Asmt 4: Frequent Items

Anirudh Narasimhamurthy(u0941400)

March 25, 2015

1 Streaming Algorithms

In this assignment we are asked to experiment with two different streaming algorithms for the given datasets.

A. Misra-Gries Algorithm

In this part of the assignment we are asked to explore and experiment with the Misra Gries algorithm. We have been provided k-1=9 counters and two streams S1 and S2 as the inputs. I implemented the Misra Gries algorithm as provided in the notes.

The logic implemented for the streaming algorithm is as follows:

- Initialize all the elements of the counter arrays and label/location arrays to zero and None respectively. The size of both these arrays is *k-1*
- Read the character from the stream.
- If the character is not there in our labels or locations and there is atleast one label which has counter =0 and label = None, then set the first such label array element to be that of the input character and set its counter to 1.
- If the character is not there in the set of labels and there is no label which has label=None, then decrement all the counters by 1.
- If the character read from the stream is already present in the label array, then increment that corresponding counter to 1.
- For each of the above steps, check if any counter value becomes less than or equal to zero. Whenever that happens set the label of that particular element to None.
- Return the counter array and label array at the end of processing the stream.

On running the Misra Gries algorithm on streams S1 and S2, the following are the output of the counters at the end of the stream :

Stream S1:

Label/Location	b	w	a	None	None	v	c	None	None
Counters	295	14	424	0	0	1	206	0	0

Table 1: Counters and labels for Misra Gries algorithm on Stream S1

Stream S2:

Label/Location	b	w	c	a	d	i	f	r	n
Counters	297	15	207	425	1	1	1	2	1

Table 2: Counters and labels for Misra Gries algorithm on Stream S2

When I analyzed both the streams S1 and S2, they were in fact both exactly the same size of 2000 characters and both of them had the exact set of 26 characters or alphabets. The only difference was in the order in which those characters appear in the stream and from our output we can see, the order in a way determines the counter and label values and so we get slightly different results for the frequencies of S1 and S2.

To report the objects which might occur more than 20 % of the time

The counters which we obtain represent $\hat{f}_q$ and the actual count for a query point q would be f_q. For our implementation of Misra Gries we have the following bound:

$f_q - m/k \leq \hat{f}_q$
For our problem, m=2000 and k=10, therefore $m/k = 200$
f_q would be 20% of m= 400
$f_q - m/k$=200
If we re-arrange the bounds, we would get, $\hat{f}_q \geq f_q - m/k$
$\hat{f}_q \geq 200$

Stream 1:

The objects which might occur more than 20 % of the time, just based on the counter values we obtain are : **a, b and c** as the respective counters are : **424, 295 and 206**.

Stream 2:

The objects which might occur more than 20 % of the time, just based on the counter values we obtain are : **a, b and c** as the respective counters are : **425, 297 and 207**.

To report the objects which must occur more than 20 % of the time

Basically in our implementation of Misra Gries the $\hat{f}_q$ is an undercount and Misra-Gries undercounts any element by atmost ϵm

For our problem, m=2000 and if we consider $\epsilon = 2/k$ then ϵ=20%, therefore f_q would be 20% of m= 400
$|f_q - \hat{f}_q| \leq \epsilon m$
$|f_q - \hat{f}_q| \leq 400$
So if an object must occur more than 20% of the time, then
$\hat{f}_q \geq 400$

Stream 1:

The objects which must occur more than 20 % of the time, just based on the counter values we obtain is : **a** and the respective counter is : **424**

Stream 2:

The objects which might occur more than 20 % of the time, just based on the counter values we obtain is : **a** and the respective counter is : **425**.

B. Count-Min Sketch

We are asked to build a count-min sketch for the streams S1 and S2 with k=10 counters and t=5 hash functions. This effectively implies our counter array is 5 x 10 2D-array.

These hash functions are generated randomly. I could have used the random hash functions created using SHA-1, which I had implemented for Assignment 2. But as Professor said, our given problem is slightly simpler and since the stream just contains characters, we could use the ASCII values of the characters and then build a family of hash functions.

I constructed my family of hash functions using the following expression:

h(x,a,b) = ((ax+b) mod p) mod m

- x is the character from the input stream
- p is a prime number greater than the maximum value of x. For this I found the highest ASCII value and then found a prime number which is between max and 2*max. This prime number is generated randomly for every run.
- a is a random number between 1 and p-1
- b is a random number between 0 and p-1
- m here will correspond to k and this will ensure that the value returned by hash function for any given character is between 0 and 9.

I implemented the algorithm provided in the lecture notes and constructed the counter.

Report estimated counts for objects a, b and c

After building the min-count sketch, to extract the required count, I selected the minimum of the t-counters for the given query point q

$$\hat{f}_q = \min_{j \in [t]} C_{j,h_j(q)}$$

Since the hash functions are generated at random, the counters will be different for every single run. I am reporting two such counter values and estimated counts among the many trials I ran for Stream 1 below:

Stream 1

The counter array which was obtained for Run 1 was :

C =[[372, 0, 0, 0, 1367, 0, 0, 261, 0, 0], [103, 48, 410, 544, 72, 163, 35, 449, 48, 128], [0, 0, 531, 0, 0, 740, 0, 0, 521, 208], [40, 344, 109, 584, 32, 152, 150, 441, 34, 114], [329, 114, 157, 63, 62, 625, 31, 64, 119, 436]]

Estimated count of a for S1 = 544
Estimated count of b for S1 = 436
Estimated count of c for S1 = 329

The counter array which was obtained for Run 2 was :

C =[[30, 334, 1074, 51, 129, 107, 47, 78, 99, 51], [539, 655, 81, 137, 92, 28, 61, 36, 22, 349], [931, 359, 114, 91, 150, 221, 134, 0, 0, 0], [80, 38, 51, 28, 432, 520, 575, 87, 66, 123], [50, 139, 547, 54, 50, 357, 23, 110, 50, 620]]

Estimated count of a for S1: 575
Estimated count of b for S1 : 432
Estimated count of c for S1: 334

If we run the exact same set of random hash functions which we created for stream S1 on stream S2, then the counters will be exactly the same in this case, since both the streams S1 and S2 are 2000 characters long and have the same set of 26 characters.

If we are applying different random functions to the streams S2 than what we had for S1, the following are two of the trial values which I had obtained.

Stream S2

The counter array which was obtained for Run 1 was :

C = [[157, 35, 432, 63, 329, 35, 91, 114, 103, 641], [90, 166, 96, 454, 567, 167, 51, 33, 35, 341], [1272, 324, 0, 0, 217, 0, 0, 187, 0, 0], [108, 0, 107, 88, 588, 0, 541, 0, 568, 0], [25, 576, 417, 87, 101, 58, 64, 173, 444, 55]]

Estimated count of a for S2 = 567
Estimated count of b for S2 = 432
Estimated count of c for S2 = 329

The counter array which was obtained for Run 2 was :

C = [[0, 372, 0, 0, 0, 1290, 0, 0, 338, 0], [607, 157, 50, 613, 0, 102, 338, 77, 56, 0], [152, 71, 109, 26, 482, 48, 152, 30, 346, 584], [112, 57, 511, 66, 46, 171, 565, 326, 114, 32], [0, 0, 197, 543, 0, 0, 729, 0, 0, 531]]

Estimated count of a for S2 = 543
Estimated count of b for S2 = 482
Estimated count of c for S2 = 326

Clearly the estimated counts which we obtain are overcounts and so $\hat{f}_q \geq f_q$

And again clearly since the hash functions are random the counter values would be different every single time. But we have the guarantee that the most frequent items would be the ones which have a really high counter.

Which of these objects with probability $1 - \delta = 31/32$ might occur more than 20% of the time.

We know that for our implementation of Count-Min Sketch, we have

$f_q \leq \hat{f}_q \leq f_q + \epsilon F_1$

The first inequality always holds and the second inequality holds with a probability of $1 - \delta$. This is what we are interested in answering for this question.

$\epsilon F_1 = 0.2 * 2000 = 400$

This implies to find an object which might occur more than 20% of the time, we have
$\hat{f}_q \geq 400$

Stream 1:

From our estimated counts, the objects which *might* occur more than 20% of the time are **a and b** as their counter values are greater tha 400 in all the runs.

Stream 2:

From our estimated counts, the objects which *might* occur more than 20% of the time are **a and b** as their counter values are greater tha 400 in all the runs.

C. How would these algorithms need to change if each object of the stream was a word seen on Twitter

When I first implemented Misra Gries algorithm and looked at this question, I couldn't find any difference in the approach required, since instead of maintaining a counter for characters which we did in our question, in the Twitter setting it would be for the set of words.

One major difference in the Twitter setting might be the domain. The set of characters which we could see in words in Twitter might be huge. And since we are talking about tweets, we could have a lot of stop words and these might not necessarily be the interesting ones we want to find out.

But if we were to think in terms of m and n and their sizes then we can propose the following solutions. That is there are m elements and they come from a domain [n].

Misra Gries Algorithm

if n << m

1. Create a map with every twitter word which comes in stream to a shorthand symbol
2. The list of stop words could be pre-defined and we could assign any of the stop words with the same symbol. 2. Labels are maintained for each word in the stream since n ¡¡ m

if n is approximately equal to m

1. Create a map with every twitter word which comes in stream to a shorthand symbol

2. Label are maintained only for words in stream which are present in the label

For both the case mentioned above, the solutions can be improved by hashes. Hash of sum of ascii characters value of each word can be used for the calculation of symbol for each twitter word. If the stop words have the maximum counter value, we can take the second highest

counter and that will give us an idea of the most frequently occuring word in the twitter stream.

Count-Min Sketch Algorithm

Here to perform the hashing on words, we can hash the sum of ascii character values of each word and we can use it for the calculation of symbol for each word. And then we can perform the hashing like we did earlier in our experiments in this assignment.

D. Advantage of Count-Min Sketch over the Misra-Gries algorithm

- In a strict Turnstile model, each count must remain positive. The operation which can be performed can either be increment or decrement. But the count must remain positive.
- Count-Min Sketch will give us a gurantee that the frequency would always be positive. This is because for any given character in the stream, our Count-Min Sketch will always maintain its history by hashing it into a particular counter. So if even the distribution of the stream is different Count-Min will make sure the counter always reamins poitive since the overcounting will counter balance any decrements due to other less frequently occuring elements.
- Whereas in Misra Gries, when we get a new distinct character , and if all the counters are filled and all the elemnents in the label array are full(and considering if we were to perform the decrement operation in Turnstile model)then the counts would become negative. Similar would be the case if there were a stream of distinct elements coming in the middle and if we were to perform decrement operation, then few counters would become negative. This could be avoided if the location counters were increased but then this will lead to the wastage of space and we cant afford the luxury of having so many counters.
- Thus Misra Gries provides a guarantee that the counters would be positive and also it makes sure it takes into consideration every charcter it sees in the stream and maintains a counter. In Misra Gries the first distinct element which we might see after all the counters are filled might not be stored in location array and we might lose track of few elements. However this would only be an issue if we want to find say all characters in stream which occur more than 1% or so. For other cases, especially the heavy hitter case, it should work fine.

Bonus / Extra Credit

Two pass streaming algorithm to solve the exact heavy hitter problem

First pass

Run Misra Gries algorithm on the input stream using t-hash functions and return the location/label array and counters array at the end of first pass

Second pass

The first pass might return few false positives.
In the second pass find the hashes again for each character/word in the stream
But this time to omit false positives, we can search and verify the minimum count for index of hash of each letter.
Omitting false negatives shouldn't be a problem as Misra Gries takes care of it already.

For $\phi = 10\%$ then assuming we take $k = 2/\epsilon$ we will have k=20 counters. And then doing the two pass algorithm as above will help us find the exact heavy hitter.