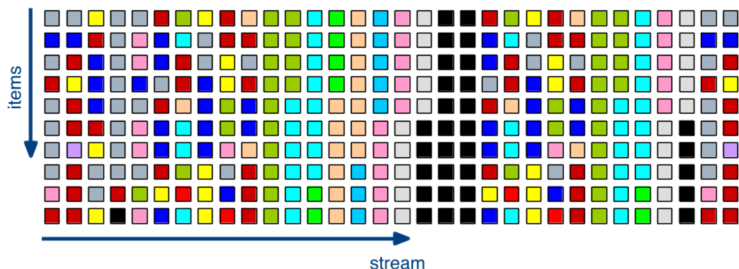
Data Stream Processing

Part II

Data Streams (recap)



- continuous, unbounded sequence of items
- unpredictable arrival times
- too large to store locally
- one pass real time processing required

Reservoir Sampling (recap)

- create representative sample of incoming data items N
- uniformly sample into reservoir of size r

Today

Counting algorithms based on stream windows

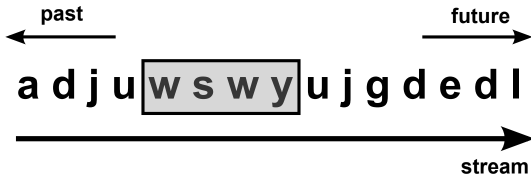
Lossy Counting

Sticky Sampling

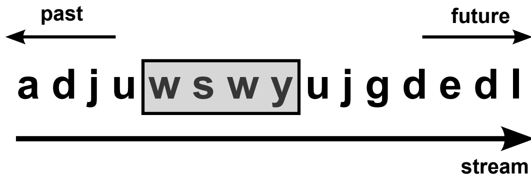
Stream Windows

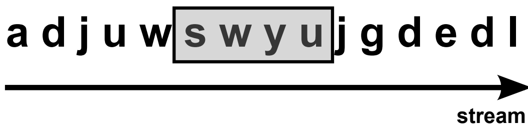
Mechanism for extracting a finite relation from an infinite stream.

Window Example

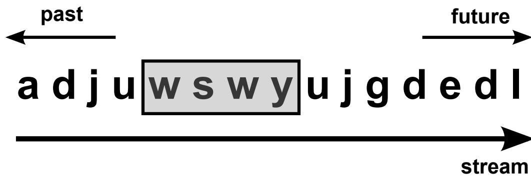


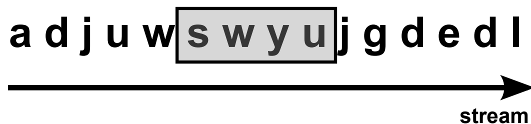
Window Example

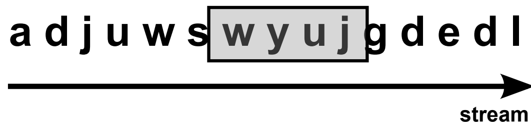




Window Example







Window Example

past

future

a d j u w s w y u j g d e d l

stream

a d j u w s w y u j g d e d l

stream

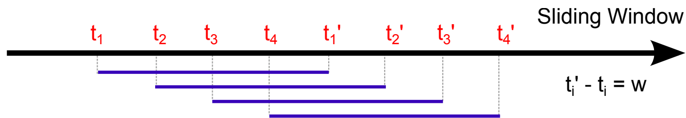
a d j u w s w y u j g d e d l

stream

Sliding Window

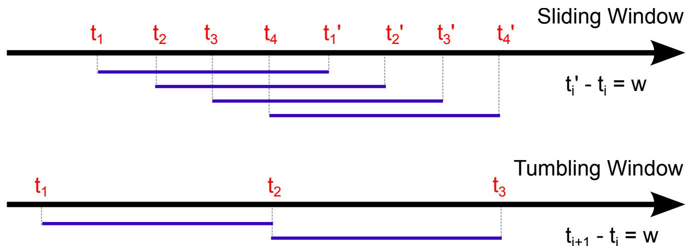
Window Types

- assumes existences of some attribute that defines the order of the stream elements (e.g. time)
- w is the window length (size) expressed in units of the ordering attribute (e.g. seconds)



Window Types

- assumes existences of some attribute that defines the order of the stream elements (e.g. time)
- w is the window length (size) expressed in units of the ordering attribute (e.g. seconds)



Count based Windows

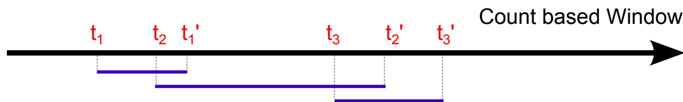
Ordering attribute can cause problems for duplicates
(e.g. same time stamps)

Use count based windows instead

Count based Windows

Ordering attribute can cause problems for duplicates (e.g. same time stamps)

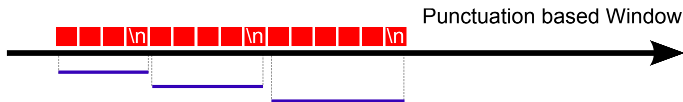
Use count based windows instead



Count based windows are potentially unpredicatable with respect to fluctuation in input rates.

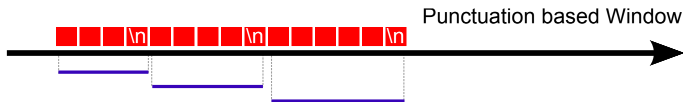
Punctuation based Windows

Split windows based on punctuations in the data



Punctuation based Windows

Split windows based on punctuations in the data



Potentially problematic if windows grow too large or too small.

Window Standing Query Example

What is the average of the integers in the window?

- Stream of integers
- Window of size $w = 4$
- Count based sliding window
- for the first w inputs, sum and count
- afterwards change average by adding $(i - j)/w$ to the previous window average

Window Standing Query Example

Window Standing Query Example

[1 3 5 4] 8 9 3 1 4 2 7 5 6 8 7

stream

$\frac{1+3+5+4}{4} = 3.25$

1 [3 5 4 8] 9 3 1 4 2 7 5 6 8 7

stream

1 3 [5 4 8 9] 3 1 4 2 7 5 6 8 7

stream

Window Standing Query Example

1 3 5 4 8 9 3 1 4 2 7 5 6 8 7

stream

1 3 5 4 8 9 3 1 4 2 7 5 6 8 7

stream

1 3 5 4 8 9 3 1 4 2 7 5 6 8 7

stream

$\frac{1+3+5+4}{4} = 3.25$

$3.25 + \frac{i-j}{w}$

with i newest value, j oldest value

Window Standing Query Example

[1 3 5 4] 8 9 3 1 4 2 7 5 6 8 7

stream

1 [3 5 4 8] 9 3 1 4 2 7 5 6 8 7

stream

1 3 [5 4 8 9] 3 1 4 2 7 5 6 8 7

stream

$\frac{1+3+5+4}{4} = 3.25$

$3.25 + \frac{i-j}{w}$

with i newest value, j oldest value

$\frac{1+3+5+4}{4} + \frac{8-1}{4} = 5$

Window Standing Query Example

1 3 5 4 8 9 3 1 4 2 7 5 6 8 7

stream

$\frac{1+3+5+4}{4} = 3.25$

1 **3 5 4 8** 9 3 1 4 2 7 5 6 8 7

stream

$3.25 + \frac{i-j}{w}$

with i newest value, j oldest value

$\frac{1+3+5+4}{4} + \frac{8-1}{4} = 5$

1 3 **5 4 8 9** 3 1 4 2 7 5 6 8 7

stream

$5 + \frac{9-3}{4} = 6.5$

Window Standing Query Example

1 3 5 4 8 9 3 1 4 2 7 5 6 8 7

stream

$\frac{1+3+5+4}{4} = 3.25$

1 3 5 4 8 9 3 1 4 2 7 5 6 8 7

stream

$3.25 + \frac{i-j}{w}$

with i newest value, j oldest value

$\frac{1+3+5+4}{4} + \frac{8-1}{4} = 5$

1 3 5 4 8 9 3 1 4 2 7 5 6 8 7

stream

$5 + \frac{9-3}{4} = 6.5$

Datastructure?

Window Average

```
#!/usr/bin/env python2
import sys
import Queue
WINDOW = 4
elems = Queue.Queue()
elem_sum = 0
for i in range(WINDOW): # initial average
    val = int(sys.stdin.readline().strip())
    elems.put(val)
    elem_sum += val

avg = float(elem_sum) / WINDOW
for line in sys.stdin:
    print(avg)
    val = int(line.strip())
    avg = avg + (val - elems.get())/float(WINDOW)
    elems.put(val)
```

Window Average

```python
#!/usr/bin/env python2
import sys
import Queue
WINDOW = 4
elems = Queue.Queue()
elem_sum = 0
for i in range(WINDOW): # initial average
    val = int(sys.stdin.readline().strip())
    elems.put(val)
    elem_sum += val

avg = float(elem_sum) / WINDOW
for line in sys.stdin:
    print(avg)
    val = int(line.strip())
    avg = avg + (val - elems.get())/float(WINDOW)
    elems.put(val)
```

Allows calculation in a single pass of each element.

Window based Algorithm

Lossy Counting

Problem Description

Maintain a count of distinct elements seen so far

Problem Description

Maintain a count of distinct elements seen so far

Examples:

- Google web crawler counting URL encounters.
- Detecting spam pages through content analysis.
- User login rankings to web services.

Problem Description

Maintain a count of distinct elements seen so far

Examples:

- Google web crawler counting URL encounters.
- Detecting spam pages through content analysis.
- User login rankings to web services.

Straight forward solution: Hashtable

Problem Description

Maintain a count of distinct elements seen so far

Examples:

- Google web crawler counting URL encounters.
- Detecting spam pages through content analysis.
- User login rankings to web services.

Straight forward solution: Hashtable

Too large for memory, too slow on disk

Algorithm Parameters

Environment Parameters

- Elements seen so far N

User-specified Parameters

- support threshold $s \in (0, 1)$
- error parameter $\epsilon \in (0, 1)$

Algorithm Guarantees

1. All items whose true frequency exceeds sN are output. There are no false negatives.
2. No items whose true frequency is less than $(s - \epsilon)N$ is output.
3. Estimated frequencies are less than the true frequencies by at most ϵN.

Example

With $s = 10\%, \epsilon = 1\%, N = 1000$

Example

With $s = 10\%, \epsilon = 1\%, N = 1000$

1. All elements exceeding frequency $sN = 100$ will be output.

Example

$$\text{With } s = 10\%, \epsilon = 1\%, N = 1000$$

1. All elements exceeding frequency $sN = 100$ will be output.
2. No elements with frequencies below $(s - \epsilon)N = 90$ are output. False positives between 90 and 100 might or might not be output.

Example

With $s = 10\%, \epsilon = 1\%, N = 1000$

1. All elements exceeding frequency $sN = 100$ will be output.
2. No elements with frequencies below $(s - \epsilon)N = 90$ are output. False positives between 90 and 100 might or might not be output.
3. All estimated frequencies diverge from their true frequencies by at most $\epsilon N = 10$ instances.

Example

With $s = 10\%, \epsilon = 1\%, N = 1000$

1. All elements exceeding frequency $sN = 100$ will be output.
2. No elements with frequencies below $(s - \epsilon)N = 90$ are output. False positives between 90 and 100 might or might not be output.
3. All estimated frequencies diverge from their true frequencies by at most $\epsilon N = 10$ instances.

Rule of thumb: $\epsilon = 0.1s$

Expected Errors

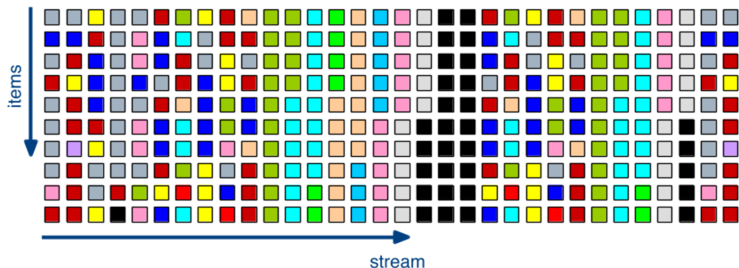
1. high frequency false positives
2. small errors in frequency estimations

Expected Errors

1. high frequency false positives
2. small errors in frequency estimations

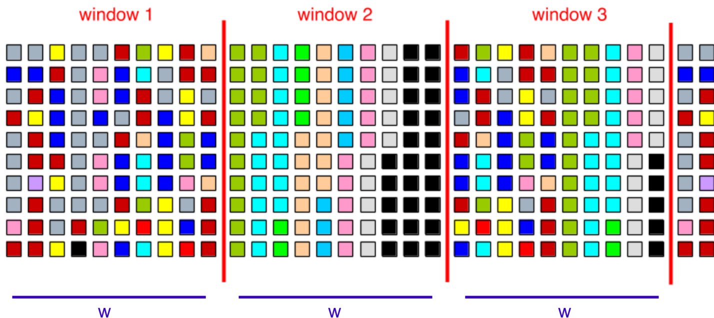
Acceptable for high numbers of N

Lossy Counting in Action



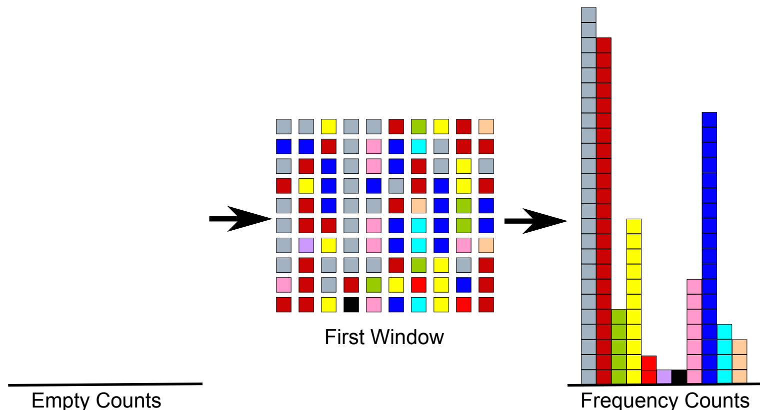
Incoming Stream of Colours

Divide into Windows/Buckets



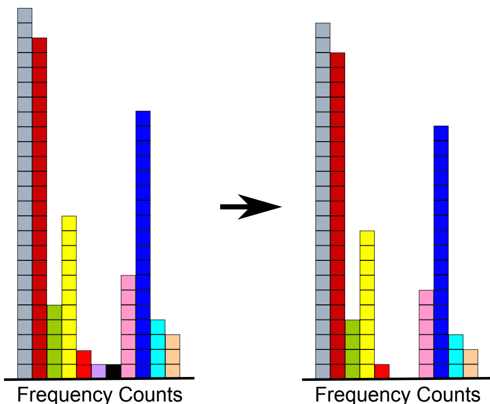
$$\text{Window Size } w = \left\lceil \frac{1}{\epsilon} \right\rceil = \left\lceil \frac{1}{0.01} \right\rceil = 100$$

First Window Comes In



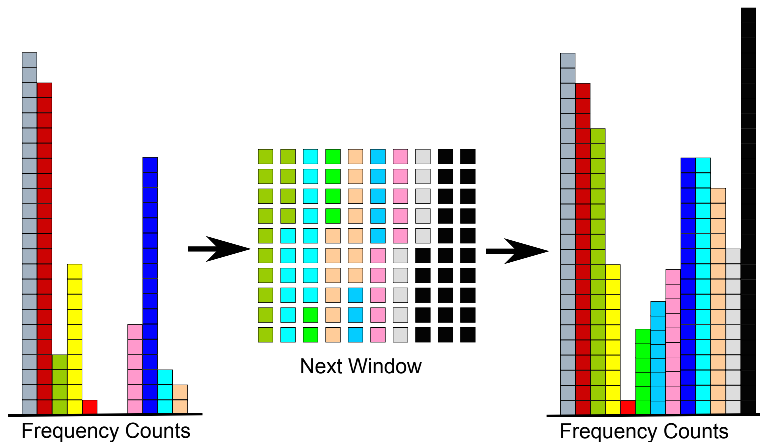
Go through elements. If counter exists, increase by one, if not create one and initialise it to one.

Adjust Counts at Window Boundaries



Reduce all counts by one. If counter is zero for a specific element, drop it.

Next Window Comes In



Count elements and adjust counts afterwards.

Lossy Counting Summary

- Split Stream into Windows
- For each window: Count elements, if no counter exists, create one.
- At window boundaries: Reduce all frequencies by one. If frequency goes to zero, drop counter.
- Process next window ...

Lossy Counting Summary

- Split Stream into Windows
- For each window: Count elements, if no counter exists, create one.
- At window boundaries: Reduce all frequencies by one. If frequency goes to zero, drop counter.
- Process next window ...

Data structure to save counters:

Lossy Counting Summary

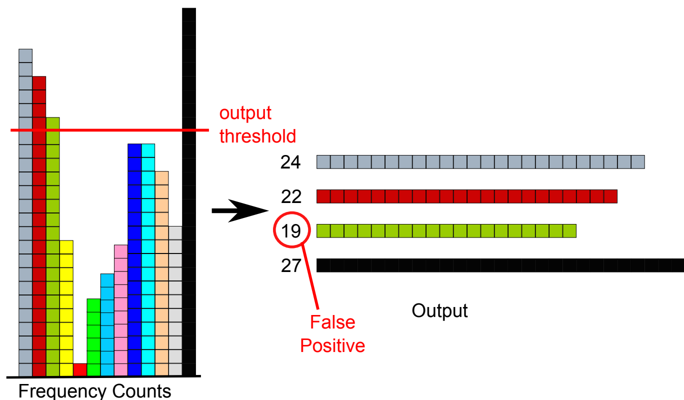
- Split Stream into Windows
- For each window: Count elements, if no counter exists, create one.
- At window boundaries: Reduce all frequencies by one. If frequency goes to zero, drop counter.
- Process next window ...

Data structure to save counters:

Hashtable<Color,Integer>

Output

With $s = 10\%, \epsilon = 1\%, N = 200$



To reduce false positives to acceptable amount, only output counters with frequency $f \geq (s - \epsilon)N = 18$.

Accuracy Improvement

Reduction step of counters follows the approach of reducing all counters by one. An improved version maintains exact frequencies and remembers for each counter at which window id it was created. At window boundaries, counters are only removed when their frequency falls below a certain level in relation to their window id.

(Color,Integer,WindowID)

See paper for details.

G. S. Manku, R. Motwani. Approximate Frequency Counts over Data Streams, VLDB, 2002.

Window based Algorithm

Sticky Sampling

Problem Description

Counting algorithm using a sampling approach.

Probabilistic sampling decides if a counter for a distinct element is created.

If a counter exists for a certain element, every future instance of this element will be counted.

Algorithm Parameters

Environment Parameters

- Elements seen so far N

User-specified Parameters

- support threshold $s \in (0, 1)$
- error parameter $\epsilon \in (0, 1)$
- probability of failure $\delta \in (0, 1)$

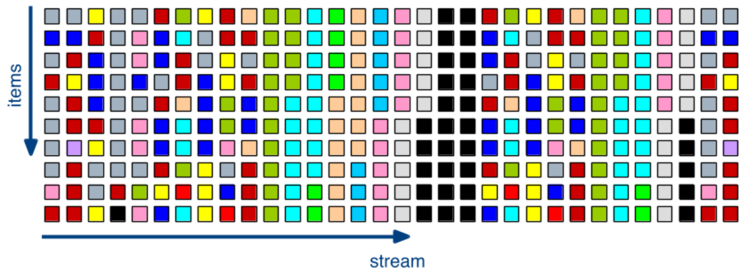
The algorithm is probabilistic and fails if any of the three guarantees is not satisfied.

Algorithm Guarantees

1. All items whose true frequency exceeds sN are output. There are no false negatives.
2. No items whose true frequency is less than $(s - \epsilon)N$ is output.
3. Estimated frequencies are less than the true frequencies by at most ϵN.

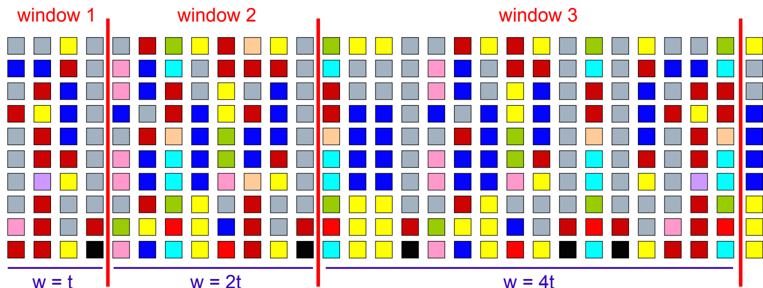
Guarantees and thereby expected errors are the same as for Lossy Counting. Except for the small probability that it might fail to provide correct answers.

Sticky Sampling in Action



Incoming Stream of Colours

Divide into Windows/Buckets

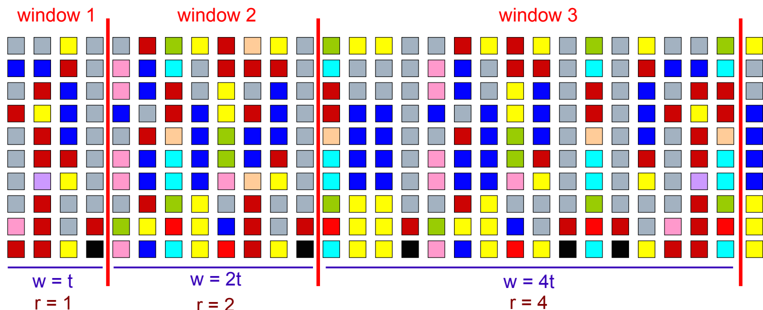


Dynamic window size with $t = \frac{1}{\epsilon} \log\left(\frac{1}{s\delta}\right)$

With $s = 10\%, \epsilon = 1\%, \delta = 0.1\%$

$t \approx 921$

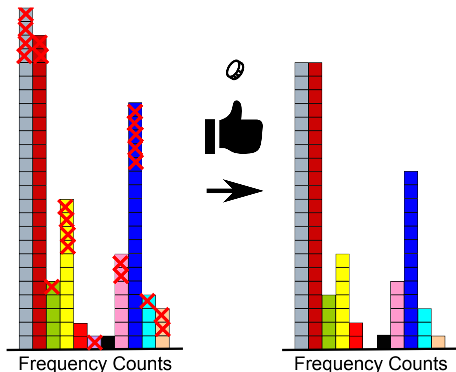
A Window Comes in



Go through elements. If counter exists, increase it. If not, create a counter with probability $\frac{1}{r}$ and initialise it to one.

Sampling rate r grows in proportion to window size.

Adjust Counts at Window Boundaries



Go through elements of each counter. Toss coin, if unsuccessful remove element, otherwise move on to next counter. If counter becomes zero, drop it.

Ensures uniform sampling

Sticky Sampling Summary

- Split stream into windows, doubling window size of each new window
- For each window: Go through elements if counter exists, increase it. If not, create one with probability $\frac{1}{r}$ with r growing at the same rate as window size.
- At window boundaries: Reduce all frequencies by tossing an unbiased coin for each counted element. Remove element if coin toss unsuccessful, otherwise move on to next counter. If frequency goes to zero, drop counter.
- Process next window ...

Output

Same principle as Lossy Counting

To reduce false positives to acceptable amount, only output counters with frequency $f \geq (s - \epsilon)N$.

Lossy Counting vs. Sticky Sampling

Feature	Lossy Counting	Sticky Sampling
Results	deterministic	probabilistic
Memory	grows with N	static (independent of N)
Theory	performs worse	performs better
Practice	performs better	performs worse

performance in terms of memory and accuracy