Lecture 6. Evolving data streams

Ricard Gavaldà

MIRI Seminar on Data Streams, Spring 2015
Contents

1. Time in data streams: Philosophical Intro
2. Sliding window model. Exponential histograms
3. Tracking statistics
4. Change detection
Time in data streams: Philosophical Intro
Model up to now:

- All items seen since time 0 taken into account
- Order of the items in the stream is adversarial
  
  = Algorithms must good answers even for worst-case order

But:

- Most streams in real life change over time
- Most recent elements are more relevant
“Most streams in real life change over time”

Q. What does this really mean?

**Statistical answer.** Occurrence of items now does not follow the same statistical laws as e.g. yesterday or last year. In particular, there are statistical laws!

**Metaphysical answer:** Streams probabilistically reflects the current state of the world, which evolves over time
“Most recent elements are more relevant”

Q. Really? Why?

My-boss-tells-me-so answer: I need to count packets sent during last month, for billing each customer

Machine learner answer: Because they help me better to predict the immediate future

This relies on the statistical interpretation!

Why should prediction be possible at all?

For discussion: and on the metaphysical interpretation?
A statistical (& metaphysical) setting

Rarely formalized, but implicit in many works

- for each time $t$, distribution $D_t$ on universe of items
- $t$-th stream item independently generated according to $D_t$
- each $D_t$ possibly different from previous ones
- but either change occurs infrequently ("shift")
- or else they are very small ("drift")
- so that statistics has some time to converge
- and adversarial cannot be totally adversarial
Ignore at your own risk: Autocorrelation, bursts, varying rates of arrival.

Example:

\[
\Pr[\text{rain tomorrow} | \text{rain today}] > \Pr[\text{rain tomorrow}]
\]

But even if \( \Pr[\text{rain today}] = 1/5 \),

\[
\Pr[\text{4 years with zero rain, followed by 1 year raining every day}] \approx 0
\]
Sliding window model. Exponential histograms
The Sliding Window Model

- Only last \( n \) items matter
- Clear way to bound memory
- Natural in applications: emphasizes most recent data
- Data that is too old does not affect our decisions

Examples:
- Study network packets in the last day
- Detect top-10 queries in search engine in last month
- Analyze phone calls in last hours
Statistics on Sliding Windows

- Want to maintain mean, variance, histograms, frequency moments, hash tables, ...
- SQL on streams. Extension of relational algebra
- Want quick answers to queries at all times
Basic Problem: Counting 1’s

Obvious algorithm, memory $n$:

- Keep window explicitly
- At each time $t$, add new bit $b$ to head, remove oldest bit $b'$ to tail,
- Add $b$ and subtract $b'$ from count

Fact:
$\Omega(n)$ memory bits are necessary to solve this problem exactly
Counting 1’s

[Datar, Gionis, Indyk, Motwani, 2002]

**Theorem:**

There is a *deterministic* algorithm that estimates the number of 1’s in a window of length \( n \) with multiplicative error \( \varepsilon \) using \( O\left(\frac{1}{\varepsilon \log n}\right) \) counters up to \( n \)

which means \( O\left(\frac{1}{\varepsilon} (\log n)^2\right) \) bits of memory

**Example:**

\( n = 10^6; \ \varepsilon = 0.1 \rightarrow 200 \) counters, \( 4000 \) bits
Idea: Exponential Histograms

Each bit has a timestamp - time at which it arrived
At time \( t \), bits with timestamp \( \leq t - n \) are *expired*
We have up to \( k \) buckets of size 1, 2, 4, 8 \ldots
We keep the counts of 1s in each bucket
Errors: expired bits in the last bucket
Last bucket \( \leq \) all buckets / \( k \approx n/k \)
Exponential Histograms

Questions:

- How do we maintain the buckets when new items arrive
- How do we obtain error (# 1’s)/k, rather than n/k
Exponential Histograms

Instead of keeping buckets with $2^i$ bits, with number of ones,

$$\ldots101110010111000101011010100100101010101\ldots$$

- now-30
- now

Store buckets with $2^i$ ones, with timestamps,

$$\ldots101110010111000100111000001001100100100101010101\ldots$$

- now-30
- now
Exponential Histograms

Init: Create empty set of buckets

Query: Return total number of bits in buckets - last bucket / 2
Exponential Histograms

Insert rule(bit):

- If bit is a 0, ignore it. Otherwise, if it’s a 1:
  - Add a bucket with 1 bit and current timestamp \( t \) to the front
  - for \( i = 0, 1, \ldots \)

  If more than \( k \) buckets of capacity \( 2^i \),
  - merge two oldest as newest bucket of capacity \( 2^{i+1} \),
    with timestamp of the older one
  - if oldest bucket timestamp < \( t - n \), drop it (all expired)
Why Does This Work?

Let \(2^C\) be capacity of largest (oldest) bucket:

- **Claim 1:** All buckets except oldest one are totally full with non-expired bits
- **Claim 2:** Oldest bucket contains between \(\geq 1\) and \(\leq 2^C\) non-expired bits
- **Claim 3:** For each capacity except that of the largest bucket there are \(k\) or \(k - 1\) buckets
- **Claim 4:** Sum of all buckets except those of largest capacity is in \[((k - 1)(2^C - 1), k(2^C - 1))\]
- **Claim 5:** Estimate is within \([1 - \frac{1}{2k}, 1 + \frac{1}{2k}]\) of correct number of 1s

So take \(k = \frac{1}{2\varepsilon}\), we’re done
Memory Estimate

- Largest bucket needed: \( k \sum_{i=0}^{C} 2^i \approx n \rightarrow C \approx \log(n/k) \)
- Total number of buckets: \( k \cdot (C + 1) \approx k \log(n/k) \)
- Each bucket contains a timestamp only (perhaps its capacity, dep. on implementation)
- Timestamps are in \( t - n \ldots t \): recycle timestamps mod \( n \)
- Memory is \( O(k \log(n/k) \log n) \) bits, \( k = 1/\varepsilon \)
Technique can be applied to maintain many natural aggregates, those satisfying well-defined conditions:

- Distinct elements
- Max, min
- Histograms
- Hash tables
- Frequency moments
A simpler version:
- Queue of $n/k$ buckets
- each of equal capacity $k$
- Memory $O(n/k \cdot \log k)$ bits

Exercise 1
Analyze this simpler version:
- $k$ to obtain error $\varepsilon$?
- Memory to obtain error $\varepsilon$? (relative to $n$? relative to number of 1’s?)
- When is it worth using the exponential histograms?
Tracking statistics
Many mining / learning algorithms compute statistics on data and combine them
Many mining / learning algorithms compute statistics on data and combine them.
Fix a statistic $\phi$ on distributions

Want to compute $\phi$ on a stream $S = \{x_t\}_t$, making recent samples more important

General method:

- Choose a decay function $f : \mathbb{N} \rightarrow [0, 1]$, of sum 1
- At time $t$, assign importance $f(i)$ to $x_{t-i}$
- Compute $y_t = \phi(\{f(i) \cdot x_{t-i}\}_i)$
Special cases, for $x_t \in \mathbb{R}$, $\phi = \text{avg}$:

- **Memory-based**: Sliding window of size $W$
  
  - $y_t = \text{avg}(x_t, \ldots, x_{t-W+1})$
  
  - $f(i) = \begin{cases} 1/W & \text{for } i < W \\ 0 & \text{for } i \geq W \end{cases}$

- **Memoryless**: EWMA: Exponentially Weighted Moving Average
  
  - $y_1 = x_1$
  
  - $y_t = \lambda \cdot x_t + (1 - \lambda) \cdot y_{t-1}$, for fixed $\lambda < 1$
  
  - Inductively, $y_t = \sum_i \lambda (1 - \lambda)^i x_{t-i}$
  
  - $f(i) = \lambda (1 - \lambda)^i$
Change detection
Stream of numbers, want to keep mean of most recent ones

My-boss-told-me-so setting:

- Sliding window: \( y_t = \text{avg}(x_t, \ldots, x_{t-W+1}) \)
- EWMA: \( y_t = \sum_i \lambda(1-\lambda)^i x_{t-i} \)

Statistical setting:

- Each \( x_t \) is the realization of a random variable \( X_t \sim D_t \)
- We want to have an estimation of \( E[X_t] \)
- We can average \( x_t, x_{t-1}, x_{t-2} \ldots \) if \( D_t \sim D_{t-1}, D_{t-2}, \ldots \)
The variance / reaction time tradeoff

Average \( \frac{(x_t + x_{t-1} + \cdots + x_{t-W+1})}{W} \)

Two sources of inaccuracy:

- Variance of each \( x_{t-i} \)
  - Decreases as we average more values

- Drift as we consider older values
  - Increases as we average over more values

Where is the sweet spot?

- Depends on variance and change rate, unknown
A simple, memoryless change detector is CUSUM, CUmulative SUM test:

- $g_0 = 0$
- $g_t = \max(0, g_{t-1} - \nu + \epsilon_t)$
- If $g_t > h$ declare change, set $g_t = 0$, reset $\epsilon_t$

Idea: $\nu, h, \epsilon$ control sensitivity
- $\nu \approx \sigma$, “normal” deviation
- $h = k\sigma$, “ alarming” deviation
- $\epsilon_t \to 0$

Similar to: Page-Hinkley test
Window-based change detection

“If $d(W_1, W_2) > \varepsilon$ then declare change”

[Kifer - Ben-David - Gehrke 04]:

$W_1$ reference, $W_2$ sliding, fixed size

Window size choice is an issue
Window-based change detection

“If $d(W_1, W_2) > \varepsilon$ then declare change”

[Gama et al, 04]: Specific for classifier error monitoring
$W_1$ achieves min error, $W_2$ all data, varying size
Slow to react after long stretches with no change
ADaptive Windowing: Look at the data to find the optimal tradeoff variance - current rate of change

No need to guess parameters a priori, or assume that “optimal parameters” are optimal forever

Idea: Keep $W$ as long as possible while consistent with

“there is no clear evidence of change within $W$”
Fix a distance among probability distributions $d$ and a statistical test $test$ such that for two samples $S_1$, $S_2$ from distributions $D_1$, $D_2$,

\[
\Pr[\text{test}(S_1, S_2, \delta) = \text{true}] = \begin{cases} 
\leq \delta & \text{if } D_1 = D_2 \\
> 1 - \delta & \text{if } d(D_1, D_2) \geq \varepsilon \\
? & \text{otherwise}
\end{cases}
\]

(Actually, $\varepsilon = \varepsilon(\lvert S_1 \rvert, \lvert S_2 \rvert, \delta))$

Try $\text{test}(W_1, W_2, \delta)$ for every partition of $W = W_1 + W_2$

If the test for $W_1$, $W_2$ returns true, shrink $W$ to $W_2$
ADWIN: The Guarantees

Theorem

At every time $t$,

- (No false positives) If there is no change in average within $W$, ADWIN does not cut $W$, with prob $\geq 1 - \delta$

- (No false negatives) If there is a split $W = W_1 + W_2$ such that $d(D_1, D_2) \geq \varepsilon(\ldots)$ then ADWIN shrinks $W$ with probability $\geq 1 - \delta$

For general sets of items, Kolmogorov-Smirnov test
For windows of bounded real numbers, a simple test using $d(D_1, D_2) = |\text{avg}(D_1) - \text{avg}(D_2)|$ can be based on Hoeffding

In practice, use heuristics variants
Built on top of exponential histograms:

Initialization:
Empty all buckets (=empty window)

At each time $t$,
for all partitions of $W$ into $W_1 + W_2$,
where $W_2$ comprises exactly a number of buckets do
if $\text{test}(W_1, W_2, \delta) \geq \varepsilon(|W_1|, |W_2|, \delta)$ then
declare change; drop oldest bucket
end if
end for
ADWIN: The Guarantees

- ADWIN uses memory $O(\log W)$ for a window of length $W$
- Time per item is $O(\log W)$; can be made amortized $O(1)$

- Can be used both as estimator and as change detector
- It cuts $W$ if and only if change has occurred
- Keeps longest window statistically consistent with the hypothesis “no change within window”
- Autonomously solves the variance - reaction time tradeoff
- Parameter-free (only $\delta$), scale-free (no guessing)
ADWIN: The Use

Pluses:
- Black box, guesswork-free module to deal with change
- Change detector + “intelligent counter”
- $O(\log W)$ memory & time
- Parameter-free, scale-free algorithm

Cons:
- Memory is not $O(1)$ like EWMA+CUSUM
- Much slower than EWMA+CUSUM (100-500 times) in direct implementation
- Optimizations should be possible
- Still, OK when in the middle of other computation or disk access