Analytics and Visualization of Big Data

Fadel M. Megahed

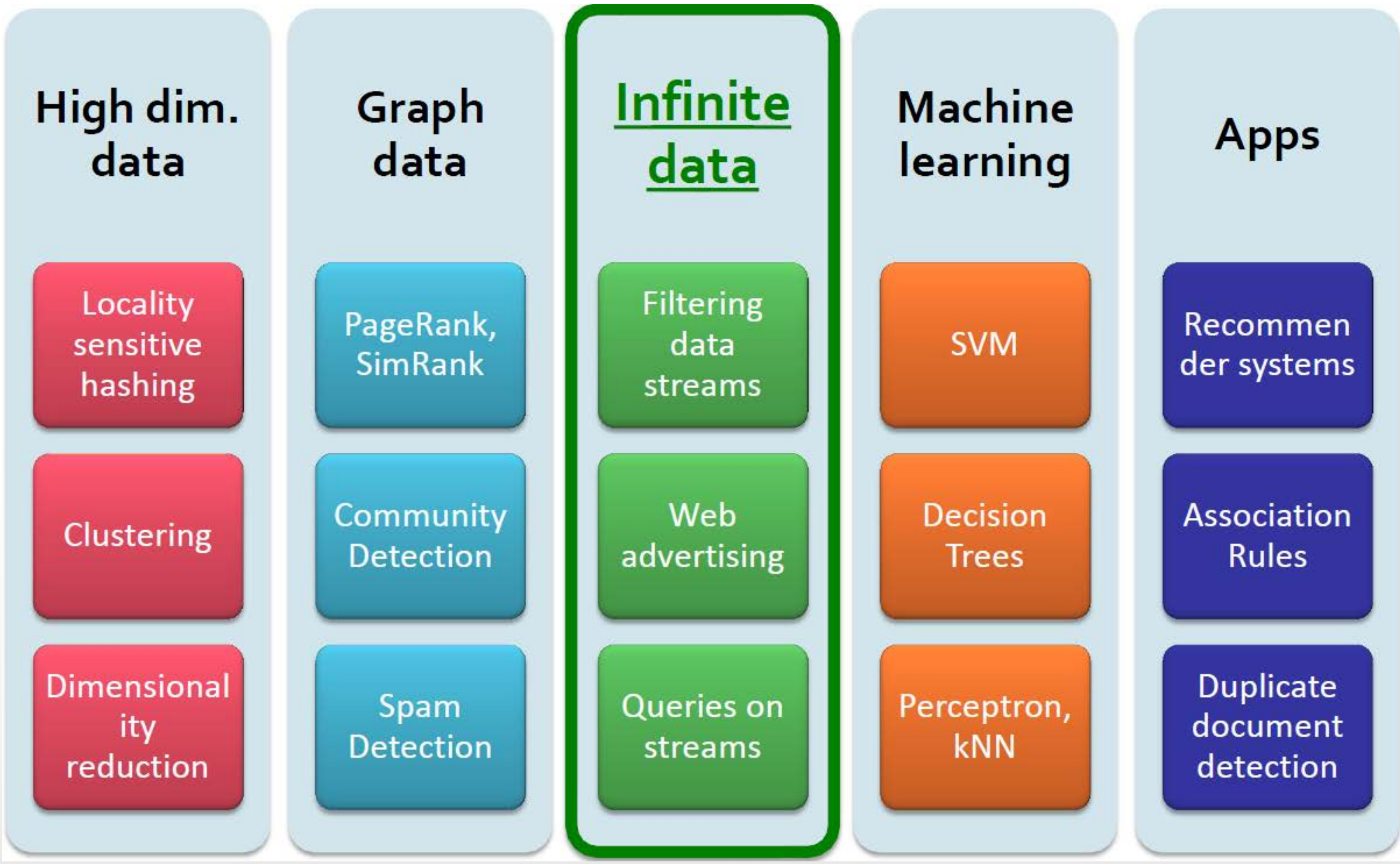
Lecture 26: Mining Data Streams

AUBURN UNIVERSITY
SAMUEL GINN
COLLEGE OF ENGINEERING

Department of Industrial and Systems Engineering

Spring 13

Final Topic: Infinite Data



Source: Jure Leskovic, Stanford CS246, Lecture Notes, see http://cs246.stanford.edu

Mining Data Streams

- Most of the algorithms that we described assumed that we are mining a database

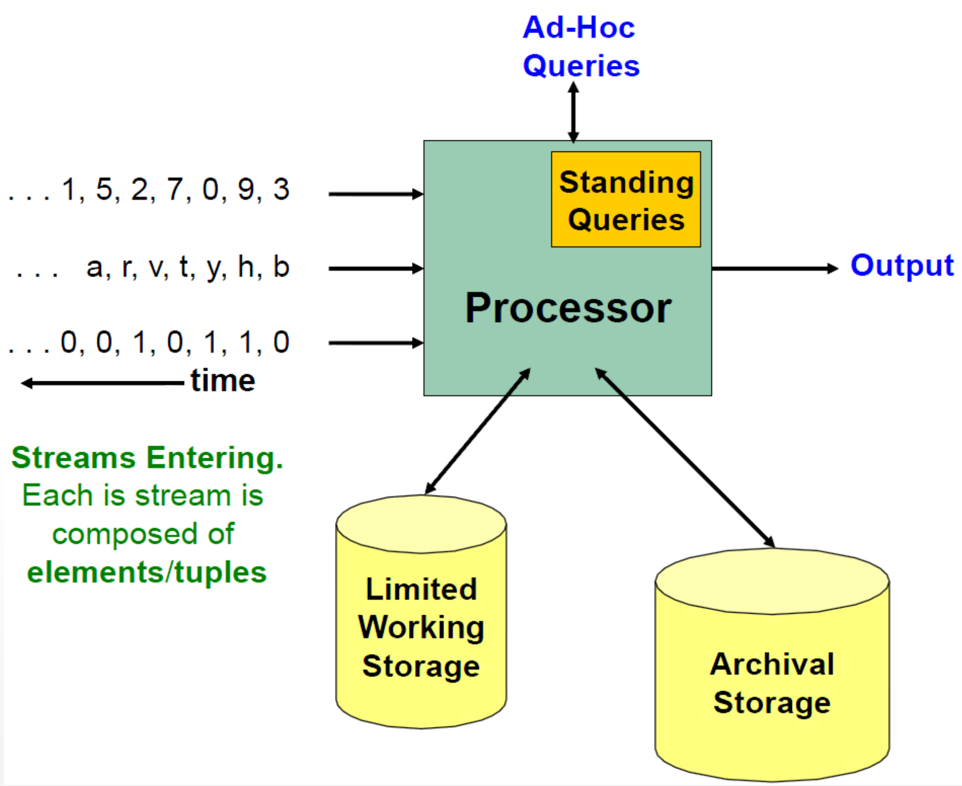
- This week, we will make another assumption that the data arrives in a stream or streams
  - If not processed immediately, it will be lost forever
  - Not feasible to store all the data and then process it

- Stream management is important when the input rate is controlled externally.

- The data can be seen as infinite and non-stationary

The Stream Data Model

- Input **elements** enter at a rapid rate, at one or more input streams
  - **We call elements of the stream tuples**
- **The system cannot store the entire stream**
- **How do you make critical calculations using a limited amount of memory?**



Source: Jure Leskovic, Stanford CS246, Lecture Notes, see http://cs246.stanford.edu

Examples of Stream Sources

- **Sensor Data**
  - Many sensors feeding into a central controller
- **Image Data**
  - Surveillance cameras producing a stream of images
  - London has six million surveillance cameras!!
- **Internet and Web Traffic**
  - Mining Query streams
  - Mining click streams
  - Mining social network news feeds

An Interesting Example of an Application

Source: www.ted.com/talks/deb_roy_the_birth_of_a_word.html

Types of Queries One wants to 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
- **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**

Sampling from a Data Stream

- Since **we can not store the entire stream**, one obvious approach is to store a **sample**

Two different problems:

1. Sample a **fixed proportion** of elements in the stream
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 seen so far has equal prob. of being sampled

Problem 1: Sampling a Fixed Proportion

- **Scenario:** Search engine query stream
  - **Stream of tuples:** (user, query, time)
  - Have space to store **1/10th** of query stream

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

- **Evaluate the solution by answering the question what fraction of queries by an average user are duplicates?**
  - **Assume that each user issues x queries once and d queries twice (total of x+2d queries)**
  - **To evaluate you need to compare the expected value of the solution vs. expected value of the true solution.**

Problem II: Maintaining a Fixed-Size Example

- **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 q d e g…

At **n= 5**, each of the fist 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 Sample Size

- **Algorithm (aka 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