Algorithmic Data Analysis

Esther Galbrun
Autumn 2022
Part III

Mining Data Streams
Problem
Data streams

Vast amounts of data are acquired automatically

- satellite images, GPS traces
- measurements from wearable and mobile devices
- web server log traces
- user interactions on social networks
- credit card transactions

Continuous, large, rapid supply of data records
Storing all incoming data for offline processing is not possible

Algorithms must cope with amounts of incoming data many times larger than available memory

→ Data stream paradigm
We do not know the whole data in advance
We can think of the data as infinite and non-stationary
How to make calculations with only limited working storage?
Online algorithms

An **online algorithm** proceeds iteratively, processing its input piece by piece.

In online learning algorithms, the model is trained iteratively, slowly adjusting the parameters, one instance at a time. E.g., *Perceptron, Stochastic Gradient Descent* more generally.

Such algorithms naturally fit the streaming scenario.
Anytime algorithms

An anytime algorithm can return a valid solution before its process completes, the quality of solutions is expected to increase as it proceeds.

In the streaming scenario, the model might be regularly updated as new data points arrive, yet be required to answer queries as they are submitted.
Example

Meteorological sensor transmitting measures

**temperature**: slow variations, so the data rate need not be high

**wind speed**: requires a higher data rate

If the sensor sends a 4-bytes real number every tenth of a second, it produces about 3.5 MB of data per day

One isolated sensor is of little interest, a large number of measurement stations, each with several sensors is needed

One million such sensors generate about 3.5 TB per day (c.a. 1 per 150km$^2$ of land)
Stream from sensor sending a 4-bytes every tenth of a second produces about 3.5 MB of data per day

<table>
<thead>
<tr>
<th>Query</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raise alarm if the wind speed rises above $\tau$</td>
<td>0</td>
</tr>
<tr>
<td>Track maximum wind speed encountered</td>
<td>4 B</td>
</tr>
<tr>
<td>Report average wind speed over the last hour</td>
<td>144 kB</td>
</tr>
<tr>
<td>Report hourly average wind speed</td>
<td>$\sim$ 6 B</td>
</tr>
</tbody>
</table>
Example

Stream from sensor sending a 4-bytes every tenth of a second produces about 3.5 MB of data per day

<table>
<thead>
<tr>
<th>Query</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raise alarm if the wind speed rises above $\tau$</td>
<td>0</td>
</tr>
<tr>
<td>Track maximum wind speed encountered</td>
<td>4 B</td>
</tr>
<tr>
<td>Report average wind speed over the last hour</td>
<td>144 kB</td>
</tr>
<tr>
<td>Report hourly average wind speed</td>
<td>$\sim$ 6 B</td>
</tr>
</tbody>
</table>

Since we cannot store the entire stream, we cannot answer entirely arbitrary queries
Stream from sensor sending a 4-bytes every tenth of a second produces about 3.5 MB of data per day

<table>
<thead>
<tr>
<th>Query</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raise alarm if the wind speed rises above $\tau$</td>
<td>0</td>
</tr>
<tr>
<td>Track maximum wind speed encountered</td>
<td>4 B</td>
</tr>
<tr>
<td>Report average wind speed over the last hour</td>
<td>144 kB</td>
</tr>
<tr>
<td>Report hourly average wind speed</td>
<td>$\sim$ 6 B</td>
</tr>
</tbody>
</table>

Knowing what kind of queries might be asked, we can store samples or summaries from the stream appropriately, to be prepared to answer
Constraints encountered in the data stream paradigm include

**One-pass** data records can be processed only once

*It’s now or never*

A decision must be made when a new stream data point arrives: process (i.e. add to working storage) or discard (and lose the opportunity to ever process it)
Constraints encountered in the data stream paradigm include

**One-pass** data records can be processed only once

**Concept-drift** the data may evolve over time

Statistical properties of the data may change in unforseen ways, the underlying distribution is non-stationnary
Constraints encountered in the data stream paradigm include

**One-pass** data records can be processed only once
**Concept-drift** the data may evolve over time
**Resources** the system might need to drop part of the data

The data is generated by an external process
The rate of arrival is not controlled and might be irregular
At peak periods the system might not be able to handle all incoming data and has to shed part of it (*loadshedding*)
Constraints

Constraints encountered in the data stream paradigm include:

- **One-pass** data records can be processed only once.
- **Concept-drift** the data may evolve over time.
- **Resources** the system might need to drop part of the data.
- **Massive domain** for streams of discrete attributes, the number of distinct values might be very large.

Monitor an e-mail network with $10^8$ users to analyse pairwise interactions between users.

There are $10^{16}$ potential pairs, representing *petabytes* of storage.

Computing even simple statistics, like counting distinct values can be challenging.
Synopsis data structures
A synopsis is a concise representation of the data stream maintained dynamically in the working storage to be leveraged for answering queries.
A synopsis is a concise representation of the data stream maintained dynamically in the working storage to be leveraged for answering queries.

**Sampling data points**

Simple, flexible and generic synopsis data structure

Almost any algorithm can be applied to the sample

Unsuitable for a few specific queries such as counting distinct elements
A **synopsis** is a concise representation of the data stream maintained dynamically in the working storage to be leveraged for answering queries.

**Sampling data points**

Simple, flexible and generic synopsis data structure

Almost any algorithm can be applied to the sample

Unsuitable for a few specific queries such as counting distinct elements

**Fixed proportion sample** sample a chosen fraction $a/b$ of data points

For each incoming data point, generate an integer $\rho$ taken uniformly at random from $[1, b]$, store the point iff $\rho \leq a$
Synopsis data structures

A synopsis is a concise representation of the data stream maintained dynamically in the working storage to be leveraged for answering queries.

Sampling data points
Simple, flexible and generic synopsis data structure
Almost any algorithm can be applied to the sample
Unsuitable for a few specific queries such as counting distinct elements

Fixed proportion sample sample a chosen fraction $a/b$ of data points

Fixed size sample maintain a sample of $k$ data points
In the static case
Maintain a sample of $k$ data points
If the number of data points is $n$, each data point has probability $k/n$ to be included in the sample

In the data stream case
Maintain a *dynamically updated* sample of $k$ data points
Knowledge is incomplete, $n$ keeps increasing and points not in the sample are irrevocably lost

→ reservoir sampling
Reservoir sampling

Maintain a *dynamically updated* sample of $k$ data points

Two decisions need to be made

1. *Should the new data point be included in the reservoir?*
2. *Which point in the reservoir should be ejected to make room for the new one?*
Reservoir sampling

Maintain a *dynamically updated* sample of $k$ data points

Two decisions need to be made

1. *Should the new data point be included in the reservoir?*
   Insert $n^{\text{th}}$ incoming data point with probability $k/n$
   $k$ first points are included deterministically

2. *Which point in the reservoir should be ejected to make room for the new one?*
   If new point is inserted, eject an old point at random
Reservoir sampling

Maintain a *dynamically updated* sample of $k$ data points

1. Insert $n^{\text{th}}$ incoming data point with probability $k/n$
2. If new point is inserted, eject an old point at random

**Theorem**

After $n$ data points have arrived, the probability of any point being included in the reservoir is the same and equal to $k/n$
Reservoir sampling

Maintain a dynamically updated sample of $k$ data points

1. Insert $n^{th}$ incoming data point with probability $k/n$
2. If new point is inserted, eject an old point at random

Theorem
After $n$ data points have arrived, the probability of any point being included in the reservoir is the same and equal to $k/n$

Trivially true after initialization with the first $k$ points
Reservoir sampling

Maintain a *dynamically updated* sample of $k$ data points

1. Insert $n^{\text{th}}$ incoming data point with probability $k/n$
2. If new point is inserted, eject an old point at random

**Theorem**

After $n$ data points have arrived, the probability of any point being included in the reservoir is the same and equal to $k/n$

Assume the theorem is true after receiving $n - 1$ points, i.e. the probability of each point encountered so far to be included is $k/(n - 1)$

Trivially true for the $n^{\text{th}}$ arriving point, since it is inserted with probability $k/n$
Reservoir sampling

Maintain a *dynamically updated* sample of \( k \) data points

1. Insert \( n^{th} \) incoming data point with probability \( k/n \)
2. If new point is inserted, eject an old point at random

**Theorem**

*After \( n \) data points have arrived, the probability of any point being included in the reservoir is the same and equal to \( k/n \)*

Assume the theorem is true after receiving \( n - 1 \) points, i.e. the probability of each point encountered so far to be included is \( k/(n - 1) \)

For another point previously encountered
Reservoir sampling

<table>
<thead>
<tr>
<th>Event</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_I$ new point inserted</td>
<td>$k/n$</td>
</tr>
<tr>
<td>$E_R$ point currently in reservoir</td>
<td>$k/(n-1)$</td>
</tr>
<tr>
<td>$E_E$ point ejected</td>
<td>$1/k$</td>
</tr>
</tbody>
</table>

- $k/n$  \cdot \frac{k}{n-1} \cdot \frac{k-1}{k} + \frac{n-k}{n} \cdot \frac{k}{n-1} = \frac{k}{n}$

- point remains in reservoir
Reservoir sampling

Maintain a *dynamically updated* sample of $k$ data points

1. Insert $n^{th}$ incoming data point with probability $k/n$
2. If new point is inserted, eject an old point at random

**Theorem**

*After $n$ data points have arrived, the probability of any point being included in the reservoir is the same and equal to $k/n*

Assume the theorem is true after receiving $n - 1$ points, i.e. the probability of each point encountered so far to be included is $k/(n - 1)$

For another point previously encountered the probability to be still included in the reservoir after this step $k/n$
Reservoir sampling

Maintain a dynamically updated sample of $k$ data points

1. Insert $n^{th}$ incoming data point with probability $k/n$
2. If new point is inserted, eject an old point at random

Theorem
After $n$ data points have arrived, the probability of any point being included in the reservoir is the same and equal to $k/n$

Data is sampled uniformly, without decay
Cannot handle concept-drift
Recent data points are considered *more important*
Older data points are considered *stale*
→ More recent points should have higher probability to belong in the reservoir
Bias-sensitive sampling

Recent data points are considered *more important*
Older data points are considered *stale*
→ More recent points should have higher probability to belong in the reservoir

Associate bias $b(r, n)$ to $r^{th}$ point at arrival of $n^{th}$ point that increases with $r$ for fixed $n$ and decreases with $n$ for fixed $r$

Design sampling scheme s.t. the probability of $r^{th}$ point to be in the reservoir at arrival of $n^{th}$ point proportional to $b(r, n)$

Open problem for arbitrary bias function
Bias-sensitive sampling

Recent data points are considered *more important*
Older data points are considered *stale*
→ More recent points should have higher probability to belong in the reservoir

Associate bias \( b(r, n) \) to \( r^{\text{th}} \) point at arrival of \( n^{\text{th}} \) point that increases with \( r \) for fixed \( n \) and decreases with \( n \) for fixed \( r \)

Design sampling scheme s.t. the probability of \( r^{\text{th}} \) point to be in the reservoir at arrival of \( n^{\text{th}} \) point proportional to \( b(r, n) \)

Solved for *exponential bias function*
Bias-sensitive sampling

*Exponential bias function* with bias rate $\lambda \in [0, 1]$

$$b(r, n) = e^{-\lambda \cdot (n-r)}$$

*Memoryless bias functions*, the future probability of retaining a current point in the reservoir is independent of its past history or arrival time

Only interesting if $k < 1/\lambda$
Bias-sensitive sampling

Exponential bias function with bias rate $\lambda \in [0, 1]$

$$b(r, n) = e^{-\lambda \cdot (n-r)}$$

Let $F(n) \in [0, 1]$ be the fraction of the reservoir that is filled before arrival of $n^{th}$ data point.

The new point is inserted with probability $k \cdot \lambda$

A coin is flipped, with success probability $F(n)$
If success, the new point replaces a randomly selected point in the reservoir, otherwise, the new point is added.
Having generated a sample of data points, we can use it to estimate statistical properties of the data.

We consider a data stream where points correspond to individuals of particular ages.

We find that the average age in the sample is $\mu_A = 35.22$. 
Having generated a sample of data points, we can use it to estimate statistical properties of the data.

We consider a data stream where points correspond to individuals of particular ages. We find that the average age in the sample is $\mu_A = 35.22$.

Is this a good estimate of the average age in the stream?

It is important to quantify the accuracy of these estimates, i.e. bound the quality of query answers.

*Probabilistic inequalities* provide such bounds.
Theorem
Let $X$ be a random variable that takes on only nonnegative random values. Then, for any constant $\alpha$ satisfying $E[X] \leq \alpha$

$$P(X \geq \alpha) \leq \frac{E[X]}{\alpha}$$

$$E[X] = \int_{x} xf_X(x)dx$$

$$= \int_{0 \leq x < \alpha} xf_X(x)dx + \int_{\alpha \leq x} xf_X(x)dx$$

$$\geq \int_{\alpha \leq x} xf_X(x)dx$$

$$\geq \int_{\alpha \leq x} \alpha f_X(x)dx = \alpha P(X \geq \alpha)$$
Theorem
Let $X$ be a random variable that takes on only nonnegative random values. Then, for any constant $\alpha$ satisfying $E[X] \leq \alpha$

$$P(X \geq \alpha) \leq \frac{E[X]}{\alpha}$$

Markov’s inequality provides a bound on the upper tail of the probability distribution of nonnegative values.
Quality bounds: Markov’s inequality

We find that the average age in the sample is \( \mu_A = 35.22 \).

Since the variable takes nonnegative values, we can use Markov’s inequality

\[
P(X \geq \alpha) \leq \frac{E[X]}{\alpha}
\]

setting the value of \( \alpha \) to obtain the desired probability.

Solving \( 35.22/\alpha = 0.05 \) we get \( \alpha = 704.4 \).

We conclude that the age of any individual in the data stream is below 704.4 years with probability greater than 0.95.
Theorem
Let $X$ be an arbitrary random variable. Then, for any constant $\alpha$

\[ P(|X - E[X]| \geq \alpha) \leq \frac{\text{var}[X]}{\alpha^2} \]

Define $Y = (X - E[X])^2$ (nonnegative)
Notice that $E[Y] = \text{var}[X]$
Apply Markov’s inequality to bound $P(Y \geq \alpha^2)$
Quality bounds: Chebychev’s inequality

**Theorem**
Let $X$ be an arbitrary random variable. Then, for any constant $\alpha$

$$P(|X - E[X]| \geq \alpha) \leq \frac{\text{var}[X]}{\alpha^2}$$

Chebychev’s inequality provides a bound on *both tails* of the probability distribution of *arbitrary values*.
We find that the average age in the sample is $\mu_A = 35.22$ and the sample variance is $\sigma^2_A = 144.1916$

Chebychev’s inequality allows us to bound our estimate both from above and from below

$$P(|X - E[X]| \geq \alpha) \leq \frac{\text{var}[X]}{\alpha^2}$$

Solving $144.1916/\alpha^2 = 0.05$ we get $\alpha = 53.70$

We conclude that the age of any individual in the data stream is within the range $[-18.48, 89.92]$ with probability greater than 0.95
Theorem
Let $X$ be a random variable that can be expressed as the sum of $n$ independent Bernoulli random variables with success probabilities respectively $p_i$.

Then, for any $\delta \in [0, 1]$

$$P(X \leq (1 - \delta)E[X]) \leq e^{-E[X]\delta^2/2} \quad \text{(lower-tail bound)}$$

and for any $\delta \in [0, 2e - 1]$

$$P(X \geq (1 + \delta)E[X]) \leq e^{-E[X]\delta^2/4} \quad \text{(upper-tail bound)}$$

Convert inequality of $X$ into inequality of $e^{-tX}$ to create parameterized bound
Apply Markov’s inequality and optimize to get the tightest possible bound
Theorem
Let $X$ be a random variable that can be expressed as the sum of $n$ independent Bernoulli random variables with success probabilities respectively $p_i$.

Then, for any $\delta \in [0, 1]$

$$P(X \leq (1 - \delta)E[X]) \leq e^{-E[X]\delta^2/2} \quad (\text{lower-tail bound})$$

and for any $\delta \in [0, 2e - 1]$

$$P(X \geq (1 + \delta)E[X]) \leq e^{-E[X]\delta^2/4} \quad (\text{upper-tail bound})$$

Chernoff bounds are tighter than Markov’s and Chebychev’s inequalities, for sum of independent binary random variables.
Chernoff bounds apply to sums of binary random variables, hence we cannot use them to bound our estimate of age average.

We can use Chernoff bounds if our goal is to estimate the proportion of individuals older than 50, instead.

Indeed, we can consider the indicator “age $\geq 50$” as a binary variable, and take its average over the sample.
In our sample of 100 individuals, we count 12 persons of 50 years or older, i.e. a proportion of 0.12. This corresponds to the sum of 100 independent binary random variables being equal to 12.

We can use Chernoff bounds to bound our estimate

\[ P(X \leq (1 - \delta)E[X]) \leq e^{-E[X]\delta^2/2} \]  
\[ P(X \geq (1 + \delta)E[X]) \leq e^{-E[X]\delta^2/4} \]

(lower-tail bound)  
(upper-tail bound)

Solving \( e^{-12\delta_l^2/2} = 0.025 \) and \( e^{-12\delta_u^2/4} = 0.025 \), we get \( \delta_l = 0.7841 \) and \( \delta_u = 1.109 \) respectively.

We conclude that the proportion of individuals over 50 years old lies within \([0.026, 0.253]\) with probability greater than 0.95.
Quality bounds: Hoeffding’s inequality

Theorem
Let \( X \) be a random variable that can be expressed as the sum of \( n \) independent random variables, each bounded in \([l_i, u_i]\).

Then, for any \( \theta \geq 0 \)

\[
P(E[X] - X \geq \theta) \leq e^{-\frac{2\theta^2}{\sum_i (u_i-l_i)^2}} \quad \text{(lower-tail bound)}
\]

\[
P(X - E[X] \geq \theta) \leq e^{-\frac{2\theta^2}{\sum_i (u_i-l_i)^2}} \quad \text{(upper-tail bound)}
\]

Again, apply Markov’s inequality on parameterized bound and optimize to get the tightest possible bound.
Theorem
Let $X$ be a random variable that can be expressed as the sum of $n$ independent random variables, each bounded in $[l_i, u_i]$.

Then, for any $\theta \geq 0$

$$
P(E[X] - X \geq \theta) \leq e^{-\frac{2\theta^2}{\sum_i (u_i - l_i)^2}}$$  \text{(lower-tail bound)}

$$
P(X - E[X] \geq \theta) \leq e^{-\frac{2\theta^2}{\sum_i (u_i - l_i)^2}}$$  \text{(upper-tail bound)}

Hoeffding’s inequality is stronger than Markov’s and Chebychev’s inequalities and applies to sum of independent bounded random variables.
The age of an individual can realistically be assumed to be a bounded random variable from the range [0, 125] and our sample consists of 100 individuals. In other words, the sum of ages in our sample is the sum of 100 random variables each taking value in [0, 125].

We can use Hoeffding’s inequality to bound our estimate

\[
P(E[X] - X \geq \theta) \leq e^{-\frac{2\theta^2}{\sum_i (u_i - l_i)^2}} \quad \text{(lower-tail bound)}
\]

\[
P(X - E[X] \geq \theta) \leq e^{-\frac{2\theta^2}{\sum_i (u_i - l_i)^2}} \quad \text{(upper-tail bound)}
\]
In other words, the sum of ages in our sample is the sum of 100 random variables each taking value in $[0, 125]$. We can use Hoeffding’s inequality to bound our estimate

$$P(E[X] - X \geq \theta) \leq e^{-\frac{2\theta^2}{\sum_i (u_i - l_i)^2}} \quad \text{(lower-tail bound)}$$

$$P(X - E[X] \geq \theta) \leq e^{-\frac{2\theta^2}{\sum_i (u_i - l_i)^2}} \quad \text{(upper-tail bound)}$$

Solving $e^{-\frac{2\theta^2}{100 \cdot 125^2}} = 0.025$ we get $\theta = 1697.63$

We conclude that the average age is within the range $[18.24, 52.20]$ with probability greater than 0.95
Quality bounds: Hoeffding’s inequality

Alternatively, we can use Hoeffding’s inequality to calculate the size of the sample needed to obtain a chosen confidence interval

\[
P(E[X] - X \geq \theta) \leq e^{-\frac{2\theta^2}{\sum_i (u_i - l_i)^2}} \quad \text{(lower-tail bound)}
\]

\[
P(X - E[X] \geq \theta) \leq e^{-\frac{2\theta^2}{\sum_i (u_i - l_i)^2}} \quad \text{(upper-tail bound)}
\]

For example, if we want to be able to estimate the average age within ±5 years with confidence 0.95, we need at least

\[-\ln(0.025)/2 \cdot (125/5)^2 = 1153 \text{ samples}\]
Massive domain scenario

In many applications, the data stream contains discrete attributes with a very large number of distinct values (IP addresses, emails, etc.)

The values in the domain of a discrete attribute are called items. Each element in the data stream takes a value in the domain of the attribute, i.e. represents an occurrence of the corresponding item.

Binary attributes can be seen as constituting each a distinct item and an element constitute an occurrence of the item if it takes a positive value.
Massive domain scenario

In many applications, the data stream contains discrete attributes with a very large number of distinct values (IP addresses, emails, etc.)

Some simple queries can already be challenging

*Has this item occurred earlier in the stream?*
→ Finding duplicates

*Does this item occur in set $S$?*
→ Allowing only elements with a particular property

spam filtering: the elements of the stream are sender email addresses, $S$ are authorized senders (whitelisting)

Sampling schemes do not work well in such cases
Approximate counting

First streaming algorithm Morris' approximate counter (1977)

Information theory: exact counting till $N$ requires $\log_2(N)$ bits

Approximate counting till $N$ with only $\log_2(\log_2(N))$ bits, estimate has large variance

initialize counter $c \leftarrow 1$

increment counter $c \leftarrow c + 1$ with probability $2^{-c}$

output count return $2^c - 2$
**Approximate counting**

*First streaming algorithm* Morris’ approximate counter (1977)

*Information theory:* exact counting till $N$ requires $\log_2(N)$ bits

*Approximate counting* till $N$ with only $\log_2(\log_2(N))$ bits, estimate has large variance

Replace base 2 by base $q \in [1, 2]$ to adjust the space-accuracy tradeoff

**initialize counter** $c \leftarrow 1$

**increment counter** $c \leftarrow c + 1$ with probability $q^{-c}$

**output count** return $(q^c - q)/(q - 1)$
Hash functions

Hash functions are crucial ingredients of probabilistic streaming algorithms. They provide *reproducible randomness*.

Hash function $h$ maps every value in the input domain uniformly to a bit-string of fixed size.
Hash functions

Hash function $h$ maps every value in the input domain uniformly to a bit-string of fixed size.

Hash functions are also useful for sampling.

Sample one-tenth of all transactions vs. all transactions of one-tenth of the customers.

Vary sampling proportion dynamically at beginning, store all elements $x$ when the memory fills up, store only $x$, $h(x) = 0 \ldots$
when the memory fills up, store only $x$, $h(x) = 00 \ldots$

etc.
Bloom filters

Does this item occur in set $S$?

Bloom filters provide a means to answer set-membership queries probabilistically, when $S$ cannot be stored explicitly in a hash table.

False positives are possible, false negatives are not.
Bloom filters

A Bloom filter consists of

- a binary bit array $B$ of length $m$, whose elements are indexed from 0 to $m - 1$
- a set of $k$ independent hash functions $h_1, \ldots, h_k$, mapping elements from the data stream to an integer in $[0, m - 1]$ uniformly at random

\[ S = \{ \} \]
Bloom filters

Initialization  \[ B[i] \leftarrow 0, \text{ for } i = 0, \ldots, m - 1 \]

\begin{align*}
  h_1 & \quad h_2 & \quad h_3 & \quad \cdots & \quad h_k \\
  \begin{bmatrix}
    0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 \\
    0 & 1 & 2 & 3 & 4 & 5 & 6 & m - 1
  \end{bmatrix}
\end{align*}

\[ S = \{ \} \]
Adding element $x$ \quad $B[h_j(x)] \leftarrow 1$, for $j = 1, \ldots, k$

$S = \{a\}$
Adding element $x$  

$$B[h_j(x)] \leftarrow 1, \text{ for } j = 1, \ldots, k$$

$S = \{a, c\}$
Bloom filters

Querying element $x$  

$$\text{return } \bigwedge_{j=1,\ldots,k} B[h_j(x)]$$

$S = \{a, c\}$

Is $d$ in $S$? No
Bloom filters

Querying element \( x \)  

\[
\text{return } \bigwedge_{j=1,\ldots,k} B[h_j(x)]
\]

\( S = \{ a, c \} \)

Is \( c \) in \( S \)? Yes
Bloom filters

Querying element $x$ return $\bigwedge_{j=1,\ldots,k} B[h_j(x)]$

$S = \{a, c\}$

Is $b$ in $S$? Yes
Bloom filters

An element $x$ is reported as belonging to $S$ when it does not → a false positive occurs

$S = \{a, c\}$

Is $b$ in $S$? Yes
A false positive occurs if each of the different array bits $B[h_i(x)]$ has been set to 1 by some other element than $x$.

$S = \{a, c\}$

Is $b$ in $S$? Yes
Bloom filters

False positives are the result of collisions, multiple elements mapping onto the same bit

\[ S = \{ a, c \} \]

Is \( b \) in \( S \)? Yes
Theorem
Consider a Bloom filter $B$ of length $m$ with $k$ hash functions. Let $n$ be the number of distinct values in $S$, and $y \not\in S$. The probability that $y$ is reported as a false positive is

$$(1 - (1 - 1/m)^{kn})^k$$
**Theorem**
Consider a Bloom filter $B$ of length $m$ with $k$ hash functions.
Let $n$ be the number of distinct values in $S$, and $y \notin S$.
The probability that $y$ is reported as a false positive is

$$\left(1 - (1 - 1/m)^n\right)^k$$

Each $x \in S$ sets $k$ bits to 1.
The probability that none of these bits is the same as $h_j(y)$ is $(1 - 1/m)^k$.
Over $n$ distinct values in $S$, this probability is $(1 - 1/m)^{kn}$.
The probability that $h_j(y)$ is set to one by at least one value of $S$ is $1 - (1 - 1/m)^{kn}$.
The probability that this is true for all $j = 1, \ldots, k$ gives the result.
Bloom filters

The probability that \( y \) is reported as a false positive is

\[
(1 - (1 - 1/m)^{kn})^k
\]

Observing that \( (1 - 1/m)^m \approx e^{-1} \) allows to rewrite the false positive probability as

\[
(1 - e^{-kn/m})^k
\]

Too large or too small \( k \) leads to poor performances

Setting \( k = m \ln(2)/n \) minimizes the probability of false positives, to \( 2^{-m \ln(2)/n} \)
Bloom filters

For a fixed false positive probability $F$ the length of the Bloom filter needs to be proportional to the size of $S$

$$\frac{m}{n} = \frac{\ln(1/F)}{\ln(2)^2}$$

Bloom filters use bits to keep track of arbitrary element types

Bloom filters are very efficient and can be combined

- in cascade, with geometrically increasing values of $k$
- in parallel, taking a conjunction over different filters
Besides set-membership queries, Bloom filters can be used for alternative purposes.

The number of distinct values (number of items) in a set can be estimated as:

\[ |S| \approx \frac{m \ln(m/m_0)}{k} \]

where \( m_0 \) is the number of bits of value 0 in the filter for \( S \). The quality of the estimates drops as the filter fills up.
Bloom filters

Besides set-membership queries, Bloom filters can be used for alternative purposes.

number of distinct values in a set

The size of the union and of the intersection of different sets can be estimated by creating one filter for each. The bitwise OR of two filters can be shown to be the same as the filter representing their union. The size of the intersection is estimated by using the relation

\[ |S_1 \cap S_2| = |S_1| + |S_2| - |S_1 \cup S_2| \]
Bloom filters

Besides set-membership queries, Bloom filters can be used for alternative purposes:

- number of distinct values in a set
- size of the union and of the intersection of different sets

Setting the corresponding bits to zero allows limited tracking of deletions (one-time).

In this case, false negatives are also possible.
Besides set-membership queries, Bloom filters can be used for alternative purposes:

- number of distinct values in a set
- size of the union and of the intersection of different sets
- limited tracking of deletions

A generalization for tracking occurrence counts of items is known as the **count-min sketch**.

A space-efficient, dedicated technique, for counting distinct values is the **Flajolet–Martin algorithm**.
A count-min sketch consists of

- a numerical array $C$ with $m$ columns and $k$ rows, where elements of each row are indexed from 0 to $m - 1$
- a set of $k$ pairwise-independent hash functions $h_1, \ldots, h_k$ mapping elements from the data stream to an integer in $[0, m - 1]$ uniformly at random, one for each row

<table>
<thead>
<tr>
<th>$h_1$</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>...</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$h_k$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>$m - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Count-min sketch

### Initialization

\[ C[j, i] \leftarrow 0, \text{ for } i = 0, \ldots, m - 1, j = 1, \ldots, k \]
Count-min sketch

Adding element $x$  $C[j, h_j(x)]$ is incremented by 1, for $j = 1, \ldots, k$
**Count-min sketch**

Querying element $x$ returns $\min_{j=1,\ldots,k} C[j, h_j(x)]$

![Diagram](image-url)
**Count-min sketch**

Querying element \( x \)  
\[
\text{return } \min_{j=1,\ldots,k} C[j, h_j(x)]
\]

Collisions of nonnegative occurrence counts of distinct items cause overestimations.
Theorem
Let $E(v)$ be the estimate of the occurrence count of item $v$ from a count-min sketch of size $k \times m$. Let $n_T$ be the sum of occurrences counts of all items (number of elements received so far), and $G(v)$ the true occurrence count of item $v$. Then, with probability at least $1 - e^{-k}$

$$E(v) \leq G(v) + \frac{n_T \cdot e}{m}.$$
Theorem
Let $E(v)$ be the estimate of the occurrence count of item $v$ from a count-min sketch of size $k \times m$. Let $n_T$ be the sum of occurrences counts of all items (number of elements received so far), and $G(v)$ the true occurrence count of item $v$. Then, with probability at least $1 - e^{-k}$

$$E(v) \leq G(v) + \frac{n_T \cdot e}{m}.$$ 

Markov’s inequality is used to bound the probability that the number of elements mapping to the same cell as $v$ in a given row exceeds $n_T \cdot e/m$. This needs to be repeated in all of the $k$ rows for the overestimation to happen.
Theorem
Let $E(v)$ be the estimate of the occurrence count of item $v$ from a count-min sketch of size $k \times m$. Let $n_T$ be the sum of occurrences counts of all items (number of elements received so far), and $G(v)$ the true occurrence count of item $v$. Then, with probability at least $1 - e^{-k}$

$$E(v) \leq G(v) + \frac{n_T \cdot e}{m}.$$

The error can be bounded with a user defined tolerance $\epsilon n_T$ and probability at least $1 - \delta$ by setting $m = e/\epsilon$ and $k = \ln(1/\delta)$.
If elements have arbitrary \textbf{positive} weights

\[ n_T \] is the sum of element weights
Count-min sketch

If elements have arbitrary positive or negative weights, cells are incremented by the relevant weight. $n_T$ is the sum of absolute values of element weights. The median value among the cells is reported.

Then, with probability at least $1 - e^{-k/4}$

$$G(i) - \frac{3n_T \cdot e}{m} \leq E(i) \leq G(i) + \frac{3n_T \cdot e}{m}$$
The **dot product of the occurrence counts** of items across two data streams is useful for estimating their join size.

It can be estimated by constructing separate count-min sketches for the two streams, having the same size and sharing the same hash functions.

Dot products of the corresponding pairs of rows are computed and the minimum is reported.

The error on the estimate can be bounded with a tolerance $n_{T_1} \cdot n_{T_2} \cdot e/m$ and probability at least $1 - e^{-k}$.

Other applications include calculating quantiles and identifying frequent items (a.k.a. **heavy hitters**).
Counting distinct values

Estimating the number of distinct values (i.e. number of items)
How many distinct email addresses appear in the data stream?

Bloom filters can be used to count distinct values

The Flajolet–Martin algorithm provides a space-efficient alternative when set-membership queries are not required
Hash function $h$ maps each element to a bit-string.

The number of distinct values can be estimated by choosing sufficiently large bit-strings, so that there are more possible results of the hash function than there are values in the domain.
Flajolet–Martin algorithm

Hash function $h$ maps each element to a bit-string

- elements with same value are always mapped to the same bit-string
- the more distinct values, the more different bit-strings should be seen
- the more different bit-strings, the more likely to see an unusual one
- unusual here is “ending with many 0s”
Hash function $h$ maps each element to a bit-string.
Choose a bit-string length $L$ such that $2^L$ upper-bounds the number of distinct values.
64 bits is a typical choice because it is convenient for implementation and sufficiently large for most applications.
Hash function $h$ maps each element to a bit-string
Choose a bit-string length $L$ such that $2^L$ upper-bounds the number of distinct values
Let $r(x)$ be the number of trailing zeros in $h(x)$
Let $R$ be the maximum of this value over encountered elements
Report $2^R$ as estimate of the number of distinct values
Flajolet–Martin algorithm

Proof of correctness requires complex mathematical analysis. It can be shown that the estimator $2^R$ has a systematic bias

$$E_n[R] = \log_2(\phi n) + \Psi(\log_2(n)) + o(1)$$

where $\Psi$ is an oscillating function of negligible amplitude and

$$\phi = 2^{-1/2} e^\gamma \frac{2}{3} \prod_{p=1}^\infty \left( \frac{(4p + 1)(4p + 2)}{(4p)(4p + 3)} \right)^{-1-\nu(p)} \approx 0.77351$$

with $\gamma$ denoting Euler’s gamma constant and $\nu(p)$ the number of 1-bits in the binary representation of $p$.

So $2^R/\phi$ is an unbiased estimator of $n$. 
Computing moments

Let $V$ be the domain of a discrete attribute and for each value $v$ in this domain (i.e. each item), let $c_v$ be the count of occurrences of that value in the data stream.

The $k^{th}$-order moment of this attribute in the data stream is

$$F_k = \sum_{v \in V} c_v^k$$

The zeroth-order moment is the number of distinct values.
The first-order moment is the sum of occurrence counts, i.e. the length of the stream.
The second-order moment measures the skew in occurrence counts.
The second-order moment

$$F_2 = \sum_{v \in V} c_v^2$$

can be seen as a variant of the Gini index, measuring the skew in occurrence counts.

If the skew is large, $F_2$ is large and close to its upper bound

$$\left( \sum_{v \in V} c_v \right)^2$$

It also provides an estimate of the self-join size of the stream. It is sometimes called the *surprise number*.
Computing moments

The Alon–Matias–Szegedy (AMS) sketch provides an estimate of the second-order moment when it is not possible to store the occurrence counts for all distinct values.
The sketch contains $m$ components. Each sketch component is associated with a 4-wise independent hash function mapping elements from the data stream to a binary value in $\{-1, 1\}$ at random.
Alon–Matias–Szegedy sketch

Each sketch component $Q_i$ is associated with a 4-wise independent hash function mapping elements from the data stream to a binary value $h_i(x) = r_x^{(i)} \in \{-1, 1\}$ at random.

For each incoming stream element $x$, each sketch component is updated by adding to it the corresponding hash value generated for the element, i.e. $Q_i \leftarrow Q_i + r_x^{(i)}$.

Therefore, the value $Q$ of a sketch component is a random variable depending on the random binary value $r_v$ generated by the hash function for each value $v$ in the domain of the attribute, such that

$$Q = \sum_{v \in V} c_v \cdot r_v$$
Alon–Matias–Szegedy sketch

Each sketch component $Q_i$ is associated with a 4-wise independent hash function mapping elements from the data stream to a binary value $r_x^{(i)} \in \{-1, 1\}$ at random

$$Q = \sum_{v \in V} c_v \cdot r_v$$

**Theorem**

*The second-order moment can be estimated by the square of a sketch component*

$$E[Q^2] = F_2$$

$$Q^2 = \sum_{v \in V} c_v^2 r_v^2 + 2 \sum_{v \in V} \sum_{v' \in V} c_v c_{v'} r_v r_{v'}.$$ For any pair of hash values $r_v^2 = r_{v'}^2 = 1$, and $E[r_v \cdot r_{v'}] = E[r_v] \cdot E[r_{v'}] = 0$ since hash functions are 2-wise independent. Hence $E[Q^2] = \sum_{v \in V} c_v^2 = F_2$. 
Each sketch component $Q_i$ is associated with a 4-wise independent hash function mapping elements from the data stream to a binary value $r^{(i)}_x \in \{-1, 1\}$ at random

$$Q = \sum_{v \in V} c_v \cdot r_v \quad E[Q^2] = F_2$$

**Theorem**

The variance of the square of a sketch component is bounded above by twice the second-order moment

$$\text{var}(Q^2) \leq 2F^2_2$$
Alon–Matias–Szegedy sketch

Each sketch component $Q_i$ is associated with a 4-wise independent hash function mapping elements from the data stream to a binary value $r^{(i)}_x \in \{-1, 1\}$ at random

$$Q = \sum_{v \in V} c_v \cdot r_v \quad E[Q^2] = F_2 \quad \text{var}(Q^2) \leq 2F_2^2$$

A tighter estimate can be obtained by combining the $m$ sketch components using the mean-median trick.
Alon–Matias–Szegedy sketch

\[ Q = \sum_{v \in V} c_v \cdot r_v \quad E[Q^2] = F_2 \quad \text{var}(Q^2) \leq 2F_2^2 \]

A tighter estimate can be obtained by combining the \( m \) sketch components using the \textit{mean-median trick}.

The \textit{mean-median trick} is commonly used in probabilistic analysis to robustly estimate a random variable when its variance is no larger than a modest factor of the square of its expected value.
With the *mean-median trick* the sketch can be used to approximate the second-order moment to within $1 \pm \epsilon$ with probability at least $1 - \delta$

Setting the number of sketches $m$ to be $O\left(\ln(1/\delta)/\epsilon^2\right)$

The sketch components are partitioned into $O\left(\ln(1/\delta)\right)$ groups of size $O\left(1/\epsilon^2\right)$

The median of the group averages is reported as final estimate
Mean-median trick

A combination of Chebychev’s inequality and Chernoff bounds is used to bound the accuracy of the estimate.

The variance of each sketch component is at most $2F_2^2$. By using the average of $16/\epsilon^2$ independent sketch components, the variance of the averaged estimate is reduced to $F_2^2 \cdot \epsilon^2/8$. Chebychev’s inequality shows that the $\epsilon$-bound is violated by the average with probability at most $1/8$.

Let $Y$ be a random variable defined as the sum of the Bernoulli indicator variables of $\epsilon$-bound violations of $q = 4 \ln(1/\delta)$ such averaged independent estimates. The expected value of $Y$ is $q/8$. Chernoff Bound shows that

$$P(Y > q/2) = P(Y > (1+3)q/8) = P(Y > (1+3)E[Y]) \leq e^{-3^2 \ln(1/\delta)/8} = \delta^{9/8} \leq \delta$$

The median can violate the $\epsilon$-bound only when more than half the group averages violate the bound, which happens with probability $P(Y > q/2)$.
Some queries can be resolved using either the count-min sketch or the Alon–Matias–Szegedy sketch. The bounds provided by the two methods are different, but neither is strictly better than the other in all cases. The count-min sketch has the advantage of being more intuitive and easier to integrate into clustering and classification algorithms.
## Synopsis data structures

<table>
<thead>
<tr>
<th>reservoir sampling</th>
<th>generic queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloom filters</td>
<td>testing set-membership</td>
</tr>
<tr>
<td>Flajolet–Martin algorithm</td>
<td>counting distinct values</td>
</tr>
<tr>
<td>count-min sketch</td>
<td>counting item occurrences</td>
</tr>
<tr>
<td>Alon–Matias–Szegedy sketch</td>
<td>estimating 2\textsuperscript{nd}-order moment</td>
</tr>
</tbody>
</table>
Finding frequent items

What is popular?

The count-min sketch and Alon–Matias–Szegedy sketch can be used to determine frequent items (a.k.a. *heavy-hitters*).

Sketches are generally better at estimating the occurrence counts of the more frequent items as compared to rare ones.
Theorem
Let $E(v)$ be the estimate of the occurrence count of item $v$ from a count-min sketch of size $k \times m$. Let $n_T$ be the sum of occurrences counts of all items (number of elements received so far), and $G(v)$ the true occurrence count of item $v$. Then, with probability at least $1 - e^{-k}$

$$E(v) \leq G(v) + \frac{n_T \cdot e}{m}.$$
Finding frequent items

What is popular?

The count-min sketch and Alon–Matias–Szegedy sketch can be used to determine frequent items (a.k.a. heavy-hitters)

Sketches are generally better at estimating the occurrence counts of the more frequent items as compared to rare ones

\[ E(v) \leq G(v) + \frac{n_T \cdot e}{m} \]

The bound is on absolute basis and the error depends on the aggregated counts, not the individual item

In relative terms, the estimates are better for frequent items
Finding frequent items: lossy counting algorithm

The stream is divided into segments of size \( w = \lceil 1/\epsilon \rceil \)
When a new element arrives, the occurrence count of the corresponding item is updated
When a segment boundary is reached, all counts are decremented by 1, items with counts of zero are pruned

When \( n \) items have been processed
\( O(n/w) = O(n\epsilon) \) segments have been processed
Any count has been decremented at most \( O(n\epsilon) \) times

If \( \lfloor n\epsilon \rfloor \) is added to all counts, none would be underestimated
Reporting frequent items using this overestimate, might lead to some false positives but no false negatives
The amount of false positives is adjusted by tuning \( \epsilon \)
This algorithm can be generalized for finding frequent itemsets by batching $\eta$ segments. That is, $\eta$ segments are read into memory and a frequent itemset mining algorithm is applied. The counts of occurrence are maintained for itemsets instead of items, and decremented by $\eta$ after each batch. The value of $\eta$ can be set depending on available memory. Combining frequent itemset mining and reservoir sampling offers an alternative approach, which is better able to adjust to concept drift.
Finding frequent items

What is currently popular?

Use a decaying window to aggregate occurrences with decaying weights, such that older occurrences are discounted.

Consider a stream of elements $x_1, x_2, \ldots x_t$, where $x_1$ is the first element to arrive, i.e. oldest one, and $x_t$ the most recent one.

Let $\gamma$ be a small constant, e.g. $10^{-6}$ or $10^{-9}$.

The **exponentially decaying window** for this stream is

$$
\sum_{i=0}^{t-1} x_{t-i}(1 - \gamma)^i
$$
Finding frequent items

Exponentially decaying window

\[
\sum_{i=0}^{t-1} x_{t-i}(1 - \gamma)^i
\]

weights are spread out back in time across the whole stream
Only keep track of the sum, when a new element \(x_{t+1}\) arrives, multiply the current sum by \(1 - \gamma\) and add \(x_{t+1}\) to it

In contrast, a **window of fixed width** \(1/\gamma\) with equal sum of weights puts equal weight 1 on the \(1/\gamma\) most recent elements and weight 0 on all previous elements
Must keep track of elements to discard them when they fall off
Finding currently popular items

For each item, imagine a separate stream where elements take value 1 when the corresponding original element is an occurrence of the item and 0 otherwise.

The popularity of each item is measured by applying the exponentially decaying window with parameter $\gamma$ to this stream.
Finding currently popular items

For each item, imagine a separate stream where elements take value 1 when the corresponding original element is an occurrence of the item and 0 otherwise.

The popularity of each item is measured by applying the exponentially decaying window with parameter $\gamma$ to this stream.

The number of distinct items might be very large, making it impossible to store all the different scores.

Threshold $\tau < 1$ is chosen, scores going below will be dropped.
Finding currently popular items

Threshold $\tau < 1$, small constant $\gamma$

When a new element arrives, assumed to be an occurrence of item $v$

- Each score currently maintained is multiplied by $1 - \gamma$
- Score for item $v$ is incremented by 1
  - if necessary new score is created for $v$ and initialized to 1
- Any score below $\tau$ is dropped
Finding currently popular items

Threshold $\tau < 1$, small constant $\gamma$

When a new element arrives, assumed to be an occurrence of item $v$

- Each score currently maintained is multiplied by $1 - \gamma$
- Score for item $v$ is incremented by 1
  - if necessary new score is created for $v$ and initialized to 1
- Any score below $\tau$ is dropped

The sum of all scores is $1/\gamma$, hence there cannot be more than $\ell/\gamma$ items with score of $1/\ell$ or more

In most cases, occurrences are concentrated on a few items, much fewer scores are maintained at any time
Classification
Concept-drift makes streaming classification challenging

One simple solution is to use a sample from the data stream, e.g. obtained via decay-based reservoir sampling

The challenges of streaming are addressed during sampling
Any conventional classification algorithm can be used

The accuracy of the model might deteriorate over time
Might need to retrain periodically with latest sample
Monitor the performance of the model and trigger retraining

Ensemble methods can help address concept-drift, selecting the model that is best suited for a particular portion of the data stream
Learning decision trees from a data stream

A dedicated approach has been proposed to learn decision trees from a data stream.

A carefully designed approach allows building a decision tree on a sample of a very large dataset such that the resulting tree is the same as would have been obtained on the original dataset, with high probability.

The key is to determine the point at which a sufficiently large data sample is available so that a split can be performed, that is the same as would have been performed with knowledge of the full data, with high probability.

Hoeffding’s inequality is used to determine when such point is reached.
Hoeffding trees

At a given step of the tree construction, consider two candidate splits with scores $s_a$ and $s_b$, respectively, when evaluated on the entire data, and scores $s'_a$ and $s'_b$, respectively, when evaluated in the same way on the sample data.

Because of discrepancies due to sampling, it might be the case that $s'_a < s'_b$ but $s_b < s_a$, leading to a different choice of split on the sample data and on the entire data.

Hoeffding’s inequality can be used to bound the probability that such undesirable case happens, as a function of the sample size.
At a given step of the tree construction, consider two candidate splits with scores $s_a$ and $s_b$, respectively, when evaluated on the entire data, and scores $s'_a$ and $s'_b$, respectively, when evaluated in the same way on the sample data.

Let $R$ denote the range of the split evaluation measure. Let $\epsilon = s'_b - s'_a > 0$ and $\delta < 1$.

If the candidate splits are evaluated on a sample of size at least

$$n = \frac{R^2 \cdot \ln(1/\delta)}{2\epsilon^2}$$

the undesirable case where $s_b < s_a$ will not occur with probability at least $1 - \delta$. 
If the candidate splits are evaluated on a sample of size at least

$$n = \frac{R^2 \cdot \ln(1/\delta)}{2\epsilon^2}$$

the undesirable case where \( s_b < s_a \) will not occur with probability at least \( 1 - \delta \).

For a fixed sample size, the Hoeffding tree approach determines whether the difference between the scores of the best and second-best split candidates is sufficiently large to warrant performing the split.

In the streaming scenario, with continuously accumulating samples, the key is to wait for the sample to be large enough before performing the split.
Hoeffding trees

tree $T$ initially consists of a single leaf (also the root)

for each new incoming element $x$
do
    sort $x$ to the leaf $\ell$ of $T$ where it belongs
    update the occurrence counts in $\ell$
    label $\ell$ with the majority class among its elements
if $\ell$ contains elements from different classes then
    generate candidate splits using the occurrence counts in $\ell$
    $s_a, s_b \leftarrow$ scores of the first and second best candidate splits
    $\epsilon \leftarrow \sqrt{R^2 \cdot \ln(1/\delta)/(2n_\ell)}$
    if $s_b - s_a > \epsilon$ then
        replace $\ell$ by the best split
        for each branch of the split do
            add a new leaf with empty occurrence counts
return $T$
Very Fast Decision Trees

Near ties correspond to small values of $\epsilon$, which will require large sample sizes, hence long waiting time before the split can be performed. A lot of time is spent on breaking ties.

The Very Fast Decision Trees (VFDT) algorithm improves over Hoffding trees by

- breaking ties more aggressively
- deactivating less promising leaf nodes
- dropping poor splitting attributes
Very Fast Decision Trees

The Very Fast Decision Trees (VFDT) algorithm improves over Hoeffding trees by

- breaking ties more aggressively
- deactivating less promising leaf nodes
- dropping poor splitting attributes

It was subsequently modified to address concept-drift, by

- using a sliding window of training samples
- growing an alternative subtree when a subtree seems out-of-date, to replace it

resulting in the Concept-adapting Very Fast Decision Trees (CVFDT) algorithm
Mining data streams presents several challenges:

- **High volume**
- **Resources constraints**
- **Massive domain**
- **Concept-drift**

A high-quality synopsis of the data stream goes a long way. The choice of synopsis depends on the application at hand:

- **Task**: what queries need to be answered
- **Data**: number and expected domain size of attributes
- **Resources**: amount of memory available, desired latency

Results are estimates, it is important to quantify their accuracy.