universite
PARIS-SACLAY

Algorithms for Data Science
Data Streams II

April 2, 2023

Table of contents

Data Streams

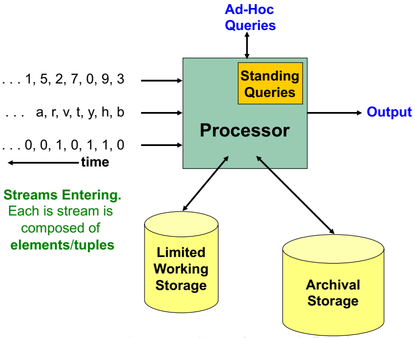
Input rate is **controlled externally** – so the data processor has no control over the speed of the data

Data streams are:

- **infinite** – one does not know the size of the data
- **non-stationary** – the distributions of the data can change (seasonally, daily, hourly)

Model: infinite sequence of items $S = (i_1, i_2, \dots, i_k \dots)$

Stream Processing Model



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Objective: asking queries on the stream – *standing* and *ad-hoc*

Restrictions: storage space and processing time – have to process it or we lose it forever!

Implementation Issues

If we had enough memory / time – data streams **would be easy**

With **restrictions**:

- more efficient to get **approximate** answers
- use space-saving techniques such as **hashing**

Problems Studied

- Sample data from a stream
- Filtering items
- Counting distinct elements
- Estimating moments
- Queries over sliding windows

Table of contents

Count-Distinct Problem

Problem: count the number of distinct items in a data stream

Applications:

- how many different words are in webpages (spam detection)?
- how many distinct products are sold in the last week?
- how many new stars do we find in space?

Count-Distinct Problem

Naïve Approach keep a set of new items found and keep a count of its size

What if we do not have enough space for all the distinct elements?

- we still want an unbiased estimator of the counts
- we accept some error in the estimation as trade-off for space

Flajolet-Martin Approach [?]

Algorithm – assume we have N items in the universe:

1. pick a hash function h mapping the N items to at least $\log_2 N$ bits
2. for each stream item s, $r(s)$ is the number of trailing 0s in the bit representation
   - for instance assume $h(s) = 12$, bit representation **1100**
   - $r(a)$ is then equal to 2
3. keep $R = \max_s r(s)$ over the entire stream

Estimator: the number of distinct items seems thus far is 2^R.

Intuition on Why It Works

Assumption: h hashes with equal probability to all N values, the values from the stream come uniformly

$h(s)$ is a **sequence of $\log_2 N$ bits**:

- a proportion of 2^{-1} (50%) will have $r(s) = 1$
- a proportion of 2^{-2} (25%) will have $r(s) = 2$
- **generally**, a proportion of 2^{-r} will have r trailing 0s

For an **uniform hash function**, it takes thus $1/2^{-r} = 2^r$ items before **we see one with r trailing 0s**

Note: it can be done with trailing 1s, or any other bit function allowing us to compute the probability

Drawbacks and Optimization

Main drawback: the expectation $E[2^R]$ can get very high

Can fix by using multiple estimators – m different hash functions

- taking the average can overestimate – if one estimator is an outlier
- taking the median is better – but it is always a power of 2
- **best approach**: hybrid, divide the hash functions in groups, compute average in each group, take the median over groups

Space Cost

Minimizes space used

- only have to keep *R* for each hash function
- we can use as many hash functions as memory permits
- time trade-off: if too many – computing the hashes (and maintaining averages, medians) can be too time costly

Table of contents

Moments of a Sequence

Assume we have a sequence/stream S having N possible distinct (ordered) values, and m_i is the number of times the ith distinct element appears in S

Moment: the nth moment of a sequence S is equal to

$$\sum_{i \in S} (m_i)^n.$$

Moments of a Sequence

Example of moments:

1. 0th moment: the number of distinct items in the stream – can be estimated using the approach presented before!
2. 1st moment: the length of the stream – easy to keep count of
3. 2nd moment: surprise number – how uneven the distribution is

Challenge: same as distinct items in stream – cannot keep all values in memory

Surprise Number

5 distinct elements not varying much: `5 4 4 4 3`

- 2nd moment (surprise number): $1^2 + 3^2 + 1^2 = 10$

5 distinct elements with outliers: `16 1 1 1 1`

- 2nd moment (surprise number): $1^2 + 4^2 = 17$

Alon-Matias-Szegedy Algorithm [?]

Assume a stream has a length n, and we have space to store a few variables and not all m_i

We keep some variables X:

- X.val – the value of the element
- X.c – the count of that element in the stream

Alon-Matias-Szegedy Algorithm (AMS):

1. choose a number i between 1 and n
2. when the stream S reaches i, set $X.\text{val} = s_i$ and $X.\text{c} = 1$
3. everytime the value in X.val is encountered in S, increment X.c

Using AMS for Estimating 2nd Moment

Estimate of the 2nd moment is:

$$n(2X.\mathrm{c} - 1)$$

The estimate can be refined by using k different X variables; the estimate is then the average of the estimates:

$$\frac{n}{k} \sum_{i \in \{1,\ldots,k\}} (2X_i.\mathrm{c} - 1)$$

AMS example

Stream ($n = 15$):

a b c b d a c d a b d c a a b

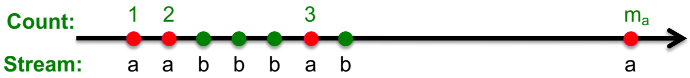
- surprise number $5^2 + 4^2 + 3^2 + 3^2 = 59$

Keep X_1, X_2, X_3, and choose 3, 8, 13 as random positions in the stream:

- X_1.val $= c$, and – at the end of the stream – X_1.c $= 3$
- X_2.val $= d$, and – at the end of the stream – X_2.c $= 2$
- X_3.val $= a$, and – at the end of the stream – X_3.c $= 2$

The final estimate is:
$15/3 \times ((2 \times 3 - 1) + (2 \times 2 - 1) + (2 \times 2 - 1)) = 55$

Why It Works



Let us write $f(X) = n(2c - 1)$, and c_t the number of times an item appears from time t on

We need to give a bound on the **expectation** of f:

$$E[f(X)] = \frac{1}{n}\sum_{t=1}^{n} n(2c_t - 1) = \sum_{i=1}^{m_i}(2i - 1)$$
$$= 2\frac{m_i(m_i + 1)}{2} - m_i = (m_i)^2$$

– **in expectation**, the formula is exactly the **second moment**!

Estimating Higher Order Moments

The algorithm works for **any moment k**, but the estimate changes

General estimator

$$n\left(c^k - (c-1)^k\right)$$

Exercise: show that it works

Infinite Streams

What happends when we do not know n?

- assume we can only hold k functions

We can use **Reservoir Sampling**

- choose the first k times for k variables
- for $n > k$ choose the item as a new variable with probability k/n, if chosen discard one of the previous k randomly
- in the estimator, use the current length of the stream as n

Table of contents

Setting: sometimes we only need to query the last N elements of a stream – queries over a sliding window

- N can be very large
- there can also be multiple stream, so keeping multiple windows is too much

Example: transactions (product was sold, ad was clicked, etc.)

Sliding Windows

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

← Past Future →

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Counting Bits

Problem: given a stream S of 0 and 1, we want to answer quries of the form

- how many 1s are in the last k bits ($k \leqslant N$)

Assumption: we cannot afford to keep the most recent N bits

- but, impossible to get an exact answer without storing the entire window
- have to settle for approximate answers

Non-Uniform Streams

In uniform streams, we can simply estimate the number of 1s by counting the number of 1s as a, 0s as b and estimate as

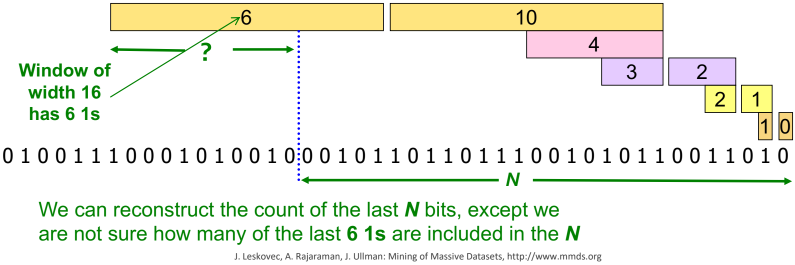
$$N\frac{a}{a+b}$$

But streams are not uniform!

Datar-Gionis-Indyk-Motwani (DGIM) Method [?]

Main Idea – exponential windows

- summarize regions of the streams in buckets, that are exponentially increasing
- keep the count for each



Pros and Cons

The advantages:

- only needs $\mathcal{O}(\log^2 N)$ bits – $\mathcal{O}(\log N)$ counts of $\log_2 N$ bits
- easy updates
- error in count not greater than the number of 1 in the "last" area
- if 1s are (relatively) evenly distributed, error is no more than 50%

The big disadvantage:

- if all the 1s are in the unknown area – error is unbounded!

The DGIM Fix

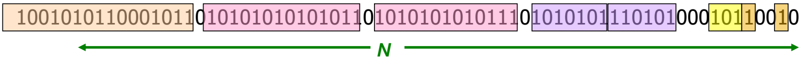
Main idea: instead of keeping fixed sizes of buckets, keep buckets containing a fixed size of 1s

- the windows increase exponentially – numbers of 1 kept as powers of 2, e.g., `1 1 2 4 16`

Buckets contain:

- the timestamp of its end – kept as timestamp modulo N, needs $\mathcal{O}(\log N)$ bits
- the number of 1s in it – since powers of 2 always, it only needs $\mathcal{O}(\log \log N)$

Restrictions on Buckets



- at most one or two buckets of the same size
- no overlap of timestamps
- new buckets are smaller than earlier ones
- buckets are removed when end time $> N$

Updating Buckets

When a new item (bit) comes, drop the last bucket if end-time after N

Update depends on the bit (0 or 1):

1. if bit is 0 – no changes needed
2. if bit is 1:
    - create a new bucket of size 1
    - if 3 buckets of size 1, combine oldest two in a new bucket of size 2
    - recurse on sizes

Updating Buckets

Current state of the stream:

10010101100010110101010101010110101010101110101010111010100010110010

Bit of value 1 arrives

001010110001011010101010101011010101010101110101010111010100010110010 1

Two orange buckets get merged into a yellow bucket

0010101100010110101010101010110101010101011101010101110101000101100101

Next bit 1 arrives, new orange bucket is created, then 0 comes, then 1:

01011000101101010101010101101010101010111010101011101010001011001011 01

Buckets get merged…

0101100010110101010101010110101010101011101010101110101000101100101101

State of the buckets after merging

0101100010110101010101010110101010101011101010101110101000101100101101

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Querying

Query:

1. sum the sizes of all buckets except the last
2. add **half the size of the last** (we do not know the proportion of the last window in N)

Error is at most 50%:

- can be reduced by maintaining r or $r-1$ buckets of each size
- **error** is then at most $\mathcal{O}(1/r)$
- **trade-off** between the number of bits and the error

Extensions

Using $k < N$ as a query parameter:

- want to query only the last k bits in the window N
- can simply "cut" at k and use the same estimator

Sum of last k integer elements:

- integers have at most m bits
- treat each bit as a separate stream and count the 1 in last k
- estimate as $\sum_{i=0}^{m-1} c_i 2^i$ where c_i is the DGIM estimator for bit i

Acknowledgments

The contents follow Chapter 4 of [**?**]. Figures in slides 4, 21, 26, 29, 32, and 34 are taken from `http://www.mmds.org/`

References i