COMP529: Streaming Analytics Assignment - Analysing Flu Tweets using Storm

Frances Crouch

Background

A large surge in flu can put pressure on NHS resources (hospital beds, nurses and doctors), possibly leading to inadequate patient care. Predicting the number of people who will require medical assistance due to flu is a not a trivial task, due to the unpredictable nature of flu outbreaks and its seasonality. Note the well known example of Google Flu. Even with the billions of search terms Google have access to, they have not been able to build a model which successfully predicts flu outbreaks in America.

Currently, Public Health England collect data on flu occurrences, however, there is a significant time lag from the data being recorded, to it being available for use. Social media could be used to provide real-time data on people suffering from flu, or flu like symptoms. This assignment demonstrates how Storm can be used to analyse a live Twitter stream to indicate the rate of occurrence of flu in the UK.

Data Analytic Design

Assume a resource manager will review hospital resources at least on a daily basis. Storm can be used to populate a dashboard detailing the number of flu tweets created in a particular region and how this rate has changed over time. This could be compared against current estimates and allocated resources.

Storm Topology

The topology used consists of: a single spout linked to Twitter, a bolt which filters out any tweets not considered to be relevant and checks the location of the tweet, a bolt which counts the number of tweets and two printing bolts: one to record the count data and one to collect the tweets for later analysis. See Appendix A for diagram.

Note: the initial design was to filter tweets by location such that data was collected for a single area in the UK (London for example), as NHS resources are controlled regionally. However, initial trials showed that only a minority of users allow access to their geo data. Therefore, to obtain a meaningful amount of data the scope was extended to include all tweets, irrespective of whether they had geo data.

Twitter Spout - The Twitter feed can be filtered to only allow through tweets containing keywords. This is submitted in the code via a comma delimited list of words. If two words are included in a single comma space, both words must be present in a tweet (e.g. an AND search).

Using the single word "flu" generates many tweets not considered to be relevant to this analysis, for example, adverts (see Appendix B); or will include tweets which contain the letters "f-l-u" within another word, for example, fluid. To avoid such tweets, the list used only included pairs of words (see line 167 of Appendix M).

This list was generated by experimenting with different pairs of words. For example, the list initially included more pairs containing the word "cold". However, this often generated unrelated tweets (because cold was used with reference to the temperature), see Appendix C for an example result of searching "cold bed".

It was also noted that many tweets referred to "stomach flu". This raised the question: what does the word "flu" encompass? Here, it is assumed that any complaint of flu like symptoms or generally feeling unwell could result in the need for medical attention, and hence will be included in the count.

Finally, the spout obtains any location data associated with the tweet. This is either: the coordinates taken from the device used to send the tweet or the users' location as per their settings.

Category Bolt - Tweets are filtered to exclude any adverts or other irrelevant items (referred to as spam). The program checks the tweet against a list of keywords deemed to relate to spam tweets (see line 53 of Appendix N). If the tweet contains one of these words, it is put into the `SPAM` category.

Again, this list was generated by trialing the program and including any words commonly associated with spam tweets. The most common and obvious words were "vaccine" and "shot". A less obvious example was the word "dog". There are several examples of adverts relating to pet vaccinations, see Appendix D.

If the tweet is deemed to be a genuine flu tweet, it is checked for geo data. If no data could be obtained, it is categorised as `NO_INFO`. Where geo data was available, the tweet is categorised as: `LONDON`, `REST_OF_UK` or `OUTSIDE_UK`.

Count Bolt - Initially, a simple counting bolt was used (Appendix O). This would suffice if the output was recorded each day and the program reset. However, the final topology used has a rolling count (Appendix P) which outputs the total count for a defined window of time, at a given frequency. So for example, this could feed into a dashboard to give the flu count for the past 2 days, every day.

Note: unlike the other connections in the topology, a field grouping is used on the count bolt. This ensures that all tweets of the same category are counted by one instance of the bolt and hence guarantees the global total is calculated and not just several local totals.

Middleware Configuration

For these experiments Storm was set up in local mode with a parallelism of just 1 thread per spout or bolt. The rationale for this is: there is just a single computer available (not a cluster), it is easier to debug a program in local mode and the data stream is sufficiently small to be run on a single computer.

If a similar analysis were to be performed on a larger scale, Storm could be set up in distributed mode and the parallelism could be set as greater than 1, to enable more than 1 instance of a bolt or spout (especially on those which are computationally heavy).

Results
The topology was run for a total of 5 hours using the rolling count set to count tweets over an hour, every 30 minutes - see Appendix F for the output of the rolling count. The two tweet output files (the "SPAM" and "tweets" txt files created by the Tweet Printer Bolt - see Appendix Q) were reviewed for accuracy. For each file, 1 in every 50 tweets were randomly selected to be reviewed, to confirm whether they were correctly classified - see Appendix E for a table of the results.

Discussion of Results
On the whole the filtering approach seems effective at removing adverts. Appendix G was the only example of an advert not detected by the filter. However, similar tweets could easily be removed in the future by including "win" in the filter list.

The more interesting and difficult task is removing tweets where people (not companies) are just commenting about flu when they don't actually have flu, see Appendix H as an example. Note here the use of an emoji. Analysing the "emotions" in a tweet could be a way to differentiate between someone tweeting about flu because they have flu, and someone just commenting about flu.

Another example of an incorrectly classified tweet can be seen in Appendix I. This was categorised as spam because the word "jab" was used. Perhaps this could have been avoided by noticing that both "my" and "I" were present in the tweet, which is very unusual for most spam tweets.

Appendix K shows a graph of the rolling count data. This shows how very few tweets have geo data attached to them - only around 5%.

Limitations of the approach and context of the question
The key sections of society who are most likely to need medical attention due to contracting flu are the old and young. Twitter's demographic is largely people aged 20 - 30, and hence the results would need to factor for this. However, it was noted that many people tweet about their children having flu, see Appendix J, so perhaps at least some of this population is being captured.

Also, the program only streamed tweets in English, however, many people in the UK speak languages other than English, especially London.

As already mentioned, there was a real lack of geo data available making it difficult to take measurements at a national level let alone a regional level, as initially planned. Perhaps, the stream could be filtered to exclude any tweets referring to, for example, the name of an American state or city. Another idea could be to only run the program during daytime hours and hence exclude at least some tweets from the major English speaking countries in different time zones.

The program only streamed tweets with the required keywords. It is not possible to quantify the number of genuine flu tweets which were excluded because they did not match the search criteria.

Further Investigations
An interesting question, not covered so far, is: is it better to exclude all spam tweets from the count or to ensure all genuine tweets are included? This is not a trivial question to answer. As stated previously, this program aims to show any extreme increases in flu occurrences, therefore, as long as the number of spam tweets remains proportional to the number of genuine tweets then perhaps the exact balance between the two is not overly important.

As mentioned, the program could be improved by refining the list of words used to filter both the stream of tweets and any spam, and analyse the use of emojis. Perhaps, instead of a binary, in or out approach, categorising tweets could be done on a points based system, where points are given or taken away for different characteristics, for example: the use of "I", the use of unhappy emojis or the use of "vaccine". Then, a tweet is only considered genuine if it reaches a threshold number of points. This could help to deal with conflicting characteristics, such as a genuine tweet using the word "shot".

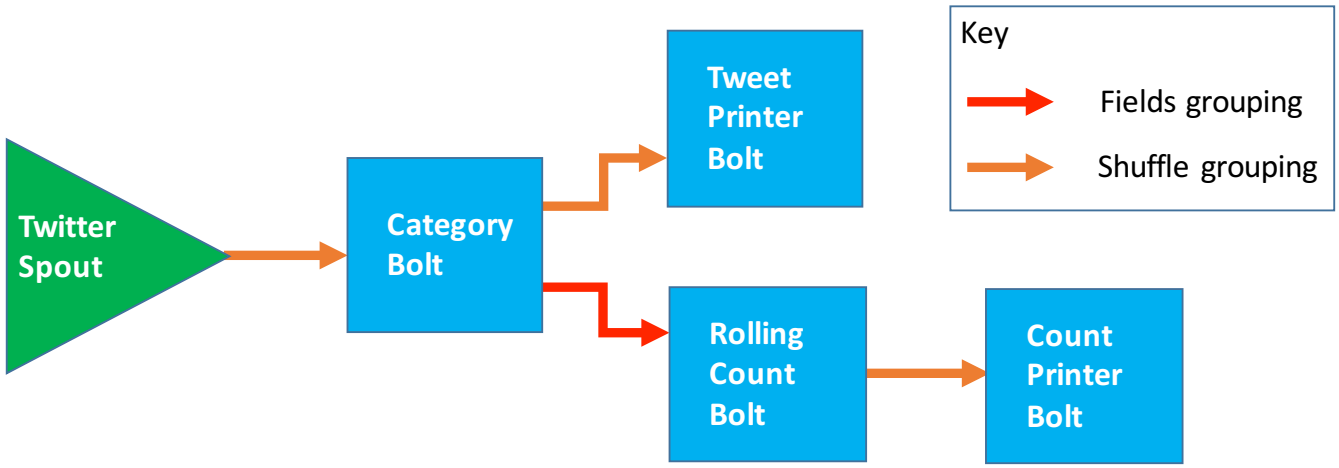
Another way to develop this approach would be to combine this analysis with other data sources, including: Facebook, Instagram and NHS data.

Conclusions and Recommendations
The main issue with this approach is the lack of geo data, making it difficult to infer levels of flu in just the UK. The program was successful in excluding adverts from the count, however, improvements could be make to distinguishing between a genuine flu tweet and someone just mentioning flu in a tweet.

It is proposed that this could be part of a bigger system which includes other data sources and calibration methods. However, the correlation between tweets with geo data and those without geo data would need to be tested.

Appendices

A Topology



B Searching “flu” in Twitter

Top news story

Global Pharma News @pharma_global · 18h

Flu Shots for Hospital Workers Save Lives - Wall Street Journal goo.gl/fb/d5LIKV

Flu Shots for Hospital Workers Save Lives

In The Wall Street Journal, M. Todd Greene and Sanjay Saint writes that flu shots for hospital workers save lives. The Department of Veterans Affairs...

wsj.com

MB Devine @devine_mb · 3m

Tylenol Is Ineffective Against **Flu** Symptoms nyti.ms/1YZR37t via @nytimeswell

View summary

C Searching “cold bed” in Twitter

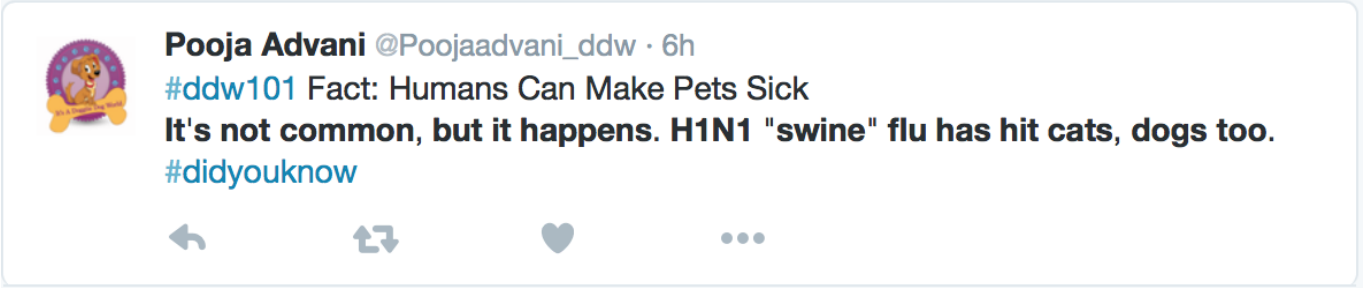
Liked 9,751 times

Ellen DeGeneres @TheEllenShow · Dec 14

I invented a new yoga. The room is **cold**, & you’re in **bed** & you have to put socks on while staying under the covers. It's called Fro Yoga.

3.2K 9.8K

D Example of SPAM tweet containing the word “dog”



E Results - Summary

Results	Total number of tweets in output	Size of sample for checking (1 in 50)	Number incorrectly classified	% incorrectly classified	Estimated number of genuine tweets
Flu tweets	797	16	4	25%	597
SPAM tweets	197	4	1	25%	50
					647

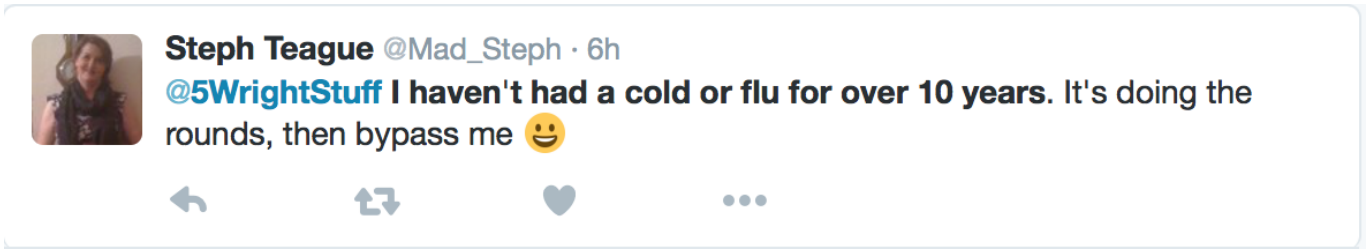
F Results - Rolling Count Output

Rolling count

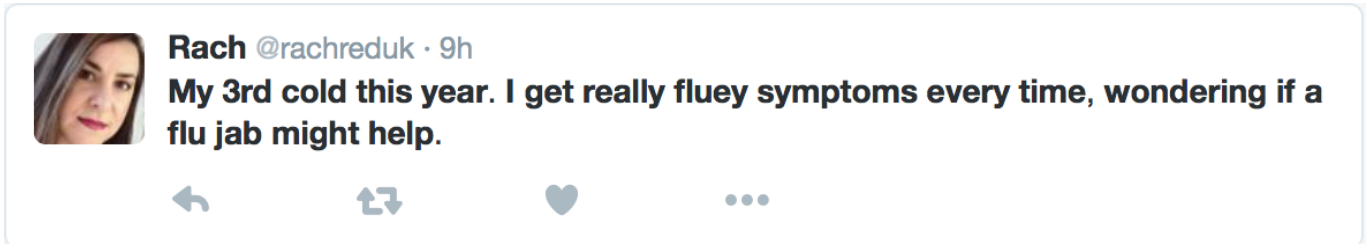
```
Count for the past 30 minutes at16/12/2015 08:36:48: OUTSIDE_UK = 2
Count for the past 30 minutes at16/12/2015 08:36:48: REST_OF_UK = 3
Count for the past 30 minutes at16/12/2015 08:36:48: NO_INFO = 72
Count for the past 30 minutes at16/12/2015 08:36:48: SPAM = 17
Count for the past 60 minutes at16/12/2015 09:06:48: OUTSIDE_UK = 3
Count for the past 60 minutes at16/12/2015 09:06:48: REST_OF_UK = 4
Count for the past 60 minutes at16/12/2015 09:06:48: NO_INFO = 121
Count for the past 60 minutes at16/12/2015 09:06:48: SPAM = 53
Count for the past 60 minutes at16/12/2015 09:36:48: OUTSIDE_UK = 2
Count for the past 60 minutes at16/12/2015 09:36:48: REST_OF_UK = 2
Count for the past 60 minutes at16/12/2015 09:36:48: NO_INFO = 98
Count for the past 60 minutes at16/12/2015 09:36:48: SPAM = 52
Count for the past 60 minutes at16/12/2015 10:06:48: OUTSIDE_UK = 5
Count for the past 60 minutes at16/12/2015 10:06:48: REST_OF_UK = 2
Count for the past 60 minutes at16/12/2015 10:06:48: LONDON = 1
Count for the past 60 minutes at16/12/2015 10:06:48: NO_INFO = 90
Count for the past 60 minutes at16/12/2015 10:06:48: SPAM = 26
Count for the past 60 minutes at16/12/2015 10:36:48: OUTSIDE_UK = 5
Count for the past 60 minutes at16/12/2015 10:36:48: REST_OF_UK = 2
Count for the past 60 minutes at16/12/2015 10:36:48: LONDON = 2
Count for the past 60 minutes at16/12/2015 10:36:48: NO_INFO = 96
Count for the past 60 minutes at16/12/2015 10:36:48: SPAM = 23
Count for the past 60 minutes at16/12/2015 11:06:48: OUTSIDE_UK = 3
Count for the past 60 minutes at16/12/2015 11:06:48: REST_OF_UK = 3
Count for the past 60 minutes at16/12/2015 11:06:48: LONDON = 1
Count for the past 60 minutes at16/12/2015 11:06:48: NO_INFO = 113
Count for the past 60 minutes at16/12/2015 11:06:48: SPAM = 25
Count for the past 60 minutes at16/12/2015 11:36:48: OUTSIDE_UK = 7
Count for the past 60 minutes at16/12/2015 11:36:48: REST_OF_UK = 4
Count for the past 60 minutes at16/12/2015 11:36:48: LONDON = 0
Count for the past 60 minutes at16/12/2015 11:36:48: NO_INFO = 124
Count for the past 60 minutes at16/12/2015 11:36:48: SPAM = 29
Count for the past 60 minutes at16/12/2015 12:06:48: OUTSIDE_UK = 9
Count for the past 60 minutes at16/12/2015 12:06:48: REST_OF_UK = 2
Count for the past 60 minutes at16/12/2015 12:06:48: NO_INFO = 143
Count for the past 60 minutes at16/12/2015 12:06:48: SPAM = 37
Count for the past 60 minutes at16/12/2015 12:36:48: OUTSIDE_UK = 6
Count for the past 60 minutes at16/12/2015 12:36:48: REST_OF_UK = 0
Count for the past 60 minutes at16/12/2015 12:36:48: NO_INFO = 147
Count for the past 60 minutes at16/12/2015 12:36:48: SPAM = 38
Count for the past 60 minutes at16/12/2015 13:06:48: OUTSIDE_UK = 3
Count for the past 60 minutes at16/12/2015 13:06:48: REST_OF_UK = 1
Count for the past 60 minutes at16/12/2015 13:06:48: NO_INFO = 163
Count for the past 60 minutes at16/12/2015 13:06:48: SPAM = 35
Count for the past 60 minutes at16/12/2015 13:36:48: OUTSIDE_UK = 6
Count for the past 60 minutes at16/12/2015 13:36:48: REST_OF_UK = 1
Count for the past 60 minutes at16/12/2015 13:36:48: NO_INFO = 191
Count for the past 60 minutes at16/12/2015 13:36:48: SPAM = 31
```

G Advert miscategorised as genuine tweet

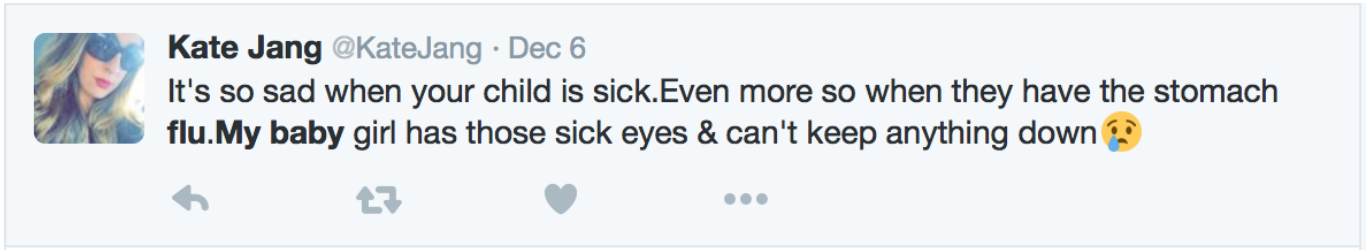
H Example of tweet which doesn't indicate someone having an illness



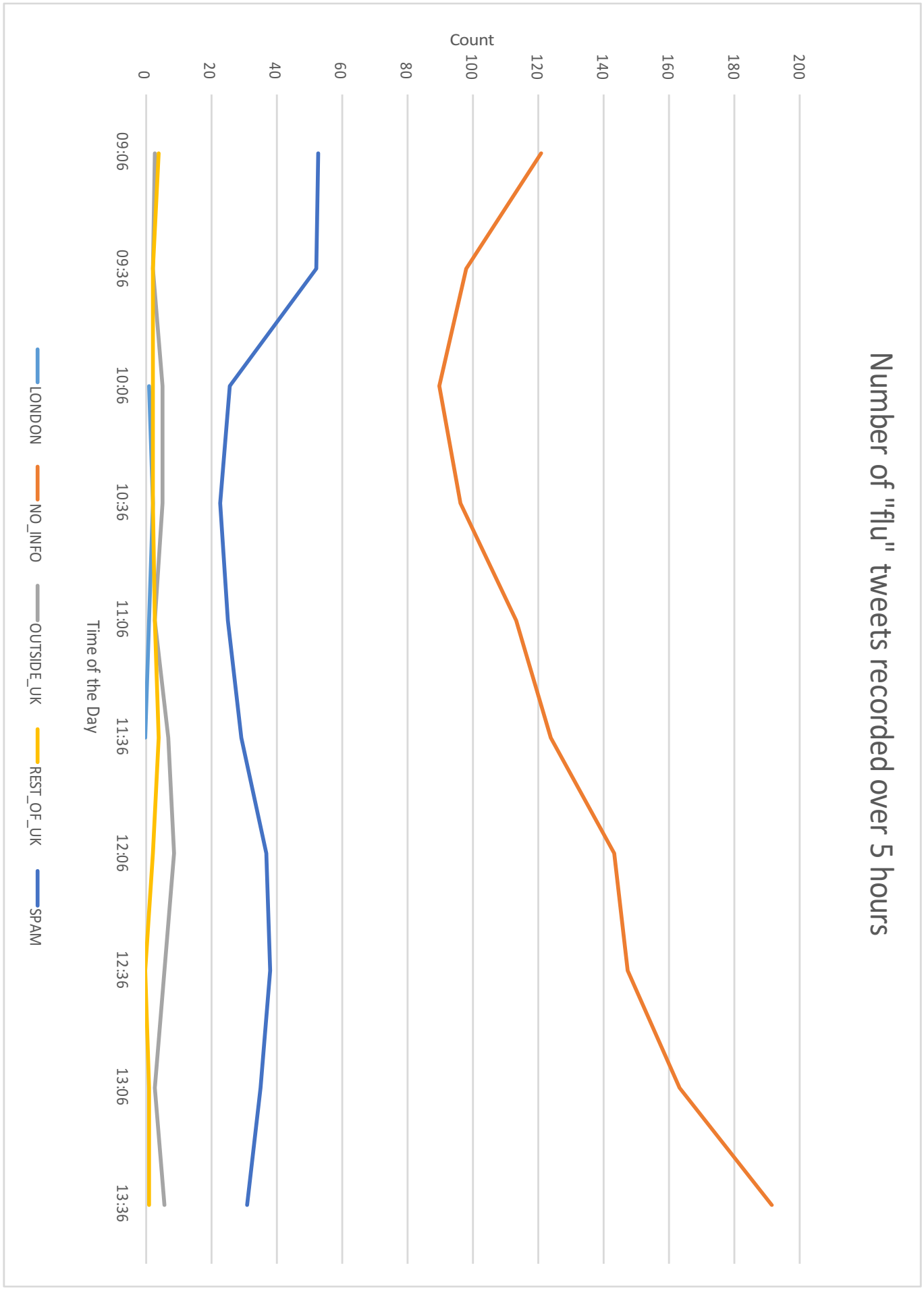
I Example of genuine flu tweet miscategorised as SPAM



J Example of tweet talking about a child having flu



K Graph of all results



L Topology

The Java code used for this analysis was based on that used by the "Real-Time Analytics with Apache Storm" online course (see webpage - `https://www.udacity.com/course/viewer#!/c-ud381/l-2731858540/m-3382678706`).

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

import backtype.storm.Config;
import backtype.storm.LocalCluster;
import backtype.storm.StormSubmitter;
import backtype.storm.spout.SpoutOutputCollector;
import backtype.storm.task.OutputCollector;
import backtype.storm.task.TopologyContext;
import backtype.storm.testing.TestWordSpout;
import backtype.storm.topology.BasicOutputCollector;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.TopologyBuilder;
import backtype.storm.topology.base.BaseBasicBolt;
import backtype.storm.topology.base.BaseRichSpout;
import backtype.storm.topology.base.BaseRichBolt;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Tuple;
import backtype.storm.tuple.Values;
import backtype.storm.utils.Utils;
import backtype.storm.topology.base.BaseRichBolt;



class TweetTopology
{

  public static void main(String[] args) throws Exception
  {
    // create the topology
    TopologyBuilder builder = new TopologyBuilder();

    //create the tweet spout with the credentials
    TweetSpout tweetSpout = new TweetSpout(
        "QTrVS64B4HU2TmNXAVwnZS9Bk",
        "gmlPIJCWoIEuvB8sTX7VZcXaKxZOuStA7JGv05lOaldXcIJuNW",
        "1469229614-EbAzs0awEM5d6cMmvs44l9DLTDTGyzhYE7NZY0q",
        "fCv3a7fVrCVlk2ga8E5OQTVSLZvIuusKPkq4WMAFAFsYr");
        //"[Your customer key]",
        //"[Your secret key]",
        //"[Your access token]",
        //"[Your access secret]"
```

Topology continued

```
    //****build the topology***********

    //Twitter Spout
    builder.setSpout("tweet-spout", tweetSpout, 1);

    // Twitter Spout ===>> Category Bolt
    builder.setBolt("category-bolt", new CategoryBolt()).shuffleGrouping("tweet-spout");
    //  Category Bolt ===>> Tweet Printer
    builder.setBolt("print-tweets", new TweetPrinterBolt()).shuffleGrouping("category-bolt");

    //code to include the stright counting bolt - uncomment to use
    //builder.setBolt("count-bolt", new CountBolt()).fieldsGrouping("category-bolt", new Fields("category"));
    //builder.setBolt("print-count", new CountPrinterBolt("counts-print")).shuffleGrouping("count-bolt");

    //Category Bolt ===>> Rolling Counter Bolt  **uses a fields grouping**
    builder.setBolt("rolling-count-bolt", new RollingCountBolt(3600,1800)).fieldsGrouping("category-bolt", new Fields("category"));
    //Rolling Counter Bolt ===>> Count Printer
    builder.setBolt("print-count", new CountPrinterBolt("rolling-counts-print")).shuffleGrouping("rolling-count-bolt");


    // Submit and run the topology

    // create the default config object
    Config conf = new Config();

    // set the config in debugging mode
    conf.setDebug(true);

      // run it in a simulated local cluster
      // set the number of threads to run - similar to setting number of workers in live cluster
      conf.setMaxTaskParallelism(3);

      // create the local cluster instance
      LocalCluster cluster = new LocalCluster();

      // submit the topology to the local cluster
      cluster.submitTopology("tweet-flu-count", conf, builder.createTopology());

      // let the topology run for 5 hours. note topologies never terminate!
      Utils.sleep(1800000);

      // now kill the topology
      cluster.killTopology("tweet-word-count");

      // we are done, so shutdown the local cluster
      cluster.shutdown();
  }//end of main
}//end of class
```

M Tweet Spout

```
37  public class TweetSpout extends BaseRichSpout
38  {
39    // Twitter API authentication credentials
40    String custkey, custsecret;
41    String accesstoken, accesssecret;
42
43    // To output tuples from spout to the next stage bolt
44    SpoutOutputCollector collector;
45
46    // Twitter stream to get tweets
47    TwitterStream twitterStream;
48
49    // Shared queue for getting buffering tweets received
50    LinkedBlockingQueue<String> queue = null;
51
52    // Class for listening on the tweet stream - for twitter4j
53    private class TweetListener implements StatusListener {
54
55      // Implement the callback function when a tweet arrives
56      @Override
57      public void onStatus(Status status)
58      {
59        //here we get the actual text from the tweet
60        //we also look for any geodata attached to the tweet
61        //the tweet and geodata into the queue buffer ready for the bolts
62
63        //check if the tweet has lat and long coordinates
64        //add the tweet and coordinates to the queue buffer
65        if(status.getGeoLocation() != null)
66        {
67          String coordinates = String.valueOf(status.getGeoLocation().getLatitude()) + "," +
68                               String.valueOf(status.getGeoLocation().getLongitude());
69
70          queue.offer(status.getText() + "DELIMITER" +
71                      "coordinates" + "DELIMITER" + coordinates);
72        }
73
74        //if no coordinate, check to see it the users location is listed
75        //add the tweet and place to the queue buffer
76        else if(status.getPlace() != null)
77        {
78          String place = String.valueOf(status.getPlace().getCountryCode()) + "," +
79                         String.valueOf(status.getPlace().getFullName());
80          queue.offer(status.getText() + "DELIMITER" + "place" + "DELIMITER" + place);
81        }
82
83        //else just sent the tweet
84        else {
85          queue.offer(status.getText() + "DELIMITER" + "noGeoInfo");
86        }
87      }//end of onStatus method
88
89      @Override
90      public void onDeletionNotice(StatusDeletionNotice sdn)
91      {
92      }
93
94      @Override
95      public void onTrackLimitationNotice(int i)
96      {
97      }
```

Tweet Spout continued

```
    @Override
    public void onScrubGeo(long l, long l1)
    {
    }

    @Override
    public void onStallWarning(StallWarning warning)
    {
    }

    @Override
    public void onException(Exception e)
    {
      e.printStackTrace();
    }
  }//end of the TweetListener class


  //Constructor for tweet spout that accepts the credentials

  public TweetSpout(
      String                key,
      String                secret,
      String                token,
      String                tokensecret)
  {
    custkey = key;
    custsecret = secret;
    accesstoken = token;
    accesssecret = tokensecret;
  }

  @Override
  public void open(
      Map                   map,
      TopologyContext       topologyContext,
      SpoutOutputCollector  spoutOutputCollector)
  {
    // create the buffer to block tweets
    queue = new LinkedBlockingQueue<String>(1000);

    // save the output collector for emitting tuples
    collector = spoutOutputCollector;


    // build the config with credentials for twitter 4j
    ConfigurationBuilder config =
        new ConfigurationBuilder()
            .setOAuthConsumerKey(custkey)
            .setOAuthConsumerSecret(custsecret)
            .setOAuthAccessToken(accesstoken)
            .setOAuthAccessTokenSecret(accesssecret);

    // create the twitter stream factory with the config
    TwitterStreamFactory fact =
        new TwitterStreamFactory(config.build());

    // get an instance of twitter stream
    twitterStream = fact.getInstance();

```

Tweet Spout continued

```
159     // provide the handler for twitter stream
160     twitterStream.addListener(new TweetListener());
161
162     //create a filter query
163     //only get tweets that are in English and contain certain pairs of words
164     FilterQuery tweetFilterQuery = new FilterQuery();
165
166     tweetFilterQuery.track(new String[]{
167       "flu has, flu have, flu poor, flu duvet,flu bed, " +
168       "flu sofa, flu cough, flu fever, flu sneeze, flu feel" +
169       "flu cold, flu i, flu me, flu man, flu throat," +
170       "cold cough, cold fever, cold sneeze"});
171
172     tweetFilterQuery.language(new String[]{"en"});
173
174     //applies the filter to the stream
175     twitterStream.filter(tweetFilterQuery);
176
177   }
178
179   @Override
180   public void nextTuple()
181   {
182     // try to pick a tweet from the buffer
183     String ret = queue.poll();
184
185     // if no tweet is available, wait for 50 ms and return
186     if (ret==null)
187     {
188       Utils.sleep(50);
189       return;
190     }
191
192     // now emit the tweet to next stage bolt
193     collector.emit(new Values(ret));
194   }
195
196   @Override
197   public void close()
198   {
199     // shutdown the stream - when we are going to exit
200     twitterStream.shutdown();
201   }
202
203
204    // Component specific configuration
205
206   @Override
207   public Map<String, Object> getComponentConfiguration()
208   {
209     // create the component config
210     Config ret = new Config();
211
212     // set the parallelism for this spout to be 1
213     ret.setMaxTaskParallelism(1);
214
215     return ret;
216   }
```

Tweet Spout continued

```
218     @Override
219     public void declareOutputFields(
220             OutputFieldsDeclarer outputFieldsDeclarer)
221     {
222       // tell storm the schema of the output tuple for this spout
223       // tuple consists of a single column called 'tweet_with_category'
224       outputFieldsDeclarer.declare(new Fields("tweet_with_category"));
225     }
226 }//end of class
227
```

N Category Bolt

```
24  public class CategoryBolt extends BaseRichBolt
25  {
26    // To output tuples from this bolt to the count bolt
27    OutputCollector collector;
28
29    @Override
30    public void prepare(
31        Map                     map,
32        TopologyContext         topologyContext,
33        OutputCollector         outputCollector)
34    {
35      // save the output collector for emitting tuples
36      collector = outputCollector;
37    }
38
39    @Override
40    public void execute(Tuple tuple)
41    {
42      // get the 'tweet_with_category' and split it by the word DELIMITER and get the tweet
43       String tweet = tuple.getStringByField("tweet_with_category").split("DELIMITER")[0];
44
45      // provide the delimiters for splitting the tweet
46      String delims = "[ .,?!]+";
47
48      // now split the tweet into words
49      //to be able to iterate through each word of the tweet
50      String[] words = tweet.split(delims);
51
52      //create a list of SPAM words
53      String spam_words = "vaccine,shot,seasonal,outbreak,jab,influenza," +
54                          "effect,free,healthy,soothe,remedy,effective," +
55                          "illness,information,dog,symptom,system,protect" +
56                          "you,learn,treat,immune";
57
58      //split into tokens
59      String[] spam_check = spam_words.split(",");
60
61      //initiate the category to null
62      String category = "null";
63
64      // for each word in the tweet
65      mainloop:
66      for (String word: words) {  //iterate through each tweet word
67          for(String spam: spam_check ){ //iterate through each SPAM word
68              if(word.toLowerCase().contains(spam)){
69                  category = "SPAM"; //if there is a match ==> tweet is considered to be SPAM
70                  collector.emit(new Values(category, tweet)); //emit the output
71                  break mainloop;//stop the for loop
72              }//end of if statement
73          }//end of second for statement
74      }//end of first for statement
75
```

Category Bolt continued

```
76      //if the tweet is not considered to be SPAM
77      if(category.equals("null")){
78
79          //find what data is available and categorize as:
80          //NO_INFO, OUTSIDE_UK, LONDON, REST_OF_UK
81          String dataAvailable = tuple.getStringByField("tweet_with_category").split("DELIMITER")[1];
82
83              if(dataAvailable.equals("noGeoInfo")){
84                  category = "NO_INFO";
85                  collector.emit(new Values(category, tweet));
86              }
87
88              else if(dataAvailable.equals("place")){
89                  String country = tuple.getStringByField("tweet_with_category").split("DELIMITER")[2].split(",")[0];
90
91                  //if not in the UK set as outside_uk
92                  if(!country.equals("GB")){
93                      category = "OUTSIDE_UK";
94                      collector.emit(new Values(category, tweet));
95                  }
96
97                  //else check if in London or not
98                  else{
99                      String london_check = tuple.getStringByField("tweet_with_category").split("DELIMITER")[2].split(",")[2];
100                     if(london_check.equals(" London")){
101                         category = "LONDON";
102                         collector.emit(new Values(category, tweet));
103                     }//end of if in london
104
105                     else{
106                         category = "REST_OF_UK";
107                         collector.emit(new Values(category, tweet));
108                     }
109                 }
110             }//end of place section
111
```

Category Bolt continued

```
            if(dataAvailable.equals("coordinates")){
                double latitude =
                Double.parseDouble(tuple.getStringByField("tweet_with_category").split("DELIMITER")[2].split(",")[0]);

                double longitude =
                Double.parseDouble(tuple.getStringByField("tweet_with_category").split("DELIMITER")[2].split(",")[1]);

                //circle around London
                double center_london_lat = 51.5073509;
                double center_london_long = -0.12775829999998223;
                double r_london = 0.22458;

                //circle around the UK
                double center_uk_lat = 55.378051;
                double center_uk_long = -3.43597299999999;
                double r_UK = 5.38987;

                //check if coordinates are in London
                if ((longitude - center_london_long)*(longitude - center_london_long) +
                        (latitude - center_london_lat)*(latitude - center_london_lat) < r_london*r_london){
                    category = "LONDON";
                    collector.emit(new Values(category, tweet));
                }/

                //else check if coordinates are in the UK
                else if ((longitude - center_uk_long)*(longitude - center_uk_long) +
                        (latitude - center_uk_lat)*(latitude - center_uk_lat) < r_UK*r_UK){
                    category = "REST_OF_UK";
                    collector.emit(new Values(category, tweet));
                }

                else{
                    category = "OUTSIDE_UK";
                    collector.emit(new Values(category, tweet));
                }
            }//end of coordinates section
        }//end of category = null statement

    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer)
    {
      // tell storm the schema of the output tuple for this spout
      // tuple consists of two columns called 'category' and 'tweet'
      declarer.declare(new Fields("category", "tweet"));
    }
  }
```

O Count Bolt

```
26  public class CountBolt extends BaseRichBolt
27  {
28    // To output tuples from this bolt to the next stage bolts, if any
29    private OutputCollector collector;
30
31    // Map to store the count of the words
32    private Map<String, Integer> countMap;
33
34    @Override
35    public void prepare(
36        Map                     map,
37        TopologyContext         topologyContext,
38        OutputCollector         outputCollector)
39    {
40
41      // save the collector for emitting tuples
42      collector = outputCollector;
43
44      // create and initialize the map
45      countMap = new HashMap<String, Integer>();
46    }
47
48    @Override
49    public void execute(Tuple tuple)
50    {
51      // get category from the Category Bolt
52      String category = tuple.getStringByField("category");
53
54      // check if the category is present in the map
55      if (countMap.get(category) == null) {
56
57        // not present, add the category with a count of 1
58        countMap.put(category, 1);
59      } else {
60
61        // already there, hence get the count
62        Integer val = countMap.get(category);
63
64        // increment the count and save it to the map
65        countMap.put(category, ++val);
66      }
67
68      // emit the word and count
69      collector.emit(new Values(category, countMap.get(category)));
70    }
71
72
73    @Override
74    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer)
75    {
76      // tell storm the schema of the output tuple for this spout
77      // tuple consists of a two columns called 'category' and 'count'
78      outputFieldsDeclarer.declare(new Fields("category","count"));
79    }
80  }
```

P Rolling Count Bolt

Note: to run this class a further 5 Java classes are required.
See - `https://github.com/udacity/ud381/tree/master/lesson3/stage4/src/jvm/udacity/storm/tools`

The only changes made to this code are in line 117 on the next page.

```
39  public class RollingCountBolt extends BaseRichBolt {
40
41    private static final long serialVersionUID = 5537727428628598519L;
42    private static final Logger LOG = Logger.getLogger(RollingCountBolt.class);
43    private static final int NUM_WINDOW_CHUNKS = 5;
44    private static final int DEFAULT_SLIDING_WINDOW_IN_SECONDS = NUM_WINDOW_CHUNKS * 60;
45    private static final int DEFAULT_EMIT_FREQUENCY_IN_SECONDS = DEFAULT_SLIDING_WINDOW_IN_SECONDS / NUM_WINDOW_CHUNKS;
46    private static final String WINDOW_LENGTH_WARNING_TEMPLATE =
47        "Actual window length is %d seconds when it should be %d seconds"
48            + " (you can safely ignore this warning during the startup phase)";
49
50    private final SlidingWindowCounter<Object> counter;
51    private final int windowLengthInSeconds;
52    private final int emitFrequencyInSeconds;
53    private OutputCollector collector;
54    private NthLastModifiedTimeTracker lastModifiedTracker;
55
56    public RollingCountBolt() {
57      this(DEFAULT_SLIDING_WINDOW_IN_SECONDS, DEFAULT_EMIT_FREQUENCY_IN_SECONDS);
58    }
59
60    public RollingCountBolt(int windowLengthInSeconds, int emitFrequencyInSeconds) {
61      this.windowLengthInSeconds = windowLengthInSeconds;
62      this.emitFrequencyInSeconds = emitFrequencyInSeconds;
63      counter = new SlidingWindowCounter<Object>(deriveNumWindowChunksFrom(this.windowLengthInSeconds,
64          this.emitFrequencyInSeconds));
65    }
66
67    private int deriveNumWindowChunksFrom(int windowLengthInSeconds, int windowUpdateFrequencyInSeconds) {
68      return windowLengthInSeconds / windowUpdateFrequencyInSeconds;
69    }
70
71    @SuppressWarnings("rawtypes")
72    @Override
73    public void prepare(Map stormConf, TopologyContext context, OutputCollector collector) {
74      this.collector = collector;
75      lastModifiedTracker = new NthLastModifiedTimeTracker(deriveNumWindowChunksFrom(this.windowLengthInSeconds,
76          this.emitFrequencyInSeconds));
77    }
78
79    @Override
80    public void execute(Tuple tuple) {
81      if (TupleHelpers.isTickTuple(tuple)) {
82        LOG.debug("Received tick tuple, triggering emit of current window counts");
83        emitCurrentWindowCounts();
84      }
85      else {
86        countObjAndAck(tuple);
87      }
88    }
```

Rolling Count Bolt continued

```
90      private void emitCurrentWindowCounts() {
91        Map<Object, Long> counts = counter.getCountsThenAdvanceWindow();
92        int actualWindowLengthInSeconds = lastModifiedTracker.secondsSinceOldestModification();
93        lastModifiedTracker.markAsModified();
94        if (actualWindowLengthInSeconds != windowLengthInSeconds) {
95          LOG.warn(String.format(WINDOW_LENGTH_WARNING_TEMPLATE, actualWindowLengthInSeconds, windowLengthInSeconds));
96        }
97        emit(counts, actualWindowLengthInSeconds);
98      }
99
100     private void emit(Map<Object, Long> counts, int actualWindowLengthInSeconds) {
101       for (Entry<Object, Long> entry : counts.entrySet()) {
102         Object obj = entry.getKey();
103         Long count = entry.getValue();
104         Integer intCount = count != null ? count.intValue() : null;
105         collector.emit(new Values(obj, intCount, actualWindowLengthInSeconds));
106       }
107     }
108
109     private void countObjAndAck(Tuple tuple) {
110       Object obj = tuple.getValue(0);
111       counter.incrementCount(obj);
112       collector.ack(tuple);
113     }
114
115     @Override //Fields updated for this topology - this is the only change made to the code
116     public void declareOutputFields(OutputFieldsDeclarer declarer) {
117       declarer.declare(new Fields("category", "count", "actualWindowLengthInSeconds"));
118     }
119
120     @Override
121     public Map<String, Object> getComponentConfiguration() {
122       Map<String, Object> conf = new HashMap<String, Object>();
123       conf.put(Config.TOPOLOGY_TICK_TUPLE_FREQ_SECS, emitFrequencyInSeconds);
124       return conf;
125     }
126   }
127
```

Q Tweet Printer Bolt

```
27  public class TweetPrinterBolt extends BaseBasicBolt {
28
29      @Override
30      public void execute(Tuple tuple, BasicOutputCollector collector) {
31
32          //if the tweet is SPAM ==> put in a file called SPAM
33          if(tuple.getStringByField("category").equals("SPAM")){
34               try {
35                 BufferedWriter output;
36                output = new BufferedWriter(new
37                FileWriter("/home/ubuntu/workspace/Stage6/SPAM.txt", true));
38
39                output.newLine();
40                output.append("NEW_TWEET**************************************");
41                output.newLine();
42                output.append(tuple.getString(1));
43                output.close();
44
45                    } catch (IOException e) { e.printStackTrace();}
46           }//end of if statement
47
48          //else put the tweet in a file called tweets
49          else{
50              try {
51                 BufferedWriter output;
52                output = new BufferedWriter(new
53                FileWriter("/home/ubuntu/workspace/Stage6/tweets.txt", true));
54
55                output.newLine();
56                output.append("NEW_TWEET**************************************");
57                output.newLine();
58                output.append(tuple.getString(1));
59                output.close();
60
61                    } catch (IOException e) { e.printStackTrace();}
62           }//end of else statement
63      }
64
65      @Override
66      public void declareOutputFields(OutputFieldsDeclarer ofd) {
67      }
68
69  }
```

R Count Printer Bolt

```
5       public CountPrinterBolt(String filename){
6           this.filename = filename;
7       }
8
9           @Override
10          public void execute(Tuple tuple, BasicOutputCollector collector) {
11
12              try {
13              BufferedWriter output;
14              output = new BufferedWriter(
15                  new FileWriter("/home/ubuntu/workspace/Stage6/"+ filename +".txt", true));
16
17              output.newLine();
18
19              int windowLenght = tuple.getInteger(2); //gets the window lenght
20              double mins = (double)(windowLenght+1)/60; //converts to minutes
21              DecimalFormat df = new DecimalFormat("0"); //formatting
22              //print the window
23              output.append("Count for the past " + df.format(mins) + " mintues ");
24
25              //get the current time
26              DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyy HH:mm:ss");
27              Calendar cal = Calendar.getInstance();//formatting
28
29              //print date and time
30              output.append("at" + dateFormat.format(cal.getTime()) + ": ");
31              //print the category
32              output.append(tuple.getString(0) + " = ");
33              //print the count
34              output.append(Integer.toString(tuple.getInteger(1)));
35
36              output.close();
37              } catch (IOException e) { e.printStackTrace();}
38          }
39
40          @Override
41          public void declareOutputFields(OutputFieldsDeclarer ofd) {
42          }
43  }//end of class
```