IIT KHARAGPUR

IIT GANDHINAGAR

NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Scalable Data Science

Lecture 1: Introduction

Sourangshu Bhattacharya

Computer Science and Engineering

IIT KHARAGPUR

In this Lecture:

- Stream processing and sketching
- Dimensionality reduction and hashing
- Frameworks for big data computation
- Scalable Machine Learning

Stream processing and sketching

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 or Facebook status updates

- We can think of the **data** as **infinite** and **non-stationary** (the distribution changes over time)

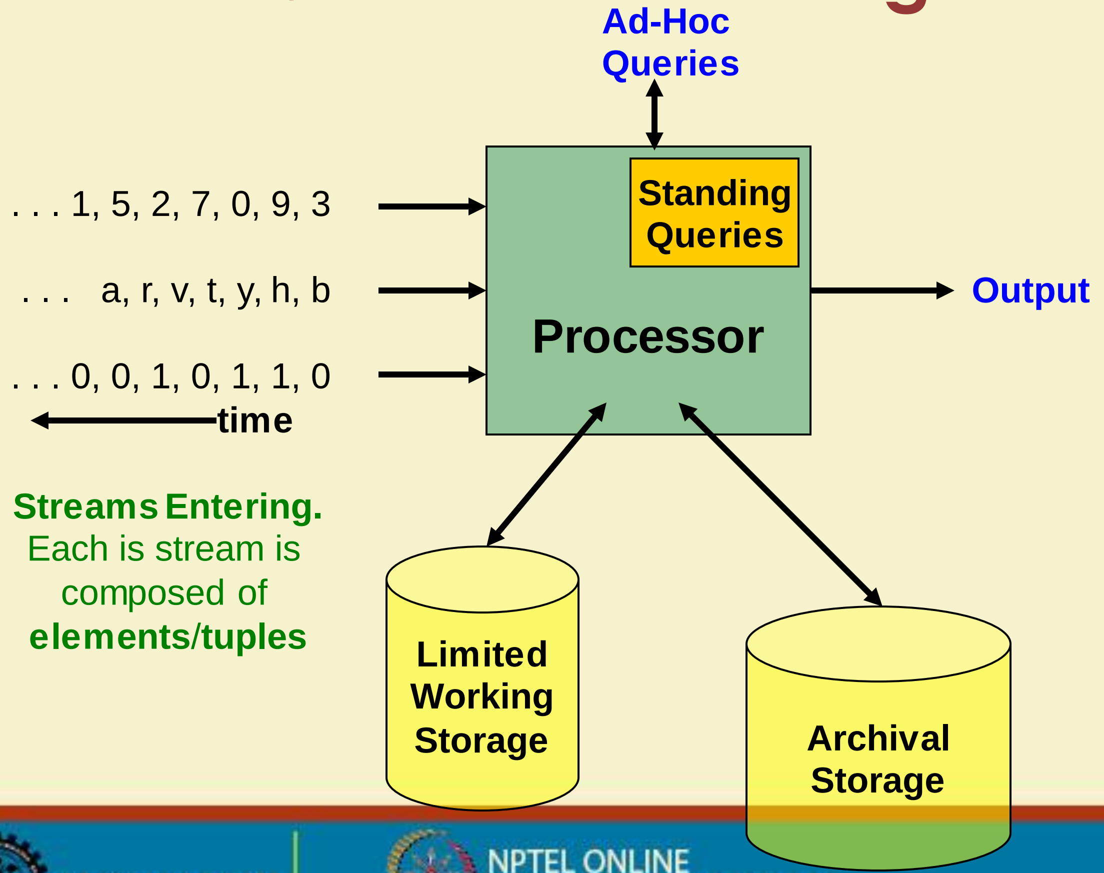
The Stream Model

- 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?**

General Stream Processing Model



Streams Entering. Each is stream is composed of **elements/tuples**

Problems on Data Streams

- **Types of queries one wants on answer on a data stream:**
  - **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

Sliding Windows

- A useful model of stream processing is that queries are about a ***window*** of length ***N*** – the ***N*** most recent elements received

- **Amazon example:**
  - For every product **X** we keep 0/1 stream of whether that product was sold in the **n**-th transaction
  - We want to answer queries, how many times have we sold **X** in the last **k** sales

Maintaining a fixed-size sample

- **Problem: 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 at 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***

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 seen so far with probability ***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***
- **Now element *n+1* arrives**
- **Inductive step:** For elements already in ***S***, probability that the algorithm keeps it in ***S*** is:

$$\underbrace{\left(1-\frac{s}{n+1}\right)}_{\text{Element } \mathbf{n+1} \text{ discarded}}+\underbrace{\left(\frac{s}{n+1}\right)}_{\text{Element } \mathbf{n+1} \text{ not discarded}}\underbrace{\left(\frac{s-1}{s}\right)}_{\text{Element in the sample not picked}}=\frac{n}{n+1}$$

- So, at time ***n,*** tuples in ***S*** were there with prob. **s/n**
- Time ***n→n+1,*** tuple stayed in ***S*** with prob. **n/(n+1)**
- So prob. tuple is in ***S*** at time ***n+1*** $= \frac{s}{n} \cdot \frac{n}{n+1} = \frac{s}{n+1}$

Problems on Data Streams

- **Types of queries one wants on answer on a data stream:**
  - **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
  - **Estimating moments**
    - Estimate avg./std. dev. of last ***k*** elements
  - **Finding frequent elements**

Applications (1)

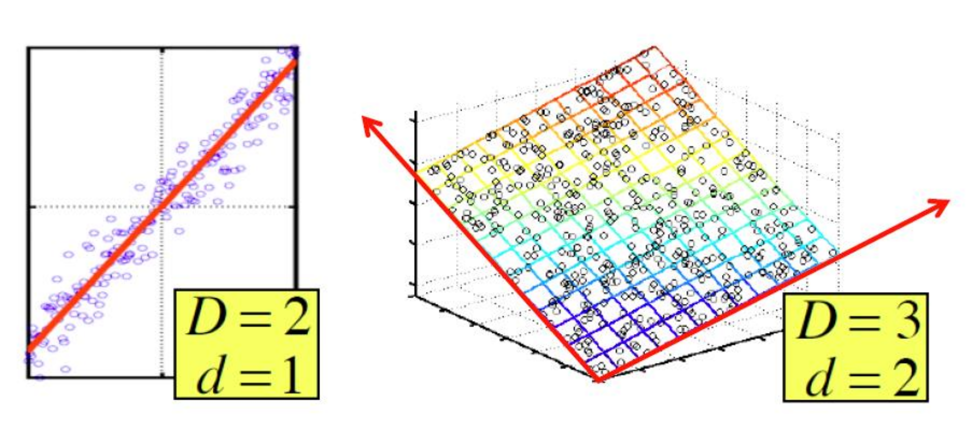
- **Mining query streams**
  - Google wants to know what queries are more frequent today than yesterday
- **Mining click streams**
  - A web company wants to know which of its pages are getting an unusual number of hits in the past hour
- **Mining social network news feeds**
  - E.g., look for trending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

Dimensionality reduction

Dimensionality Reduction



- **Assumption:** Data lies on or near a low *d*-dimensional subspace
- **Axes of this subspace are effective representation of the data**

Dimensionality Reduction

- **Compress / reduce dimensionality:**
  - 10^6 rows; 10^3 columns; no updates
  - Random access to any cell(s); **small error: OK**

day / customer	We 7/10/96	Th 7/11/96	Fr 7/12/96	Sa 7/13/96	Su 7/14/96
ABC Inc.	1	1	1	0	0
DEF Ltd.	2	2	2	0	0
GHI Inc.	1	1	1	0	0
KLM Co.	5	5	5	0	0
Smith	0	0	0	2	2
Johnson	0	0	0	3	3
Thompson	0	0	0	1	1

The above matrix is really "2-dimensional." All rows can be reconstructed by scaling [1 1 1 0 0] or [0 0 0 1 1]

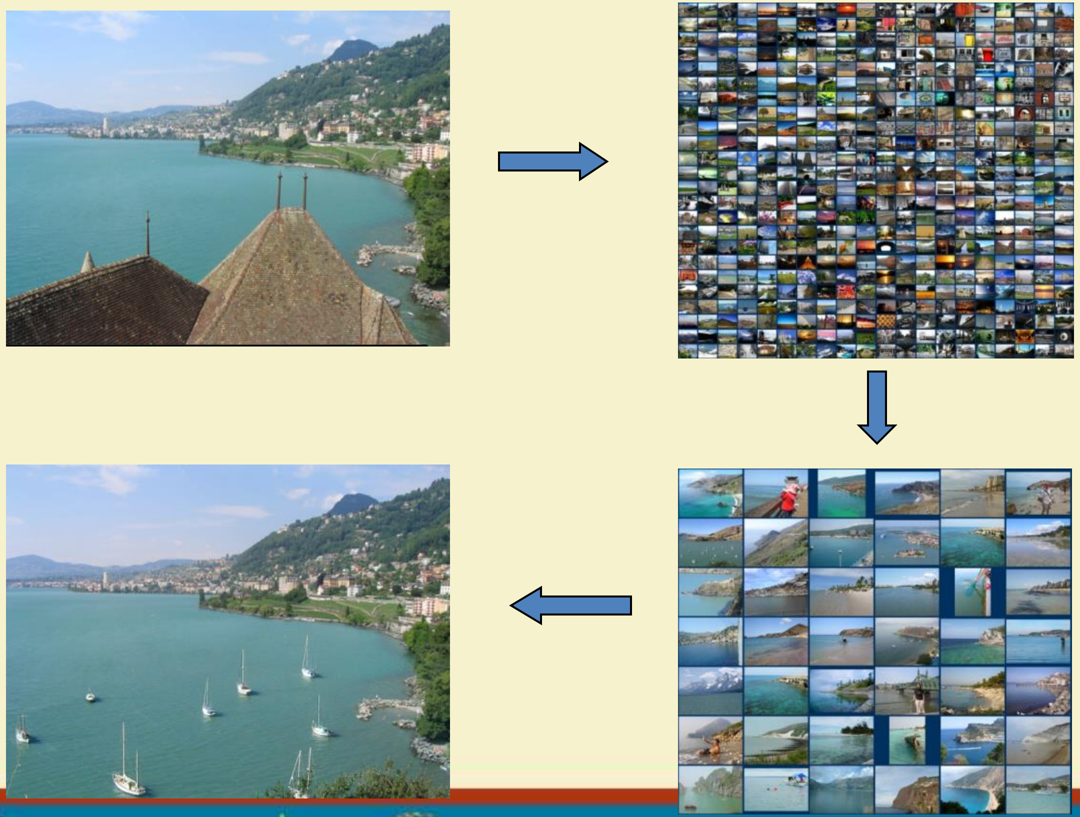
Why Reduce Dimensions?

Why reduce dimensions?

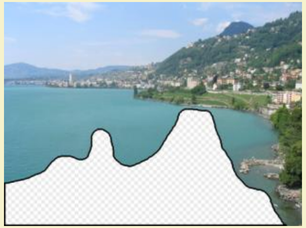
- **Discover hidden correlations/topics**
  - Words that occur commonly together
- **Remove redundant and noisy features**
  - Not all words are useful
- **Interpretation and visualization**
  - Genres of movies
- **Easier storage and processing of the data**

Locality sensitive hashing

Scene Completion Problem



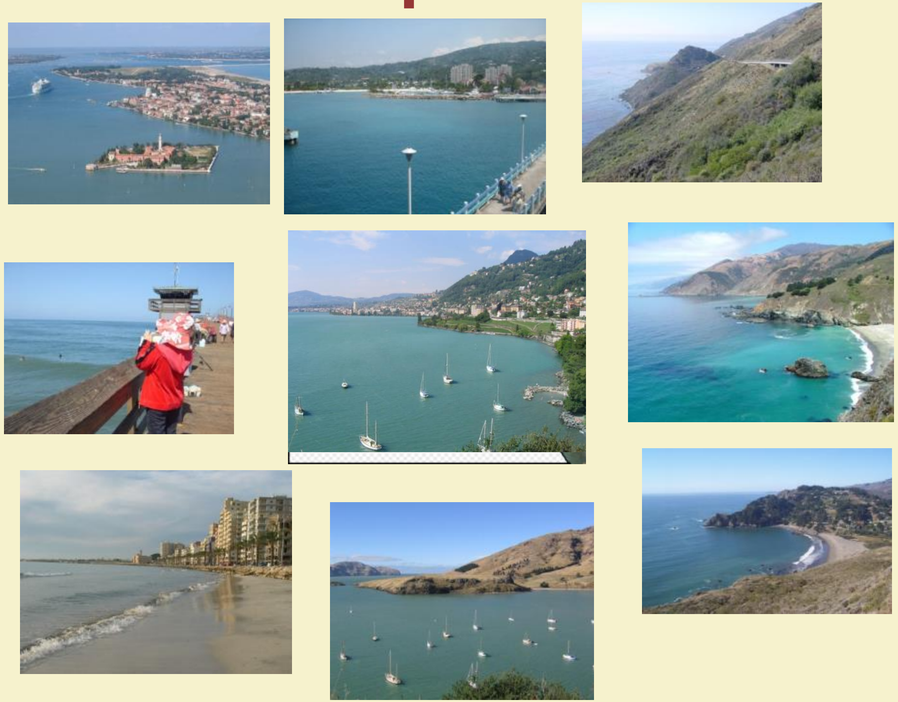
Scene Completion Problem



Scene Completion Problem

10 nearest neighbors from a collection of 20,000 images

Scene Completion Problem



10 nearest neighbors from a collection of 2 million images

A Common Metaphor

- **Many problems can be expressed as finding “similar” sets:**
  - **Find near-neighbors in high-dimensional space**
- **Examples:**
  - **Pages with similar words**
    - For duplicate detection, classification by topic
  - **Customers who purchased similar products**
    - Products with similar customer sets
  - **Images with similar features**
    - Users who visited similar websites

Problem definition

- **Given: High dimensional data points $x_1, x_2, \ldots$**
  - **For example:** Image is a long vector of pixel colors

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow [1\ 2\ 1\ 0\ 2\ 1\ 0\ 1\ 0]$$

- **And some distance function $d(x_1, x_2)$**
  - Which quantifies the “distance” between x_1 and x_2
- **Goal:** Find **all pairs of data points** (x_i, x_j) that are within some distance threshold $d(x_i, x_j) \leq s$
- **Note:** Naïve solution would take $O(N^2)$ ☹

  where N is the number of data points
- **MAGIC: This can be done in $O(N)$!! How?**

Frameworks for big data computation

MapReduce

- Much of the course will be devoted to **large scale computing** for **data mining**
- **Challenges:**
  - How to distribute computation?
  - Distributed/parallel programming is hard

- **Map-reduce** addresses all of the above
  - Google's computational/data manipulation model
  - Elegant way to work with big data

Example: Language Model

- **Statistical machine translation:**
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents

- **Very easy with MapReduce:**
  - **Map:**
    - Extract (5-word sequence, count) from document
  - **Reduce:**
    - Combine the counts

Example: Host size

- **Suppose we have a large web corpus**
- Look at the metadata file
  - Lines of the form: (URL, size, date, …)
- **For each host, find the total number of bytes**
  - That is, the sum of the page sizes for all URLs from that particular host

- **Other examples:**
  - Link analysis and graph processing
  - Machine Learning algorithms

Scalable Machine Learning

Big Data

- 6 Billion web queries per day.
  ~ 6 TB per day, ~ 2.5 PB per year
- 10 Billion display ads per day.
  ~ 15 TB per day, ~ 5.5 PB per year
- 30 Billion text ads per day.
  ~ 30 TB per day, ~ 11 PB per year
- 150 Million Credit card transactions per day.
  ~ 150 GB per day, ~ 5.5 TB per year
- 100 Billion emails per day.
  ~ 1 PB per day, ~ 360 PB per year

Machine Learning on Big Data

- 6 Billion web queries per day.
  ~ 6 TB per day, ~ 2.5 PB per year
- 10 Billion display ads per day.
  ~ 15 TB per day, ~ 5.5 PB per year
- 30 Billion text ads per day.
  ~ 30 TB per day, ~ 11 PB per year
- 150 Million Credit card transactions per day.
  ~ 150 GB per day, ~ 5.5 TB per year
- 100 Billion emails per day.
  ~ 1 PB per day, ~ 360 PB per year

- Ranking search results
  Training ranking algorithms from past searches
- Segmentation of customers e.g. "high income male"
  View count by customer segments
- Click through rate estimation
  Training logistic regression
- Fraudulent transactions
  Anomaly detection
- Personalised spam filtering
  Multi-task binary classification

Large Scale Machine Learning

- **Main question:**
  How to efficiently train
  (build a model/find model parameters)**?**

- **Auxiliary question: fast / scalable optimization**
  - **Stochastic / online optimization**
  - **Distributed optimization.**

References:

- Jure Leskovec, Anand Rajaraman, Jeff Ullman. **Mining of Massive Datasets.** *2nd edition. - Cambridge University Press.* http://www.mmds.org/

Thank You!!