Trendy Tweets

Abhishek Raorane (abhishek.raorane@utdallas.edu), Fatima Merchant (fatima.merchant@utdallas.edu),
Gaurav Kamath(gaurav.kamath@utdallas.edu), Zeel Shah(zxs120830@utdallas.edu),
Chandak Rajesh Ghanshyam(cxg130030@utdallas.edu)

Abstract— **Processing live tweets from Twitter with Storm followed by storage in Cassandra, we look forward to find out trending topics at Twitter. For better performance we use distributed Rolling Count, also known as sliding window, algorithm in Storm. We are trying to understand the most trendy topic in every 10sec interval.**

I. Introduction

The project titled "Trendy Tweets" will take the count of the words that are tweeted with a # in front of the word say #Storm. If this word appears 200 times within 10secs then the value of 200 will be stored on the sliding window for that time interval for the word "Storm". Before describing the algorithm, we give you a brief introduction on the key terms and systems used on this report.

A. Twitter:

Twitter is an online social networking and microblogging service that enables users to send and read "tweets"[1].

B. Tweets:

Text messages on Twitter limited to 140 characters. New Tweets per second (TPS) record: 143,199 TPS. Typical day: more than 500 million Tweets sent; average 5,700 TPS[2].

C. Trending topics:

A word, phrase or topic that is tagged at a greater rate than other tags is said to be a trending topic. Trending topics become popular either through a concerted effort by users or because of an event that prompts people to talk about one specific topic. These topics help Twitter and their users to understand what is happening in the world[3].

D. #Hashtags in Tweets:

A hashtag is any word or phrase with the # symbol immediately in front of it. This symbol turns the word into a link that makes it easier to find and follow a conversation about that topic. We could opt to use a simple pattern matching algorithm that treats #hashtags in tweets as topics. Here, we would consider a tweet such as

@userid_xyz: #Storm project rocks for #data processing! to mention the topics (case insensitive)

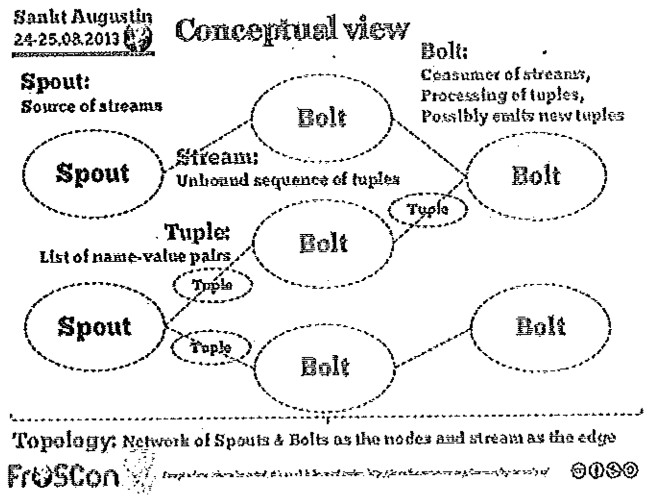
1. Storm or storm
2. data or Data

We design our system so that it considers topic A more popular than topic B (for a given time span) if topic A has been mentioned more often in tweets than topic B. This means we only need to *count* the number of occurrences of topics in tweets.

$$popularity(A) \geq popularity(B) \Leftrightarrow mentions(A) \geq mentions(B)$$

E. Storm:

Storm is free and open source distributed and fault-tolerant real-time computation system. Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing. It is fast, a benchmark clocked it over a million tuples processed per tuple per node. It is scalable and gurantees your data will be processed and is easy to setup and operate.



[4] **Figure1:** Conceptual View of Spout and Bolt

Streams will process the live text tweets coming from Twitter. We even get the metadata of the tweets but for this project we only need the text data of the tweets and we keep the metadata out. In general, Stream will receive text messages from a source (Twitter in our case). Stream will then pass this text messages to Spout(s). Spout can talk with Queues, Web logs, API calls and Event data. Spout then breaks the text messages into tuples to be processed by different bolts in parallel. Bolt does the main processing on the tuples like applying transformations, aggregations, filtering, stream joins, access DBs, APIs etc as seen pictorially below[4]:

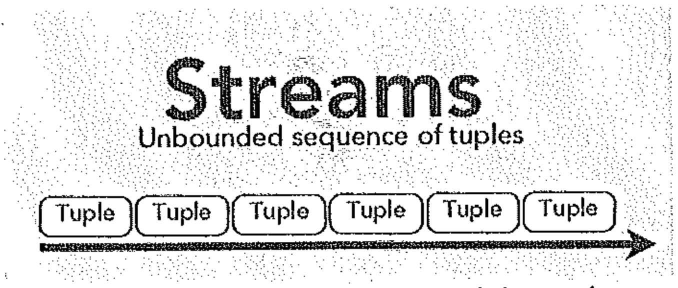


Figure2: Source passes data to Streams and Stream in turn passes it to Spouts

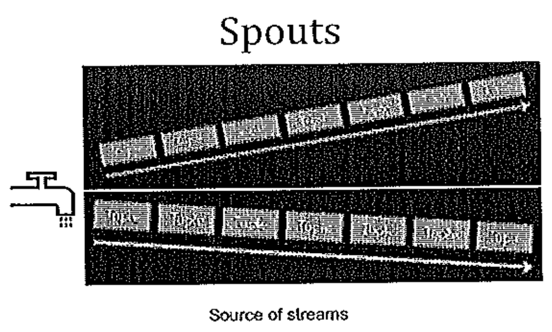


Figure3: Spout receives the data from Stream and passes it to one or more Bolts

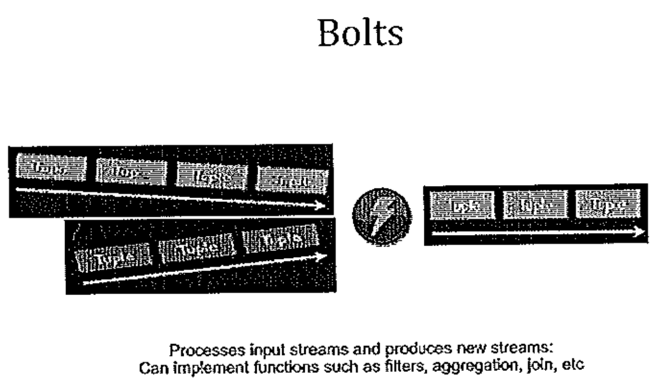

Bolts
Processes input streams and produces new streams:
Can implement functions such as filters, aggregation, join, etc

Figure4: Bolts processes the data coming from Spouts or intermediate Bolts

Distributed processing in Storm:

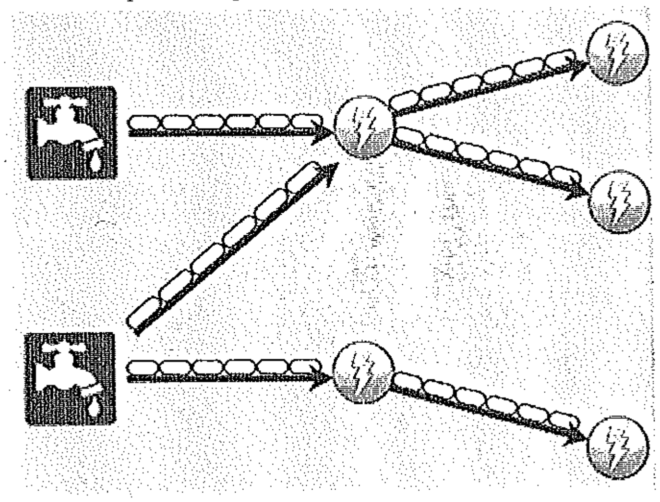


Figure5: Distributed processing in Storm

Many Spouts emit to many Bolts and Bolts also pass their output to other Bolts for faster and additional processing of data. The Storm topology described in this report will be able to identify in real-time the trending topics in this input data using a time-sensitive rolling count algorithm (rolling counts are also known as sliding windows) coupled with a ranking step. The former aspect takes care of filtering user input by time span, the latter of ranking the most trendy topics at the top the list[3].

Eventually we want our Storm topology to periodically produce the top N of trending topics similar to the following example output, where t0 to t2 are different points in time:

```
Rank @ t0  ----->  t1  ----->  t2
-------------------------------------------
1.  java  (33)  ruby  (41)  scala  (32)
2.  php   (30)  scala  (28)  python (29)
3.  scala  (21)  java  (27)  ruby  (24)
4.  ruby  (16)  python (21)  java  (21)
5.  python (15)  php   (14)  erlang (18)
```

In this example we can see that over time "scala" has become the hottest trending topic.

II. Testing Environment:

With the help of Virtual Machine, Ubuntu was installed on Windows 8 64bit machine. Storm was then installed on the Ubuntu Operating System as it is not supported on Windows Operating System.

III. Implementation:

We used the Storm Starter Project[5] as our base and performed changes as per the project. We had to work on Twitter Spout, Hashtag Bolt, Roll Count Bolt, intermediate Bolts and the final Bolt to fetch the desired output onto a text file. We also had to ignore metadata(s) to only consider the text from the tweets and consider some more filters other than just space for extracting the #hashtags.

A: Download real-time tweets from Twitter:

We used "twitter4j" library to extract live data from Twitter. We need the library for using Twitter API.

Twitter4J[6] is an unofficial Java library for the Twitter API. With Twitter4J, you can easily integrate your Java application with the Twitter service.

Twitter4J is featuring:
✔100% Pure Java - works on any Java Platform version 5 or later
✔Android platform and Google App Engine ready
✔Zero dependency : No additional jars required
✔Built-in OAuth support
✔Out-of-the-box gzip support
✔100% Twitter API 1.1 compatible

B. #Hashtags in Tweets:

A hashtag is any word or phrase with the # symbol immediately in front of it. This symbol turns the word into a link that makes it easier to find and follow a conversation about that topic. we could opt to use a simple pattern matching algorithm that treats #hashtags (case insensitive) in tweets as topics. Here, we would consider a tweet such as

@userid_xyz: #Storm project rocks for #data processing!

@userid_abc: #Hadoop has shaken the traditional #Data storage platform!

1. Storm will be counted once.
2. Data will be counted twice.
3. Hadoop will be counted once.

We design our system so that it considers Data (count=2) more popular than topic Storm or Hadoop (for a given time span, count=1 each) considering topic 'Data' has been mentioned more often in tweets than topic 'Hadoop' or 'Storm'.

C. RollingCount(Sliding Window)Algorithm in Storm[3]:

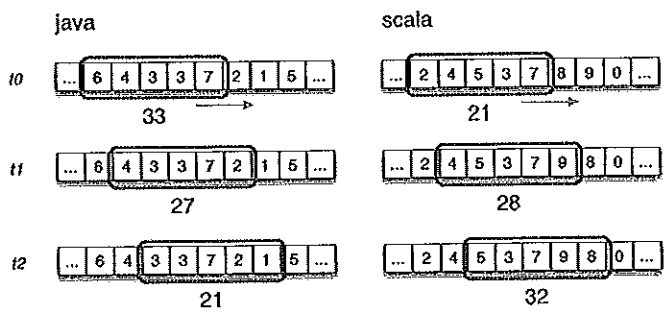


Figure 6: As the sliding window advances, the slice of its input data changes. In the example above the algorithm uses the current sliding window data to compute the sum of the window's elements.

From size to time: If the window is advanced with time, say every *N* minutes, then the individual elements in the input represent data collected over the same interval of time (here: *N* minutes). In that case the window size is equivalent to *N x m* minutes. Simply speaking, if *N=1* and *m=5*, then our sliding window algorithm emits the latest five-minute aggregates every one minute.

Here by using RollingCount Algorithm, we are overcoming the problem that could be caused following the naive approach. The naive way could have been to keep adding the count, on the sliding window, for every new element i.e. for every new time interval and perform addition of all the elements on the sliding window for that time frame. Here we could run short of memory at some point if we keep adding elements on the window, also by adding all k elements of the window every time may hurt the performance.

With the Rolling Count Algorithm (circular queue), every time we add a new element to the window, we only subtract the first element on the window and add the last element, that was newly added, to perform addition. By doing this, we reduce k additions into one addition and one subtraction thus highly increasing the performance. Also RollingCount Algorithm is a circular queue algorithm, thus we can save a lot of memory.

D. Project Topology:

The Topology followed while building the code is as shown:

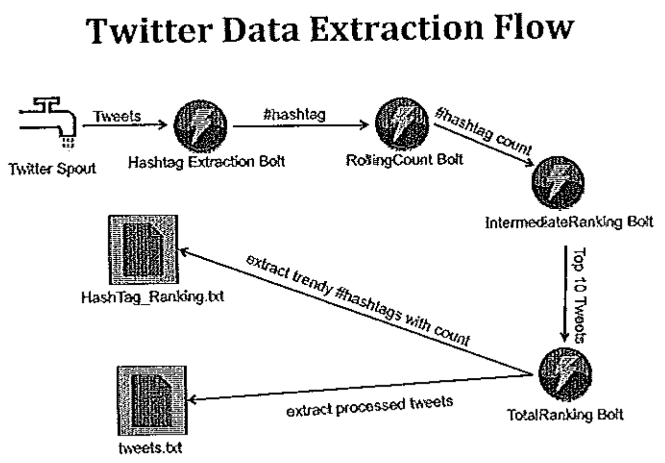


Figure7: Project Topology

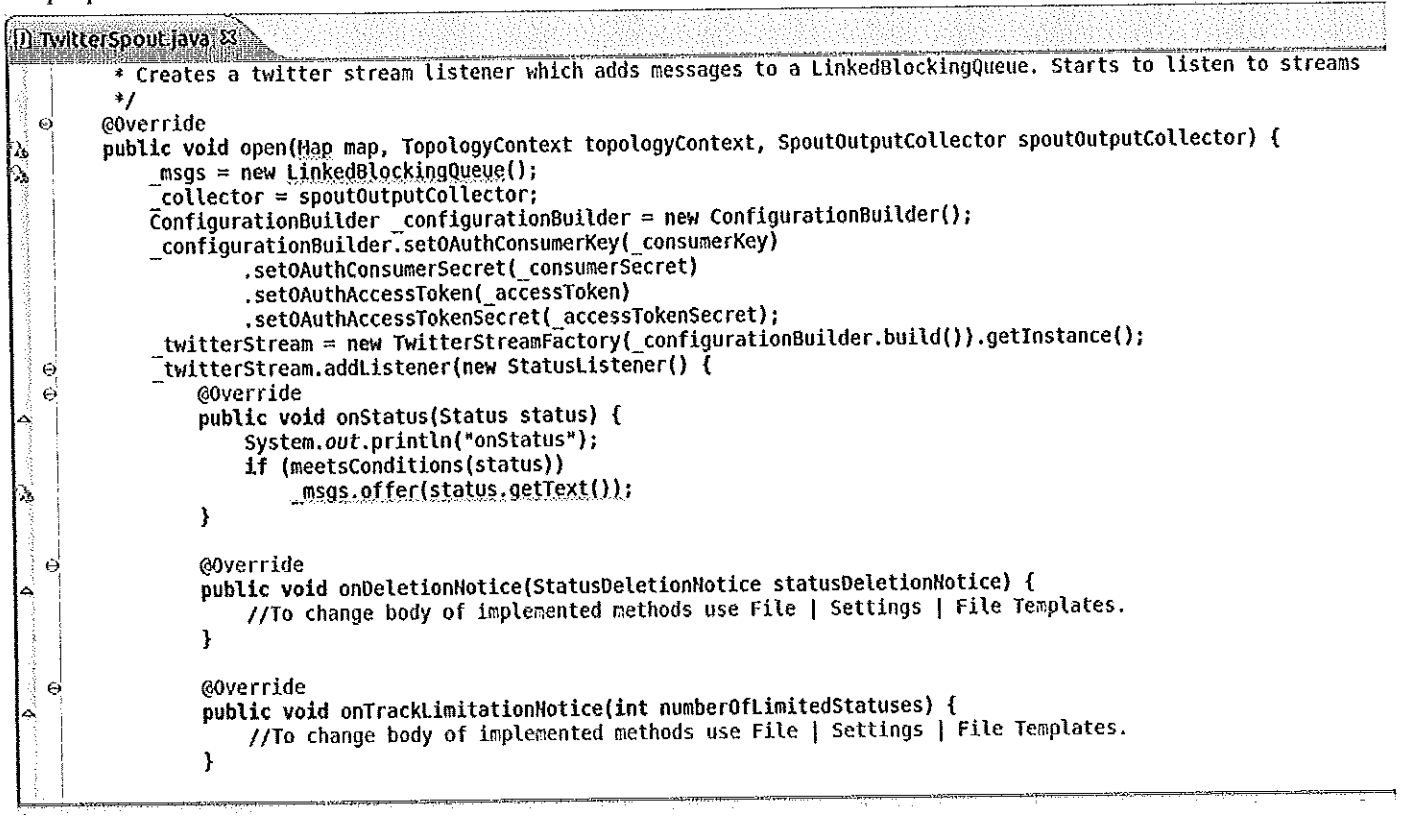
Topology Explanation:

We use the Twitter Spout to extract the live tweets from Twitter. We make use of twitter4j library here to make use of Twitter API to fetch the live tweets. StreamFactory is used to periodically get instances from twitter and is now passed to Hashtag Extraction Bolt from here. The Hashtag Extraction is now ready to fetch the tuples flowing from our Twitter Spout. This Bolt will only remove #hashtags and forward it to the RollingCount Bolt. Here the words are counted, per time interval i.e. 10secs for our project, and are stored temporarily on the sliding window. The concept of sliding key was explained in the introduction section of this document. The counts are then passed onto the Intermediate Ranking Bolt where the counts of the trendy topics are sorted in descending order and these are now send to the TotalRanking Bolt. The Total Ranking Bolt will receive many trendy topics from the Intermediate Ranking Bolt but this will only pass the Top 10 trendy topics to the output file. We carry out different operations on different Bolts because we could want things to happen in parallel. For now things may seem to be flowing sequentially but if there is an enhancement on this existing project, it would be nice to have the functionalities separated on each of the Bolts (i.e. following Single Responsibility Principle). We also got some unexpected errors when we had not implemented the project via Single Responsibility Project[7], thus it would be better to have the principle followed for proper functioning of the project.

As seen. there are two output files being generated namely tweets.txt and HashTag_Ranking.txt. HashTag_Ranking file will store the top 10 trendy topics which were passed from TotalRanking Bolt whereas tweets.txt will store the tweets that were processed for this time interval. We noticed that the location currently is worldwide and thus we are not able to get high counts for each topic. The output shown on the HashTag_Ranking next page will show this clearly. Time could also be a factor as we are only considering 10secs window. To do a better job at analyzing what's going on, we would suggest to use this project with a time interval of at least 30secs and having a small geographical location selected. Another option could be to extract the data when a popular TV show is running or on the day of release of a new movie or any other festival or occasion.

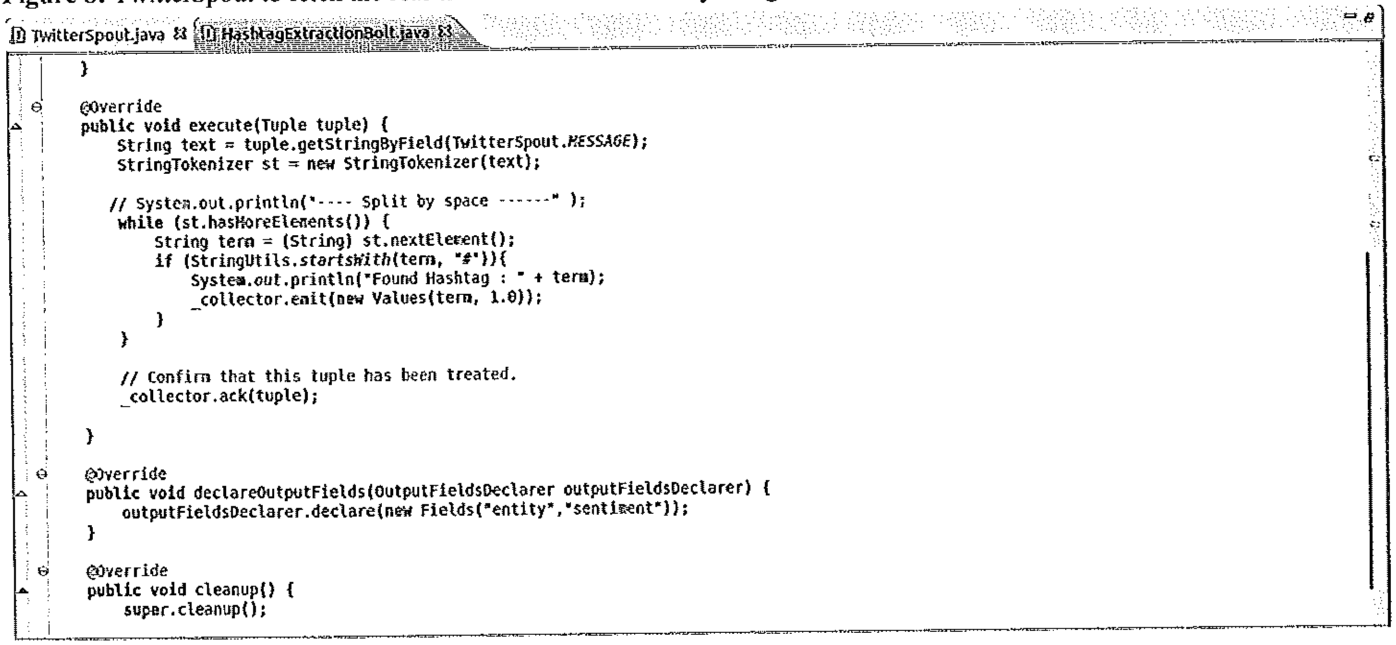
E. Code:

Sample piece of code to extract data from Twitter into TwitterSpout and then from TwitterSpout to HashtagExtractionBolt:



```
     * Creates a twitter stream listener which adds messages to a LinkedBlockingQueue. Starts to listen to streams
     */
    @Override
    public void open(Map map, TopologyContext topologyContext, SpoutOutputCollector spoutOutputCollector) {
        _msgs = new LinkedBlockingQueue();
        _collector = spoutOutputCollector;
        ConfigurationBuilder _configurationBuilder = new ConfigurationBuilder();
        _configurationBuilder.setOAuthConsumerKey(_consumerKey)
                .setOAuthConsumerSecret(_consumerSecret)
                .setOAuthAccessToken(_accessToken)
                .setOAuthAccessTokenSecret(_accessTokenSecret);
        _twitterStream = new TwitterStreamFactory(_configurationBuilder.build()).getInstance();
        _twitterStream.addListener(new StatusListener() {
            @Override
            public void onStatus(Status status) {
                System.out.println("onStatus");
                if (meetsConditions(status))
                    _msgs.offer(status.getText());
            }

            @Override
            public void onDeletionNotice(StatusDeletionNotice statusDeletionNotice) {
                //To change body of implemented methods use File | Settings | File Templates.
            }

            @Override
            public void onTrackLimitationNotice(int numberOfLimitedStatuses) {
                //To change body of implemented methods use File | Settings | File Templates.
            }
```

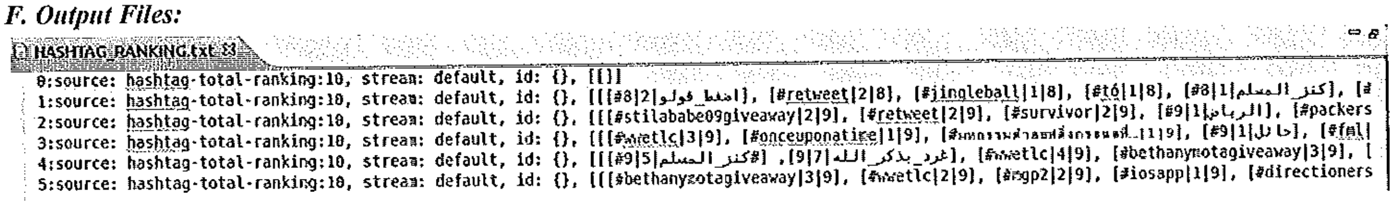
Figure 8: TwitterSpout to fetch the real-time data from Twitter by using Twitter API



```
    }

    @Override
    public void execute(Tuple tuple) {
        String text = tuple.getStringByField(TwitterSpout.MESSAGE);
        StringTokenizer st = new StringTokenizer(text);

       // System.out.println("---- Split by space ------" );
        while (st.hasMoreElements()) {
            String term = (String) st.nextElement();
            if (StringUtils.startsWith(term, "#")){
                System.out.println("Found Hashtag : " + term);
                _collector.emit(new Values(term, 1.0));
            }
        }

        // Confirm that this tuple has been treated.
        _collector.ack(tuple);

    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
        outputFieldsDeclarer.declare(new Fields("entity","sentiment"));
    }

    @Override
    public void cleanup() {
        super.cleanup();
```

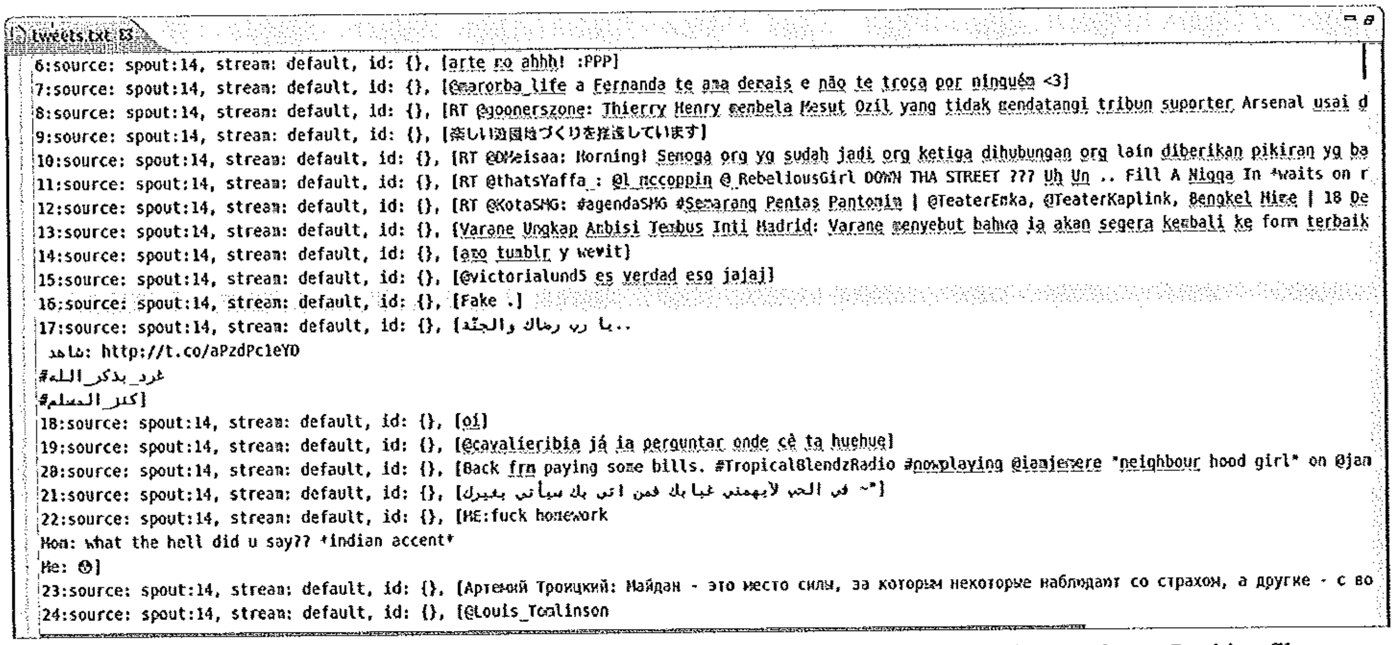
Figure 9: Hashtag Extraction Bolt to remove #hashtags from the words

F. Output Files:



```
0:source: hashtag-total-ranking:10, stream: default, id: {}, [[]]
1:source: hashtag-total-ranking:10, stream: default, id: {}, [[[#8|2|اسلط فولو], [#retweet|2|8], [#jingleball|1|8], [#tó|1|8], [#8|1|كنز المسلم], [#
2:source: hashtag-total-ranking:10, stream: default, id: {}, [[[#stilababe09giveaway|2|9], [#retweet|2|9], [#survivor|2|9], [#9|1|الرياض], [#packers
3:source: hashtag-total-ranking:10, stream: default, id: {}, [[[#wwetlc|3|9], [#onceuponatime|1|9], [#[illegible]|1|9], [#9|1|حايل], [#fml|
4:source: hashtag-total-ranking:10, stream: default, id: {}, [[[#9|5|#كنز المسلم] ,[9|7|غرد بذكر الله], [#wwetlc|4|9], [#bethanymotagiveaway|3|9], [
5:source: hashtag-total-ranking:10, stream: default, id: {}, [[[#bethanymotagiveaway|3|9], [#wwetlc|2|9], [#mgp2|2|9], [#iosapp|1|9], [#directioners
```

Figure 10: HashTag_Ranking showing the trendy topics for a given time interval



```
6:source: spout:14, stream: default, id: {}, [arte ro ahhh! :PPP]
7:source: spout:14, stream: default, id: {}, [@marorba_life a Fernanda te ama demais e não te troca por ninguém <3]
8:source: spout:14, stream: default, id: {}, [RT @goonerszone: Thierry Henry membela Mesut Ozil yang tidak mendatangi tribun suporter Arsenal usai d
9:source: spout:14, stream: default, id: {}, [楽しい遊園地づくりを推進しています]
10:source: spout:14, stream: default, id: {}, [RT @DMeisaa: Morning! Semoga org yg sudah jadi org ketiga dihubungan org lain diberikan pikiran yg ba
11:source: spout:14, stream: default, id: {}, [RT @thatsYaffa_: @l_mccoppin @_RebeliousGirl DOWN THA STREET ??? Uh Un .. Fill A Nigga In *waits on r
12:source: spout:14, stream: default, id: {}, [RT @KotaSMG: #agendaSMG #Semarang Pentas Pantomim | @TeaterEmka, @TeaterKaplink, Bengkel Mime | 18 De
13:source: spout:14, stream: default, id: {}, [Varane Ungkap Ambisi Tembus Inti Madrid: Varane menyebut bahwa ia akan segera kembali ke form terbaik
14:source: spout:14, stream: default, id: {}, [amo tumblr y wevit]
15:source: spout:14, stream: default, id: {}, [@victorialund5 es verdad eso jajaj]
16:source: spout:14, stream: default, id: {}, [Fake .]
17:source: spout:14, stream: default, id: {}, [يا رب رماك والجند..
 شاهد: http://t.co/aPzdPc1eYD
#غرد_بذكر_الله
#اكثر_المسلم]
18:source: spout:14, stream: default, id: {}, [oi]
19:source: spout:14, stream: default, id: {}, [@cavalieribia já ia perguntar onde cê ta huehue]
20:source: spout:14, stream: default, id: {}, [Back frm paying some bills. #TropicalBlendzRadio #nowplaying @iamjemere "neighbour hood girl" on @jam
21:source: spout:14, stream: default, id: {}, [في الحب لايهمني غيابك فمن اتى بك سياتي بغيرك ~*]
22:source: spout:14, stream: default, id: {}, [ME:fuck homework
Mom: what the hell did u say?? *indian accent*
Me: ☹]
23:source: spout:14, stream: default, id: {}, [Артемий Троицкий: Майдан - это место силы, за которым некоторые наблюдают со страхом, а другие - с во
24:source: spout:14, stream: default, id: {}, [@Louis_Tomlinson
```

Figure 11: tweets.txt showing the processed tweets, the trendiness for which can be examined on HashTag_Ranking file

G. Learning: On this project we learnt how to use Storm to extract real-time data from Twitter. The Spout and Bolt Topology that could be used for processing the extracted data. Minor enhancements on this project could be to store the extracted data on some database than on a text file. Our aim was to learn more on the data mining side than storage and thus we did not worry much about storage. The starter project is a good way to begin as it already takes care of many hidden loop holes.

REFERENCES

[1] http://en.wikipedia.org/wiki/Twitter#cite_note-10
[2] https://blog.twitter.com/2013/new-tweets-per-second-record-and-how
[3] http://www.michael-noll.com/blog/2013/01/18/implementing-real-time-trending-topics-in-storm/
[4] Lecture slides
[5] https://github.com/nathanmarz/storm-starter
[6] http://twitter4j.org/en/index.html
[7] http://en.wikipedia.org/wiki/Single_responsibility_principle