Mining Streams

Shantanu Jain

Data Streams

- In many data mining situations, we do not know what data will arrive in advance

- Stream Management is important when the input rate is controlled externally:
  - Google queries
  - Twitter or Facebook status updates

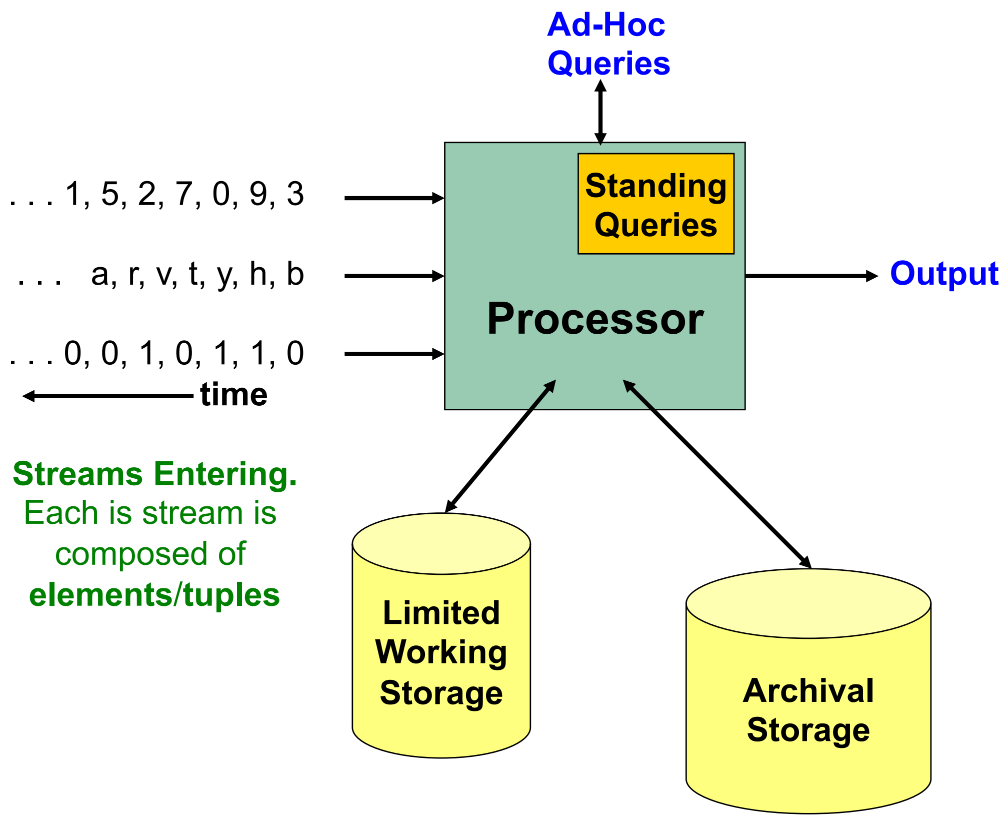
- Can think of streams as infinite and non-stationary (the distribution changes over time)

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

The Stream Model

- Input elements enter at a rapid rate,
  at one or more input ports (i.e., streams)
  - We often represent elements as tuples
- The system cannot store the entire stream
- Q: How do you make calculations about
  the stream using a limited amount
  of (secondary) memory?

General Stream Processing Model



- Common Types of Queries:
  - Sampling data from a stream
    - Construct a random sample
  - Queries over sliding windows
    - Number of items of type ***x*** in the last ***k*** elements of the stream
  - Filtering a data stream
    - Select elements with property ***x*** from the stream
  - Counting distinct elements
    - Number of distinct elements in last ***k*** elements of the stream
  - Estimating moments
    - Estimating frequency/surprise

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

Applications (1)

- Mining query streams
  - Google wants to know what queries are more frequent today than yesterday
- Mining click streams
  - Wikipedia wants to know which of its pages are getting an unusual number of hits in the past hour
- Mining social network news feeds
  - E.g., look for trending topics on Twitter, Facebook

Applications (2)

- Sensor Networks
  - Many sensors feeding into a central controller
- Telephone call records
  - Data feeds into customer bills as well as settlements between telephone companies
- IP packets monitored at a switch
  - Gather information for optimal routing
  - Detect denial-of-service attacks

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

Mining Streams

Shantanu Jain

Sampling from Streams

Sampling from a Data Stream

- Since **we can not store the entire stream**, one obvious approach is to store a sample (i.e. a random subset of elements)

- Two different approaches:
  1. Sample a fixed fraction of elements (say 1 in 10)
  2. Maintain a random sample of fixed size over a potentially infinite stream
     - At "any time" ***k*** we would like a random sample of ***s*** elements
       - For all time steps ***k***, each of ***k*** elements so far should have an equal probability of being included in the ***s*** elements

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

Approach 1: Sampling a Fixed Proportion

- Scenario: Search engine query stream
  - Stream of tuples: (user, query, time)
  - Answer questions such as: How many queries from a typical user in past 30 days are repeat queries.
  - Have space to store **1/10th** of query stream

- Naïve solution: Random subsampling
  - Generate a random number **$u \sim$ Uniform([0, 1))**
  - Store the query if **$u < 0.1$**, discard if **$u \geq 0.1$**

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

Problem with Naïve Approach

- Suppose each user issues ***x*** number of queries once and ***d*** number of queries twice (total of ***x*+2*d*** queries)
  - True Fraction of Duplicates (unknown): *d* / (*x*+*d*)

- Naive Estimate: Keep 10% of the queries
  - Sample will contain ***x* / 10** of the singleton queries and **2*d* / 10** of the duplicate queries at least once
  - But only ***d* / 100** pairs of duplicates

    d/100 = 1/10 · 1/10 · d
  - Of ***d*** "duplicates" ***18d / 100*** appear exactly once

    18d / 100 = ((1/10 · 9/10) + (9/10 · 1/10)) · d

Fraction in Sample $\dfrac{\frac{d}{100}}{\frac{x}{10}+\frac{d}{100}+\frac{18d}{100}} = \dfrac{\boldsymbol{d}}{\mathbf{10}\boldsymbol{x}+\mathbf{19}\boldsymbol{d}}$

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

Problem with Naïve Approach

- Suppose each user issues n_x number of queries once and n_d number of queries twice (total of $n_x + 2n_d$ queries)
  - True Fraction of Duplicates (unknown): $n_d/(n_x + n_d)$

- Naive Estimate: Keep 10% of the queries
  - Sample will contain $n_x/10$ of the singleton queries and $2n_d/10$ of the duplicate queries at least once
  - But only $n_d/100$ pairs of duplicates

    $n_d/100 = 1/10 \cdot 1/10 \cdot n_d$
  - Of *d* "duplicates" $18n_d / 100$ appear exactly once

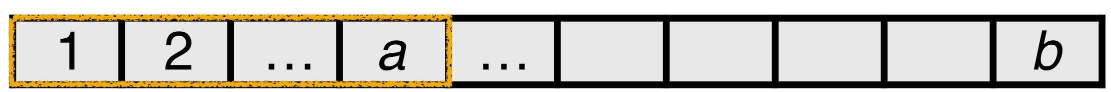
    $18n_d / 100 = ((1/10 \cdot 9/10) + (9/10 \cdot 1/10)) \cdot n_d$

Fraction in Sample $\dfrac{\frac{n_d}{100}}{\frac{n_x}{10} + \frac{n_d}{100} + \frac{18n_d}{100}} = \dfrac{n_d}{10n_x + 19n_d}$

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

Better Solution: Sample Keys

- Assume tuples with keys:
  - Key is some subset of each tuple's components
    - e.g., tuple is **(user, search, time)**; key is user
  - Choice of key depends on application
- To get a sample of a / b fraction of the stream:
  - Hash each tuple's key uniformly into *b* buckets
  - Pick the tuple if its hash value is at most *a*



Hash table with **b** buckets, pick the tuple if its hash value is at most **a.**
How to generate a 30% sample?
Hash into b=10 buckets, take the tuple if it hashes to one of the first 3 buckets

Approach 2: Fixed-size Sample

- Suppose we want to maintain a random sample ***S*** of size exactly ***s*** tuples
  - E.g., main memory size constraint
  - Why? May not know length of stream in advance
- Goal: Ensure equal probability of inclusion
  - Suppose at time ***n*** we have seen ***n*** items
  - Each item should occur in the sample ***S*** with probability ***s / n***

Approach 2: Fixed-size Sample

- Algorithm (a.k.a. Reservoir Sampling)
  - Store all the first ***s*** elements of the stream to ***S***
  - Suppose we have seen ***n-1*** elements, and now the n^{th} element arrives ($n > s$)
    - With probability s / n, keep the n^{th} element, else discard it
    - If we picked the n^{th} element, then it replaces one of the ***s*** elements in the sample ***S***, picked uniformly at random

- Claim: This algorithm maintains a sample ***S*** with the desired property:
  - After ***n*** elements, the sample contains each element seen so far with probability s / n

Proof: By Induction

- We prove this by induction:
  - Assume that after *n* elements, the sample contains each element seen so far with probability *s/n*
  - We need to show that after seeing element *n+1* the sample maintains the property
    - Sample contains each element so far with prob *s/(n+1)*
- Base case:
  - After we see *n=s* elements the sample *S* has the desired property
    - Each out of *n=s* elements is in the sample with probability s/s = 1

Proof: By Induction

- Inductive hypothesis: After *n* elements, the sample *S* contains each element seen so far with prob. *s / n*
- Inductive step: When element *n+1* arrives, the probability for retention of each of the first *n* elements is

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

Element **n+1** discarded — Element **n+1** *not* discarded — Element in ***S*** *not* replaced

The probability for retention after *n*+1 steps is

First ***n*** Elements

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

Retained after ***n*** steps — Retained in step ***n*+1**

New Element

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

Element **n+1** *not* discarded

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

Mining Streams

Shantanu Jain

Counting with Exponetially Decaying Windows

Sliding Windows

- One model for stream processing is to apply queries to a *window* of ***N*** most recent elements

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

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

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

← Past Future →

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

Sliding Windows

- Difficult case: Window size ***N*** is so large that the data cannot be stored in memory, or even on disk
  - Or, there are so many streams that windows for all cannot be stored
- Amazon example:
  - For every product ***X*** we keep **0/1** stream of whether that product was sold in the **n-th** transaction
  - We want answer queries, such as finding frequent items that were sold more than ***s*** times

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

Exponentially Decaying Windows

- Sliding window: Count occurrences in last ***N*** elements

$$\sigma_t^{\mathrm{SW}}(x) = \sum_{i=t-N}^{t} I[a_i = x] \qquad I[a_i = x] = \begin{cases} 1 & a_i = x, \\ 0 & a_i \neq x. \end{cases}$$

"Indicator" function: returns 1 when query matches, 0 when not

- Exponentially decaying window: Give lower "weight" to occurences that are farther back in time

$$\sigma_t^{\mathrm{SDW}}(x) = \sum_{i=1}^{t} I[a_i = x](1-c)^{t-i}$$

c is a small constant (e.g. **0.001**) such that **(1-*c*)** is close to **1**, but **$(1\text{-}c)^{t\text{-}i}$** decays to **0** when **$t \gg i$**

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

Exponentially Decaying Windows

- **Convenient Property:** Can compute sum at time ***t*** from sum at time ***t-1***

$$\sigma_t^{\mathrm{EDW}}(x) = \sum_{i=1}^{t} I[a_i = x]\,(1-c)^{t-i}$$

$$= \underbrace{I[a_t = x]}_{\text{Term for } i = t} + \underbrace{\sum_{i=1}^{t-1} I[a_i = x](1-c)^{t-i}}_{\text{Terms for } i < t}$$

$$= I[a_t = x] + (1-c)\,\sigma_{t-1}^{\mathrm{EDW}}(x)$$

Don't need to keep items $\mathbf{a_1, \ldots, a_t}$ in memory,
just need to keep track of running weights $\boldsymbol{\sigma(x)}$

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

Counting Items with Decaying Windows

$$\sigma_t^{\mathrm{EDW}}(x) = I[a_t = x] + (1-c)\,\sigma_{t-1}^{\mathrm{EDW}}(x)$$

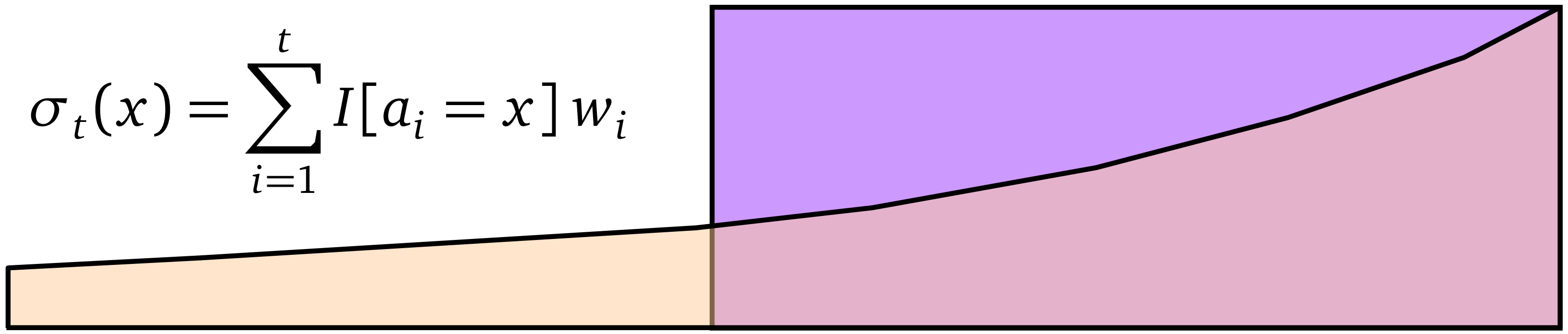
- Initialization: Set **σ(x) = 0** for all items x in some set **X**
- For each new item **a**
  - Apply decay factor to weights **σ(x) = (1-c) σ(x)**
  - Increment weight for current item **σ[a] = σ[a]+1**

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

Sliding vs Exponential Windows

- Sliding and Exponential windows compute a weighted sum

$$\sigma_t(x) = \sum_{i=1}^{t} I[a_i = x]\, w_i$$



- What differs is the definition of the weights

Sliding $w_i = \begin{cases} 1 & i > t - N, \\ 0 & i \leq t - N. \end{cases}$

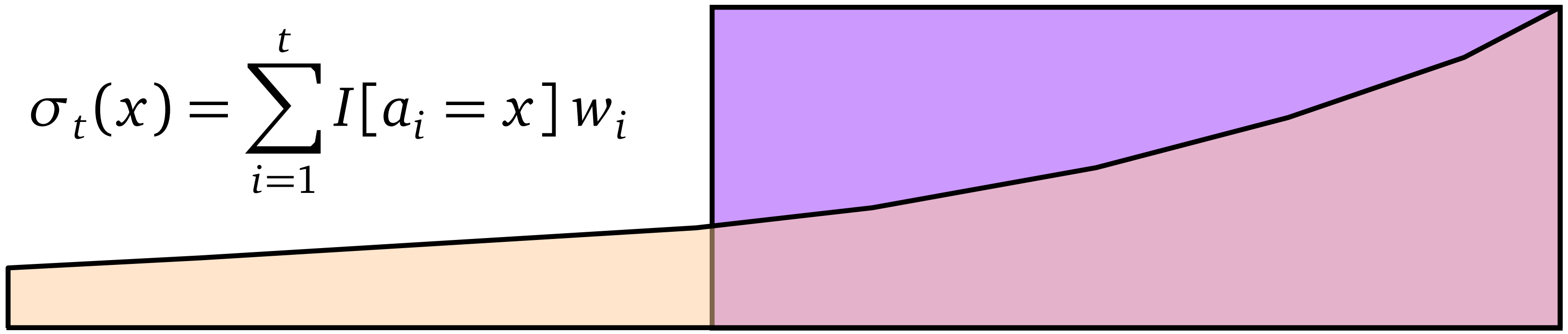
Exponential Decaying $w_i = (1-c)^{t-i}$

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

Sliding vs Exponential Windows

- Sliding and Exponential windows compute a weighted sum

$$\sigma_t(x) = \sum_{i=1}^{t} I[a_i = x]\, w_i$$



- In a sliding window, the sum of the weights is ***N***
- In a exponentially window, the sum is a *geometric series*

$$\lim_{t\to\infty} \sum_{i=1}^{t} w_i = \lim_{t\to\infty} \sum_{i=1}^{t} (1-c)^{t-i} = \frac{1}{1-(1-c)} = \frac{1}{c}$$

We can think of **1/c** as the "effective window size"

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