

**e-PGPathshala**  
**Subject : Computer Science**  
**Paper: Data Analytics**  
**Module No 20: CS/DA/20 - Data Analytics -**  
**Mining Streams-Part III**  
**Quadrant 1 – e-text**

### **1.1 Introduction**

This chapter gives an overview on the basics of computing moments, algorithms for computing moments and item sets from streams. It also deals with basics of decaying window concept.

### **1.2 Learning Outcomes**

- Learn the basics of computing moments and itemsets from streams
- Understand the algorithms for estimating moments and counting item sets from streams
- Know basics of decaying window concept

### **1.3 Moment**

A **moment** is a specific quantitative measure, used in both mechanics and statistics, of the shape of a set of points. Suppose a stream consists of elements chosen from a universal set. Assume the universal set is ordered so we can speak of the  $i^{\text{th}}$  element for any  $i$ . Let  $m_i$  be the number of occurrences of the  $i^{\text{th}}$  element for any  $i$ . Then the  $k^{\text{th}}$ -order moment (or just  $k^{\text{th}}$  moment) of the stream is the sum over all  $i$  of  $(m_i)^k$ .

Example: The 0th moment is the sum of 1 for each  $m_i$  that is greater than 0. That is, the 0th moment is a count of the number of distinct elements in the stream. The 1st moment is the sum of the  $m_i$ 's, which must be the length of the stream. Thus, first moments are especially easy to compute; just count the length of the stream seen so far.

The second moment is the sum of the squares of the  $m_i$ 's. It is sometimes called the surprise number, since it measures how uneven the distribution of elements in the stream is. To see the distinction, suppose we have a stream of length 100, in which

eleven different elements appear. The most even distribution of these eleven elements would have one appearing 10 times and the other ten appearing 9 times each. In this case, the surprise number is  $10^2 + 10 \times 9^2 = 910$ . At the other extreme, one of the eleven elements could appear 90 times and the other ten appear 1 time each. Then, the surprise number would be  $90^2 + 10 \times 1^2 = 8110$ .

### 1.3.1 Generalization: Moments

Suppose a stream has elements chosen from a set  $A$  of  $N$  values. Let  $m_i$  be the number of times value  $i$  occurs in the stream, then

The  $k^{\text{th}}$  moment is  $\sum_{i \in A} (m_i)^k$

- 0<sup>th</sup> moment = number of distinct elements
- 1<sup>st</sup> moment = count of the numbers of elements = length of the stream
  - Easy to compute

2<sup>nd</sup> moment = a measure of how uneven the distribution is = *surprise number*

$$S = \sum_{i \in A} (m_i)^2$$

Example 1: Surprise Number: Consider a stream of length 100 with 11 distinct values. The item counts: 10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9. Then the surprise  $S = 910$ .

If the item counts: 90, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, then the surprise  $S = 8,110$

Example 2: Suppose the stream is a, b, c, b, d, a, c, d, a, b, d, c, a, a, b, the length of the stream is  $n = 15$ . Then the second moment for the stream is  $5^2 + 4^2 + 3^2 + 3^2 = 59$ .

### 1.4 The Alon-Matias-Szegedy (AMS) Algorithm for Second Moments

For now, let us assume that a stream has a particular length  $n$ . We shall show how to deal with growing streams in the next section. Suppose we do not have enough space to count all the  $m_i$ 's for all the elements of the stream. We can still estimate the second moment of the stream using a limited amount of space; the more space we use, the more accurate the estimate will be. We compute some number of variables. For each variable  $X$ , we store:

1. A particular element of the universal set, which we refer to as  $X.\text{element}$ , and
2. An integer  $X.\text{value}$ , which is the value of the variable. To determine the value of a variable  $X$ , we choose a position in the stream between 1 and  $n$ , uniformly and at random. Set  $X.\text{element}$  to be the element found there, and initialize  $X.\text{value}$  to 1. As we read the stream, add 1 to  $X.\text{value}$  each time we encounter another occurrence of  $X.\text{element}$

Example 1 : Suppose the stream is a, b, c, b, d, a, c, d, a, b, d, c, a, a, b. The length of the stream is  $n = 15$ . Since a appears 5 times, b appears 4 times, and c and d appear three times each, the second moment for the stream is  $5^2 + 4^2 + 3^2 + 3^2 = 59$ . Suppose we keep three variables,  $X_1$ ,  $X_2$ , and  $X_3$ . Also, assume that at “random” we pick the 3rd, 8th, and 13th positions to define these three variables.

When we reach position 3, we find element c, so we set  $X_1.\text{element} = c$  and  $X_1.\text{value} = 1$ . Position 4 holds b, so we do not change  $X_1$ . Likewise, nothing happens at positions 5 or 6. At position 7, we see c again, so we set  $X_1.\text{value} = 2$ .

At position 8 we find d, and so set  $X_2.\text{element} = d$  and  $X_2.\text{value} = 1$ . Positions 9 and 10 hold a and b, so they do not affect  $X_1$  or  $X_2$ . Position 11 holds d so we set  $X_2.\text{value} = 2$ , and position 12 holds c so we set  $X_1.\text{value} = 3$ . At position 13, we find element a, and so set  $X_3.\text{element} = a$  and  $X_3.\text{value} = 1$ . Then, at position 14 we see another a and so set  $X_3.\text{value} = 2$ . Position 15, with element b does not affect any of the variables, so we are done, with final values  $X_1.\text{value} = 3$  and  $X_2.\text{value} = X_3.\text{value} = 2$ . We can derive an estimate of the second moment from any variable  $X$ . This estimate is  $n(2X.\text{value} - 1)$ .

## 1.5 Higher-Order Moments

We estimate  $k^{\text{th}}$  moments, for  $k > 2$ , in essentially the same way as we estimate second moments. The only thing that changes is the way we derive an estimate from a variable. We can use the formula  $n(2v - 1)$  to turn a value  $v$ , the count of the number of occurrences of some particular stream element  $a$ , into an estimate of the second moment. This formula works as : the terms  $2v - 1$ , for  $v = 1, 2, \dots, m$  sum to  $m^2$ , where  $m$  is the number of times  $a$  appears in the stream.

Notice that  $2v - 1$  is the difference between  $v^2$  and  $(v - 1)^2$ . Suppose we wanted the third moment rather than the second. Then all we have to do is replace  $2v-1$  by  $v^3-(v-1)^3 = 3v^2-3v+1$ . Then  $\sum_{v=1}^m 3v^2-3v+1 = m^3$ , so we can use as our estimate of the third moment the formula  $n(3v^2-3v+1)$ , where  $v = X.\text{value}$  is the value associated with some variable  $X$ . More generally, we can estimate  $k$ th moments for any  $k \geq 2$  by turning value  $v = X.\text{value}$  into  $n(v^k - (v - 1)^k)$ .

## 1.6 Counting item sets

We shall present the simplest case of an algorithm called DGIM. This version of the algorithm uses  $O(\log^2 N)$  bits to represent a window of  $N$  bits, and allows us to estimate the number of 1's in the window with an error of no more than 50%. Later, we shall discuss an improvement of the method that limits the error to any fraction  $\epsilon > 0$ , and still uses only  $O(\log^2 N)$  bits (although with a constant factor that grows as  $\epsilon$  shrinks).

To begin, each bit of the stream has a timestamp, the position in which it arrives. The first bit has timestamp 1, the second has timestamp 2, and so on. Since we only need to distinguish positions within the window of length  $N$ , we shall represent timestamps modulo  $N$ , so they can be represented by  $\log_2 N$  bits. If we also store the total number of bits ever seen in the stream (i.e., the most recent timestamp) modulo  $N$ , then we can determine from a timestamp modulo  $N$  where in the current window the bit with that timestamp is.

We divide the window into buckets, consisting of:

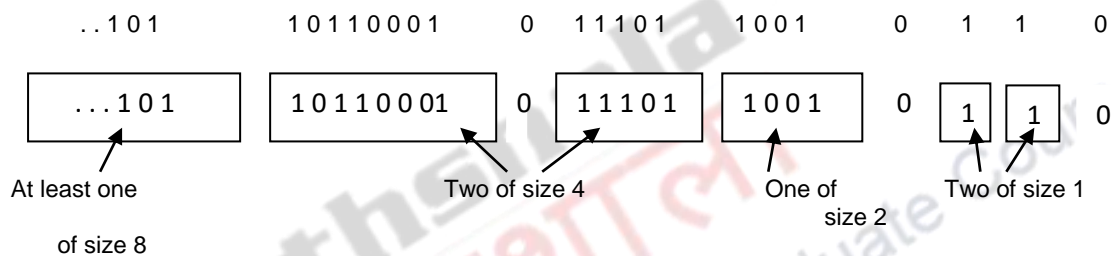
1. The timestamp of its right (most recent) end.
2. The number of 1's in the bucket. This number must be a power of 2, and we refer to the number of 1's as the size of the bucket.

To represent a bucket, we need  $\log_2 N$  bits to represent the timestamp (modulo  $N$ ) of its right end. To represent the number of 1's we only need  $\log_2 \log_2 N$  bits. The reason is that we know this number  $i$  is a power of 2, say  $2^j$ , so we can represent  $i$

by coding  $j$  in binary. Since  $j$  is at most  $\log_2 N$ , it requires  $\log_2 \log_2 N$  bits. Thus,  $O(\log N)$  bits suffice to represent a bucket.

There are six rules that must be followed when representing a stream by buckets.

- I. The right end of a bucket is always a position with a 1.
- II. Every position with a 1 is in some bucket.
- III. No position is in more than one bucket.
- IV. There are one or two buckets of any given size, up to some maximum size.
- V. All sizes must be a power of 2.
- VI. Buckets cannot decrease in size as we move to the left (back in time).



**Figure 1.** A bit-stream divided into buckets following the DGIM rules

In principle, you could count frequent pairs or even larger sets the same way as one stream per itemset. The drawbacks are like, it is approximate and number of itemsets is way too big.

## 1.7 Exponentially Decaying Windows

Rather than fixing a window size, we can imagine that the window consists of all the elements that ever arrived in the stream, but with the element that arrived  $t$  time units ago weighted by  $e^{-ct}$  for some time-constant  $c$ . Doing so allows us to maintain certain summaries of an exponentially decaying window easily. For instance, the weighted sum of elements can be recomputed, when a new element arrives, by multiplying the old sum by  $1 - c$  and then adding the new element.

If stream is  $a_1, a_2, \dots$  and we are taking the sum of the stream, take the answer at time  $t$  to be:  $\sum_{i=1,2,\dots,t} a_i e^{-c(t-i)}$ .

$c$  is a constant, presumably tiny, like  $10^{-6}$  or  $10^{-9}$ .

*Example:*

If each  $a_i$  is an “item” we can compute the *characteristic function* of each possible item  $x$  as an E.D.W.

That is:  $\sum_{i=1,2,\dots,t} \delta_i e^{-c(t-i)}$ , where  $\delta_i = 1$  if  $a_i = x$ , and 0 otherwise.

Call this sum the “*count*” of item  $x$ .

Counting items:

1. Suppose we want to find those items of weight at least  $\frac{1}{2}$ .
2. Important property: sum over all weights is  $1/(1 - e^{-c})$  or very close to  $1/[1 - (1 - c)] = 1/c$ .
3. Thus: at most  $2/c$  items have weight at least  $\frac{1}{2}$ .

Initiation of New Counts


1. Start a count for an itemset  $S \subseteq B$  if every proper subset of  $S$  had a count prior to arrival of basket  $B$ .
2. Example: Start counting  $\{i, j\}$  iff both  $i$  and  $j$  were counted prior to seeing  $B$ .
3. Example: Start counting  $\{i, j, k\}$  iff  $\{i, j\}$ ,  $\{i, k\}$ , and  $\{j, k\}$  were all counted prior to seeing  $B$ .
4. Counts for single items  $\leq (2/c)$  times the average number of items in a basket.
5. Counts for larger itemsets = ???. But we are conservative about starting counts of large sets.

If we counted every set we saw, one basket of 20 items would initiate 1M counts

## 1.8 Frequent Itemsets

The problem of finding frequent itemsets differs from the similarity search. Here we are interested in the absolute number of baskets that contain a particular set of items. The difference leads to a new class of algorithms for finding frequent itemsets. a set of items that appears in many baskets is said to be “frequent.” To be formal, we assume there is a number  $s$ , called the support threshold. If  $I$  is a set of items, the support for  $I$  is the number of baskets for which  $I$  is a subset. We say  $I$  is frequent if its support is  $s$  or more.

The original application of the market-basket model was in the analysis of true market baskets. That is, supermarkets and chain stores record the contents of every market basket (physical shopping cart) brought to the register for checkout. Here the “items” are the different products that the store sells, and the “baskets” are the sets of items in a single market basket. A major chain might sell 100,000 different items and collect data about millions of market baskets.

	<b>Case Studies</b>
A) <a href="http://dimacs.rutgers.edu/~graham/pubs/papers/fwddecay.pdf">http://dimacs.rutgers.edu/~graham/pubs/papers/fwddecay.pdf</a>	
<p><b>Forward Decay: A Practical Time Decay Model for Streaming Systems.</b></p> <p>A new class of “forward” decay functions based on measuring forward from a fixed point in time is discussed and shows that this model captures the more practical models already known, such as exponential decay and landmark windows, but also includes a wide class of other types of time decay.</p>	

## Summary

- Estimating moments though complex, but finds wide application
- Mining frequent itemsets from data stream leads to effective product recommendations