# INTRODUCTION TO DATA SCIENCE

Mining Data Streams (MMDS4)

# Mining Data Streams

- Database mining
  - All data is available

- Stream mining
  - Data arrives as a stream
  - Immediately processed/stored
  - Only feasible to store and interact with a small portion of the data
    - Rapid rate of arrival

# The Stream Data Model

# A data-stream-management system

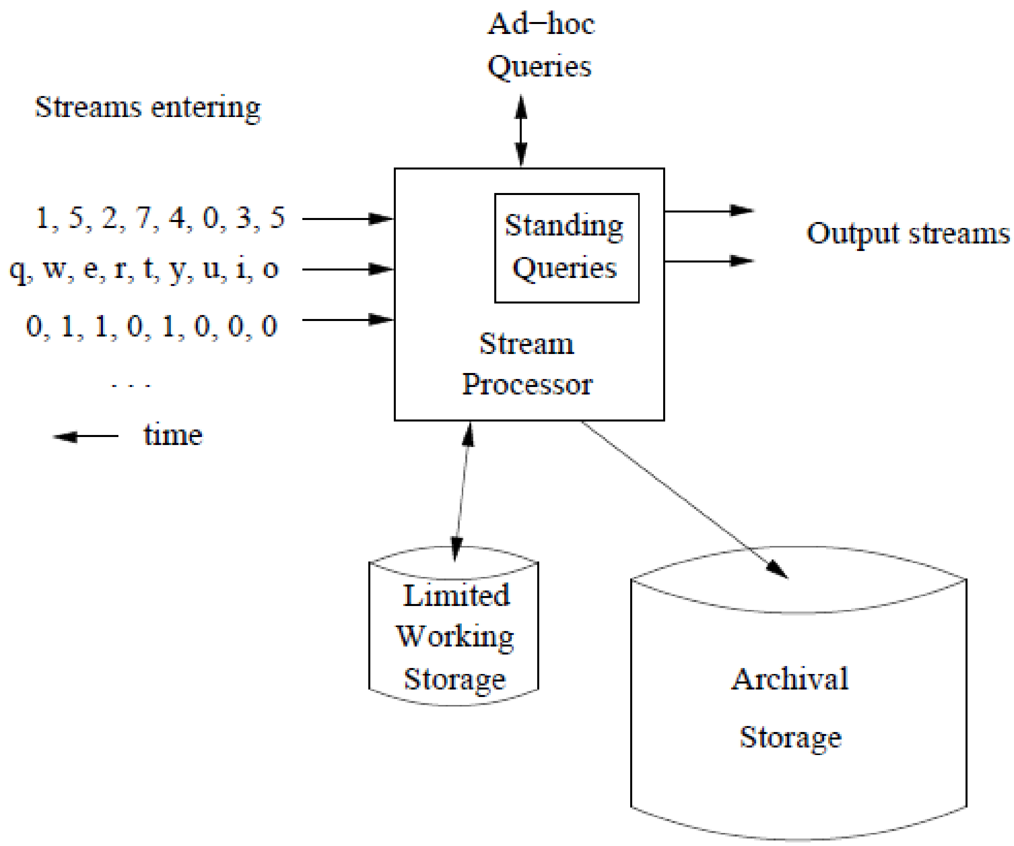



Figure 4.1: A data-stream-management system

# Examples

- Sensor data
  - Temperature sensors equipped with GPS floating in the ocean
    - 10 readings per second, 4 bytes = 3.5Mb per day
    - 1M sensors 3.5T per day
- Image data
  - Satellites producing images each second
- Internet and web traffic
  - Switching node monitors streams of IP packets to report problems or rerouting decisions
  - Google's stream of queries(popular queries) or Yahoo! stream of clicks (popular pages)

# Stream Queries

- Standing queries
  - Permanently executing, producing outputs at appropriate times
  - Alert whenever the temperature exceed a certain value
  - Produce the average of last 24 reading when new data arrives
- Ad-hoc queries
  - Asked once about the current state

# Sliding window

- Store a sliding window in the working store
  - All elements arrived during last t time units
  - Most recent n elements
  - Keep it fresh
- Treat elements of the stream as tuples
  - Treat sliding window as a relation
  - Query with SQL query

# Example: Number of Unique Users

- Relation Logins(name,time)

```
SELECT COUNT(DISTINCT(name))
FROM Logins
WHERE time >= t;
```

- Maintain entire stream of logins for a past month in the working storage

# Issues in Stream Processing

- Rate of arrival might be high
  - Multiple streams
  - Process elements in real-time
  - No disk access
- Approximation
  - Much more efficient to get an approximate answer
  - Hashing to introduce useful randomness

# Sampling Data in a Stream

# A motivating example

- Input: Stream of queries (query,user)
- Problem
  - What fraction of the typical user's queries were repeated over the past month
    - Average user perfumed n queries, k of them were repeated more then once
- Limitation
  - Can store only tenth of stream data

# Example: Obvious approach

- Generate random a 0-9 number for each query
- Store a tuple only if the random number is 0

- Wrong answer
  - Suppose s queries once and d queries twice, and no queries more then twice
    - s/10 queries will appear once
    - d/100 will appear twice
    - $d\left[\frac{1}{10}\frac{9}{10}+\frac{9}{10}\frac{1}{10}\right]=\frac{18}{100}d$ of those who appears twice in the original stream, will appear in the sample exactly once
    - The correct answer is d/(s+d), using the sampling
      - $\frac{d/_{100}}{d/100+{}^{s}/_{10}+{}^{18d}/_{100}}=\frac{d}{19d+10s}$

# Representative sample

- Sample users and store queries of all selected users

- Option I: keep list of all users
  - Check if user exists
  - Generate 0-9 random number for a new user
  - If number is 0 , start to track this user's queries

- Option II: hashing
  - Hash every user id to 0-9
  - Store queries for user that has value 0

# The General Sampling Problem

- Stream tuples have 3 components
  - Key, Value, Time
  - Selection based on keys
  - Sample all tuples with a subset of key values

- Too take a sample of size a/b
  - Hash key value for a tuple into b buckets
  - Store when has value is less than a

# Varying the Sample Size

- Have a budget on total stored tuples
  - Remove samples as more new samples are coming in
- Hash to large number of buckets B
  - Maintain a threshold t
  - Maintain samples such that their keys K have h(K)<t
- Low the threshold of number of samples exceed the allocated space
  - Reduce to t-1
  - Drop all samples hashed to t

# Filtering Streams

# Motivating Example

- Stream: e-mails
- Input: list of 1B legitimate e-mail addresses
  - Others are spam
- Filtering: Allow only legitimate addresses

- E-mail address is 20 bytes
- Not feasible to store the list in RAM

- What can we do with 1G of memory?

# Bloom Filtering

- Use 1Gb RAM as bit array
  - 8B of bits
  - Hash every address to 8B buckets
  - Approximately 1/8 will be ones
    - Some will hash to the same bit

- New e-mail arrives
  - Hash and check if bit is set
  - Let it through if one
  - Some spam will get through
    - Cascade bloom filters

# The Bloom Filter

A *Bloom filter* consists of:

1. An array of $n$ bits, initially all 0's.
2. A collection of hash functions $h_1, h_2, \ldots, h_k$. Each hash function maps "key" values to $n$ buckets, corresponding to the $n$ bits of the bit-array.
3. A set $S$ of $m$ key values.

- Initially: set a bit to one if at least one element of S and at least one hash function set it to one
- To test a key K: let through if all bits are one

$$h_1(K), h_2(K), \ldots, h_k(K)$$

# Analysis of Bloom Filtering

- All keys from S should pass
- What is the probability of false positive

- Throwing darts at targets using a single hash
  - x targets: bits locations
  - y darts: member of S
  - Probability that a given target won't be hit by any darts (i.e. zero bit)

- Probability that given dart will not hit a given targets is $\frac{x-1}{x}$

- Probability that none of y darts will hit a given target is

$$\left(\frac{x-1}{x}\right)^{y} = \left(1-\frac{1}{x}\right)^{x\frac{y}{x}} \rightarrow e^{-\frac{y}{x}}$$

# Example

- 1B e-mails/darts
- 8B targets/bits
- Probability that any given bit will be zero is

$$e^{-\frac{y}{x}} = e^{-\frac{1}{8}}$$

- Probability that given bit will be 1 is

$$1 - e^{-\frac{1}{8}} = 0.1175$$

- Slightly less than 1/8=0.125

# Generalization

- S has m members, array has n bits and there are k hash function
  - Targets x=n
  - Number of darts y=km

- Want proportion of 0 be large
  - So non-S will hash to zero at least once
  - Choose k to be n/m or less
    - Probability of 0 $e^{-\frac{y}{x}} = e^{-\frac{km}{n}} = 0.37 = 37\%$
    - Probability of 1 is $1 - e^{-\frac{km}{n}}$
    - Probability of false positive $\left(1 - e^{-\frac{km}{n}}\right)^k$

# Example

- In the previous example
  - Fraction on 1's is 0.1175
  - Also the probability of false positive

- Use two different hash functions
  - 2B darts on 8B targets
  - Probability of zero is $e^{-\frac{1}{4}}$
  - False positive $\left(1 - e^{-\frac{1}{4}}\right)^2$=0.0493

# Counting Distinct Elements

# The Count-Distinct Problem

- How many different elements in the stream
  - Counting from the beginning or known time in the past
  - Example: number of unique users

- The obvious way
  - Keep list of all elements seen in the RAM
  - Check new element against the list

- Approximation
  - Estimate the number of distinct elements

# The Flajolet-Martin Algorithm

- Hash element to a sufficiently long bit string
  - More possible hash values then elements
  - 64 bits for URLs ($2^{64}$ $values$)

- Whenever number of distinct elements increases
  - Number of different hash-values increases
  - The probability of “special/unusual“ hash-value increases

- Unusual value
  - Ending in many 0’s

# Tail length

- $h(a)$ is hashed to a bit string
- Number of zeros at the end is tail lengths of $h(a)$
- Let R be maximum tail length seen so far
- Estimate number of different elements as $2^R$

# Analysis

- Probability that any given element has tail length at least r is $2^{-r}$

- For $m$ distinct element in the stream, the probability that none of them has at least $r$ is
$$(1-2^{-r})^m = (1-2^{-r})^{2^{-r}m2^r} \approx e^{-m2^{-r}}$$

- Estimate of m
  - For $m \gg 2^r$, $e^{-m2^{-r}}$ is small, at least one has r zeroes
  - For $m \ll 2^r$, $e^{-m2^{-r}}$ is large, no elements have r zeroes

# Combining Estimates

- Assume we have many hash functions
- How to combine their estimates $(2^R)$?
- Averaging does not work well
  - The more functions we have, the more the probability of large number of trailing zeroes just by chance
  - If R grows by 1 , the impact grows by factor 2

- Robust fusion
  - Median: OK with outliers, but only powers of 2
  - Combine into estimations into small groups
    - Average within the group
    - Median of group averages

# Space Requirements

- Keep a single number per stream per hash
  - Largest tail length seen so far

# Estimating Moments

# Definition of Frequency Moments

- Generalization of Count-Distinct problem

- Assume set of possible elements is an ordered set
  - We can define an $ith$ element in this order
  - Let $m_i$ be the number of occurrences of ith element

- The k-th order moment (k-th moment) of the stream is

$$M_k = \sum_i (m_i)^k$$

- k=0 is the number of distinct elements
- K=1 is the length of the stream
- K=2 is the surprise number

# Surprise number

- Measure how uneven the distribution of elements in the stream
  - Assume a stream of length 100 with 11 distinct elements (a1….a11)

- Case I : "Even" distribution
  - a1 appears 10 times, a2..a11 appear 9 times each
  - Surprise number is $10^2 + 10 \times 9^2 = 910$

- Case II: "Uneven" distribution
  - a1 appears 90 times, a2..a11 appear 1 time each
  - Surprise number is $90^2 + 10 \times 1^2 = 8110$

# Alon-Matias-Szegedy Algorithm

- Estimating 2nd moment using limited space
- Start with a finite stream of length n, extend later
- Compute a number of variables $X_1, \dots, X_k$
- For each X store two values X.*element* and X.value
- To compute X
  - Select a random position i between 1 and n
  - Set X.element to be the element at this position
  - Initialize X.value to 1
  - Starting from the next position , scan the stream to the end
  - Add 1 to X.value for each new appearance of X.element

# Example

- Stream is *{a,b,c,b,d,a,c,d,a,b,d,c,a,a,b}*
- *Frequencies a:5, b:4,c:3,d:3*
- *Surprise number* $5^2 + 4^2 + 3^2 + 3^2 = 59$

- *3 variables, 3 random positions 3,8,13*
- $X_1 = \{c, 3\}, X_2 = \{d, 2\}, X_3 = \{a, 2\}$

- *Second moment estimator is* $n \times (2 \times X.value - 1)$
- 55 is the average of
  - 15*(6-1), 15*(4-1), 15*(4-1)
- Compare 55 vs. 59

# Analysis

- Let $e(i)$ be an element at position $i$
- Let $c(i)$ be number of appearances of $e(i)$ starting from position $i$
- The expected value of $n \times (2 \times X.value - 1)$ *for a given* stream is:
  - $E[n \times (2 \times X.value - 1)\ ] = \frac{1}{n}\sum_{i=1}^{n} n(2c(i) -$

# Higher-Order Moments

- Turn a number of occurrences $v$ into estimation of the second moment is $n \times (2v - 1)$
  - Why?
  - Assume a counter goes from $v - 1$ to $v$. What is the change in contribution of an element a.
  - $\Delta(m_a)^2 = v^2 - (v-1)^2 = 2v - 1$
  - They sum up to $m^2$ , their average is $m^2/n$

- For 3rd moment compute $v^3 - (v-1)^3 = 3v^3 - 3v + 1$
  - $n \times (3v^3 - 3v + 1\ ), v = X.value$
- For general k-th order moment
  - $n \times \left(v^k - (v-1)^k\right), v = X.value$

# Dealing with Infinite Streams

- Finite number of variables
  - Keep stream length so far (n)
- Selecting position
  - Once and for all
    - Bias in favor of early positions
  - Wait too long
    - Won't have enough variables for early estimations
- Maintain same number of variables
  - Throw and replace some variables as stream grows

# Replacing Variables

- Assume want to maintain s variables
- When (n+1) element arrives
  - Pick this position with probability s/(n+1)
  - If picked, then with equal probability select a variable
    - Throw it away and replace with the new variable

- Prove that probability of every position to be selected is s/(n+1)
  - It's sure true for the current (n+1) position

# Analysis

- Prove that probability to be selected is s/(n+1) for every position 1…n+1
  - By induction on n

- Assume it's true for n
  - With probability 1-(s/(n+1)) , (n+1)st  position won't be selected
    - Probability of each of the first n positions remains s/n
    - If (n+1)st selected than probability of first n reduced by factor $\frac{s-1}{s}$
    - Overall, the probability of each first n position to be selected is
    - $\left(1-\frac{s}{n+1}\right)\left(\frac{s}{n}\right)+\left(\frac{s}{n+1}\right)\left(\frac{s-1}{s}\right)\left(\frac{s}{n}\right)=\left(1-\frac{s}{n+1}\right)\left(\frac{s}{n}\right)+\left(\frac{s-1}{n+1}\right)\left(\frac{s}{n}\right)=\left(\frac{s}{n}\right)\left[1-\frac{s}{n+1}+\frac{s-1}{n+1}\right]=\frac{s}{n+1}$

# A general Stream-Sampling Problem

- Maintain a sample of s elements such that every element of the stream is equally likely to be selected

- Solution
  - Select new sample with probability $s/(n+1)$
  - If selected, replace any old sample at random

# Counting Ones in a Window

# Exact Solution

- Assume binary infinite stream
- At any point at time to be able to answer following query
  - How many 1's in the last M bits of the stream (for all $\mathrm{M} \leq N$)

- Claim: Exact solution requires at least N bits for stream of length N

- Claim: Event answering queries only for a fixed M requires storing entire window (M bits)
  - Non-trivial: why not to count (logM bits)?

# Proof: Any M

- Assume representation less then N bits
  - $2^N$possible streams
  - Have to be at least 2 streams with the same representation $x, w$.
  - Let k to be the position they are differ (from the right)
    - w= $0101, x = 1010, k = 1$
    - w= $1001, x = 0101, k = 3$
  - Query: “ how many bits in window of length M=k’
    - Same representation->same answer
    - Different number of bits

# Proof: Fixed M

- Find a pair (w,x) of length M with the same representation
- Let k to be position (from the right) they are differ
- Create new pair
  - Take k bits from w and x and add M-k identical bits
  - Any algorithm will produce the same answer

- Exact solution require storing entire stream

# The Datar-Gions-Indyk-Motwani(DGIM) Algorithm

- Approximation
  - For a fixed window of length N

- Simple version
  - Space $O(log^2 N)$
  - Error: No more than 50%

- Improvements
  - Error: Any $\varepsilon > 0$
  - Space $O(log^2 N)$ with factor that depends on $\varepsilon$

# DGIM:Timestamps

- A timestamp of a bit- the position it arrives
- First bit has timestamp 1
- Only need to distinguish positions within the window
  - Represent timestamps modulo N ($log_2 N\ bits$)
  - Also store total number of elements ever seen modulo N
    - Use timestamp of element in the window modulo N and length of stream modulo N to find the position of this element within the stream
    - (10,11,12,13,14,15) mod 6 is (4,5,0,1,2,3), last modulo N is 3

# Buckets

- Divide the window into buckets
  - Different from hash buckets
- For each bucket stores
  - The timestamp of its rightmost bit (most recent)
  - The number of 1's in the bucket (size of the bucket)
    - Select the bucket so it size will be a power of two
    - Example: total 7 bits, divide into buckets of size 1,2,4
- Total bits per bucket
  - Timestamp $log_2 N \ bits$
  - Size of the bucket $log_2 log_2 N$
    - Bucket size $i = 2^j$, need only represent $j$
    - Maximum value of $i \ is \ N$, maximum value of $j \ is \ log_2 N$
    - Total $log_2 log_2 N$ bits for representing every value of $j$

# Buckets rules

- The right end of a bucket is always a position with a 1.
- Every position with a 1 is in some bucket.
- No position is in more than one bucket.
- There are one or two buckets of any given size, up to some maximum size.
- All sizes must be a power of 2.
- Buckets cannot decrease in size as we move to the left (back in time).

# Example

. . 1 0 1 1 0 1 1 0 0 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0

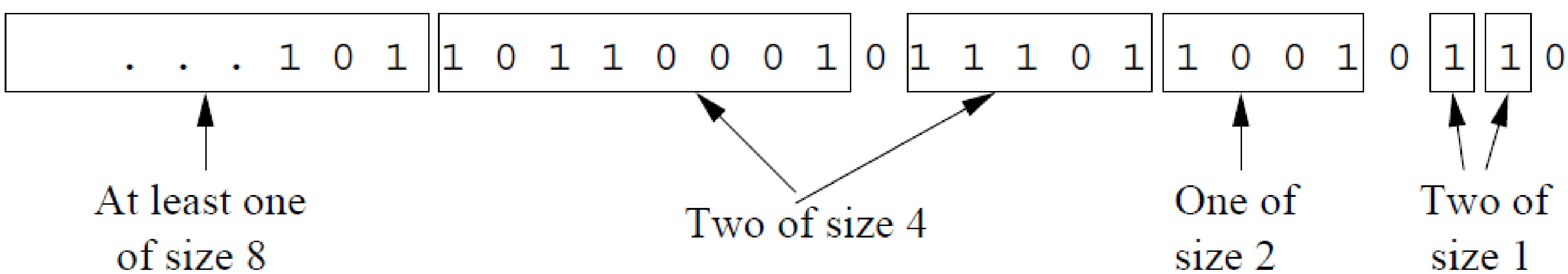


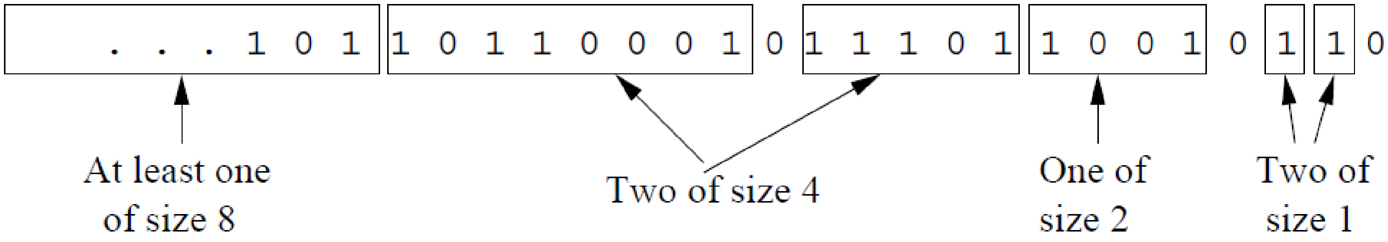

Figure 4.2: A bit-stream divided into buckets following the DGIM rules

# Storage Requirements for the DGIM

- Each bucket requires $O(log_2N + log_2log_2N)$
- Maximum possible number of buckets in the window
  - Maximum number of 1's is N
  - Largest bucket size is $2^j, j \leq log_2N$
  - At most 2 buckets of each size
    - Total, no more than $2log_2N$ buckets
- Total space requirements is
  - $O(log_2N \times 2log_2N) = O(log^2N)$

# Answering queries

- Answer queries about number of 1's in the last k elements
  - Find the leftmost bucket $b$ with at least some bits from k-window
  - Sum sizes of all buckets to the right of b and half the size of bucket $b$
- Example: k=10: 0110010110, 5 bits
  - (4/2)+2+1+1=6



# The error

- Assume correct answer c
- Assume the size of bucket $b$ is $2^j$

- Case I: The estimate is less than c
  - Worst case: all of b are within the k: estimate misses half of bucket b: $2^{j-1}$
  - Rule: At least one bucket of each size: c is at least $\sum_{i=0}^{j-1} 2^i = 2^j$
  - Error: 50%

- Case II: The estimate is greater than c
  - Worst case: only a single bit from b within the k: estimates overestimate by half
  - Rule: At least one bucket of each size: c is at least $\sum_{i=0}^{j-1} 2^i = 2^j$
  - Error: 50%

- General idea: allow larger errors for larger k by allowing buckets of greater size

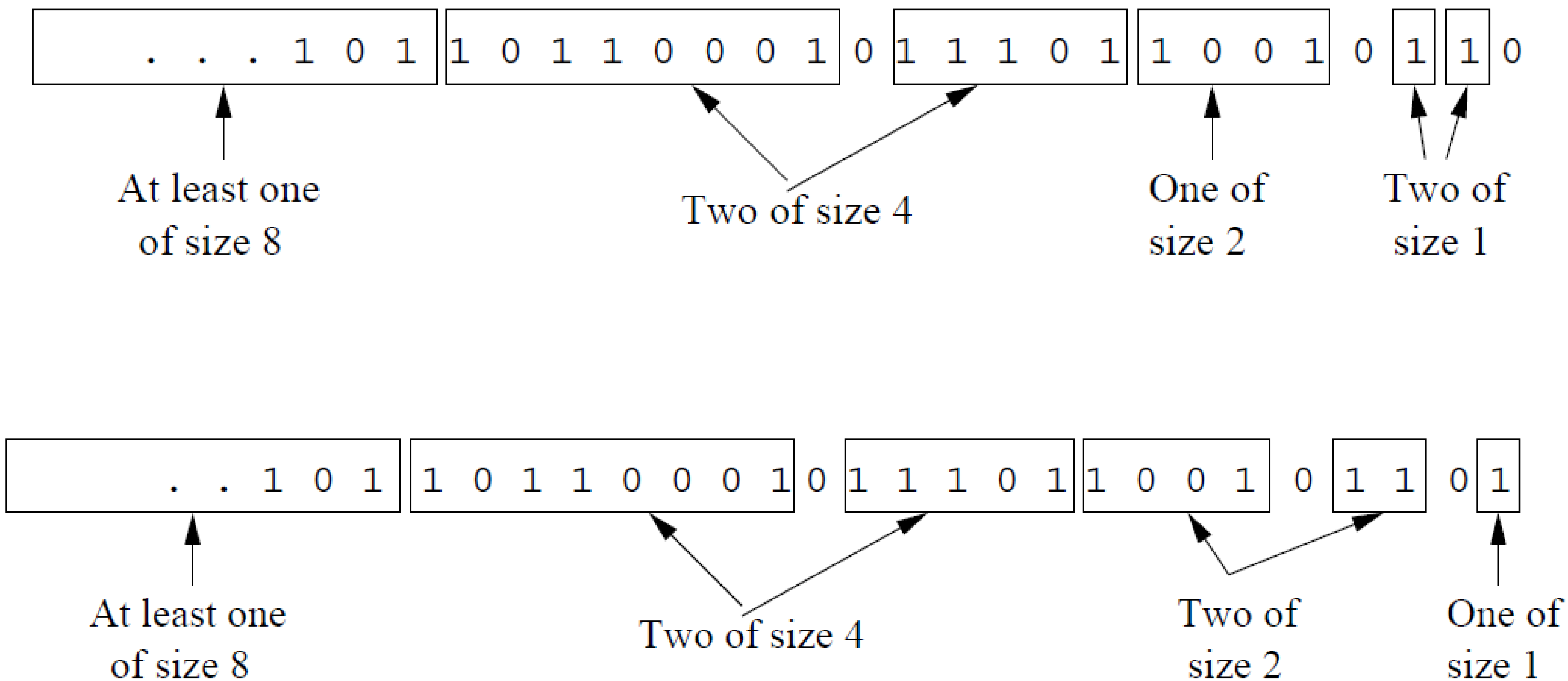
# Maintaining the DGIM Conditions

- Assume proper buckets in a window of size N

- How to modify with a new bit coming in
  - Drop leftmost bucket if its timestamp (rightmost) is out of the range
  - If new bit is 0 do nothing, If new bit is one , than
    - Create a new bucket with the current timestamp and size 1
    - If only a single bucket of size 1, done
    - If not, combine buckets to obtain bucket of size 2
    - Continue to combine If more then 2 buckets of the same size obtained

- Complexity
  - At most $2log_2N$ buckets to combine, total O$(logN)$

# Example: 1 enters



# Reducing the Error

- Allow r or (r-1) buckets of any size
  - r=2 for previous algorithm

- Exception
  - Allow any number from (1 to r) of buckets of size 1 or of the largest size

- If r+1 buckets of size $2^j$ *are crated*
  - Combine 2 leftmost buckets into a new one of size $2^{j+1}$
  - Continue combining while required
  - Should be at least r-1 buckets of size 1 for any other size to exist

# Error

- Assume only1 bit of b(size $2^j$) is in the range
  - Error is $2^{j-1} - 1$
- True count is at least
  - $1 + (r - 1)\left(2^{j-1} + 2^{j-2} + .. + 1\right)$
- Relative error no more than
  - $\frac{2^{j-1}-1}{1+(r-1)\left(2^{j-1}+2^{j-2}+..+1\right)} \le \frac{2^{j-1}}{(r-1)2^j} \le \frac{1}{2(r-1)} \le \frac{1}{r-1}$
- Pick $r$ for any bound $\varepsilon$

# Extensions

- Sum of last k numbers in stream of integers

- Unlikely for positive and negative stream
  - The leftmost bucket can have very large positive and negative numbers that sum up to zero

- Possible for positive only stream
  - Assume range 1 to $2^m$
  - Treat each of m bit as a separate stream
  - Estimate sum as $\sum_{i=0}^{m-1} c_i 2^i$

# Decaying Windows

# Example: Most-Common Elements

- Stream
  - Tickets purchased with the name of a movie
- Query
  - Most popular movies ‘currently’
- How to interpret ‘currently’?
  - Popular movies of past year/month/day
  - Popular movies of past hour

# Using bit streams

- Represent each movie by a bit stream
  - i-th element 1 is i-th ticket is for that movie
  - Estimate number of 1s in each stream in window of size k
  - Rank movies according to estimate

- Won't work if millions of items/streams
  - Most popular amazon products
  - Most popular pages/tweets etc.

# Decaying Window

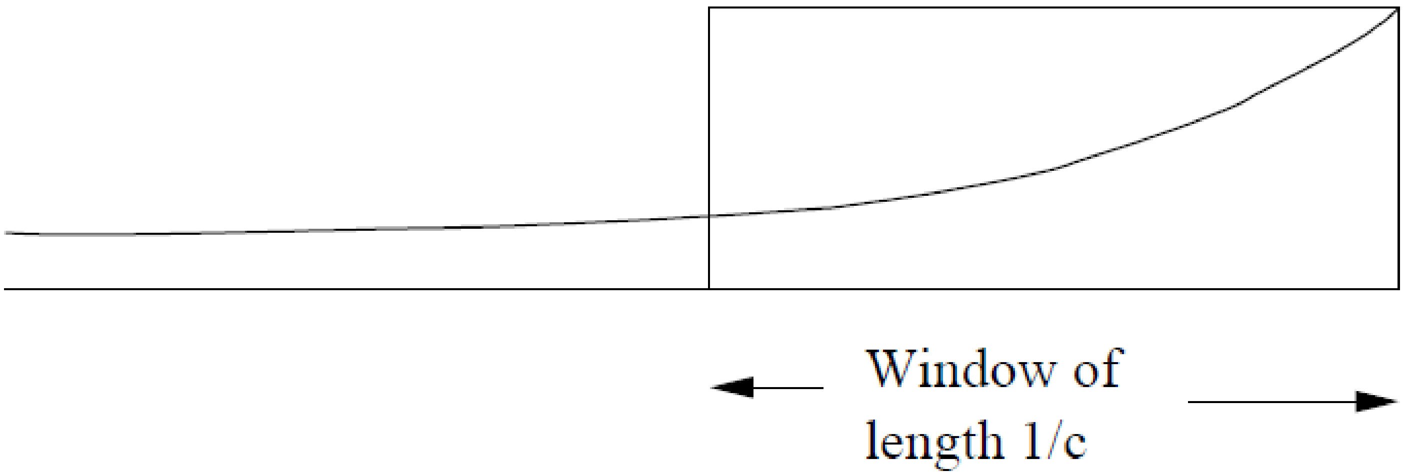
- Compute weighted sum of 1's ever seen
  - Exponentially decaying weights
  - Further back in the stream, less weight is given

- Exponentially decaying window is the sum

$$\sum_{i=0}^{t-1} a_{t-i}(1-c)^i$$

  - c is a small constant $(e.g. 10^{-6}\ or\ 10^{-9})$
  - Note $\sum_{i=0}^{\infty}(1-c)^i = \frac{1}{c}$
- Fixed window pit equal weights to all element in the window and zero weight to all others
  - Fixed window of same total weight will put 1 for most recent $^1/_c$ elements

# Fixed vs. Decaying Window Weights



# Computing Decaying Window

- Much easier vs. fixed window
  - No need to store old elements

- Simple Algorithm

1. Multiply the current sum by $1 - c$.
2. Add $a_{t+1}$.

# Find the Most Popular Elements

- $c = 10^{-9}$, approximate a 1B sliding window
- Large number of movies, maintain score only for popular movies
- For each new ticket

1. For each movie whose score we are currently maintaining, multiply its score by $(1-c)$.
2. Suppose the new ticket is for movie $M$. If there is currently a score for $M$, add 1 to that score. If there is no score for $M$, create one and initialize it to 1.
3. If any score is below the threshold $1/2$, drop that score.

# Analysis

- Number of scores is limited at any time
  - Sum of all scores is 1/c
    - Each new ticket adds only single 1 to total scores
  - No more than 2/c movies with score ½ or more
    - Else sum of scores will exceed 1/c
- Number of active scores is limited by 2/c
  - In practice, small number of blockbusters

# Summary

# Summary

- Stream Data Model
  - limited active storage for maintaining summaries
- Sampling of streams
  - Sampling by key attributes
- Bloom filters
  - Cascade bit array lookups
- Counting Distinct Elements
  - Count special hash values (trailing zeroes)
- Moments of streams
  - Frequency moments
- Second Moment
  - Count appearances from random location
  - Higher moments by different formula
- Estimating the Number of 1s
  - Longer buckets farther in time
  - Reduce error using r,r+1 rule
- Exponentially decaying windows
  - Easy to compute