Project presentation

Sentimental analysis and ranking of hashtags from live tweets from a specific geographic area

Julia Ye Jacopo Castello

The aim of our project is to provide a simple analysis of hashtags from live tweets from a specific geographic area by conducting both a sentimental and a frequency analysis on the them.
Therefore the final result will be a ranking of the found hashtags based on both their frequency and fixed Sentimental value.

To achieve so we had to use Spark Streaming and make use of the Streaming API made available by Twitter to retrieve a live stream of tweets.
Older versions of Spark, until version 1.6.3, used to support access to Twitter stream through "TwitterUtils" but this is not available anymore in more recent versions, for instance we used Spark 2.4.3.
In fact for recent versions it is needed to use something that provides extensions to Spark in order to reach different streaming connectors like, in our case, the Twitter one. The easier option was Apache Bahir so we used that to reach Twitter from our version of Spark.

Since we had problems implementing Apache Bahir on our machine we decided to write and run our code in Databricks Community Edition.

The **notebook is accessible here or at** https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/6547909126626669/4170943410387995/23449215446808/latest.html were is possible to import the Notebook to our own account and then run it.

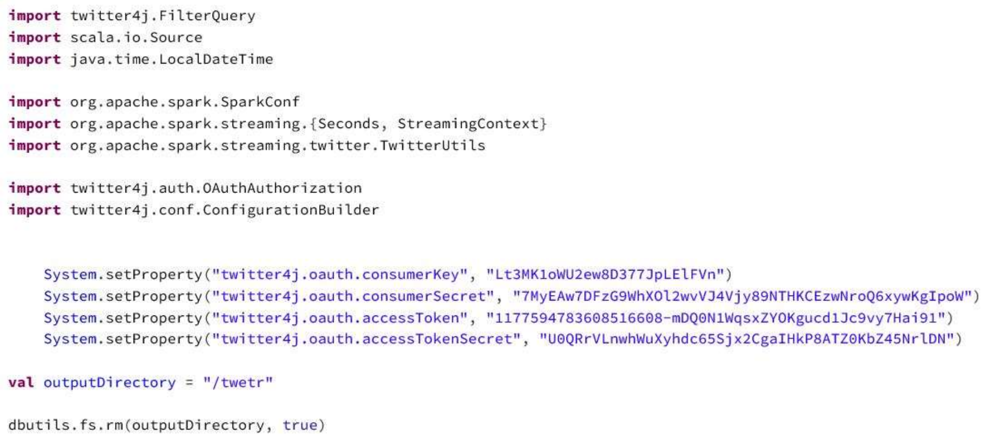
In order to run it, as mentioned, we needed to implement Apache Bahir.
In Databricks this is possible by going to "Clusters" → "Libraries" → "Install new" → "Maven" → "Search Packages", look for Bahir, select it and then install it on the cluster.

The other documents in this zip file are:

- "**Sentimental analysis.scala**": the actual Scala file containing the source code
- **"sent.txt":** the text file storing the word and their sentimental values, used for the sentimental evaluation.
- **"Results in document":** example of output document created by the code containing the results of the run.

- **"Results live":** example of results progressively displayed in Databricks during the execution. The results are from a different run from "Results live".

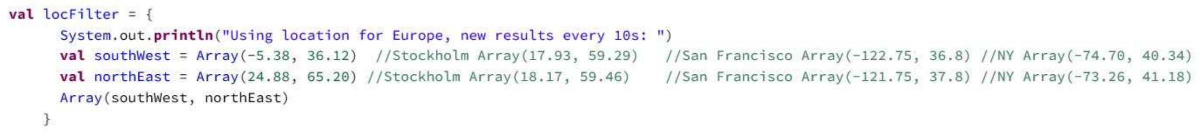
The code can be logically divided in six parts. In the first part (**see picture 1**) we just took care of the set up by importing the needed libraries, mainly the ones for the interaction with Twitter.
Also in this part we define the **consumer key**, **consumer secret**, **access token** and **access token secret** which are required in order to correctly "connect" to the Twitter stream. These four can be retrieved by just accessing the "Twitter for developers" section and create an App.
Finally we also define the directory in which we will create the output file containing the results.



```
import twitter4j.FilterQuery
import scala.io.Source
import java.time.LocalDateTime

import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.twitter.TwitterUtils

import twitter4j.auth.OAuthAuthorization
import twitter4j.conf.ConfigurationBuilder


    System.setProperty("twitter4j.oauth.consumerKey", "Lt3MK1oWU2ew8D377JpLElFVn")
    System.setProperty("twitter4j.oauth.consumerSecret", "7MyEAw7DFzG9WhXOl2wvVJ4Vjy89NTHKCEzwNroQ6xywKgIpoW")
    System.setProperty("twitter4j.oauth.accessToken", "1177594783608516608-mDQ0N1WqsxZYOKgucd1Jc9vy7Hai91")
    System.setProperty("twitter4j.oauth.accessTokenSecret", "U0QRrVLnwhWuXyhdc65Sjx2CgaIHkP8ATZ0KbZ45NrlDN")

val outputDirectory = "/twetr"

dbutils.fs.rm(outputDirectory, true)
```

Picture 1: importing needed libraries and setting of authentications for accessing Twitter stream

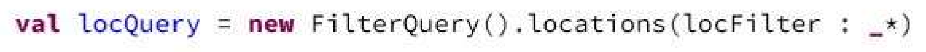
Since our analysis wants to focus on a specific area we must define the area of interest. In the second part we do so by defining "**locFilter**" which is an array containing two geographical points. Each point is defined by an array of length two storing its latitude and longitude. (**Picture 2**)
The two points are used to define the area: using a **rectangular area** they are respectively the bottom left corner and the top right corner; the area will be everything in between their latitude and longitude.
Our original focus was supposed to be on the Stockholm area but after a couple run we only got a few results so we decided to expand our focus on almost all Europe. Nevertheless, the area can be easily changed by just modifying the two point used, as can be seen in the picture where for example we have commented points for S. Francisco, New York, etc.



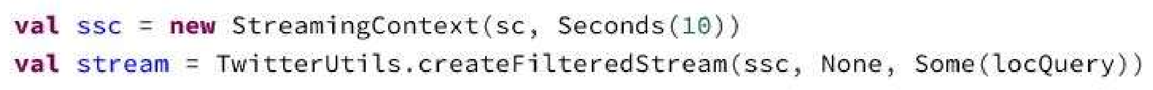
```
val locFilter = {
      System.out.println("Using location for Europe, new results every 10s: ")
      val southWest = Array(-5.38, 36.12)  //Stockholm Array(17.93, 59.29)   //San Francisco Array(-122.75, 36.8) //NY Array(-74.70, 40.34)
      val northEast = Array(24.88, 65.20) //Stockholm Array(18.17, 59.46)    //San Francisco Array(-121.75, 37.8) //NY Array(-73.26, 41.18)
      Array(southWest, northEast)
    }
```

Picture 2: defining of the area used to filter the Twitter stream by defining two opposite points which will be used as opposite corner of the rectangular area

In the following lines we just create a new object of type "**FilterQuery"** and we feed it the area we just defined in the previous section.
The FilterQuery is then used, together with the StreamingContext, to create a **FilteredStream** which will be used to "connect" to the Twitter stream and filter it. (**Picture 3**)



```
val locQuery = new FilterQuery().locations(locFilter : _*)
```

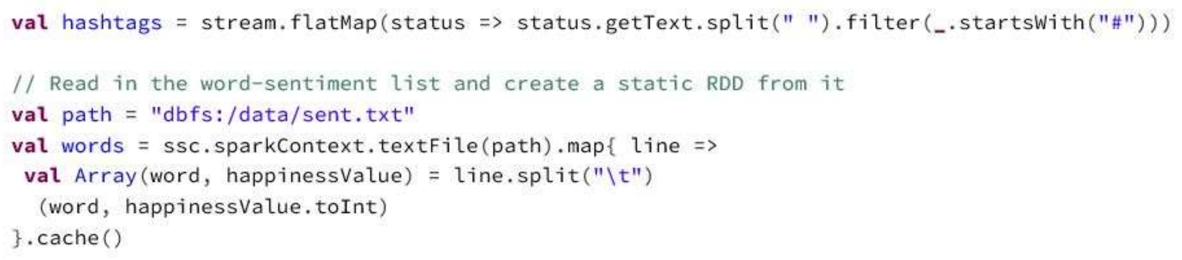


```
    val ssc = new StreamingContext(sc, Seconds(10))
    val stream = TwitterUtils.createFilteredStream(ssc, None, Some(locQuery))
```

Picture 3: creation of the FilterQuery(), streaming context and filtered stream used to access the Twitter stream

We then proceed to extract the **hashtags** from the tweets retrieved through the Twitter Stream and the **pairs** of type (word, value) stored in our file "**sent.txt**", which will be then used to do the sentimental evaluation of the hashtags.(**Picture 4)**

The hashtags are obtained by just **splitting** the content of each tweet, using the space as separator, and by **filtering** on the words which starts with a **"#"** sign.
The **pairs** (word, HValue) are obtained by accessing the file "sent.txt", **splitting** each line of the document using the "**tab**" as separator and then storing the result into a pair of the format desired.**(Picture 4)**

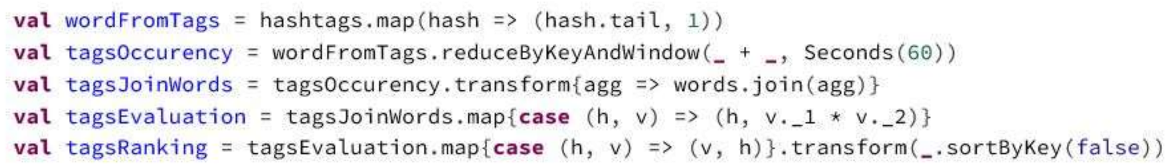


```
val hashtags = stream.flatMap(status => status.getText.split(" ").filter(_.startsWith("#")))

// Read in the word-sentiment list and create a static RDD from it
val path = "dbfs:/data/sent.txt"
val words = ssc.sparkContext.textFile(path).map{ line =>
 val Array(word, happinessValue) = line.split("\t")
  (word, happinessValue.toInt)
}.cache()
```

Picture 4: extraction of hashtags from the filtered stream and of words associated with their sentimental value from the local file

Finally we can start the **operations on the hashtags and words in order to obtain our analysis.(Picture 5)**
First we access the **hashtags** collection obtained from the **stream** and for each hashtag we remove the "#" sign at the beginning, by using .**tail** which gives back everything but the first character, and by then **creating a pair of type (tag, 1)** where the value 1 will be used in the reducing part to count the **frequency** of the tag. We store the results in "**wordFromTags**".
In the reducing part we reduce by **key**, the hashtag, and by the desired **window**, 60 seconds in this case. The results will be of type (tag, sum) where sum will be the sum of all the **occurrences** of each tag in the last 60 seconds. We store the results in "**tagsOccurency**".
We then perform a **join** between the pair of type (word, HValue), contained in "**words**" and the pair of type (tag, sum) contained in "**tagsOccurency**": the results will be of

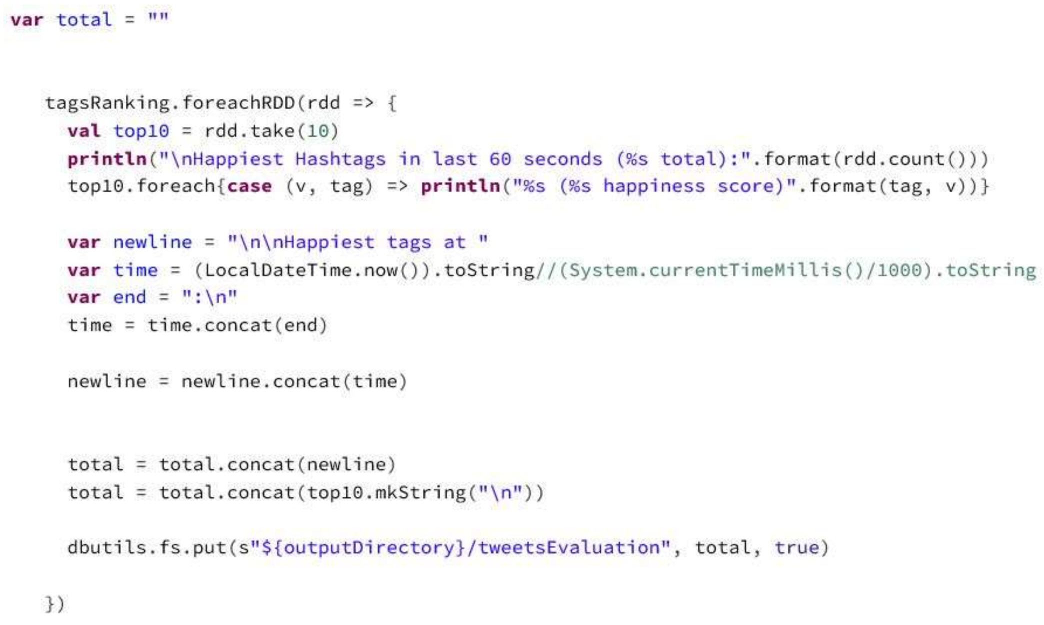
type (word, (Hvalue, sum)) so the word from the **hashtag**, its **Hvalue** and the **number of occurrences.** We store the results in "**tagsJoinWords**"
We can now proceed to calculate the final **Sentimental Value** for each tag by just taking each pair of type (word, (Hvalue, sum)) and **multiplying Hvalue for sum** (Happiness value for the number of occurrences); the two values are accessed by using **v._1 and v._2**. The results are stored into "**tagsEvaluation**" and are of type (word, fValue) with **fValue being the final value which take in consideration both the frequency of a tag and its fixed Happiness value.**
Last, since **we want to have a ranking** of the hashtags we swap the format from (word, fValue) to (fValue, word) and then **use ".sortByKey(false)" to sort the result in descending order based on their fValue**. SortByKey is used with "false" because otherwise the order will be ascending, hence the ranking would be upside-down. We store the final result in "**tagsRanking**".



```scala
val wordFromTags = hashtags.map(hash => (hash.tail, 1))
val tagsOccurency = wordFromTags.reduceByKeyAndWindow(_ + _, Seconds(60))
val tagsJoinWords = tagsOccurency.transform{agg => words.join(agg)}
val tagsEvaluation = tagsJoinWords.map{case (h, v) => (h, v._1 * v._2)}
val tagsRanking = tagsEvaluation.map{case (h, v) => (v, h)}.transform(_.sortByKey(false))
```

Picture 5: actual computation on the hashtags and words in order to define the sentimental value for each hashtag

Finally we access "tagsRanking" and for RDD underlying the Stream we take the top 10 tags based on their fValue and we print on screen and save the results into a file in our desired output directory. (**Picture 6)**

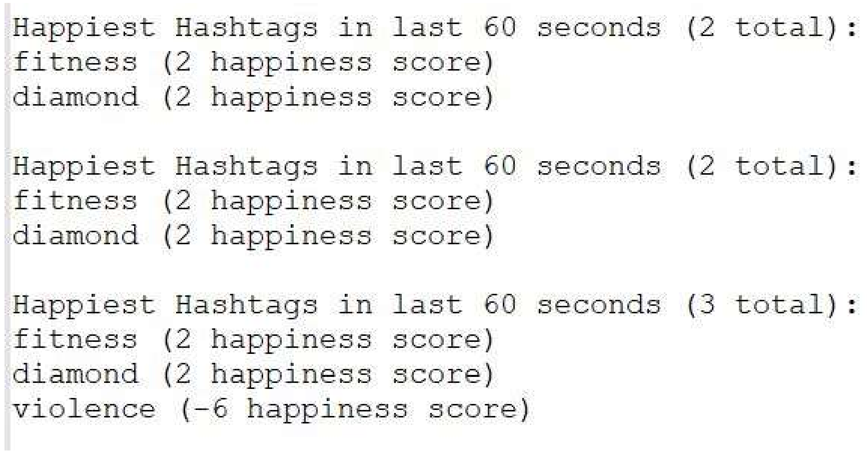


```scala
var total = ""


  tagsRanking.foreachRDD(rdd => {
    val top10 = rdd.take(10)
    println("\nHappiest Hashtags in last 60 seconds (%s total):".format(rdd.count()))
    top10.foreach{case (v, tag) => println("%s (%s happiness score)".format(tag, v))}

    var newline = "\n\nHappiest tags at "
    var time = (LocalDateTime.now()).toString//(System.currentTimeMillis()/1000).toString
    var end = ":\n"
    time = time.concat(end)

    newline = newline.concat(time)


    total = total.concat(newline)
    total = total.concat(top10.mkString("\n"))

    dbutils.fs.put(s"${outputDirectory}/tweetsEvaluation", total, true)

  })
```

Picture 6: printing of the results on screen and saving into the output file

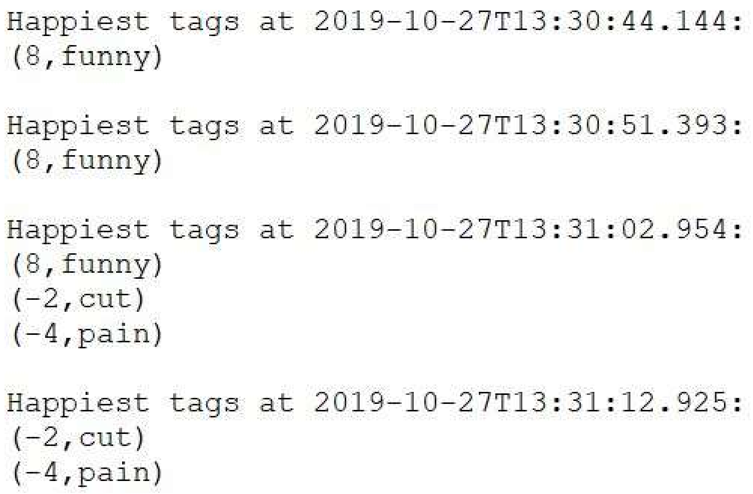
As mentioned, the data used for the sentimental evaluation are stored into “sent.txt”, present in this zip file. It stores on each line a word and its sentimental value separated by “tab”.

The results are also in this zip file, both the one printed on screen and the one stored in the output file.
They are like (the two pictures are from results from different runs):



```
Happiest Hashtags in last 60 seconds (2 total):
fitness (2 happiness score)
diamond (2 happiness score)

Happiest Hashtags in last 60 seconds (2 total):
fitness (2 happiness score)
diamond (2 happiness score)

Happiest Hashtags in last 60 seconds (3 total):
fitness (2 happiness score)
diamond (2 happiness score)
violence (-6 happiness score)
```

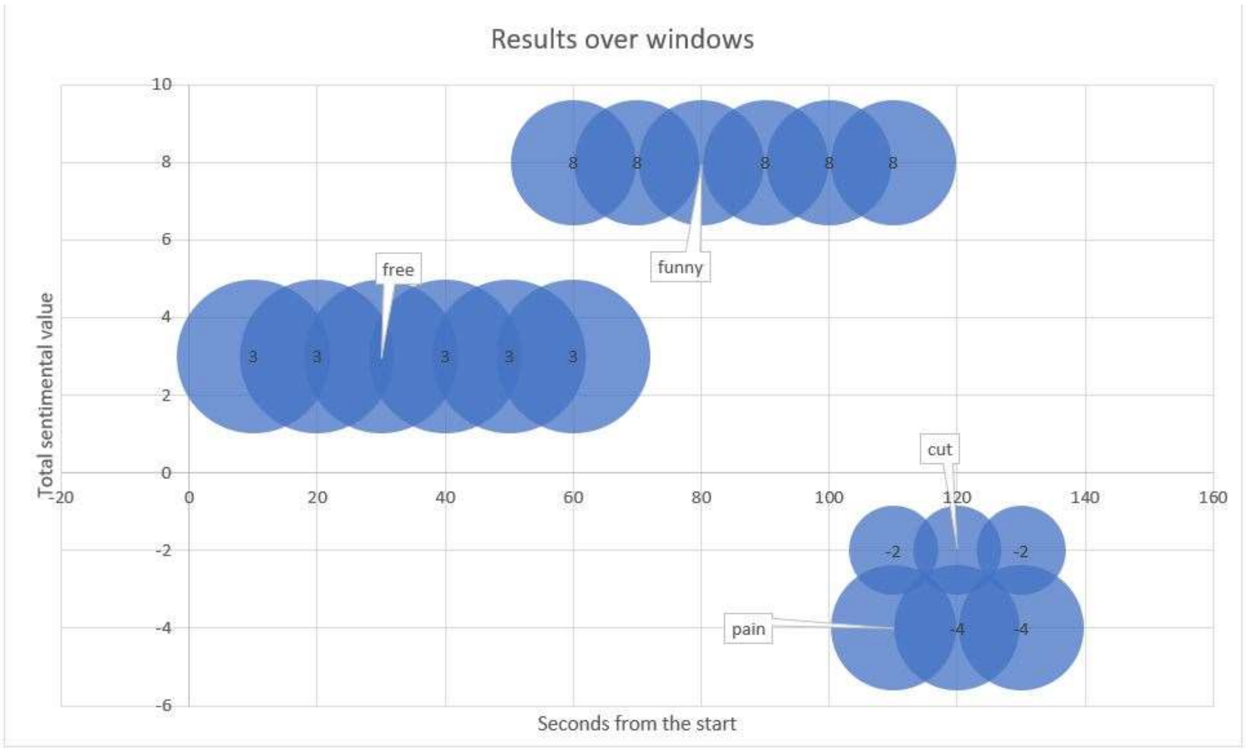
Results printed on screen in Databricks

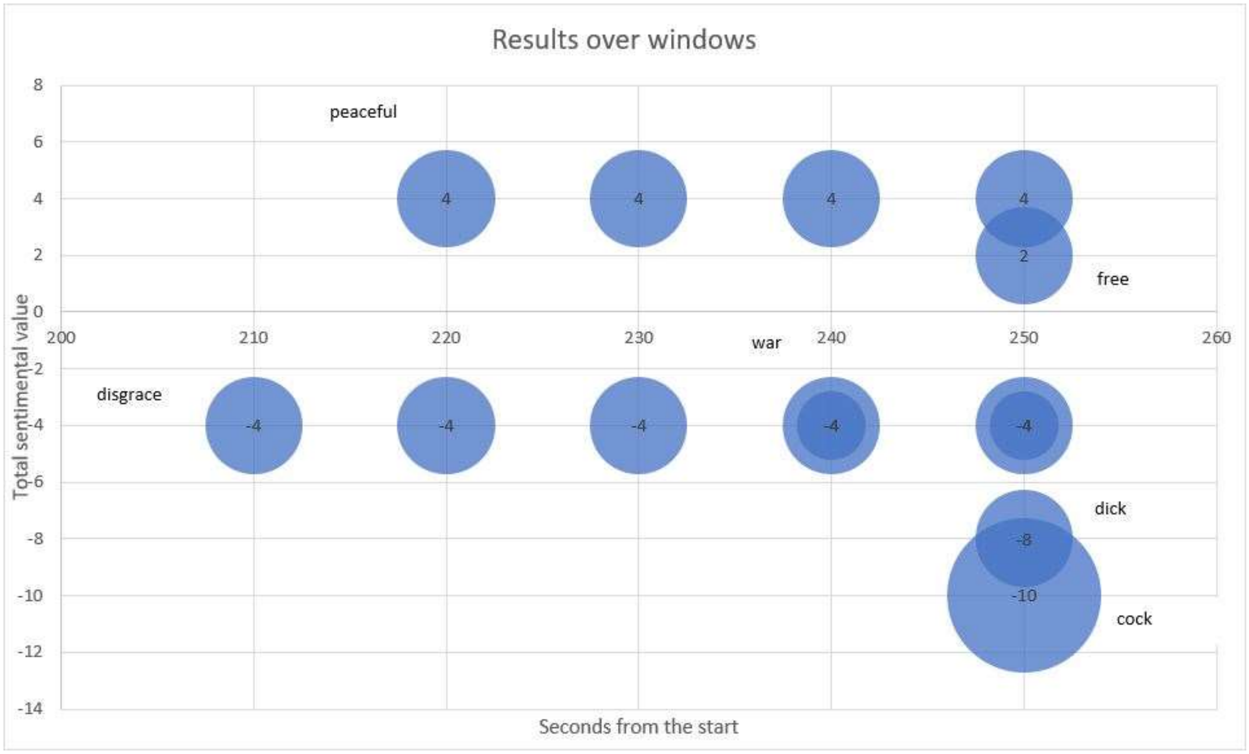


```
Happiest tags at 2019-10-27T13:30:44.144:
(8,funny)

Happiest tags at 2019-10-27T13:30:51.393:
(8,funny)

Happiest tags at 2019-10-27T13:31:02.954:
(8,funny)
(-2,cut)
(-4,pain)

Happiest tags at 2019-10-27T13:31:12.925:
(-2,cut)
(-4,pain)
```

Results into the output document

Following you can see two pictures of the same graph, but in different intervals, reporting our results over a single run of the program: the X axis represent the second passed from the start, the Y axis represent the total sentimental value for the hashtag and the size of the bubble is the rate of the hashtag.
We can see that positive hashtags are more popular/frequent than the negative ones and the hashtag with the overall highest sentimental value is “funny” with a value of 8 while the biggest bubbles are from “free” and “cock” hence they have the highest rates.


Results over windows
Total sentimental value
Seconds from the start
free
funny
cut
pain
3
8
-2
-4


Results over windows
Total sentimental value
Seconds from the start
peaceful
free
war
disgrace
dick
cock
4
2
-4
-8
-10