Data Science Algorithms

Assignment 1

Full Name – Student ID: Terizi Chrysoula - 430
Date: 14 – 01 – 2020

Part B – Algorithms for Data Streams

(2a) Twitter STREAM

The collection of data was based on the suggested video. No specific keywords were used but a random sample. The data have been crawled since 27 to 29 December 2019. The file *tweets.json* contains all the data (2.5 GB). The total number of tweets is ~416k.

Implementation instructions:

`>> python3 crawlerScript.py 18000`, where the last input(e.g. 18000) is the seconds one wishes to collect data.

(2b) Recording Stories

Initially, the data file contains information for deleted tweets, consequently ~132k tweets were deleted from the archive. Information is recorded for each post, the seven fields which are essential to find the Original Tweet Stories[1] are the followings,

- **created_at:** UTC time when this Tweet was created.
- **id_str:** The string representation of the unique identifier for this Tweet.
- **in_reply_to_status_id_str:** If the represented Tweet is a reply, this field will contain the string representation of the original Tweet's ID.
- **quoted_status_id_str, quoted_status.created_at:** This is the string representation Tweet ID of the quoted Tweet. A quote tweet is a kind of retweet. Quote tweets are sometimes also referred to as a "Retweet with comment.
- **retweeted_status.id_str, retweeted_status.created_at:** Returns the original Tweet with Retweet details embedded.

It was observed that the creation time of a quoted post is earlier than this one from a retweeted post. The output of this implementation is the *gt.csv* file which logs for each new entry, the OTS, the current tweet Id and the serial number of the OTS in the stream. In Figure 1, notes that the 98 percent of the OTS appear one time during the recorded stream and there are few OTS with more than four hundred appearances. About ~72k ids exist more than one time.

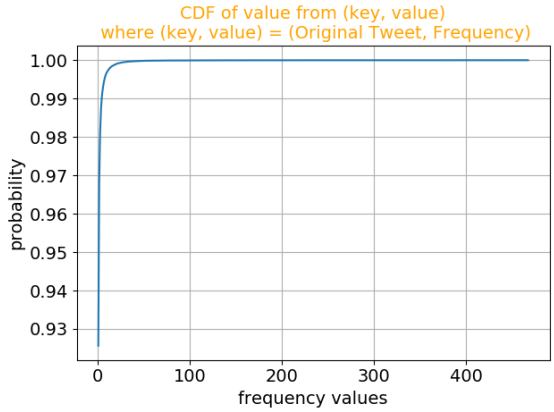


Figure 1: Cumulative distribution plot for the frequency of occurrence of the Original Tweet Id Stories.

Implementation instructions:

`>> python3 extractFields.py tweets.json`, where tweets.json is the archive with the tweets.

1 OTS (Original Tweet Story)

(2c) Statistical Processing of Stories on Twitter

- Flajolet-Martin algorithm[2]

FM ia a useful algorithm to estimate the number of unique elements in a stream. A passage to universal data is enough to make the assessment. A pseudocode is presented below,

Step 1: For various number of hash functions, i ∈ [10, 50, 100, 150, 200, 250, 300]:
Step 2: For j in [1, i]:
Step 3: Generate hash functions ((a * OTS_id + b) % p) % m, where p = 2^{64}-355, m = 2^{63}-1, a ∈ [1, p-1] and b ∈ [0, p-1]
Step 4: For each one of the OTS_ids:
Step 5: Calculate v = ((a * OTS_id + b) % p) % m
Step 6: Convert v into 64-bits binary format
Step 7: Calculate the number of zeros at the end of the binary format (let it be R)
Step 8: Keep the max value of R from all OTS_ids and (let it be max_R)
Step 9: Calculate 2^{max_R}
Step 10: Test for any possible division of functions into groups the median of 2^{max_R} per group and the average of the medians.

Implementation instructions:

>> python3 FM_AMS.py gt.csv 0 10000, where 10k is the stream length for which the count of distinct OTS will be made
>> python3 FM_AMS.py gt.csv 0 100000
>> python3 FM_AMS.py gt.csv 0 -1, where -1 is for all data in *gt.csv* file

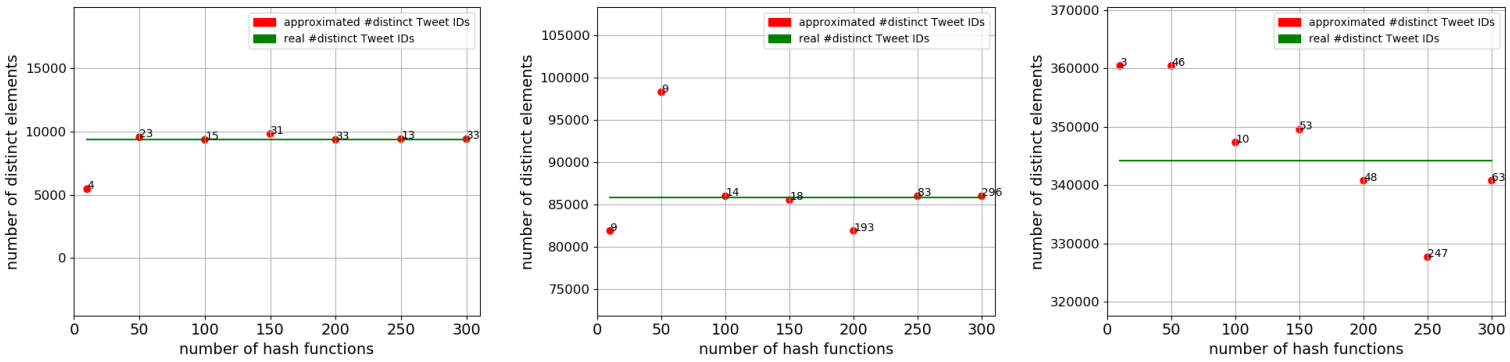


Figure 2: Plot the approximate and the real number of distinct Original Tweet Stories for the seven different values of a plurality of hash functions for Flajolet and Martin algorithm. Also, depicts the best number of elements per group that achieves the best approach. The N parameter varies in values (a)10k (b)100k and (c)all tweets.

In Figure 2(a), focusing on the 10k tweets the FM algorithm achieves very good approximation of discrete elements from fifty to three hundred hash functions. Furthermore, for all three different values of N parameter(see Figures 2(a)-(c)) and 10 hash functions, the approximation is the worst. In Figures 2(b) and (c), as the number of tweets in stream increases, the effect of the FM algorithm is still satisfactory, nevertheless, the use of three hundred hash functions a good approach to the actual number of discrete OTS.

- Alon-Matias-Szegedy algorithm[3]

The AMS algorithm applies to problems whose purpose is to distribute the frequencies of the discrete elements of the flow. The second order torque is the sum of the squares of the number of impressions of all the elements of the flow. This sum also called as surprise number because it measures how uneven the distribution of the stream elements is. The pseudocode of the implementation is presented below,

2 FM (Flajolet-Martin algorithm)
3 AMS (Alon-Matias-Szegedy algorithm)

Step 1: Input variables: k ∈ [2, ∞], N ∈ $\mathbb{Z}^+$

Step 2: Initializing parameters: number of variables ∈ $\mathbb{Z}^+$ = 500 if N > 500, otherwise number of variables = N

Step 3: For each one of the variables X_i, i ∈ [1, n] maintain three fields: X_i.element, X_i.value and X_i.n.
Keep this info in a dictionary(let it be INFO with size n):

Step 4: Select the n first tweets of the stream as the initial random positions and update the info for the first n X_i elements

Step 5: For every new entry use reservoir sampling method. Chose the new entry with probability $\frac{number\ of\ variables}{current\ n\ value+1}$ and uniformly randomly(with probability $\frac{1}{number\ of\ variables}$) selected a variable to replace. If the new entry is selected then update the three fields of the variable. Stop after the N-th tweet.

Step 6: For each possible separation of the variables into groups:

Step 7: Calculate $max_n * [INFO[i]['value']^k - (INFO[i]['value'] - 1)^k]$ for each variable per group, where max_n is the maximum value of n parameter from all variables

Step 8: Find the median per group

Step 9: Calculate the average of medians

Step 10: Return the approximation of 3 elements per group

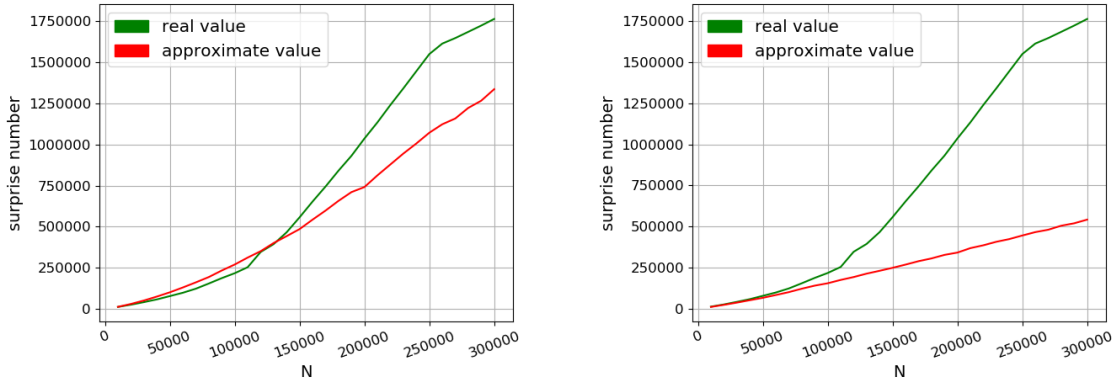
Implementation instructions:

>> python3 FM_AMS.py gt.csv 2 1000

1000 1026 942 → (N parameter, real value for surprise number, approximation for surprise number)

>> python3 ams_script.py gt.csv > a.txt (minimum N = 10000, maxmum N = 300000, step = 10000)

>> python3 plot_ams_script.py a.txt (export relevant figure)



(a) 1 element per group (b) 3 elements per group

Figure 3: Plot between actual and approximate surprise number where N parameter ranges from 10k to 300k by step 10k. Split variables in groups of (a) 1 element per group (b) 3 elements per group.

Observing the results that presented above(see Figure 3), in the case of three elements per group, this means that there are 167 teams(the implementation uses 500 variables in total), the approximate solution for AMS algorithm is worst than this one in case of 1 element per group. Increasing the elements per team, the approximate tends to get much worse. Consequently, the following calculation $\frac{\sum_{i=X1}^{X\,number\ of\ variables} max_n * ((i.value)^k - (i.value - 1)^k)}{number\ of\ variables}$ succeeds the best results.