Mining Data Streams

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Mina Ghashami, Amazon
http://cs246.stanford.edu

New Topic: Infinite Data

High dim. data	Graph data	**Infinite data**	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	Decision Trees	Recommen der systems
Clustering	Community Detection	Queries on streams	SVM	Association Rules
Dimensional ity reduction	Spam Detection	Web advertising	Parallel SGD	Duplicate document detection

So far

- So far we have worked datasets or data bases where all data is available
- In contrast, in **data streams**, data arrives one element at a time often at a **rapid rate** that:
  - If it is not **processed immediately** it is lost forever.
  - It is not feasible to **store** it all

Data Streams

- **In many data mining situations, we do not know the entire data set in advance**
- **Stream Management** is important when the input rate is controlled **externally:**
  - Google queries
  - Twitter posts or Facebook status updates
  - e-Commerce purchase data.
  - Credit card transactions
- Think of the **data** as **infinite** and **non-stationary** (the distribution changes over time)
  - This is the fun part and why interesting algorithms are needed

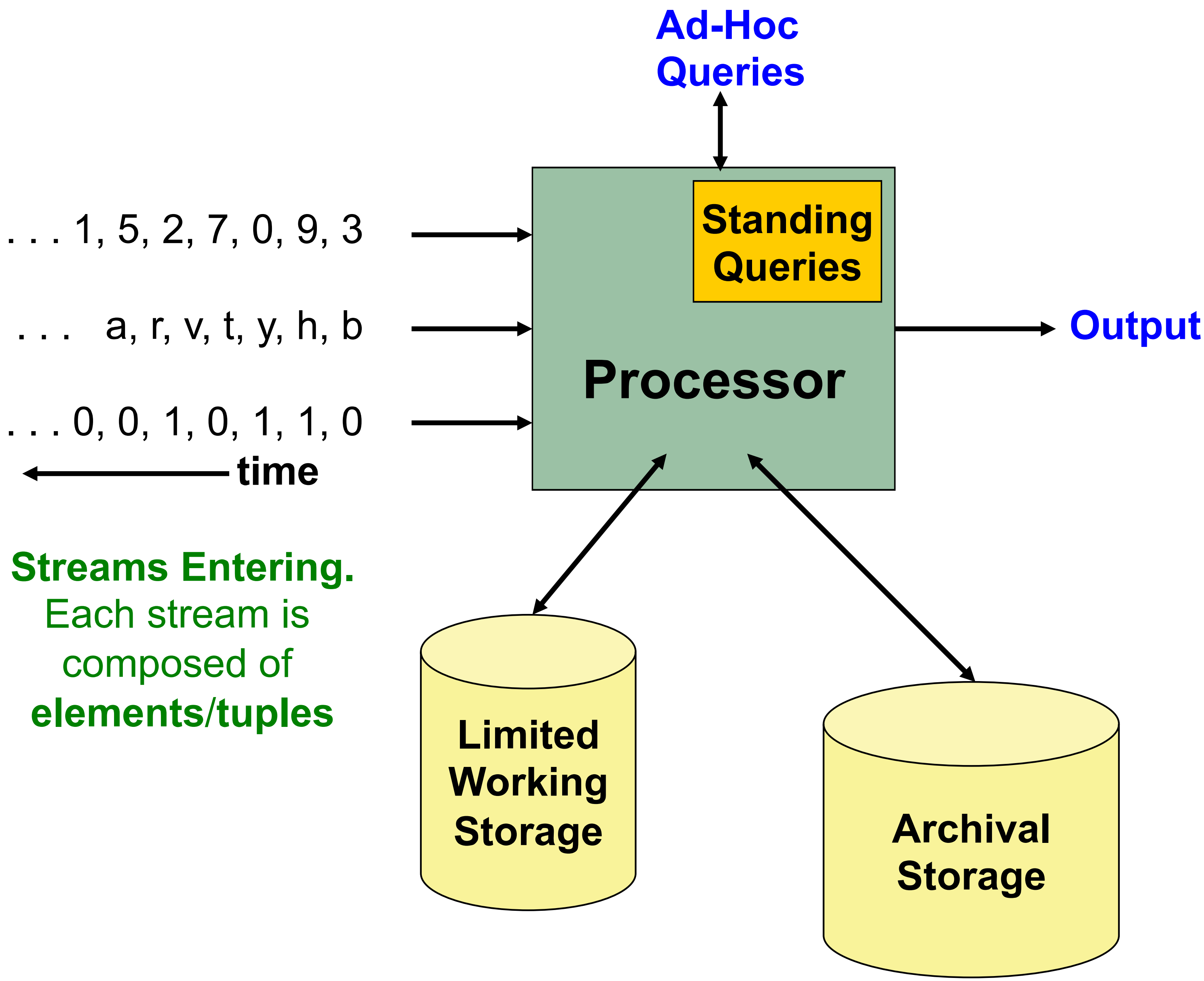
The Stream Model

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

Side note: SGD is a Streaming Alg.

- **Stochastic Gradient Descent (SGD) is an example of a streaming algorithm**
- **In Machine Learning we call this: Online Learning**
  - Allows for modeling problems where we have a continuous stream of data
  - We want an algorithm to learn from it and slowly adapt to the changes in data
- **Idea: Do small updates to the model**
  - **SGD** makes small updates
  - **So:** First train the classifier on training data
  - **Then:** For every example from the stream, we slightly update the model (using small learning rate)

General Stream Processing Model



Problems on Data Streams

- **Types of queries one wants to answer on a data stream:**
  - **Sampling data from a stream**
    - Construct a random sample
  - **Filtering a data stream**
    - Select elements with property ***x*** from the stream
  - **Counting distinct elements**
    - Number of distinct elements in the last ***k*** elements of the stream
  - **finding most frequent elements**

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**
  - 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

Sampling from a Data Stream: Sampling a fixed proportion

As the stream grows the sample also gets bigger

Sampling from a Data Stream

- Why is this important?
  - Since we cannot store the entire stream, a representative **sample** can act like the stream
- **Two different problems:**
  - **(1)** Sample a **fixed proportion** of elements in the stream (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 of the stream 1..*k*
      - **What is the property of the sample we want to maintain?** For all time steps ***k***, each of the ***k*** elements seen so far must have **equal probability** of being sampled

Sampling a Fixed Proportion

- **Problem 1: Sampling a fixed proportion**
  - E.g. sample 10% of the stream
  - As stream gets bigger, sample gets bigger

- **Naïve solution:**
  - Generate a random integer in **[0...9]** for each query
  - Store the query if the integer is **0**, otherwise discard

- **Any problem with this approach?**
  - We have to be very careful what query we answer using this sample

Problem with Naïve Approach

- **Scenario:** Search engine query stream
  - **Stream of tuples:** (user, query, time)
  - **Question:** What fraction of unique queries by an average user are duplicates?
    - Suppose each user issues ***x*** queries once and ***d*** queries twice (total of ***x*+2*d*** query instances) then the correct answer to the query is ***d/(x+d)***
  - **Proposed solution:** **We keep 10% of the queries**
    - Sample will contain **(*x+2d*)/10** elements of the stream
    - Sample will contain ***d*/100** pairs of duplicates
      - **d/100** = **1/10 · 1/10 · d**
    - There are ***(10x+19d)/100*** unique elements in the sample
      - **(*x+2d*)/10** - ***d*/100** = ***(10x+19d)/100***
  - **So the sample-based answer is** $\frac{\frac{d}{100}}{\frac{10x}{100}+\frac{19d}{100}} = \frac{d}{10x+19d}$

Problem with Naïve Approach

- **Scenario:** Search engine query stream
  - **Stream of tuples:** (user, query, time)
  - **Query:** What fraction of unique queries by an average user are duplicates?
    - Suppose each user issues ***x*** queries once and ***d*** queries twice (total of ***x*+2*d*** query instances) then the correct answer to the query is ***d/(x+d)***

Sample underestimates

  - **Proposed solution: We keep 10% of the queries**
    - Sample will contain ***(x+2d)/10*** elements of the stream
    - Sample will contain ***d/100*** pairs of duplicates
      - **d/100 = 1/10 · 1/10 · d**
    - There are ***(10x+19d)/100*** unique elements in the stream
      - ***(x+2d)/10 - d/100 = (10x+19d)/100***
  - **So the sample-based answer is** $\frac{\frac{d}{100}}{\frac{10x}{100}+\frac{19d}{100}} = \frac{d}{10x+19d}$

Solution: Sample Users

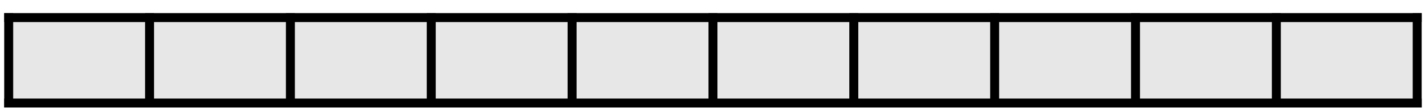
Solution:

- Don't sample queries, sample users instead
- Pick **1/10th** of **users** and take all their search queries in the sample

- Use a hash function that hashes the user name or user id uniformly into 10 buckets

Generalized Solution

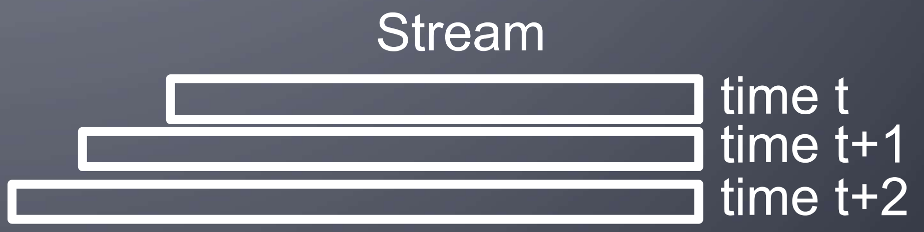
- **Stream of tuples with keys:**
  - Key is some subset of each tuple's components
    - e.g., tuple is (user, search, time); key is **user**
  - Choice of key depends on application
- **To get a sample of *a/b* fraction of the stream:**
  - Hash each tuple's key uniformly into ***b*** buckets
  - Pick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

Sampling from a Data Stream: Sampling a fixed-size sample

The sample is of fixed size



Maintaining a fixed-size sample

- **Problem 2: Fixed-size sample**
- **Suppose we need to maintain a random sample *S* of size exactly *s* tuples**
  - E.g., main memory size constraint
- **Why?** Don't know length of stream in advance
- **Suppose by time *n* we have seen *n* items**
  - **Each item is in the sample *S* with equal prob. *s/n***

How to think about the problem: say s = 2

Stream: a x c y z k c d e g…

At **n= 5,** each of the first 5 tuples is included in the sample **S** with equal prob.

At **n= 7,** each of the first 7 tuples is included in the sample **S** with equal prob.

Impractical solution would be to store all the *n* tuples seen so far and out of them pick *s* at random

Solution: 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***