Crawl and Split Data). And have 1 consumer (in the application code) to read the messages from topic to do the sentiment analysis then send messages to four topics("iphone_positive"/"iphone_negative"/"iphone_neutral") (see figure 2 for the structure)(see code in Appendix: Sentiment Analysis). And have another consumer to do basic statistics about retweeted_status field of the tweet then push result to "iphone_stats" topic(Kafka) (see code in Appendix: Compute Statistics)

Split Data into Different topics

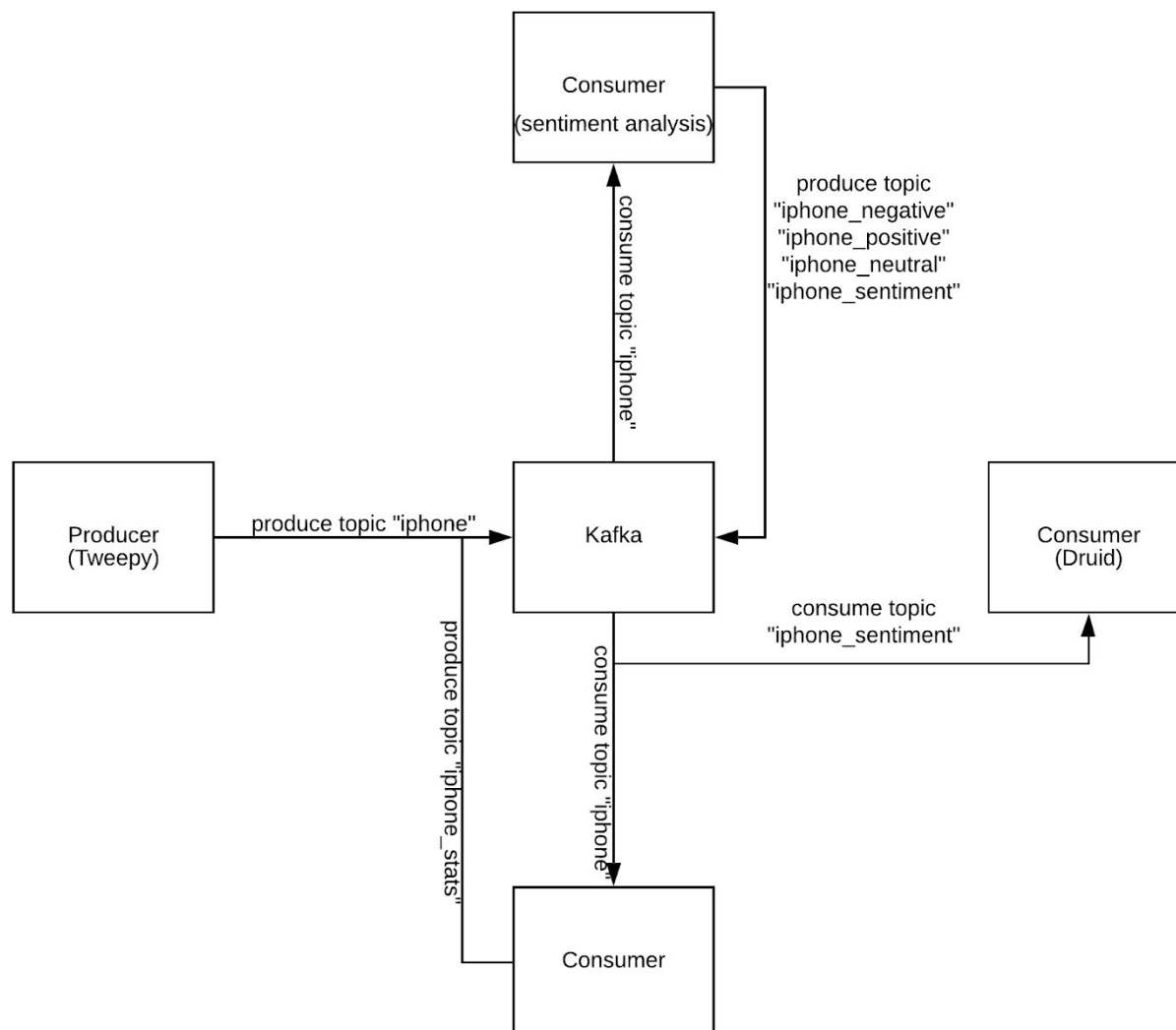


Figure 2. The Data Processing Structure of Kafka

Query and Analyze Data

After we split the messages into topic "iphone", we want to analyze the text that belongs to them. Here we choose Druid, why? Because Druid is good at dealing with real-time data with timestamp, has sql-like query mechanism, and can also decode json data and split them into different fields automatically. So we choose Druid to exploit the topics deeply (see appendix for Druid JSON Ingestion Spec).

2.2 Data Properties

Crawl Related Data

As we introduced in the Data Source section, we use twitter api to crawl and filter the tweets that we are interested in (see code in Appendix: Crawl and Split Data). The average size of every message is about 20K and the whole size of this data is more than 10G. We use the standard filter [1], which can filter 400 keywords, 5,000 user ids and 25 location boxes.

In the Sample Data section of Appendix, we crawl all the tweets which contain “iphone”. The sample message we catch includes fundamental attributes such as message id, created_at time, text, retweet count, replies, etc. A more detailed document could be referenced at [3]. Here are some most frequently used fields:

Key	Description	Type
id	The integer representation of the unique identifier for this Tweet.	Int64
created_at	UTC time when this Tweet was created.	String
text	The actual UTF-8 text of the status update.	String
retweet_count	Number of times this Tweet has been retweeted.	int
favorite_count	Indicates approximately how many times this Tweet has been liked by Twitter users.	int
user	Detailed information about the user who posted this Tweet.	User object
retweeted_status	Users can amplify the broadcast of Tweets authored by other users by retweeting . Retweets can be distinguished from typical Tweets by the existence of a retweeted_status attribute. This attribute contains a representation of the <i>original</i> Tweet that was retweeted. Note that retweets of retweets do not show representations of the intermediary retweet, but only the original Tweet. (Users can also unretweet a retweet they created by deleting their retweet.)	Tweet

2.3 Analytical Questions

Suppose you are Tim Cook, you’re giving the speech in the iphone 11 launch event. During the event, millions of tweets are generated to comment on the new product. You want to analyze these tweets in real-time to get the market’s feedback and decide what you would like to talk about in the next. You may want to ask these questions.

2.3.1 Easy Questions (by kafka and druid)

1. range/min/max/average of favorite count of tweets(favorite_count field) about iphone
2. range/min/max/average of followers number of tweet owner(followers_count field) about iphone
3. The line graph of number of tweets talking about iphone per minute

Since we are dealing with streaming data, all new tweets come in with zero favorite count, but we do have the favorite count data of the tweets that were retweeted by other accounts. By checking the `retweeted_status` attribute, we will find the statistics of the original post.

2.3.1.1 Answer by Kafka:

We use python application code to consume data from Kafka topics. After computation is done then push results to a stats topic in real time.

```
{
    "topic": "iphone",
    "timestamp_ms": "1572565422577",
    "favorite_count_min": 0,
    "favorite_count_max": 1132886,
    "favorite_count_average": 77018.2912781478,
    "favorite_count_range": [
        0,
        1132886
    ],
    "followers_count_min": 0,
    "followers_count_max": 78895964,
    "followers_count_average": 508343.3225499524,
    "followers_count_range": [
        0,
        78895964
    ]
}
{
    "topic": "iphone_negative",
    "timestamp_ms": "1572565422577",
    "favorite_count_min": 0,
    "favorite_count_max": 266352,
    "favorite_count_average": 153341.50328625366,
    "favorite_count_range": [
        0,
        266352
    ],
    "followers_count_min": 6,
    "followers_count_max": 8282130,
    "followers_count_average": 35503.88994546217,
    "followers_count_range": [
        6,
```



```

        8282130
    ]
}
{
    "topic": "iphone_neutral",
    "timestamp_ms": "1572565418127",
    "favorite_count_min": 0,
    "favorite_count_max": 1132886,
    "favorite_count_average": 6915.144655887823,
    "favorite_count_range": [
        0,
        1132886
    ],
    "followers_count_min": 0,
    "followers_count_max": 66384013,
    "followers_count_average": 202646.57404492775,
    "followers_count_range": [
        0,
        66384013
    ]
}
{
    "topic": "iphone_positive",
    "timestamp_ms": "1572565422228",
    "favorite_count_min": 0,
    "favorite_count_max": 626367,
    "favorite_count_average": 73492.44129249333,
    "favorite_count_range": [
        0,
        626367
    ],
    "followers_count_min": 0,
    "followers_count_max": 78895964,
    "followers_count_average": 832691.7019955912,
    "followers_count_range": [
        0,
        78895964
    ]
}

```

2.3.1.2 Answer by Druid:

Notes:

- On the basis of the original business questions, to stress the real-time characteristic of the streaming data, one more assumption was added: only tweets created within one hour would be taken into account. It would make more sense to care about the most popular tweets posted at once, instead of the tweets that were posted several months ago but with more favorite counts due to the longer time span.
- Druid loads data from kafka in JSON format. To answer the business question, these attributes are needed: `retweeted_status.created_at`, `retweeted_status.favorite_count`, `user.followers_count`. They are nested fields that require manually flattened and add into dimension metrics. The complete JSON ingestion spec can be found in the appendix.

```
"flattenSpec": {
  "fields": [
    {
      "type": "path",
      "name": "retweeted_status.created_at",
      "expr": "$.retweeted_status.created_at"
    },
    {
      "type": "path",
      "name": "user.followers_count",
      "expr": "$.user.followers_count"
    },
    {
      "type": "path",
      "name": "retweeted_status.favorite_count",
      "expr": "$.retweeted_status.favorite_count"
    }
  ]
}
```

- The statistical results are similar to what was given by kafka. To avoid the waste of space, here we only list the queries we used to get those results.

SELECT

```
MIN("retweeted_status.favorite_count")
FROM "iphone"
WHERE "created_at" >= CURRENT_TIMESTAMP - INTERVAL '1' HOUR
AND "retweeted_status.favorite_count" <> NULL
```

SELECT

```
MAX("retweeted_status.favorite_count")
FROM "iphone"
WHERE "created_at" >= CURRENT_TIMESTAMP - INTERVAL '1' HOUR
AND "retweeted_status.favorite_count" <> NULL
```

SELECT

```

    AVERAGE("retweeted_status.favorite_count")
FROM "iphone"
WHERE "created_at" >= CURRENT_TIMESTAMP - INTERVAL '1' HOUR
AND "retweeted_status.favorite_count" <> NULL

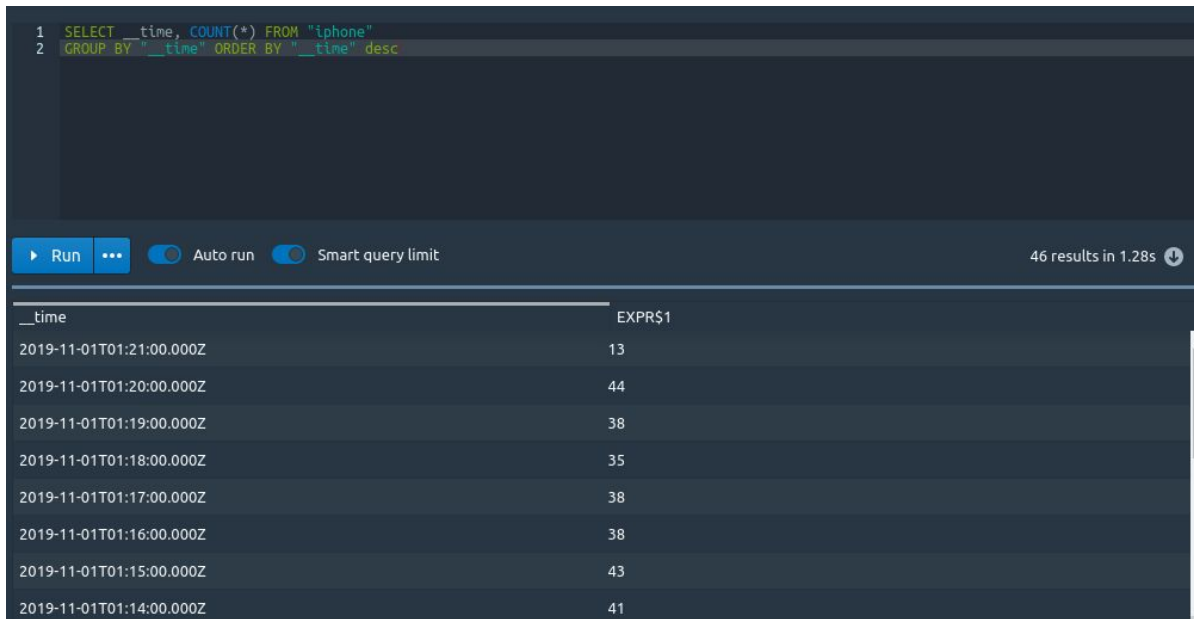
SELECT
    MIN("user.followers_count")
FROM "iphone"
WHERE "created_at" >= CURRENT_TIMESTAMP - INTERVAL '1' HOUR
AND "user.followers_count" <> NULL

```

2.3.1.3 Answer by Druid

First we configure druid to connect with Kafka's topic - iphone (see 5.1 druid configuration).

Then we can query in druid to search number of tweets talking about iphone per minute (2.2.1 - 1).



```

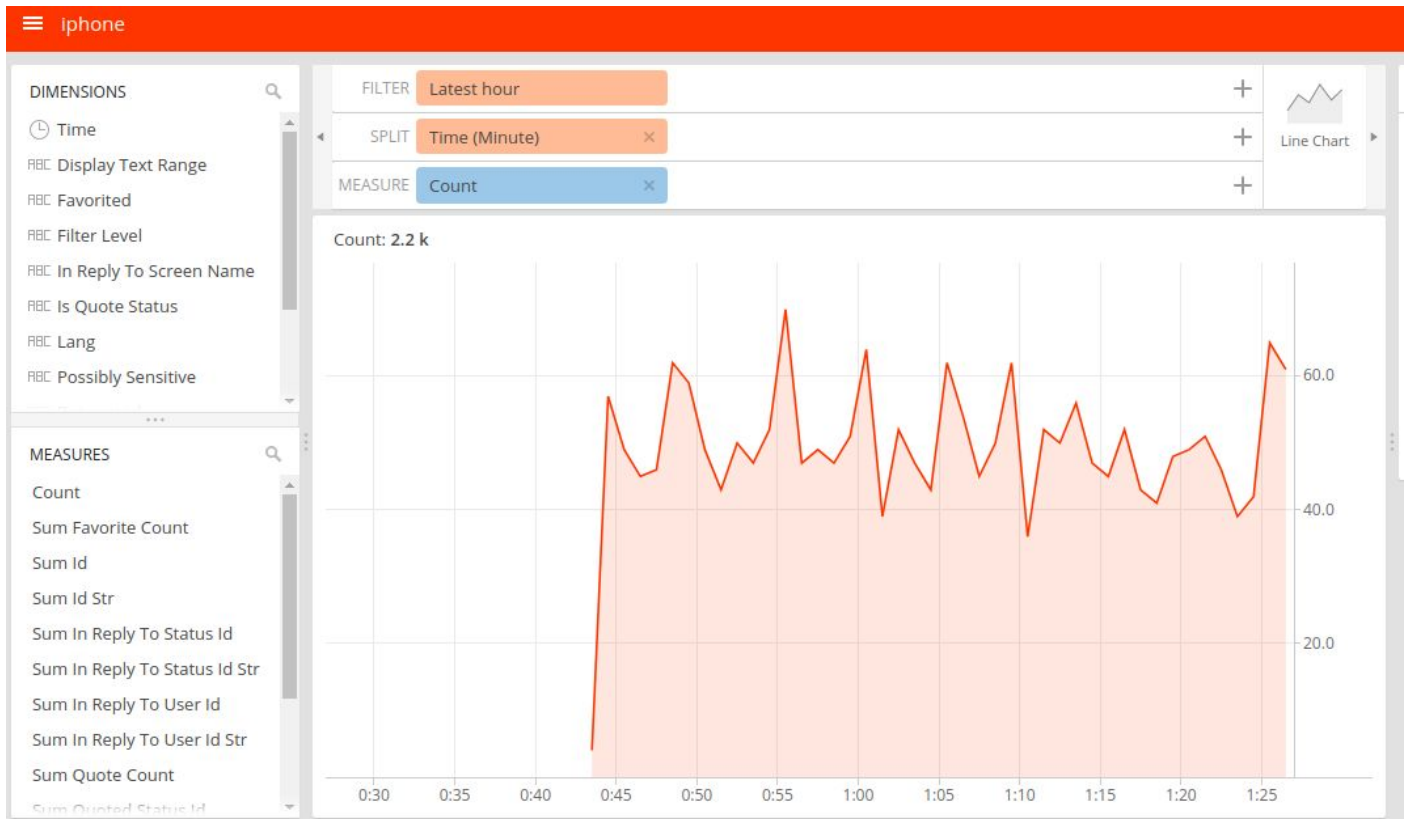
1 SELECT __time, COUNT(*) FROM "iphone"
2 GROUP BY "__time" ORDER BY "__time" desc

```

Run ... Auto run Smart query limit 46 results in 1.28s

__time	EXPR\$1
2019-11-01T01:21:00.000Z	13
2019-11-01T01:20:00.000Z	44
2019-11-01T01:19:00.000Z	38
2019-11-01T01:18:00.000Z	35
2019-11-01T01:17:00.000Z	38
2019-11-01T01:16:00.000Z	38
2019-11-01T01:15:00.000Z	43
2019-11-01T01:14:00.000Z	41

To see the tendency, we inject our data into turnilo. Here we can see the trend of number of tweets talking about iphone per minute.

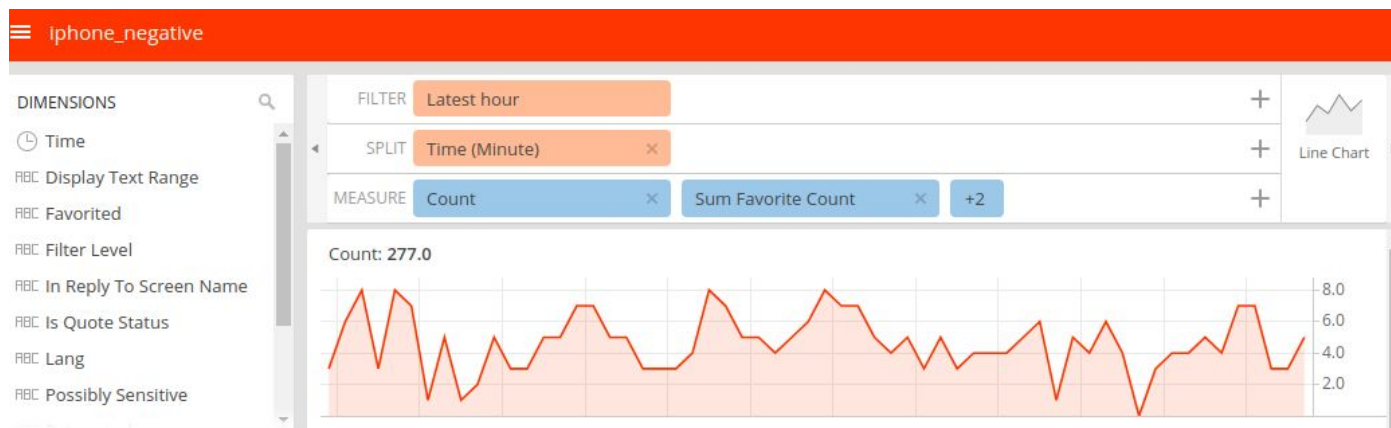


2.3.2 Difficult Questions (by Kafka or Druid)

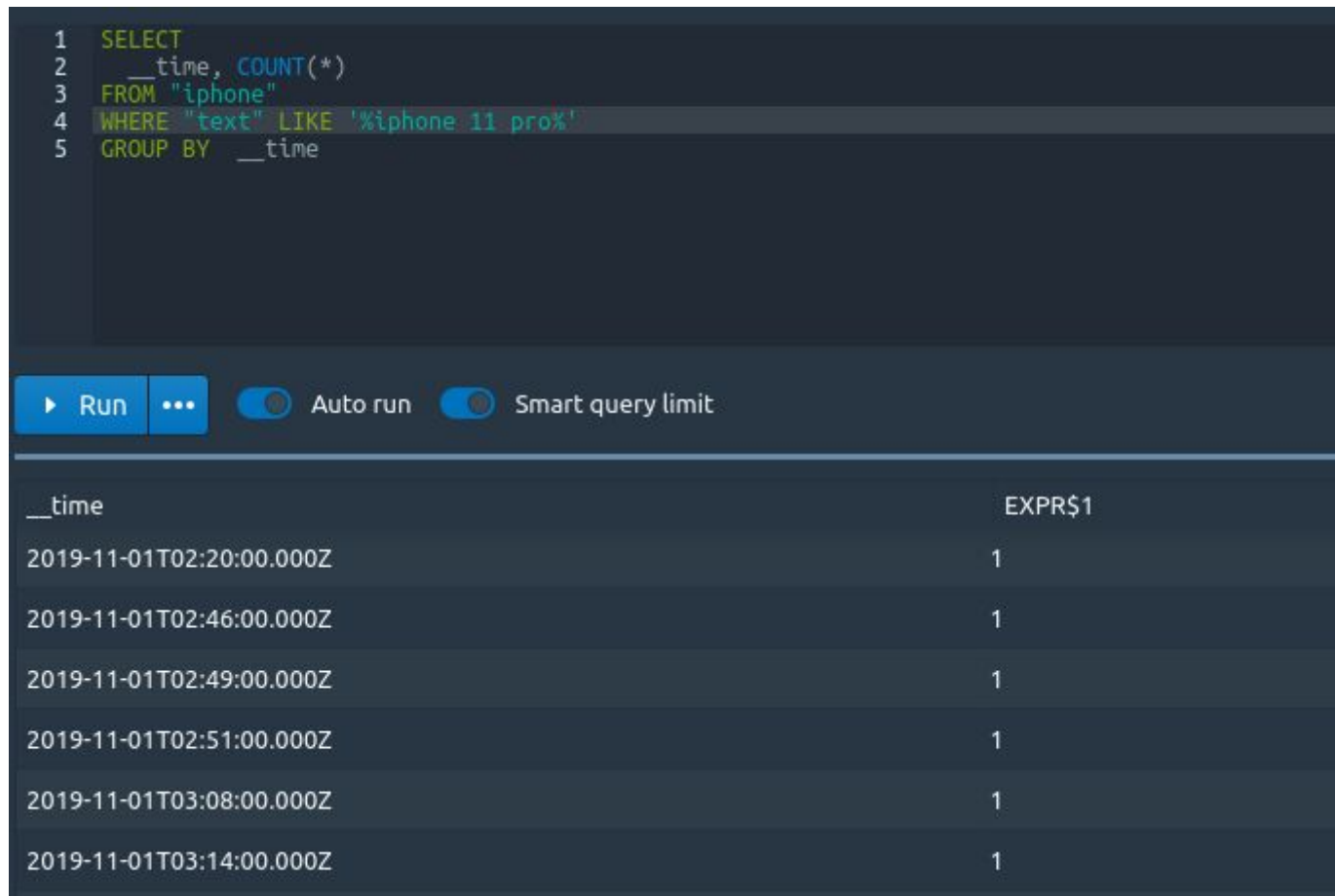
1. The line graph of number of tweets talking about iphone positively/negatively per minute
We write the application code in Kafka to use sentiment analysis tool textblob to classify all the tweets we receive into 3 topics: negative, positive, and neutral (see the complete code in Sentiment Analysis - appendix).

```
if tweet.sentiment.polarity < 0:
    producer.send(key+'_'+negative', msg.value)
    tweet_text_json['sentiment'] = 'negative'
    # sentiment = "negative"
elif tweet.sentiment.polarity == 0:
    # sentiment = "neutral"
    producer.send(key+'_'+neutral', msg.value)
    tweet_text_json['sentiment'] = 'neutral'
else:
    producer.send(key+'_'+positive', msg.value)
    tweet_text_json['sentiment'] = 'positive'
```

Then we inject the data to druid and turnilo.



2. The number of tweets talking about iphone and containing keywords: iphone 11 pro/11 per minute



```

1 SELECT
2   __time, COUNT(*)
3 FROM "iphone"
4 WHERE "text" LIKE '%iphone 11%'
5 GROUP BY __time

```

▶ Run ... Auto run Smart query limit

__time	EXPR\$1
2019-11-01T02:20:00.000Z	1
2019-11-01T02:23:00.000Z	1
2019-11-01T02:24:00.000Z	1
2019-11-01T02:27:00.000Z	1
2019-11-01T02:33:00.000Z	1
2019-11-01T02:35:00.000Z	2
2019-11-01T02:37:00.000Z	1
2019-11-01T02:40:00.000Z	1

- The latest 20 tweets which gives positive/negative feedback

SELECT TOP 20

"text"

FROM "iphone_positive"

WHERE "created_at" >= CURRENT_TIMESTAMP - INTERVAL '1' HOUR

ORDER BY "created_at" DESC

text

RT @WolfRewardz: iPhone 11 Giveaway | 20 Winners • Like + Retweet • Follow @WolfRewardz • Turn notifications on or you can't win! • Co...

RT @RudyGiuliani: Hey @NBCNews, last I checked the FBI, last year, had to ask Apple to unlock an iPhone too! We're all human, just maybe...

RT @TropicGaws: iPhone 11 Pro 64GB Giveaway! How to win: • Follow us @TropicGaws • Retweet and turn on our notifications • Comment: Legit...

RT @D_iW: When babes wan dress up tomorrow morning and they see their sandals and Mickey Mouse iPhone case <https://t.co/L17Uoe5ffx>

RT @MKBHD: Shot on Pixel 4 Lit by iPhone 11 📱📱📱📱📱 <https://t.co/QfcrJqBAfo>

RT @kevinabstract: I got the new iPhone <https://t.co/cehxiGpkh5>

RT @renato_mariotti: @RudyGiuliani @NBCNews It wasn't their own iPhone. It belonged to a suspect. You couldn't unlock your own phone.

RT @KingBach: 📱 IM GIVING AWAY 15 AirPod Pro's, 1 iPhone 11 Pro, and a \$17,000 chain! 📱 — HOW TO ENTER: 1 : TEXT: (310) 620-9822 2 : ...

RT @matthewamad: Kings, don't let peer pressure and need to impress chics push you into spending a fortune on iPhone whatever! Na so Samsun...

RT @FLYGIRLDEBBY: but clock dey the phone noww <https://t.co/PuSp67NVDI>

RT @JuliaDavisNews: "Very sloppy": Less than a month after he was named President Donald Trump's cybersecurity adviser, Rudy Giuliani need...

RT @jdawsey1: Soon after being named President Trump's top cybersecurity guru, Rudy Giuliani walked into a San Francisco Apple store becaus...

2.3.3 Challenging Questions (by druid)

1. Sentiment analysis of tweets(text field)

We use a python library called Textblob to do the sentiment analysis of tweets.(see code in Appendix: Crawl and Split Data)

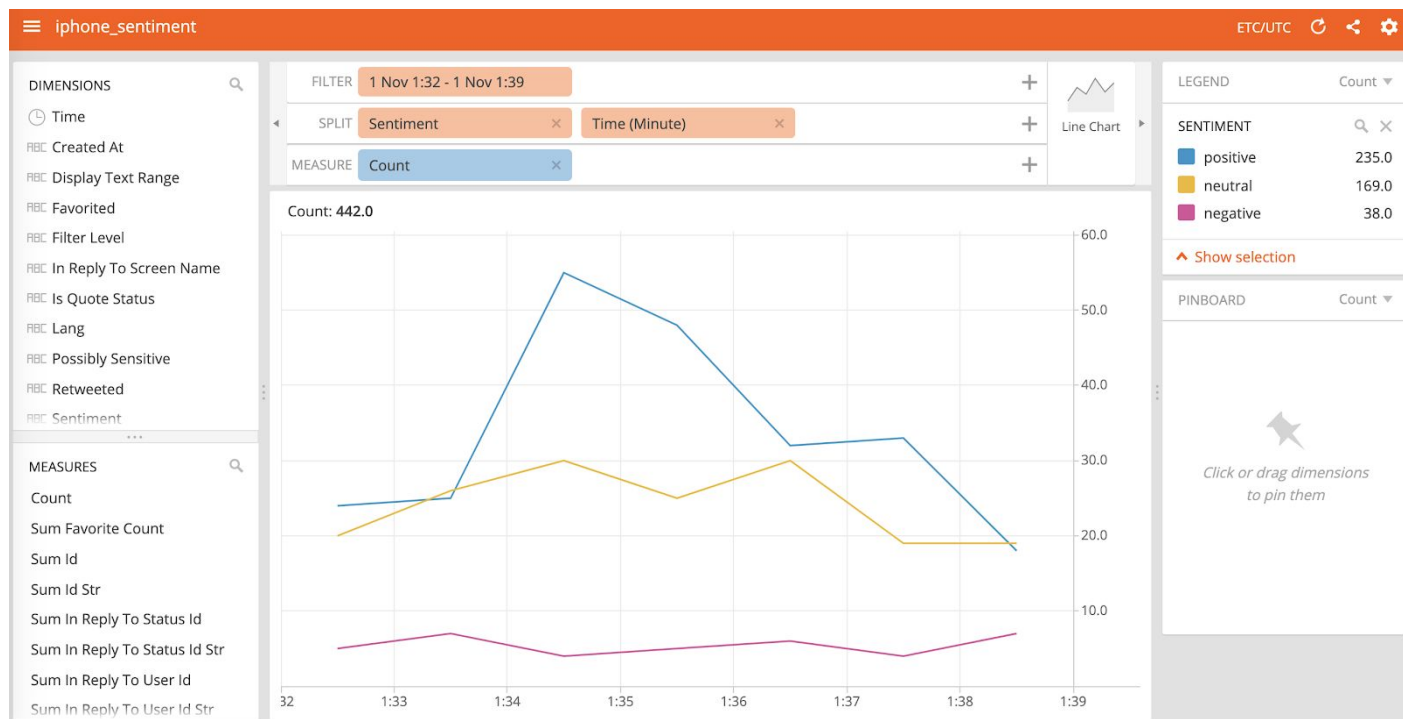
2. Correlation of the fields(other than text field) and sentiment of the tweet

For the keyword “iphone”, we got overall positive sentiment from time THU 31 2019 18:54 to THU 31 2019 18:58. From above statistics about followers of owner of tweets, we can see average followers of positive tweets is larger than others.

"followers_count_average_positive": 832691.7019955912,

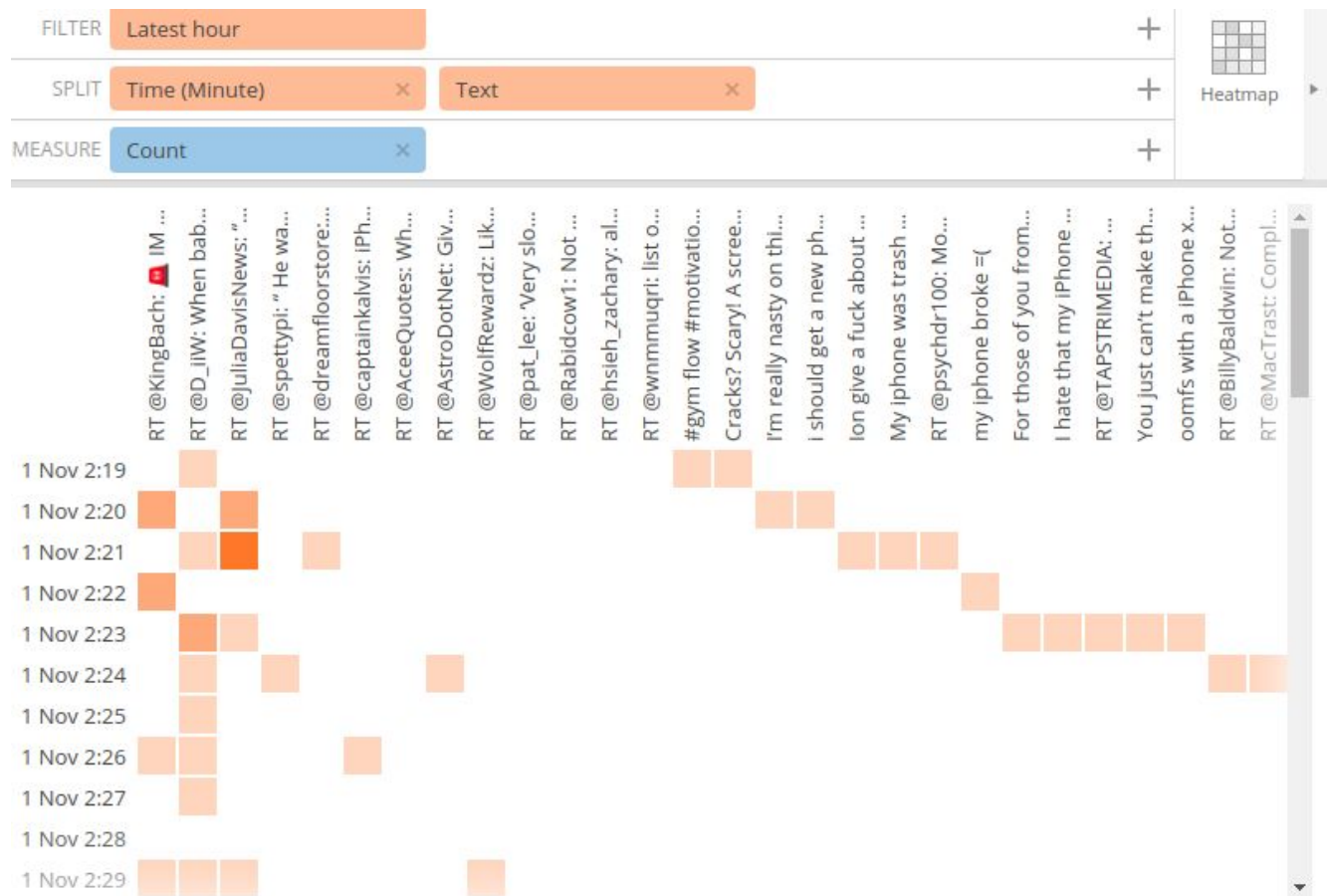
"followers_count_average_neutral": 202646.57404492775,

"followers_count_average_negative": 35503.88994546217



3. The hottest original tweets about iphone in last hour

Then we feed the data into turnilo to generate the heatmap.



See the most dense point, here is this tweet we found which is the hottest about iphone in 1 Nov 2:21



Julia Davis
@JuliaDavisNews

Follow

“Very sloppy”:

Less than a month after he was named President Donald Trump's cybersecurity adviser, Rudy Giuliani needed Apple genius help to unlock his iPhone, after he had forgotten the passcode and entered the wrong one at least 10 times.



Giuliani needed Apple genius help to unlock his iPhone after named Trump c...

Giuliani's handling of the situation calls into question his understanding of basic security measures, two former FBI cyber experts told NBC News.

[nbcnews.com](https://www.nbcnews.com)

11:52 AM - 31 Oct 2019

810 Retweets 1,931 Likes



4. The top 4 n-gram keywords per minute in the negative/positive tweets about iphone
We use pydruid with application code, nltk library to solve this problem, see appendix -keyword extraction, we query the druid every 2 seconds:
Here are the top 4 keywords for negative tweets on iphone:

	frequency
11	39
11 pro	39
512gb	39
512gb unlocked	38

Seems a lot of people are complaining about its size when we use druid to look into it.

```
SELECT
  __time, COUNT(*)
FROM "iphone_negative"
WHERE "text" LIKE '%512gb%'
GROUP BY __time
```



3. Potential Issues and Challenges

1. The first potential issue will be the confidence of the sentiment analysis using NLP technology. Although machine learning NLP technology is getting better through these years, it still will fail in some cases. And solving this issue will be a big challenge. There is a workaround for this situation. That is, we show the confidence level of our results to users and let users decide what final result is correct.

2. The second potential issue will be the response time of our system. Along with data growth, the analyzing speed of our system will be decreased. We need to design our algorithm as concurrent as possible. The algorithm design is a big challenge.

4. Lessons learned

1. We are very glad to have Kafka in our system. Our application functionality code is either producer or consumer of Kafka. All we have to do is build our system around Kafka. Kafka exponentially reduces implementation complexity.
2. In our system implementation, we have built a Kafka cluster in aws ec2 instance. Cluster has 3 zookeeper and 3 kafka brokers. Docker is a very good tool to isolate programming environments, though docker port mapping takes a lot of time to figure out.
3. Our system can handle the case where 2 brokers are crashed. The kafka producer is still able to serve data to consumer without data loss. We didn't do the high replication configuration because of the availability of ec2 instance disk space.
4. Druid has a very good response time. We have produced and consumed 10+G of data in total, and still extract, visualize them in real time.

5. References

1. <https://developer.twitter.com/en/docs/tweets/filter-realtime/overview>
2. http://docs.tweepy.org/en/v3.4.0/streaming_how_to.html#streaming-with-tweepy
3. <https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object>
4. <https://github.com/allegro/turnilo>
5. <https://druid.apache.org/docs/latest/design/>
6. <https://github.com/Batu-MGL/BigDataProject>

6. Appendix

System Setup

1. Local Kafka Cluster Setup
 - 1.1 install kafka on the machine
 - 1.2 install docker and docker-compose
 - 1.3 run command: `docker-compose -f zk-multiple-kafka-multiple.yml up` (reference 6.)
2. Local Python Environment Setup
 - 2.1 install python library tweepy/ kafka-python/textblob
3. Local Node.js(Turnilo) Environment Setup
 - 3.1 install turnilo visualization library (reference 5.)
4. Programs Running Order (reference 6)
 - 4.1 run `bigdata_project1_producer1.py`
 - 4.2 run `bigdata_project1_sentiment_analysis_consumer.py`
 - 4.3 run `bigdata_project1_stats_consumer.py`

Crawl and Split Data

```
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream
from kafka import KafkaProducer
from kafka.client import SimpleClient
from kafka.producer import SimpleProducer
from kafka import KafkaConsumer
import json
import os
import sys

# client = SimpleClient("127.0.0.1:9092")
# producer = SimpleProducer(client)

producer = KafkaProducer(bootstrap_servers=['localhost:9092'])
consumer_key= 'tYCzKOoBOZft1oy0cC85l0mxU'
consumer_secret= 'tS2e6cMQv60ZtvoBaAXenLrkr26oSg8c22fyrclSUvHZGNylus'
access_token= '4808614249-s758NEUKUDHjmsPTsqqak1zSNkmgOI3mOyio0tv'
access_token_secret= 'JBW4qGXXs3MAXCgwplERYOqnCiJyIN3uF8oKIPWn3r7wn'
class StdOutListener(StreamListener):
    key=""
    def set_keys(self,key):
        self.key = key
    def on_data(self, data):
        # if str(sys.argv[1]) in str(data):
        # if 'Rams' in str(data):
        if self.key in str(data):
            # producer.send(str(sys.argv[1]), bytes(data,'utf-8'))
            # producer.send('rams', bytes(data,'utf-8'))
            producer.send(self.key, bytes(data,'utf-8'))
        print(data)
        return True
    def on_error(self, status):
        print(status)
if __name__ == '__main__':
    key='iphone'
```

```
# os.system('kafka-topics.sh --zookeeper localhost:2181 --create --topic'+ ' '+key+' --partitions 3  
--replication-factor 2')
```

```
l = StdOutListener()  
l.set_keys(key)  
auth = OAuthHandler(consumer_key, consumer_secret)  
auth.set_access_token(access_token, access_token_secret)  
stream = Stream(auth, l)  
stream.filter(track=[key],languages=["en"])
```

For more info at https://github.com/Batu-MGL/BigDataProject/blob/master/bigdata_project1_producer.py

Sentiment Analysis

```
from kafka import KafkaConsumer  
import json  
from textblob import TextBlob  
from kafka import KafkaProducer  
  
# consumer = KafkaConsumer('rams',group_id='my_favorite_group')  
key='iphone'  
consumer1 = KafkaConsumer(key,group_id='sentiment_analysis_group',  
bootstrap_servers=['localhost:9092'])  
producer = KafkaProducer(bootstrap_servers=['localhost:9092'])  
  
for msg in consumer1:  
    tweet_text_json=json.loads(msg.value.decode('utf-8'))  
    tweet = TextBlob(tweet_text_json['text'])  
    if tweet.sentiment.polarity < 0:  
        producer.send(key+'_'+ 'negative', msg.value)  
        tweet_text_json['sentiment'] = 'negative'  
        # sentiment = "negative"  
    elif tweet.sentiment.polarity == 0:  
        # sentiment = "neutral"  
        producer.send(key+'_'+ 'neutral', msg.value)  
        tweet_text_json['sentiment'] = 'neutral'  
    else:  
        producer.send(key+'_'+ 'positive', msg.value)  
        tweet_text_json['sentiment'] = 'positive'  
        # sentiment = "positive"  
    tweet_text_json['sentiment_value'] =tweet.sentiment.polarity
```

```
producer.send(key+'_'+'sentiment', bytes(json.dumps(tweet_text_json), 'utf-8'))

print(tweet_text_json['sentiment'])
```

For more info at

https://github.com/Batu-MGL/BigDataProject/blob/master/bigdata_project1_sentiment_analysis_consumer.py

Compute Statistics

```
from kafka import KafkaConsumer
import json
from textblob import TextBlob
from kafka import KafkaProducer

# consumer = KafkaConsumer('rams',group_id='my_favorite_group')

key='iphone'
consumer1 = KafkaConsumer(key,auto_offset_reset='earliest',
bootstrap_servers=['localhost:9092'])
favorite_count_min=100000000
favorite_count_max=0
favorite_count_sum=0
favorite_count_n=0
favorite_count_range=(100000000,0)
favorite_count_average=0
followers_count_min=100000000
followers_count_max=0
followers_count_sum=0
followers_count_n=0
followers_count_range=[100000000,0]
followers_count_average=0

producer = KafkaProducer(bootstrap_servers=['localhost:9092'])
tweet_dict= {}

for msg in consumer1:
    tweet_text_json=json.loads(msg.value.decode('utf-8'))
    try:
```

```

        tweet_favorite_count =
tweet_text_json['retweeted_status']['favorite_count']
        tweet_followers_count =
tweet_text_json['retweeted_status']['user']['followers_count']

        favorite_count_sum=favorite_count_sum+tweet_favorite_count
        favorite_count_n=favorite_count_n+1
        favorite_count_average=favorite_count_sum/favorite_count_n

        followers_count_sum=followers_count_sum+tweet_followers_count
        followers_count_n=followers_count_n+1
        followers_count_average=followers_count_sum/followers_count_n

    if tweet_followers_count < followers_count_min:
        followers_count_min=tweet_followers_count
        followers_count_range=[followers_count_min,followers_count_range[1]]
    if tweet_followers_count > followers_count_max:
        followers_count_max=tweet_followers_count
        followers_count_range=[followers_count_range[0],followers_count_max]
        # sentiment = "negative"
    if tweet_favorite_count < favorite_count_min:
        # sentiment = "neutral"
        favorite_count_min=tweet_favorite_count
        favorite_count_range=[favorite_count_min,favorite_count_range[1]]
    if tweet_favorite_count > favorite_count_max:
        # sentiment = "neutral"
        favorite_count_max=tweet_favorite_count
        favorite_count_range=[favorite_count_range[0],favorite_count_max]

    tweet_dict['topic']=key
    tweet_dict['timestamp_ms']=tweet_text_json['timestamp_ms']
    tweet_dict['favorite_count_min']=favorite_count_min
    tweet_dict['favorite_count_max']=favorite_count_max
    tweet_dict['favorite_count_average']=favorite_count_average
    tweet_dict['favorite_count_range']=favorite_count_range
    tweet_dict['followers_count_min']=followers_count_min
    tweet_dict['followers_count_max']=followers_count_max
    tweet_dict['followers_count_average']=followers_count_average
    tweet_dict['followers_count_range']=followers_count_range
    producer.send(key+'_'+stats', bytes(json.dumps(tweet_dict), 'utf-8'))

```

```

print(json.dumps(tweet_dict))
except KeyError:
    print("No retweet status")

```

For more info at

https://github.com/Batu-MGL/BigDataProject/blob/master/bigdata_project1_stats_consumer.py

Sample Data

```

{
  "created_at": "Tue Oct 29 03:53:14 +0000 2019",
  "id": 1189027386546884608,
  "id_str": "1189027386546884608",
  "text": "RT @cupidsucker: the iphone 11 max pro camera is so clear u can just
take a picture of a man and see all the fucking lies",
  "source": "\u003ca href=\"http://twitter.com/download/iphone\"
rel=\"nofollow\"\u003eTwitter for iPhone\u003c/a\u003e",
  "truncated": false,
  "in_reply_to_status_id": null,
  "in_reply_to_status_id_str": null,
  "in_reply_to_user_id": null,
  "in_reply_to_user_id_str": null,
  "in_reply_to_screen_name": null,
  "user": {
    "id": 4880501386,
    "id_str": "4880501386",
    "name": "Karol Elizondo",
    "screen_name": "karolelizondo6",
    "location": "Monterrey, Nuevo Le\u00f3n",
    "url": null,
    "description": "XIX \/\ Snap: karolelizondo",
    "translator_type": "none",
    "protected": false,
    "verified": false,
    "followers_count": 725,
    "friends_count": 610,
    "listed_count": 3,
    "favourites_count": 42287,
    "statuses_count": 45124,
    "created_at": "Sat Feb 06 06:50:42 +0000 2016",
    "utc_offset": null,
    "time_zone": null,

```

```

    "geo_enabled": true,
    "lang": null,
    "contributors_enabled": false,
    "is_translator": false,
    "profile_background_color": "F5F8FA",
    "profile_background_image_url": "",
    "profile_background_image_url_https": "",
    "profile_background_tile": false,
    "profile_link_color": "1DA1F2",
    "profile_sidebar_border_color": "CODEED",
    "profile_sidebar_fill_color": "DDEEF6",
    "profile_text_color": "333333",
    "profile_use_background_image": true,
    "profile_image_url":
"http://pbs.twimg.com/profile_images/1134704366013825025/bA0oEJih_normal.jpg",
    "profile_image_url_https":
"https://pbs.twimg.com/profile_images/1134704366013825025/bA0oEJih_normal.jpg"
,
    "profile_banner_url":
"https://pbs.twimg.com/profile_banners/4880501386/1554426836",
    "default_profile": true,
    "default_profile_image": false,
    "following": null,
    "follow_request_sent": null,
    "notifications": null
},
"geo": null,
"coordinates": null,
"place": null,
"contributors": null,
"retweeted_status": {
    "created_at": "Mon Oct 28 22:40:21 +0000 2019",
    "id": 1188948648215773184,
    "id_str": "1188948648215773184",
    "text": "the iphone 11 max pro camera is so clear u can just take a picture
of a man and see all the fucking lies",
    "source": "\u003ca href=\"http://twitter.com/download/iphone\"
rel=\"nofollow\"\u003eTwitter for iPhone\u003c/a\u003e",
    "truncated": false,
    "in_reply_to_status_id": null,

```



```

"in_reply_to_status_id_str": null,
"in_reply_to_user_id": null,
"in_reply_to_user_id_str": null,
"in_reply_to_screen_name": null,
"user": {
  "id": 1162610583419514880,
  "id_str": "1162610583419514880",
  "name": "\u2027\u208a\u02da\u2661",
  "screen_name": "cupidsucker",
  "location": null,
  "url": null,
  "description": "\uff65 \uff61\uff9f\u2606: *.\u263d .*
:\u2606\uff9f++nina",
  "translator_type": "none",
  "protected": false,
  "verified": false,
  "followers_count": 16818,
  "friends_count": 142,
  "listed_count": 36,
  "favourites_count": 769,
  "statuses_count": 194,
  "created_at": "Sat Aug 17 06:22:28 +0000 2019",
  "utc_offset": null,
  "time_zone": null,
  "geo_enabled": true,
  "lang": null,
  "contributors_enabled": false,
  "is_translator": false,
  "profile_background_color": "F5F8FA",
  "profile_background_image_url": "",
  "profile_background_image_url_https": "",
  "profile_background_tile": false,
  "profile_link_color": "1DA1F2",
  "profile_sidebar_border_color": "C0DEED",
  "profile_sidebar_fill_color": "DDEEF6",
  "profile_text_color": "333333",
  "profile_use_background_image": true,
  "profile_image_url":
"http://pbs.twimg.com/profile_images/1188134952900022277/5D1V6mz8_normal.jpg",

```

```

        "profile_image_url_https":
"https://pbs.twimg.com/profile_images/1188134952900022277/5DlV6mz8_normal.jpg"
,
        "profile_banner_url":
"https://pbs.twimg.com/profile_banners/1162610583419514880/1570860976",
        "default_profile": true,
        "default_profile_image": false,
        "following": null,
        "follow_request_sent": null,
        "notifications": null
    },
    "geo": null,
    "coordinates": null,
    "place": null,
    "contributors": null,
    "is_quote_status": false,
    "quote_count": 168,
    "reply_count": 54,
    "retweet_count": 2213,
    "favorite_count": 13813,
    "entities": {
        "hashtags": [],
        "urls": [],
        "user_mentions": [],
        "symbols": []
    },
    "favorited": false,
    "retweeted": false,
    "filter_level": "low",
    "lang": "en"
},
"is_quote_status": false,
"quote_count": 0,
"reply_count": 0,
"retweet_count": 0,
"favorite_count": 0,
"entities": {
    "hashtags": [],
    "urls": [],
    "user_mentions": [

```

```

    {
      "screen_name": "cupidsucker",
      "name": "\u2027\u208a\u02da\u2661",
      "id": 1162610583419514880,
      "id_str": "1162610583419514880",
      "indices": [
        3,
        15
      ]
    },
    "symbols": []
  },
  "favorited": false,
  "retweeted": false,
  "filter_level": "low",
  "lang": "en",
  "timestamp_ms": "1572321194670"
}

```

For more info at

<https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object>

Druid JSON Ingestion Spec

```

{
  "type": "kafka",
  "dataSchema": {
    "dataSource": "iphone",
    "parser": {
      "type": "string",
      "parseSpec": {
        "format": "json",
        "timestampSpec": {
          "column": "created_at",
          "format": "EEE MMM dd HH:mm:ss xxxxx yyyy"
        },
      },
      "flattenSpec": {
        "fields": [
          {
            "type": "path",
            "name": "retweeted_status.created_at",
            "expr": "$.retweeted_status.created_at"
          },
          {
            "type": "path",
            "name": "user.followers_count",
            "expr": "$.user.followers_count"
          },
          {
            "type": "path",
            "name": "retweeted_status.favorite_count",
            "expr": "$.retweeted_status.favorite_count"
          }
        ]
      }
    }
  }
}

```

```

    }
  ]
},
"dimensionsSpec": {
  "dimensions": [
    "text",
    "user.followers_count",
    "retweeted_status.created_at",
    "retweeted_status.user.id",
    "retweeted_status.favorite_count"
  ]
}
},
"metricsSpec": [
  {
    "type": "count",
    "name": "count"
  }
],
"granularitySpec": {
  "type": "uniform",
  "segmentGranularity": "MINUTE",
  "queryGranularity": "SECOND",
  "rollup": true,
  "intervals": null
},
"transformSpec": {
  "filter": null,
  "transforms": []
}
}

```

Keyword Extraction

```

from pydruid.client import *
from pydruid.utils.aggregators import doublesum
from datetime import datetime
from pydruid.utils.filters import Dimension
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import time
from sklearn.feature_extraction.text import CountVectorizer

query = PyDruid("http://localhost:8082", 'druid/v2')
datasource = 'iphone_negative'

while (True):
    current_time = datetime.today().strftime('%Y-%m-%d')
    ts = query.select(
        datasource=datasource,
        granularity='minute',
        intervals=current_time+'/p1d',
        paging_spec={'pagingIdentifies': {}, 'threshold': 1}
    )
    df = query.export_pandas()

```

```
word_vectorizer = CountVectorizer(ngram_range=(1, 2), analyzer='word')
sparse_matrix = word_vectorizer.fit_transform(df['text'])
frequencies = sum(sparse_matrix).toarray()[0]
res = pd.DataFrame(frequencies, index=word_vectorizer.get_feature_names(),
columns=['frequency']):10].sort_values('frequency', ascending=False)[:4]#.sort_values('frequency',
ascending=False):10]
print(res)
time.sleep(2)
```