Mining Periodic Property of (Sparse) Data Stream: Using Double-CM Sketch

Zhiwei Ying, Wenjian Sun, Jingyu Xie

Overview

- Propose a new structure-double cover min sketch to process data stream.
- Shed light on the practical significance and application prospects of our work results.

Problem Statement

Data stream model: Given a data stream *S=(e1,e2, e3,...,ei,...)*, given an item *e*, let *ti* be the *ith* appearance of *e*. Then *e* has many intervals, where interval *Vi* is *t(i+1) - ti*.

Periodic Items: For *e*, if one of its interval occurs many times, we consider it as a periodic item. Periodic items have two aspects: interval and frequency. The frequency is defined as the number of arriving intervals that fall in the range *[V- Δ T,V+ Δ T]*.

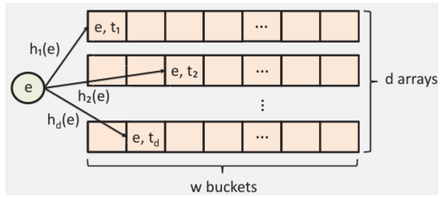
Finding Top-K Periodic Items: Counting Periodic Items with most precision: Given a data stream, the problem is to count periodic items accurately, especially for the K largest ones.

Example: Given a data stream *S=(a,b,a,b,d,d,a,c,a,c,e,e,a,b,a,b)*, suppose they arrives at the constant speed and *V=2, Δ T=0* for convenience. Then we obtain the periodic items as follows. Top 1: *(a,2)* occurs 3 times. Top 2: *(a,4)* occurs twice. Top 2: *(b,2)* occurs twice.

Method

Cover-Min Sketch:

Insertion: When inputting an item *e* with timestamp *t*, it is mapped to one bucket in each of the *d* arrays using d hash functions. The Cover-Min sketch then updates the timestamp in each corresponding bucket to the current time.



Algorithm 1: Insertion and report of a coming item e.

Input: input a coming item e with the timestamp t
Output: output the time interval V of e

```
min_t ← infinity;
foreach i (0 ≤ i ≤ d − 1) do
    min_t ← min(min_t, S_V[i][h_i(e)]);
    S_V[i][h_i(e)] = t;
return t − min_t;
```

Report: For item e and timestamp *t*, we calculate the d hash functions to find the *d* buckets and extract the timestamps. By subtracting the smallest timestamp from the current time, we obtain the time interval *V* (*V = t − min_t*) and report it with *e*.

Count-Min Sketch:

Create an array of length *x* for counting, initializing each element to 0. For a new element,hash it to an index between 0 and x-1, and increment the count at that index. To query an element's frequency, return the count at its hash index.

To handle hash collisions, use multiple arrays and hash functions. Query an element's frequency by taking the minimum count across the different arrays.

Algorithm 2: CountMinInit(w, d, p)

```
C[1,1]...C[d,w] ← 0;
for j ← 1 to d do
    Pick a_j, b_j uniformly from [1...p];
N = 0;
```

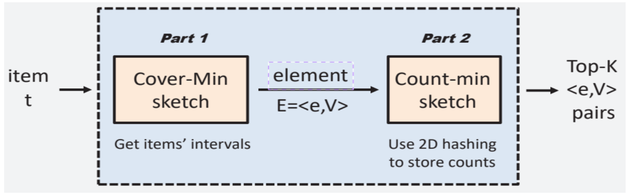
Algorithm 3: CountMinUpdate(i, c)

```
N ← N + c;
for j ← 1 to d do
    h_j(i) = (a_j × i + b_j  mod p)  mod w ;
    C[j, h_j(i)] ← C[j, h_j(i)] + c;
```

Algorithm 4: CountMinEstimate(i)

```
e ← ∞;
for j ← 1 to d do
    h_j(i) = (a_j × i + b_j  mod p)  mod w ;
    e ← min(e, C[j, h_j(i)]) ;
return e
```

Double-CM Sketch:



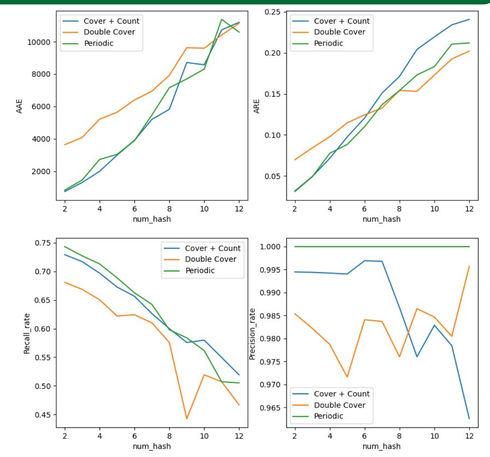
The Double-CM sketch includes a Cover-Min sketch and a Count-Min sketch. The Cover-Min sketch records and reports the interval for each incoming item, while the Count-Min sketch keeps the number of periodic items with little space complexity. Given a timestamp t for an item *e*, we insert it into the Cover-Min sketch to get the interval *V*. We then form an element *E=(e,V)* and insert it into the Count-Min sketch. Note that the same item ID with different intervals is treated as different elements and the hash mapping in Count-Min is 2D-mapping. Then we can calculate a value for each element with space complexity as small as possible, and the results are accurate with high probability.

Results

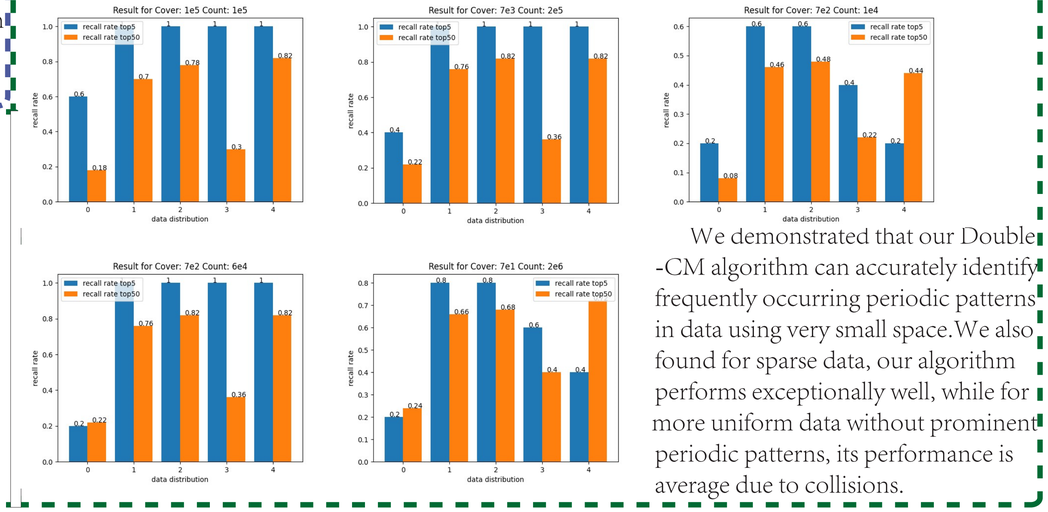
Based on the original PeriodicSketch project, we conducted two attempts: replacing the GSU sketch in PeriodicSketch with Cover-Min sketch and Count-Min sketch respectively. We found that the latter achieved better results, similar to the original project, and ultimately became our research outcome.

In our model, increasing num_hash result in higher errors, lower recall rates, and fluctuating precision. The algorithm misses more frequent items with more hash functions, reducing its pattern recall ability. Although it may still identify some items accurately, too many hash functions increase false positives. Thus, optimizing num_hash based on application needs and dataset characteristics is crucial.

1 Results for different models with differnrt num_hash



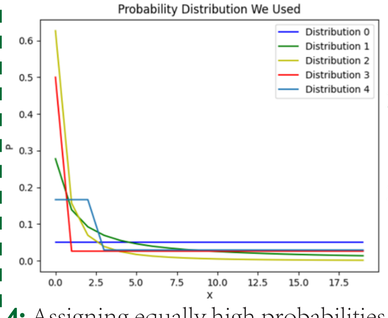
2 Results for different space complexity (with 1e6 input data)



We demonstrated that our Double-CM algorithm can accurately identify frequently occurring periodic patterns in data using very small space.We also found for sparse data, our algorithm performs exceptionally well, while for more uniform data without prominent periodic patterns, its performance is average due to collisions.

Application Prospects

We introduce a novel algorithm named Double-CM sketch for real-time maintaining the frequency characteristics of elements in real time with high accuracy in data streams. This algorithm has broad applications, including real time data processing scenarios like network traffic monitoring, financial transaction monitoring, and sensor data analysis. It helps in promptly detecting and tracking periodic events or patterns, thus aiding businesses in making informed decisions based on insights derived from big data analysis, such as market behavior and user activities.



0: Uniform distribution
1: Probability is inversely proportional
2: Probability is inversely proportional to the square of the quantity.
3: Assigning very high probabilities to 1 and equal probabilities to other numbers.
4: Assigning equally high probabilities to 1, 2, and 3, while assigning equal probabilities to other numbers.