Mining Data Streams I

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University
June 15, 2023

Today

Overview

- Intro: A Data Stream Management Model
- Sampling Data in a Stream
- Filtering Streams: Bloom Filters
- Counting Distinct Elements: Flajolet-Martin algorithm

Learning Goals: Understand these topics and get familiarized

Mining Data Streams: Introduction

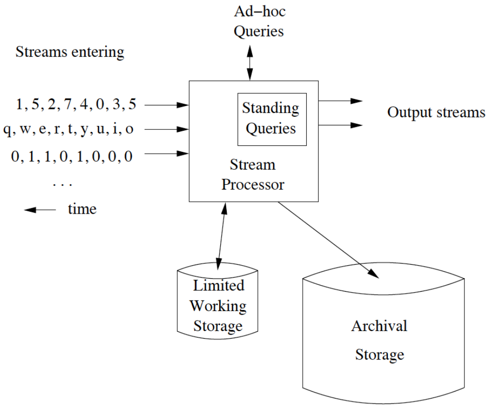
Mining Data Streams: Introduction I

- *Situation:* Data arrives in a stream (or several streams)
  - Too much to be put in active storage (main memory, disk, database)
  - If not processed immediately or stored (in inaccesible archives), lost forever
- *Algorithms* involve some summarization of stream(s); e.g.
  - create useful samples of stream(s)
  - filter the stream(s)
  - focus on windows of appropriate length (last n elements)

Data Streams: Examples

- Sensor data:
  - Ocean data (temperature, height): terabytes per day
  - Tracking cars (location, speed)
- Image data from satellites
- Internet/web traffic
  - Switches that route data also decide on denial of service
  - Tracking trends via analyzing clicks

Data Stream Management System



A data stream management system

Adopted from `mmds.org`

DATA STREAM QUERIES

- *Standing queries*
  - need to be answered throughout time
  - Answers need to be updated when they change
  - Example: current or maximum ocean temperature
- *Ad-hoc queries*
  - ask immediate questions
  - *Example:* number of unique users of a web site in the last 4 weeks
  - Not all data can be stored/processed
    ☞ Only certain questions feasible
  - Need to prepare for queries
    ☞ For example, store data from sliding windows

DATA STREAM QUERIES

Issues

- Streams deliver elements rapidly: need to act quickly
- Thus, data to work on should fit in main memory
- New techniques required:

☞ Compute approximate, not exact answers

☞ Hashing is a useful technique

Sampling Elements from a Stream

SAMPLING ELEMENTS

- *Situation:*
  - Select subsample from stream to store
  - Subsample should be representative of stream as a whole
- *Running Example:*
  - Search engine processes stream of search queries
  - Stream consists of tuples $(\text{user}, \text{query}, \text{time})$
  - Can store only $1/10$-th of data
  - Consider search queries

$$(\text{user}_1, \text{query}_1, \text{time}_1) \quad \text{and} \quad (\text{user}_2, \text{query}_2, \text{time}_2)$$

    Search query is *repeated* iff

$$\text{user}_1 = \text{user}_2 \quad \text{and} \quad \text{query}_1 = \text{query}_2$$

  - *Stream Query:* Fraction of repeated search queries?

RUNNING EXAMPLE: PITFALL

- *Running Example:*
  - *Stream Query:* Fraction of repeated search queries?

Naive and bad approach

- For each query, generate random integer from $[0, 9]$
- Keep only queries if 0 was generated
- *Scenario:* Suppose a user has issued
  - s queries one time
  - d queries two times
  - no queries more than two times
- *Correct answer* is $\frac{d}{d+s}$

Running Example: Pitfall

- *Running Example:*
  - *Stream Query:* Fraction of repeated search queries?

Naive and bad approach

- *Correct answer* is $\frac{d}{d+s}$
- But on randomly selected queries, we see that
  - Of one-time queries, $s/10$ appear to show once
  - Of two-time queries, $d/10 \times d/10$ appear to show twice
  - Of two-time queries, $d(1/10 \times 9/10) \times 2$ appear to show once
  - Resulting in *estimate*

$$\frac{0.01d}{(0.1s + 0.18d) + 0.01d} = \frac{d}{10s + 19d}$$

for repeated search queries, which is wrong for positive s, d

Running Example: Pitfall

- *Running Example:*
  - *Stream Query:* Fraction of repeated search queries?

Better approach

- For each user (not query!), generate random integer from $[0, 9]$
- Keep $1/10$th of users, e.g. if 0 was generated
- Implement randomness by hashing users to 10 buckets
  - ☞ avoids storing for each user whether he was in or out
- For maintaining sample for a/b-th of data, use b buckets, and keep users in buckets 0 to $a - 1$

Running Example: Pitfall

Better approach

- *General Sampling Problem:* Generalize from one-valued key to arbitrary-valued keys, keep a/b-th of (multi-valued) keys by the same technique
- *Reducing sample size:* On increasing amounts of data, ratio of data used for sample to be lowered
  - When lowering is necessary, decrease a by 1, so 0 to $a - 2$ are still accepted
  - Remove all elements with keys hashing to $a - 1$

Filtering Streams

Filtering Streams: Motivating Example

- *Problem:* Filter for data for which certain conditions apply
- Can be easy: data are numbers, select numbers of at most 10
- *Challenge:*
  - There is a set S that is too large to fit in main memory
  - Condition is too check whether stream elements belong to S
- *Illustration:*
  - Let $S = \{y_1, ..., y_m\}$, where m is huge
  - Incoming data stream: $..., x_n, .x_{n-1}, ..., x_1, x_0 \longrightarrow$
  - Check for each $x_i, i \geq 0$ whether $x_i \in S$
  - That is, for each x_i look up whether there is $j \in \{1, ..., m\}$ such that

$$x_i = y_j \in S$$

Filtering Streams: Motivating Example

Motivating Example: Email Spam

- Streamed data: pairs (email address, email text)
  - Set S is one billion (10^9) *approved (no spam!) addresses*
  - Only process emails from these addresses
    ☞ need to determine whether arbitrary address belongs to them
  - *But*, addresses cannot be stored in main memory
- *Option 1:* make use of disk accesses
- *Option 2 (preferrable):* Devise method without disk accesses, and determine set membership correctly in majority of cases
- *Solution:* "Bloom Filtering"

Bloom Filtering: Running Example

- Assume that main memory is 1 GB
- Bloom filtering: use main memory as bit array

$$[x_1, x_2, ..., x_{n-1}, x_n], n = 8 \cdot 10^9, x_i \in \{0, 1\}$$

  Recall that one byte is 8 bits.
- Devise hash function h that hashes email addresses a to buckets

$$h(a) = k \in \{1, 2, ..., 8 \cdot 10^9\}$$

  where k corresponds to bucket number
- Hash each member of S (allowed email addresses) to one of the buckets

Bloom Filtering: Running Example

- Hash each member a of S (allowed email addresses) to one of the buckets
- Set bits of hashed-to buckets to 1, leave other bits 0
- After hashing members of S

$$x_i = \begin{cases} 1 & \text{if there is } a \text{ s.t. } h(a) = i \\ 0 & \text{otherwise} \end{cases}$$

- About 1/8-th of bits are 1
- *Main memory status, afterwards*, for example,

$$[0, 0, 1, ..., 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, ..., 1, 0, 0, 0]$$

with seven times more zeroes than ones.

BLOOM FILTERING: RUNNING EXAMPLE

- Hash any new email address a_{new}:
  - If hashed-to bit is 1

    $$h(a_{new}) = i \quad \text{where} \quad x_i = 1$$

    classify address as no spam
  - If hashed-to bit is 0

    $$h(a_{new}) = i \quad \text{where} \quad x_i = 0$$

    classify address as spam
- Each address hashed to 0 is indeed spam
- *But* not every a_{new} hashed to 1 is no spam

Bloom Filtering: Running Example

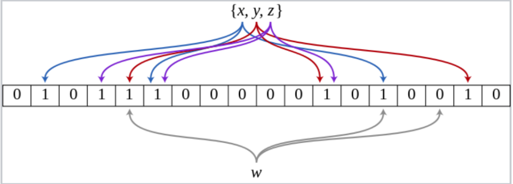
- Each address hashed to 0 is indeed spam
- *But* not every a_{new} hashed to 1 is no spam
- *Because* about 1/8-th of spam emails hash to 1 by chance
- *Still:* 80% of emails are spam, filtering out 7/8-th big deal
- *Filter cascade:* filter out 7/8-th of (remaining) spam in each step

Bloom Filter: Definition

Definition [Bloom Filter]

A *Bloom filter* consists of

- A bit array B of n bits, initially all zero
- A set S of m key values
- Hash functions $h_1, \ldots, h_k$ hashing key values to bits of B
  - ☞ Number of buckets is n



A Bloom filter for set $S = \{x, y, z\}$ using three hash functions

$m = 3, k = 3, n = 18$

From Wikipedia, by David Eppstein

Bloom Filter: Definition

Definition [Bloom Filter]

A *Bloom filter* consists of

- A bit array B of n bits, initially all zero
- A set S of m key values
- Hash functions $h_1, ..., h_k$ hashing key values to bits of B

Bloom Filter Workflow

- *Initialization*
  - Take each key value $K \in S$
  - Set all bits $h_1(K), ..., h_k(K)$ to one
- *Testing keys:*
  - Take key K_{new} to be tested: $K_{new} \in S$?
  - Declare $K_{new} \in S$ if all $h_1(K_{new}) = ... = h_k(K_{new}) = 1$ are one

Bloom Filtering: Analysis

- If $K \in S$, all $h_1(K), ..., h_k(K)$ are one, so K passes
- If $K \notin S$, all $h_1(K), ..., h_k(K)$ could be one, so K mistakenly passes
  ☞ *False positive!*
- *Goal:* Calculate probability of false positives
- *For that,* calculate probability that bit is zero after initialization

Bloom Filtering: Analysis

- Calculate probability that bit is zero after initialization
- *Relates to* throwing y darts at x targets, where
- Targets are bits in array, so $x = n$, so target is

$$[x_1, ..., x_n]$$

- Darts are members in S ($= m$) times hash functions ($= k$)

$$h_l(y_j), l = 1, ..., k; j = 1, ..., m$$

  so $y = km$
- Dart hitting target:

$$h_l(y_j) = x_i$$

  ☞ What is the probability that target is not hit by any dart?

Bloom Filtering: Analysis

Throwing y darts at x targets:

- Probability that a given dart will not hit a given target is $(x-1)/x$
- Probability that none of the y darts will hit a given target is

$$(\frac{x-1}{x})^y = (1-\frac{1}{x})^{x\frac{y}{x}} \tag{1}$$

- By $(1-\epsilon)^{1/\epsilon} = 1/e$ for small ϵ, we obtain that (1) is $e^{-y/x}$
- $x = n, y = km$: probability that a bit remains 0 is $e^{-km/n}$
- Would like to have fraction of 0 bits fairly large
- If k is about n/m, then probability of a 0 is e^{-1} (about 37%)
- In general, false positive comes from hitting k 1-bits by chance
- This evaluates as

$$(1-e^{-\frac{km}{n}})^k \tag{2}$$

UNIVERSITÄT
BIELEFELD

Counting Distinct Elements

The Flajolet-Martin Algorithm

Counting Distinct Elements: Problem

- *Problem:* Elements in streams can be identical
- *Question:* How many different elements has the stream brought along?
- *Model:* Consider the universal set of all possible elements
- Consider stream as a subset of the universal set
- *Question becomes:* What is the cardinality (size) of this subset?
- *Example:* Unique users of website
  - Amazon: determine number of users from user logins
  - Google: determine number of users from search queries

Counting Distinct Elements: Problem

- *Situation:* Stream picks elements from universal set
- *Question:* Size of subset of elements appearing in stream?
- *Obvious, but expensive:*
  - Keep stream elements in main memory
  - Store them in efficient search structure (hash table, search tree)
  - Works for sufficiently small amounts of distinct elements
- *If too many distinct elements, or too many streams:*
  - Use more machines ☞ Ok if affordable
  - Use secondary memory (disk) ☞ slow
  - *Here:* Estimate number of distinct elements using much less main memory than needed for storing all distinct elements
  - The *Flajolet-Martin algorithm* does this job

The Flajolet-Martin Algorithm

- *Central idea:* Hash elements a to bit strings $h(a)$ of sufficient length
  - For example, to hash URL's, 64-bit strings are sufficiently long

$$h(\text{URL}) = y_1 y_2 ... y_{63} y_{64}, y_j \in \{0, 1\}, j = 1, ..., 64$$

- *Intuition:*
  - The more different elements, the more different bit strings
  - The more different bit strings, the more "unusual" bit strings
  - Unusual here = bit string starts with many zeroes

$$y_j = 0, j = 1, ..., m \quad \text{that is} \quad y = \underbrace{00...00}_{m \text{ times}}101000...$$

where m is sufficiently large

The Flajolet-Martin Algorithm

Definition [Tail Length]

- Let h be the hash function that hashes stream elements a to bit strings $h(a)$
- The *tail length* of $h(a)$ is the number of zeroes by which it begins
- *Alternatively:* $h(a)$ number of zeroes a string ends with

Flajolet Algorithm

- Let A be the set of stream elements
- Let

$$R := \max_{a \in A} h(a) \tag{3}$$

  be the maximum tail length observed among stream elements
- *Estimate* 2^R for the number of distinct elements in the stream

Flajolet-Martin Algorithm: Example

15 users

User	Hashed Bitstring
sean	01111101
todd	11010001
aaron	10000111
kat	01110001
don	01011010
sara	01000001
linda	01010011
eric	00001001
jack	01101001
steph	10001100
terry	00111110
tim	00010000
wanda	11110001
chris	01101110
jane	00010010

→ Approximate Count $= 2^4 = 16$

Because the longest leading sequence of zeros is 4 bits long, we can say that there may be approximately 16 users

Hashing user names to 8-bit strings

From `towardsdatascience.com`

Flajolet-Martin Algorithm: Explanation

- Probability that bit string $h(a)$ starts with r zeroes is 2^{-r}
- Probability that none of m distinct elements has tail length at least r is

$$(1-2^{-r})^m = ((1-2^{-r})^{2^r})^{m2^{-r}} \overset{(1-\epsilon)^{1/\epsilon}\approx 1/e}{=} e^{-m2^{-r}} \tag{4}$$

- Let $P_{m,r} := 1-(1-2^{-r})^m \approx 1-e^{-m2^{-r}}$ be the probability that for m stream elements, the maximum tail length R observed is at least r.
- Conclude:
  - For $m >> 2^r$, it holds that $P_{m,r}$ approaches 1
  - For $m << 2^r$, it holds that $P_{m,r}$ approaches 0
  - So, 2^R is unlikely to be much larger or much smaller than m

Flajolet-Martin Algorithm: Combining Estimates

- *Idea:* Use several hash functions $h_k, k = 1, ..., K$
- Combine their estimates $X_k, k = 1, ..., K$
- *Pitfall 1: Averaging*
  - Let p_r be the probability that the maximum tail length of h_k is r
  - One can compute that

$$p_r \geq \frac{1}{2} p_{r-1} \geq ... \geq 2^{-r+1} p_1 \geq 2^{-r} p_0$$

  - So $E(X_k)$, the expected value of X_k computes as

$$E(X_k) = \sum_{r \geq 0} p_r 2^r \geq p_o \sum_{r \geq 0} 2^{-r} 2^r = p_0 \sum_{r \geq 0} 1 = \infty$$

  - Therefore $\frac{1}{K} \sum_{k=1}^{K} E(X_k)$ the expected value of the average of the X_k turns out to be infinite as well
  - *Conclusion:* Overestimates spoil averaging

Flajolet-Martin Algorithm: Combining Estimates

- *Idea:* Use several hash functions $h_k, k = 1, ..., K$
- Combine their estimates $X_k, k = 1, ..., K$
- *Pitfall 2: Computing Medians*
  - The median is always a power of two
    ☞ makes only very limited sense
- *Solution:*
  - Group the hash functions into small groups and take averages within groups
  - Estimate m as median of group averages
  - Groups should be of size $C \log_2 m$ for some small C
- *Space Requirements:* Need to store only value of X_k, requiring little space as a maximum

MATERIALS / OUTLOOK

- See *Mining of Massive Datasets*: sections 4.1–4.4
- As usual, see `http://www.mmds.org/` in general for further resources
- For deepening your understanding, consider voluntary *homework*: read 2.6.7 and try to make sense of this
- Next lecture: "Mining Data Streams II / PageRank I"
  - See *Mining of Massive Datasets* 4.5–4.7