Algorithms for Data Science
Data Streams I

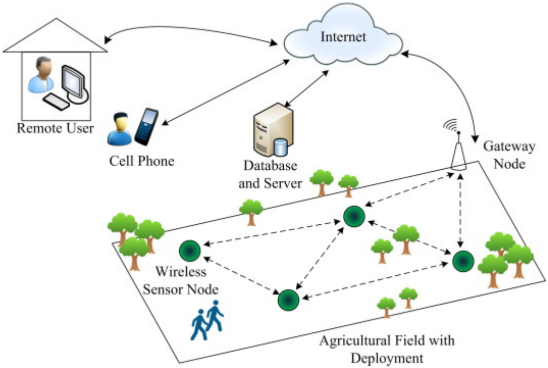
Slides provided by: **Silviu Maniu**, Presented by: **Pierre-Henri Paris**

September 26th, 2024

M2 Data Science

Table of contents

Sensor Data / Internet of Things



High-Speed Trading

Offline vs. online

Databases assume that the entirety of datasets are available offline

This is not always true – sometimes data is only online:

- Twitter status updates, queries on search engines
- data from sensor networks
- telephone calls
- IP packets on the Internet
- high-speed trading data

Data Stream Characteristics

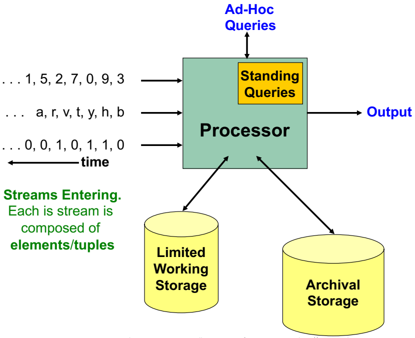
Input rate is controlled externally – so the data processor has no control over the speed of the data

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 $S = (i_1, i_2, \ldots, i_k \ldots)$

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

Sampling Problem

Objective: keeping a representative sample of the items in the stream – to deal with the limited space issues

Sub-problems:

- sample a fixed proportion of elements
- keep a sample of fixed size

Sampling a Fixed Proportion of Elements

Objective keep a proportion p of items in a stream

First solution:

- say, e.g., we want to keep 1 in 10 elements
- for each item, we can generate a random number from 0 to 9
- keep the item only we generate 0

Motivating example

Stream: tuples of (user, query, time) – queries of users on a search engine

Problem: how often does an user run the same query – what fraction of queries are duplicates

Motivating example

Issue of the above solution (assume we have space for 10% of the stream):

- suppose a queries are only once, b queries are double, total $a + 2b$ – corect answer is $b/(a + b)$
- prob. we see the singleton queries $a/10$
- prob. we see a double query twice $b/100 = b \times 1/10 \times 1/10$
- prob. we see a double query only once $18b/100 = (1/10 \times 9/10 + 9/10 \times 1/10)b$
- hence our wrong estimation is

$$\frac{b}{10a + 19b}$$

A Better Solution

It is better to sample the users, instead of the queries – so we sample **all the queries** of a proportion of the users

- this can be done by hashing strings to integers

Takeaway: one has to be careful what sample one keeps, depending on the applications

Fixed-Size Samples

Assume we have to keep a sample of exactly s items – i.e., max space in memory

Objective: each item in the stream S should be in the s with equal probability – after n items prob. should be s/n

Reservoir Sampling

Reservoir Sampling Algorithm [Vitter, 1985]

1. store first s elements in the stream in the sample
2. when element n arrives ($n > s$)
    - with probability s/n keep the element, else discard
    - if the element is kept, it replace one element in the sample (chosen randomly)

Reservoir Sampling – Proof

Claim the algorithm maintains a sample s with the desired property – each item is in s with probability s/n

Proof (induction):

- base case: first s elements are in the sample with probability $s/s = 1$
- inductive hypothesis: after n elements, the sample contains each element with prob. s/n

Reservoir Sampling – Proof

Claim the algorithm maintains a sample s with the desired property – each item is in s with probability s/n

Proof (induction):

- inductive step: element $n+1$ arrives
  - probability that it is kept in s is

$$\left(1 - \frac{s}{n+1}\right) + \frac{s}{n+1} \cdot \frac{s-1}{s} = \frac{n}{n+1}$$

  - at time n tuples are in the sample with prob. s/n, and are kept with probability $n/n+1$
  - so the probability that they "survive" in the sample at time $n+1$ is

$$\frac{s}{n} \cdot \frac{n}{n+1} = \frac{s}{n+1}$$

Table of contents

Filtering Streams

Problem: we want to let only some items in the stream, but we do not have the space to store the keys for comparison

Motivating example – e-mail filtering

- large numbers of emails come every minute, a few of them are **spam**
- we cannot keep the list of good emails in main memory (to compare), but we still want to keep only non-spam emails
- **solution**: hashing

Using Hashing to Filter Items

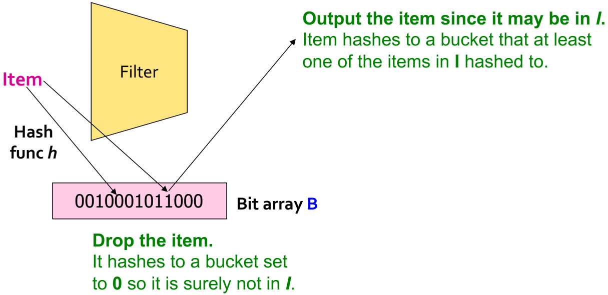
1. Set of item keys I that we want to keep / filter
2. Keep a bit array B of n bits, initialized to 0
3. Choose a hash function h with range $[0, n)$, and hash each $i \in I$ to one of the n buckets; i.e., set $B[h(i)] = 1$

Process: for each item s in the stream S, output it only if $B[h(s)] = 1$

Using Hashing to Filter Items

No **false negatives**, but some **false positives**

- some spam emails might still get through



Probability of False Positives

A good hash function – each item in the stream S is **equally likely** to hash to one of the n buckets

Assume m **unique items** (e.g., e-mails addresses)

What is the probability that a spam email hashes to a good email bit?

- equivalent: throwing m darts at n target – **what is the probability that a target gets at least one dart?**

Probability of False Positives

Equals **1/e** as **n** $\to\infty$

Equivalent

$$\boxed{1 - \boxed{(1 - 1/n)}^{\,n(m/n)}}$$

$\mathbf{1 - e^{-m/n}}$

Probability some target **X** not hit by a dart

Probability at least one dart hits target **X**

Fraction of 1 in the array B is Probability of false positives $1 - e^{-m/n}$

Example

$|I|$ – 1 billion email addresses (darts)

$|B|$ – 1GB = 8 billion bits (targets)

False positive rate: $1 - e^{-1/8} = 0.1175$

- 11% of the spam email passes through

Can we do better?

Bloom Filters [Bloom, 1970]

Structure:

- an array B of n bits, set to 0
- a **collection** of hash function $h_1, h_2, \ldots, h_k$ each mapping to the same n buckets
- set I of keys of item

Initialization:

- take each key $i \in I$ and hash it using each h_j; set to 1 each bit in B that has $h_j(i) = 1$

Function:

- for each item s from the stream, check that $h_1(s), h_2(s), \ldots, h_k(s)$ **all** map to 1 in B; discard it otherwise

Bloom Filter Analysis

Equivalent: throwing km darts at n targets; fraction of 1 is

$$1 - e^{\frac{-km}{n}}$$

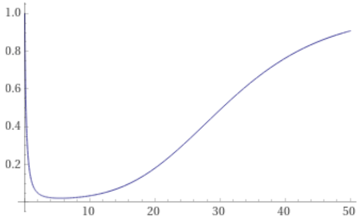
We have k independent hash functions; elements s only passes if all k hash to a bucket of 1

False Positive Probability

$$\left(1 - e^{-\frac{km}{n}}\right)^k$$

Bloom Filter Analysis

The false positive probability changes with the number k of hash functions!



Optimal Number of Hash Functions

$$k = \frac{n}{m} \ln 2$$

Bloom Filters – Takeaways

Can **optimize** the space taken, while having **no false negatives** and minimizing **false positives**

Can be implemented efficiently – parallel hash functions

Can divide B in k parts – **equivalent** but simpler to keep one bit array

Acknowledgments

The contents and some figures taken from Chapter 4 of [Leskovec et al., 2020]. https://www.mmds.org/

References i

Bloom, B. H. (1970).
Space/time trade-offs in hash coding with allowable errors.
Commun. ACM, 13(7):422–426.

Leskovec, J., Rajaraman, A., and Ullman, J. (2020).
Mining of Massive Datasets.
Cambridge University Press.

Vitter, J. S. (1985).
Random sampling with a reservoir.
ACM Trans. Math. Softw., 11(1):37–57.