14 / July / 2017

Security Level: **Open**

Online distributed streaming machine learning

Big Data, Fast Data, All Data

Dr. Cristian Axenie

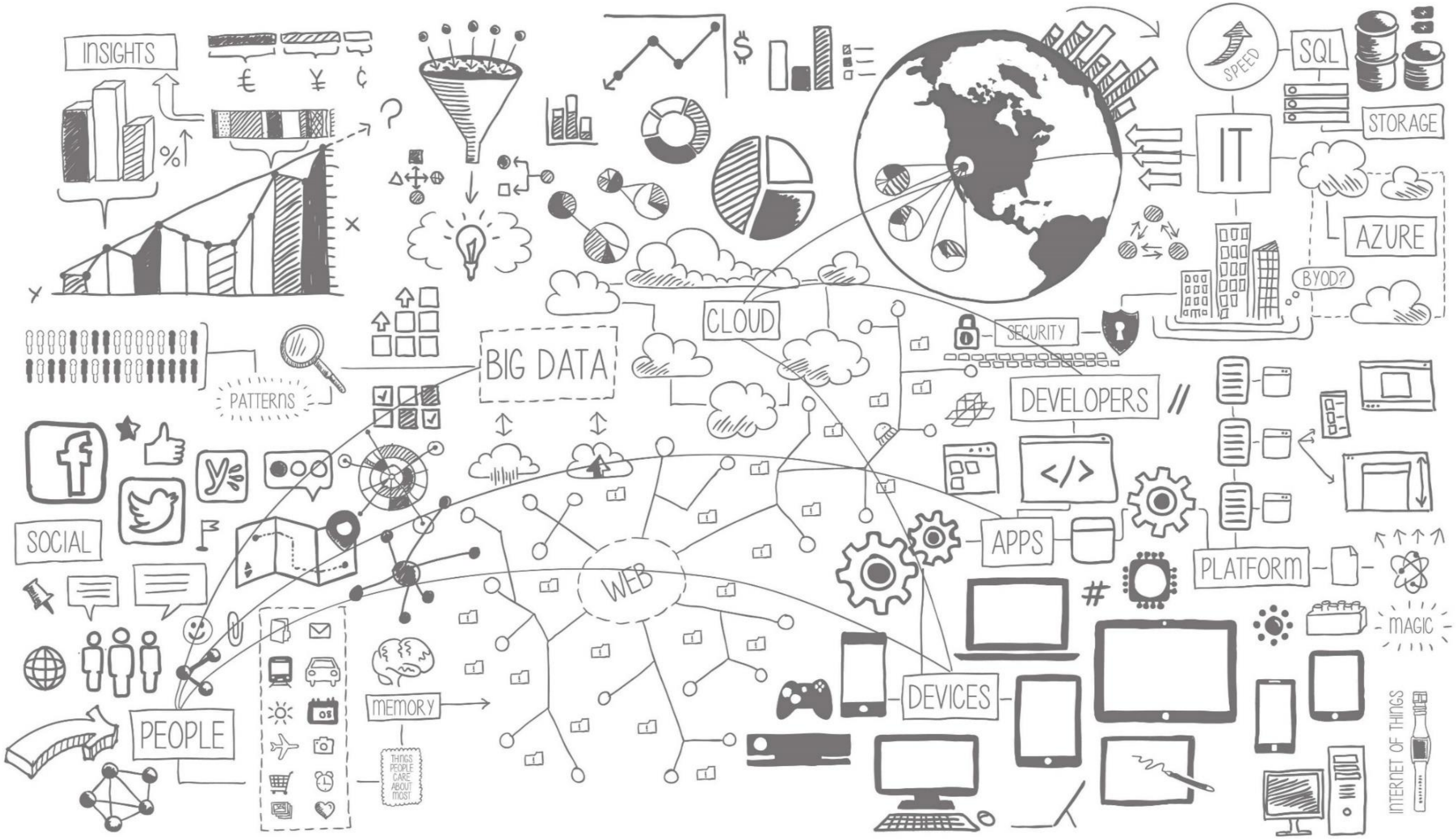
Senior Research Engineer

Big Data Streaming Technology Team

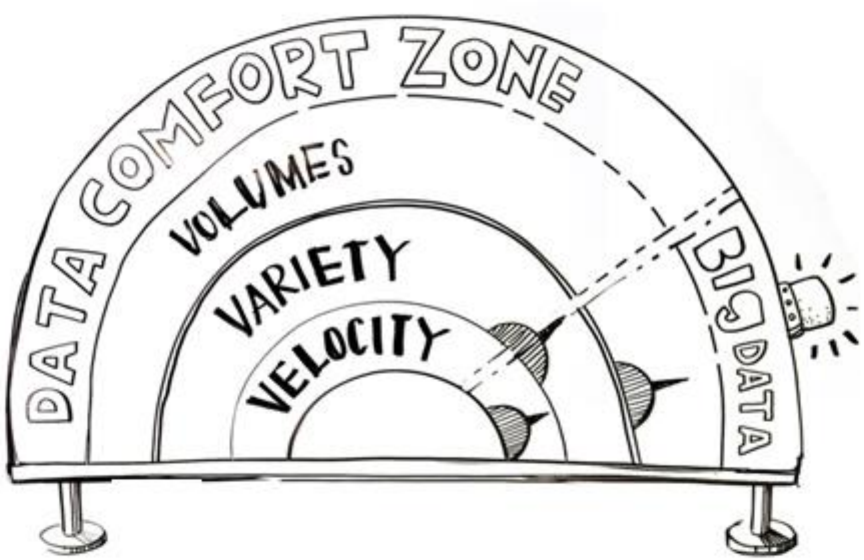
HUAWEI

European Research Center

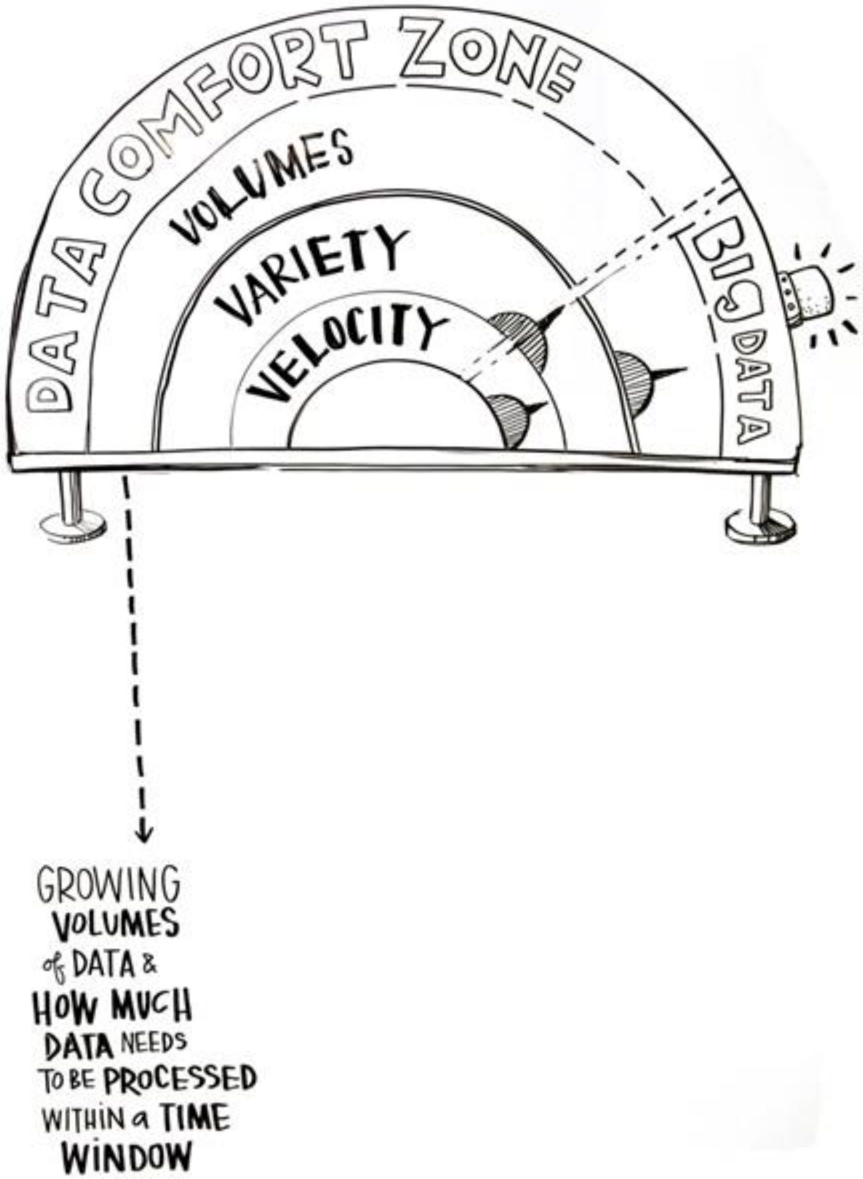
(Dis)ambiguating Big Data



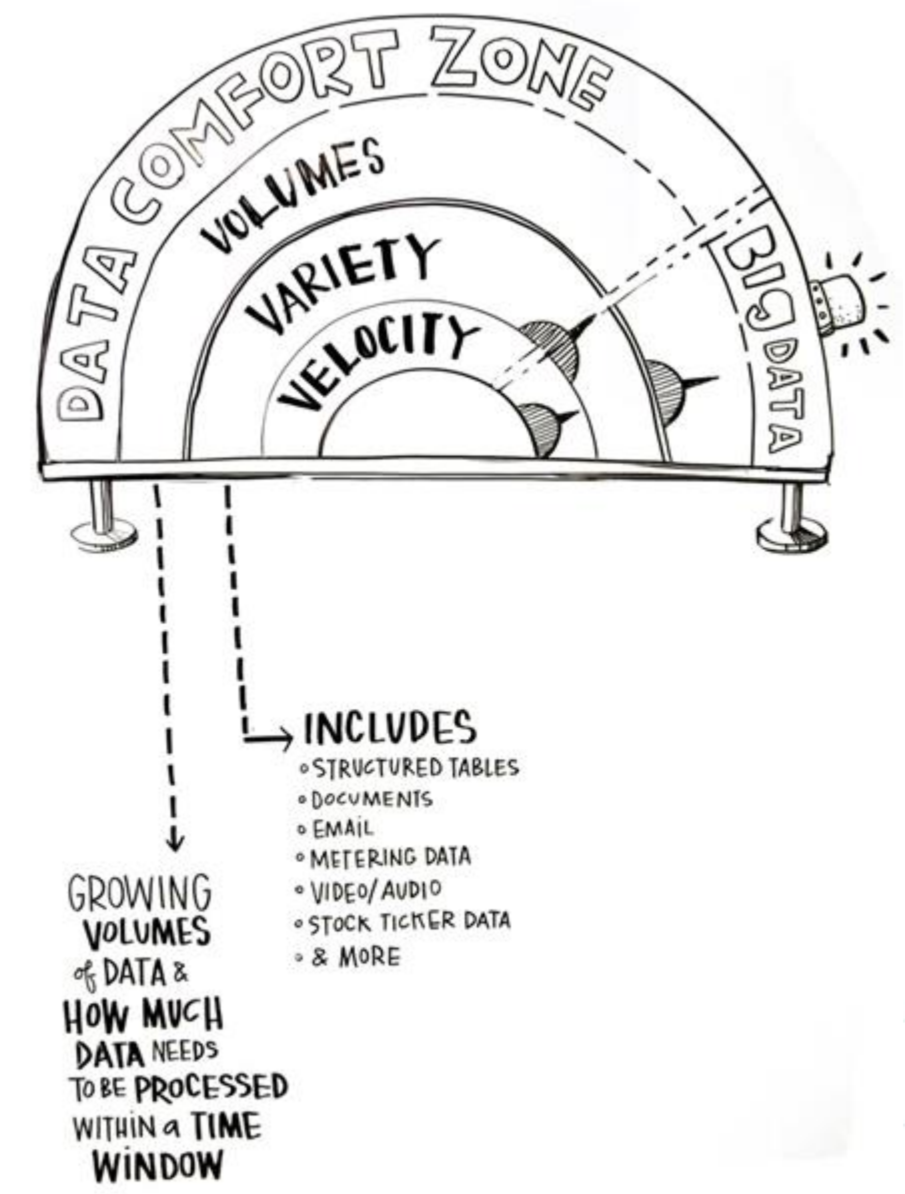
Big Data in a Nutshell



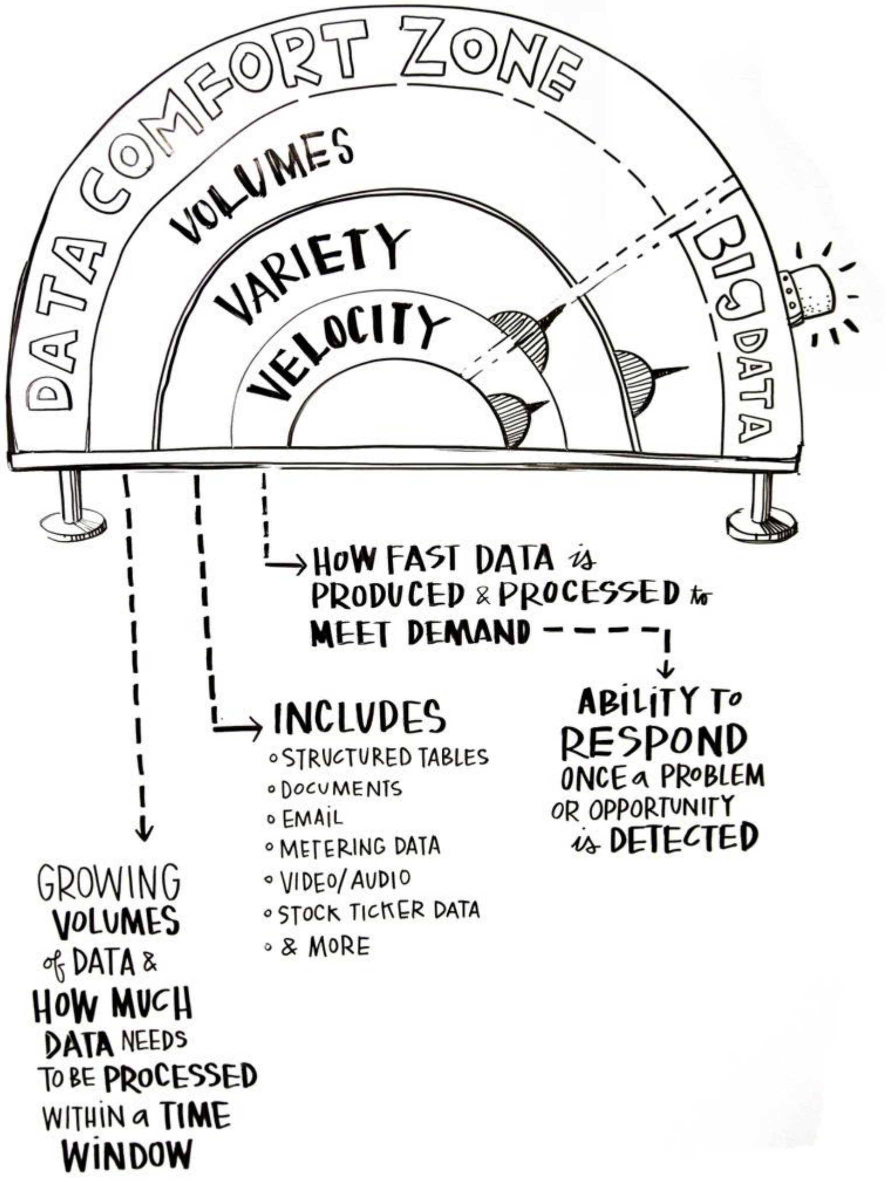
Big Data in a Nutshell



Big Data in a Nutshell



Big Data in a Nutshell



Big Data Stream Processing: A gentle introduction

Big Data Stream Processing: A gentle introduction

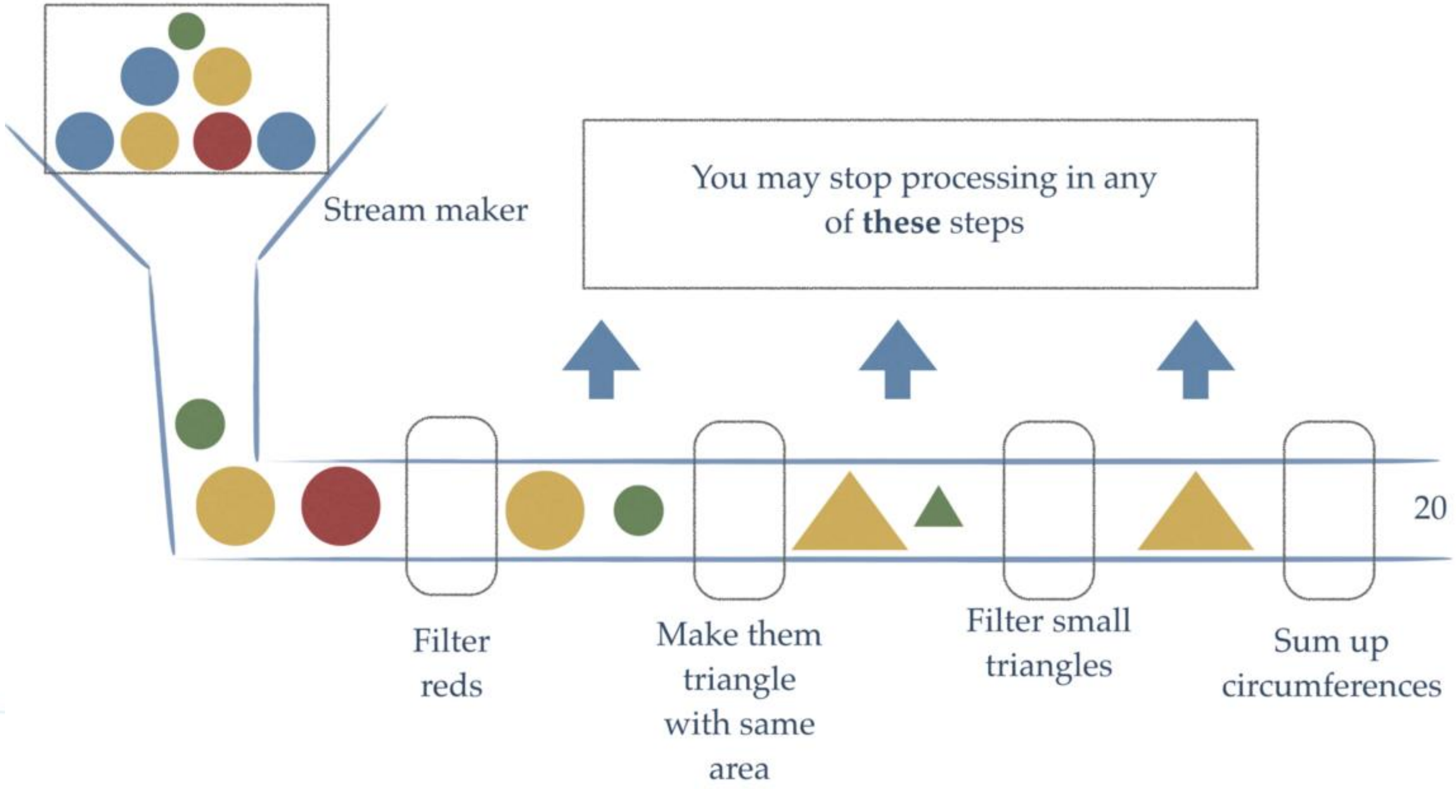
Stream processing paradigm simplifies parallel software and hardware by restricting the parallel computation that can be performed.

Given a sequence of data (***a stream***), a series of operations (***functions***) is applied to each element in the stream, in a declarative way, we specify **what** we want to achieve and **not how**.

Big Data Stream Processing: A gentle introduction

Stream processing paradigm simplifies parallel software and hardware by restricting the parallel computation that can be performed.

Given a sequence of <u>data</u> (***a stream***), a <u>series of operations</u> (***functions***) is applied to each element in the stream, in a declarative way, we specify **what** we want to achieve and **not how**.



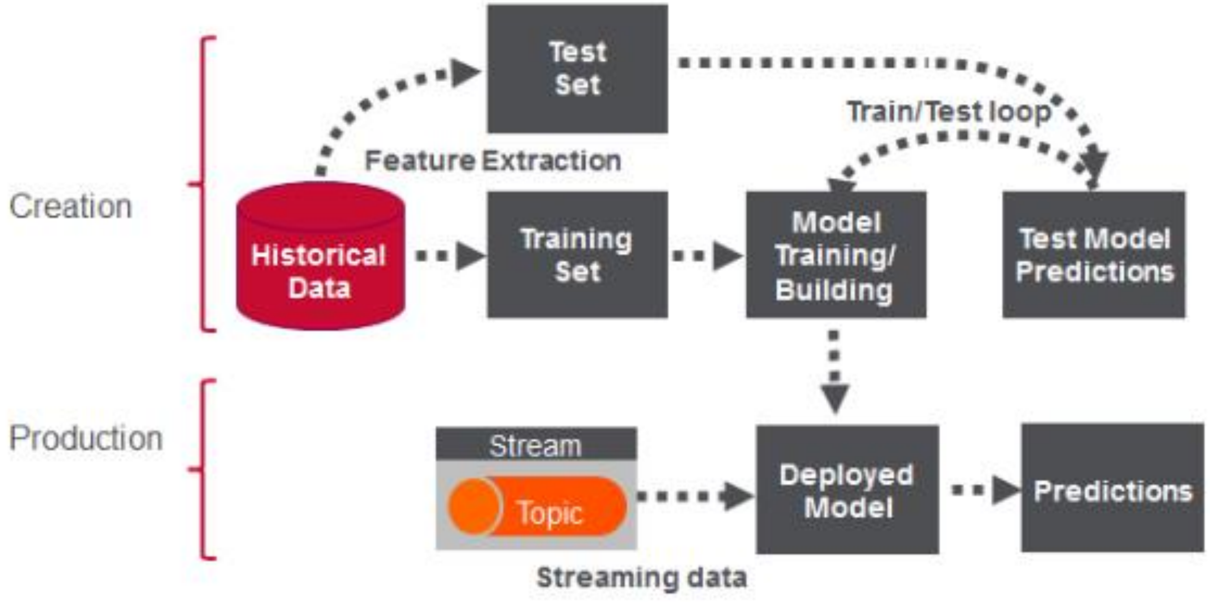
Big Data Stream Learning: Why is it different?

Big Data Stream Learning: Why is it different?

Big Data Stream Learning is more challenging than **batch** or **offline learning**, since the data may **not preserve the same distribution** over the lifetime of the stream.
Moreover, each example coming in a stream can only be **processed once**, or needs to be summarized with a **small memory footprint**, and the learning algorithms must be **efficient**.

Big Data Stream Learning: Why is it different?

Big Data Stream Learning is more challenging than **batch** or **offline learning**, since the data may **not preserve the same distribution** over the lifetime of the stream.
Moreover, each example coming in a stream can only be **processed once**, or needs to be summarized with a **small memory footprint**, and the learning algorithms must be **efficient**.



Big Data Stream Learning: Where's the catch?

Big Data Stream Learning: Where’s the catch?

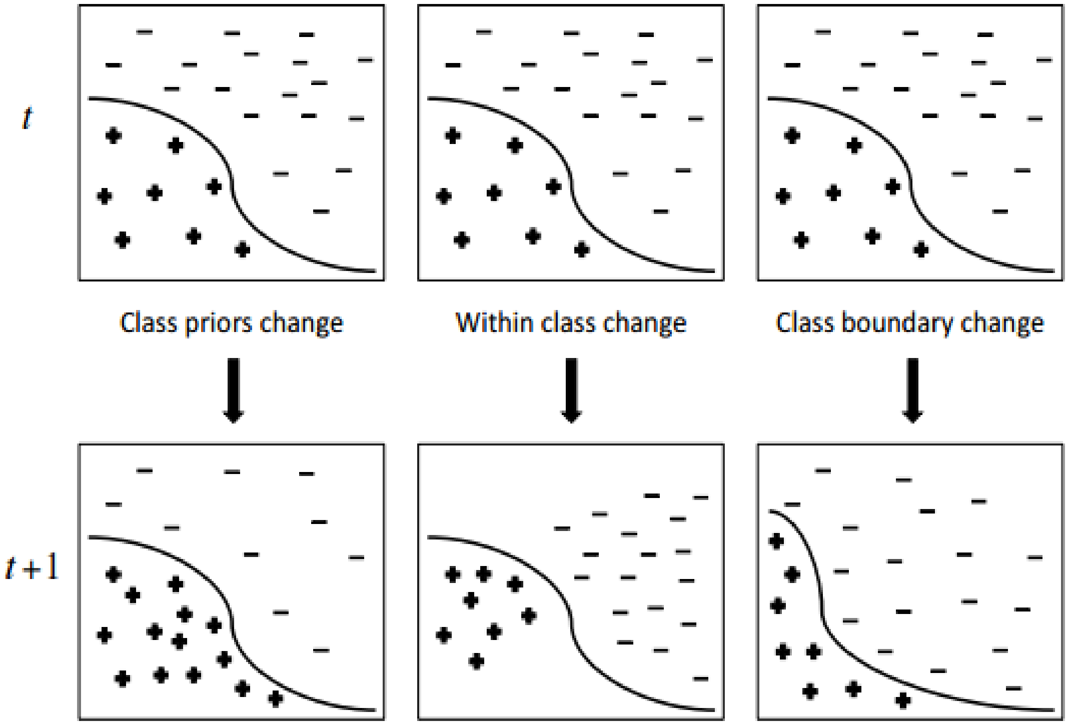
In order to deal with **evolving data streams**, the model learnt from the streaming data must **capture up-to-date trends and transient patterns** in the stream.

Updating the model by incorporating **new examples**, we must also **eliminate the effects of outdated examples** representing outdated concepts.

Big Data Stream Learning: Where's the catch?

In order to deal with **evolving data streams**, the model learnt from the streaming data must **capture up-to-date trends and transient patterns** in the stream.

Updating the model by incorporating **new examples**, we must also **eliminate the effects of outdated examples** representing outdated concepts.



The need for Streaming Machine Learning

The need for Streaming Machine Learning

How to **compute the entropy** of a collection of **infinite data**, where the **domain of the variables can be huge** and the **number of classes** of objects is **not known** a priori?

The need for Streaming Machine Learning

How to **compute the entropy** of a collection of **infinite data**, where the **domain of the variables can be huge** and the **number of classes** of objects is **not known** a priori?

How to **maintain the k-most frequent items** in a retail data warehouse with **3 TB of data**, **100s of GB of new sales records** updated daily with **1000000 of different items**?

The need for Streaming Machine Learning

How to **compute the entropy** of a collection of **infinite data**, where the **domain of the variables can be huge** and the **number of classes** of objects is **not known** a priori?

How to **maintain the k-most frequent items** in a retail data warehouse with **3 TB of data**, **100s of GB of new sales records** updated daily with **1000000s different items**?

What becomes of **statistical computations** when the learner can only afford **one pass through each data sample** because of **time** and **memory constraints**; when the learner has to **decide on-the-fly** what is **relevant** and **process** it and what is **redundant** and could be **discarded**?

Elements of Streaming Machine Learning

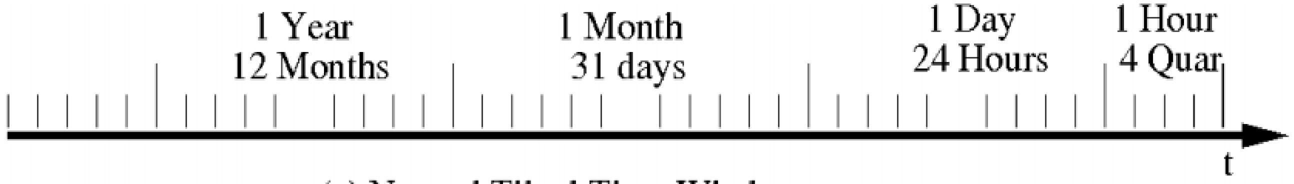
Elements of Streaming Machine Learning

Most strategies use variations of the **sliding window technique**: a window is maintained that keeps the most **recently read examples**, and from which **older examples are dropped** according to some **set of rules**.

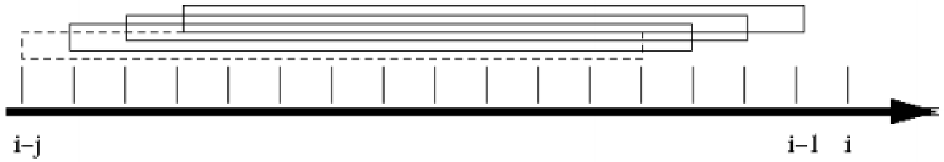
Elements of Streaming Machine Learning

Most strategies use variations of the **sliding window technique**: a window is maintained that keeps the most **recently read examples**, and from which **older examples are dropped** according to some **set of rules**.

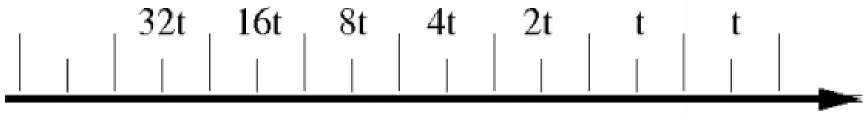
(a) Landmark Window



(a) Natural Tilted Time Window



(b) Sliding Window



b) Logarithmic Tilted Time Window

Elements of Streaming Machine Learning

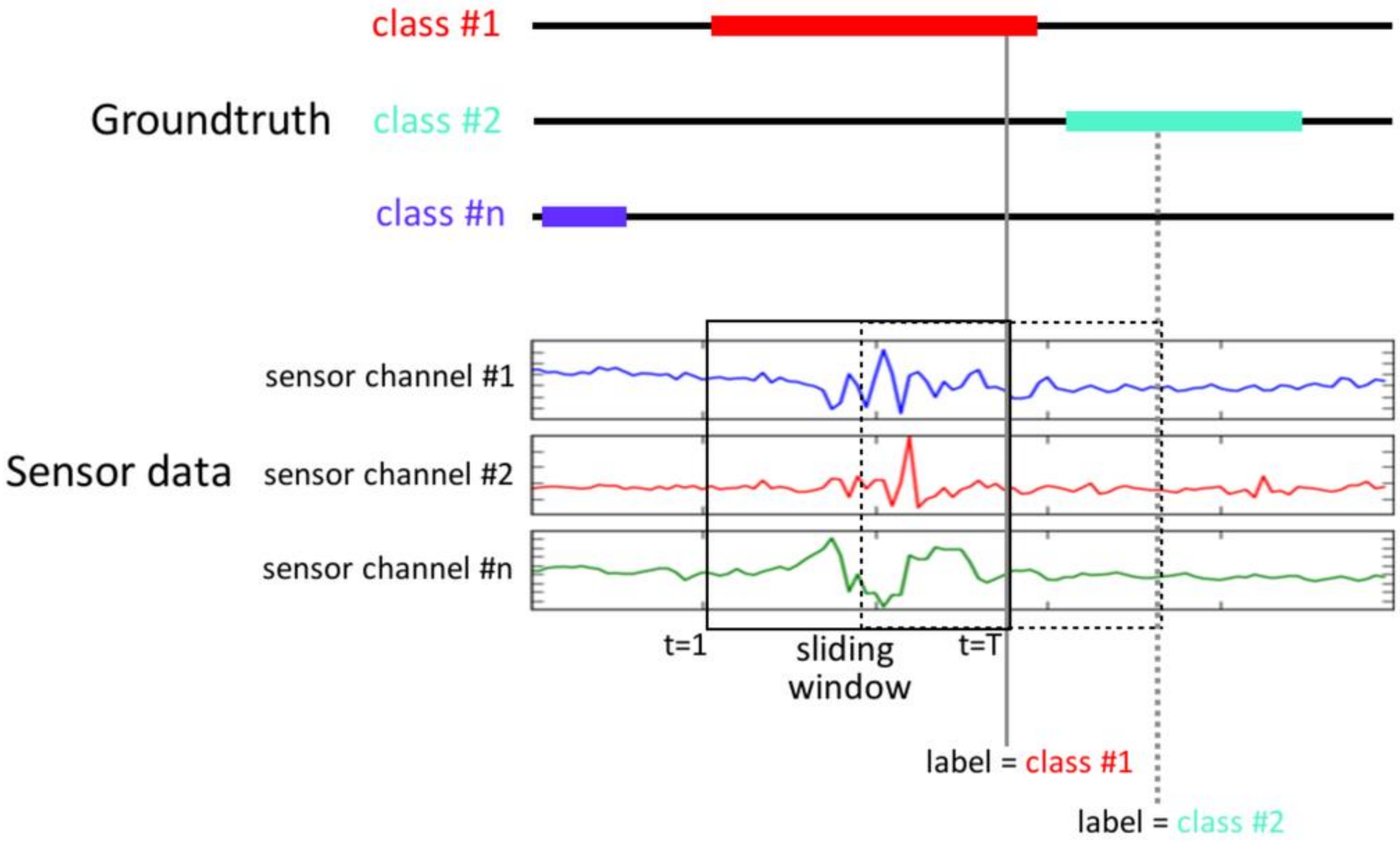
The contents of the **sliding window** can be used for the three tasks:

1) to **detect change** (e.g., by using some statistical test on different sub-windows),
2) to obtain **updated statistics / criteria** from the **recent examples**, and
3) to have **data** to **rebuild** or **update the model** after **data has changed**.

Elements of Streaming Machine Learning

The contents of the **sliding window** can be used for the three tasks:

1) to **detect change** (e.g., by using some statistical test on different sub-windows),
2) to obtain **updated statistics / criteria** from the **recent examples**, and
3) to have **data** to **rebuild** or **update the model** after **data has changed**.



Elements of Streaming Machine Learning

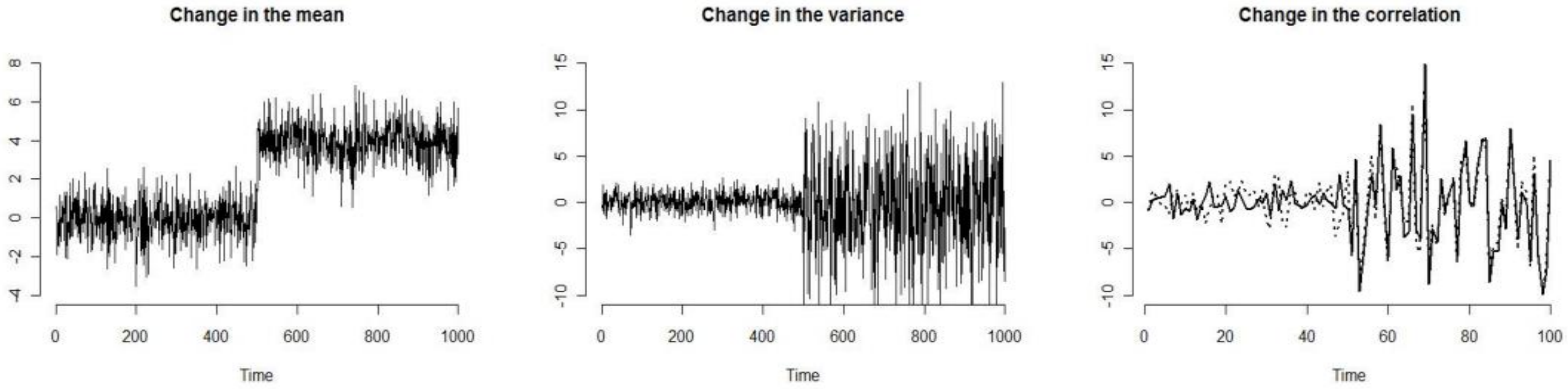
Normally, the user is caught in a **tradeoff without solution**:

- a **small size** (so that the **window reflects** accurately the current **distribution**)
- a **large size** (so that many **examples** are available to work on, **increasing accuracy** in periods of stability).

Elements of Streaming Machine Learning

Normally, the user is caught in a **tradeoff without solution**:

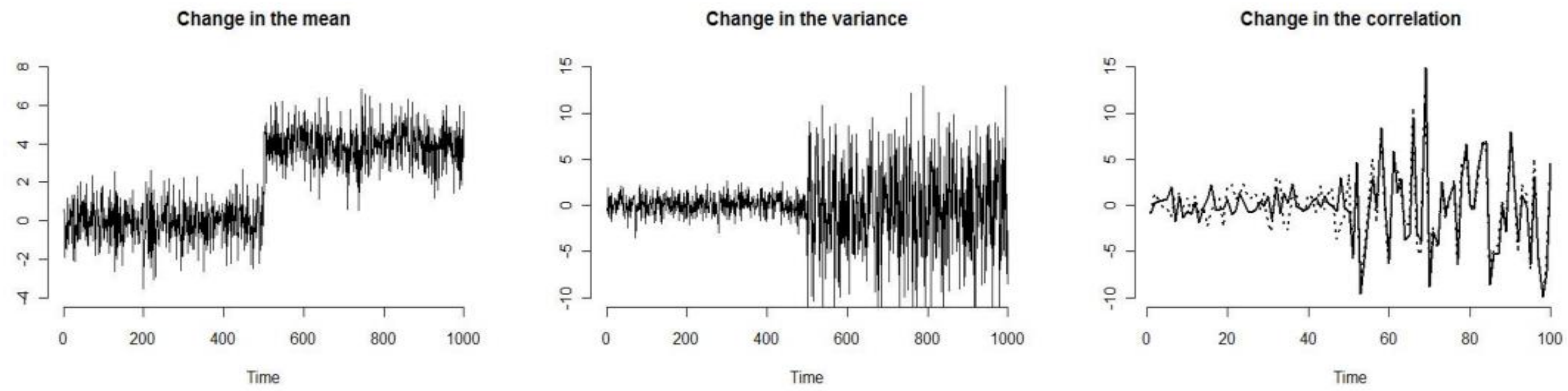
- a **small size** (so that the **window reflects** accurately the current **distribution**)
- a **large size** (so that many **examples** are available to work on, **increasing accuracy** in periods of stability).



Elements of Streaming Machine Learning

Normally, the user is caught in a **tradeoff without solution**:

- a **small size** (so that the **window reflects** accurately the current **distribution**)
- a **large size** (so that many **examples** are available to work on, **increasing accuracy** in periods of stability).

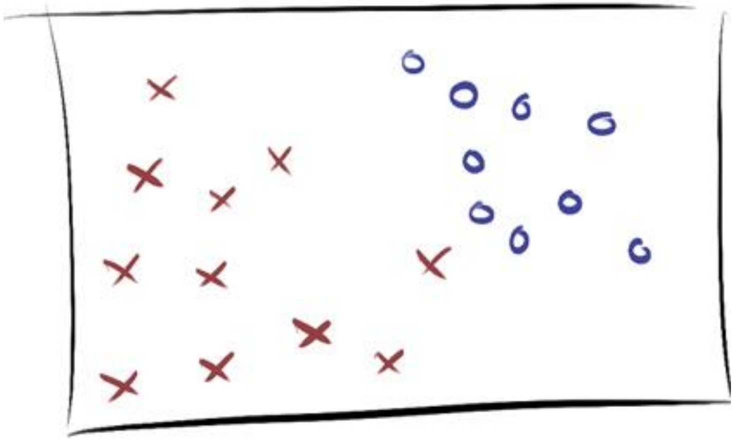


Currently, it has been proposed to use **windows** of **variable size**.

Streaming Machine Learning: *a view on classification*

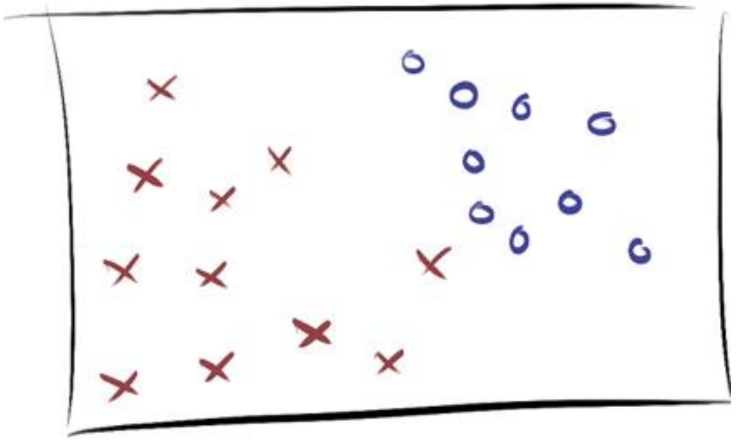
Streaming Machine Learning: *a view on classification*

Given a set of training examples belonging to n different classes, a classifier algorithm builds a model that predicts for every unlabeled instance x the class C to which it belongs.



Streaming Machine Learning: *a view on classification*

Given a set of training examples belonging to n different classes, a classifier algorithm builds a model that predicts for every unlabeled instance x the class C to which it belongs.

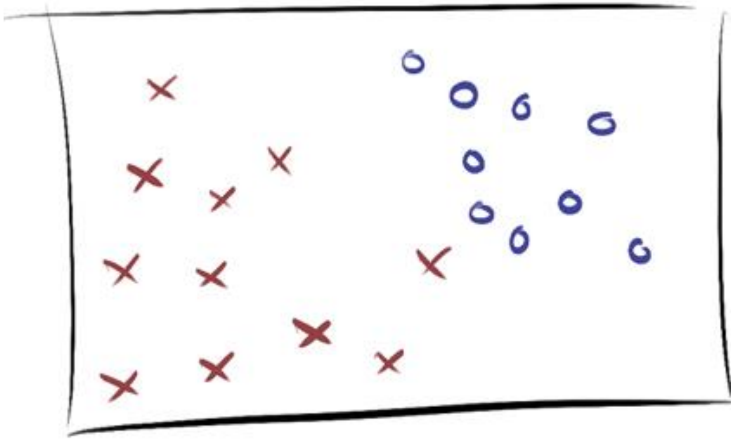


Streaming constraints:

1. One example at a time, used at most once
2. Limited memory
3. Limited time
4. Anytime prediction

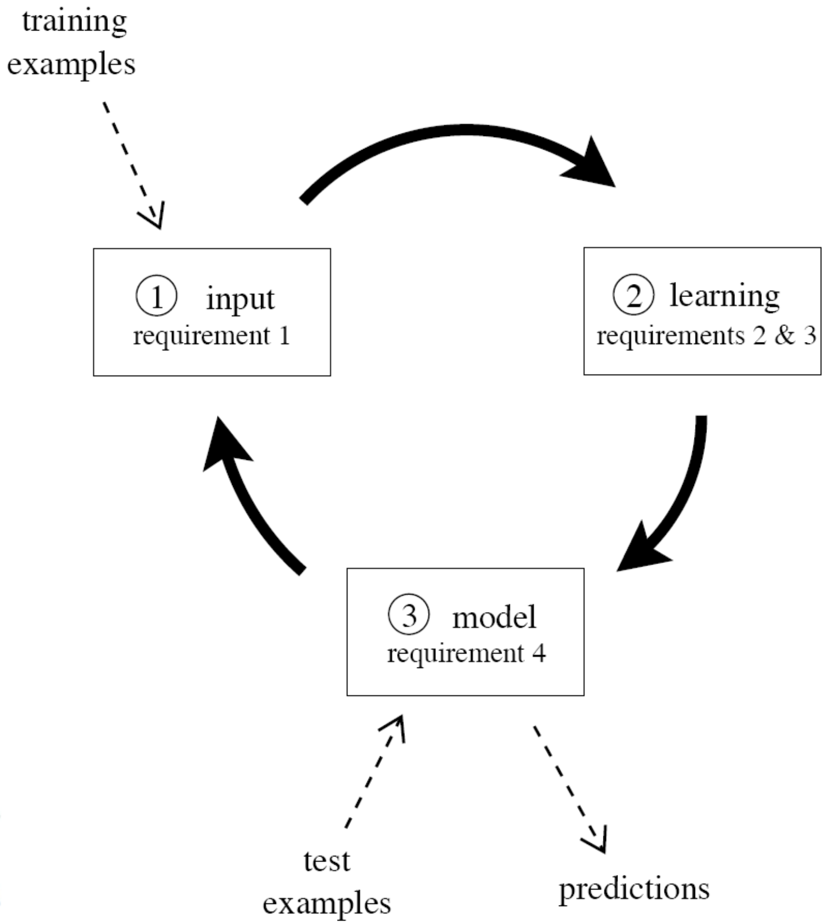
Streaming Machine Learning: *a view on classification*

Given a set of training examples belonging to n different classes, a classifier algorithm builds a model that predicts for every unlabeled instance x the class C to which it belongs.



Streaming constraints:

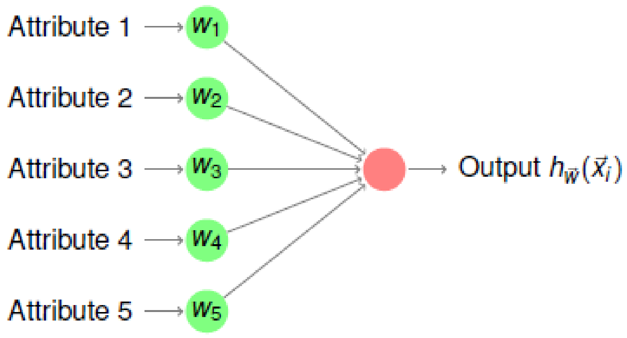
1. One example at a time, used at most once
2. Limited memory
3. Limited time
4. Anytime prediction



Streaming Machine Learning: *a view on classification*

Perceptron

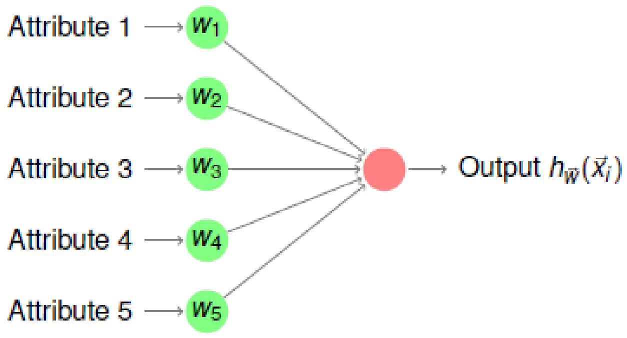
- Linear classifier
- Data stream: $\langle \vec{x}_i, y_i \rangle$
- $\tilde{y}_i = h_{\vec{w}}(\vec{x}_i) = \sigma(\vec{w}_i^T \vec{x}_i)$
- $\sigma(x) = 1/(1+e^{-x})$ $\sigma' = \sigma(x)(1-\sigma(x))$
- Minimize MSE $J(\vec{w}) = ½\sum(y_i - \tilde{y}_i)^2$
- SGD $\vec{w}_{i+1} = \vec{w}_i - \eta \nabla J \vec{x}_i$
  - $\nabla J = -(y_i - \tilde{y}_i)\tilde{y}_i(1-\tilde{y}_i)$
  - $\vec{w}_{i+1} = \vec{w}_i + \eta(y_i - \tilde{y}_i)\tilde{y}_i(1-\tilde{y}_i)\vec{x}_i$



Streaming Machine Learning: *a view on classification*

Perceptron

- Linear classifier
- Data stream: $\langle \vec{x}_i, y_i \rangle$
- $\tilde{y}_i = h_{\vec{w}}(\vec{x}_i) = \sigma(\vec{w}_i^T \vec{x}_i)$
- $\sigma(x) = 1/(1+e^{-x})$ $\sigma' = \sigma(x)(1-\sigma(x))$
- Minimize MSE $J(\vec{w}) = \frac{1}{2}\sum(y_i - \tilde{y}_i)^2$
- SGD $\vec{w}_{i+1} = \vec{w}_i - \eta \nabla J \vec{x}_i$
  - $\nabla J = -(y_i - \tilde{y}_i)\tilde{y}_i(1-\tilde{y}_i)$
  - $\vec{w}_{i+1} = \vec{w}_i + \eta(y_i - \tilde{y}_i)\tilde{y}_i(1-\tilde{y}_i)\vec{x}_i$



Perceptron Learning

Perceptron Learning($Stream, \eta$)

for each class
    do Perceptron Learning($Stream, class, \eta$)

Perceptron Learning($Stream, class, \eta$)

$\triangleright$ Let w_0 and $\vec{w}$ be randomly initialized
for each example $(\vec{x}, y)$ in Stream
    do if $class = y$
        then $\delta = (1 - h_{\vec{w}}(\vec{x})) \cdot h_{\vec{w}}(\vec{x}) \cdot (1 - h_{\vec{w}}(\vec{x}))$
        else $\delta = (0 - h_{\vec{w}}(\vec{x})) \cdot h_{\vec{w}}(\vec{x}) \cdot (1 - h_{\vec{w}}(\vec{x}))$
      $\vec{w} = \vec{w} + \eta \cdot \delta \cdot \vec{x}$

Perceptron Prediction($\vec{x}$)

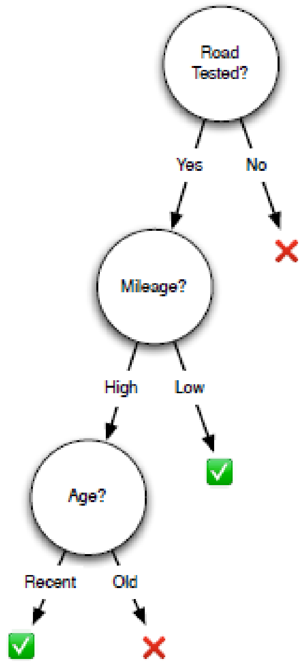
return $\arg\max_{class} h_{\vec{w}_{class}}(\vec{x})$

Streaming Machine Learning: *a view on classification*

Decision Tree

- Each node tests a features
- Each branch represents a value
- Each leaf assigns a class
- Greedy recursive induction
  - Sort all examples through tree
  - x_i = most discriminative attribute
  - New node for x_i, new branch for each value, leaf assigns majority class
  - Stop if no error | limit on #instances

Car deal?

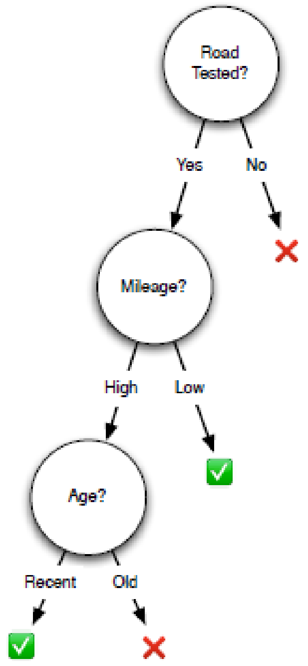


Streaming Machine Learning: *a view on classification*

Decision Tree

- Each node tests a features
- Each branch represents a value
- Each leaf assigns a class
- Greedy recursive induction
  - Sort all examples through tree
  - x_i = most discriminative attribute
  - New node for x_i, new branch for each value, leaf assigns majority class
  - Stop if no error | limit on #instances

Car deal?



Very Fast Decision Tree

- AKA, Hoeffding Tree
- A small sample can often be enough to choose a near optimal decision
- Collect sufficient statistics from a small set of examples
- Estimate the merit of each alternative attribute
- Choose the sample size that allows to differentiate between the alternatives

Streaming Machine Learning: *a view on classification*

Leaf Expansion

- When should we expand a leaf?
- Let x_1 be the most informative attribute, x_2 the second most informative one
- Is x_1 a stable option?
- Hoeffding bound
  - Split if $G(x_1) - G(x_2) > \varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$

Streaming Machine Learning: *a view on classification*

Leaf Expansion

- When should we expand a leaf?
- Let x_1 be the most informative attribute, x_2 the second most informative one
- Is x_1 a stable option?
- Hoeffding bound
  - Split if $G(x_1) - G(x_2) > \varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$

HT Induction

$\text{HT}(\textit{Stream}, \delta)$

▷ Let HT be a tree with a single leaf(root)
▷ Init counts n_{ijk} at root
for each example (x, y) in Stream
    do $\text{HTGROW}((x, y), HT, \delta)$

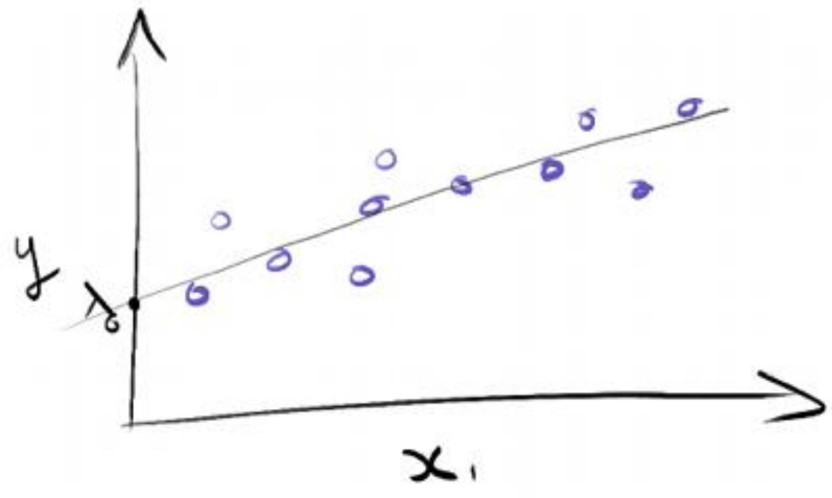
$\text{HTGROW}((x, y), HT, \delta)$

▷ Sort (x, y) to leaf l using HT
▷ Update counts n_{ijk} at leaf l
if examples seen so far at l are not all of the same class
    then ▷ Compute G for each attribute
        if $G(\text{Best Attr.}) - G(\text{2nd best}) > \sqrt{\frac{R^2 \ln 1/\delta}{2n}}$
            then ▷ Split leaf on best attribute
                for each branch
                    do ▷ Start new leaf and initiliatize counts

Streaming Machine Learning: *a view on regression*

Streaming Machine Learning: *a view on regression*

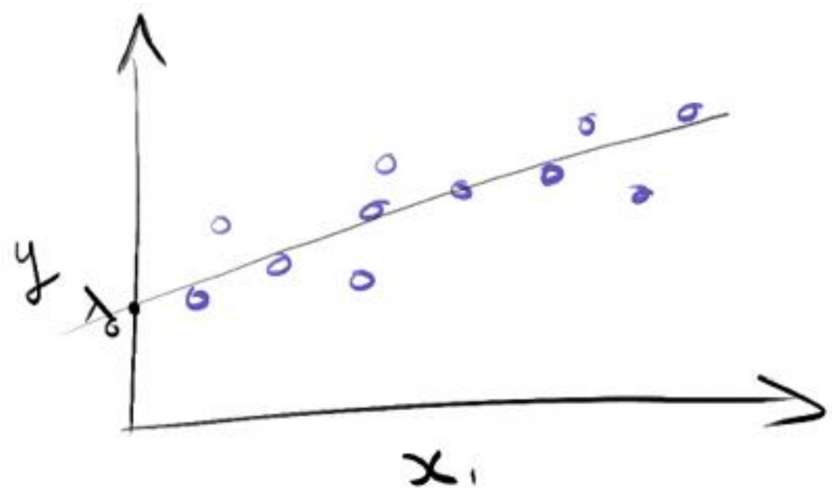
Given a set of training examples with a numeric label, a regression algorithm builds a model that predicts for every unlabeled instance x the value y=f(x) with high accuracy.



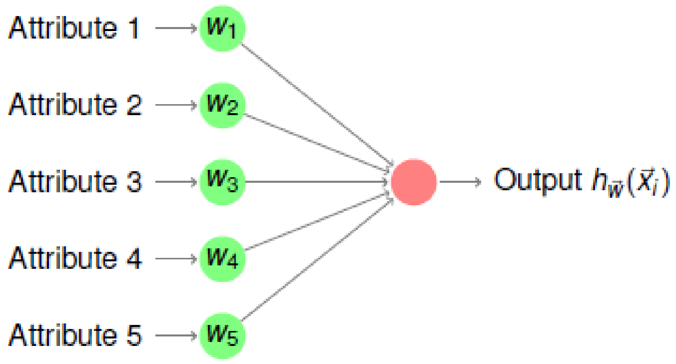
Streaming Machine Learning: *a view on regression*

Given a set of training examples with a numeric label, a regression algorithm builds a model that predicts for every unlabeled instance x the value $y=f(x)$ with high accuracy.

Perceptron



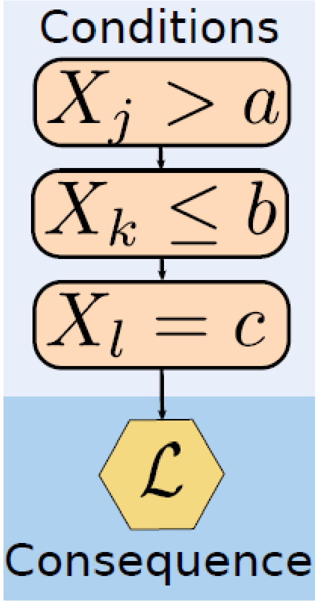
- Linear regressor
- Data stream: $\langle \vec{x}_i, y_i \rangle$
- $\tilde{y}_i = h_{\vec{w}}(\vec{x}_i) = \vec{w}^T \vec{x}_i$
- Minimize MSE $J(\vec{w}) = \frac{1}{2}\sum(y_i - \tilde{y}_i)^2$
- SGD $\vec{w}' = \vec{w} - \eta \nabla J \vec{x}_i$
  - $\nabla J = -(y_i - \tilde{y}_i)$
  - $\vec{w}' = \vec{w} + \eta(y_i - \tilde{y}_i)\vec{x}_i$



Streaming Machine Learning: *a view on regression*

Rules

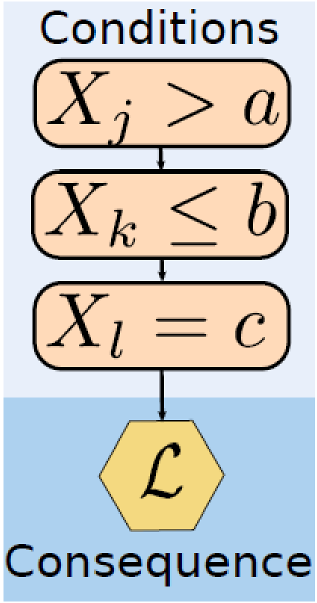
- Problem: very large decision trees have context that is complex and hard to understand
- Rules: self-contained, modular, easier to interpret, no need to cover universe
- $\mathcal{L}$ keeps sufficient statistics to:
  - make predictions
  - expand the rule
  - detect changes and anomalies



Streaming Machine Learning: *a view on regression*

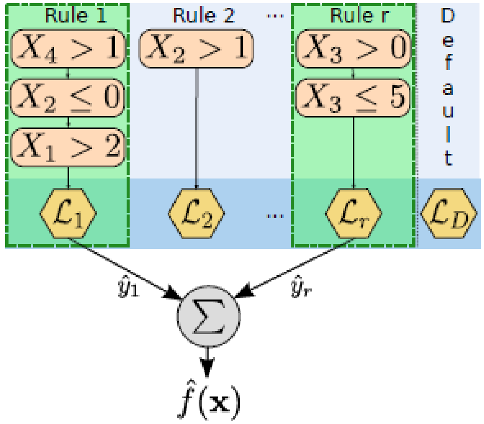
Rules

- Problem: very large decision trees have context that is complex and hard to understand
- Rules: self-contained, modular, easier to interpret, no need to cover universe
- $\mathcal{L}$ keeps sufficient statistics to:
  - make predictions
  - expand the rule
  - detect changes and anomalies



Adaptive Model Rules

- Ruleset: ensemble of rules
- Rule prediction: mean, linear model
- Ruleset prediction
  - Weighted avg. of predictions of rules covering instance x
  - Weights inversely proportional to error
  - Default rule covers uncovered instances



E.g: $\mathbf{x} = [4, -1, 1, 2]$

$$\hat{f}(\mathbf{x}) = \sum_{R_l \in S(\mathbf{x}_i)} \theta_l \hat{y}_l,$$

Stream Processing Engines

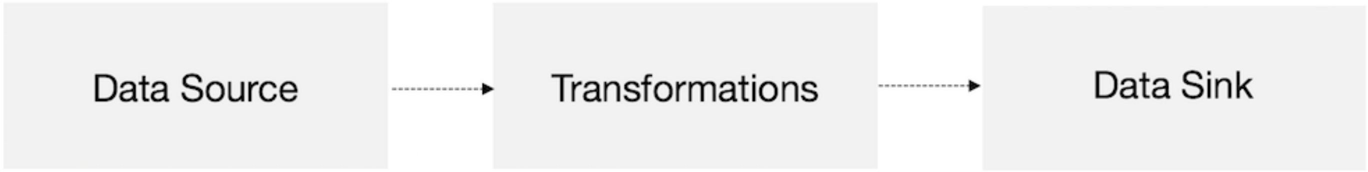
Stream Processing Engines

Flink

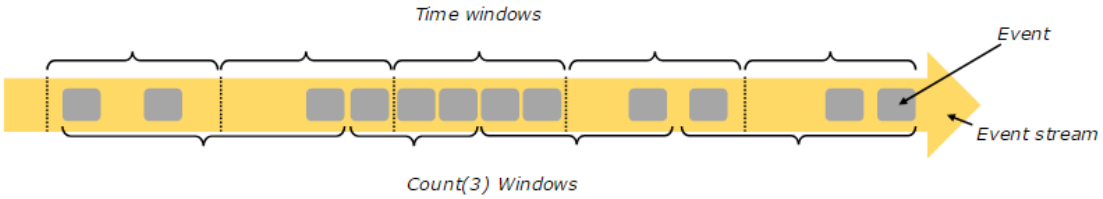
Data Source → Transformations → Data Sink

Stream Processing Engines

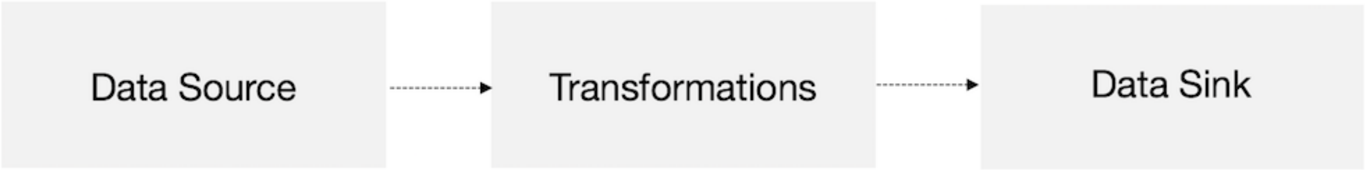
Flink



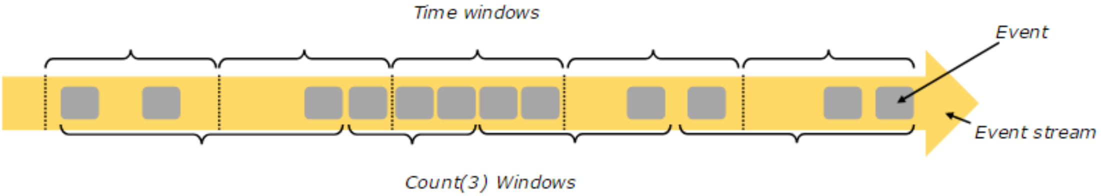
Aggregating events (e.g., counts, sums) works differently on streams because it is **impossible to count** all (unbounded).



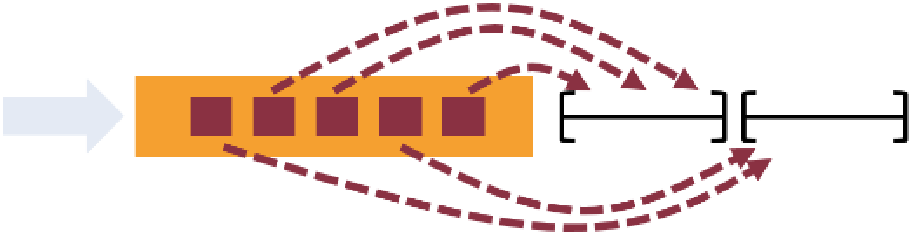
Stream Processing Engines



Aggregating events (e.g., counts, sums) works differently on streams because it is **impossible to count** all (unbounded).

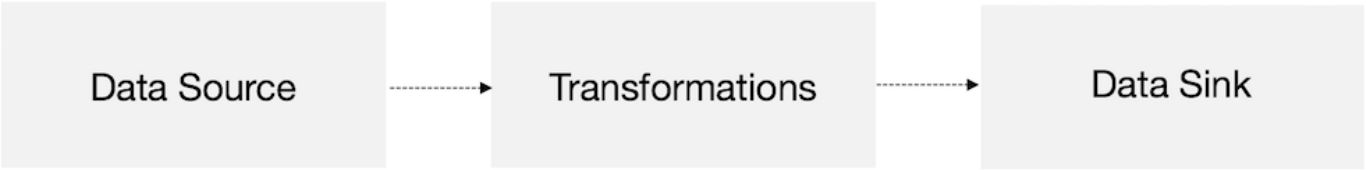


Stream processing and **windowing** makes it easy to compute accurate results over streams where **events** arrive **out of order** and where **events may arrive delayed**.

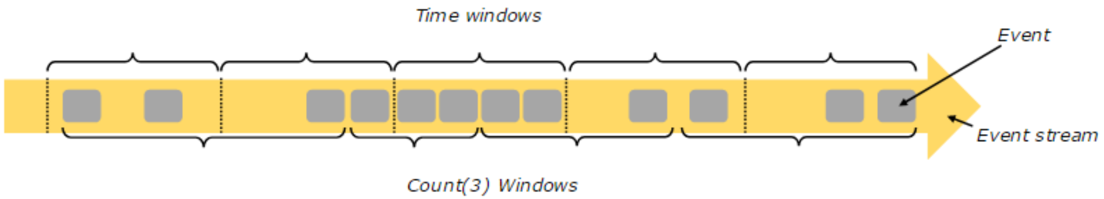


Stream Processing Engines

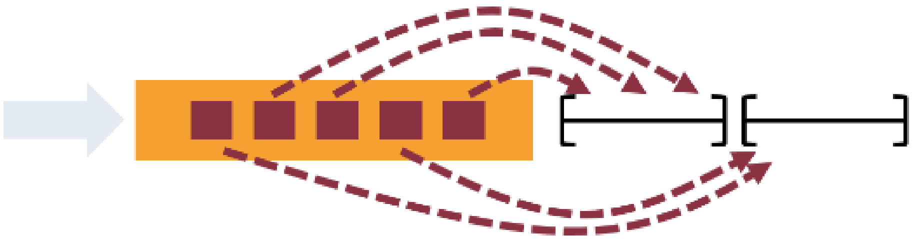
Flink



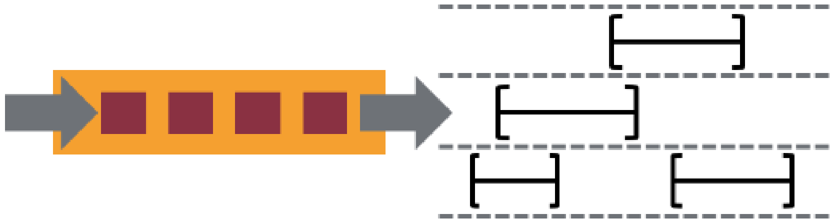
Aggregating events (e.g., counts, sums) works differently on streams because it is **impossible to count** all (unbounded).



Stream processing and **windowing** makes it easy to compute accurate results over streams where **events** arrive **out of order** and where **events may arrive delayed**.



Windowing based on time, count, and data-driven windows. Windows can be customized with **flexible triggering** conditions to support **sophisticated streaming patterns**.



Stream Processing Engines

Flink

When executed, Flink programs are mapped to **streaming dataflows**, consisting of **streams** and **transformation operators**.

Each **dataflow** starts with one or more **sources** and ends in one or more **sinks**.

The dataflows resemble arbitrary directed acyclic graphs (DAGs).

Stream Processing Engines

Flink

```
DataStream<String> lines = env.addSource(
                                new FlinkKafkaConsumer<>(…));

DataStream<Event> events = lines.map((line) -> parse(line));

DataStream<Statistics> stats = events
        .keyBy("id")
        .timeWindow(Time.seconds(10))
        .apply(new MyWindowAggregationFunction());

stats.addSink(new RollingSink(path));
```

Source

Transformatior

Transformatior

Sink

When executed, Flink programs are mapped to **streaming dataflows**, consisting of **streams** and **transformation operators**.

Each **dataflow** starts with one or more **sources** and ends in one or more **sinks**.

The dataflows resemble arbitrary directed acyclic graphs (DAGs).

Stream Processing Engines

Flink

When executed, Flink programs are mapped to **streaming dataflows**, consisting of **streams** and **transformation operators**.

Each **dataflow** starts with one or more **sources** and ends in one or more **sinks**.

The dataflows resemble arbitrary directed acyclic graphs (DAGs).

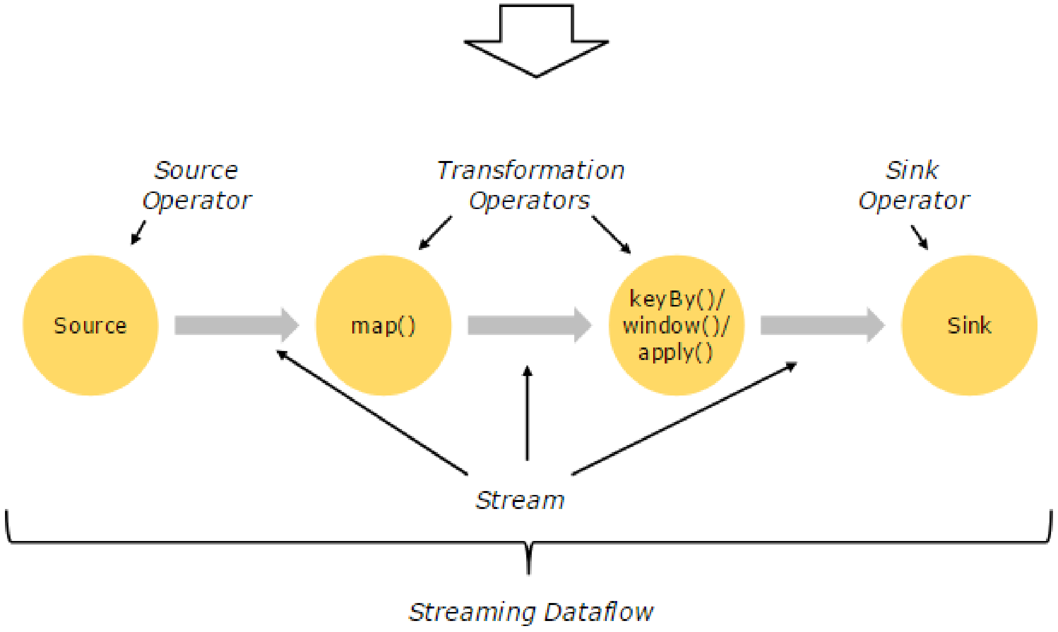
```
DataStream<String> lines = env.addSource(
                                new FlinkKafkaConsumer<>(…));

DataStream<Event> events = lines.map((line) -> parse(line));

DataStream<Statistics> stats = events
        .keyBy("id")
        .timeWindow(Time.seconds(10))
        .apply(new MyWindowAggregationFunction());

stats.addSink(new RollingSink(path));
```

Source

Transformatior

Transformatior

Sink



Stream Processing Engines

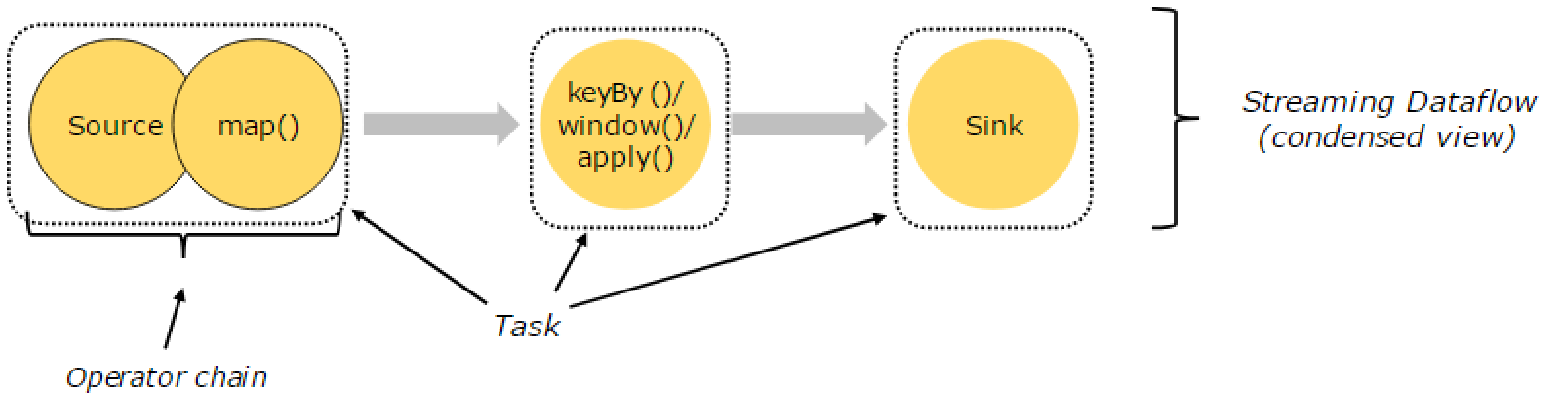
Flink

For distributed execution, Flink chains operator **subtasks** together into **tasks**. Each task is executed by one **thread**.

Stream Processing Engines

Flink

For distributed execution, Flink chains operator **subtasks** together into **tasks**. Each task is executed by one **thread**.

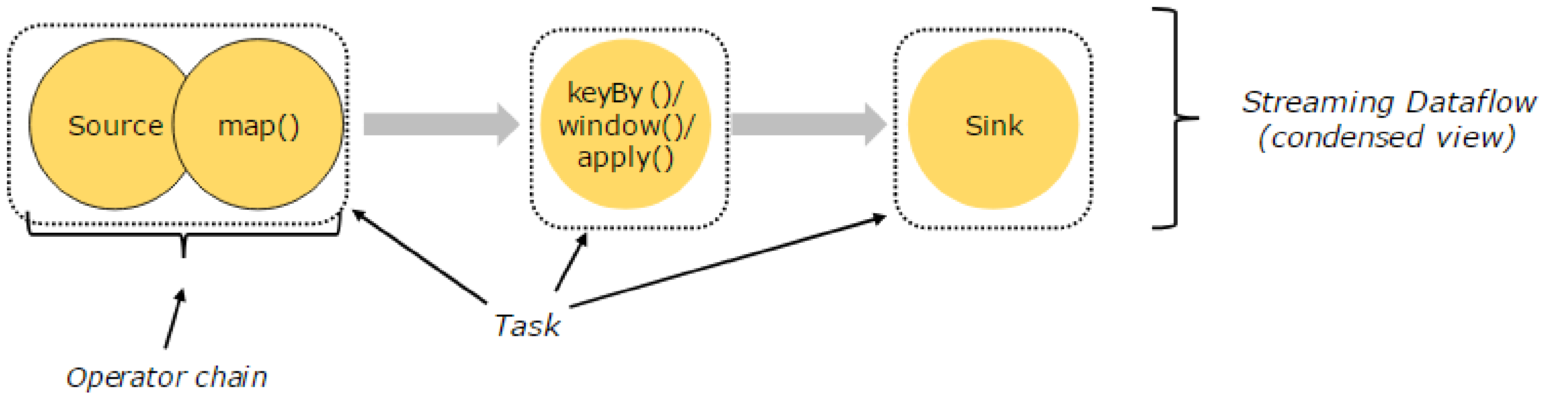


Stream Processing Engines

Flink

For distributed execution, Flink chains operator **subtasks** together into **tasks**. Each task is executed by one **thread**.

Chaining operators together into tasks is a useful optimization: it **reduces** the **overhead** of thread-to-thread handover and **buffering**, and **increases overall throughput** while **decreasing latency**.

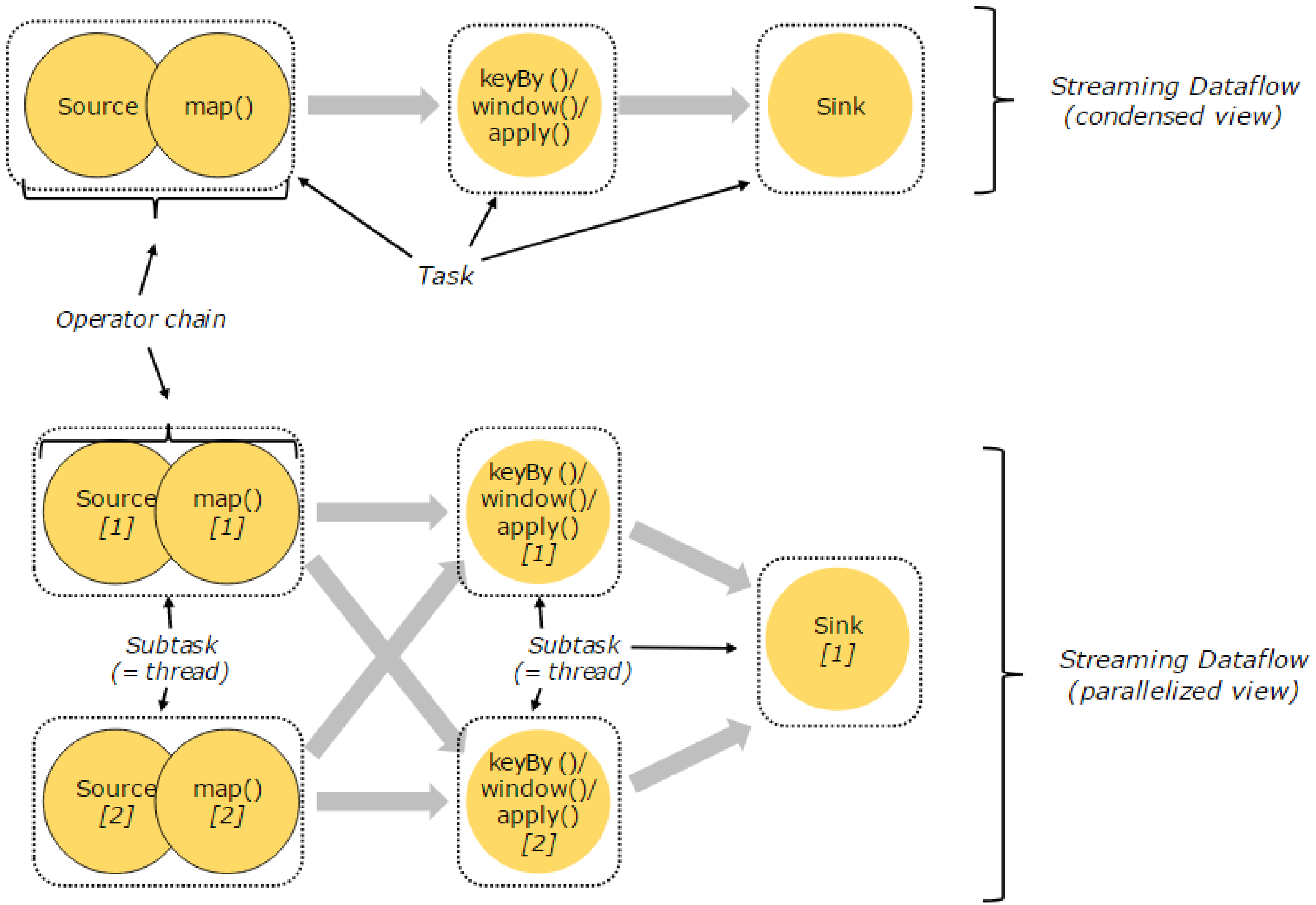


Stream Processing Engines

Flink

For distributed execution, Flink chains operator **subtasks** together into **tasks**. Each task is executed by one **thread**.

Chaining operators together into tasks is a useful optimization: it **reduces** the **overhead** of thread-to-thread handover and **buffering**, and **increases overall throughput** while **decreasing latency**.

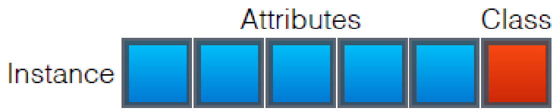


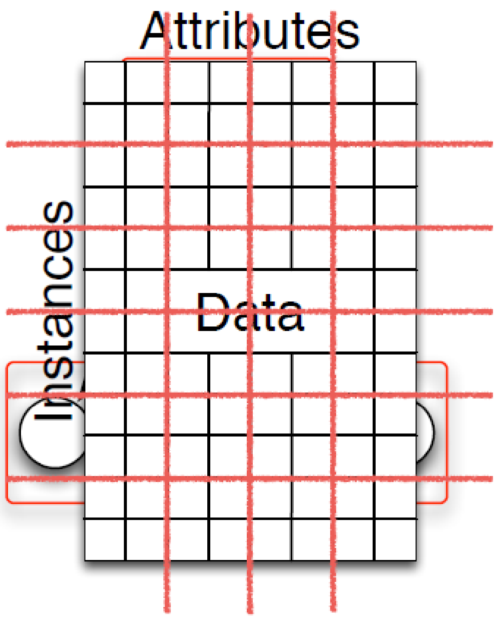
Online Distributed Streaming Machine Learning: *a streamed view on classification*

Online Distributed Streaming Machine Learning: *a streamed view on classification*

Parallel Decision Trees

- Which kind of parallelism?
  - Task
  - Data
    - Horizontal
    - Vertical

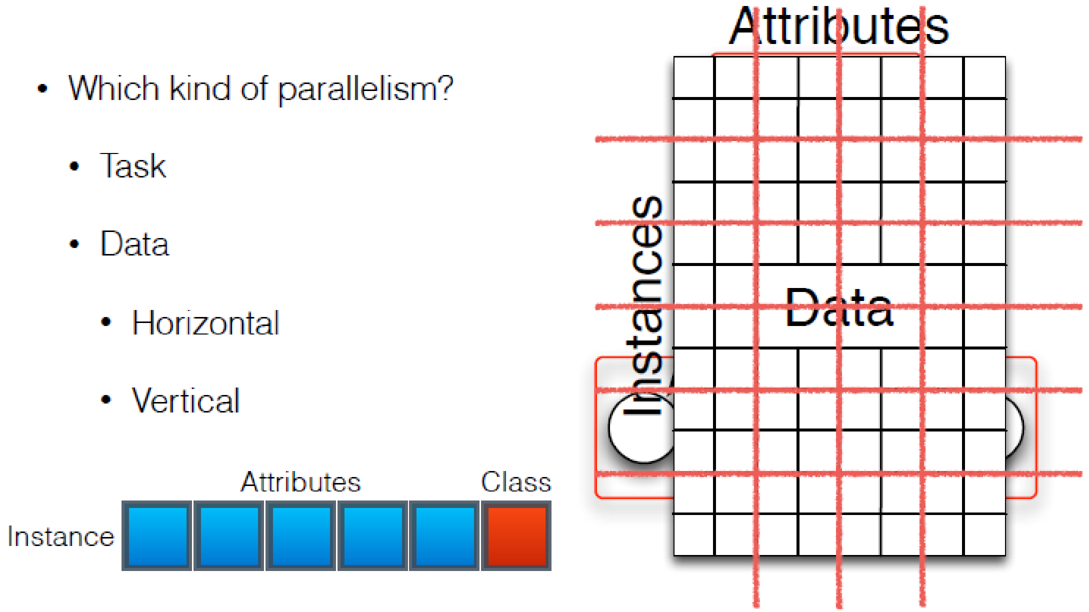




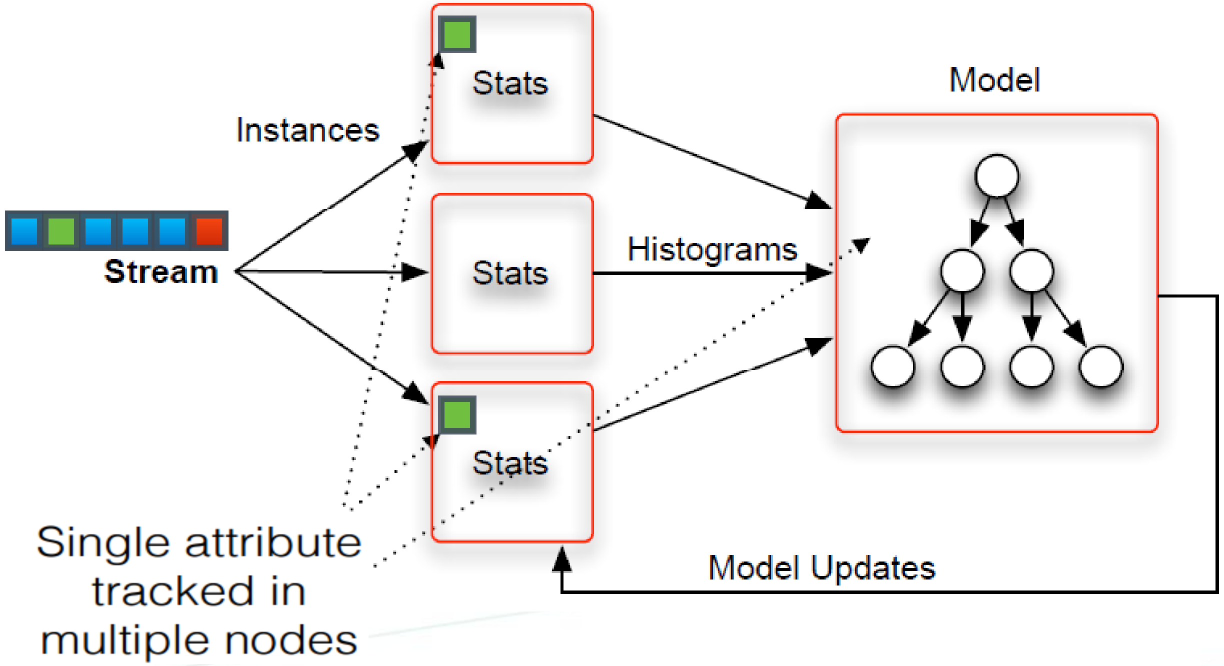
Online Distributed Streaming Machine Learning:
a streamed view on classification

Parallel Decision Trees

- Which kind of parallelism?
  - Task
  - Data
    - Horizontal
    - Vertical

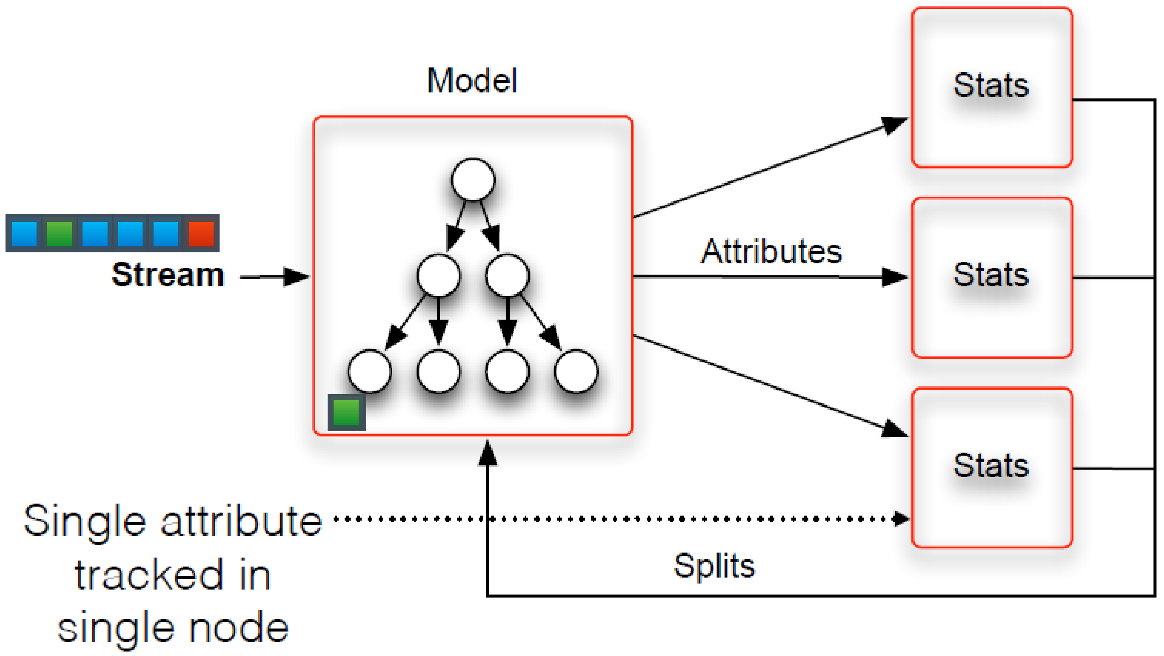


Horizontal Partitioning



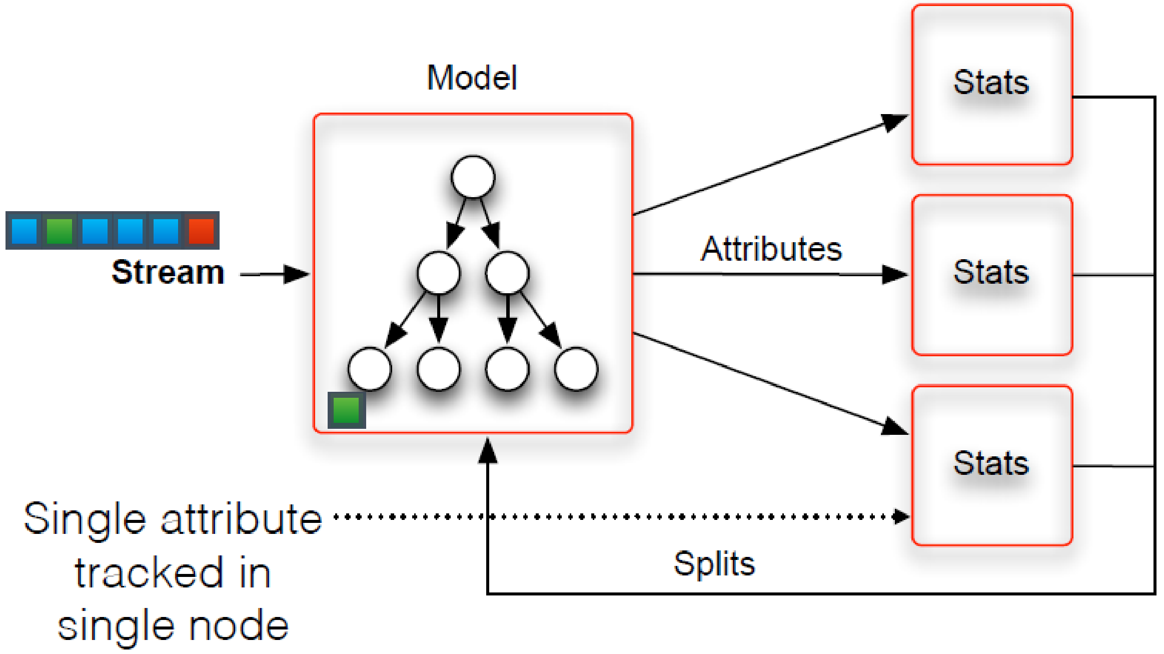
Online Distributed Streaming Machine Learning:
a streamed view on classification

Vertical Partitioning



Online Distributed Streaming Machine Learning: *a streamed view on classification*

Vertical Partitioning



Advantages of Vertical Parallelism

- High number of attributes => high level of parallelism (e.g., documents)
- vs. task parallelism
  - Parallelism observed immediately
- vs. horizontal parallelism
  - Reduced memory usage (no model replication)
  - Parallelized split computation

Online Distributed Streaming Machine Learning:
a streamed view on regression

Online Distributed Streaming Machine Learning: *a streamed view on regression*

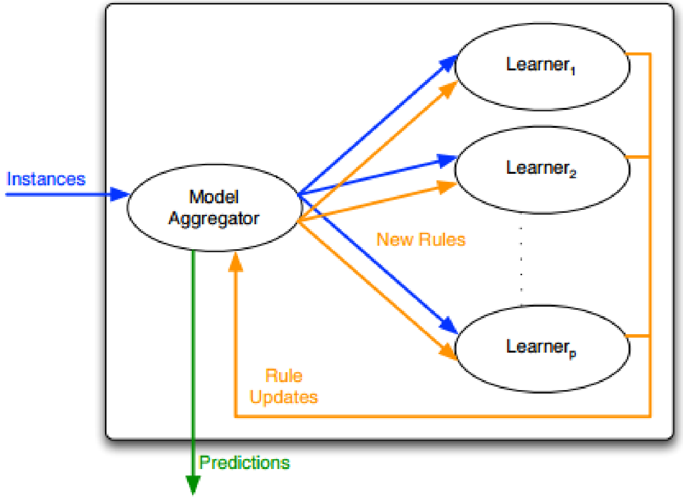
VAMR

- Vertical AMRules
- Model: rule body + head
  - Target mean updated continuously with covered instances for predictions
  - Default rule (creates new rules)
- Learner: statistics
  - Vertical: Learner tracks statistics of independent subset of rules
  - One rule tracked by only one Learner
  - Model -> Learner: key grouping on rule ID

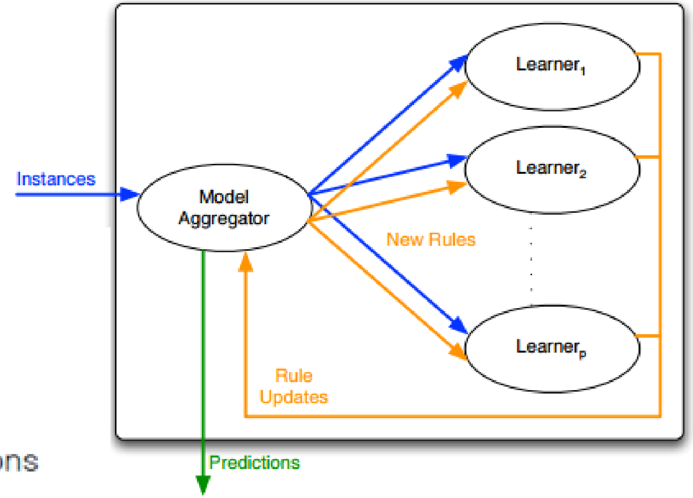
Online Distributed Streaming Machine Learning: *a streamed view on regression*

VAMR

- Vertical AMRules
- Model: rule body + head
  - Target mean updated continuously with covered instances for predicti
  - Default rule (creates new rules)
- Learner: statistics
  - Vertical: Learner tracks statistics of independent subset of rules
  - One rule tracked by only one Learner
  - Model -> Learner: key grouping on rule ID



Online Distributed Streaming Machine Learning: *a streamed view on regression*



VAMR

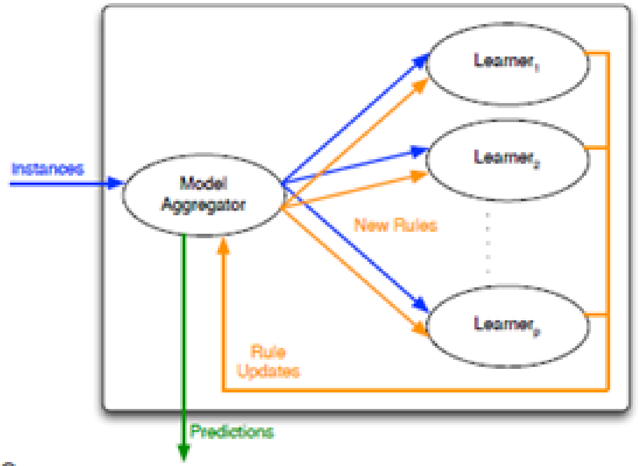
- Vertical AMRules
- Model: rule body + head
  - Target mean updated continuously with covered instances for predictions
  - Default rule (creates new rules)
- Learner: statistics
  - Vertical: Learner tracks statistics of independent subset of rules
  - One rule tracked by only one Learner
  - Model -> Learner: key grouping on rule ID

HAMR

- VAMR single model is bottleneck
- Hybrid AMRules (Vertical + Horizontal)
  - Shuffle among multiple Models for parallelism
- Problem: distributed default rule decreases performance
  - Separate dedicate Learner for default rule

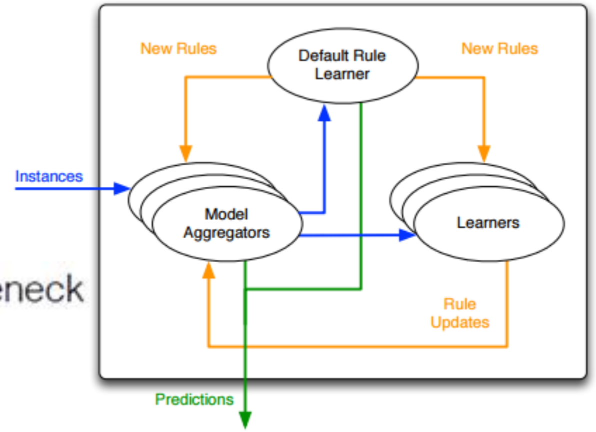
Online Distributed Streaming Machine Learning:
a streamed view on regression

VAMR



- Vertical AMRules
- Model: rule body + head
  - Target mean updated continuously with covered instances for predictions
  - Default rule (creates new rules)
- Learner: statistics
  - Vertical: Learner tracks statistics of independent subset of rules
  - One rule tracked by only one Learner
  - Model -> Learner: key grouping on rule ID

HAMR



- VAMR single model is bottleneck
- Hybrid AMRules (Vertical + Horizontal)
  - Shuffle among multiple Models for parallelism
- Problem: distributed default rule decreases performance
  - Separate dedicate Learner for default rule

Conclusions

Conclusions

Induced by ubiquitous scenarios *finite training sets*, *static models*, and *stationary distributions* must be completely redefined.

Conclusions

Induced by ubiquitous scenarios *finite training sets*, *static models*, and *stationary distributions* must be completely redefined.

The characteristics of the streaming data entail a new vision due to the fact that:

Conclusions

Induced by ubiquitous scenarios *finite training sets*, *static models*, and *stationary distributions* must be completely redefined.

The characteristics of the streaming data entail a new vision due to the fact that:

- Data are made available through **unlimited streams** that **continuously flow**, eventually at **high speed**, over time;

Conclusions

Induced by ubiquitous scenarios *finite training sets*, *static models*, and *stationary distributions* must be completely redefined.

The characteristics of the streaming data entail a new vision due to the fact that:

- Data are made available through **unlimited streams** that **continuously flow**, eventually at **high speed**, over time;
- The **underlying regularities may evolve** over time rather than being stationary;

Conclusions

Induced by ubiquitous scenarios *finite training sets*, *static models*, and *stationary distributions* must be completely redefined.

The characteristics of the streaming data entail a new vision due to the fact that:

- Data are made available through **unlimited streams** that **continuously flow**, eventually at **high speed**, over time;
- The **underlying regularities may evolve** over time rather than being stationary;
- The data can **no longer** be considered as **independent and identically distributed**;

Conclusions

Induced by ubiquitous scenarios *finite training sets*, *static models*, and *stationary distributions* must be completely redefined.

The characteristics of the streaming data entail a new vision due to the fact that:

- Data are made available through **unlimited streams** that **continuously flow**, eventually at **high speed**, over time;
- The **underlying regularities may evolve** over time rather than being stationary;
- The data can **no longer** be considered as **independent and identically distributed**;
- The **data** are now often **spatially** as well as **time situated**.

Conclusions

Induced by ubiquitous scenarios *finite training sets*, *static models*, and *stationary distributions* must be completely redefined.

The characteristics of the streaming data entail a new vision due to the fact that:

- Data are made available through **unlimited streams** that **continuously flow**, eventually at **high speed**, over time;
- The **underlying regularities may evolve** over time rather than being stationary;
- The data can **no longer** be considered as **independent and identically distributed**;
- The **data** are now often **spatially** as well as **time situated**.

A new era of machine learning?

Thank You.