Data Wrangling and Data Analysis

Data Streams

Hakim Qahtan

Department of Information and Computing Sciences

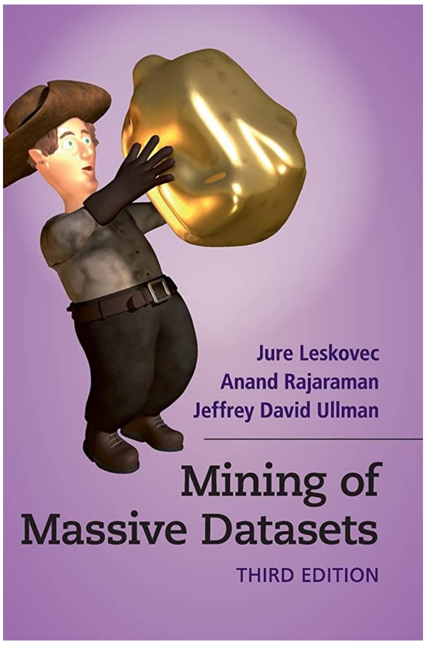
Utrecht University

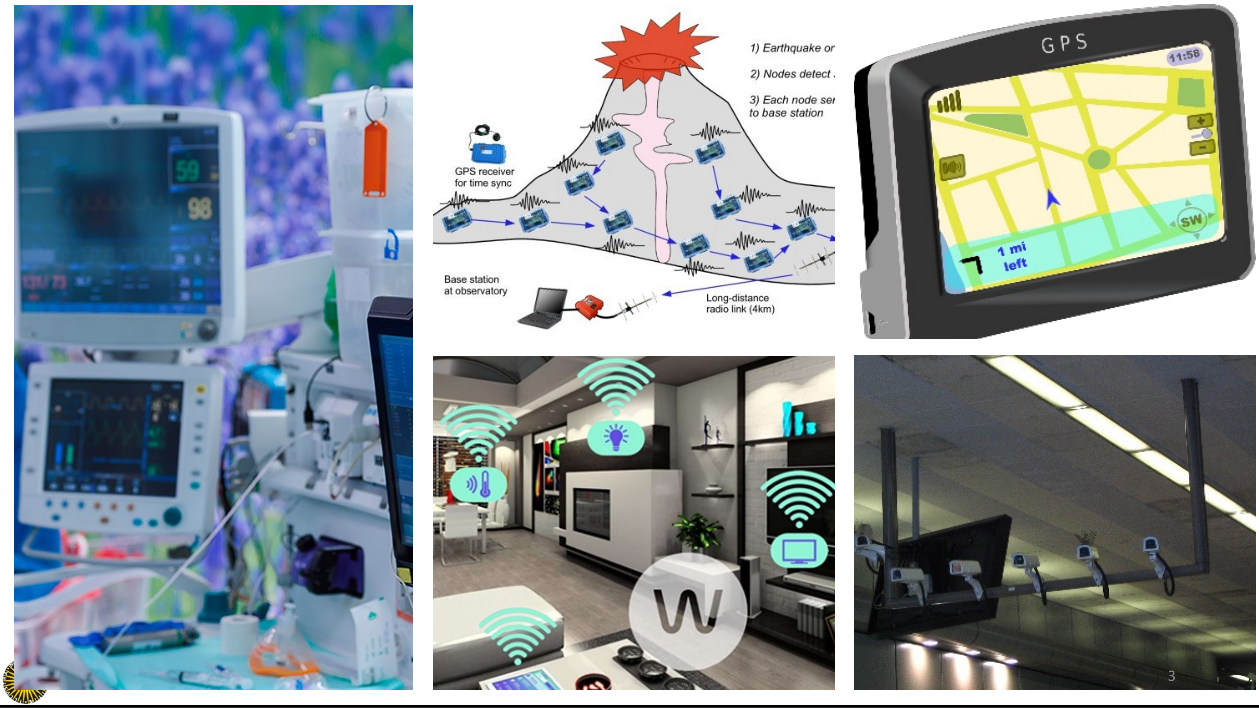
Topics for Today

- Mining Data Streams

Reading Material

- Mining of Massive Datasets
  - Chapter 4





3

3

Large Hadron Collider
generates 40TB data
per second

A Boeing Jet Engine
creates 20TB
information every Hour

YouTube viewers watch
over one billion hours of videos
on its platform every single day

Utrecht University

4

4

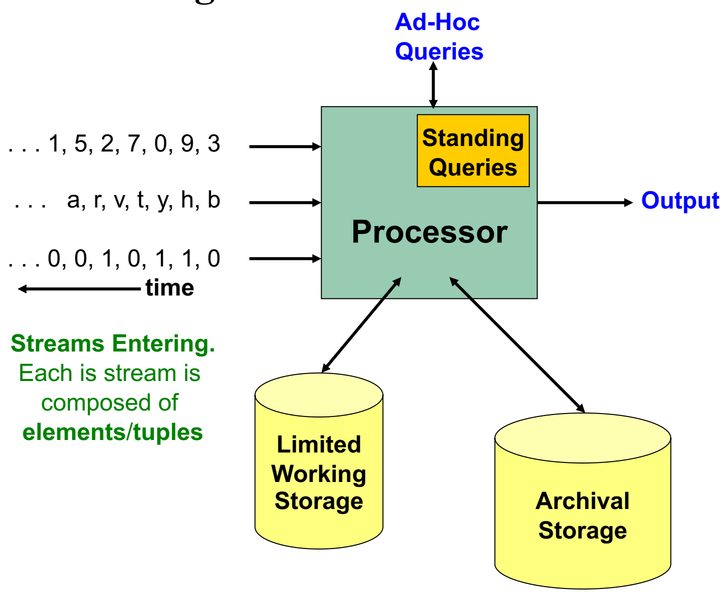
Applications

- Telecommunication calling records
- Business: credit card transaction flows
- Network monitoring and traffic engineering
- Financial market: stock exchange
- Engineering & industrial processes: power supply & manufacturing
- Sensor, monitoring & surveillance: video streams
- Web logs and Web page click streams
- Massive data sets (even saved but random access is too expensive)

Characteristics of Data Streams

- Entire data is not available
- Data arrives (more likely) at a high speed rate
- The system cannot store the entire stream, but only a small fraction
- Huge volume of continuous data (possibly infinite)
  - Requires single scan algorithms (can only have one look)
- Distribution is non-stationary
  - Requires fast, real-time response

General Stream Processing Model



Utrecht University 7

7

How can we perform critical calculations on data streams using a limited size of memory?

Utrecht University 8

8

Handling Data Streams

- Online learning
- Sampling data from data streams
- Windowing functions (models)

Online Learning

- Main idea: perform small changes to update the model
- Training: use the first batch of the data to train a model
- Update: upon the arrival of a new samples from the stream, slightly update the model

$w_1 \leftarrow 0$

FOR $t = 1$ to T DO

$w_{t+1} \leftarrow w_t - \eta_t l(w_t^\top x_t, y_t)$

- Problem: concept drifts

Sampling Data Stream

Sampling from Data Streams

Since we cannot 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 (say 1 in 10)
2. Maintain a random sample of fixed size over a potentially infinite stream
   - At any "time" t, we would like a random sample of s elements
     - What is the property of the sample we want to maintain? For all time steps k, each of the k elements seen so far has equal prob. of being sampled

Sampling from Data Streams – Sample a fixed proportion

Assume we have space to store 1/10-th of the stream

- Naïve solution:
  - Generate a random integer in [0..9] for each query
  - Store the sample if the integer is 0, otherwise discard
- Problem:
  - As the stream grows, the sample size will grow also

Sampling from Data Streams – Sample a fixed Size sample

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

Sampling from Data Streams – Sample a fixed Size sample

- How to think about the problem: say $s = 4$
- Stream: a x c y z k c d e g…
- We need to maintain:
  - When $n = 5$, each of the first 5 tuples is included in the sample S with equal prob.
  - When $n = 7$, each of the first 7 tuples is included in the sample S with equal prob.
- Impractical solution:
  - store the n tuples seen so far
  - pick s at random

Utrecht University

15

15

Sampling from Data Streams – Reservoir Sampling

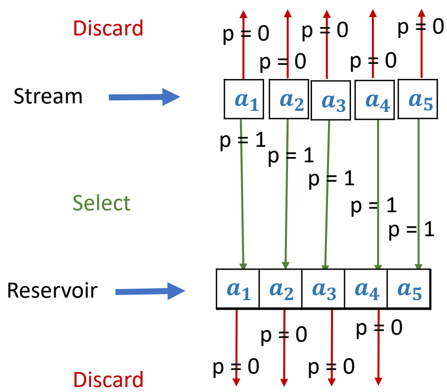
- Store all the first s elements of the stream to S
- Suppose we have seen $n - 1$ elements, and now the n-th element arrives ($n > s$)
- With probability s/n, keep the n-th element, else discard it
- If we picked the n-th element, then it replaces one of the s elements in the sample S, picked uniformly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

Utrecht University

16

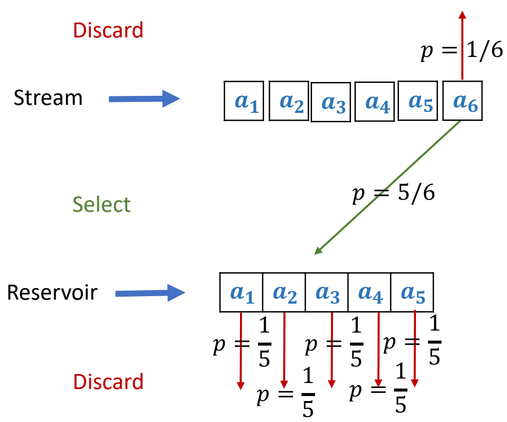
16

Sampling from Data Streams – Reservoir Sampling



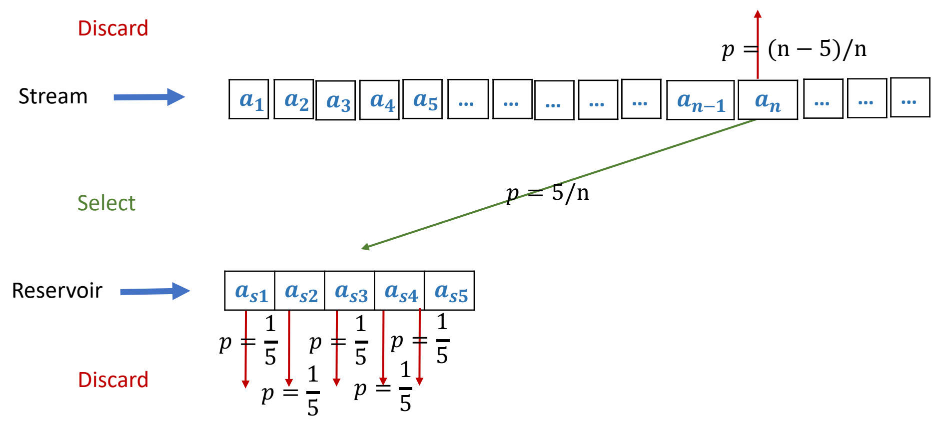
17

Sampling from Data Streams – Reservoir Sampling



18

Sampling from Data Streams – Reservoir Sampling



Utrecht University

19

19

Windowing Models

Utrecht University

20

20

Windowing Models

- A useful model of stream processing is that queries are about a window of length N – the N most recent elements received
- Interesting cases:
  - 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, how many times have we sold X in the last k sales

Sliding Window

← Past Future →

q w e r t y u i o p $\boxed{\text{a s d f g h}}$

q w e r t y u i o p a $\boxed{\text{s d f g h j}}$

q w e r t y u i o p a s $\boxed{\text{d f g h j k}}$

q w e r t y u i o p a s d $\boxed{\text{f g h j k l}}$

N = 6

- Upon the arrival of a new item from the stream
  - Discard the oldest item

Tumbling Window (Disjoint Windows)

← Past Future →

q w e r t y

q w e r t y u i o p a s

q w e r t y u i o p a s d f g h j k

q w e r t y u i o p a s d f g h j k l g m s p v

N = 6

- Upon the arrival of a new batch of items (of size N) from the stream
  - Discard the previous batch

Utrecht University 23

23

Hopping Window

← Past Future →

q w e r t y

q w e r t y u i

q w e r t y u i o p

q w e r t y u i o p a s

N = 6, Step = 2

- Upon the arrival of a new (Step) of items from the stream
  - Keep the last N items only

Utrecht University 24

24

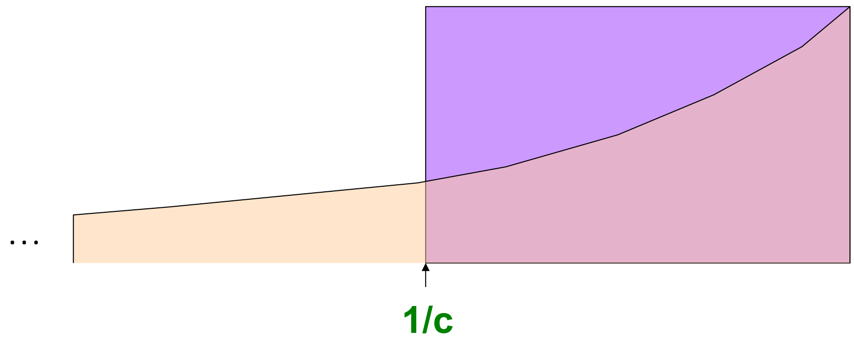
Exponentially Decaying Windows

- Main Idea:
  - Every sample in the stream is important
  - Different levels of importance
  - Recent values are more important
- How it works:
  - Pick a constant $c \in [0,1]$
  - The weight of the element (item) arrived at time t is proportional to $(1-c)^t$
  - $f_t = cf(a_t) + (1-c)\sum_{i=1}^{t} f(a_i)$

Utrecht University 25

25

Sliding vs Exponentially Decaying Windows



- **Important property:**
  - Sum over all weights $\sum_t (1-c)^t$ is $\frac{\mathbf{1}}{\mathbf{1-(1-c)}} = \frac{\mathbf{1}}{\boldsymbol{c}}$

Utrecht University 26

26

Examples of Queries over Data Streams

Querying Data Streams (Examples)

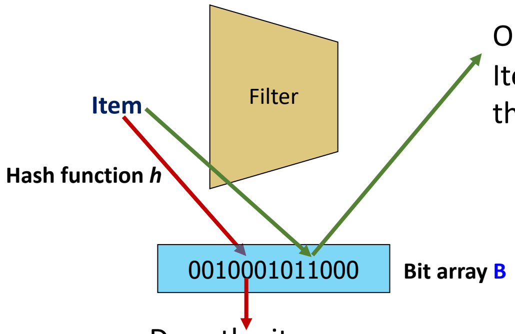
- Filtering a data stream
  - Select elements with property x from the stream
  - Email spam filtering
- Counting distinct elements
  - Number of distinct elements in the last k elements of the stream
  - How many distinct products have we sold in the last week?
- Estimating moments
  - Estimate avg./std. dev. of last k elements
- Finding frequent elements
  - What are “currently” the most popular movies?

Filtering Data Streams

Filtering Data Streams

- Given: a set of Keys *S*
- Determine: which tuples of the stream are in *S*
- Obvious solution: Hash Table
- Problem: we may not have enough memory

Filtering Data Streams



Output the item since it may be in S.
Item hashes to a bucket that at least one of the items in *S* hashed to.

Drop the item.
It hashes to a bucket set to 0 so it is surely not in *S*.

- If the item in *S* (set of keys), return it.
- If the item is not in *S*, it may still be returned (no FNs, but FPs)

Filtering Data Streams (discussion)

- We have:
  - |S| = 1M (We have one million legitimate email addresses)
  - |B| = 1MB (Bit array with 8 million bits)
- Question:
  - What is the probability that an email with un-registered address is going through?

Filtering Data Streams (discussion)

- Approximately 1/8 of the bits will be set to 1
- Given a spam email, it will hash to a bit that includes 1 with *p = 1/8*
- Approximately (1/8 = 0.125) of the spam emails may go through.
  - This is called the false positive ratio (FP ratio)

Filtering Data Streams (discussion)

- More accurate estimation using throwing darts
  - |S| = M, |B| = N
  - Probability of hitting $p_h = \frac{1}{N}$ and missing $p_m = 1 - \frac{1}{N}$
  - After M trials, $p_m = \left(1 - \frac{1}{N}\right)^M = (1-\frac{1}{N})^{N(\frac{M}{N})} = ((1-\frac{1}{N})^N)^{(\frac{M}{N})}$
  - $(1-\frac{1}{N})^N \xrightarrow[N\to\infty]{} \frac{1}{e}$
  - Hence $p_m = e^{-\frac{M}{N}}, p_h = 1 - e^{-\frac{M}{N}}$
  - M = 1M and N = 8M then $p_h = 1 - e^{-0.125} = 0.1175$ (FP ratio)
- How can we reduce the false positive rate?

Bloom Filters

- We have:
  - |S| = M, |B| = N
- Use k hashing functions $H = h_1, h_2, \ldots, h_k$

$B \leftarrow zeros$

FOR $m \in M$ DO

FOR $h_i \in H$ DO

$B[h_i(m)] \leftarrow 1$

Bloom Filters

- Upon receiving an item x from the stream

$exists \leftarrow 1$

FOR $h_i \in H$ DO

if $B[h_i(x)] == 0$ DO

$exists \leftarrow 0$

Return $(exists)$

- Declare x is in S if the items hashes to a bit with 1 for every hashing function in H

Bloom Filters

- Using the previous analysis
- We have kM trials towards the N targets
  - Fractions of 1s is $(1 - \mathrm{e}^{-\frac{kM}{N}})$
  - Hitting k 1s for the k hashing functions with probability $p_h = \left(1 - \mathrm{e}^{-\frac{kM}{N}}\right)^k$
  - is the probability of a FP

Bloom Filters

- When (|S| = M = 1M and |B| = N = 8M):
  - $k = 1{:}\ (1 - \mathrm{e}^{-0.125})^1 = 0.1175$
  - $k = 2{:}\ (1 - \mathrm{e}^{-0.25})^2 = 0.0489$
  - For this example, optimal case when $k = 6{:}\ (1 - \mathrm{e}^{-0.75})^6 = 0.0216$
  - $k = 7{:}\ \left(1 - \mathrm{e}^{-0.7/8}\right)^7 = 0.0229$

Counting Distinct Elements

Counting Distinct Elements

- Problem:
  - Data stream consists of a universe of elements
  - Maintain a count of the number of distinct elements seen so far
- Obvious approach: Maintain the set of elements seen so far
  - That is, keep a hash table of all the distinct elements seen so far

Example Queries

- How many different words are found among the Web pages being crawled at a site?
  - Unusually low or high numbers could indicate artificial pages (spam?)
- How many different Web pages does each customer request in a week?
- How many distinct customers accepted to receive promotional offers during the last month?

Using Small Storage

- Real problem: What if we do not have space to maintain the set of elements seen so far?
- Estimate the count in an unbiased way
- Accept that the count may have a little error, but limit the probability that the error is large

Flajolet-Martin Approach

- Pick a hash function h that maps each of the N elements to at least $\log_2(N)$ bits
- For each stream element a, let $r(a)$ be the number of trailing 0s in $h(a)$
  - $r(a)$ = position of first 1 counting from the right
    - E.g., say $h(a) = 12$, then 12 is 1100 in binary, so $r(a) = 2$
- Record R = the maximum $r(a)$ seen
  - $R = \max_a r(a)$, over all the items a seen so far
- Estimated number of distinct elements = 2^R

Classification

Classifying Data Streams

- Offline classification
  - train a classifier (model) using labelled examples
  - the model is used to predict the label for unlabelled instances
- Best practices
  - split the labelled dataset into train/validate/test
  - maybe use cross-validation to train accurate model
- Online (streaming) classification
  - no clear separation between train/validate/test sets

Classifying Data Streams

- Restrictions
  - process one instance at a time, and inspect it (at most) once
  - limited time to process each instance
  - limited memory
  - be ready to give predictions at any time
  - adapt to changes (concept drifts)

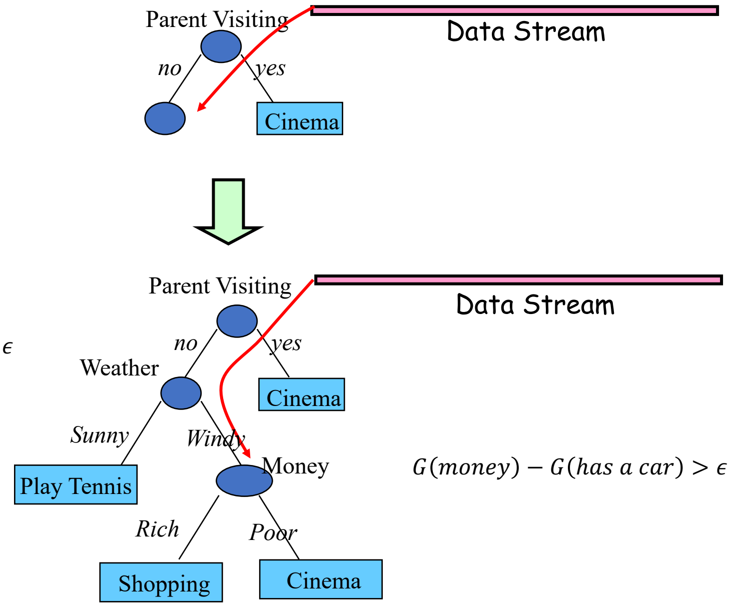
Hoeffding Tree (HT)

- With high probability, HT has similar accuracy as classical DT
- Uses small sample – based on Hoeffding bound
  - X is a random variable
  - R is the domain of X
  - n is the number of observations
  - $\bar{\mu}$ is the sample average (computed using the n observations)
  - With prob. $1-\delta$, the distance from μ to $\bar{\mu}$ is at most ϵ, where:

$$\epsilon = \sqrt{\frac{R^2 \ln\left(\frac{1}{\delta}\right)}{2n}}$$

Hoeffding Tree (HT) Algorithm

```
Input:
      S: sequence of observations
      A: set of attributes {A1, A2, …, Am}
      G(.): Attribute Selection Measure
      δ: desired accuracy
Procedure:
      FOR each observation in S:
            compute G(Ai), 1 ≤ i ≤ m
            retrieve Ap, Aq (with two highest G value)
            if (G(Ap) - G(Aq) > ϵ):
                  split on Ap
                  recurse to next node
                  break
```



$G(weather) - G(has\ a\ car) > \epsilon$

$G(money) - G(has\ a\ car) > \epsilon$

Utrecht University

49

HT Strengths and Weaknesses

- Strengths
  - Scales better than traditional methods
  - Incremental: new examples are added as they come
- Weaknesses
  - Could spend a lot of time with ties
  - Memory used with tree expansion
  - Number of candidate attributes

Utrecht University

50

50

Very Fast Decision Tree (VFDT)

- Modifications to Hoeffding Tree
  - Near-ties broken more aggressively
  - G computed every n_{min} (a user defined parameter)
  - Deactivates certain leaves to save memory
  - Poor attributes dropped
  - Initialize with traditional learner
- Compare to Hoeffding Tree: Better time and memory
- Compare to traditional decision tree
  - Similar accuracy
  - Better runtime with 1.61 million examples
    - 21 minutes for VFDT compared to 24 hours for C4.5

Clustering

Clustering Data Streams

- **Input:** Data stream points from metric space
- **Goal:** Find k clusters in the stream (based on k-median algorithm)
- **Constant factor approximation algorithm**
  - Two step algorithm:
  - Depending on the size of memory, divide the batch of data into l sets $(S_1, \dots, S_l,\ l \gg k)$
  - Select one center c_i from each S_i, $1 \le i \le l$
  - Assign each observation in S_i to its closest center
  - Let $C = \{c_1, \dots, c_l\}$ with each center weighted by number of points assigned to it
  - Cluster C to find k centers (medians)

Utrecht University 53

53

CluStream

- Divide the clustering process into online and offline components
  - Online component: stores summary statistics about the stream data
    - A micro-cluster for n points is defined as a $(2.d + 3)$ tuple $(\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, n)$
  - Offline component: answers various user questions based on the stored summary statistics
- Initialization
  - Use the first batch from the stream to cluster the data into q micro-cluster
  - q is significantly larger than the actual number of clusters

Utrecht University 54

54

CluStream

- Online incremental update of micro-clusters
  - Upon the arrival of a new observation
    - Observation is within max-boundary of one micro-cluster, insert into the micro-cluster
    - Otherwise, create a new cluster
    - May delete obsolete micro-cluster or merge two closest ones
- Query-based macro-clustering (offline)
  - Based on a user-specified time-horizon h and the number of macro-clusters K, compute macro-clusters using the k-medians (or k-means) algorithm

55

Wrap-Up

- We discussed
  - The data streams
  - Models for handling data streams
  - Example queries over data streams (filtering, counting distinct elements)
  - Classification and Clustering of Data streams

56

Extra Material for Interested Students

Utrecht University 57

57

Does each sample have the same probability to be in the reservoir?

Utrecht University 58

58

Sampling from Data Streams – Reservoir Sampling

- Claim: each element is kept in the reservoir with prob. $p = s/n$
- Proof: We prove that using **mathematical induction**
- Base case:
  - After we see $n = s$ elements, the sample S has the desired property
    - Each sample, (out of $n = s$ elements), is in the sample with probability $s/s = 1$
- Inductive hypothesis:
  - After n elements, the sample S contains each element seen so far with prob. $\frac{s}{n}$
- Now element n+1 arrives

Utrecht University 59

Sampling from Data Streams – Reservoir Sampling

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

Utrecht University 60

Flajolet-Martin Approach for Counting Distinct Items in Data Streams

Why Flajolet-Martin Approach Works?

- Intuition: for a given element a
  - h hashes a to any of the M keys with the same probability
  - h will have a sequence of $\lceil log_2 M \rceil$ bits
  - 2^{-r} is the ratio of the keys that have a tail of 0's
    - Approximately 50% of the keys will hash to ********0
    - Approximately 25% of the keys will hash to ******00
    - Example, if a key hashes to *****100, probably 4 keys were hashed (2^2 keys)

Why Flajolet-Martin Approach Works?

- Formally:
  - We use p_r to be the probability of finding a tail of r zeros and $\tilde{p}_r$ is the probability of finding NO tail of r zeros, show:
  - $p_r \xrightarrow[M \gg 2^r]{} 1$ and $p_r \xrightarrow[M \ll 2^r]{} 0$
  - $M = |S|$ is the number of keys – distinct elements from the steam
- Proof:
  - $h(x)$ hashed the elements uniformly at random
  - $p_r(h(x))$ has a tail of r 0's is 2^{-r}
  - $\tilde{p}_r(h(x)) = 1 - 2^{-r}$ (probability of finding **NO** tail of r 0's)

Utrecht University

63

63

Why Flajolet-Martin Approach Works?

- After hashing the M keys,
  - $\tilde{p}_r(h(x)) = (1 - 2^{-r})^M$ (prob. that $h(x)$ has **NO** tail of length r, $\forall x \in S$)
  - $\tilde{p}_r(h(x)) = (1 - 2^{-r})^M = (1 - 2^{-r})^{2^r(M2^{-r})} \xrightarrow[2^r \to \infty]{} e^{-M2^{-r}}$
  - When $M \ll 2^r$ then $\tilde{p}_r(h(x)) \to 1$ and $p_r(h(x)) \to 0$
  - When $M \gg 2^r$ then $\tilde{p}_r(h(x)) \to 0$ and $p_r(h(x)) \to 1$

Utrecht University

64

64