



INTRODUCTION TO THE STREAM MINING (SM) COURSE

Stream mining (SM)

Imre Lendák, PhD, Associate Professor

Péter Kiss, PhD candidate

Outline

- About the course
- Draft list of lecture topics & references
- Streaming intro



Stream mining

ABOUT THE COURSE

About the course

Theory and labs

- Lectures according to schedule
 - Initially on Teams only
 - PDF slides posted on Canvas
- Occasional assignments during the semester for extra points
- Course team:
 - Imre Lendák, Associate Professor
 - Péter Kiss, PhD candidate

Exam

- 60% for the projects
 - 5-member teams
 - Joint project for the OST and SM courses
- 40% for the theory
 - Oral examination
 - Defended project is entry criteria
- Note: both exam elements are obligatory

Stream mining

LECTURE LIST (DRAFT)

Lectures list (draft from 2019)

Part I: Stream mining intro

- Introduction, i.e. these slides
- Basics of stream mining
- Basics of stream mining
- Concept drift

Part II: Stream analysis

- Clustering
- Frequent pattern mining
- Novelty detection
- Time series
- Distributed stream mining

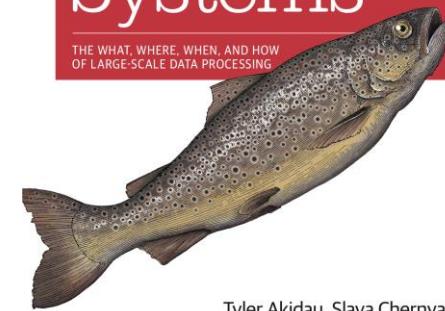
References

- Akidau T., Chernyak S., Lax R. Streaming Systems: The What, Where, When, and how of Large-scale Data Processing, O'Reilly Media, 2018.
 - <http://streamingsystems.net>
 - Streaming 101 & 102: The world beyond batch
- Gama, J. Knowledge discovery from data streams. Chapman and Hall/CRC, 2010.
- Aggarwal, C. C., ed. Data streams: models and algorithms. Vol. 31. Springer Science & Business Media, 2007.

O'REILLY

Streaming Systems

THE WHAT, WHERE, WHEN, AND HOW
OF LARGE-SCALE DATA PROCESSING



Tyler Akidau, Slava Chernyak
& Reuven Lax

Stream mining

INTRO TO STREAMING

Why streaming?

- Businesses need timely (i.e. immediate) insights into their data
- Massive, unbounded datasets are increasingly common in different business domains
- Processing data as it arrives spreads workloads more evenly over time → consistent and predictable consumption of computing resources (e.g. if we rent cloud-based resources)
 - This is the opposite of hoarding large amounts of data and periodically running high CPU/memory use analyses

Kinds of streams



Click streams



Sensor measurements



Satellite imaging data



Power grid electricity distribution



Banking/e-commerce transactions



Security monitoring data

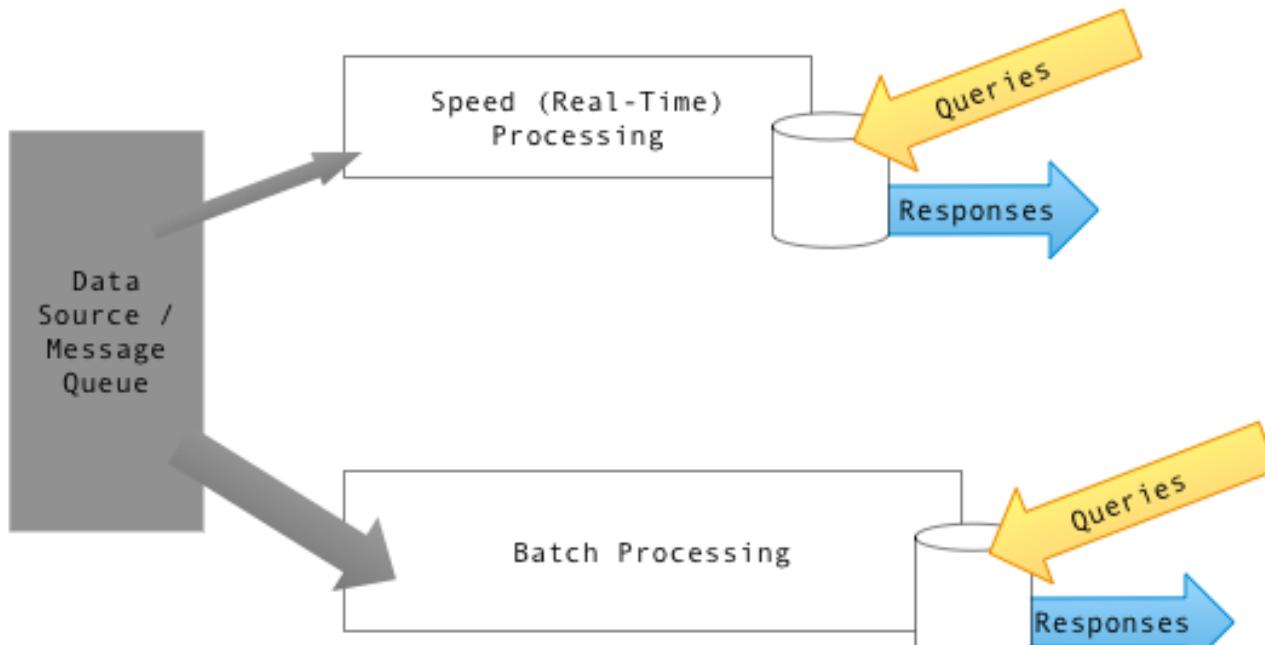
Necessary definitions

- Dataset cardinality
 - DEF: **Bounded data** is finite in size.
 - DEF: **Unbounded data** is infinite in size.
- Data constitution
 - DEF: A **table** represents a holistic (~complete) view of a dataset at a specific point in time.
 - DEF: A **stream** is an element-by-element view of the evolution of a dataset over time.
 - Alternate definition: A data stream is an ordered (not necessarily always) and potentially infinite sequence of data points (e.g. numbers, words, sequences).
- DEF: A **streaming system** is a data processing engine designed for handling infinite (unbounded) datasets.

Traditional streaming

- Characteristics of traditional streaming systems:
 - The good: low latency
 - The bad: inaccurate, i.e. lack of consistency → non-deterministic
- **DEF:** Batch systems are deterministic as they provide eventually correct results, i.e. once all relevant data is acquired and analyzed

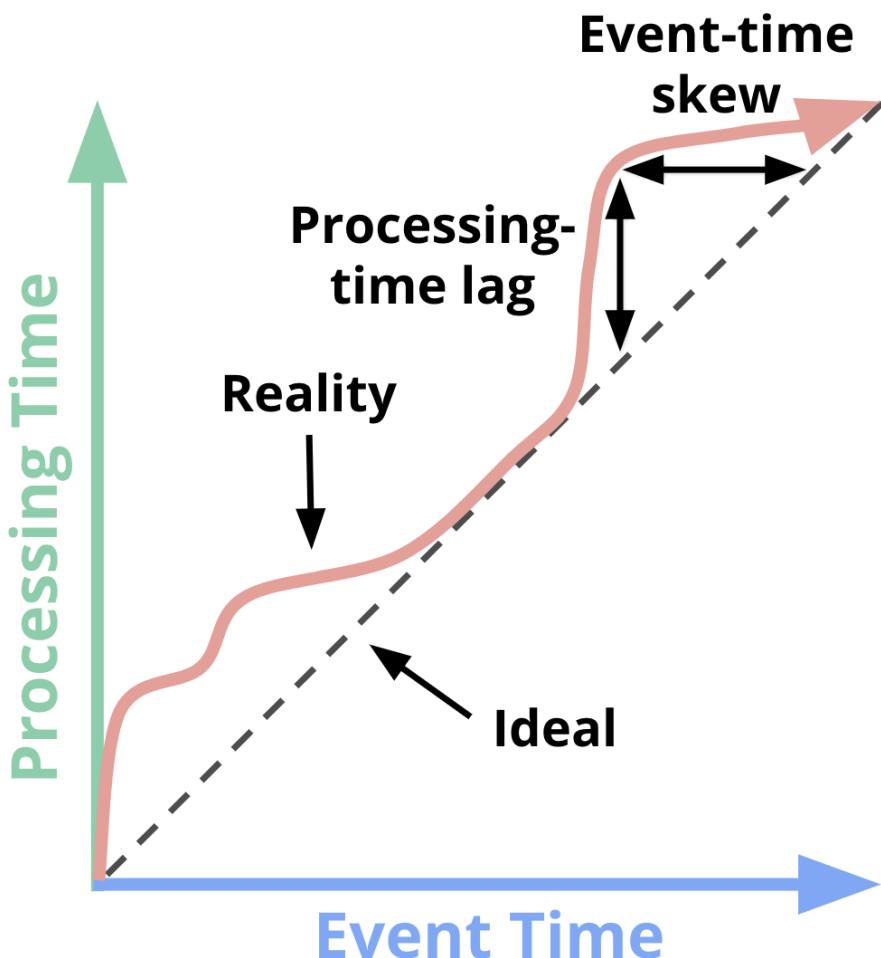
Lambda architecture



https://en.wikipedia.org/wiki/Lambda_architecture

- Lambda architecture: running a traditional streaming system in parallel with a batch data analysis solution
 - The good: low-latency (though inaccurate) results from the streaming element, correct results from the batch subsystem
 - The bad: hassle to implement and maintain (2 systems!)

Event time vs processing time



- X axis → event-time completeness in the system → the time X in event time up to which all data with event times less than X have been observed.
- Y axis → the progress of processing time → normal clock time as observed by the data processing system as it executes.

<http://streamingsystems.net/fig/1-1>

‘Modern’ streaming

1. Correctness
 - Consistent storage
 - Exactly-once processing
2. Reasoning about time
 - Techniques for reasoning about time in the presence of unbounded, unordered data of varying event time skew

Up next...

- Streaming basics
- Concept drift
- Stream analysis



Thank you for your attention!



STREAMING BASICS

Stream mining (SM)

Imre Lendák, PhD, Associate Professor

Péter Kiss, PhD candidate

Outline

- Streaming basics
- Reasoning about time
- Batch data processing 101
- Stream processing 101
- Triggers
- Watermarks
- Accumulation



Stream mining

STREAMING BASICS

Why streaming?

- Businesses need timely (i.e. immediate) insights into their data
- Massive, unbounded datasets are increasingly common in different business domains
- Processing data as it arrives spreads workloads more evenly over time → consistent and predictable consumption of computing resources (e.g. if we rent cloud-based resources)
 - This is the opposite of hoarding large amounts of data and periodically running high CPU/memory use analyses

Kinds of streams



Click streams



Sensor measurements



Satellite imaging data



Power grid electricity distribution



Banking/e-commerce transactions



Security monitoring data

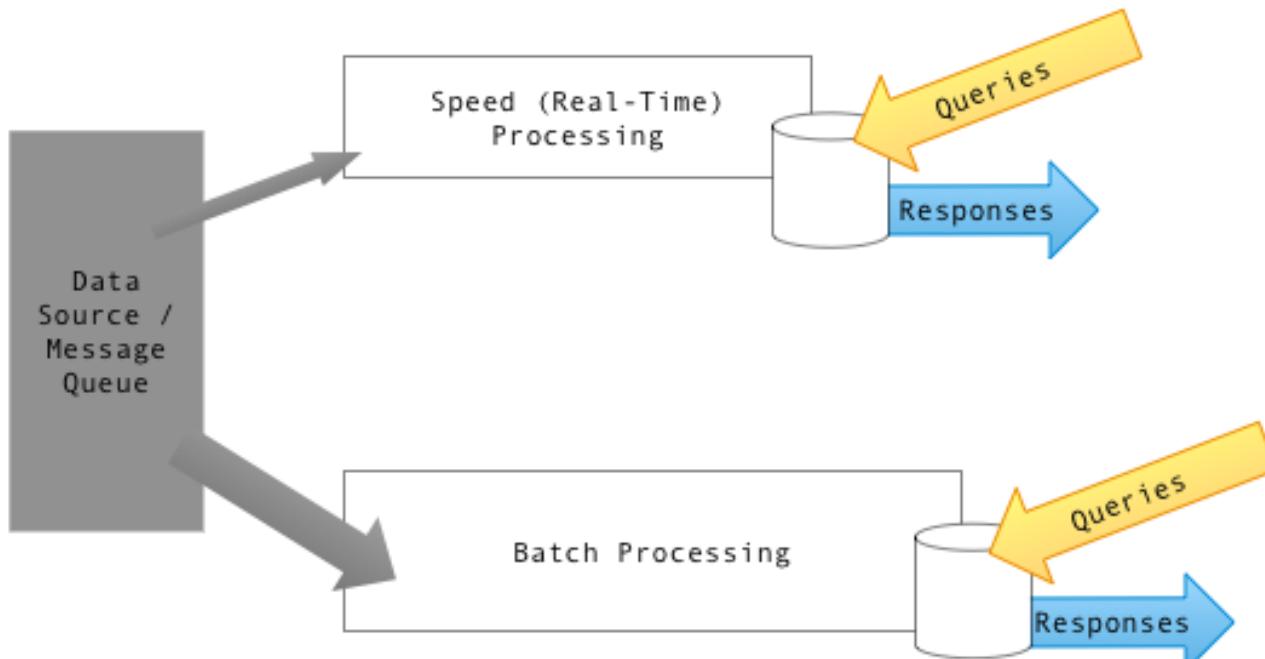
Necessary definitions

- Dataset cardinality
 - DEF: **Bounded data** is finite in size.
 - DEF: **Unbounded data** is infinite in size.
- Data constitution
 - DEF: A **table** represents a holistic (~complete) view of a dataset at a specific point in time.
 - DEF: A **stream** is an element-by-element view of the evolution of a dataset over time.
 - Alternate definition: A data stream is an ordered (not necessarily always) and potentially infinite sequence of data points (e.g. numbers, words, sequences).
- DEF: A **streaming system** is a data processing engine designed for handling infinite (unbounded) datasets.

Traditional streaming

- Characteristics of traditional streaming systems:
 - The good: low latency
 - The bad: inaccurate, i.e. lack of consistency → non-deterministic
- **DEF:** Batch systems are deterministic as they provide eventually correct results, i.e. once all relevant data is acquired and analyzed

Lambda architecture



https://en.wikipedia.org/wiki/Lambda_architecture

- Lambda architecture: running a traditional streaming system **in parallel** with a batch data analysis solution
 - The good: low-latency (though inaccurate) results from the streaming element, correct results from the batch subsystem
 - The bad: hassle to implement and maintain (2 systems!)

‘Modern’ streaming

1. Correctness

- Consistent storage
- Exactly-once processing

2. Reasoning about time

- Techniques for reasoning about time in the presence of unbounded, unordered data of varying event time skew

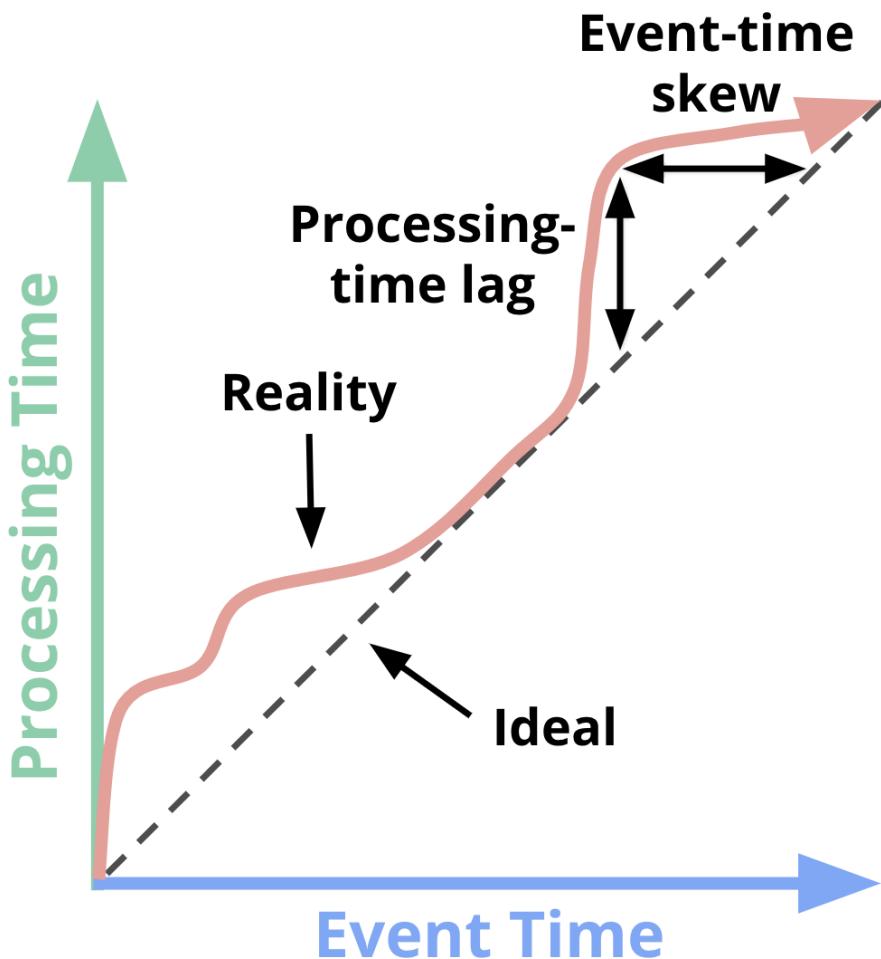
Stream mining

REASONING ABOUT TIME

Processing vs event time

- **Processing time** = The time at which events are observed by the system
- **Event time** = The time at which events actually occurred
- In non-ideal systems there can be a lag between processing and event times caused by a temporary lack of network connectivity, e.g. mobile device in airplane mode, maintenance crew doing repairs of power lines in remote (geographic) locations
- In industrial control systems and general purpose supervisory control and data acquisition systems (SCADA) event time is sometimes called ‘server’ time and processing time is ‘client’ time

Event time vs processing time



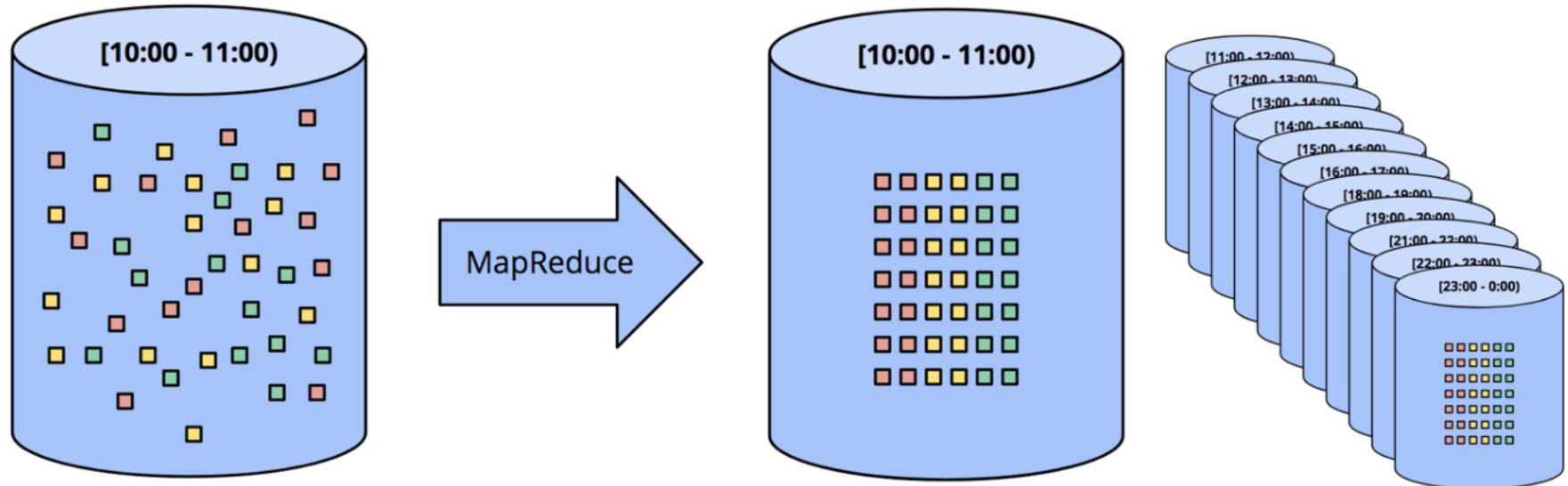
- X axis → event-time completeness in the system → the time X in event time up to which all data with event times less than X have been observed.
- Y axis → the progress of processing time → normal clock time as observed by the data processing system as it executes.

<http://streamingsystems.net/fig/1-1>

Stream mining

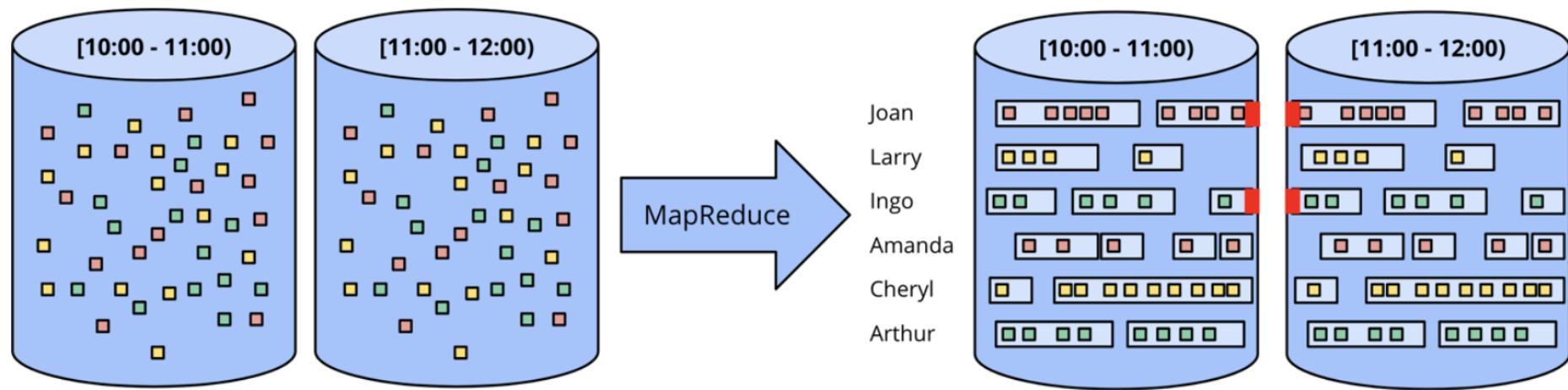
BATCH DATA PROCESSING 101

Fixed windows (in processing time)



<http://streamingsystems.net/fig/1-3>

Sessions



Stream mining

STREAM PROCESSING 101

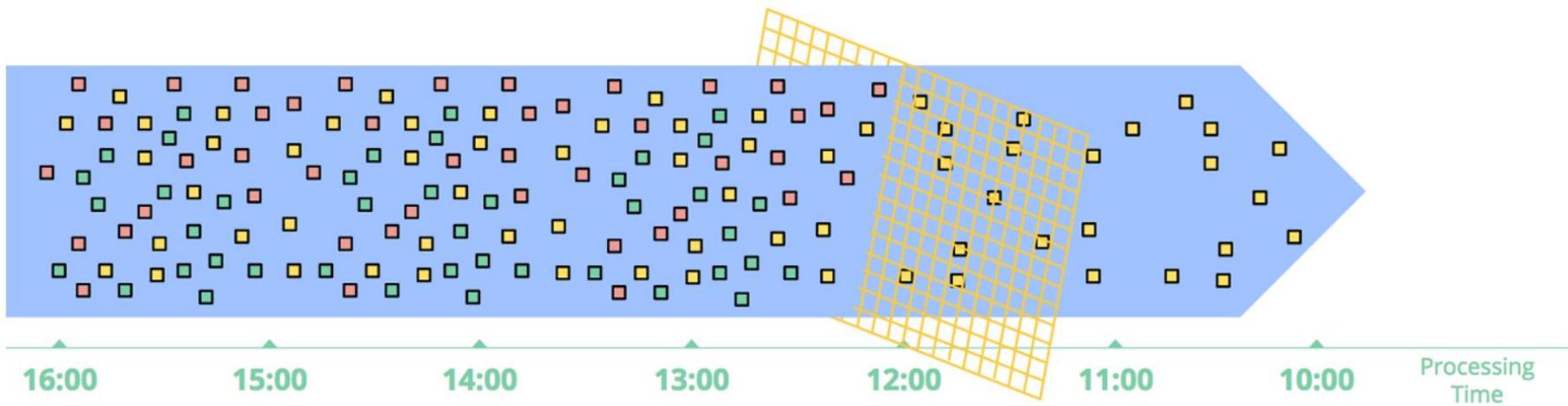
Stream data characteristics

- Unbounded → potentially infinite incoming data
- Highly unordered in respect to event time → it is necessary to ‘shuffle’ data in the processing pipeline
- Varying event time skew → we cannot assume that most data with event time X will arrive within constant epsilon time Y

Potential processing solutions

- Depending on the specific requirements of the data processor, we differentiate the following approaches to stream processing
 - Time-agnostic
 - Approximation
 - Windowing by processing time
 - Windowing by event time

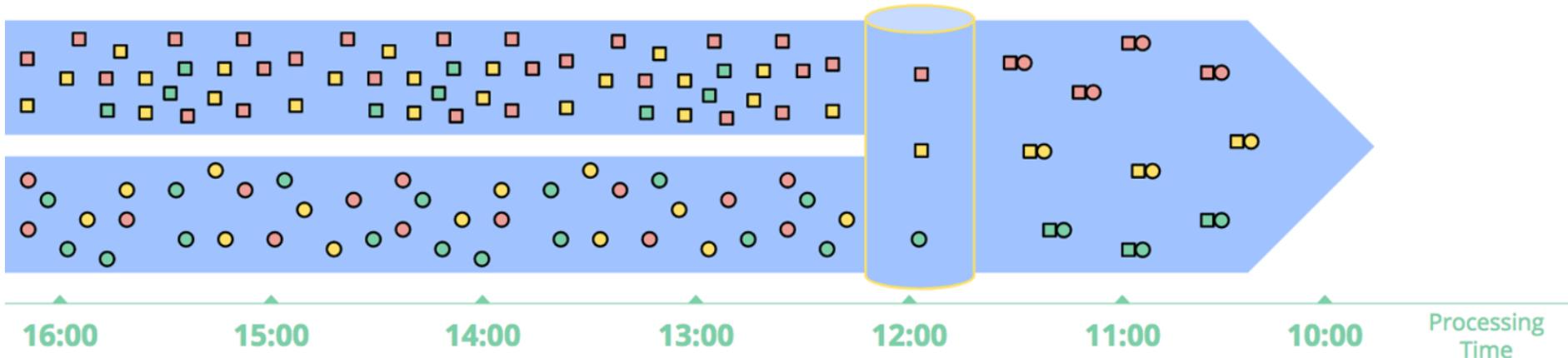
Time agnostic #1: Filtering



<http://streamingsystems.net/fig/1-5>

- Look for data which satisfy a certain filter, e.g. social network users from China & disregard all else

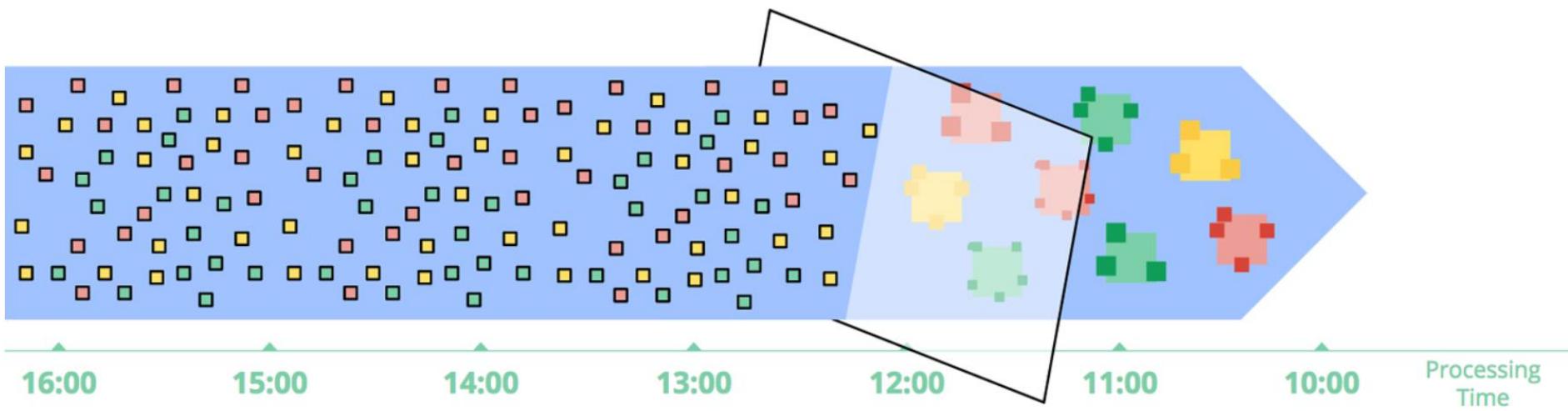
Time agnostic #2: Inner joins



<http://streamingsystems.net/fig/1-6>

- Wait until (a) certain piece(s) of data (marked in red, yellow and green in the figure above) appear(s) in multiple input (data) sources, join them and continue

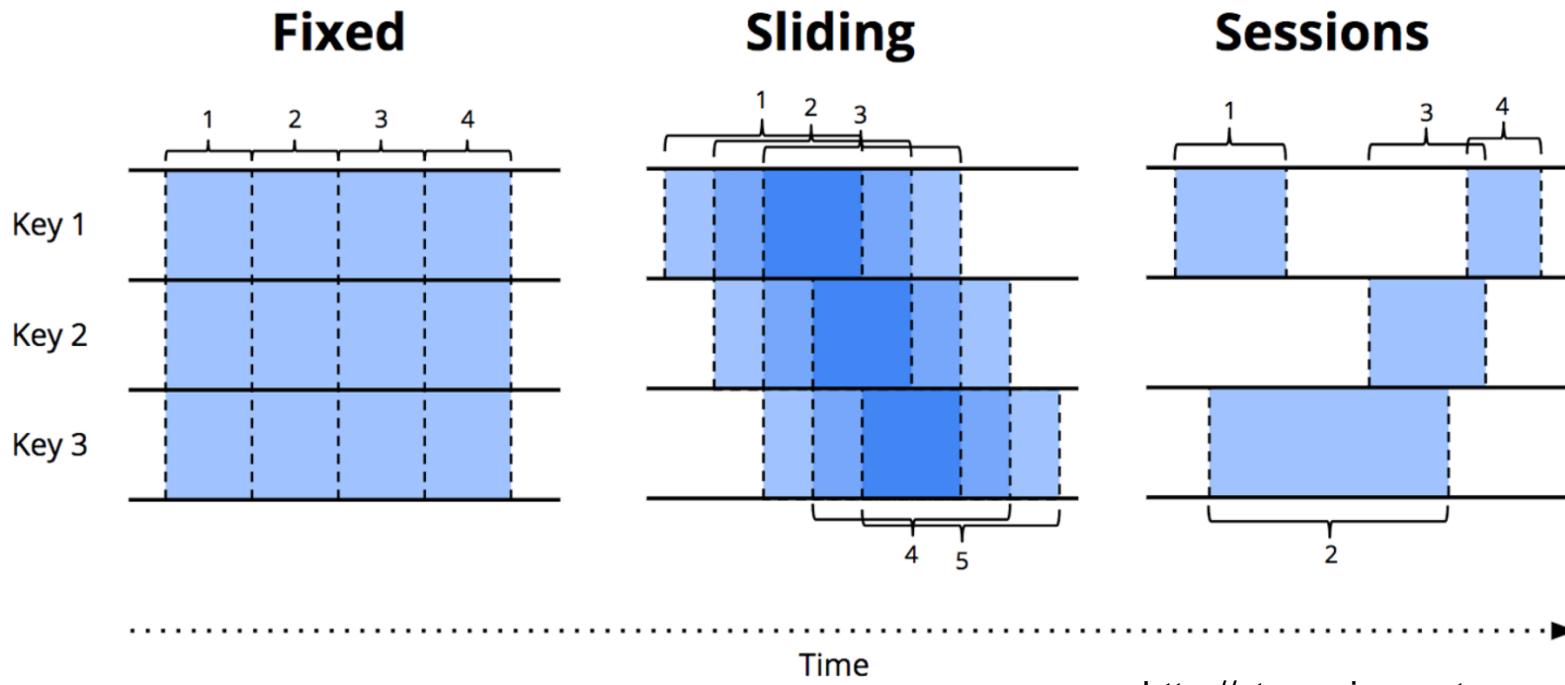
Approximations



<http://streamingsystems.net/fig/1-7>

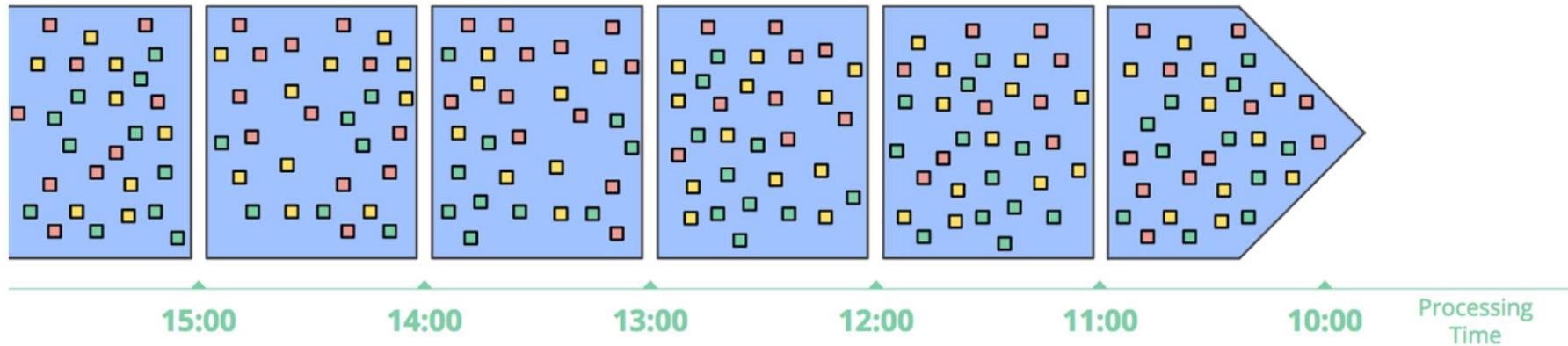
- Goal: ‘look’ at the data and make educated guesses, i.e. approximations
- Pro: low overhead
- Contra: limited number of such algorithms, challenging topic with limited numbers of new contributions

General windowing strategies



- Fixed windows → Slice time into fixed-sized temporal length, usually across all keys/variables.
- Sliding windows → Defined by fixed length & fixed period.
- Sessions → Sequences of events terminated by a gap.

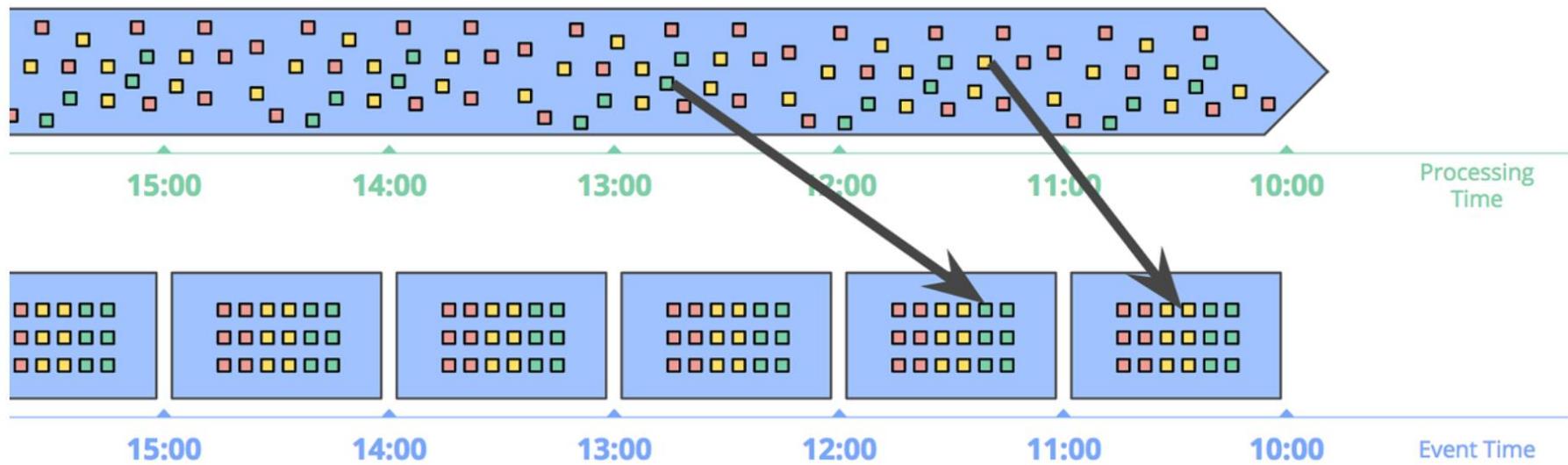
Windowing by processing time



<http://streamingsystems.net/fig/1-9>

- Windows are created based on absolute (processing) time, i.e. data is put into windows based on the order they arrive in

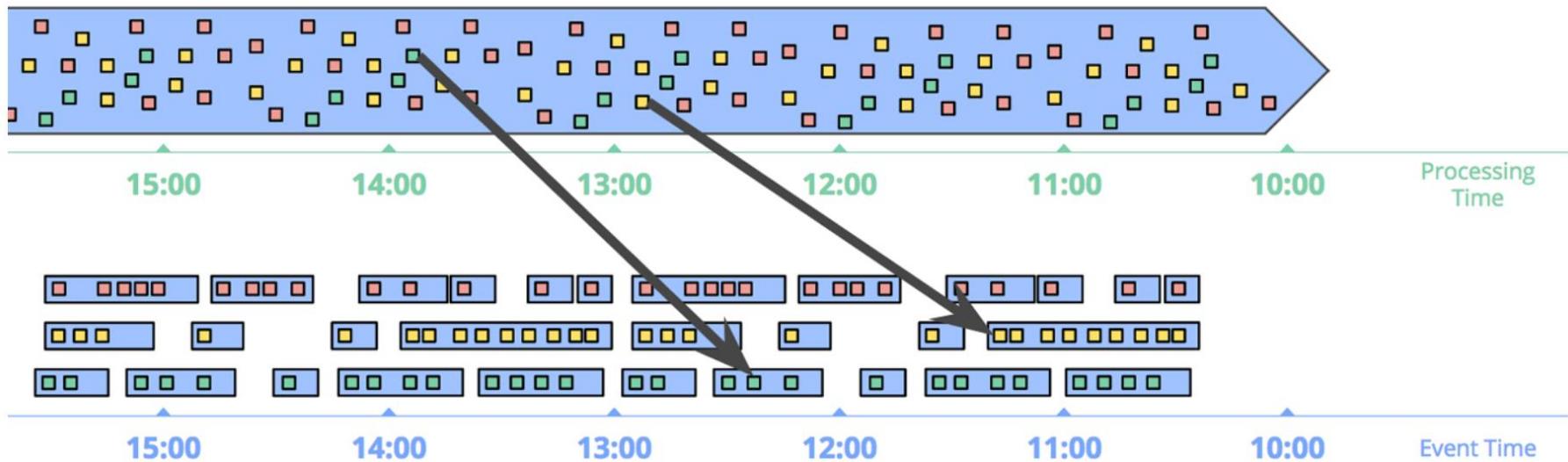
Windowing by event time – Fixed



<http://streamingsystems.net/fig/1-10>

- Data is collected and windowed based on times at which it occurred.

Windowing by event time – Sessions



<http://streamingsystems.net/fig/1-11>

- Data is collected into sessions based on temporal proximity and key/user.

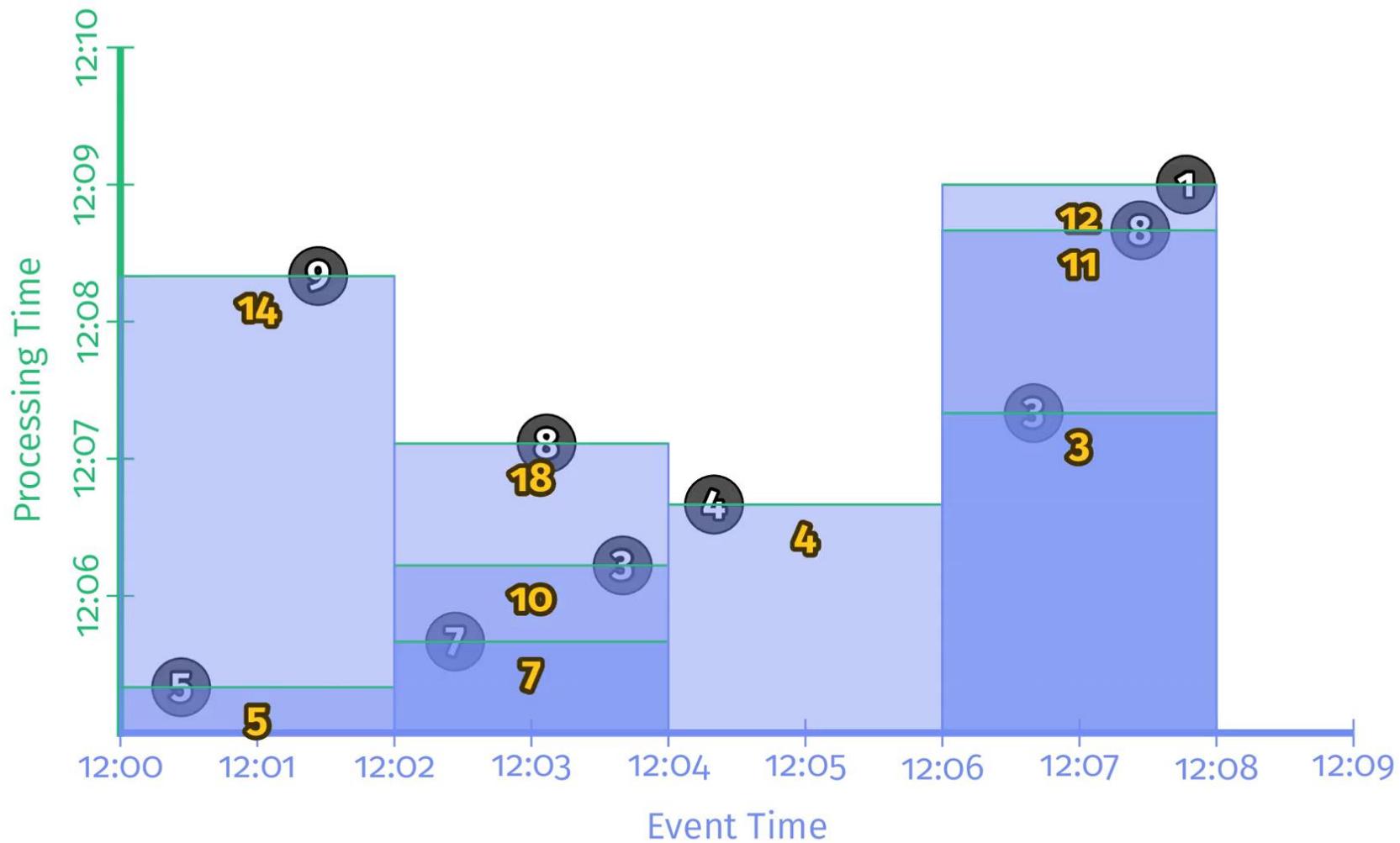
Stream mining

TRIGGERS

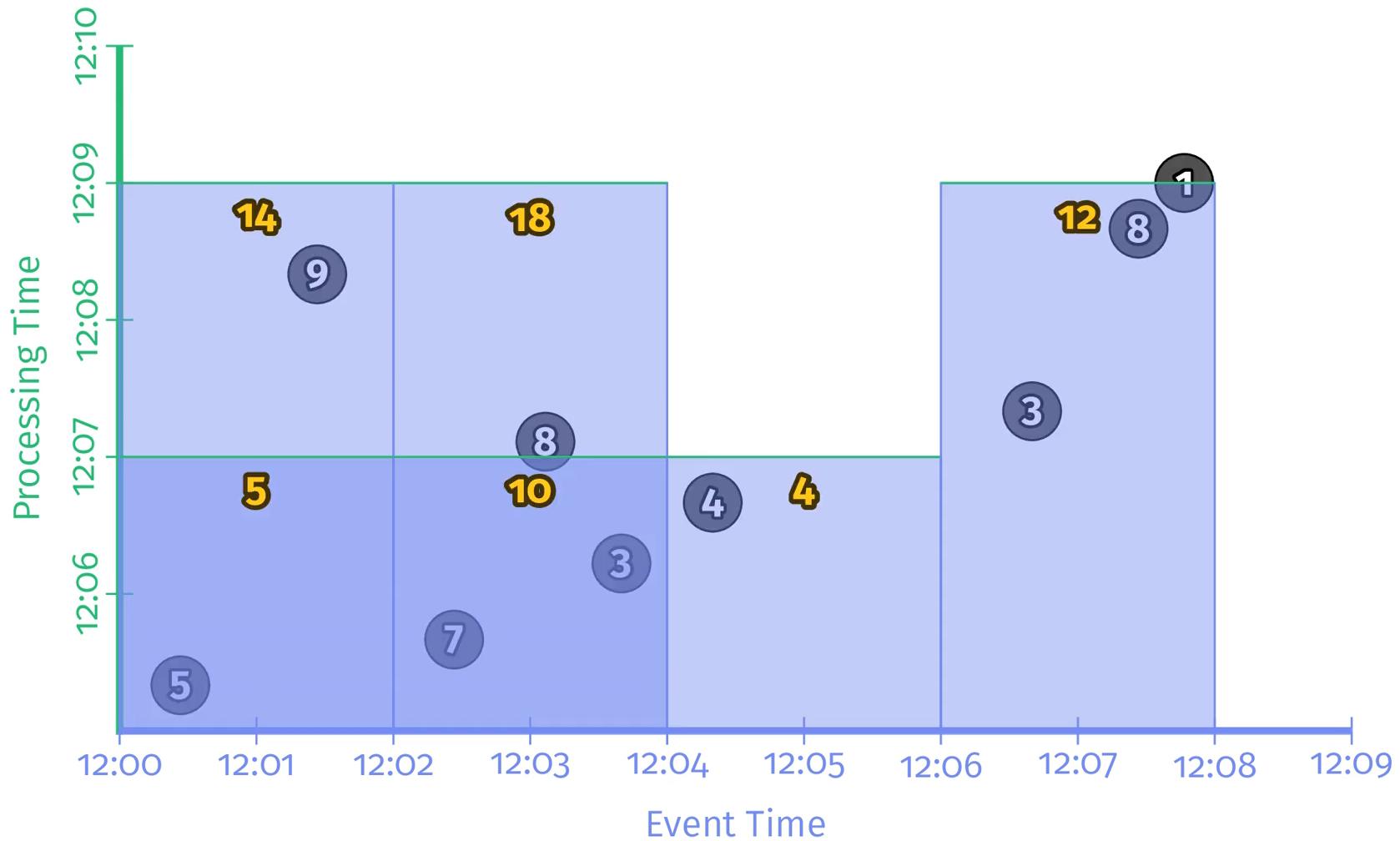
Triggers & variants

- **DEF:** Triggers declare when output for a window should happen in processing time, i.e. when data should be processed
- Trigger types:
 - **Repeated update triggers** = most common in streaming systems, generate periodic updates for a window. Updates materialized for
 - Every new record → too many processing → high CPU load
 - After some processing time delay, e.g. 5 minutes
 - **Completeness triggers** = materialize each time a window is (believed to be) complete to certain degree, e.g. ~80% of data, ~90% of data, etc.
 - Allow reasoning about missing and late data

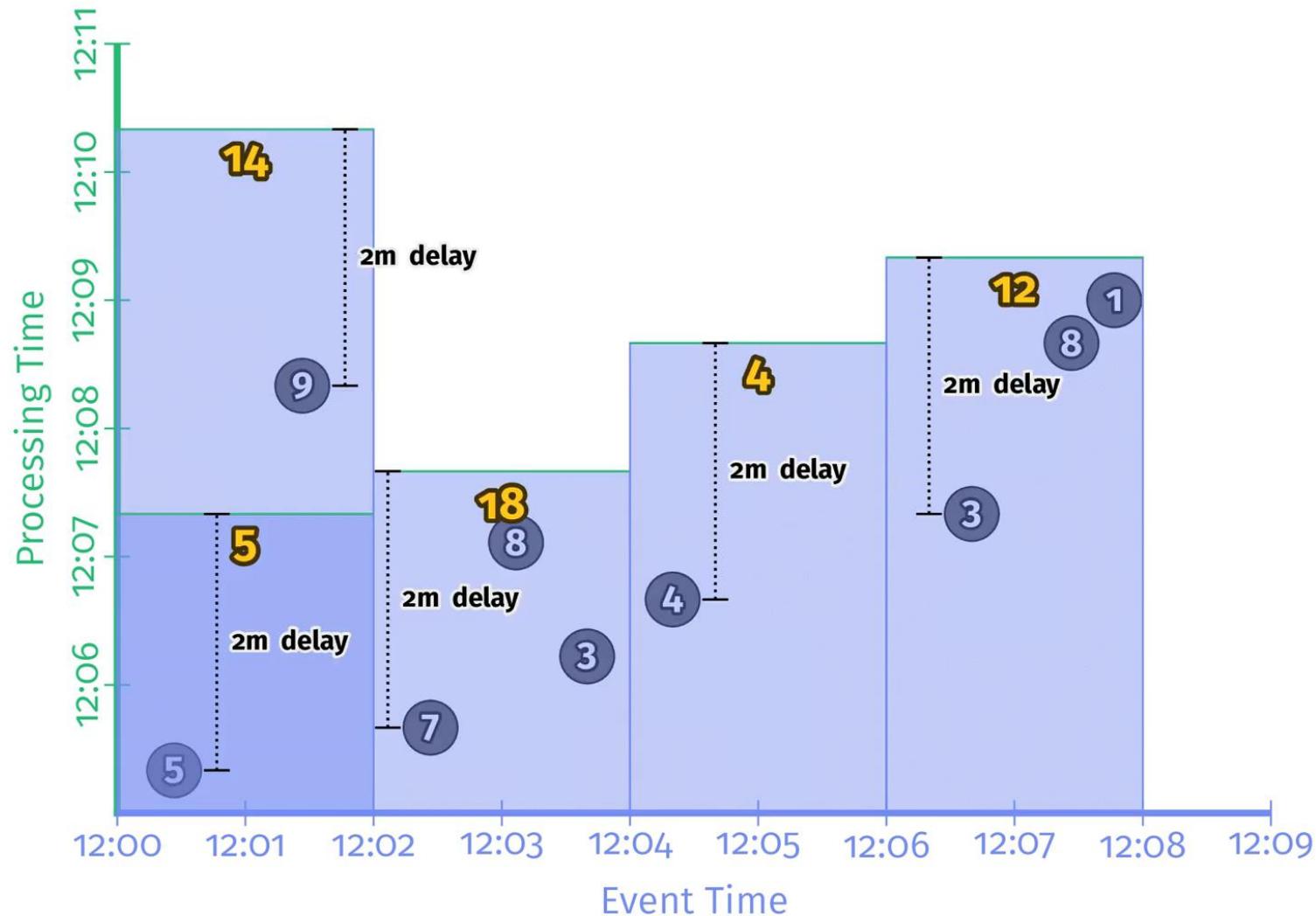
Per record triggering



2-minute aligned delay triggering ~ microbatch streaming



2-minute unaligned delay triggering



Stream mining

WATERMARKS

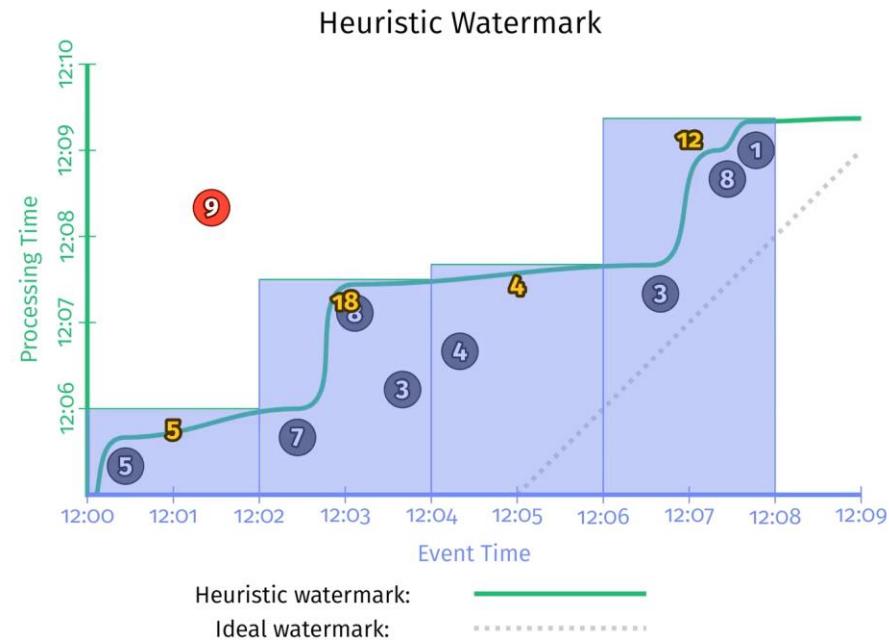
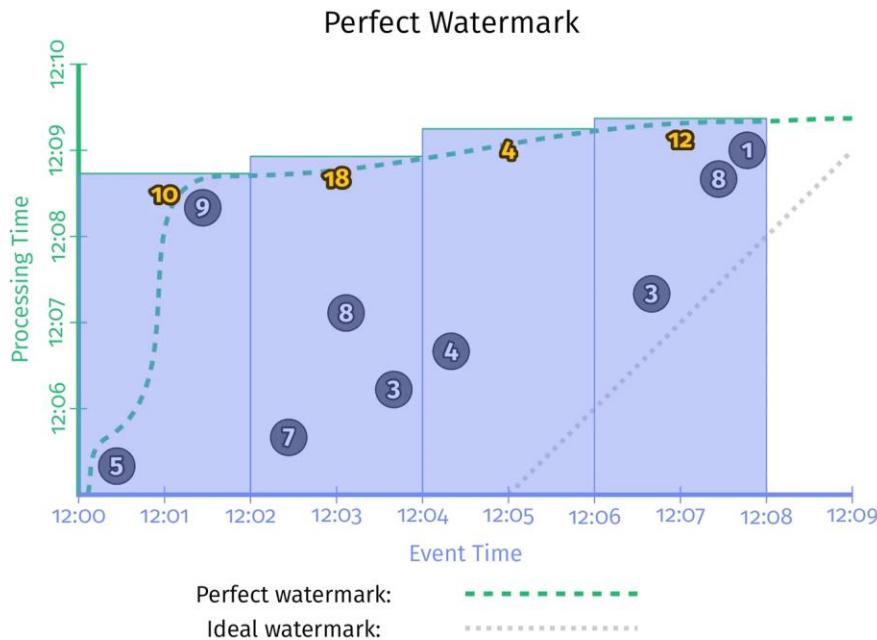
Watermarks

- **DEF:** Watermarks are temporal notions of input completeness in the event time domain
 - ‘Guesses’ about data completeness, i.e. all relevant data received
- Alternative definition: Watermarks allow streaming systems to measure progress and completeness relative to event times of the records being processed

Watermark types

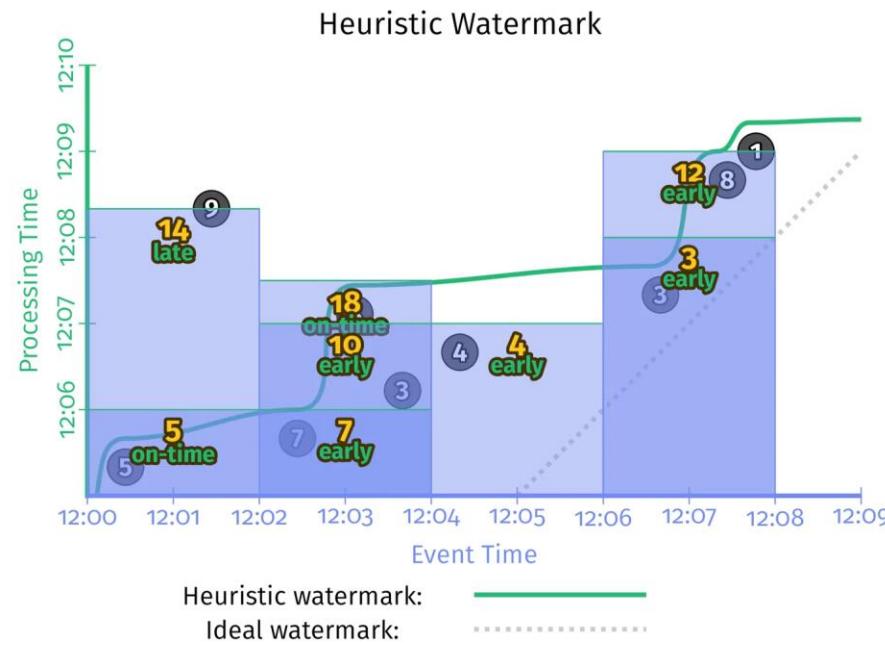
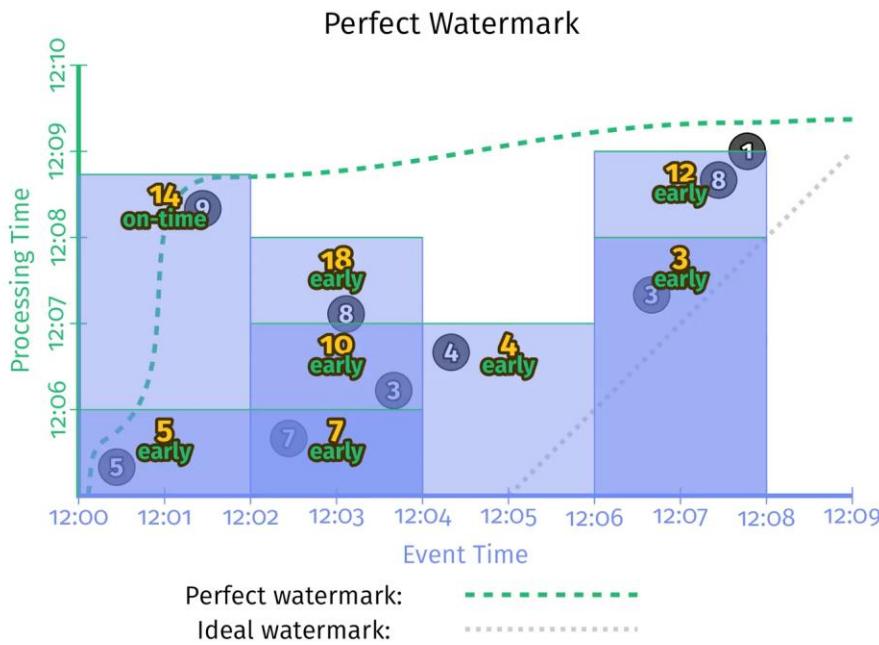
- **Perfect watermark** = usable when we have perfect knowledge about the input data → all data on time (i.e. no late data)
- **Heuristic watermark** = provide an estimate of (data) progress as good as it gets
 - Problem #1: ‘Too slow’ = the watermark is correctly delayed due to known unprocessed data, e.g. due to network delays
 - Problem #2: ‘Too fast’ = when a watermark is advanced earlier than it should be → relevant data is discarded and not processed

Perfect vs heuristic watermarks



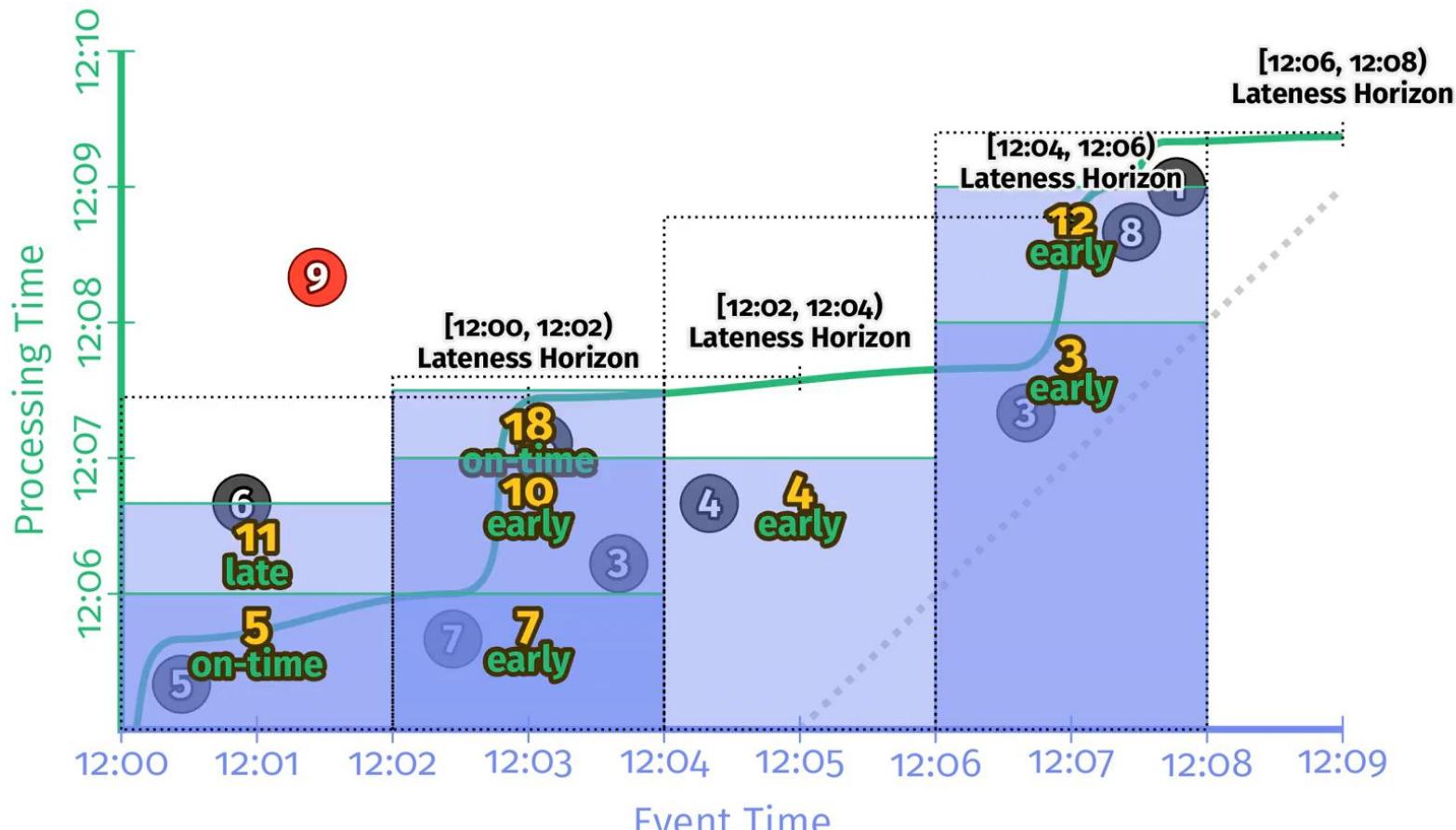
<http://streamingsystems.net/fig/2-10>

Watermarks meet triggers



- Early/on-time/late triggers
 - Zero or more early triggers **periodically** firing
 - Single **on-time** trigger on completeness/watermark
 - Zero or more triggers for **late data** unaccounted for by the watermark

Allowed lateness



Heuristic watermark:

Ideal watermark:

<http://streamingsystems.net/fig/2-12>

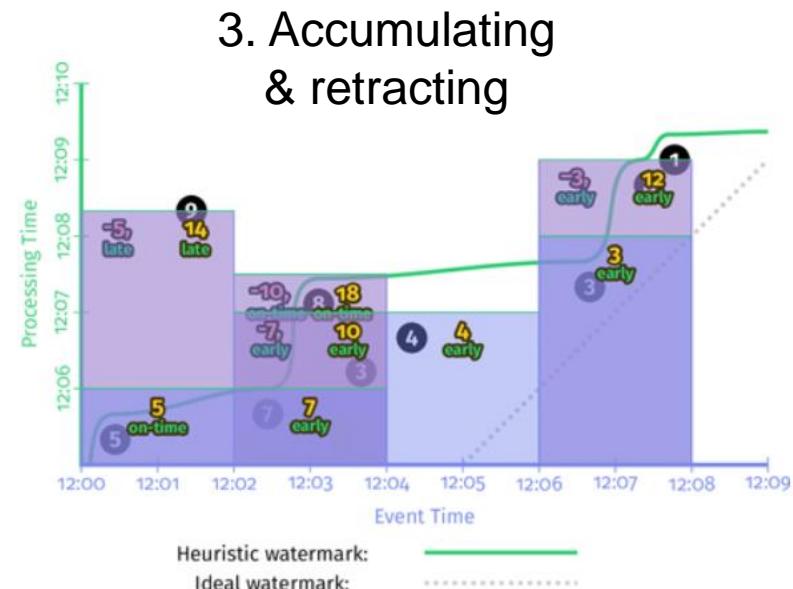
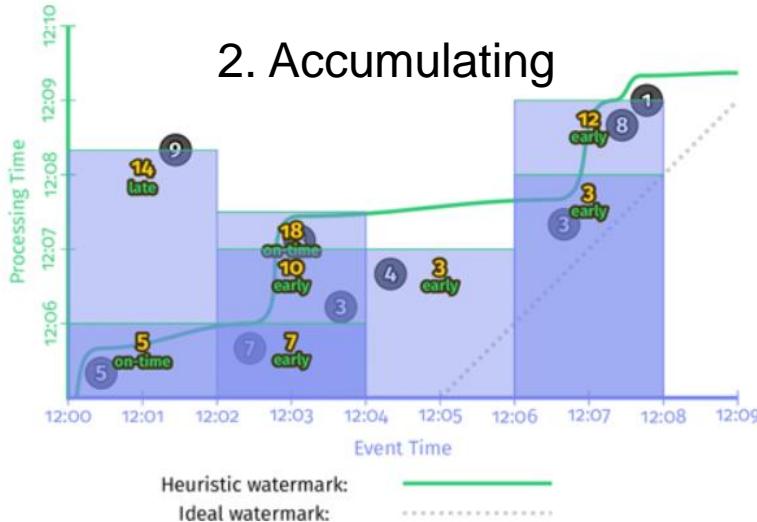
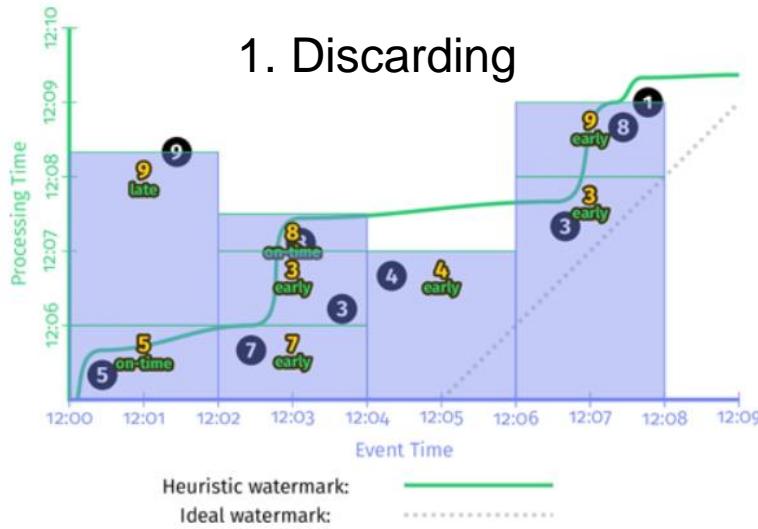
Stream mining

ACCUMULATION

Accumulation intro

- Types of accumulation when triggers produce multiple results (processing or panes) during a single window
 - **Discarding** = any prior data originating from an earlier processing triggered during the same window is **discarded**
 - **Accumulating** = stored state is retained (as opposed to discarding) – useful for generating aggregates based on ‘deltas’
 - **Accumulating and retracting** = an extension of the accumulating approach, but additionally retaining the difference between the current processed value and the prior value, e.g. if the current sum is 12 and the previous sum was 9, then retain (12, -3)

Accumulation types side-by-side



Summary

- Streaming basics
- Reasoning about time
- Batch data processing 101
- Stream processing 101
- Triggers
- Watermarks
- Accumulation



Thank you for your attention!



WATERMARKS AND WINDOWING (W2)

Stream mining (SM)

Imre Lendák, PhD, Associate Professor

Péter Kiss, PhD candidate

Outline

- Advanced watermarking
- Advanced windowing



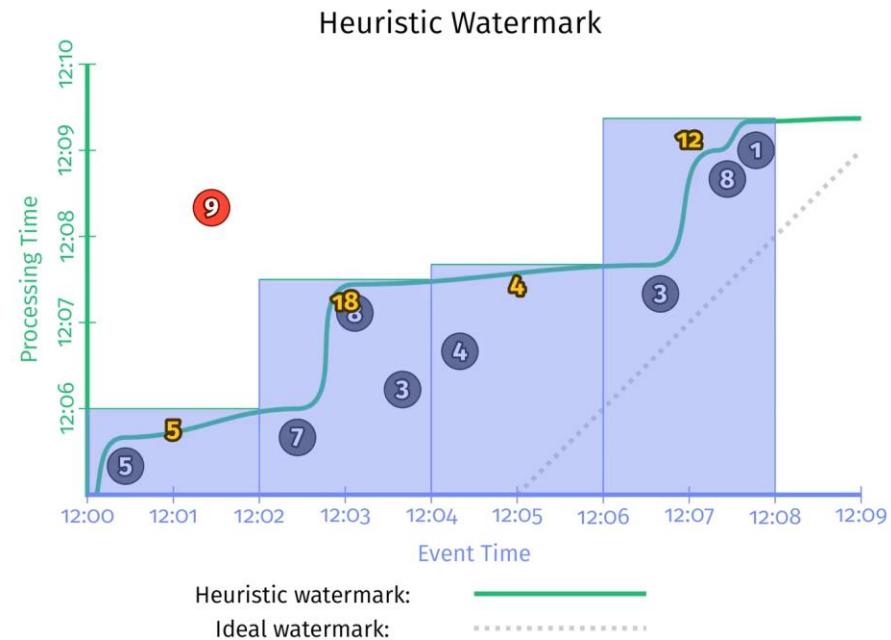
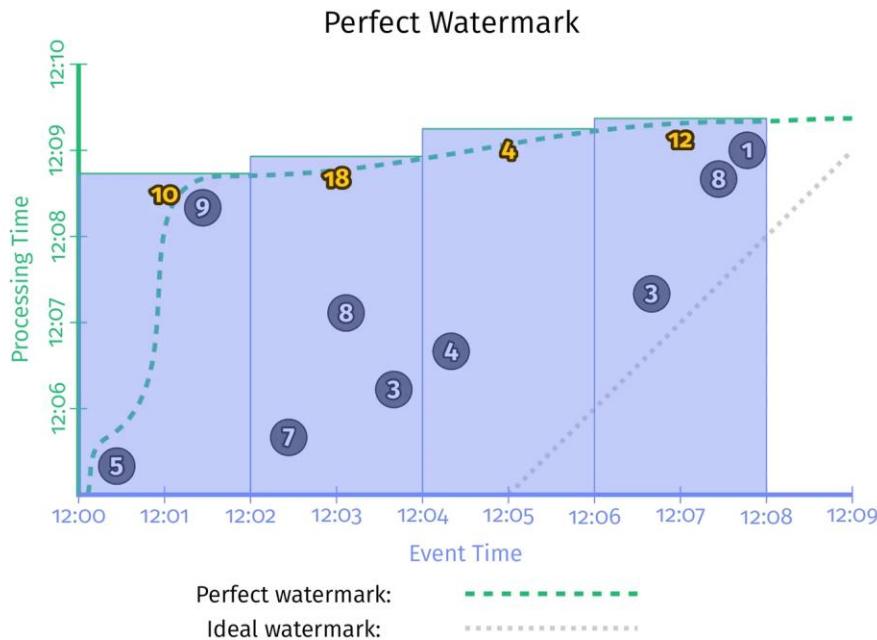
Stream mining

ADVANCED WATERMARKING

Watermarks

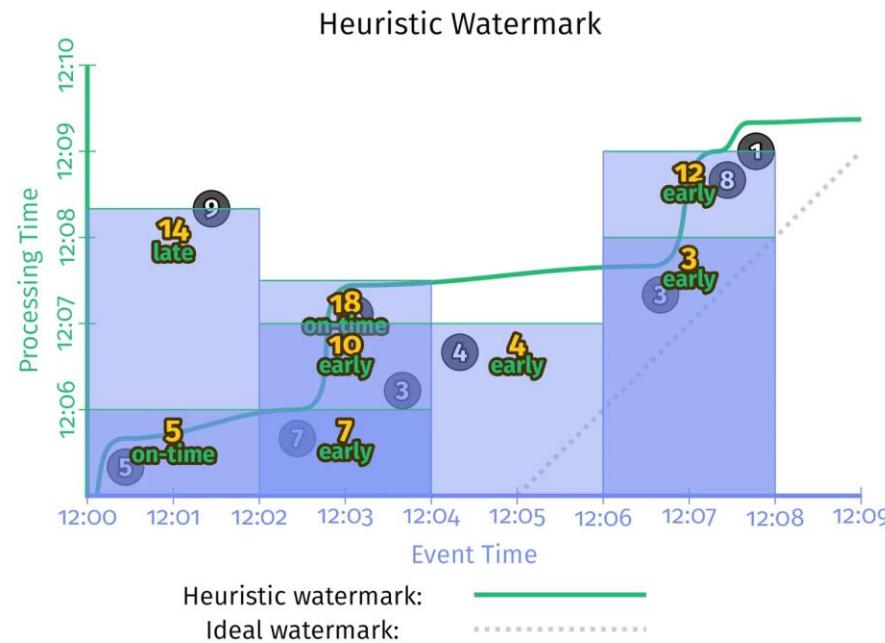
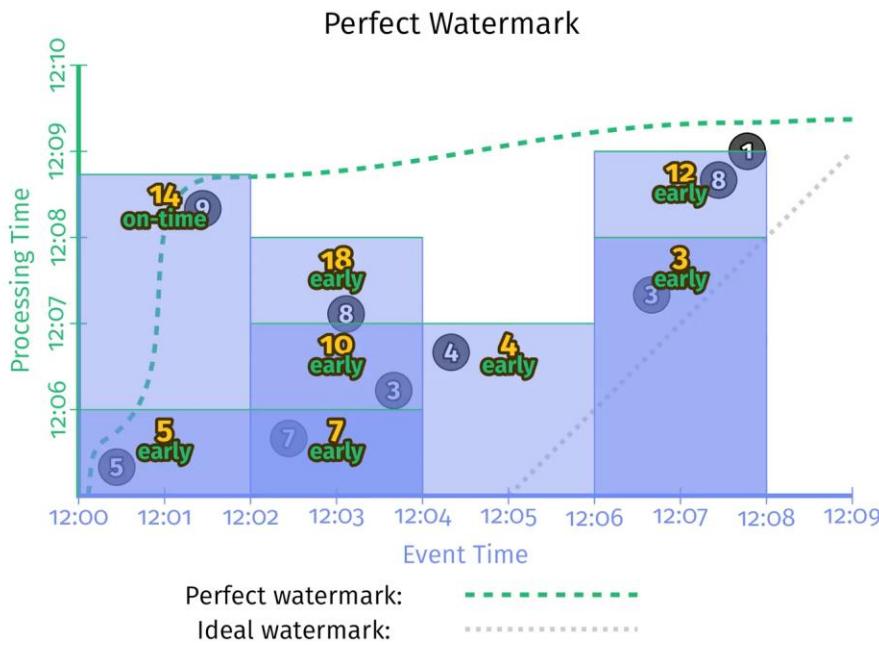
- **DEF #1:** Watermarks are temporal notions of input completeness in the event time domain
 - ‘Guesses’ about data completeness, i.e. all relevant data received
- **DEF #2:** Watermarks allow streaming systems to measure progress and completeness relative to event times of the records being processed
- Watermark varieties:
 - Perfect watermark = usable when we have perfect knowledge about the input data → all data on time (i.e. no late data)
 - Heuristic watermark = provide an estimate of (data) progress as good as it gets → usually based on domain knowledge
- **NOTE:** Additional watermark management challenges in data processing pipelines with multiple processing steps

Perfect vs heuristic watermarks



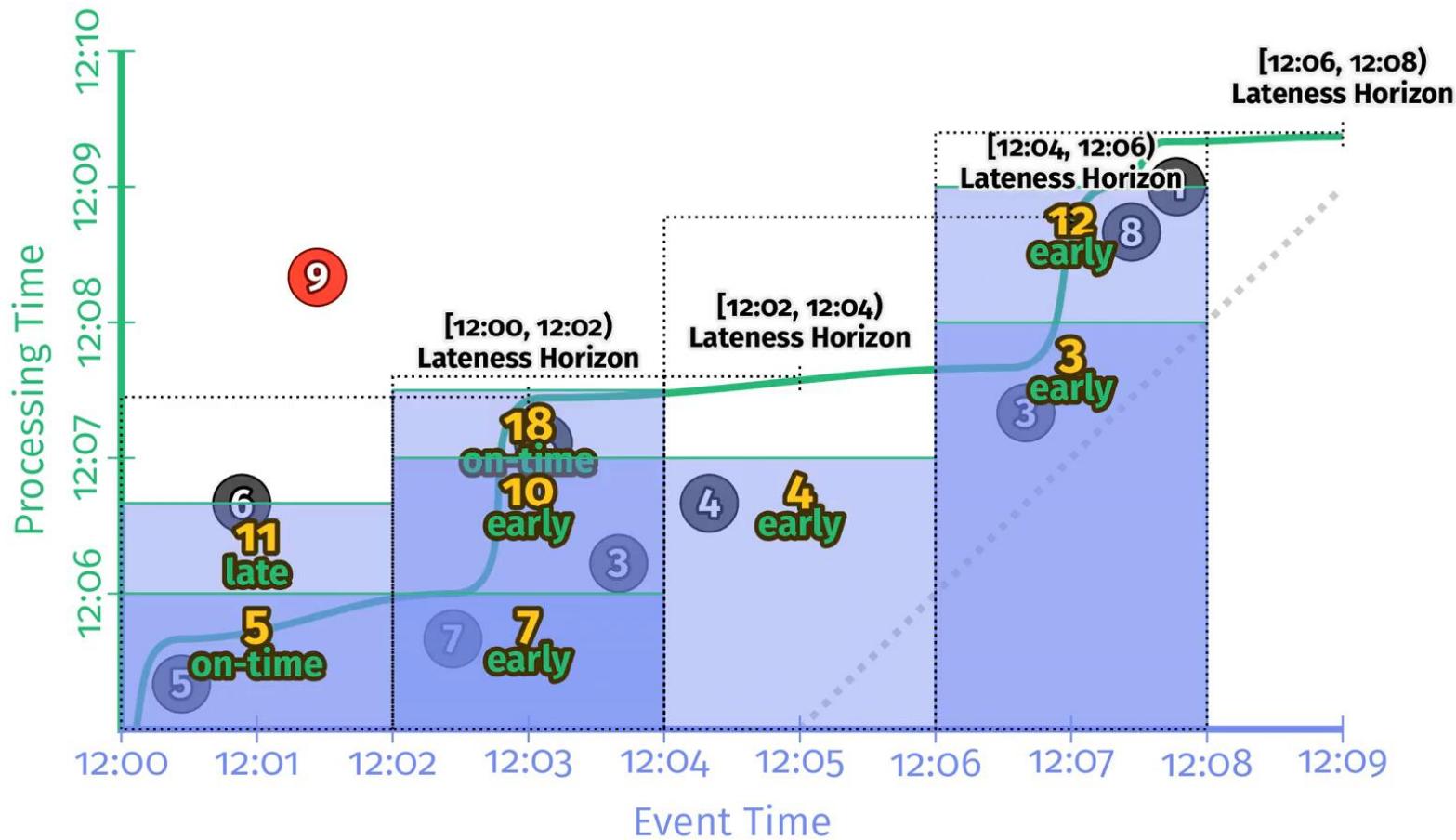
<http://streamingsystems.net/fig/2-10>

Watermarks meet triggers



- Early/on-time/late triggers
 - Zero or more early triggers **periodically** firing
 - Single **on-time** trigger on completeness/watermark
 - Zero or more triggers for **late data** unaccounted for by the watermark

Allowed lateness



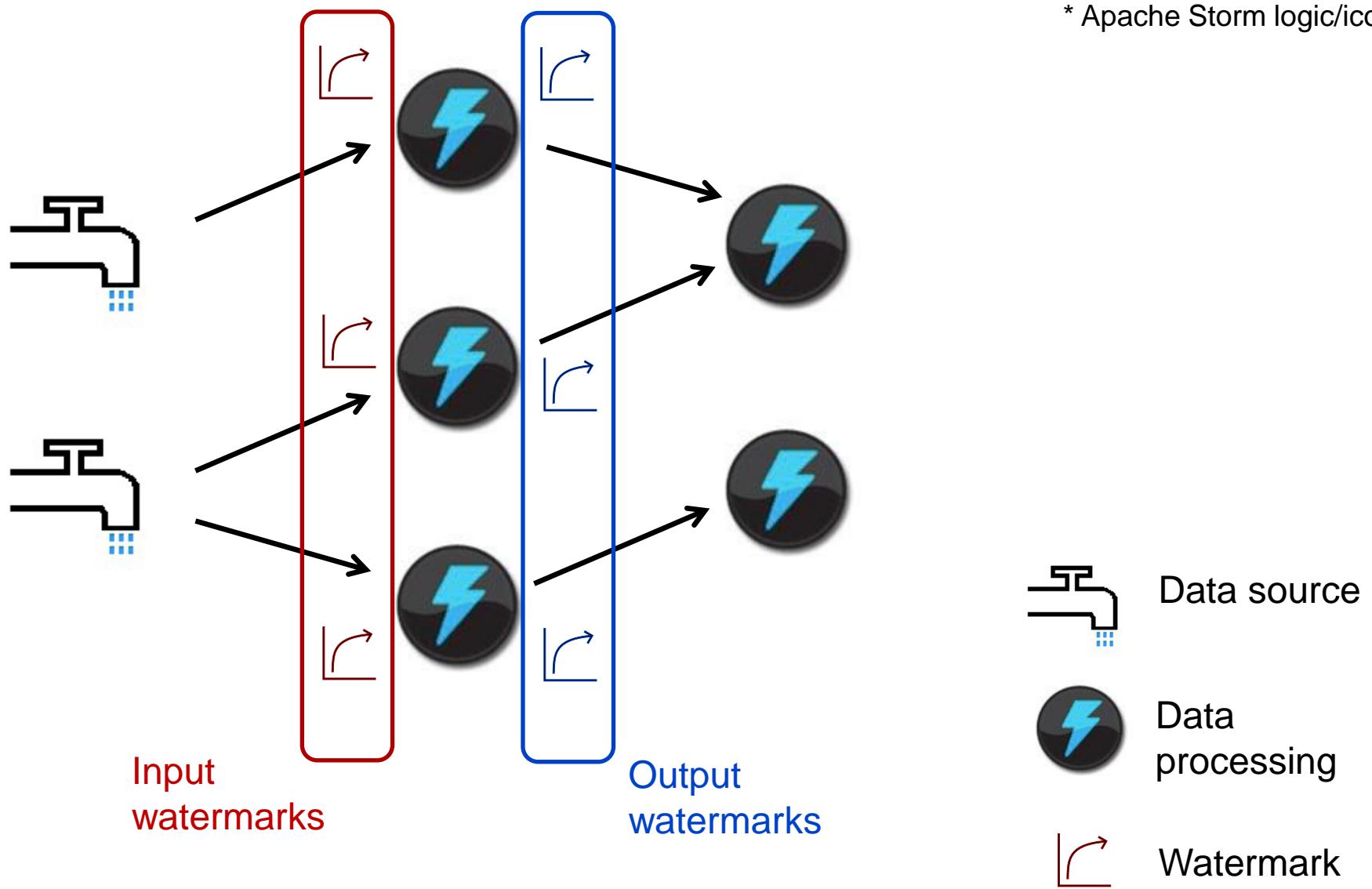
Heuristic watermark:

Ideal watermark:

<http://streamingsystems.net/fig/2-12>

Watermark @ boundaries

* Apache Storm logic/icons



Input watermarks

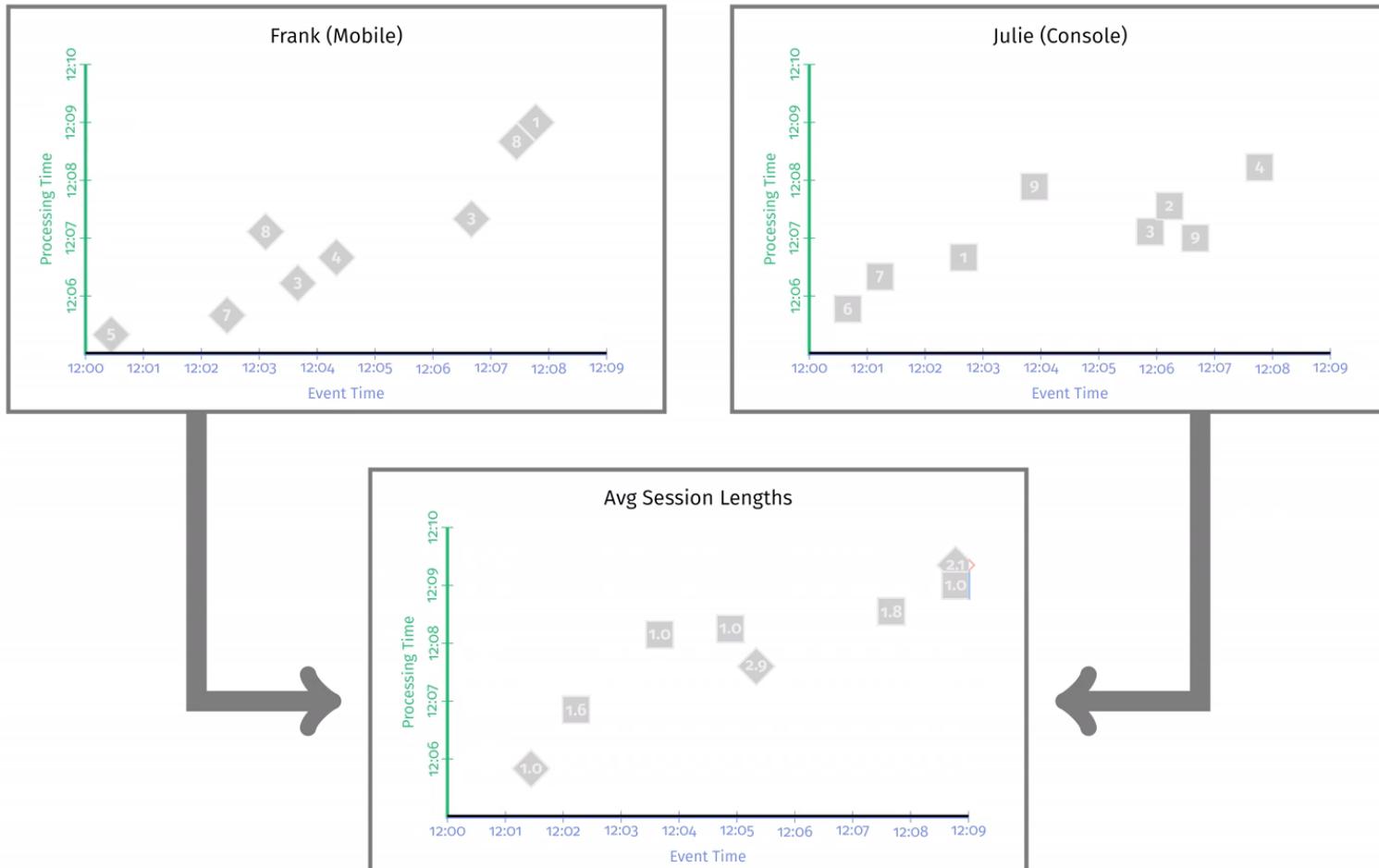
- **Input watermark** = calculated based on the data processing pipeline's progress upstream ('to the left') of the input of the current processing stage
- For (data) sources there is a specific function generating the watermark. It is created based on our understanding of the input data source
- For non-sources it is the **minimum** of all dependent output watermarks upstream
 - Why the minimum?

Output watermarks

- Each processing node can **buffer** active messages until some operation is completed, e.g. aggregation in a previous stage.
 - Each buffer can be tracked with its own watermark
- **DEF:** Output watermarks are calculated at the output(s) of processing nodes and incorporate/observe input + processing (time) delay
- Output watermarks are (usually) based on a combination of the following watermarks
 - Data source → source watermarks
 - External input → other external sources, e.g. upstream processing
 - (Internal) state → there is an internal state-machine inside the processing node with clearly defined rules of progress
 - Per-output buffer → times of (data) processing results written to output buffers



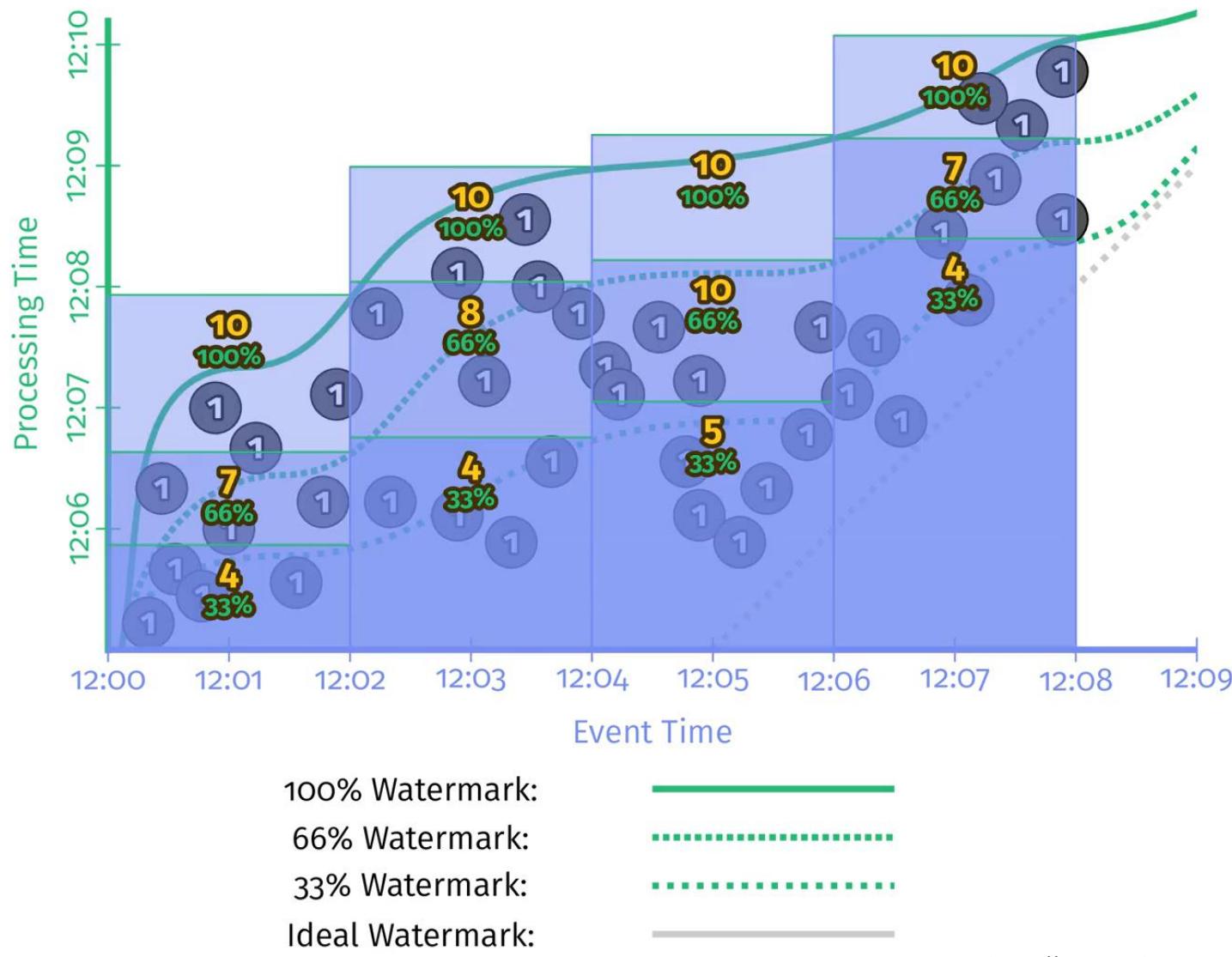
Watermark propagation



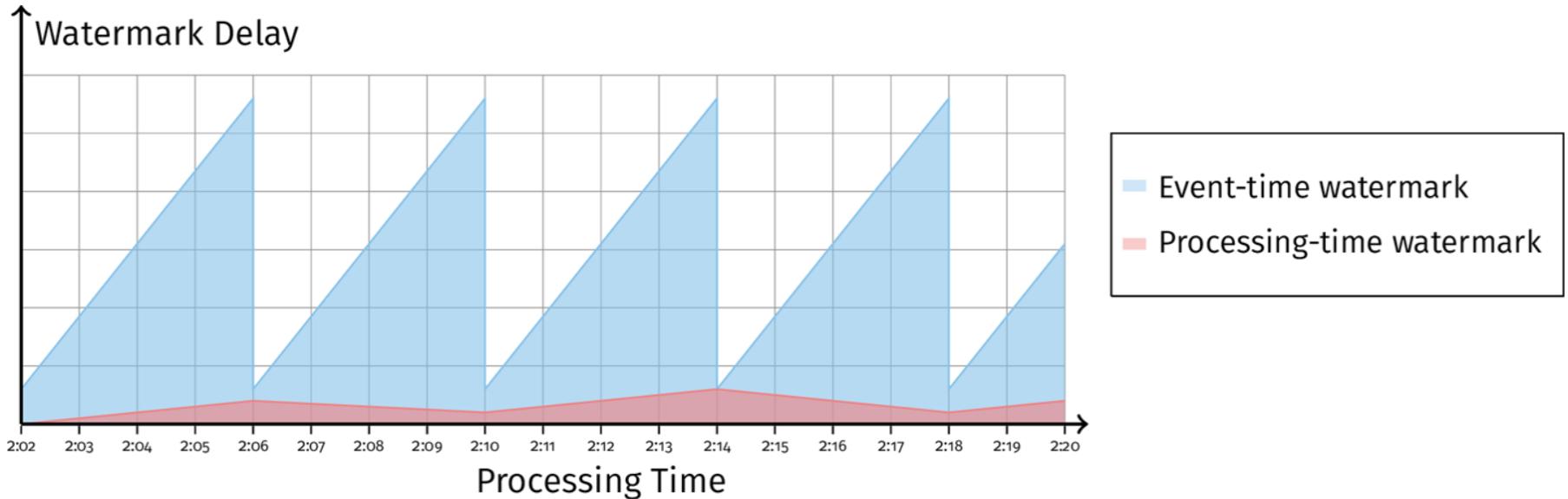
Mobile Scores - Input Watermark:
Mobile Scores - Output Watermark:
Console Scores - Input Watermark:
Console Scores - Output Watermark:
Top Teams - Input Watermark:

<http://streamingsystems.net/fig/3-5>

Percentile watermarks



Processing time watermarks



- The processing time watermark is constructed based on the timestamp of the oldest (processing) operation not yet completed
 - Allows insight into processing delay separate from data delay
 - Growing processing watermark → faults preventing (process) operations to complete → user/administrator action needed
- Useful to distinguish (streaming) system latency from data (source) latency

<http://streamingsystems.net/fig/3-15>

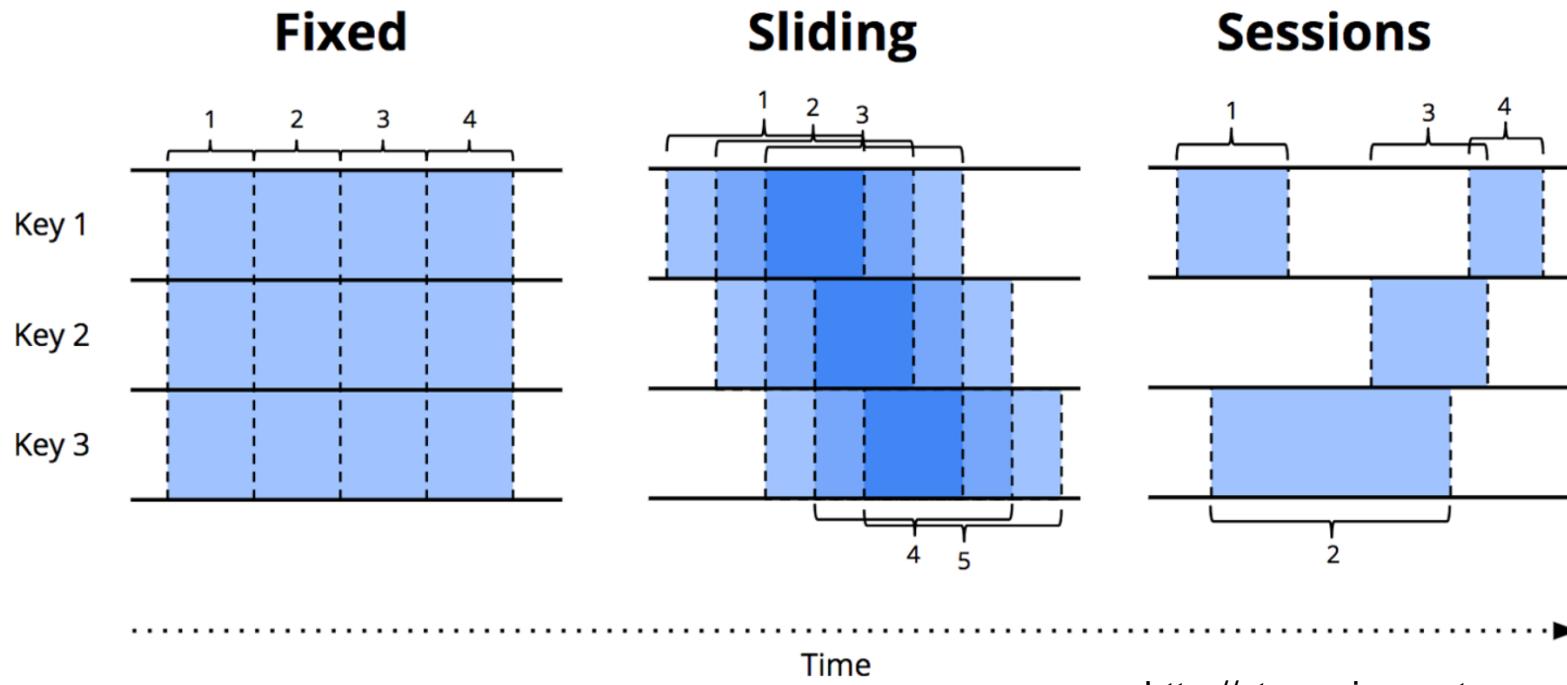
Stream mining

WINDOWING

Advanced windowing intro

- Processing-time windowing is useful for use cases in which data is observed as received, e.g. web server traffic
 - Implementation via triggers or ingress time
 - Time-sensitive → use only when data is received in perfect order or when we do not care about when events actually happened
- Event-time windowing is used in use cases in which the exact sequence of events is relevant, e.g. billing, user behavior
 - Order agnostic, i.e. does not care about the order in which the data is received as these solutions are able to re-order pieces of data received from different sources in jumbled order

General windowing strategies

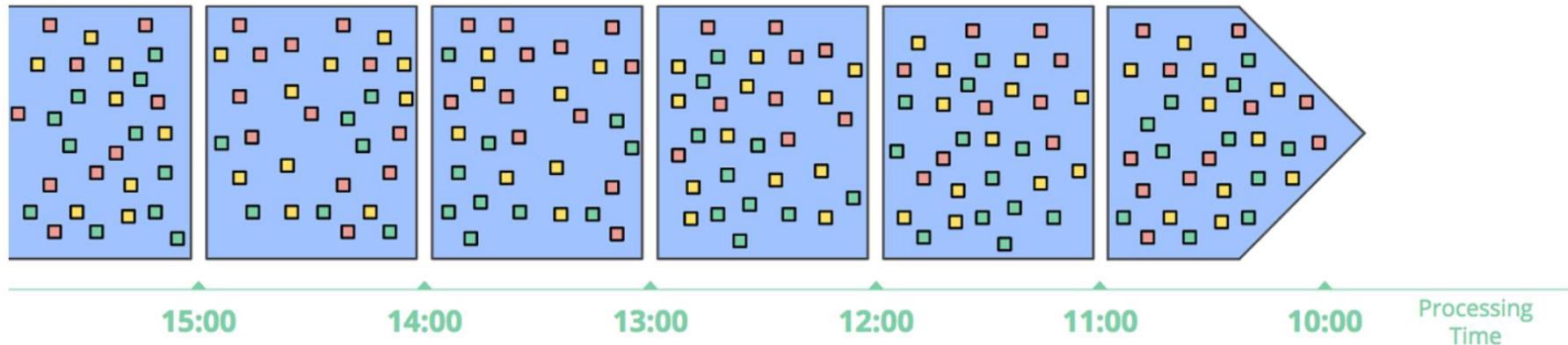


- Fixed windows → Slice time into fixed-sized temporal length, usually across all keys/variables.
- Sliding windows → Defined by fixed length & fixed period.
- Sessions → Sequences of events terminated by a gap.

Stream mining

PROCESSING-TIME WINDOWING

Windowing by processing time



<http://streamingsystems.net/fig/1-9>

- Windows are created based on absolute (processing) time, i.e. data is put into windows based on the order they arrive in

Processing-time windowing & triggers

- Triggers fire periodically in the processing-time domain and initiate data processing, e.g. every 2 minutes
- Accumulating or discarding depending on the use case
- Most traditional streaming systems worked like this

PT with triggers in action



- Processing-time windowing via triggers can create different results for the same input data set with different processing-time ordering

Processing-time windowing & ingress time

- Ingress time windowing basics
 - Event times are replaced with ingress times, i.e. the times when they were received by the streaming system
 - Utilize perfect watermarks made possible via ingress time → processing is triggered when the watermark is reached
 - Accumulation mode is based on the use case
- Outputs can be different for different ordering of the same set of input data

PT with ingress time in action

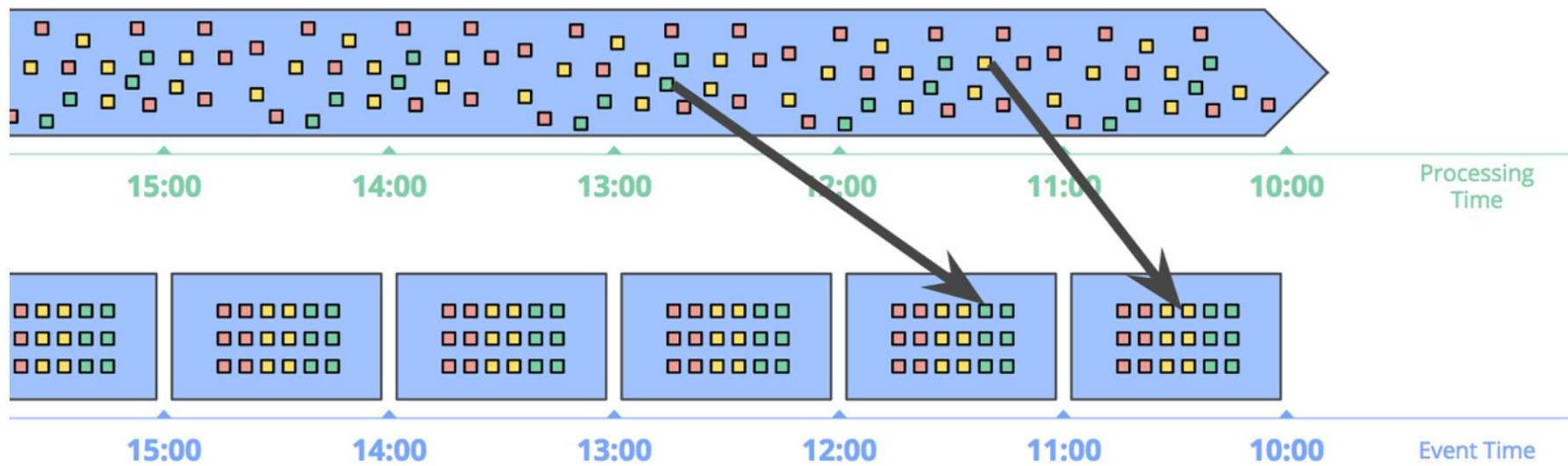


- Note: a perfect watermark is made possible via the use of ingress time

Stream mining

EVENT-TIME WINDOWING

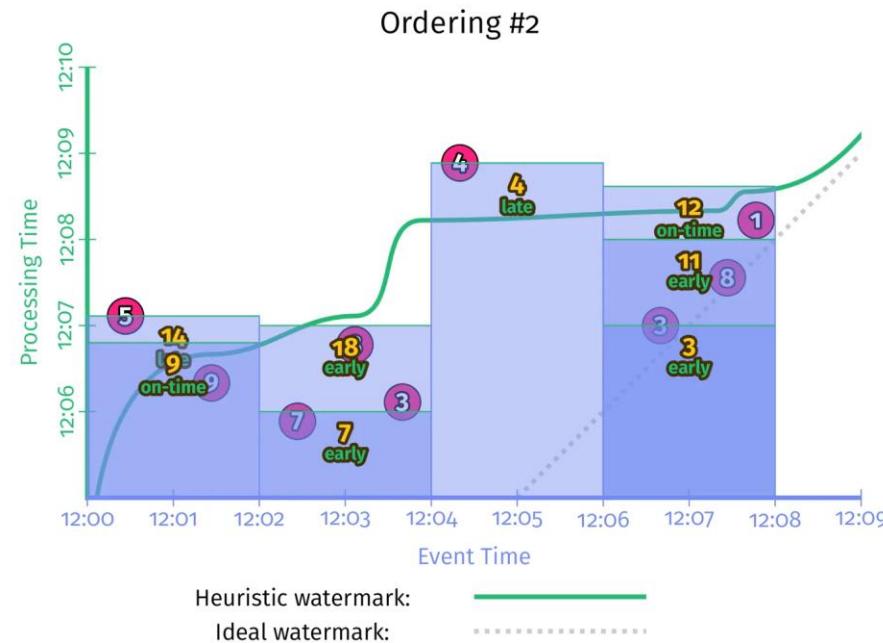
Windowing by event time – Fixed



<http://streamingsystems.net/fig/1-10>

- Data is collected and windowed based on times at which it occurred, i.e. event time.

Event-time windowing in action



- Sum calculus in event-time windows over the same pieces of data in two different processing time orderings.

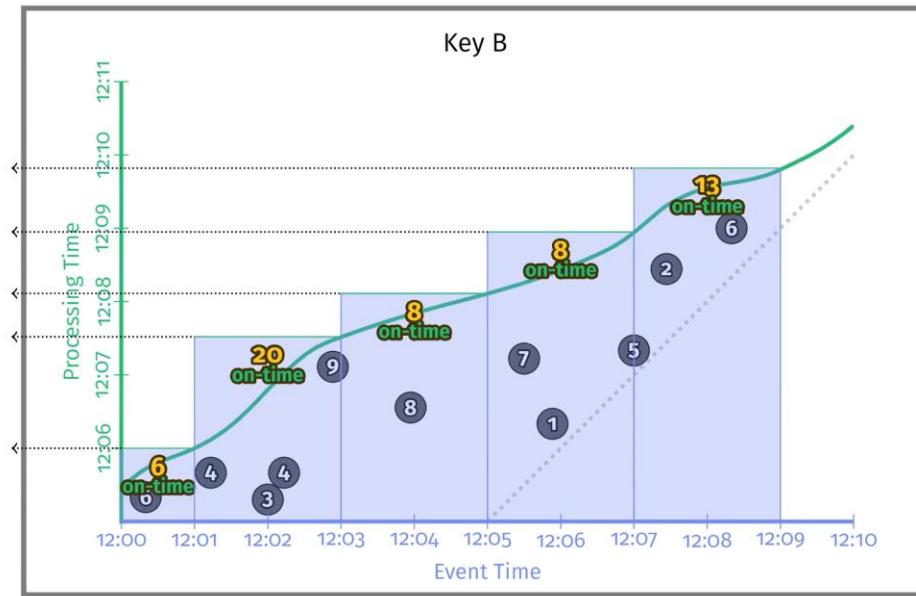
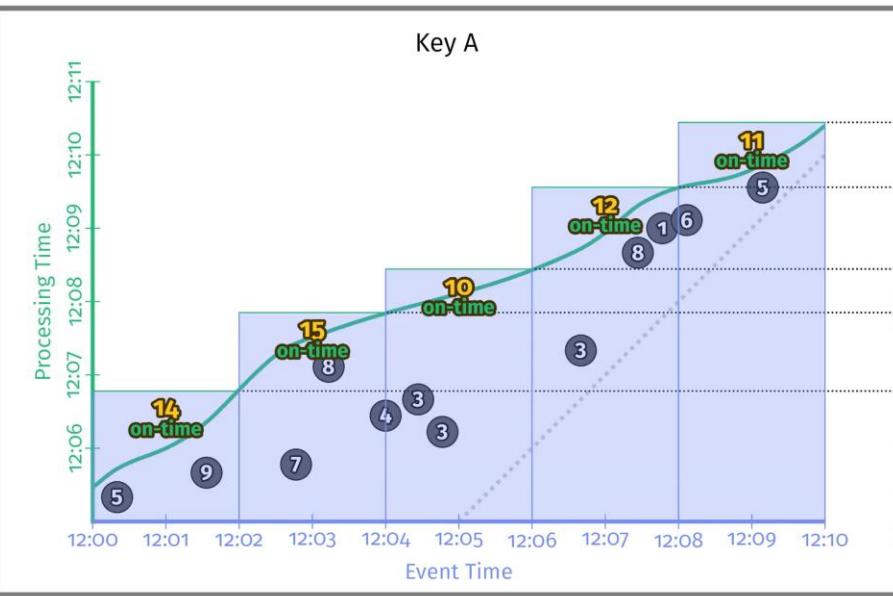
Variation #1: Aligned fixed windows

- Aligned fixed windows start and end at the same time for all keys/observations
 - Example: thousands of measurement points on a factory floor with advanced motor monitoring with high frequency data sampling for early fault detection, which are processed every minute (in absolute, processing time)
- Pro:
 - Implementing fixed windows is relatively easy
- Contra:
 - The data is seldom occurring in a windowed fashion in real-life streaming systems
 - There is a high processing (CPU, RAM) load at the end of the windows when all windows are processed at the same time

Variation #2: Unaligned fixed windows

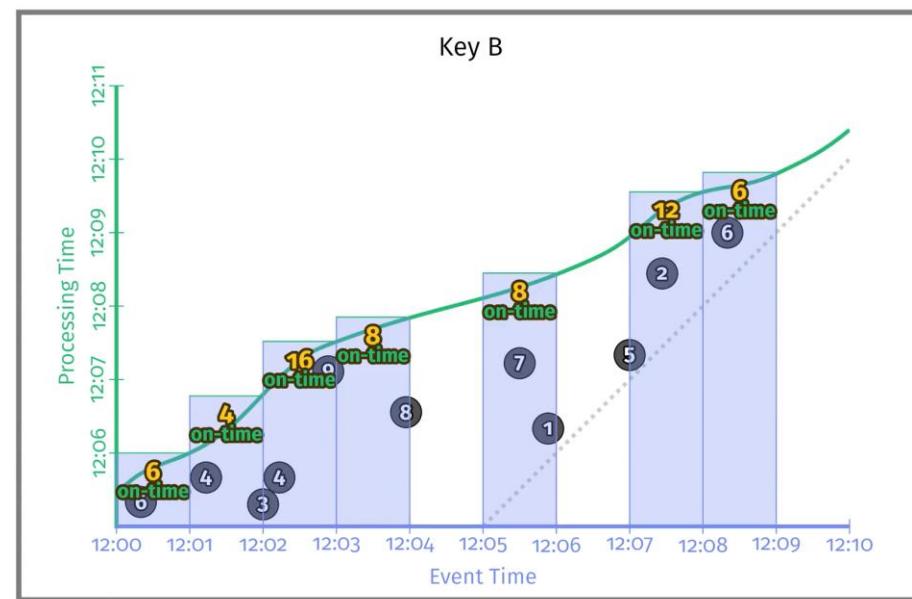
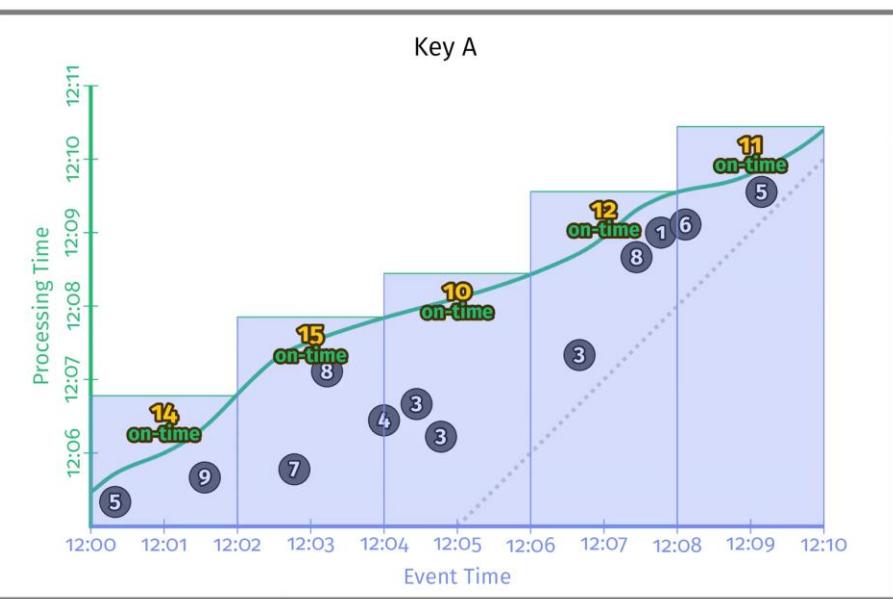
- Unaligned fixed windows do not start at the same time for all keys/observations
- The length of these windows is still pre-defined
- Pro:
 - More evenly spread processing loads
- Contra:
 - The data is seldom occurring in a windowed fashion in real-life streaming systems
 - Not as easy to implement

Unaligned fixed windows in action



- The horizontal lines mark the ‘ends’ of the windows → that’s when processing is triggered

Variation #3: Per-element/key fixed windows

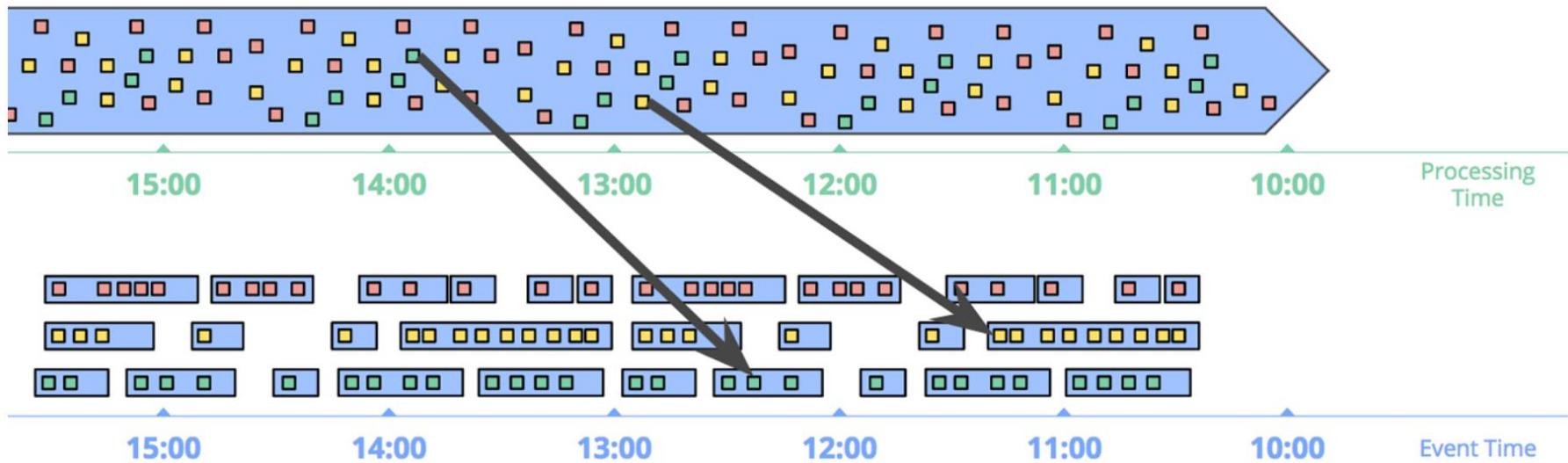


- Fixed windows with different key sizes:
 - Key A with fixed window size = 2 minute(s)
 - Key B with fixed window size = 1 minute(s)
- Useful when different stream analysis requirements exist

Stream mining

SESSIONS IN WINDOWING

Windowing by event time – Sessions



<http://streamingsystems.net/fig/1-11>

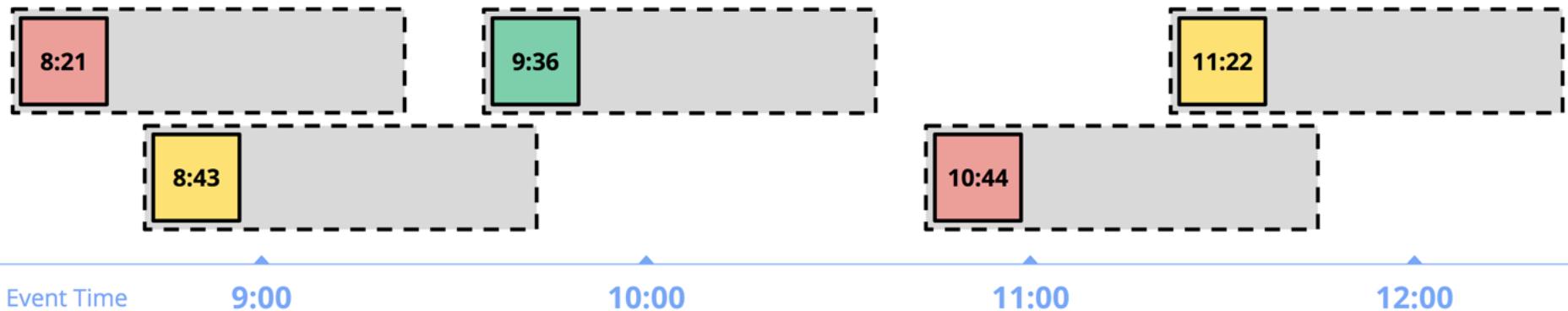
- Data is collected into sessions based on temporal proximity and key/user.

Session windows

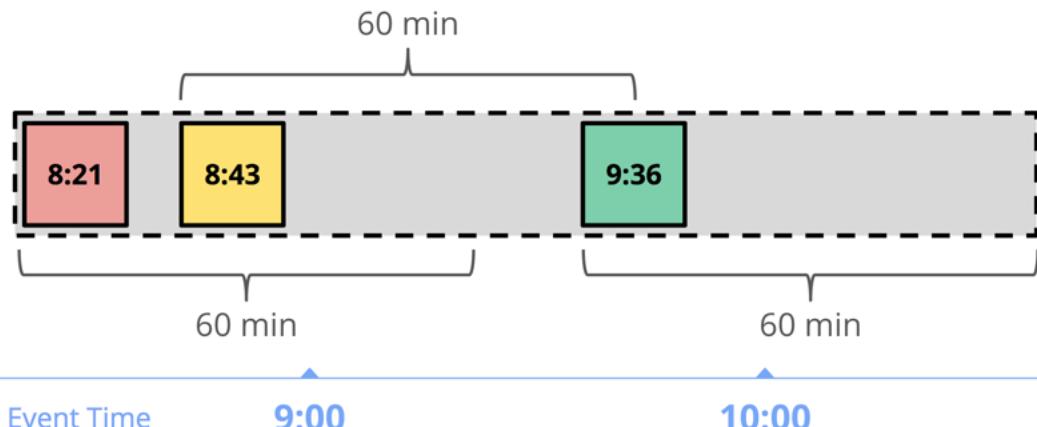
- Session windows are data-driven → the location and length of windows is dictated by the input data itself
 - This is opposed to fixed windows, which are finalized after the expiration of a certain time period either in event or processing time
- Sessions are unaligned
 - There is absolutely no guarantee that data will occur at the exact same time for different keys
- Session creation can be based on
 - Predefined (time) gap which separates two sessions – this is done more often
 - Predefined tag which is assigned to each piece of data – this is a less frequent scenario

Proto sessions & merging

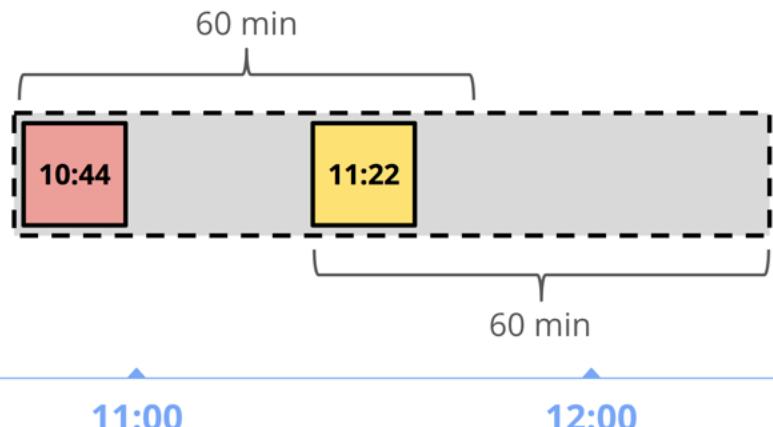
Unmerged Proto-sessions - 60 min each



Merged Session #1 - 135 min



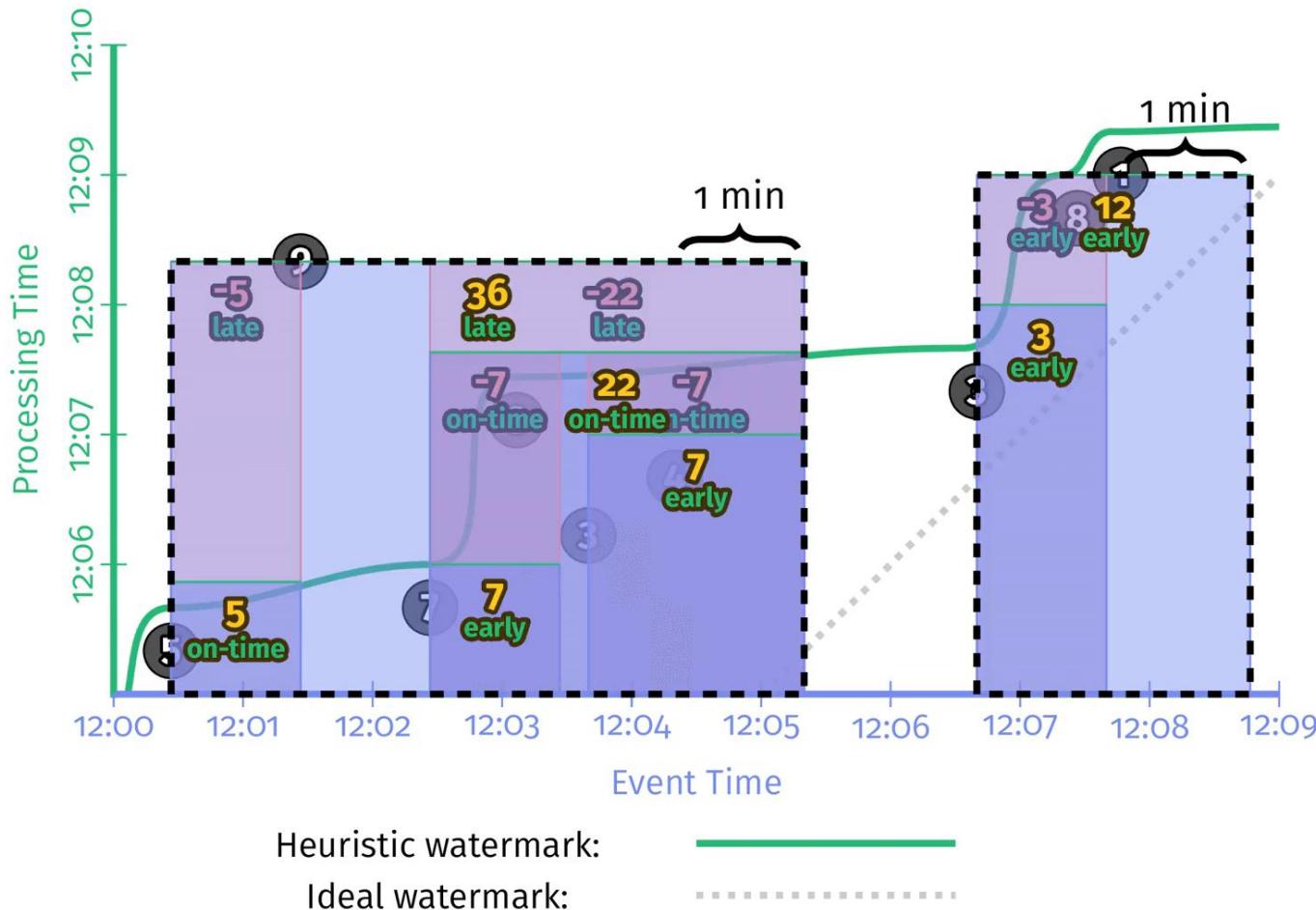
Merged Session #2 - 98 min



Session creation steps

- Phase 0: The gap duration is defined as the length of time between two sessions
- Phase I: Assignment
 - Each element is initially placed in a proto-session window
 - The proto-session starts with the occurrence of the event
 - The proto-session extends for the gap duration
- Phase II: Merging
 - A grouping strategy is defined
 - All eligible proto-sessions are sorted
 - All overlapping proto-sessions are grouped into sessions

Session creation in action

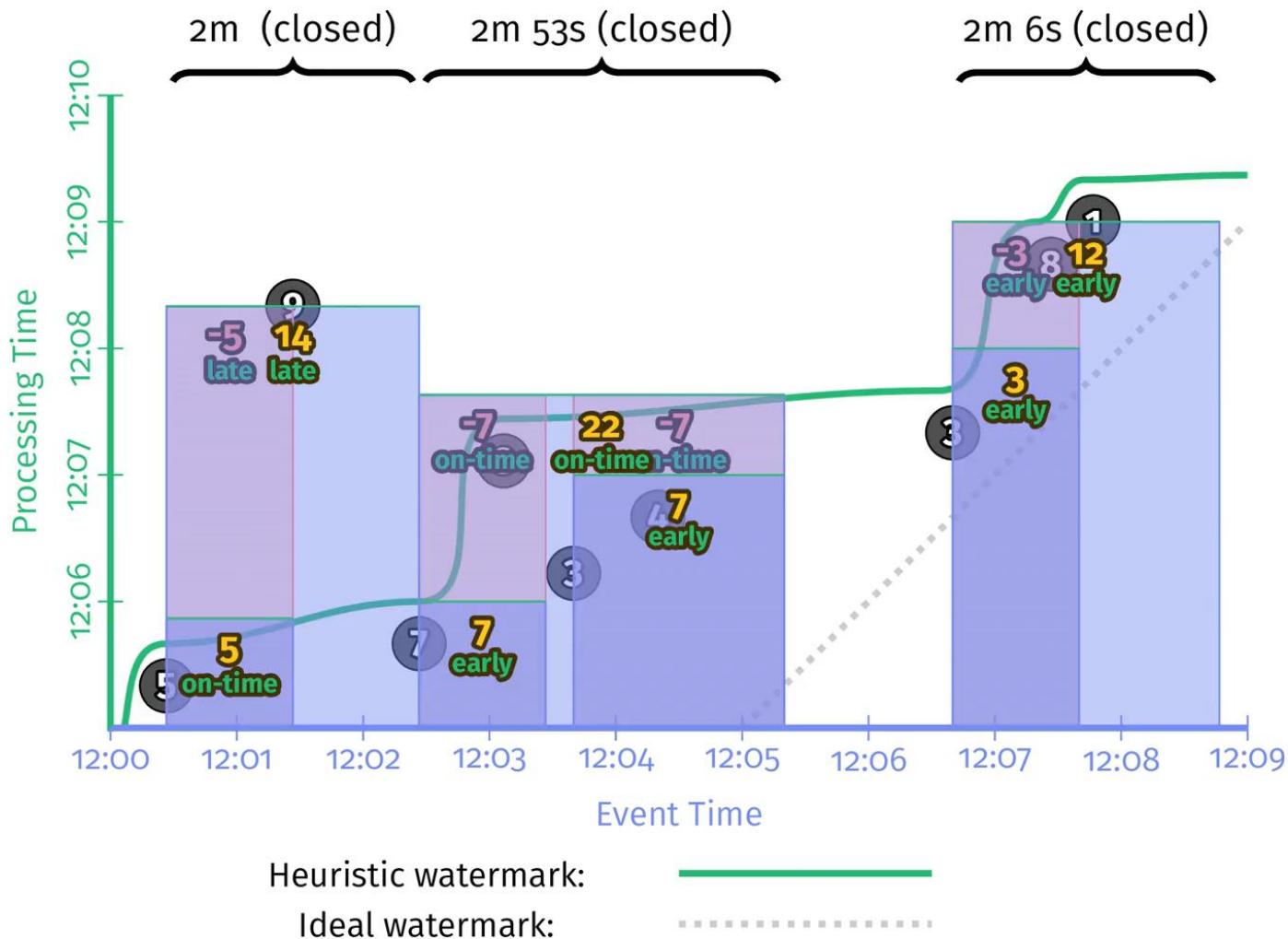


<http://streamingsystems.net/fig>

Variation #1: Bounded sessions

- Bounded sessions are not allowed to grow beyond a predefined size
- The ‘size’ can be defined in time, element count, aggregate value or other dimension
- Example: time-bounded sessions, e.g. a maximum session length of 3 minutes

Bounded sessions in action



<http://streamingsystems.net/fig>

Summary

- Watermarks
 - Perfect vs heuristic
 - Input & output
 - Percentile
 - Processing time
- Advanced windowing
 - Processing-time
 - Event-time
 - Sessions
 - No one-size-fits-all windowing strategy !



Thank you for your attention!



CONSISTENCY IN STREAM PROCESSING PIPELINES

Stream mining (SM)

Imre Lendák, PhD, Associate Professor

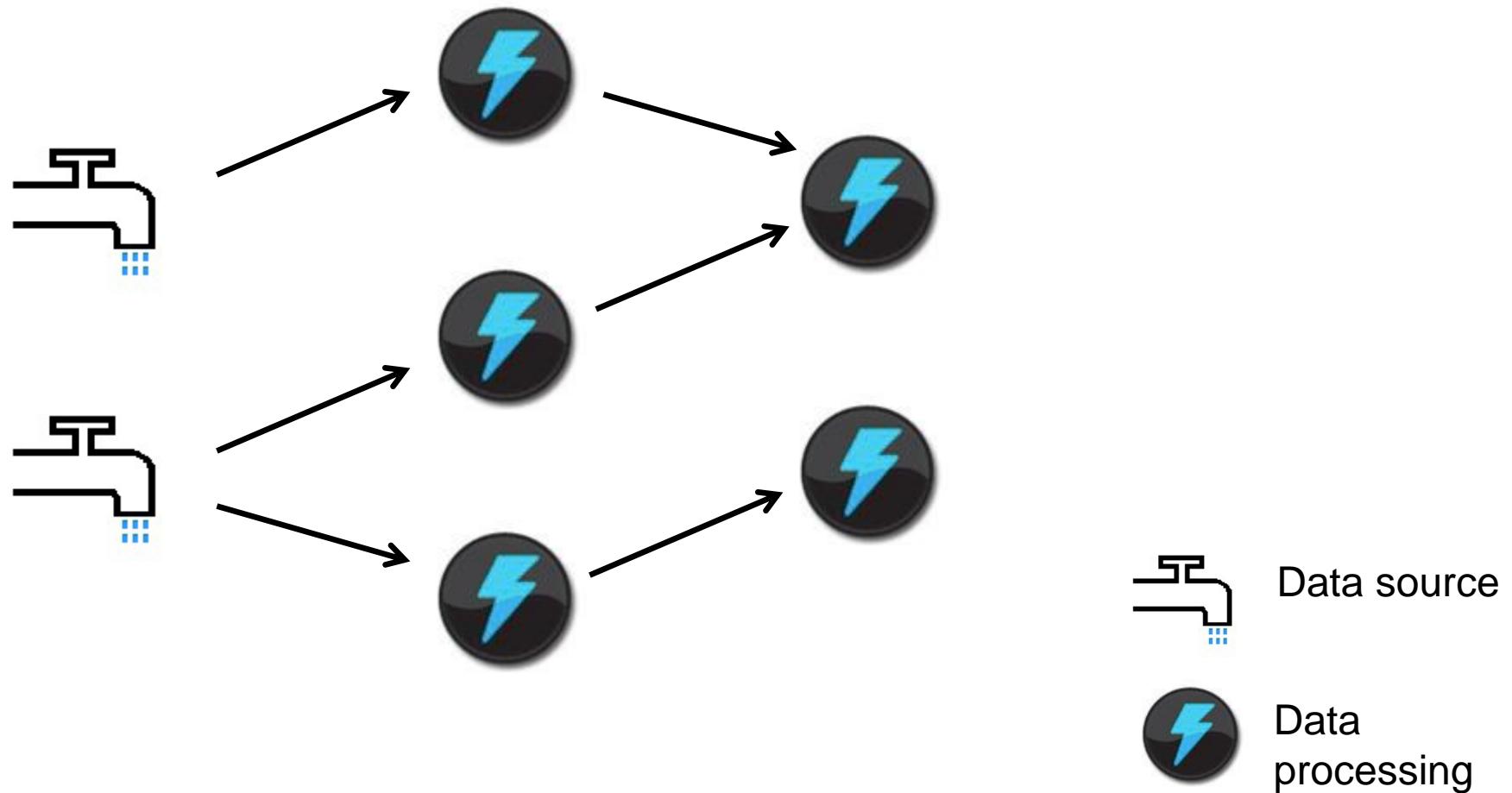
Péter Kiss, PhD candidate

Outline

- Consistency intro
- Consistency ‘levels’
- Checkpoints
- Platforms & consistency

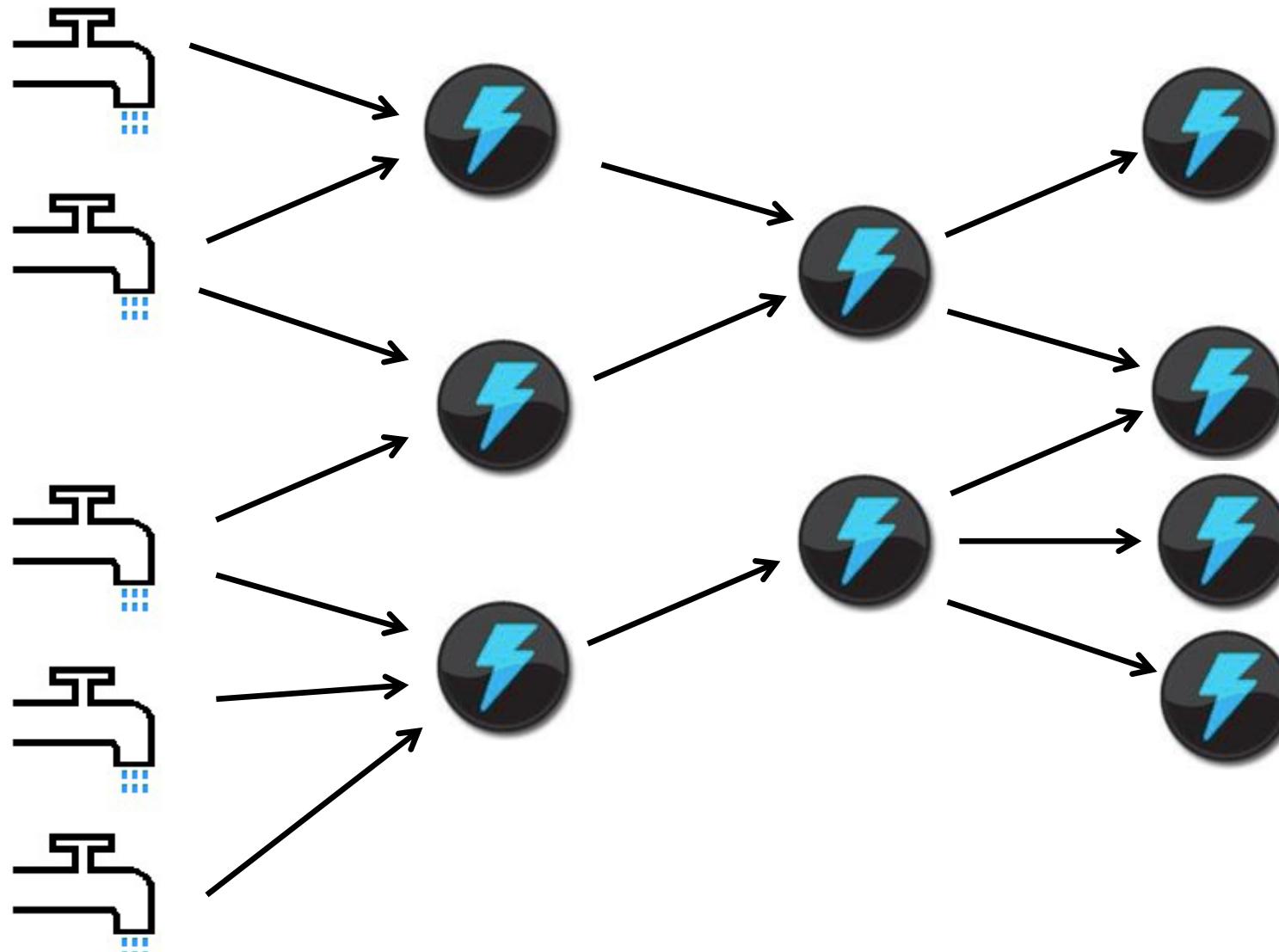


Simple stream processing pipeline



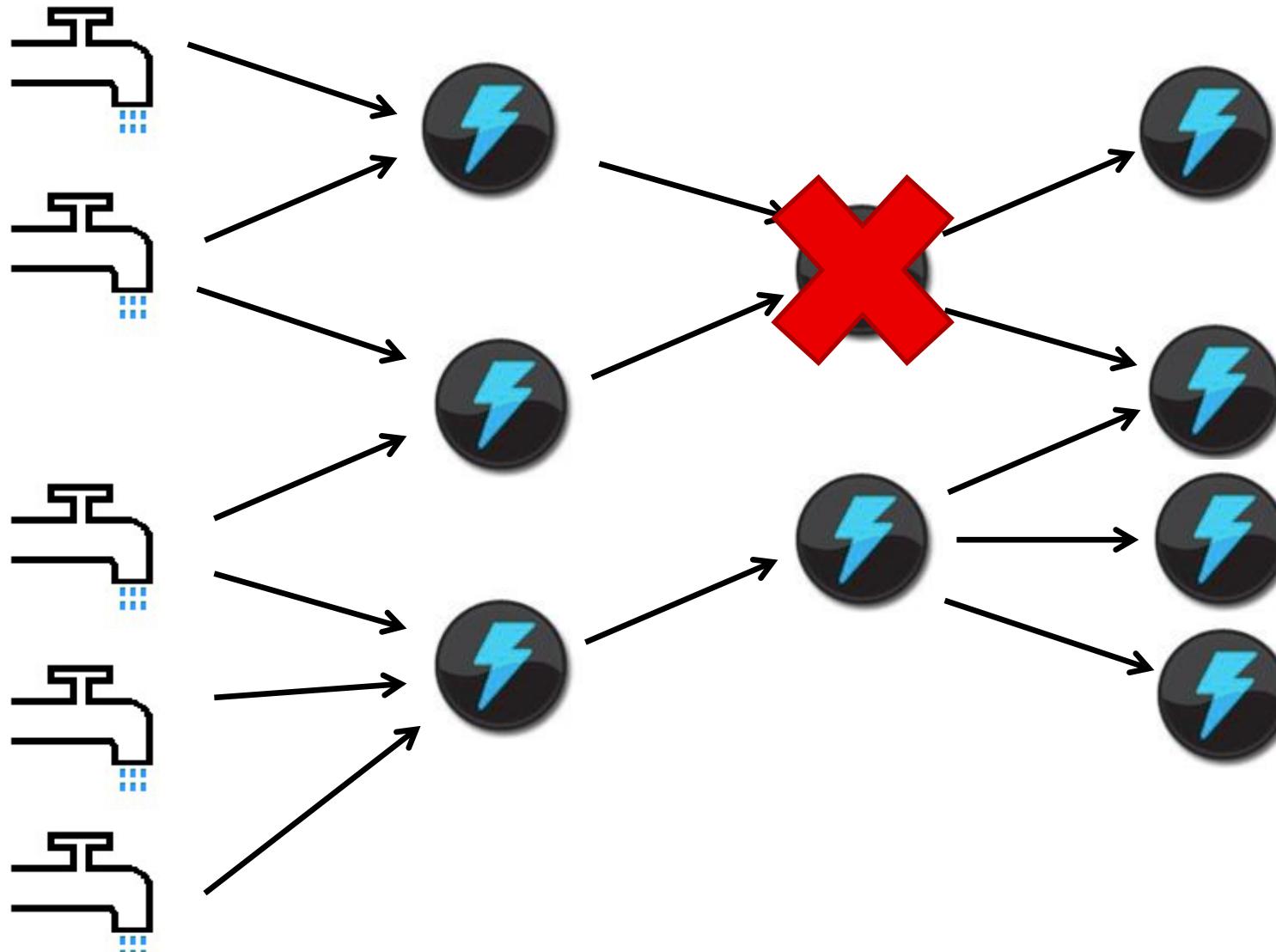
* Apache Storm logic/icons

Complex stream processing pipeline



* Apache Storm logic/icons

Outage in a processing pipeline



* Apache Storm logic/icons

Planned service interruptions

- Hardware maintenance downtimes, e.g. replacement of outdated/failed gear
- Code changes, e.g. migrating from Spark 1.x to a newer version
- Issues can be caused by the human element as well
 - E.g. misconfiguration, configuring/unplugging the wrong device

Unplanned service interruptions

- Hardware failures are inevitable in any distributed system
 - CPU/memory/IO device/power supply failure
 - Communication channel related failure, e.g. cabling, misconfigured switches/routers
- Software failures are inevitable in any distributed system
 - Firmware in networking gear, e.g. switches, routers
 - Operating systems
 - Middleware
 - Applications

Detecting unplanned interruptions

Fault types in DS

- Crash failure
 - Processing node crashes
- Omission failure
 - Input data lost due to communication error
- Timing failure
 - Allowed lateness missed
- Response failure
 - Output buffer write error
- Byzantine fault
 - Processing node generates unplanned outputs

Fault detection types

- Active → based on heartbeat signal
- Passive → based on polling
- Centralized detection
 - Watchdog process monitors processing node statuses
- Distributed detection
 - Detectors assigned to processing nodes
 - Hard to implement
 - Numerous control messages

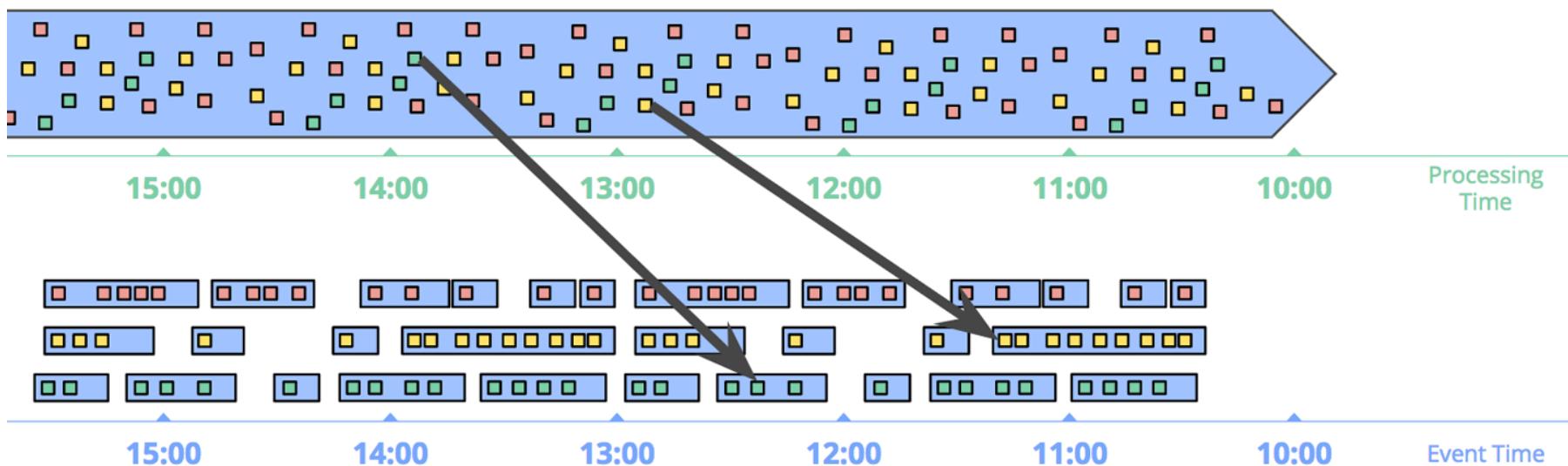
Key challenges

- The assumption of **batch processing systems** (e.g. RDBMS) was that data is stored in persistent storage and processing can be repeated
- The assumption of the **1st generation streaming systems** was that data is ephemeral (i.e. not stored) and processed at least once if received
- Both above use cases:
 - Assume that failures are infrequent
 - Accept the extra cost of recomputation
- The key change brought by the **2nd generation streaming systems** (since ~2011-2012) is that they provide consistency guarantees
 - A few systems provide strong consistency guarantees

TABLES AND STREAMS

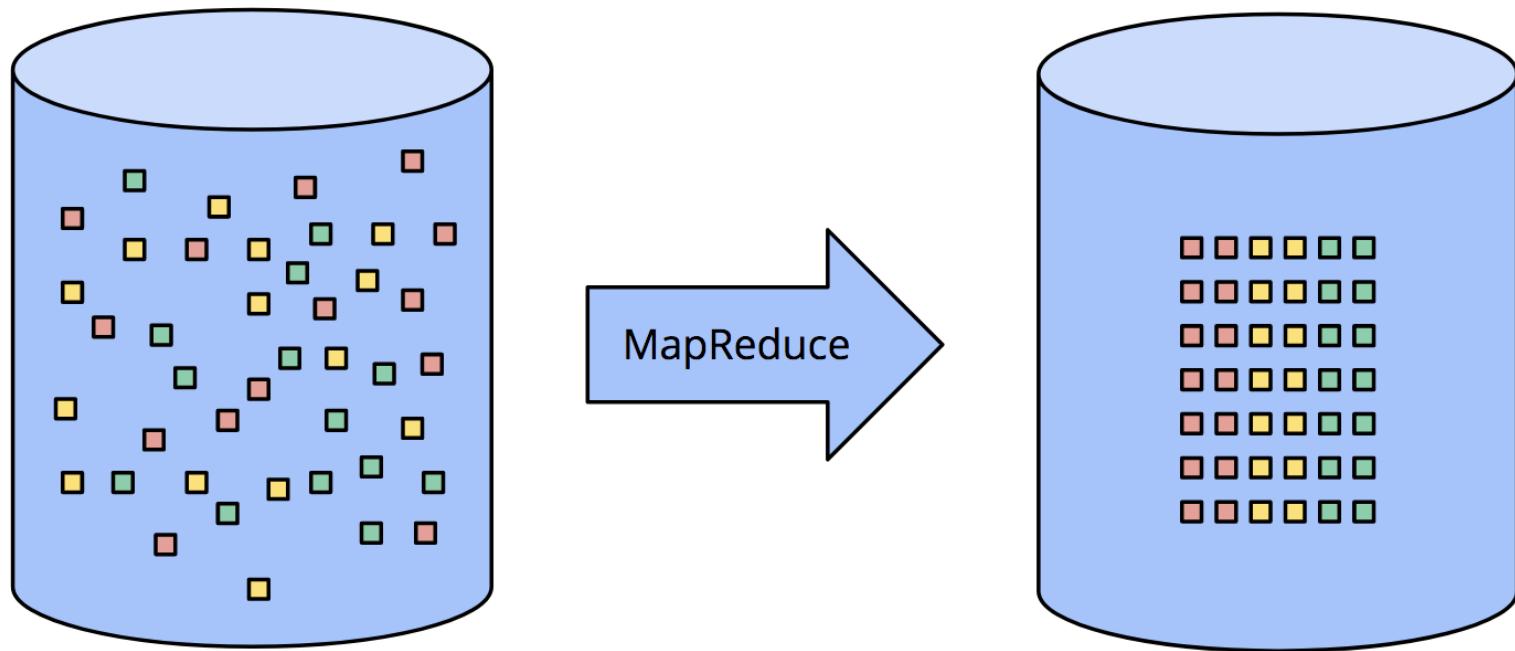
Streams

- **Cardinality:** **Unbounded data** is infinite in size.
- **Constitution:** A **stream** is an element-by-element view of the evolution of a dataset over time.
- **Alternate stream definition:** A data stream is an ordered (not necessarily always) and potentially infinite sequence of data points (e.g. numbers, words, sequences).

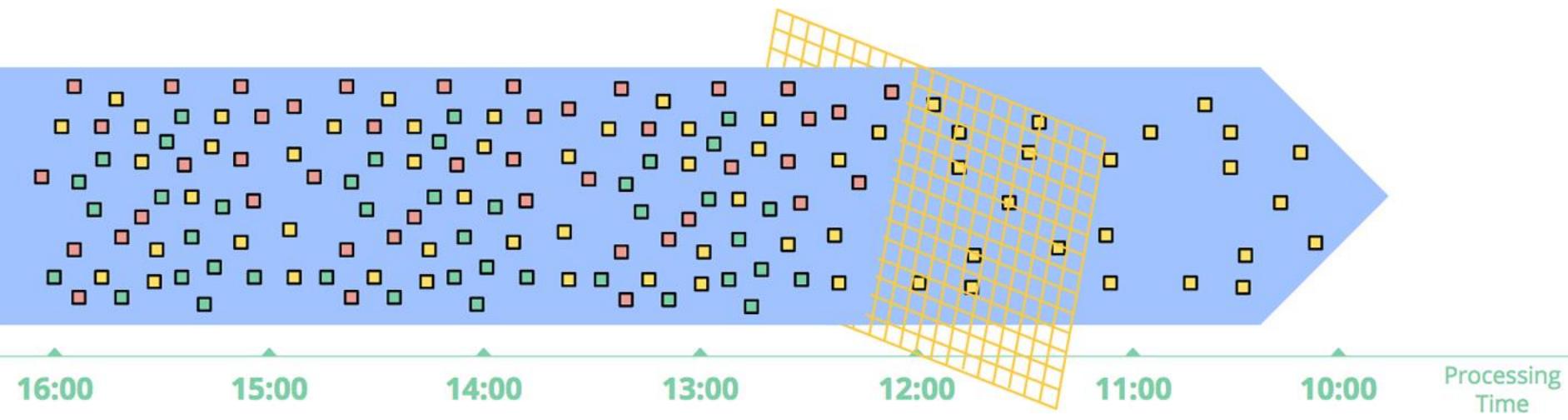


Tables

- **Cardinality:** **Bounded data** is finite in size.
- **Constitution:** A **table** represents a holistic (~complete) view of a dataset at a specific point in time.



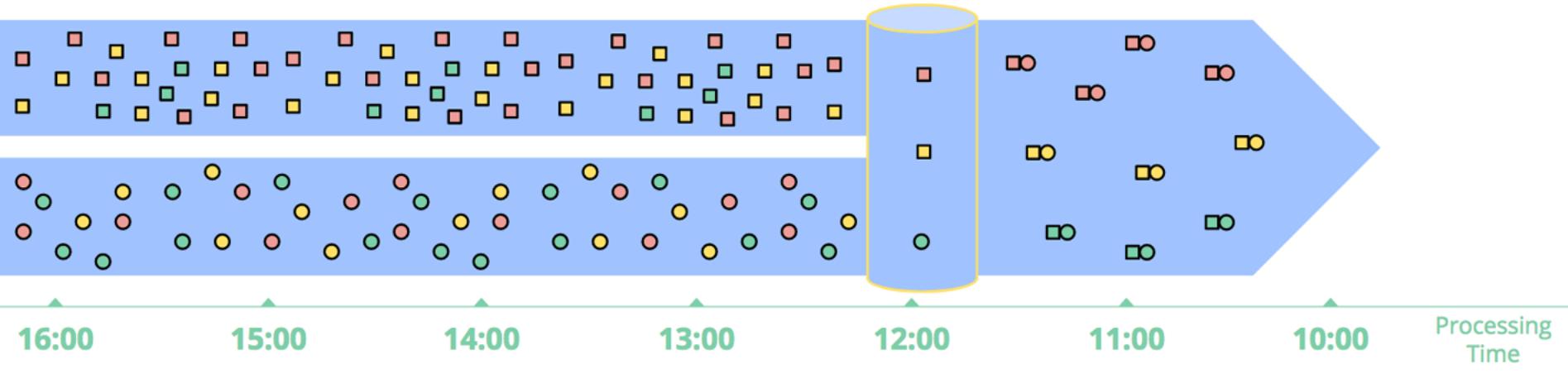
Non-grouping operations



<http://streamingsystems.net/fig/1-5>

- Non-grouping operations in stream processing pipelines accept streams as inputs and transform them into new streams
- **Input:** **stream** of records/observations
- **Output:** transformed **stream** of records

Grouping operations



<http://streamingsystems.net/fig/1-6>

- Grouping operations in stream processing pipelines accept streams, transform and group them
 - E.g. joins, aggregations, ML training, histogram creation
- **Input:** **stream** of records/observations
- **Output:** a **table**

Triggers (revisited)

- **DEF:** Triggers declare when output for a window should happen in processing time, i.e. when data should be processed
- Trigger types:
 - **Repeated update triggers** = most common in streaming systems, generate periodic updates for a window. Updates materialized for
 - Every new record → too many processing → high CPU load
 - After some processing time delay, e.g. 5 minutes
 - **Completeness triggers** = materialize each time a window is (believed to be) complete to certain degree, e.g. ~80% of data, ~90% of data, etc.
 - Allow reasoning about missing and late data

Triggers in table-to-stream conversion

- Grouping operations = **stream-to-table conversion** (see previous slides)
- After (stream) data is grouped into windows, triggers dictate when to process and send data downstream
- In stream processing pipelines the (above) downstream data triggered is again a stream → triggers drive **table-to-stream** conversion in streaming systems
- **Alternate trigger definition:** Triggers are special procedures applied to a that materializes transformed data in response to relevant events
- **Note:** A parallel can be drawn between triggers in streaming systems and in traditional database management systems → they are the same thing in different systems

PROCESSING GUARANTEES

Processing guarantees

- There are four levels of processing guarantees in general distributed systems
 - No guarantee → we do not know if delivery will succeed or not similarly to UDP over IP
 - At-least-once
 - At-most-once
 - Exactly-once
- These processing guarantees apply in streaming systems, which are just a specific type of DS

At-most-once

- In general-purpose distributed systems the at-most-once guarantee means that communication messages are **delivered zero or one times**
 - English: each message may be lost, but not duplicated
- In streaming systems this translates to each data record being processed zero or one times
- **Pro:** high performance, simple
- **Contra:** no correctness guarantees, non-deterministic systems

At-least-once

- In general purpose distributed systems at-least-once means that a communication message will be **delivered at least once to each recipient** (multiple recipients in message-oriented architectures!)
 - Potentially multiple attempts are made at delivering the communication messages → messages might be duplicated
- In streaming systems this means that the data record (i.e. observation) is delivered at least once to each processing node
- It might happen that the message is delivered **more than once!**
- Possible reasons leading to more-than-once:
 - Timing error – the acknowledge message ‘came’ with a delay longer than the configured timeout
 - Upstream processing or source failure
 - ...
- **Pro:** higher level of consistency
- **Contra:** complexity, duplicates, a bit more latency

Exactly-once

- In general-purpose distributed systems exactly-once delivery is a guarantee that each communication message is delivered exactly once to each recipient of that message
 - English: the message cannot be lost and/or duplicated
- In streaming systems an exactly-once guarantee means that each data record in the handled streams will be **processed exactly one time**
- **Pro:** correctness
- **Contra:** complexity, added latency
- Additional reading:
<https://www.ververica.com/blog/high-throughput-low-latency-and-exactly-once-stream-processing-with-apache-flink>

Exactly-once in sources

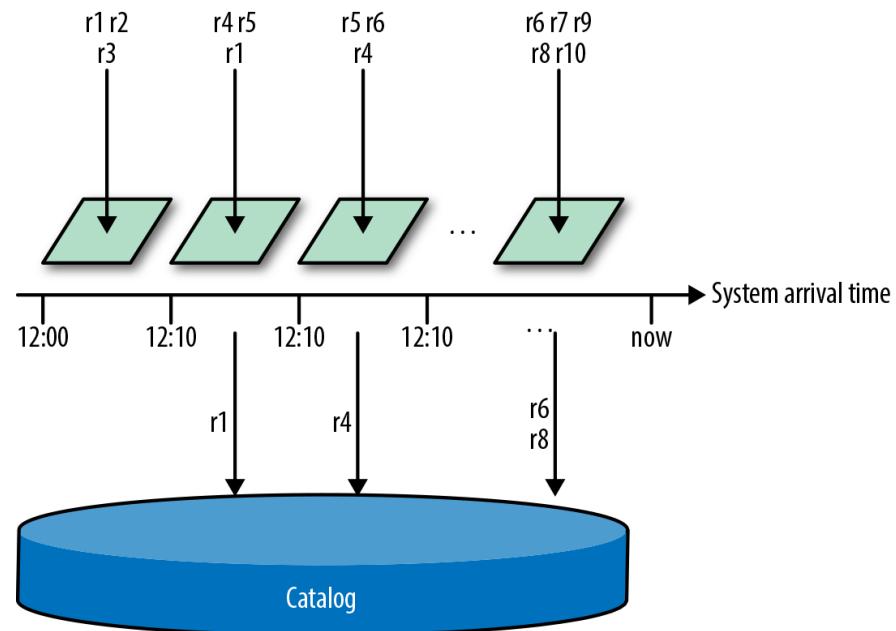
- Exactly-once is relatively straightforward to implement in **deterministic sources**
 - The records in a log file are deterministic, always located at the same offset
 - Some streaming transports also provide such deterministic guarantees, e.g. Kafka (discussed later)
- Exactly-once is a non-trivial task with **non-deterministic sources**, which might emanate different data records and send them to different recipients
 - Publish-subscribe message brokers are usually non-deterministic → if processing node fails, it may or may not send the same messages to the same subscribers in the same order
 - Solution: de-duplication handled in the processing nodes

Exactly-once in sinks

- Built-in sinks in modern streaming systems provide an **exactly-once output guarantee**
 - This means that output records are written to sinks with exactly-once guarantees in the presence of failures in the processing pipeline
- If it is necessary to implement a custom sink, then exactly-once can be guaranteed by implementing a **2-phase commit in custom sinks**
 - Phase I: prepare all necessary output data
 - Phase II: write output data to the sink(s)

Bloom filters vs duplicates

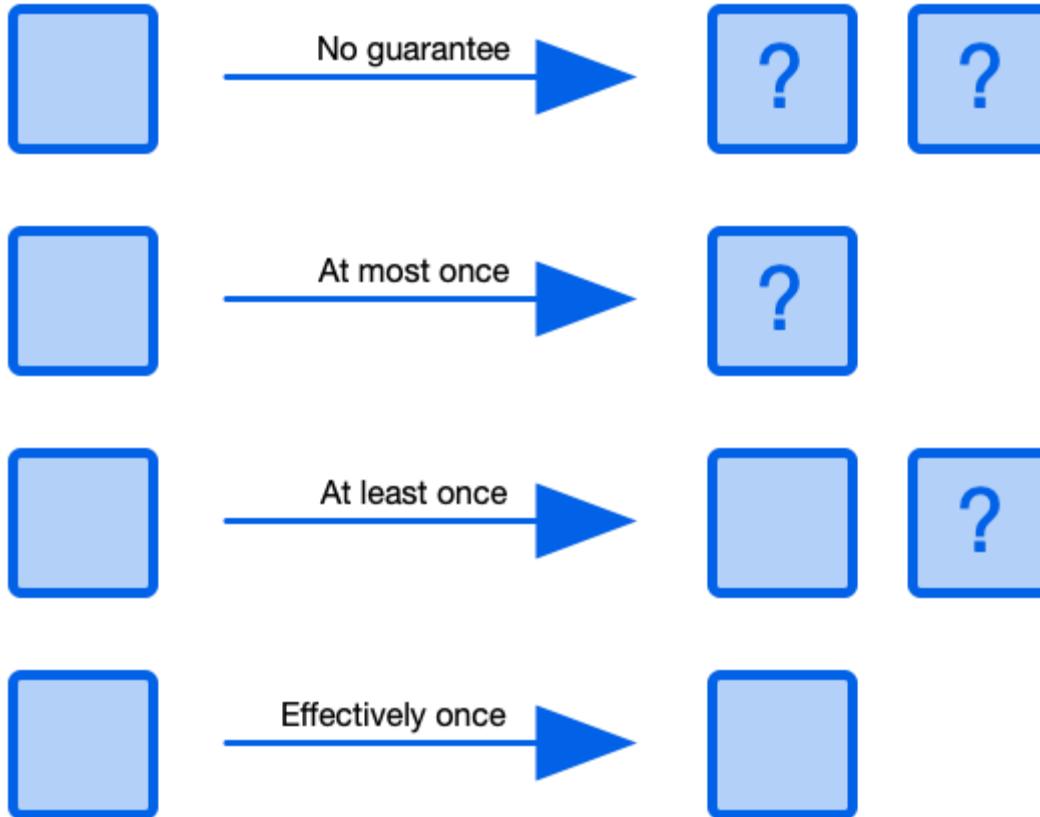
- **DEF:** Bloom filters are simple data structures designed for efficient set membership checks
- Characteristics of Bloom filters
 - They might return false positives
 - They never return false negatives
- In de-duplication they are used as they are
 - capable to always spot non-duplicates, and
 - sometimes trigger on false duplicates (i.e. false positives)
- Bloom filters fill up over time and the false positive rate increases → reconstruct periodically, e.g. every 10 min



Garbage collection

- In streaming systems it is usually not feasible to persist state for an extensive time at each processing stage
- Possible approaches to **garbage collection**, i.e. the deletion of data records which are no longer necessary for correctness
 - Assign **unique, strictly increasing identifiers** assigned to each record
 - **Ingress (processing) timestamps** assigned to each data record → as observed by the processing node
- If ingress timestamps are used, a **garbage collection watermark** can be calculated to trigger the optimal execution of garbage collection, i.e. delete observed inputs

Guarantee levels revisited



<https://medium.com/@andy.bryant/processing-guarantees-in-kafka-12dd2e30be0e>

CHECKPOINT WHAT? AND WHY?

Distributed streaming systems

- **DEF:** **Distributed systems** are complex systems designed to achieve one or more stated goals. They consist of hardware, software, documentation and people.
- Types of distributed systems
 - **Distributed computing systems**, e.g. streaming systems
 - Distributed information systems, e.g. global bank
 - Distributed pervasive systems, e.g. large-scale sensor network → Internet of Things
- Algorithms run in distributed computing systems
 - The processes executing the **basic algorithm** exchange basic messages and execute part of the distributed computation → processing nodes
 - The processes executing the **control algorithm** exchange control messages and oversee the basic algorithm's execution

Checkpoint intro

- **DEF:** Checkpoints are defined as snapshots of a complete or partial state of a distributed system
 - The DS consists of process nodes and communication channels and its state consists of process node and channel state fragments
 - Checkpoints allow **backward recovery** after failures
- It is a challenging task to create a checkpoint in any DS
 - **Storage:** Where to store the checkpoint?
 - **Timing:** When to initiate and how to sync the checkpoint?
 - **Performance:** How to minimize the additional load in the DS during the checkpoint creation process?
- Checkpoint creation types:
 - **Autonomous:** process nodes create checkpoint fragments separately (i.e. not synced)
 - **Coordinated:** selected elements create checkpoint fragments in a coordinated fashion

Message logging

- **DEF:** **Message logging** in distributed systems is the process of storing communication messages which alter the state of the DS
 - Note: messages storage in persistent storage
- Recovery can be a combination of **checkpoint + message logging** since the last checkpoint
 - Phase I: Revert the system to the last checkpoint
 - Phase II: Replay all messages since the last (complete or partial) checkpoint
- **Preconditions:**
 - the underlying system is **deterministic** → message M received by processing node P in state S1 will always transition P to state S2
 - all (relevant) communication messages (i.e. data records in streaming systems) are **persisted**

Checkpoint levels

- Processing nodes might implement different checkpointing strategies
- Any such checkpointing strategy needs to balance between two extremes
 - Always-persist-everything → good for consistency, bad for efficiency
 - Never-persist-anything → bad for consistency, good for efficiency
- We will analyze the following strategies
 - Always-persist-everything
 - Persist groupings/increments
 - Generalized state

Always persist everything

- In the ‘always-persist-everything’ scenario the stream processing pipeline is capable to store
 - each piece of data
 - at each processing node
- How does it work? Atomic elements of the data stream are appended to the list of all prior elements received
- **Pro:** excellent consistency
- **Contra:**
 - Not a viable solution in real-life scenarios with extremely high loads, e.g. monitoring websites with extremely large numbers of users
 - High CPU load with periodic processing triggers

Persist groupings/increments

- In the persist groupings/increments strategy the processing nodes persist groupings of the inputs
 - Also known as ‘incremental combining’ in the Streaming systems book
 - E.g. a SUM processing stage can store the partial sum of observed inputs and the number of inputs
- It is necessary that the operation performed by the processing stage is both commutative and associative
 - $\text{OP}(a,b) == \text{OP}(b,a)$
 - $\text{OP}(\text{OP}(a,b),c) == \text{OP}(a, \text{OP}(b,c))$
- These use cases are excellent for parallelization on multiple physical or virtual computers

Generalized state

- If it is infeasible to buffer all data records and/or the (processing) operation is not commutative and/or associative we need more flexibility
→ generalized state with (the below) three levels of flexibility
- Flexibility in the use of **data structures** → in generalized state the process node is not limited to storing a list, number or pair of numbers. It should be allowed to store different data types, e.g. set, map
- Flexibility and **write and read granularity** → the processing node should be capable to choose the amount and type (see above) of data stored for optimal efficiency. It should be also possible to do large read/write in parallel.
- Flexibility in **scheduling** processing → ability to choose the time at which processing occurs:
 - **Completeness triggers** bound to watermarks (event time)
 - **Repeated update triggers** in processing time
 - **Timers** bind processing to a specific point in time in either of the two supported time domains

PLATFORMS AND CONSISTENCY

Spark Streaming & consistency

- Apache Spark Streaming implements a microbatch architecture for stream analysis
 - Under the hood Spark Streaming groups the data into a series of Resilient Distributed Datasets (RDDs) which are processed sequentially → all this happens in the processing time domain
 - Spark implements an **exactly-once guarantee via batch processing**
- **Pro:** exactly-once
- **Contra:** latency (due to batch processing) → clever tuning necessary to avoid long delays

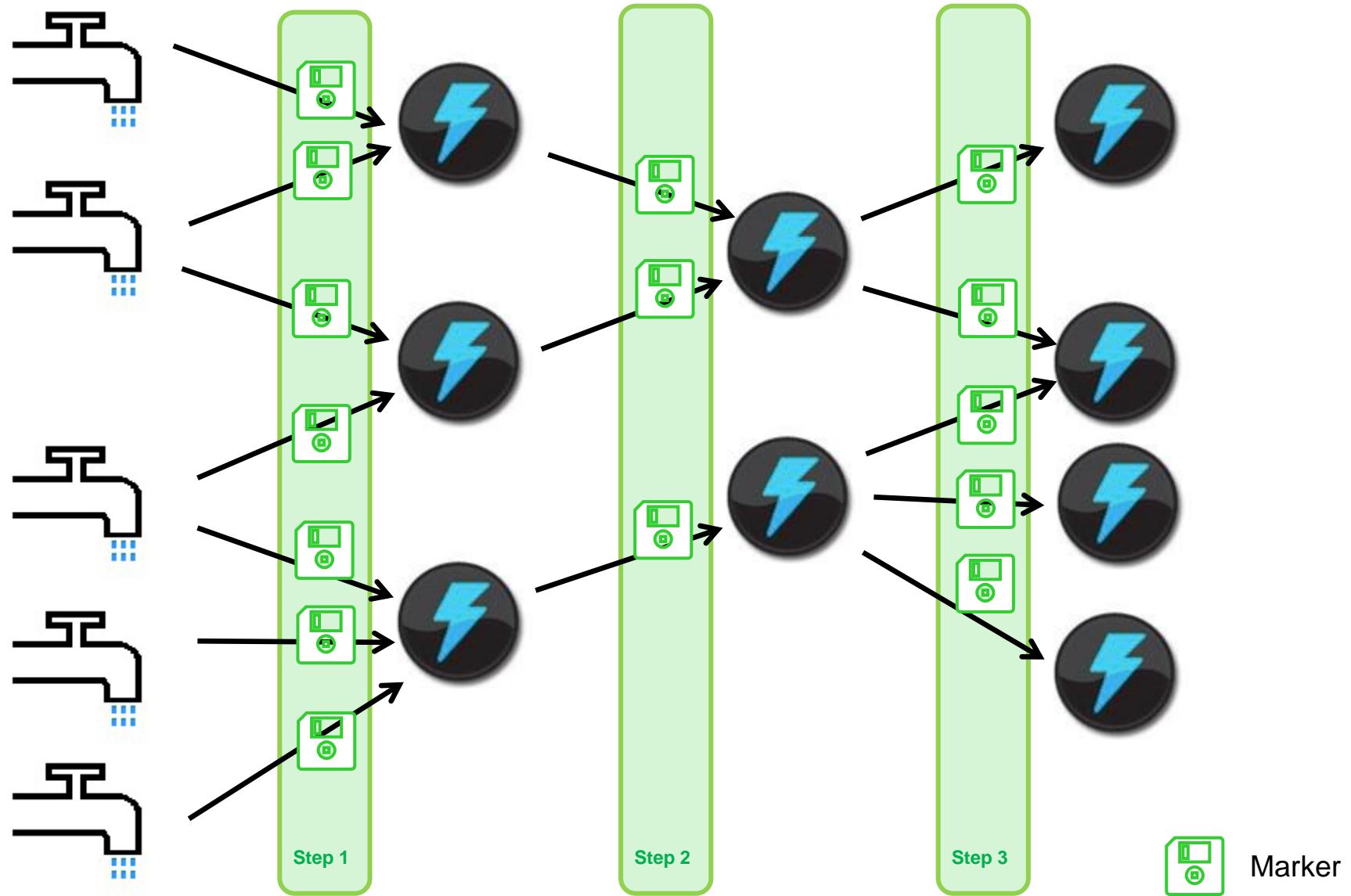
Kafka & consistency

- **DEF:** Kafka is a **(streaming) transport layer**, not a full-blown stream (processing) framework
 - Kafka is a persistent streaming transport implemented as a set of partitioned logs
- Kafka introduced **durable, replayable input sources** in 2011-2012 (among the 1st of its kind)
- Prior to Kafka streaming systems handled streams as ephemeral data, which meant:
 - Lack of durability → streams not persisted
 - Lack of replayability → streams could not be (easily) repeated for testing, (ML) training or during fault recovery
- **Note:** Kafka is often used as part of modern stream processing pipelines as a transport layer

Flink & consistency

- Apache Flink processing pipelines periodically compute **consistent snapshots** → point-in-time state of the entire stream processing pipeline
 - Flink snapshots are computed without halting the base algorithm
 - Assumption: processing tasks are statically allocated to ‘workers’
- Flink **adds markers** to data records flowing through the stream processing pipeline which aid its snapshot creation process
- When a processing node receives a snapshot marker, it **persists its state** and **hands the marker** to downstream processing nodes (via emitted data records)
 - The snapshot is completed when all nodes do the above
- **Sinks wait until snapshot completion** before sending their results to the outside world
- **Pro:** exactly-once in a truly streaming system manner
- **Contra:** added latency

Marker-based snapshot creation



Summary

- Consistency levels
- Tables and streams
- Checkpoints
- Platforms & consistency



Thank you for your attention!

Data Stream Mining

Péter Kiss, Teaching Assistant

Eötvös Loránd University, Budapest, Hungary

October 7, 2020

Data Stream

Definition

A data stream is an ordered (not necessarily always) and potentially infinite sequence of data points.

$$x_1, x_2, x_3, \dots,$$

where x_i s are tuples constituted of numbers, words, sequences, etc.

Such streams are ubiquitous and we are always around them. For example:

- Click streams
- Sensor measurements
- Satellite imaging data
- Power grid electricity distribution
- Banking/e-commerce transactions

Examples of Data Streams :

Top chat ▾

sparaj Bhatra 😊😊

ruchi verma verma dhananjay I asked something ..
PLEASE try to answer me

mahesh mehra ruchi verma kya kr rahi h 😊

mahesh mehra [message retracted]

Knowlage and Lesson Pakistan zindabad

Dhananjay Patel ruchi verma lots of time 😊 avi v to
wohi face 😊 kr raha hu ma'am g

Tejram Patel hi ruchi

mahesh mehra Pakistanmurdabd

Welcome to live chat! Remember to guard your

Figure: Youtube live comments

Data Stream Mining Algorithms

Under stream mining algorithms we usually involve some summarization of the stream into a model.

Tasks related to data streams (planned):

- **Sampling and Filtering**
- Counting distinct elements
- estimating moments
- Clustering
- Frequent pattern mining
- Change/Concept drift detection
- Time series

Literature

Rajaraman, Anand, and Jeffrey David Ullman. Mining of massive datasets. Cambridge University Press, 2011.

Gama, Joao. Knowledge discovery from data streams. CRC Press, 2010.

Characteristics of data streams

A data stream has several distinguishing features such as:

- Unbounded Size
 - Transient (that means it lasts for only few seconds or minutes)
 - Single-pass over data
 - Only summaries can be stored
 - Real-time processing (in-memory)
- Data streams are not static
 - Incremental algorithms: incorporate new data into the model
 - Some models : decremental updates - discard influence of data points if they are outdated/outliers
 - Concept Drifts - Temporal order may be important

Adaptive algorithms

Desirable properties for learning high speed, time-changing data streams:

- Incrementality
- on-line learning
- constant time to process
- single scan over training set
- Ability to handle concept drift
- limited computational resources
- anytime protocol -
- support for distributed data gathering and processing

Querying streams

Standing queries

Permanently executed, the process itself have been designed to answer them.

Examples - Ocean-surface temperature:

- ① Alert when temperature exceeds 25 Centigrade
- ② Alert when average of most recent 24 measurements exceeds 25 Centigrade
- ③ Maximum temperature ever recorded

Querying streams

Ad-hoc queries

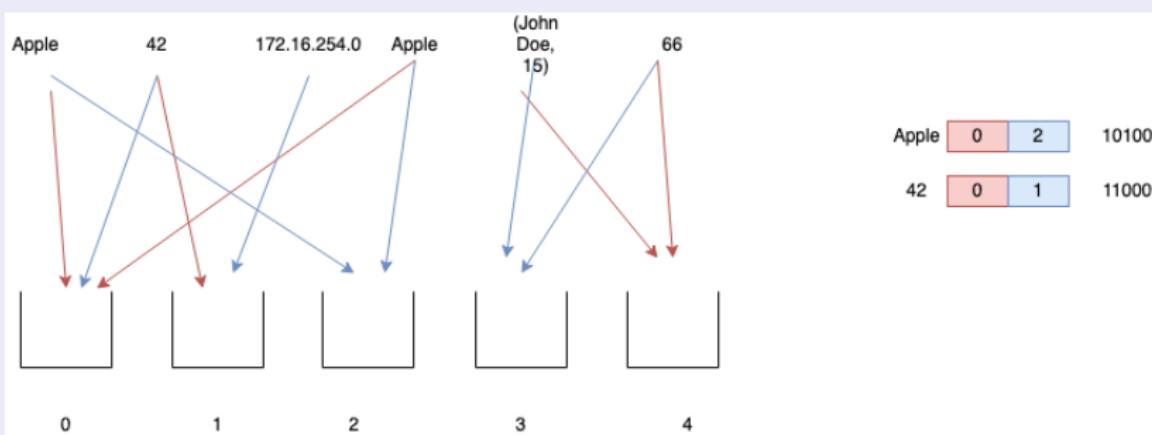
A question asked once about the current state of a stream. If we do not store all the data items we have seen, we cannot expect to be able to answer arbitrary queries.

Techniques to answer

If we have some idea however what kind of queries may be asked we can prepare for them by storing appropriate summaries of streams.

- ① Sliding windows - collect all the data arrived in a given time period/or a range of most recent data
- ② Sampling - maintain an appropriate representation of all data.

Hashing



Hashing

"*Hashset*" for example in Java

Definition

A hash function h takes some value (of any data type) the so-called *hash-key*, and it assigns it to one of B buckets (that is produces a *bucket number*).

The key requirement of a hash function is that if *hash-keys* are drawn randomly from a population, h should send similar number of values to each of the B buckets.

Hashing

Example : Hashing integer keys

simple way is the division method : $h(x) = k \bmod B$, that returns the remainder when x divided by B .

$$36 \% 8 = 4$$

$$18 \% 8 = 2$$

$$72 \% 8 = 0$$

$$43 \% 8 = 3$$

$$6 \% 8 = 6$$

[0] [1] [2] [3] [4] [5] [6] [7]

| | | | | | | | |
|----|--|----|----|----|--|---|--|
| 72 | | 18 | 43 | 36 | | 6 | |
|----|--|----|----|----|--|---|--|

source and more methods: <http://faculty.cs.niu.edu/~freedman/340/340notes/340hash.htm>

Hashing

Example : Hashing integer keys

simple way is the division method : $h(x) = k \bmod B$, that returns the remainder when x divided by B .

$$36 \% 8 = 4$$

$$18 \% 8 = 2$$

$$72 \% 8 = 0$$

$$43 \% 8 = 3$$

$$6 \% 8 = 6$$

[0] [1] [2] [3] [4] [5] [6] [7]

| | | | | | | | |
|----|--|----|----|----|--|---|--|
| 72 | | 18 | 43 | 36 | | 6 | |
|----|--|----|----|----|--|---|--|

Problem

What if the 4th element is 44?

Hashing

Hashing non-integer keys

All data type have values that always can be translated to bits, and bit series always can be interpreted as integers.

ASCII

| ASCII Code Chart | | | | | | | | | | | | | | | |
|------------------|-----|-----|-----|-----|-----|-----|-----|-----|----|-----|-----|----|----|----|-----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| ! | " | # | \$ | % | & | ' | (|) | * | + | , | - | . | / | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| P | Q | R | S | T | U | V | W | X | Y | Z | [| \ |] | ^ | _ |
| . | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| p | q | r | s | t | u | v | w | x | y | z | { | | } | ~ | DEL |

String hashing

For small B : sum the values

For big B : concatenate digits



Motivation

General Problem

selecting a subset of a stream so that we can ask ad-hoc queries about the selected subset and have the answers be statistically representative of the stream as a whole.

Example:

Domain : Stream of queries arrived at a search engine

Query: What fraction of user's queries are repeated?

Repeated search example

Let us assume that we can store 10% of the searches. How can we answer the question?

Repeated search example

Let us assume that we can store 10% of the searches. How can we answer the question? Idea: save each search with 10% probability. Law of Large Numbers → about 1/10 of the searches of each particular user will be stored.

Repeated search example

Let us assume that we can store 10% of the searches. How can we answer the question? Idea: save each search with 10% probability. Law of Large Numbers → about 1/10 of the searches of each particular user will be stored.

s searches once , d twice, and no searches more times

The correct answer:

$$\frac{\text{no. of double search-queries}}{\text{all search queries}} = \frac{d}{(d+s)} \quad (1)$$

Repeated search example

But what we get with our method?

Repeated search example

But what we get with our method? Probabilities of a single search appears in summary 1/10

Probability of double one appears once :

$$1/10 \cdot 9/10 + 9/10 \cdot 1/10 = 18/100$$

Probability that both occurrence of a double search appears:

$$1/10 \cdot 1/10$$

Thus the result:

$$\frac{d/100}{s/10 + 18d/100 + d/100} = \frac{d}{(10s + 19d)} \neq \frac{d}{d+s} \quad (2)$$

what does that mean?

The sample was NOT REPRESENTATIVE!!

Repeated search example

How we obtain a representative sample?

Instead of taking the proportion of searches, take all the searches
1/10 part of the users!

When a new user appears generate a random number from 0 to 9,
and if it let us say 0 then store the queries of the user.

Repeated search example

How we obtain a representative sample?

Instead of taking the proportion of searches, take all the searches
1/10 part of the users!

When a new user appears generate a random number from 0 to 9,
and if it let us say 0 then store the queries of the user.

What is the problem with this?

Repeated search example

What is the problem with this?

How many users can we store, and how much time does it takes to decide whether the given user's activity is recorded?

Repeated search example

What is the problem with this?

How many users can we store, and how much time does it takes to decide whether the given user's activity is recorded?

Solution: HASH the user names into 10 buckets, and if it goes to bucket 0, store the search query.

General Sampling Problem

Fixed fraction sample

A stream consists of tuples of some components (user query, time for example), and some subset of these components are *key* components.

Our goal is to get a representative sample of some size a/b . Then hash key value x for each tuple into b buckets, and store the tuple if $h(x) < a$.

Maintain sample of bounded size with time:

Very large number of buckets B , and threshold variable t . At the beginning $t = B - 1$. store tuple if $h(K) < t$. If allocated space is about to run out , throw away all tuples with $h(K) = t$, then $t --$.

Filtering

Problem

Select interesting data points from a huge stream.

Filtering

Problem

Select interesting data points from a huge stream.

Easy

The easy problem when the selection criterion can be calculated, for example the tuple has an attribute with a given value.

Filtering

Problem

Select interesting data points from a huge stream.

Easy

The easy problem when the selection criterion can be calculated, for example the tuple has an attribute with a given value.

Hard

The hard problem: criterion based on membership in a group.

Example : Checking if a user-id is present

Can you guess how gmail checks if a user-id is available?

Example : Checking if a user-id is present

Can you guess how gmail checks if a user-id is available?

Ans: **Bloom Filter**

Example : Checking if a user-id is present

Can you guess how gmail checks if a user-id is available?

Ans: **Bloom Filter**

The key idea: Maintain the following:

- ➊ A bit-array of length n .
- ➋ A number of hash functions $h_1(), h_2(), \dots, h_k()$.
- ➌ A set S of m key values.

Intuition

We iterate over all the members at training, and hash each one of them onto the array. Any bit of the array, that has been produced by any hash function for any element will be set to 1. Thus if we test an element, and if it maps to all 1s we assume, that it is a member of S , otherwise we can be certain that it is not.

Bloom Filter

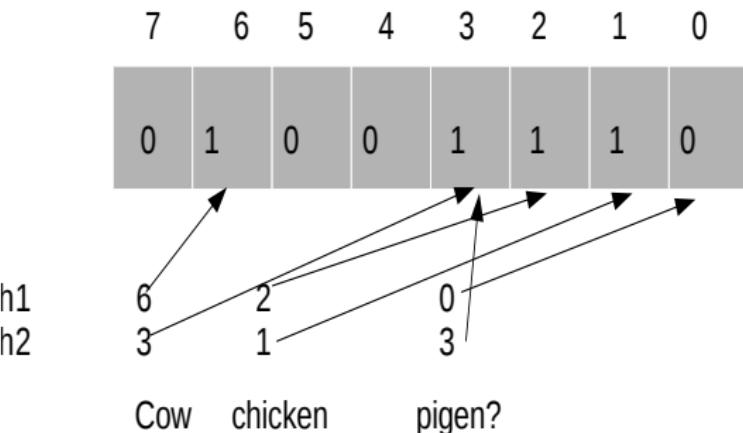


Figure: Hashing items in a bit-array. h_1 and h_2 are hash functions.

Bloom filter can answer if an item passes through it or not. That is, it gives **guaranteed answer** if a user id is available and probabilistic answer if a user-id is NOT available by measuring the collision. If at least one bit is not set, user-id is available and if all bits are set, it means either user-id *may* be available.

Limitations of bloom filter

- ① Size of the bloom filter is very important. Smaller bloom filter, larger false positive (FP) and vice versa.
- ② Number of hash functions? Larger number of hash functions, quicker the bloom filter fills as well as slow filter. Too few hash functions, too many FPs unless all are *good* hash functions.

Limitations of bloom filter

Probability of false positives

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S , and k number of hash function.

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S , and k number of hash function.

Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.

After throwing all the darts, how many target we expect to be hit at least once?

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S , and k number of hash function.

Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.

After throwing all the darts, how many target we expect to be hit at least once?

Probabilities:

- given dart does not hit a given target:

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S , and k number of hash function.

Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.

After throwing all the darts, how many target we expect to be hit at least once?

Probabilities:

- given dart does not hit a given target: $\frac{(x-1)}{x}$ (the probability that one of the other target is hit)
- none of the y darts hit a given target:

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S , and k number of hash function.

Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.

After throwing all the darts, how many target we expect to be hit at least once?

Probabilities:

- given dart does not hit a given target: $\frac{(x-1)}{x}$ (the probability that one of the other target is hit)
- none of the y darts hit a given target: $(\frac{(x-1)}{x})^y$

$$\left(\frac{(x-1)}{x}\right)^y = \left(1 - \frac{1}{x}\right)^{x\frac{y}{x}}. \text{ With } (1 - \epsilon) \approx 1/e \text{ for small } \epsilon:$$

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

Limitations of bloom filter

Probability of false positives

Limitations of bloom filter

Probability of false positives is function of n bit array length, m the number of members of S , and k number of hash function.

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S , and k number of hash function.

Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.

Found that the probability of no darts hitting a particular target:
 $e^{-y/x}$

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S , and k number of hash function.

Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.

Found that the probability of no darts hitting a particular target:
 $e^{-y/x}$

Each of the n bit of the filter is a target, and each k hash value of the m member of S is a dart. ($y = k \cdot m$, and $x = n$) The probability of a bit is

- 0 is the same as it got no hit : $e^{-km/n}$
- 1 is the same as it got at least one hit : $1 - e^{-km/n}$

Limitations of bloom filter

Probability of false positives

Limitations of bloom filter

Probability of false positives is function of n bit array length, m the number of members of S , and k number of hash function. We want the probability for any bit is 0 to be as high as possible, otherwise the probability for nonmember element (email address) will be hashed do any 0 elemnt will be too low. That is, if it maps to all 0-s we will say that it is already occupied and reject

Contd...

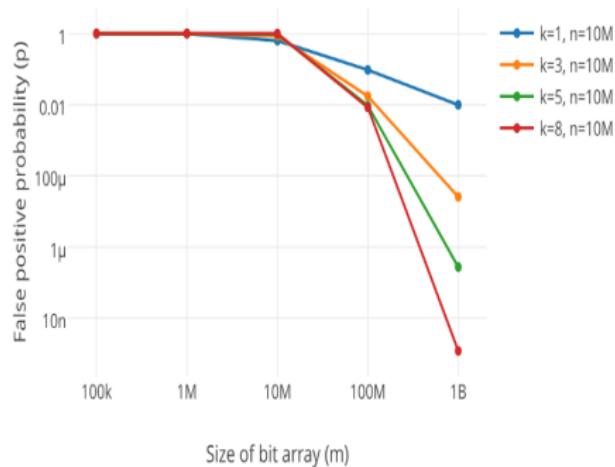


Figure: Number of hash functions vs FPR. Source:
<https://www.semantics3.com/blog/use-the-bloom-filter-luke-b59fd0839fc4/>

Recap
oooo

The Problem
oooo

Preliminaries
ooooo

Sampling
oooooo

Filtering
oooooooo●

Bibliography I

Data Stream Mining- Lecture 2

Chandresh Kumar Maurya, Assistant Professor, Péter Kiss,
Teaching Assistant

Eötvös Loránd University, Budapest, Hungary

October 14, 2020

Approximation

Most of the streaming methods are approximate. (Why?) So, need good approximation techniques such as:

- ϵ -approximation: answer is within some small fraction ϵ of the true answer.
- (ϵ, δ) -approximation: the answer is within $1 \pm \epsilon$ of the true answer with probability $1 - \delta$

Approximation

Most of the streaming methods are approximate. (Why?) So, need good approximation techniques such as:

- ϵ -approximation: answer is within some small fraction ϵ of the true answer.
- (ϵ, δ) -approximation: the answer is within $1 \pm \epsilon$ of the true answer with probability $1 - \delta$

trade-off between the size of summary, and the accuracy, typically space requirements:

$$O\left(\frac{1}{\epsilon^2} \log(1/\delta)\right)$$

Count distinct elements
○●

Frequency of elements
○○○○○○○○○○

Moments
○○○○○○○○○○○○

Frequency of elements - Count Min-Sketch

The Problem

Count frequency of A in the stream: A, B, C, A, A, C,...? number of packets from each IP seen at a server

Frequency of elements - Count Min-Sketch

The Problem

Count frequency of A in the stream: A, B, C, A, A, C,...? number of packets from each IP seen at a server

What accuracy we want?

First specify: desired probability level δ , and admissible error ϵ

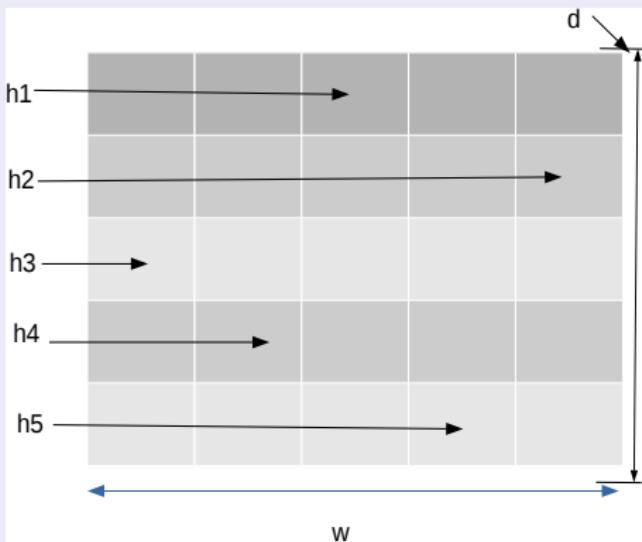
Sketch Table

create table CM of d rows and w columns, full of 0s

$$w = 2/\epsilon, d = \lceil \log(1/\delta) \rceil$$

Frequency of elements - Count Min-Sketch

MC Table



Method

define d $h(\cdot)_i$, ($i \in 1..d$) hash function - one corresponding to one row of CM
each entry x of a stream is mapped to one cell per row using the corresponding $h()$, and increase the number of that cell.

Frequency of elements

Answer

ESTIMATE (y)

$$\begin{cases} h_1(y)=2 \\ h_2(y)=6 \\ h_3(y)=4 \end{cases}$$

| | | CMS | | | | | | | |
|---|---|-----|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 3 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

$$E(y) = \min(1, 3, 1) = 1$$

CORRECT ESTIMATE

ESTIMATE (x)

$$\begin{cases} h_1(x)=3 \\ h_2(x)=6 \\ h_3(x)=1 \end{cases}$$

| | | CMS | | | | | | | |
|---|---|-----|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 3 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

$$E(x) = \min(5, 3, 5) = 3$$

OVERESTIMATE!!!
(CORRECT IS $x=2$)

Question: How many times have we seen the given key x (IP)?

Get the set of cells where some of the $h()$ maps x , and take the minimum:

Unique values in a stream

How many different values have we seen in the stream?
 $\{0, 1, \dots, M - 1\}$ and stream looks like:

0, 1, 0, 0, 0, 1, 1, 2, 3, ...

Here $M = 3$. Trivial if you have space linear in M (why?). Can you do better? e.g. using space only $\log(M)$. Answer: Use Flajolet and Martin Algorithm (1985).

Flajolet - Martin Algorithm

Algorithm 1: Flajolet and Martin (FM) Algorithm

Input: stream M

Output: Cardinality of M

1 initialization: BITMAP OF L bits initialized to 0.;

2 **for** each x in M **do**

- ① Calculate hash function $h(x)$.;
- ② get binary representation of hash output, call it $bin(h(x))$.;
- ③ Calculate the index i such that $i = \rho(bin(h(x)))$, where $\rho()$ is such that it outputs the **position** of the least-significant set bit,i.e., position of 1.;
- ④ set $BITMAP[i] = 1$.;

3 **end**

4 Let R denote the largest index i such that $BITMAP[i] = 1$;

5 Cardinality of M is $2^R/\phi$ where $\phi \approx 0.77351$;

Flajolet and Martin Algorithm - Intuition

- ① We have a hash function h that maps x to $h(x)$
- ② then take its binary representation of $h(x)$ a bit string
- ③ The more different element in the stream produces more different hash values. → that will end up in the bit string end in some number of 0s
- ④ let R be the maximum tail length of any element seen so far then the estimate for the number of distinct elements 2^R (actually $2^R/\phi$, for $\phi = 0.7735$)
- ⑤ Complexity : $O(n)$ for $n = |X|$ time and $O(\log(m))$ space complexity

Flajolet and Martin Algortihm

www.BANDICAM.com

Given:

Input Stream= 1,3,2,1,2,3,4,3,1

Let's $h(x) = 3x + 1 \bmod 5$ Solve:

| Calculation | Reminder | Binary Conversion |
|-----------------------------|----------|-------------------|
| $h(1) = 3(1) + 1 \bmod 5 =$ | 4 | 100 |
| $h(3) = 3(3) + 1 \bmod 5 =$ | 0 | 000 |
| $h(2) = 3(2) + 1 \bmod 5 =$ | 2 | 010 |
| $h(1) = 3(1) + 1 \bmod 5 =$ | 4 | 100 |
| $h(2) = 3(2) + 1 \bmod 5 =$ | 2 | 010 |
| $h(3) = 3(3) + 1 \bmod 5 =$ | 0 | 000 |
| $h(4) = 3(4) + 1 \bmod 5 =$ | 3 | 011 |
| $h(3) = 3(3) + 1 \bmod 5 =$ | 0 | 000 |
| $h(1) = 3(1) + 1 \bmod 5 =$ | 4 | 100 |

Trailing Zeros: 2,0,1,2,1,0,0,0,2

 $r=2$

$$2^r = 2^2 = 4$$

Source: <https://www.youtube.com/watch?v=TG48mumSIaw>**Attention, 0 counts as 0 trailing zeros!!!**

Intuition

The basic idea behind FM algorithm is that of using a hash function that maps the strings in the stream to uniformly generated random integers in the range $[0, \dots, 2^L - 1]$. Thus we expect that:

- 1/2 of the numbers will have their binary representation end in 0 (divisible by 2)
- 1/4 of the numbers will have their binary representation end in 00 (divisible by 4)
- 1/8 of the numbers will have their binary representation end in 000 (divisible by 8)
- In general, $1/2^R$ of the numbers will have their binary representation end in $[0]^R$

Then the number of unique strings will be approx. 2^R . (because using n bits, we can represent 2^n integers)

Probabilities

- The probability that a given element a has $h(a)$ ending in at least r 0s is $(\frac{1}{2})^r$.
- if there are m distinct element, the probability that NONE of them has a tail length at least r :

$$(1 - 2^{-r})^m, \text{ where}$$

$$P(\text{an element does not have } r \text{ 0s in tail}) = (1 - 2^{-r})$$

$((1 - 2^{-r})^m = ((1 - 2^{-r})^{2^r})^{m2^{-r}} \approx e^{-m2^{-r}} = \frac{1}{e^{\frac{1}{2^r}m}}$, for large enough r , using $(1 - \epsilon)^{\frac{1}{\epsilon}} \approx 1/e$ for large ϵ)

- Thus the probability of not finding r 0s for the true distinct element number m , that is much larger than 2^r : $\frac{1}{e^{\frac{1}{2^r}m}} \rightarrow 0$, and subsequently $P(\text{there is a tail at least } r) \rightarrow 1$,
- The other way, if $2^r \gg m$: $P(\text{there is a tail at least } r) \rightarrow 0$
- Summarizing intuition: the more is the number of distinct element m , the bigger the probability of finding a longer tail.
- the proposed approximation is unlikely to be either too much or too low.

Combine multiple estimates

Since the above method is probabilistic how can we be more certain? Combine them! 2^r - our estimate based on the maximal observed tail length r .

The problem that there is always a chance to observe longer tail, than the right one. So if we have four element, there is a probability, that we might face $r = 3$.

Let us run the method 3 times, and assume we get

$$r_1 = 1, r_2 = 2, r_3 = 3.$$

$$\text{Average: } 2^1 + 2^2 + 2^3 / 3 = 14/3 = 4,66$$

For $m = 8$, assume $r_1 = 2, r_2 = 3, r_3 = 4$: the average

$$2^2 + 2^3 + 2^4 = 28/3 = 9.33$$

For $m = 16$, assume $r_1 = 3, r_2 = 4, r_3 = 5$: the average

$$2^3 + 2^4 + 2^5 = 28/3 = 18.66$$

... For these examples: median

Combine multiple estimates

Since the above method is probabilistic how can we be more certain? Combine them! 2^r - our estimate based on the maximal observed tail length r .

The problem that there is always a chance to observe longer tail, than the right one. So if we have four element, there is a probability, that we might face $r = 3$.

Let us run the method 3 times, and assume we get

$$r_1 = 1, r_2 = 2, r_3 = 3.$$

$$\text{Average: } 2^1 + 2^2 + 2^3 / 3 = 14/3 = 4,66$$

For $m = 8$, assume $r_1 = 2, r_2 = 3, r_3 = 4$: the average

$$2^2 + 2^3 + 2^4 = 28/3 = 9.33$$

For $m = 16$, assume $r_1 = 3, r_2 = 4, r_3 = 5$: the average

$$2^3 + 2^4 + 2^5 = 28/3 = 18.66$$

... For these examples: median this can produce only powers of 2

Count distinct elements
oo

Frequency of elements
oooooooo●

Moments
oooooooooooo

Combine multiple estimates

The way for better estimates: Take small groups of hash functions, and calculate the average, than take the median of those...

Moments in statistics

Moments: quantitative measures related to the shape of the function's graph.

First moment: $\mathbb{E}[X] = \sum_{i=1}^n x_i$

Second moment: $\mathbb{E}[X^2] = \sum_{i=1}^n x_i^2$

Mean:

$$\mu = \mathbb{E}[X] \quad (1)$$

Variance:

$$\sigma^2 = \mathbb{E}[(X - \mathbb{E}[X])^2] \quad (2)$$

$$= \mathbb{E}[X^2 - 2X \cdot \mathbb{E}[X] + \mathbb{E}[X]^2] \quad (3)$$

$$= \mathbb{E}[X^2] - 2 \cdot \mathbb{E}[X]\mathbb{E}[X] + \mathbb{E}[X]^2 \quad (4)$$

$$= \mathbb{E}[X^2] - \mathbb{E}[X]^2 \quad (5)$$

Standard deviation:

$$\sigma = \sqrt{\mathbb{E}[X^2] - \mathbb{E}[X]^2} \quad (6)$$

Simple statistics over data streams

- **Mean/Expected value**, with \bar{x}_t denoting the estimate at time t

$$\bar{x}_t = \frac{(t-1)\bar{x}_{t-1} + x_t}{t} \quad (7)$$

for this we need to maintain the actual mean \bar{x}_t , and the count of elements seen i

- **Variance**, with σ_t standing for its approximation at time t :

$$\sigma_t = \sqrt{\frac{\sum_{i=1}^t x_i^2 - \frac{(\sum_{i=1}^t x_i)^2}{t}}{t-1}} = \sqrt{\frac{\mathbb{E}[X^2] \cdot t - \frac{(t \cdot \mathbb{E}[X])^2}{t}}{t-1}} \quad (8)$$

$$= \sqrt{\frac{t}{t-1} (\mathbb{E}[X^2] - \mathbb{E}[X]^2)} \quad (9)$$

for what we need to store the sum of elements, or the first moment $\sum_{i=1}^t x_i = \mathbb{E}[X] \cdot t$ and the sum of squared element or second moments: $\sum_{i=1}^t x_i^2 = \mathbb{E}[X^2] \cdot t$

Bessel's Correction

$$\sigma_t = \sqrt{\frac{t}{t-1}(\mathbb{E}[X^2] - \mathbb{E}[X]^2)}$$

Bessel's Correction: "*minus one to correct for bias...*"

Sample standard deviation is a biased estimator - sample is a subset of a population it intends to represent

The fewer sample the ones far from the population mean will have a bigger influence on the sample mean, thus the deviation might be underestimated. Taking the square root of the deviation also further decreases it.

The more sample we use the effect of the correction will vanish as
 $\frac{t}{t-1} \rightarrow 1, t \rightarrow \infty$

blog post: <https://towardsdatascience.com/the-reasoning-behind-bessels-correction-n-1-eeea25ec9bc9>

Frequency Moments over streams

Frequency moments

m_i the number of occurrences of the i th element of an ordered universal base set (from which the elements are taken).

k th order moment of DS $\sum_i m_i^k$

0th moment - FM algorithm

number distinct elements $\sum_{i \in X} \underbrace{m_i^0}_{=1}$, if we define $0^0 = 0$

Frequency Moments over streams

Frequency moments

m_i the number of occurrences of the i th element of an ordered universal base set (from which the elements are taken).

k th order moment of DS $\sum_i m_i^k$

0th moment - FM algorithm

number distinct elements $\sum_{i \in X} \underbrace{m_i^0}_{=1}$, if we define $0^0 = 0$

1st moment - CMS algorithm

sum of all m_i s, that is the **length of the stream**.

Second moment

2nd moment

surprise number, the higher the number is for the same length, the less even the distribution of the element is.

Example

11 value, length of 100 What is the most even distribution?

Second moment

2nd moment

surprise number, the higher the number is for the same length, the less even the distribution of the element is.

Example

11 value, length of 100 What is the most even distribution?

$9 \times 10 = 90$ $10 \times 1 = 10$ so the surprise factor

$$9^2 \times 10 + 10^2 \times 1 = 910$$

What is the biggest possible?

Second moment

2nd moment

surprise number, the higher the number is for the same length, the less even the distribution of the element is.

Example

11 value, length of 100 What is the most even distribution?

$9 \times 10 = 90$ $10 \times 1 = 10$ so the surprise factor

$$9^2 \times 10 + 10^2 \times 1 = 910$$

What is the biggest possible? $1 \times 10 = 10$ and $90 \times 1 = 90$ (one appears 90 times and the others 1 time each) so the surprise factor

$$1^2 \times 10 + 90^2 \times 1 = 8110$$

Alon-Matias-Szegedy algorithm for second moments

- First assume a fixed length n
- No enough space to store all elements to count all m_i s
- Take a number of variables J , and j take a value v_j and a count c_j .
- Pick J random number $i_j \in \{1..n\}$
- When we read stream X , at reaching an i_j , set $v_j = X[i_j]$ and $c_j = 1$ from that point, any time we met v_j in the stream c_i++ .
- approximate the second moment from any j : $n(2c_j - 1)$

Why Alon-Matias-Szegedy algorithm does work?

- for $c[i]$ denoting the number of appearance of the element at the i th position ($v[i] = a$), in the range $i, i+1, \dots, n$

$$\mathbb{E}_n[n(2 \cdot c - 1)] = \frac{1}{n} \sum_{i=1}^n n(2 \cdot c[i] - 1) = \sum_{i=1}^n (2 \cdot c[i] - 1)$$

- grouping together the positions that represent the same element we get

$$\sum_{i=1}^n (2 \cdot c[i] - 1) = \sum_a \sum_{i=1}^n (2 \cdot c[i] - 1) I(v[i] = a):$$

- for the last position an element appears: $2 \cdot c[i] - 1 = 2 \cdot 1 - 1 = 1$

for the next to last: $2 \cdot c[i] - 1 = 2 \cdot 2 - 1 = 3$

and for the next : $2 \cdot c[i] - 1 = 2 \cdot 3 - 1 = 5$

...

for the very first occurrence $2 \cdot c[i] - 1 = 2(\cdot m_a - 1)$

- and then : $1 + 3 + 5 + 2 \cdots + 2(\cdot m_a - 1) = m_a^2$, thus
 $\mathbb{E}[n(2 \cdot c - 1)] = \sum_a (m_a)^2$.

Explanation

$$1 + 3 + 5 + 2 \cdots + 2(\cdot m_{v[i]} - 1) = m_{v[i]}^2$$

difference between two squared number always odd:

$$k^2 - (k-1)^2 = k^2 - (k^2 - 2k + 1) = 2k - 1$$

| | | | | | | | | |
|---|---|---|---|----|-----|-----|-----------|-------|
| 0 | 1 | 4 | 9 | 16 | 25 | ... | $(n-1)^2$ | n^2 |
| 1 | 3 | 5 | 7 | 9 | ... | | 2n - 1 | |

Hence a squared number can be
always written as :

$$k^2 = \sum_{i=1}^k (2i - 1)$$

https:

[//en.wikipedia.org/wiki/
Square_pyramidal_number](https://en.wikipedia.org/wiki/Square_pyramidal_number)

$$\begin{array}{lllll} 1^2 & = & 1 & & \\ 2^2 & = & 1 & 3 & \\ 3^2 & = & 1 & 3 & 5 \\ 4^2 & = & 1 & 3 & 5 & 7 \\ 5^2 & = & 1 & 3 & 5 & 7 & 9 \\ \vdots & & \vdots & & \ddots & \\ (n-1)^2 & = & 1 & \cdots & \cdots & 2n-3 \\ n^2 & = & 1 & \cdots & \cdots & 2n-3 & 2n-1 \end{array}$$

AMS Example

Example "stream"

Stream $X = a, b, c, b, d, a, c, d, a, b, d, c, a, a, b$

length of the stream $n = |X| = 15$

a : 5 times, b : 4 times, c and d 3 times $\rightarrow 5^2 + 4^2 + 3^2 + 3^2 = 59$

AMS

Let $J = 3$, and the random indices be: $i_1 = 3, i_2 = 8, i_3 = 13!$

Start reading from the beginning!

AMS Example

Example "stream"

Stream $X = a, b, c, b, d, a, c, d, a, b, d, c, a, a, b$

length of the stream $n = |X| = 15$

a : 5 times, b : 4 times, c and d 3 times $\rightarrow 5^2 + 4^2 + 3^2 + 3^2 = 59$

AMS

Let $J = 3$, and the random indices be: $i_1 = 3, i_2 = 8, i_3 = 13!$

Start reading from the beginning!

at index 3: $v_1 = c, c_1 = 1$

AMS Example

Example "stream"

Stream $X = a, b, c, b, d, a, c, d, a, b, d, c, a, a, b$

length of the stream $n = |X| = 15$

a : 5 times, b : 4 times, c and d 3 times $\rightarrow 5^2 + 4^2 + 3^2 + 3^2 = 59$

AMS

Let $J = 3$, and the random indices be: $i_1 = 3, i_2 = 8, i_3 = 13!$

Start reading from the beginning!

at index 3: $v_1 = c, c_1 = 1$

at index 7: $c_1 = 2$

AMS Example

Example "stream"

Stream $X = a, b, c, b, d, a, c, d, a, b, d, c, a, a, b$

length of the stream $n = |X| = 15$

$a: 5 \text{ times}, b: 4 \text{ times}, c \text{ and } d 3 \text{ times} \rightarrow 5^2 + 4^2 + 3^2 + 3^2 = 59$

AMS

Let $J = 3$, and the random indices be: $i_1 = 3, i_2 = 8, i_3 = 13!$

Start reading from the beginning!

at index 3: $v_1 = c, c_1 = 1$

at index 7: $c_1 = 2$

at index 8: $v_2 = d, c_2 = 1$

AMS Example

Example "stream"

Stream $X = a, b, c, b, d, a, c, d, a, b, d, c, a, a, b$

length of the stream $n = |X| = 15$

$a: 5 \text{ times}, b: 4 \text{ times}, c \text{ and } d 3 \text{ times} \rightarrow 5^2 + 4^2 + 3^2 + 3^2 = 59$

AMS

Let $J = 3$, and the random indices be: $i_1 = 3, i_2 = 8, i_3 = 13!$

Start reading from the beginning!

at index 3: $v_1 = c, c_1 = 1$

at index 7: $c_1 = 2$

at index 8: $v_2 = d, c_2 = 1$

at index 8: $c_2 = 2$

AMS Example

Example "stream"

Stream $X = a, b, c, b, d, a, c, d, a, b, d, c, a, a, b$

length of the stream $n = |X| = 15$

$a: 5 \text{ times}, b: 4 \text{ times}, c \text{ and } d 3 \text{ times} \rightarrow 5^2 + 4^2 + 3^2 + 3^2 = 59$

AMS

Let $J = 3$, and the random indices be: $i_1 = 3, i_2 = 8, i_3 = 13!$

Start reading from the beginning!

at index 3: $v_1 = c, c_1 = 1$

at index 7: $c_1 = 2$

at index 8: $v_2 = d, c_2 = 1$

at index 8: $c_2 = 2$

...

at the end: Approximation:

$$n(2 \cdot c_i - 1)$$

$$c_1 = 3 - > 15(2 \cdot 3 - 1) = 75$$

$$c_2 = 2 - > 15(2 \cdot 2 - 1) = 45$$

$$c_3 = 2 - > 15(2 \cdot 2 - 1) = 45$$

The true value is 59, and if we take the average of the three estimate we get: $\frac{75+45+45}{3} = 55$

AMS for Infinite streams

- How to pick positions i for variables:
- danger of being biased towards the ones at the "beginning"
- Idea: store as many variables as we can all time, throw some out as the stream grows - being the discarded ones replaced by new ones.
- Let us suppose there is space for s variables.
- inductively: let us assume that we have some n_0 elements, then the positions will be distributed uniformly with $p = s/n_0$. When a new element arrives, with the probability of $s/(n + 1)$ pick the new position as variable position, and if we picked throw away uniformly an old one.

Count distinct elements
oo

Frequency of elements
oooooooooooo

Moments
oooooooooooo●

Bibliography I

Data Stream Mining- Lecture 3

Statistics on recent data

Chandresh Kumar Maurya, Assistant Professor, Péter Kiss,
Teaching Assistant

Eötvös Loránd University, Budapest, Hungary

October 22, 2020

Data Synopsis

Machine Learning / Data Mining - give a model for the data - compress knowledge, statistics.

For data streams

- unbound nature
- concept drift

Need to compute an estimate of the stream due to 1) low memory, 2) fast computation. Data synopsis can be done in two ways:

- Sliding Window
- Data Reduction

Sliding window

Up to now we tried to approximate statistics for the whole stream.
Why we need sliding window?

Sliding window

Up to now we tried to approximate statistics for the whole stream.

Why we need sliding window?

For capturing recent data

Sliding window

Up to now we tried to approximate statistics for the whole stream.

Why we need sliding window?

For capturing recent data

Types:

- **Sequence based:** they contain sequences of data and size of the window is decided based on the number of data sequences they contain.
- **Timestamp based:** The size of the window is decided based on the time interval considered.

Example for query recent statistics

What is the standard deviation of the last 100 element?

First moment: $100 \cdot \mathbb{E}[\bar{X}] (= 100 \cdot \mu)$

$$= \sum_{i=n-99}^n x_i \quad (1)$$

Second moment: $100 \cdot \mathbb{E}[\bar{X}^2]$

$$= \sum_{i=n-99}^n x_i^2 \quad (2)$$

Standard deviation: σ

$$= \sqrt{\mathbb{E}[\bar{X}^2] - \mathbb{E}[\bar{X}]^2} \quad (3)$$

How to maintain the value?

Example for query recent statistics

What is the standard deviation of the last 100 element?

$$\text{First moment: } 100 \cdot \mathbb{E}[\bar{X}] (= 100 \cdot \mu) = \sum_{i=n-99}^n x_i \quad (1)$$

$$\text{Second moment: } 100 \cdot \mathbb{E}[\bar{X}^2] = \sum_{i=n-99}^n x_i^2 \quad (2)$$

$$\text{Standard deviation: } \sigma = \sqrt{\mathbb{E}[\bar{X}^2] - \mathbb{E}[\bar{X}]^2} \quad (3)$$

How to maintain the value? any time t a new element appears :

$$\mathbb{E}[\bar{X}]_t = \mathbb{E}[\bar{X}]_{t-1} - \frac{x_{t-100}}{100} + \frac{x_t}{100} \quad (4)$$

$$\mathbb{E}[\bar{X}^2]_{t-1} = \mathbb{E}[\bar{X}^2]_{t-1} - \frac{x_{t-100}^2}{100} + \frac{x_t^2}{100} \quad (5)$$

Thus for this we need to keep 200 elements in memory. Similar problem for timestamp based windowing, based on granularity continuous updates, plus also the size of window changes by time..

Sequence based window: Examples

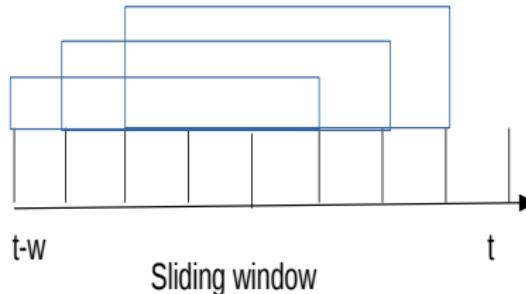
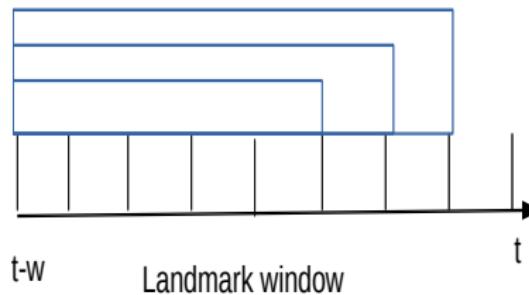


Figure: Sequence based windows. Top figure: Landmark window and bottom figure is sliding window (used in packet transmission)

Timestamp based window: Examples

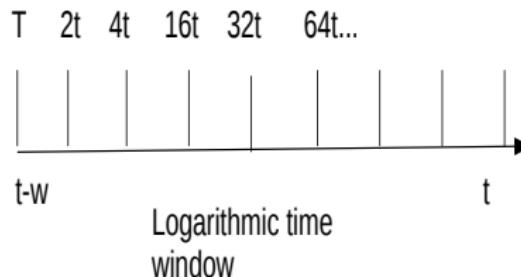
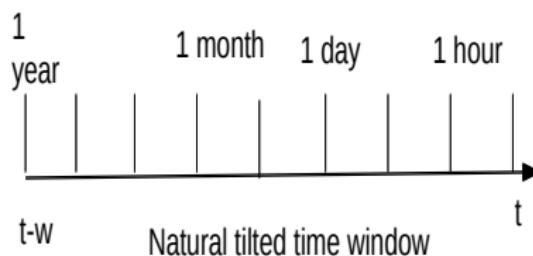


Figure: Timestamp based windows. Top figure: natural tilted window and bottom figure is logarithmic time window

Exponential histograms for sliding windows - DGIM

The Goal: can we reduce space necessary for maintain statistics of the window?

Exponential histograms for sliding windows - DGIM

The Goal: can we reduce space necessary for maintain statistics of the window?

DGIM : maintains approximations of aggregate functions on sliding windows of real numbers on $\mathcal{O}(\log |W|)$ memory vs $\mathcal{O}(|W|)$

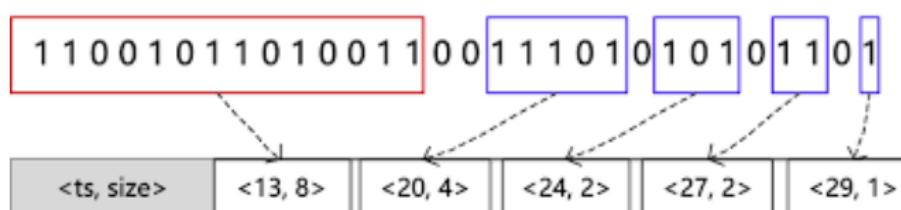
Exponential histograms for sliding windows - DGIM

The Goal: can we reduce space necessary for maintain statistics of the window?

DGIM : maintains approximations of aggregate functions on sliding windows of real numbers on $\mathcal{O}(\log |W|)$ memory vs $\mathcal{O}(|W|)$

- each element has a timestamp : the position in which it arrives
- partition W int sequence of subwindows - each is represented by a bucket with a capacity of a power of 2
- timestamp an integer that is the timestamp of the most recent 1 in the bucket

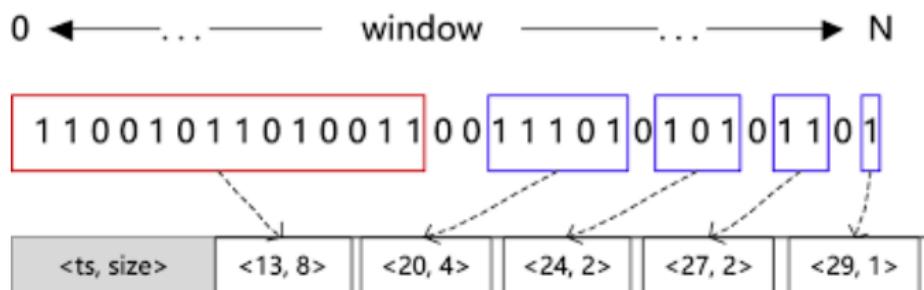
0 ← ... window → N



Source:

https://link.springer.com/chapter/10.1007/978-981-15-3281-8_12

Frame Title



- parameter r no. of buckets with a given capacity: $1, 2, 4, \dots, 2^m$, with an m for

$$r \sum_{i=0}^{m-1} 2^i < |W| \leq r \sum_{i=0}^m 2^i. \quad (6)$$

Exponential histograms for sliding windows - DGIM

DGIM :maintains approximations of aggregate functions on sliding windows of real numbers

- Example for DGIM Query: How many ones in the last k items?
- aggregate function : sum as and elements are bits
- Answer :
 - ① find the bucket b with the earliest timestamp t that includes some of the most recent bits
 - ② sum all the capacities of the buckets completely within k plus half of the size of b

Exponential histograms for sliding windows - DGIM

| | | | | | | |
|------------|---|--|--|--|--|--|
| Bucket: | 1011100 | 10100101 | 100010 | 11 | 10 | 1000 |
| Capacity: | 4 | 4 | 2 | 2 | 1 | 1 |
| Timestamp: | $t - 24$ | $t - 14$ | $t - 9$ | $t - 6$ | $t - 5$ | $t - 3$ |

- Example: the most recent with $t - 3$, the least recent with $t - 28$ for a desired window size $|W| = 25$,
- our last bucket will be the one that includes the timestamp $t - 25$
- so our estimate: $1 + 1 + 2 + 2 + 4 + 4/2 = 12$, while the true count is 11.
- Relative Error: $(12 - 11)/11 \approx 9\%$. https://www.cms.waikato.ac.nz/~abifet/book/chapter_4.html

Reduce Error - trade- off

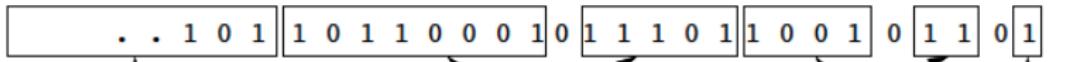
- The error is introduced only by the oldest bucket b
- it must contain at least one 1 with the timestamp greater than $t - |W|$, otherwise it would have been dropped.
- it also might however contain up to $2^m - 1$ 1s in worst case with a timestamp at most $t - |W|$, that should not be counted.
- Thus the error is at most half of the capacity of the oldest bucket
- There are either k or $k - 1$ buckets of each sizes , but the largest one → the relative error is at most:
$$(2^m)/(2^m + (k - 1)(2^{m-1} + \dots + 1)) \approx 1/2k.$$
- with a bound of relative error ϵ , it suffices $k = 1/2\epsilon$. Number of buckets $(\log |W|)/2\epsilon$

Maintaining DGIM condition

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



At least one of size 8 Two of size 4 One of size 2 Two of size 1



At least one of size 8 Two of size 4 Two of size 2 One of size 1

Exponential histograms for sliding windows - DGIM

EXPONENTIAL HISTOGRAMS

```
1  Init( $k, W$ ):  
2       $t \leftarrow 0$   
3      create a list of empty buckets  
4  Update( $b$ ):  
5       $t \leftarrow t + 1$   
6      if  $b = 1$     ▷ do nothing with 0s  
7          let  $t$  be the current time  
8          create a bucket of capacity 1 and timestamp  $t$   
9           $i \leftarrow 0$   
10         while there are more than  $k$  buckets of capacity  $2^i$   
11             merge the two oldest buckets of capacity  $2^i$  into a  
12                 bucket of capacity  $2^{i+1}$ : the timestamp of the new bucket  
13                 is the largest (most recent) of their two timestamps  
14              $i \leftarrow i + 1$   
15             remove all buckets whose timestamp is  $\leq t - W$   
16  Query():  
17      return the sum of the capacities of all existing buckets  
18          minus half the capacity of the oldest bucket
```

Computing Statistics over Sliding Window: The ADWIN algorithm

Why we need to estimate statistics over window? Because can not store all items in the window or want to perform some operation.

Solution: Adaptive Sliding Window Algorithm
(ADWIN)[Bifet and Gavalda, 2007] .

ADWIN

A change detector and estimator algorithm using an adaptive size sliding window

Problem

Average of the stream of bits or real values.

ADWIN

Problem

What is the average of the data NOW?

ADWIN

Problem

What is the average of the data NOW?

ADWIN algorithm

- variable length window of recently seen items
- keeps the maximal length window, that is statistically consistent with the hypothesis: there has been no change in the average values \bar{X} , and \bar{X}^2
- old fragment is dropped iff there is enough evidence that its average differs enough from the other parts

Inputs :

- $\delta \in (0, 1)$ confidence value, and $X = x_1, x_2, \dots, x_t, \dots$
- x_t is only available at time t , and assume that $x_t \in [0, 1]$
- a sample /element $x_t \sim \mathcal{D}_t$ independently, $\mathbb{E}[X]_{X \sim \mathcal{D}_t} = \mu_t$

ADWIN

- $\bar{\mu}_W$ observed average over window W , μ_W the unknown true average
- whenever "large enough" (m) subwindows shows "distinct enough" (ϵ_{cut}) averages, we conclude that the corresponding expected values are different.

m is the harmonic mean of W_0 and W_1 :

$$m = \frac{2}{1/|W_0| + 1/|W_1|} \quad (7)$$

$$\epsilon_{cut} = \sqrt{\frac{1}{2m} \cdot \ln \frac{4|W|}{\delta}} \quad (8)$$

- if so, older portion of the window is dropped

Computing Statistics over Sliding Window: The ADWIN algorithm

Algorithm 1: ADWIN

Input : Sequence $\{x_t\}$ and confidence value δ

Initialization: Window W

```
1 for  $t > 0$  do
2    $W \rightarrow W \cup x_t$  (add items to the head of  $W$ )
3   do
4     | Drop elements from the tail of  $W$ 
5   while  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \epsilon_{cut}$  holds for all split of  $W$  into  $W_0$ 
        and  $W_1$ ;
6 end
7 Output:  $\hat{\mu}_W$ 
```

Computing Statistics over Sliding Window: The ADWIN algorithm

Algorithm 2: ADWIN

Input : Sequence $\{x_t\}$ and confidence value δ

Initialization: Window W

```
1 for  $t > 0$  do
2    $W \rightarrow W \cup x_t$  (add items to the head of  $W$ )
3   do
4     | Drop elements from the tail of  $W$ 
5     | while  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \epsilon_{cut}$  holds for all split of  $W$  into  $W_0$ 
       and  $W_1$ ;
6   end
7   Output:  $\hat{\mu}_W$ 
```

demonstration from : Joao Gama (LIAAD-INESC Porto, University of Porto, Portugal)

Algorithm ADaptive Sliding WINdow

Example

$$W = \boxed{101010110111111}$$

$$W_0 = \boxed{1}$$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
 - 5 Drop elements from the tail of W
 - 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
 - 7 for every split of W into $W = W_0 \cdot W_1$
 - 8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

Example

$$W = \boxed{101010110111111}$$

$$W_0 = \boxed{1} \quad W_1 = \boxed{01010110111111}$$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
 - 5 Drop elements from the tail of W
 - 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
 - 7 for every split of W into $W = W_0 \cdot W_1$
 - 8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

Example

$$W = \boxed{101010110111111}$$

$$W_0 = \boxed{10} \quad W_1 = \boxed{1010110111111}$$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
 - 5 Drop elements from the tail of W
 - 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
 - 7 for every split of W into $W = W_0 \cdot W_1$
 - 8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

Example

$$\begin{aligned}W &= \boxed{101010110111111} \\W_0 &= \boxed{101} \quad W_1 = \boxed{010110111111}\end{aligned}$$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
 - 5 Drop elements from the tail of W
 - 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
 - 7 for every split of W into $W = W_0 \cdot W_1$
 - 8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

Example

$$W = \boxed{101010110111111}$$

$$W_0 = \boxed{1010} \quad W_1 = \boxed{10110111111}$$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
 - 5 Drop elements from the tail of W
 - 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
 - 7 for every split of W into $W = W_0 \cdot W_1$
 - 8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

Example

$$W = [101010110111111]$$

$$W_0 = [10101] \quad W_1 = [0110111111]$$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
 - 5 Drop elements from the tail of W
 - 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
 - 7 for every split of W into $W = W_0 \cdot W_1$
 - 8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

Example

$$W = \boxed{101010110111111}$$

$$W_0 = \boxed{101010} \quad W_1 = \boxed{110111111}$$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
 - 5 Drop elements from the tail of W
 - 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
 - 7 for every split of W into $W = W_0 \cdot W_1$
 - 8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

Example

$$W = \boxed{101010110111111}$$

$$W_0 = \boxed{1010101} \quad W_1 = \boxed{10111111}$$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
 - 5 Drop elements from the tail of W
 - 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
 - 7 for every split of W into $W = W_0 \cdot W_1$
 - 8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

Example

$$W = \boxed{101010110111111}$$

$$W_0 = \boxed{10101011} \quad W_1 = \boxed{0111111}$$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
 - 5 Drop elements from the tail of W
 - 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
 - 7 for every split of W into $W = W_0 \cdot W_1$
 - 8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

Example

$W = \boxed{101010110111111} \quad |\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c : \text{CHANGE DET.!}$

$W_0 = \boxed{101010110} \quad W_1 = \boxed{111111}$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
- 5 Drop elements from the tail of W
- 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
- 7 for every split of W into $W = W_0 \cdot W_1$
- 8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

Example

$W = \boxed{101010110111111}$ Drop elements from the tail of W
 $W_0 = \boxed{101010110} \quad W_1 = \boxed{111111}$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
- 5 Drop elements from the tail of W
- 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
- 7 for every split of W into $W = W_0 \cdot W_1$
- 8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

Example

$W = \boxed{01010110111111}$ Drop elements from the tail of W
 $W_0 = \boxed{101010110} \quad W_1 = \boxed{111111}$

ADWIN: ADAPTIVE WINDOWING ALGORITHM

- 1 Initialize Window W
- 2 **for** each $t > 0$
- 3 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
- 4 **repeat**
- 5 Drop elements from the tail of W
- 6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
- 7 for every split of W into $W = W_0 \cdot W_1$
- 8 Output $\hat{\mu}_W$

Frame Title

trade-off between reacting quickly to changes and having few false alarm:

- ① (False positive rate bound). If μ_t has remained constant within W , the probability that ADWIN shrinks the window at this step is at most δ ;
- ② (False negative rate bound). Suppose that for some partition of W in two parts $W_0 W_1$ (where W_1 contains the most recent items) we have $|_{0.1}| > \epsilon_{\text{cut}}$. Then with probability $1 - \delta$ ADWIN shrinks W to W_1 , or shorter.

ADWIN in action

ADWIN can be used for two purposes 1) as change detector and 2) as an estimator of the average.

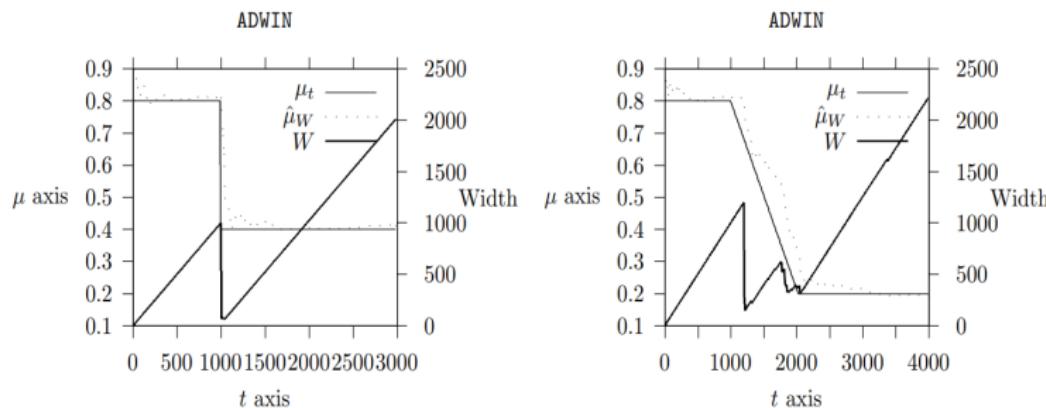


Figure: Output of ADWIN algorithm for different change rates (a)
Output with abrupt change (b) output from gradual change.

Bibliography I



Bifet, A. and Gavalda, R. (2007).

Learning from time-changing data with adaptive windowing.

In *Proceedings of the 2007 SIAM international conference on data mining*, pages 443–448. SIAM.

Data Stream Mining- Lecture 3

Clustering of Data Streams

Chandresh Kumar Maurya, Research Assistant Professor, Peter Kiss Teaching Assistant

Eötvös Loránd University, Budapest, Hungary

October 22, 2020

What is clustering?

Clustering is an unsupervised technique to group (clusters henceforth) the data such that the examples in the group satisfy two requirements:

- Examples in the same cluster are similar
- Examples in the different cluster are dissimilar

Many methods (not all!) build on the assumption :

- that instances are points in some space
- we can define some *distance function* d ,
- "more similar" examples are those that are closer according to d

Clustering

centroid based methods: a set of centroid points in space defines the clustering

for some k number of clusters , P set of instances

A clustering algorithm computes C set of centroids/cluster centers that minimizes an objective cost function

$$\text{cost}(C, P) = \frac{1}{|P|} \sum_{x \in P} d(x, C) \quad (1)$$

$$\text{for } d(x, C) = \min_{c \in C} d(x, c). \quad (2)$$

(minimize the sum of distances of points x to the closest centroid point c)

Clustering of data streams

Clustering of streaming data is non-trivial due to the following challenges:

- ① Concept drifts
- ② High data rate
- ③ Low memory footprints
- ④ Low processing time
- ⑤ Single (or 2-3 passes)
- ⑥ Algorithms needs to be robust
- ⑦ and so on.

Traditional algorithms needs to be adapted to account for the aforementioned issues.

Types of clustering

Should we cluster examples or features?

- Partition based clustering (e.g. K-means, k-medoids)
- Hierarchical clustering (e.g. Agglomerative vs Divisive)
- Density-based clustering (e.g. DBSCAN, OPTICS)
- (Grid-based clustering (e.g. CLIQUE and STING))
- (Modal-based clustering(e.g. COWEB, SOM))

Basic concepts

Requirements of data stream clustering:-

- ① Need a compact representation of the data stream (Why?).
- ② Fast and incremental processing of incoming examples.
- ③ Tracking cluster changes.
- ④ Clear and fast identification of outliers.

Cluster feature

Definition

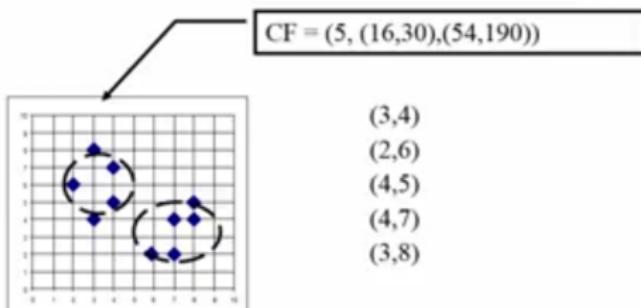
Cluster feature: A cluster feature is a triple (N, LS, SS) used to store the sufficient statistics of the data points.

where

- **N**—number of data points.
- **LS**—a vector to store linear sum of N points.
- **SS**—a vector to store sum of squares of the points.

CF easy to maintain:

- Incrementality
- Additivity
- Centroid
- Radius



Statistics of a CF

Centroid : middle point- the average of the points $c = \frac{\sum_{i=1}^n x_i}{n}$

Radius: average distance from the centroid $R = \sqrt{\frac{\sum_{i=1}^n (x_i - c)^2}{n}}$

Diameter: average pairwise distance of the members

$$D = \sqrt{\frac{\sum_{i=1}^n \sum_{j=1}^n (x_i - x_j)^2}{n(n-1)}}$$

The Leader Algorithm

- Input: user specified distance threshold - maximum distance of example and centroid $d(x, c)$
- At any step assigns the current example to the most similar cluster(if it is allowed)
- otherwise the example is added itself as a leader

Thus:

- no need for prior information on the number of clusters
- unstable - depends too much on the order of examples, and on the correct guess of the threshold

The Leader Algorithm

Algorithm 1: The Leader

Input : X: A sequence of Examples x_i
 δ : Control distance parameter

Output : Centroid of k clusters

Initialization: Initialize the set of centroids $C = x_1$

```
1 for  $x_i \in X$  do
2   Find the cluster  $c_r$  whose center is closest to  $x_i$ ;
3   if  $d(x_i, c_r) < \delta$  then
4     |  $C = C \cup x_i$ 
5   end
6   else
7     | make a new leader with centroid  $x_i$  ;
8   end
9 end
```

Single Pass k -Means Algorithm

- ① the stream is processed in blocks
- ② we use a buffer, where points of the dataset are kept in a compressed way
- ③ at the beginning of a round all the available space on the buffer is filled with points
- ④ based on the buffer find k centers such that the sum of distances of the points from their centroid is minimal
- ⑤ store the k centroid as CFs
- ⑥ in the next round fill again and the clusters will be initialized with the CFs found in previous round.

Single Pass k -Means Algorithm

Algorithm 11: Algorithm for Single Pass k -Means Clustering.

```
input :  $S$ : A Sequence of Examples  
        $k$ : Number of desired Clusters.  
output: Centroids of the  $k$  Clusters  
begin  
    Randomly initialize cluster means.;  
    Each cluster has a discard set that keeps track of the sufficient  
    statistics.;  
    while TRUE do  
        Fill the buffer with examples ;  
        Execute iterations of  $k$ -means on points and discard set in the  
        buffer, until convergence. ;  
        /* For this clustering, each discard set is treated  
           like a regular point weighted with the number of  
           points in the discard set. */  
    ;  
    foreach group do  
        update sufficient statistics of the discard set with the  
        examples assigned to that group;  
    Remove points from the buffer;
```

Figure: Single pass k -means

Hierarchical clustering

- Create clusters of various sizes
- two types: Agglomerative and divisive clustering
- An example is BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)
- Extension of BIRCH for data streams is CluStream [Aggarwal et al., 2003]

BIRCH- Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)

- agglomerative clustering integrated with other flexible clustering methods
- low level micro clustering preserves inherent clustering structure
- high level macro clustering - (clustering CF-s) provide flexibility for integration with other methods
- using a tree representation to find the closes CF
- parameters: branching factor maximum number for children and maximum diameter for CF

concerns

- sensitive to order of data points
- fixed size of leaf nodes - clusters might not be so natural

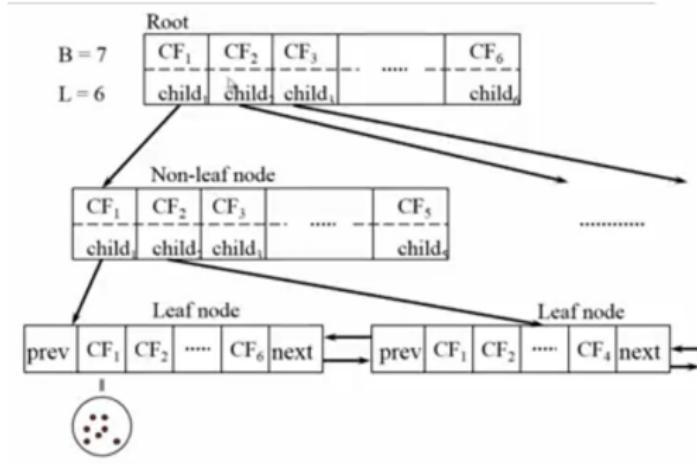
BIRCH - CF -tree

CF-tree Tree

- ① CF tree a height balanced tree that stores the clustering features
 - ② non leaf nodes store the sums of its children incremental insertion

Insertion:

- 1 for each point in the input find the closest leaf entry add point to leaf and update corresponding CF
 - 2 if diameter > max_diameter split the leaf, and possibly parents



CluStream

adaptive extension of BIRCH
additional features :

- LT sum of timestamps
- ST sum of squares of timestamps

Key Ideas:

- It is a two-phase algorithm
- In the first phase, it stores summary statistics of the data points seen so far into **micro-clusters** (with a CF-tree).
- In the second-phase, which is an offline phase, actual clustering happens for example with k-means.

Online Cluster Maintenance

How to maintain clusters online?

- CluStream maintains q initial micro-clusters created by applying k -means algorithm on CF in the offline phase
- Whenever a new data points arrive, calculate distance of it from existing CFs, and if the distance is within the **maximum boundary** (RMS of deviations of the points) of the CF, absorb it.
- Otherwise create a new micro-cluster with a **unique ID** associated to it.
- Whenever, number of micro clusters exceeds more than q , either i) delete the oldest micro-cluster or ii) merge two cluster sharing some similarity.

Density based clustering

- clusters might be formed not only by distance (spherical)
- density of connections -

DBSCAN



k-means



From: <https://github.com/NSHipster/DBSCAN>

Density based clustering

- ϵ -neighbourhood of x set of points a distance $\leq \epsilon$
- core point -whose ϵ - neighbourhood has a weight (fraction of datapoint) $\geq \mu$
- *density area* union of ϵ -neighbourhoods of the core points
- a point y is directly reachable from x if y is in the ϵ -neighbourhood of x
- a point y is reachable from x if there is a sequence x_1, \dots, x_n , that $x_1 = x$ and $x_n = y$, and each x_{i+1} is directly reachable from x_i
- a core point c forms a cluster with all the points reachable from it
- outliers: oints that are not reachable from core points

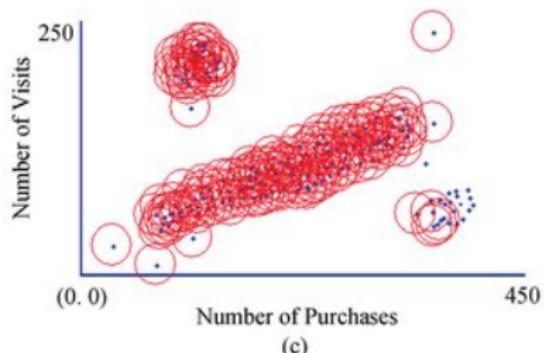
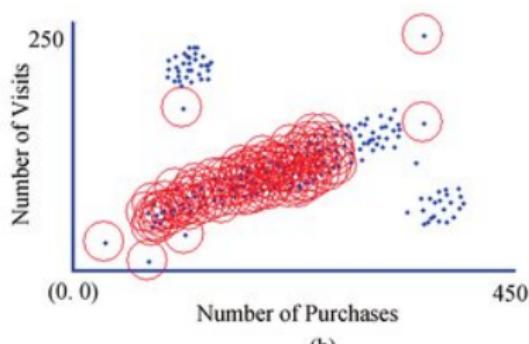
DBSCAN

A cluster is described by any core point unique clustering given ϵ and μ

- visits the points p in an arbitrary order, and finds all directly ϵ -reachable points from it
- if p found to be μ - core point new cluster is formed and all points reachable will added to it, and marked visited
- otherwise temporarily marked as outlier
- outlier later might be found asto be reacheable from a core point

Strongly nonstreaming - multiple pass

DBscan - visualization:



https://www.researchgate.net/publication/258442676_On_Density-Based_Data_Streams_Clustering_Algorithms

Den-Stream

- based on idea of dbscan
- uses microclusters - to compute online statistics quickly
- damped window model - weight of each data point decreases exponentially $f(t) = 2^{-\lambda t}, \lambda > 0$
- for a microcluster with center c and points p_1, \dots, p_n , with the corresponding timestamps T_1, \dots, T_n at each timestep t the algorithm updates the following values:

$$\text{Weight } w = \sum_{i=1}^n f(t - T_i) \quad (3)$$

$$\text{Center } c = \sum_{i=1}^n f(t - T_i) p_i / w \quad (4)$$

$$\text{Radius } r = \sum_{i=1}^n d(p_i, c) / w \quad (5)$$

(for the radius in fact sum of the squares of p_i is kept, from which r can be computed whenever it is needed)

Den-Stream

microclusters

- o(outlier)-microcluster when $w < \mu$
- p(potential)-microcluster when $w > \mu$

two phase clustering

online: when a new datapoint arrives:

- ① try to merge with a p-mc
- ② if not possible - try to merge with an o-mc → if its weight increases over μ promote it
- ③ if no merge - new microcluster

offline:

- ① remove o-microclusters
- ② batch DBScan over on mc-s

Den-stream

DEN-STREAM(*Stream*, λ , μ , β)

Input: a stream of points, decaying factor λ ,
 core weight threshold μ , tolerance factor β

```

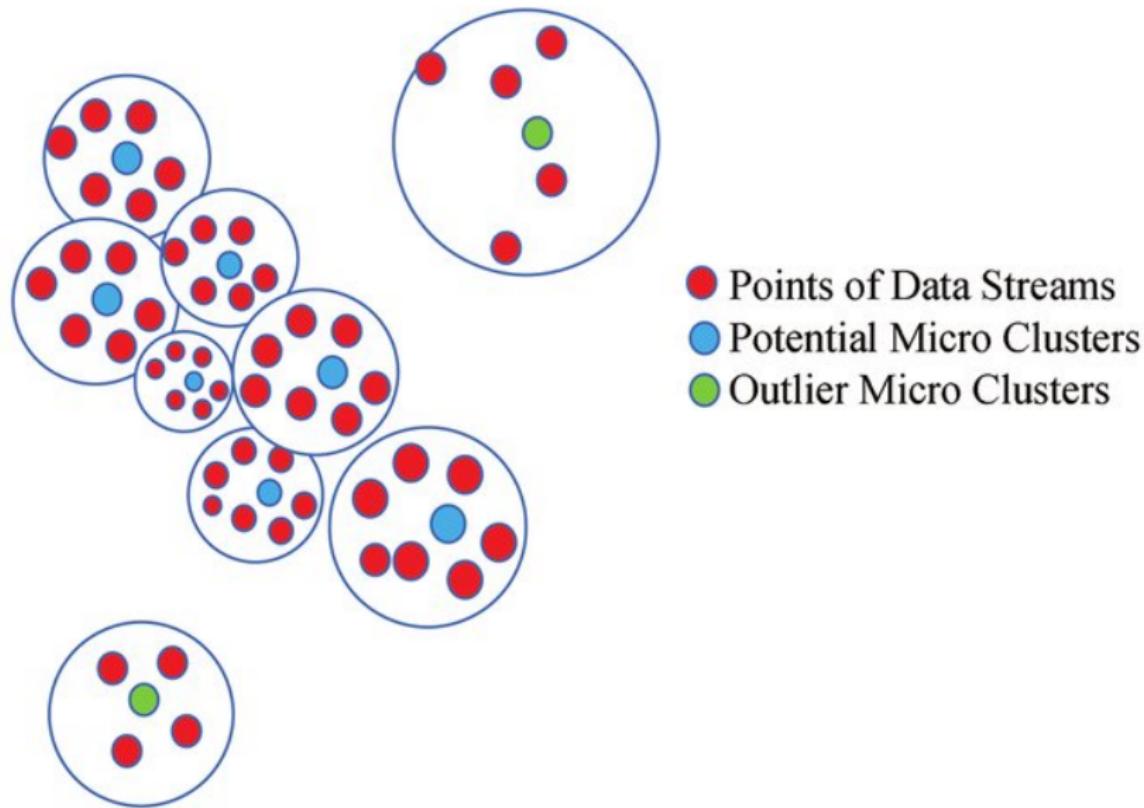
1  ▷ Online phase
2   $T_p \leftarrow \lceil \frac{1}{\lambda} \log\left(\frac{\beta\mu}{\beta\mu-1}\right) \rceil$ 
3  for each new point that arrives
4      do try to merge to a p-microcluster; if not possible,
5          merge to nearest o-microcluster
6          if o-microcluster weight >  $\beta\mu$ 
7              then convert the o-microcluster to p-microcluster
8          else create a new o-microcluster

9  ▷ Offline phase
10 if ( $t \bmod T_p = 0$ )
11     then for each p-microcluster  $c_p$ 
12         do if  $w_p < \beta\mu$ 
13             then remove  $c_p$ 
14         for each o-microcluster  $c_o$ 
15             do if  $w_o < (2^{-\lambda(t-t_o+T_p)} - 1)/(2^{-\lambda T_p} - 1)$ 
16                 then remove  $c_o$ 
17         apply DBSCAN using microclusters as points

```

Source: https://www.cms.waikato.ac.nz/~abifet/book/chapter_9.html

Den-stream



Bibliography I



Aggarwal, C. C., Watson, T. J., Ctr, R., Han, J., Wang, J., and Yu, P. S. (2003).

A framework for clustering evolving data streams.
In *In VLDB*, pages 81–92.

Data Stream Mining- Lecture 4

Classification of Data Streams

Peter Kiss, Teaching Assistant

Eötvös Loránd University, Budapest, Hungary

November 4, 2020

Classification

Source : [Bifet et al., 2018]

Online https://www.cms.waikato.ac.nz/~abifet/book/chapter_6.html

Classification

- supervised learning task
- given a list of groups (often called classes), classification seeks to predict which group a new instance may belong to
- output:
 - the given class
 - probability distribution over classes

examples - binary classification:

- spam filtering - is a given email a spam?
- sentiment analysis - is a tweet express positive or negative sentiment

Training classifiers

- (x, y) is a *training example*, where $x = x_1, \dots, x_d$ is a *feature vector* and $y \in C$ is the *label*, that is the indication of the class, and C is the set of classes
 - the goal is to build a classifier : $y = f(x)$
 - **training:**
 - load all the training data into the memory,
 - and making multiple passes over the data and *fit* classifier function f .

Classification of Stationary Data vs Streams

- a very important assumption in Data mining /ML an especially in classification is **iid**-ness, that is data is independently and identically distributed
- stationary distribution (does not change over time) producing the data in random order,
- In streaming environment not true at all: or certain time periods the labels or classes of instances are correlated.
 - In intrusion detection, there are long periods where all class labels are no-intrusion, mixed with infrequent, short periods of intrusion.

Baseline classifiers

- **Majority class** - new instance will be predicted to be the most frequent class - corresponds to random guess classifier in stationary learning
 - **No-Change classifier** - label of the new instance = last label (see intrusion detection example)

Naive Bayes

Bayes theorem:

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}} \quad (1)$$

$$Pr(c|d) = \frac{Pr(c) \cdot Pr(d|c)}{Pr(d)} \quad (2)$$

where $Pr(c)$ prior the initial probability of event c , $Pr(c|d)$ is the posterior, the probability after accounting d), $Pr(d|c)$ is the likelihood of event d given that event c occurs, and $Pr(d)$

Derivation - from conditional probability

$$Pr(c \cap d) = Pr(c)Pr(d|c) = Pr(d)Pr(c|d) \quad (3)$$

$$Pr(c|d) = \frac{Pr(c) \cdot Pr(d|c)}{Pr(d)} \quad (4)$$

Building NB classifier

incremental -thus well suited for streaming

Let x_1, \dots, x_d discrete attributes, assuming that x_i may take n_i different values.

Upon receiving an unlabelled instance $I = (x_1 = v_1, \dots, x_d = v_d)$ the naive Bayes classifier assigns the probability for I belonging to class $c \in C$

$$Pr(C = c|I) = Pr(C = c) \cdot \prod_{i=1}^d Pr(x_i = v_i | C = c) \quad (5)$$

$$Pr(C = c|I) = Pr(C = c) \cdot \prod_{i=1}^d \frac{Pr(x_i = v_i \wedge C = c)}{Pr(C = c)} \quad (6)$$
$$(7)$$

where $Pr(C = c)$ and $Pr(x_i = v_i \wedge C = c)$ comes from simply counting the occurrences in the training data.

Naive Bayes example for sentiment analysis - Training

Training: First let us assume that we have 4 labelled example

| ID | Text | Sentiment |
|----|--------------------|-----------|
| T1 | glad happy glad | + |
| T2 | glad joyful glad | + |
| T3 | glad pleasant | + |
| T4 | miserable sad glad | - |

Then count how many times a given word appears in the example - transform into vector of attributes, where attributes are element of the dictionary, and values 0 or 1/true or false:

| Id | glad | happy | joyful | pleasant | miserable | sad | Sentiment |
|----|------|-------|--------|----------|-----------|-----|-----------|
| T1 | 1 | 1 | 0 | 0 | 0 | 0 | + |
| T2 | 1 | 0 | 1 | 0 | 0 | 0 | + |
| T3 | 1 | 0 | 0 | 1 | 0 | 0 | + |
| T4 | 1 | 0 | 0 | 0 | 1 | 1 | - |

Naive Bayes example for sentiment analysis- Training

then check in how many examples of each class the words occur

| Class | Value | glad | happy | joyful | pleasant | miserable | sad |
|-------|----------|------|-------|--------|----------|-----------|-----|
| + | 1 | 3 | 1 | 1 | 1 | 0 | 0 |
| + | 0 | 0 | 2 | 2 | 2 | 3 | 3 |
| - | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| - | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

Naive Bayes example for sentiment analysis- Prediction

| ID | Text | Sentiment |
|----|----------------------------------|-----------|
| T5 | glad sad miserable pleasant glad | ? |

translating into a feature vector(feature = vocabulary)

| ID | glad | happy | joyful | pleasant | miserable | sad | Sentiment |
|----|------|-------|--------|----------|-----------|-----|-----------|
| T5 | 1 | 0 | 0 | 1 | 1 | 1 | ? |

Naive Bayes example for sentiment analysis- Prediction

| ID | glad | happy | joyful | pleasant | miserable | sad | Sentiment |
|----|------|-------|--------|----------|-----------|-----|-----------|
| T5 | 1 | 0 | 0 | 1 | 1 | 1 | ? |

Probabilities :

$$Pr(+|T5) = Pr(+) \cdot Pr(glad = 1|+) \cdot Pr(joyful = 0|+) \cdot Pr(happy = 0|+) \quad (8)$$

$$\cdot Pr(pleasant = 1|+) \cdot Pr(miserable = 1|+) \cdot Pr(sad = 1|+) \quad (9)$$

$$= \frac{3}{4} \cdot \frac{3}{3} \cdot \frac{2}{3} \cdot \frac{2}{3} \cdot \frac{1}{3} \cdot \frac{0}{3} \cdot \frac{0}{3} = 0 \quad (10)$$

where $Pr(glad = 1|+)$ means in what fraction of the positive cases occurs the attribute value $glad = 1$. That is $Pr(glad = 1 \wedge C = +) / Pr(glad = 1 \wedge C = +) + Pr(glad = 0 \wedge C = +)$.

| Class | Value | glad | happy | joyful | pleasant | miserable | sad |
|-------|-------|------|-------|--------|----------|-----------|-----|
| + | 1 | 3 | 1 | 1 | 1 | 0 | 0 |
| + | 0 | 0 | 2 | 2 | 2 | 3 | 3 |
| - | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| - | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

Naive Bayes example for sentiment analysis- Prediction

| ID | glad | happy | joyful | pleasant | miserable | sad | Sentiment |
|----|------|-------|--------|----------|-----------|-----|-----------|
| T5 | 1 | 0 | 0 | 1 | 1 | 1 | ? |

$$Pr(-|T5) = Pr(-) \cdot Pr(glad = 1|-) \cdot Pr(joyful = 0|-) \cdot Pr(happy = 0|-) \quad (11)$$

$$\cdot Pr(pleasant = 1|-) \cdot Pr(miserable = 1|-) \cdot Pr(sad = 1|-) \quad (12)$$

$$= \frac{1}{4} \cdot \frac{1}{1} \cdot \frac{1}{1} \cdot \frac{1}{1} \cdot \frac{0}{1} \cdot \frac{1}{1} \cdot \frac{1}{1} = 0 \quad (13)$$

| Class | Value | glad | happy | joyful | pleasant | miserable | sad |
|-------|----------|------|-------|----------|----------|-----------|-----|
| + | 1 | 3 | 1 | 1 | 1 | 0 | 0 |
| + | 0 | 0 | 2 | 2 | 2 | 3 | 3 |
| - | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| - | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

Naive Bayes example for sentiment analysis- Prediction

There is a minor problem however:

$$Pr(-|T5) = \frac{1}{4} \cdot \frac{1}{1} \cdot \frac{1}{1} \cdot \frac{1}{1} \cdot \frac{0}{1} \cdot \frac{1}{1} \cdot \frac{1}{1} = 0 \quad (14)$$

$$Pr(+|T5) = \frac{3}{4} \cdot \frac{3}{3} \cdot \frac{2}{3} \cdot \frac{2}{3} \cdot \frac{1}{3} \cdot \frac{0}{3} \cdot \frac{0}{3} = 0 \quad (15)$$

Naive Bayes example for sentiment analysis- Prediction

There is a minor problem however:

$$Pr(-|T5) = \frac{1}{4} \cdot \frac{1}{1} \cdot \frac{1}{1} \cdot \frac{1}{1} \cdot \frac{0}{1} \cdot \frac{1}{1} \cdot \frac{1}{1} = 0 \quad (14)$$

$$Pr(+|T5) = \frac{3}{4} \cdot \frac{3}{3} \cdot \frac{2}{3} \cdot \frac{2}{3} \cdot \frac{1}{3} \cdot \frac{0}{3} \cdot \frac{0}{3} = 0 \quad (15)$$

Laplace correction : $Pr(d|c) = \frac{n_{dc}+1}{n_d+n_c}$, n_{dc} - how many times the given value occurs in cases where the class is c , n_c is number of classes, n_d number of occurrence of the class

in practice we initialize the counting table with 1s:

| Class | Value | glad | happy | joyful | pleasant | miserable | sad |
|-------|----------|------|-------|--------|----------|-----------|-----|
| + | 1 | 4 | 2 | 2 | 2 | 1 | 1 |
| + | 0 | 1 | 3 | 3 | 3 | 4 | 4 |
| - | 1 | 2 | 1 | 1 | 1 | 2 | 2 |
| - | 0 | 1 | 2 | 2 | 2 | 1 | 1 |

Naive Bayes example for sentiment analysis- Prediction

$$\Pr(+|T5) = \Pr(+) \cdot \Pr(glad = 1|+) \cdot \Pr(joyful = 0|+) \cdot \Pr(happy = 0|+) \\ \cdot \Pr(pleasant = 1|+) \cdot \Pr(miserable = 1|+) \cdot \Pr(sad = 1|+) \quad (16)$$

$$\cdot \Pr(pleasant = 1|+) \cdot \Pr(miserable = 1|+) \cdot \Pr(sad = 1|+) \quad (17)$$

$$= \frac{3}{4} \cdot \frac{4}{5} \cdot \frac{3}{5} \cdot \frac{3}{5} \cdot \frac{1}{5} \cdot \frac{1}{5} = 0.0128 \quad (18)$$

$$\Pr(-|T5) = \Pr(+) \cdot \Pr(glad = 1|-) \cdot \Pr(joyful = 0|-) \cdot \Pr(happy = 0|-) \\ \cdot \Pr(pleasant = 1|-) \cdot \Pr(miserable = 1|-) \cdot \Pr(sad = 1|-) \quad (19)$$

$$\cdot \Pr(pleasant = 1|-) \cdot \Pr(miserable = 1|-) \cdot \Pr(sad = 1|-) \quad (20)$$

$$= \frac{1}{4} \cdot \frac{2}{3} \cdot \frac{2}{3} \cdot \frac{2}{3} \cdot \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{2}{3} = 0.0987 \quad (21)$$

Prediction: $0.0128 = \Pr(+|T5) < \Pr(-|T5) = 0.0987$ -*i* therefore our prediction is **negative**.

Multinomial Naive Bayes (MNB) [McCallum et al., 1998]

- Extension for document classification
- a document is considered as *bag of words*

$$Pr(c|d) = \frac{Pr(c) \prod_{w \in d} Pr(w|c)^{n_{wd}}}{Pr(d)} \quad (22)$$

for $Pr(d)$ normalization factor (thus we omit it usually), n_{wd} is the number of times that word w occurs in document d .

- Key difference is the interpretation of the likelihoods $Pr(w|c)$ - here stands for number of occurrences of w in documents of class c over the total number of words in documents in c . That is the **probability of observing w at any position of a document belonging to c** .

MNB

$$Pr(c|d) = \frac{Pr(c) \prod_{w \in d} Pr(w|c)^{n_{wd}}}{Pr(d)} \quad (23)$$

- Thus absence of a word in a document does not make any class more likely than any other, since it means that the exponent of the probability is 0
- $n_{wd}, PR(w|c), Pr(c)$ are trivial to estimate on a data stream, by keeping the appropriate counts.
- (Laplace correction can be also used, as we do here too)

MNB

| ID | Text | Sentiment |
|----|--------------------|-----------|
| T1 | glad happy glad | + |
| T2 | glad joyful glad | + |
| T3 | glad pleasant | + |
| T4 | miserable sad glad | - |

compute the occurrences of w per class c (all the documents, all the occurrences)

| Class | glad | happy | joyful | pleasant | miserable | sad | Total |
|-------|------|-------|--------|----------|-----------|-----|---------------|
| + | 6 | 2 | 2 | 2 | 1 | 1 | 8+6=14 |
| - | 2 | 1 | 1 | 1 | 2 | 2 | 3+6=9 |

Frame Title

Now the prediction:

$$Pr(+)|T5) = Pr(+) \cdot Pr(glad = 1|+) \cdot Pr(joyful = 0|+) \cdot Pr(happy = 0|+) \quad (24)$$

$$\cdot Pr(pleasant = 1|+) \cdot Pr(miserable = 1|+) \cdot Pr(sad = 1|+) \quad (25)$$

$$= \frac{3}{4} \cdot \left(\frac{6}{14}\right)^2 \cdot \left(\frac{2}{14}\right)^1 \cdot \left(\frac{1}{14}\right)^1 \cdot \left(\frac{1}{14}\right)^1 = 10.04 \cdot 10^{-5} \quad (26)$$

$$Pr(-|T5) = Pr(+) \cdot Pr(glad = 1|-) \cdot Pr(joyful = 0|-) \cdot Pr(happy = 0|-) \quad (27)$$

$$\cdot Pr(pleasant = 1 | -) \cdot Pr(miserable = 1 | -) \cdot Pr(sad = 1 | -) \quad (28)$$

$$= \frac{1}{4} \cdot \left(\frac{2}{9}\right)^2 \cdot \left(\frac{1}{9}\right)^1 \cdot \left(\frac{2}{9}\right)^1 \cdot \left(\frac{2}{9}\right)^1 = 6.77 \cdot 10^{-5} \quad (29)$$

thus now $Pr(+|T5) > Pr(-|T5)$, and our prediction is **positive**

Decision tree

- very popular, easy to interpret and visualize
- each internal node corresponds to an attribute that splits into a branch for each attribute value and the leaves correspond to classification predictor usually majority class classifiers.
- Building decision tree
 - ① if instances are classified perfectly stop
 - ② let attribute A be the best decision criterion of the actual leaf node
 - ③ for each value of A create a new leaf

Decision Tree - Spam classification

example[Bifet et al., 2018]

| Contains “Money” | Domain type | Has attach. | Time received | spam |
|---------------------|----------------|----------------|------------------|------|
| yes | com | yes | night | yes |
| yes | edu | no | night | yes |
| no | com | yes | night | yes |
| no | edu | no | day | no |
| no | com | no | day | no |
| yes | cat | no | day | yes |



Splitting nodes

How to pick A ?

Our goal is to have leaves in the tree, that sort data points into *pure* groups

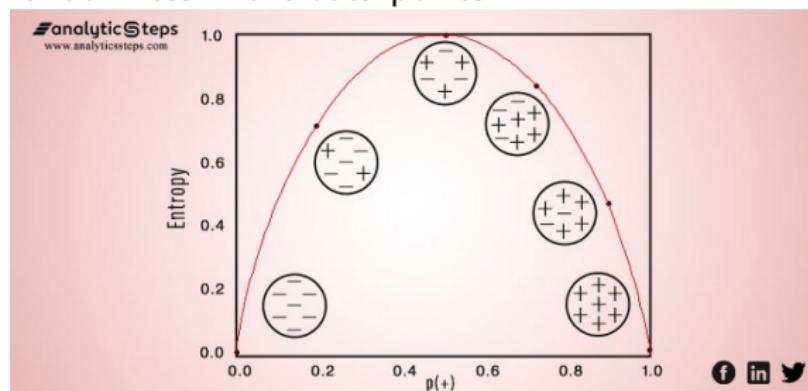
MEasures of purity:

- Entropy
- Giniindex

Split the nodes if it reduces *impurity*

Entropy

Entropy $H(S) = - \sum_c p_c \log(p_c)$ - a measurement of the impurity or randomness in the data points



Information gain

- which feature provides maximal information
- difference between entropy of the class before and after splitting
- after splitting the attribute: $H(S, A) = \sum_a H(S_a)|S_a|/|S|$ for S_a the subset of S where $A = a$
- the information gain $IG(S, A) = H(S) - H(S, A)$. for C4.5 DT algorithm

Gini

- Gini index =gini impurity calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly. 0 means; purity 0.5 of the Gini Index shows an equal distribution of elements over some classes, that is the less is the better. It is a nonlinear measure of dispersion of C:
$$G(C) = \sum_c p_c(1 - p_c) = \sum_{c \neq c'} p_c p_{c'}$$
- Gini impurity reduction: difference between the Gini index before and after splitting the attribute (CART DT alg)

$$Gini(S, A) = \sum_{a \in A} \sum_{c \in C} p_c(1 - p_c)|S_a|/|S| \quad (30)$$

$$= \sum_{a \in A} \left(1 - \sum_{c \in C} p_c^2\right)|S_a|/|S| \quad (31)$$

(since $\sum_c p_c = 1$), with p_c here $p_c = Pr(C = c \wedge A = a)$

- Gini reduction or what : $\Delta Gini(S, A) = Gini(S) - Gini(S, A)$

Attribute selection

we want an attribute, where the sum of the gini / entropy is minimal if we split across that is

$$\max_A \Delta Gini(S, A) \quad \text{or} \quad \max_A IG(S, A) \quad (32)$$

Hoeffding Tree

When should we split: confidence interval for estimating entropy of the node:

$$\epsilon = \sqrt{\frac{R^2 \ln 1/\delta}{2n}} \quad (33)$$

where R is the range of the random variable, δ the desired probability of an estimate not being within ϵ of its expected value, and n is the number of examples collected at the node.
(using Hoeffding bound is formally incorrect but still widely used,)
for Information gain the entropy is in the range $[0, \dots, \log n_c]$

Hoeffding Tree - Streaming DT

[Domingos and Hulten, 2000]

- maintain in each node statistics for splitting
- for discrete attributes it is the same as for NB: for each triple count $(x_i, v_j, c) n_{i,j,c}$ where $x_i = v_j$ and a one dimensional table for the counts $C = c$ that is for a class a given attribute how many times takes the different possible values:

| Class | Value | glad | happy | joyful | pleasant | miserable | sad |
|-------|-------|------|-------|--------|----------|-----------|-----|
| + | 1 | 3 | 1 | 1 | 1 | 0 | 0 |
| + | 0 | 0 | 2 | 2 | 2 | 3 | 3 |
| - | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| - | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

- memory depends on the number of leaves not examples
- if at the node the examples seen so far are not from the same class compute G for each attribute - (G=IG or Gini reduction(TODO name?)) we want to split at the biggest "impurity"

if $G(\text{bestattribute}) - G(\text{secondbestattribute}) > \sqrt{\frac{R^2 \ln 1/\delta}{2n}}$ split the node on the best

Very Fast Decision Tree (VFDT)

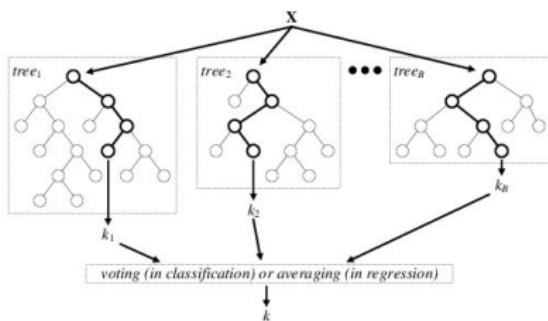
- Instead of computing the best attributes to split at every new instance, wait for n_{\min} instances
- to reduce memory usage - deactivate least promising node: that is lowest number for $p_l \times e_l$ for
 - p_l is the probability to reach leaf l
 - e_l error in node l
- method can be started from any existing tree → possibility to pretrain model on existing data (HT grows slowly and performance might be poor initially)

Numeric attributes

- much harder for streaming - **discretization**:
 - *Equal width* equal length bins simplest - but vulnerable to outliers and skewed distributions
 - *equal frequency* -same number of elements - need for sorting elements more processing time
 - *Fayyad and Irani's method* finding best cutting points as in decision trees - first sort then take each pair as a candidate - maximum IG - recursively - stop when intervals become clear

Ensemble

- combinations individual predictions of smaller models to form a final prediction
- voting, averaging, ...
- tend to improve prediction accuracy
- more time and memory
- parallelizable



Source: [Verikas et al., 2016]

Weighted voting

- simplest method
- returns most popular class
- for C_i being class predicted by i th model, for binary classification and a suitable threshold θ :

$$C(x) = \begin{cases} 1 & \text{if } \sum_i w_i C_i(x) > \theta \\ 0 & \text{otherwise} \end{cases} \quad (34)$$

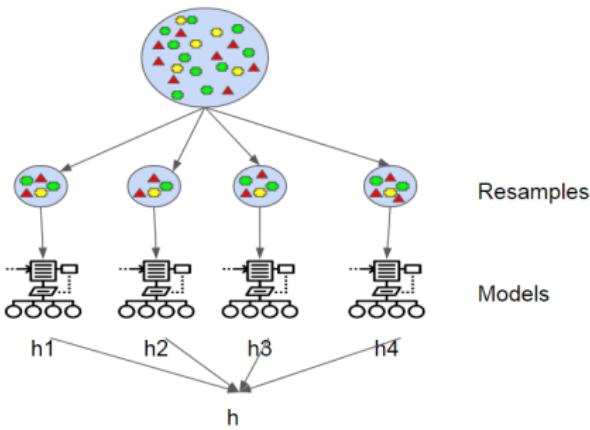
- weights can be fixed, or vary over time

Accuracy-Weighted Ensembles(AWE)[Wang et al., 2003]

- mine evolving datastreams using nonstreaming learners
- stream is processed in chunks
- new classifier for each chunk
- every time new classifier added, oldest removed
- [Wang et al., 2003] proposes to have $w_i \propto err_r - err_i$ for err_r the error of random classifier

Bagging[Breiman, 1996]

- use a base learning algorithm to produce M model by *bootstrapping*:
 - take samples from the data with *replacement*
- prediction by majority vote



Source:
<https://commons.wikimedia.org/wiki/File:Bagging.png>

Online bagging

- difficult to implement sampling with replacement
- simulation instead:
 - in a bootstrap replica the number of copies K for each of the n samples:

$$P(K = k) = \binom{n}{k} p^k (1-p)^{n-k} = \binom{n}{k} \frac{1}{n}^k (1 - \frac{1}{n})^{n-k} \quad (35)$$

- for large value of n sample Binomial \rightarrow Poisson(1) distribution, for

$$\text{Poisson}(1) = \frac{e^{-1}}{k!} \quad (36)$$

- OzaBag[Oza and Russell, 2001][Oza, 2005]: for each example the models will be updated with the weight of the example Poisson(1)

OzaBag

ONLINE BAGGING(*Stream*, M)

Input: a stream of pairs (x, y) , parameter M = ensemble size

Output: a stream of predictions \hat{y} for each x

- 1 initialize base models h_m for all $m \in \{1, 2, \dots, M\}$
- 2 **for** each example (x, y) in *Stream*
 - 3 **do** predict $\hat{y} \leftarrow \arg \max_{y \in Y} \sum_{t=1}^T I(h_t(x) = y)$
 - 4 **for** $m = 1, 2, \dots, M$
 - 5 **do** $w \leftarrow \text{Poisson}(1)$
 - 6 update h_m with example (x, y) and weight w

Bernoulli

Describes a single trial with 2 possible outcome: success $X = 1$ and failure $X = 0$

Pmf: $P(X = x) = p^x(1 - p)^{1-x}$ for $x = 0$ or 1

$$\mu = p$$

$$\sigma^2 = p(1 - p)$$

Significance

Common discrete prob distributions are built on the assumption of independent Bernoulli trials:

- **Binomial**: number of successes in n independent Bernoulli trials,
- **Geometric**: distribution of the number of trials to get the first success in independent Bernoulli trials,
- **Negative Binomial**: distribution of the number of trials to get the r th success in independent Bernoulli trials.

Binomial Distribution

$$\text{Binomial coefficient: } \binom{n}{x} = \frac{n!}{x!(n-x)!} \quad (37)$$

Assumptions

- n independent trials = an outcome does not give any information on other trials
- each trial result in one of two possible outcomes success or failure

$$p = P(\text{success}), P(\text{failure}) = 1 - p$$

Binomial distribution

number of successes in n independent Bernoulli trials has a binomial distribution

X represents the number of successes in n trials

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}, \text{ for } x = 1, \dots, n$$

$$\mu = \mathbb{E}(X) = np \quad \sigma^2 = np(1-p)$$

$p^x (1-p)^{n-x}$ probability of one specific ordering, and there are $\binom{n}{x}$ possible orderings

Poisson Distribution

Counting the number of occurrences of an event in a given unit of time.

Assumption

- events occur independently - given that one event happened gives no information at all when another even will occur

probability that an event occurs in a given length of time does not change over time

Poisson distribution

X the number of events in a fixed unit of time has a Poisson distribution

$$\text{Pmf: } p(x) = P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!} \quad x = 0, 1, \dots, \infty$$

$$\text{Mean: } \mu = \lambda$$

$$\text{Variance: } \sigma^2 = \lambda$$

Binomial and Poisson

Relationship

- is useful to decide whether a variable has a Poisson distribution or not
- The Poisson distribution with $\lambda = np$ closely approximates the Binomial distribution if n large and p small
- that is Bin \rightarrow Poi, as $m \rightarrow \infty, p \rightarrow 0,$

YouTube - for the distributions

Bernoulli: https://www.youtube.com/watch?v=bT1p5tJwn_0&list=PLvx0uBpazmsNIHP5cz37o0PZx0JKyNsZN&index=3

Binomial:<https://www.youtube.com/watch?v=qIzC1-9PwQo&list=PLvx0uBpazmsNIHP5cz37o0PZx0JKyNsZN&index=4>

Poisson:

<https://www.youtube.com/watch?v=jmqZG6roVqU&t=165s>

Bibliography I



Bifet, A., Gavaldà, R., Holmes, G., and Pfahringer, B. (2018).
Machine Learning for Data Streams with Practical Examples in MOA.
MIT Press.
<https://moa.cms.waikato.ac.nz/book/>.



Breiman, L. (1996).
Bagging predictors.
Machine learning, 24(2):123–140.



Domingos, P. and Hulten, G. (2000).
Mining high-speed data streams.
In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80.



McCallum, A., Nigam, K., et al. (1998).
A comparison of event models for naive bayes text classification.
In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer.



Oza, N. C. (2005).
Online bagging and boosting.
In *2005 IEEE International conference on systems, man and cybernetics*, volume 3, pages 2340–2345. Ieee.

Bibliography II



Oza, N. C. and Russell, S. (2001).

Experimental comparisons of online and batch versions of bagging and boosting.
In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 359–364.



Verikas, A., Vaiciukynas, E., Gelzinis, A., Parker, J., and Olsson, M. C. (2016).
Electromyographic patterns during golf swing: Activation sequence profiling and prediction of shot effectiveness.
Sensors, 16:592.



Wang, H., Fan, W., Yu, P. S., and Han, J. (2003).

Mining concept-drifting data streams using ensemble classifiers.
In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 226–235.



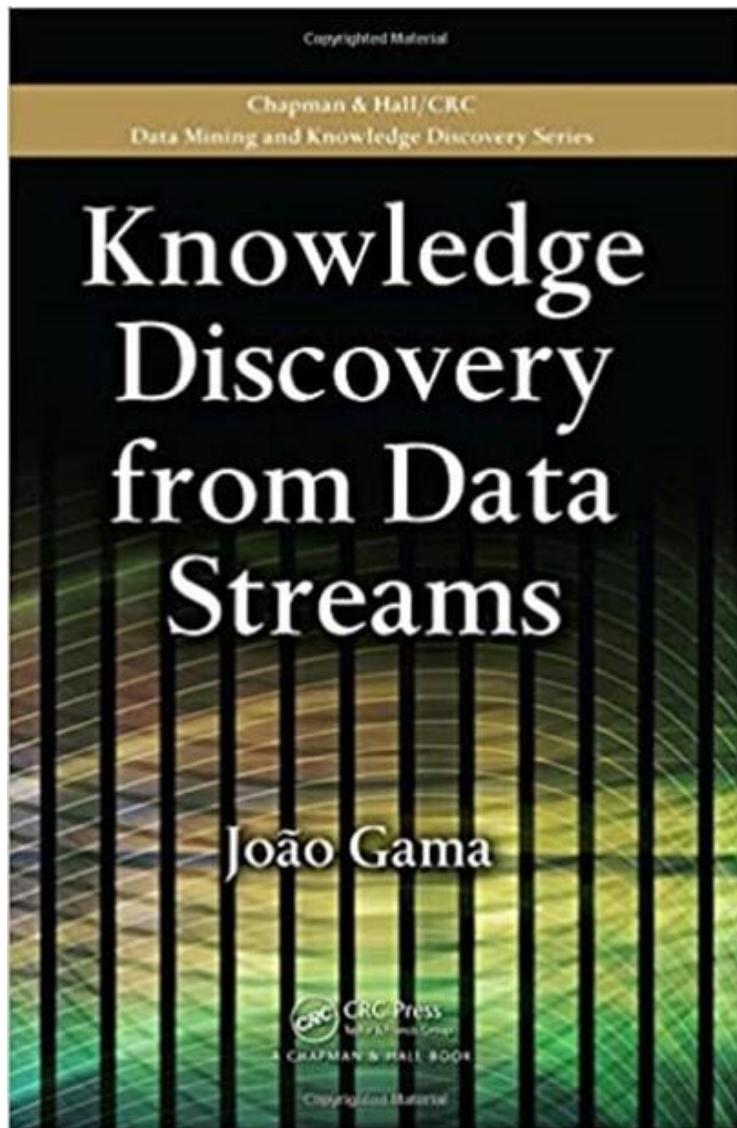
TIME SERIES ANALYSIS FOR DATA STREAMS

Stream mining (SM)

Imre Lendák, PhD, Associate Professor

Szegedi Gábor, PhD Student

Overview & lecture topics



- Introduction
- Time series categories
- Trends & seasonality
- Time series similarity
- Clustering
- Classification
- Anomaly detection
- Forecasting

- **Note:** we use a diverse range of other sources besides the KDDS book

Definitions

- **DEF:** **Time series** are sequences of measurements that follow non-random orders.
- Time series X notation:
$$x_1, x_2, \dots, x_{t-1}, x_t, \dots$$
- **DEF:** **Time series analysis** applies different data analysis techniques to model dependencies in the sequence of measurements
- Common components of time series analysis
 - **Trend** = represents a general systematic linear or (most often) nonlinear component that changes over time
 - **Seasonality** = represents re-occurring patterns appearing in systematic intervals over time
 - **Cycle** = the data exhibit rises and falls that are not of a fixed frequency.
 - **Noise** = a non-systematic component that is nor trend or seasonality within the data

Common use cases

- **Classification**, e.g. disease identification based on a one-off ECG diagram
 - **Clustering**, e.g. identify novel patterns in large sets of medical measurements or financial data
 - **Anomaly detection**, e.g. anomalous reading in an ECG data stream of a hospitalized patient → urgent reaction by the hospital staff
 - **Forecasting**, e.g. predict future FOREX pair values based on historical data
-
- At least for classification, clustering and anomaly detection it is necessary to be able to (as) exactly (as possible) measure the **similarity** between time series

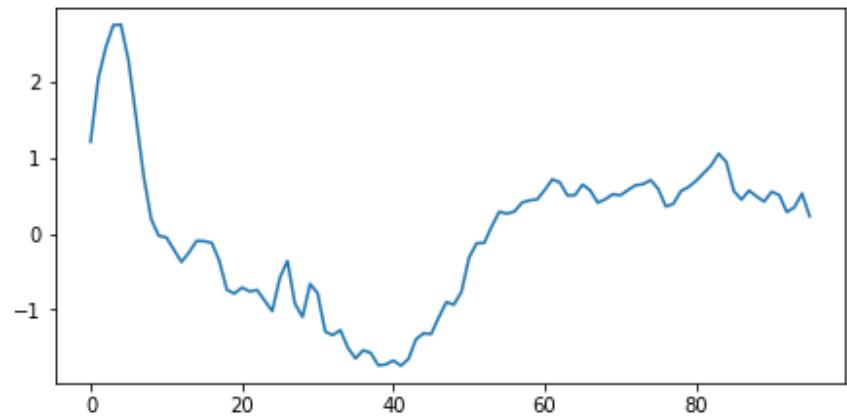
TIME SERIES CATEGORIES

Sample dimensionality categories

- We can categorize time series data based on what is the dimensionality of a single sample
- **Univariate:** A sample in the sequence is a single value.
 - Example: Daily changes in the average temperature.
- **Multivariate:** A sample in the sequence is a vector.
 - Example: Daily closing values of all the stocks on the NYSE stock market.
- **Complex:** A sample in the sequence is of higher dimension.
 - Example: A video feed.

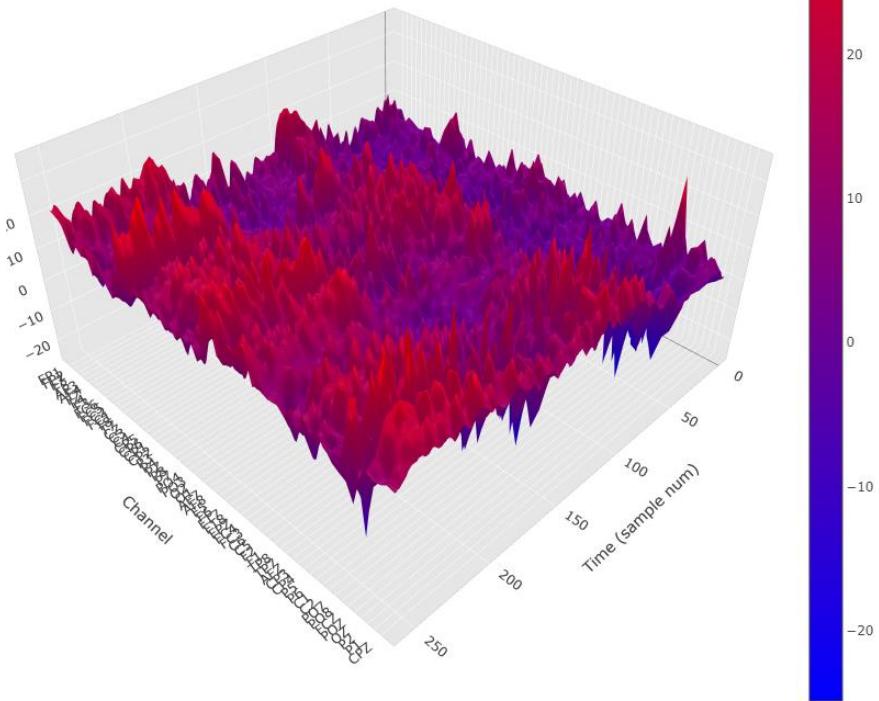
Univariate time series

- A single variable as a function of time
 - E.g. a single load measurement in electric power systems, a flow meter in a water management system, stock price
- $TS = (x_1, \dots, x_n), x_i \in R$

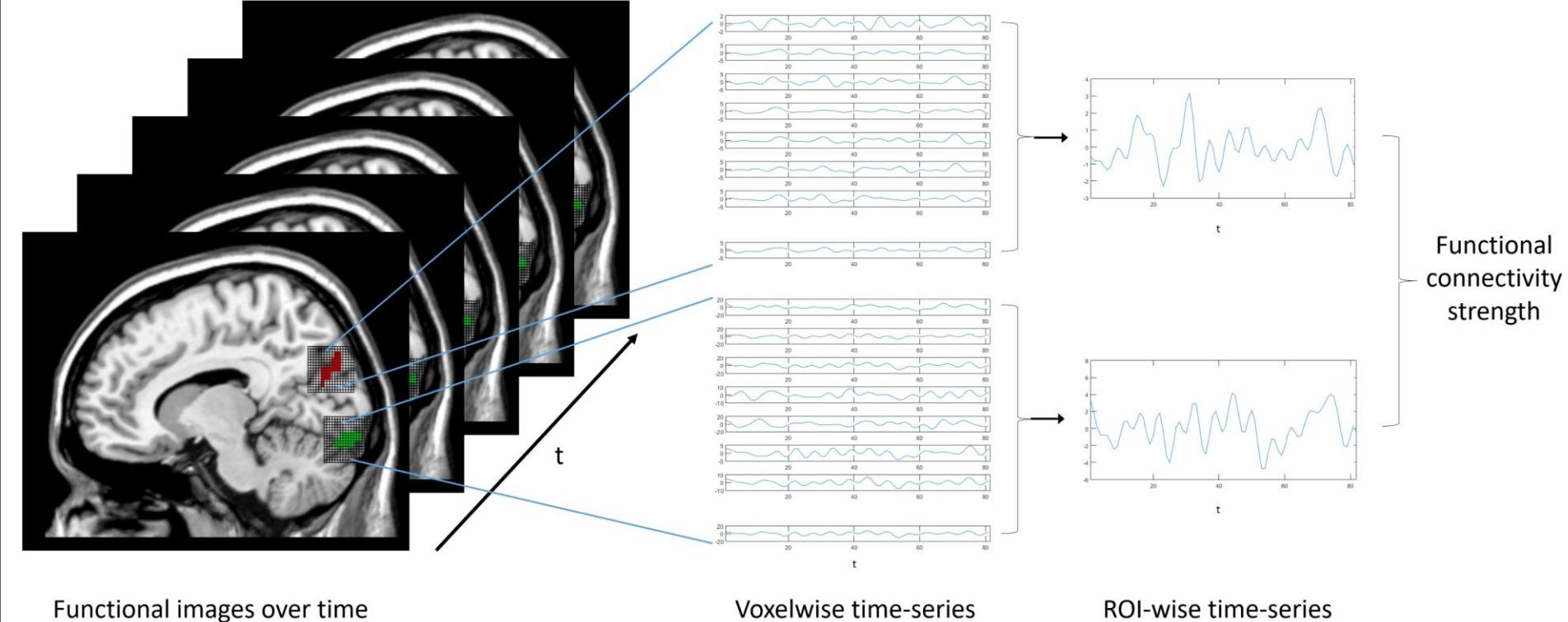


Multivariate time series

- Sequence of vectors
 - E.g. measurements describing weather conditions, ECG, EEG



Complex time series

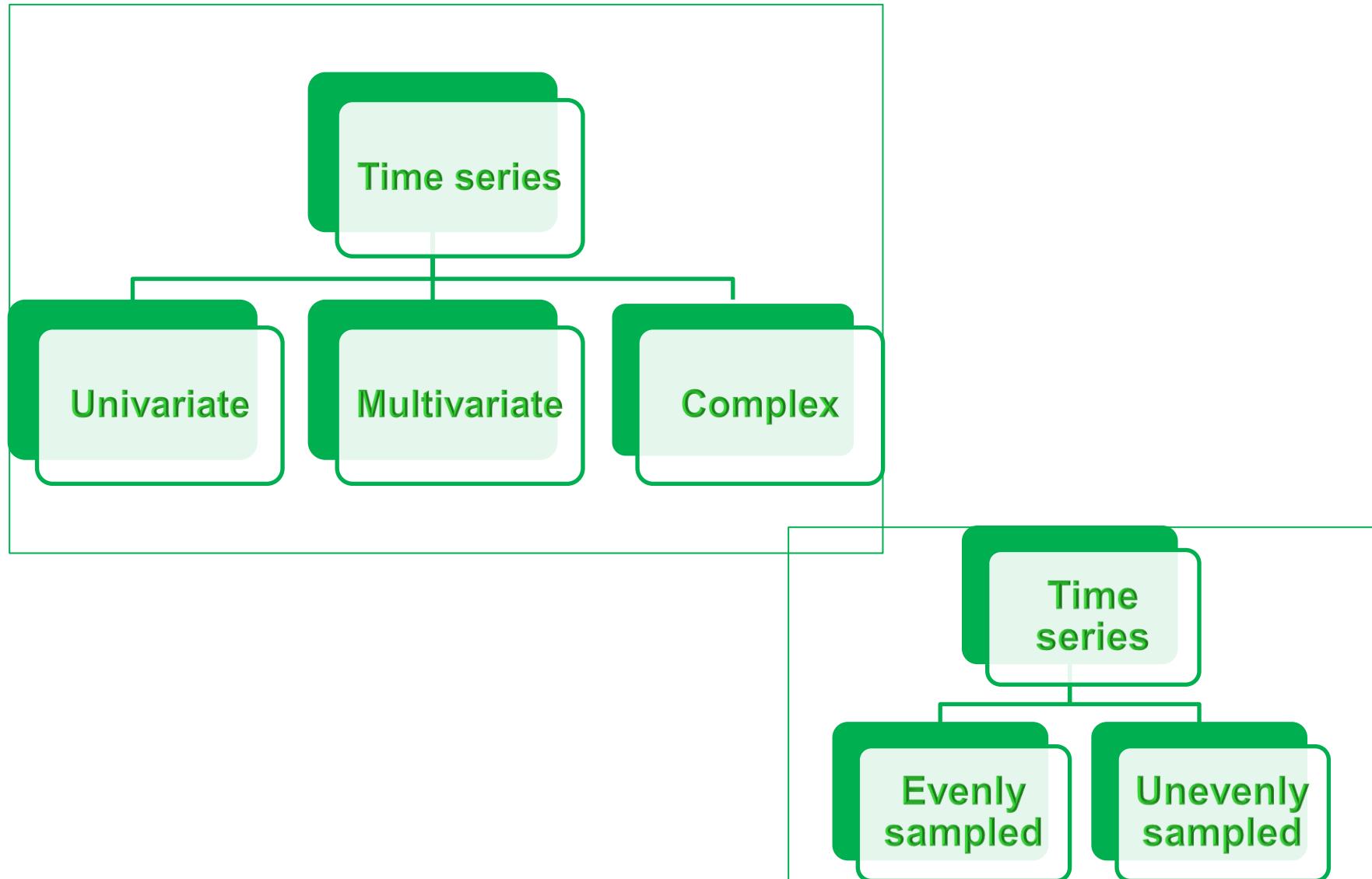


- E.g. functional magnetic resonance imaging (fMRI) data
- May be transformed to simpler time series for analysis

Sample frequency classification

- We can categorize time series data based on what is the frequency of samples (the time between 2 samples)
- **Evenly sampled:** Samples are distributed evenly during the time span of the series.
 - E.g. load measurement(s) in electric power systems sampled every 15 minutes for trading and forecasting
- **Unevenly sampled:** The frequency of samples is varying.
 - Each record is associated with a timestamp, but the frequency of samples is varying.
 - $TS = (t_1: x_1, \dots, t_n: x_n)$
 - Note: observations x_i can be of any datatype
 - E.g. blood pressure of a patient (self-)measured twice a day, but at different times

Categorization summary



TRENDS

Trend primer



<https://www.babypips.com/learn/forex/using-moving-averages>

Trend intro

- **DEF:** A **trend** is a general systematic linear or (most often) nonlinear component that changes over time
- Trend-related challenges:
 - What is the mean of a time series with a trend? Or multiple trends?
 - What are the distributions of values?
 - Moving averages lag behind trends
- **DEF:** A **trend reversal** occurs when the direction of an existing trend changes (to the opposite)
 - Trend reversals are quite important in financial data analytics



speedtrader.com › methods-for-determining-trend-reversals

Trend analysis – 2

- **Moving averages** are used in trend detection → smooth out short-term fluctuations in the data → highlight longer-term trends or cycles
- **Averaging methods** → all items have the same relevance
- **Weighted averaging methods** → data points are associated with weights which depict their relevance

Moving averages

- **Moving average (MA)** = the mean of the previous n data points:

$$MA_t = MA_{t-1} - \frac{x_{t-n+1}}{n} + \frac{x_{t+1}}{n}$$

- **Cumulative moving average (CA)** = the average of all of the data up until the current data point

$$CA_t = CA_{t-1} - \frac{x_t - CA_{t-1}}{t}$$

Weighted moving averages

- **Weighted moving average** = different weights to different data points, usually the most recent data points are more “important”

$$WMA_t = \frac{nx_t + (n - 1)x_{t-1} + \cdots + 2x_{t-n+2} + x_{t-n+1}}{n + (n - 1) + \cdots + 2 + 1}$$

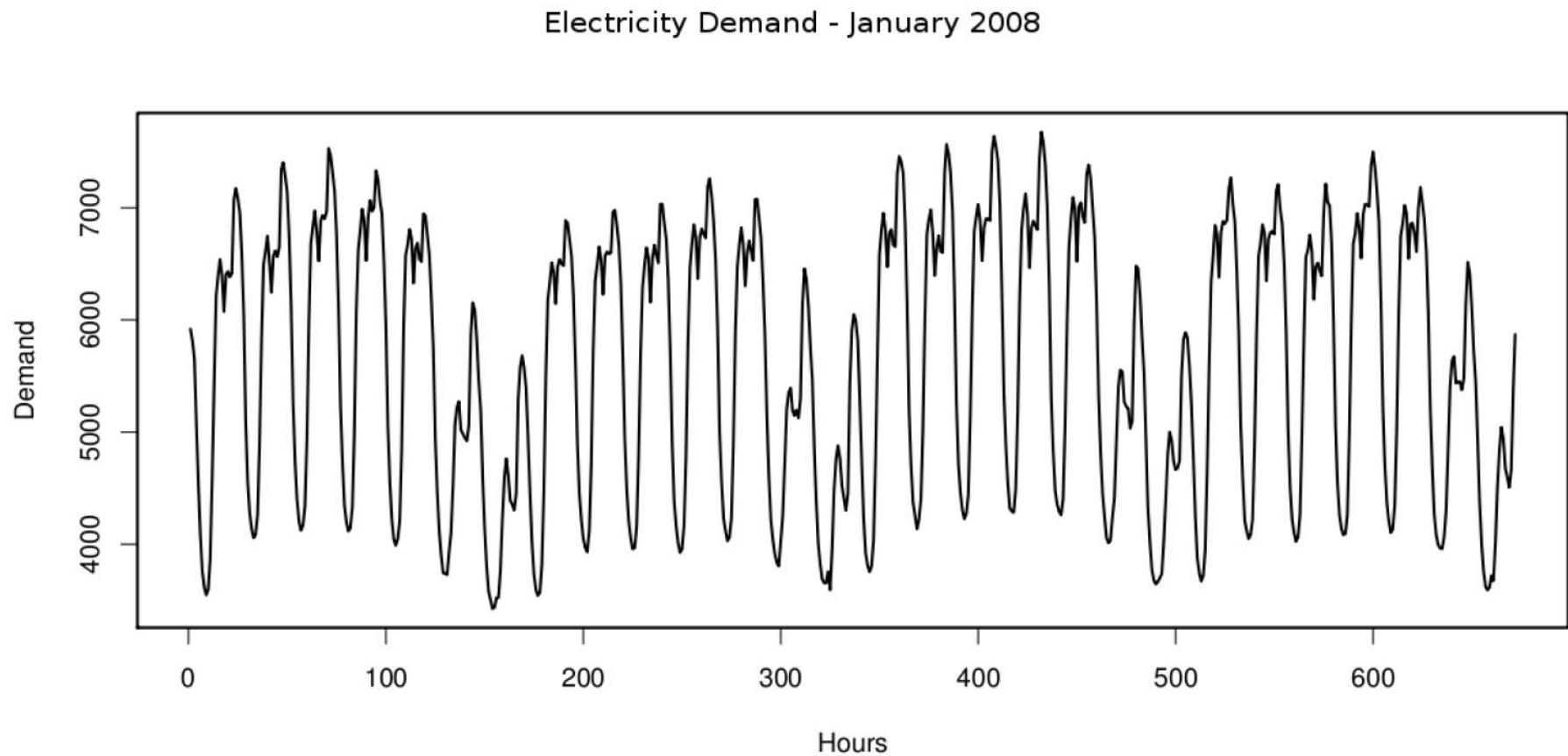
- **Exponential moving average** = the weighting for each older data point decreases exponentially, giving more importance to recent observations while still not discarding older observations entirely

$$EMA_t = \alpha \times x_t + (1 - \alpha) \times EMA_{t-1}$$

- **Note 1:** more weight to recent items
- **Note 2:** choosing an adequate α is a difficult problem.

SEASONALITY

Seasonality primer



Gama J. Knowledge discovery in data streams. CRC Press. 2010.

Seasonality intro

- **DEF:** Seasonality is a time series characteristic which signifies regular and predictable changes
 - E.g. different (electricity) load patterns occurring yearly (or different time periods, e.g. weeks)
- A simple way to remove the seasonal component is differencing
- Seasonality is caused by various external and internal factors affecting the system under observation and producing the time series
 - Weather conditions → less travel during icy periods
 - Vacation periods → lower electricity consumption if people travel (not during covid)
 - Other sources? Discuss!

Seasonality and autocorrelation

- **DEF:** **Correlation** is a statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate)

$$\text{corr}_{x,y} = \frac{\text{cov}_{x,y}}{\sigma_x \sigma_y} = \frac{E[(x - \mu_x)(y - \mu_y)]}{\sigma_x \sigma_y},$$

x, y : random variables

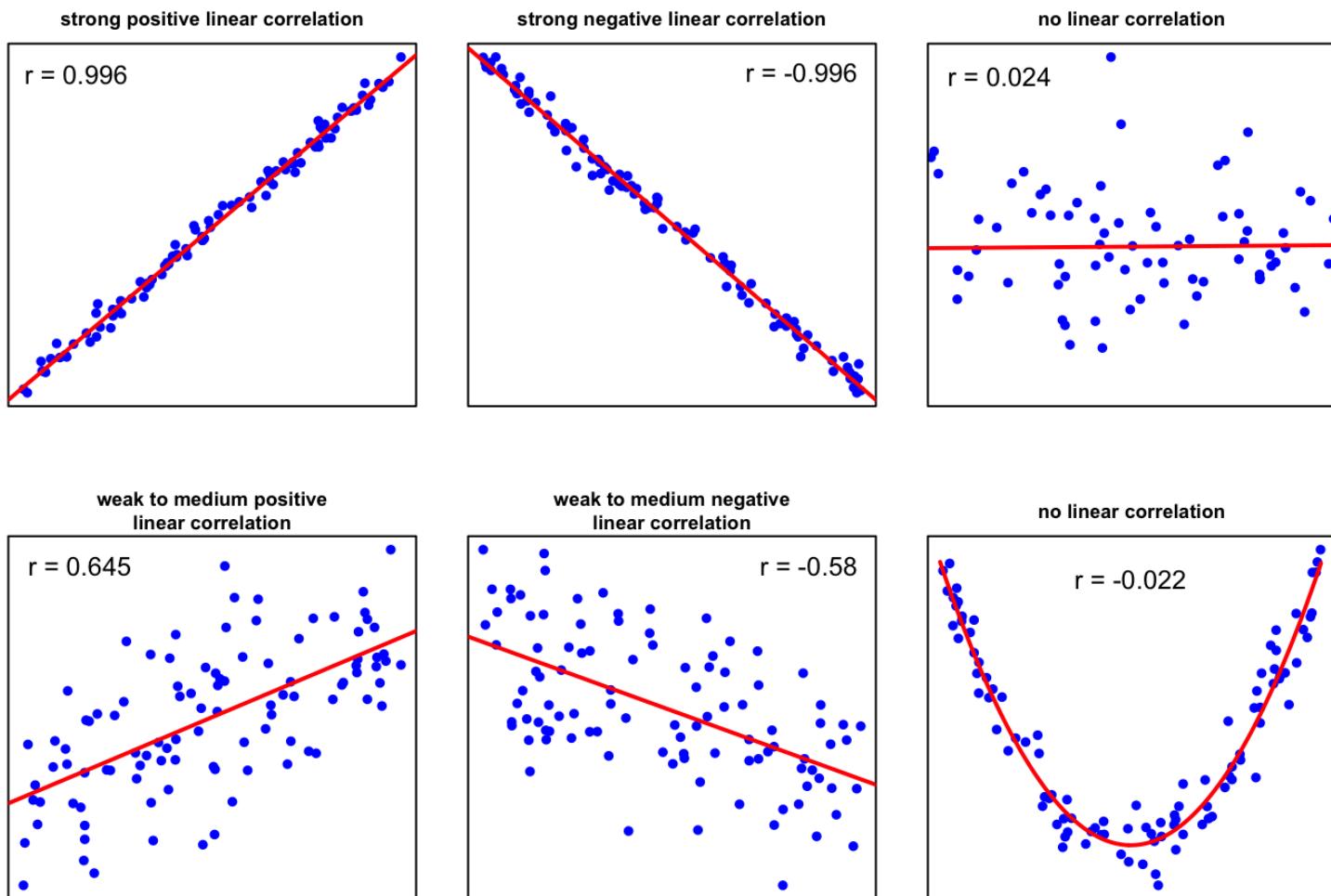
μ_x, μ_y : expected values

σ_x, σ_y : standard deviations

- **DEF:** **Autocorrelation** is the correlation of a signal with a delayed copy of itself as a function of delay
 - Autocorrelation is the cross-correlation of a time-series with itself

$$r(x, l) = \frac{\sum_{i=1}^{n-l} (x_i - \bar{x})(x_{i+l} - \bar{x})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

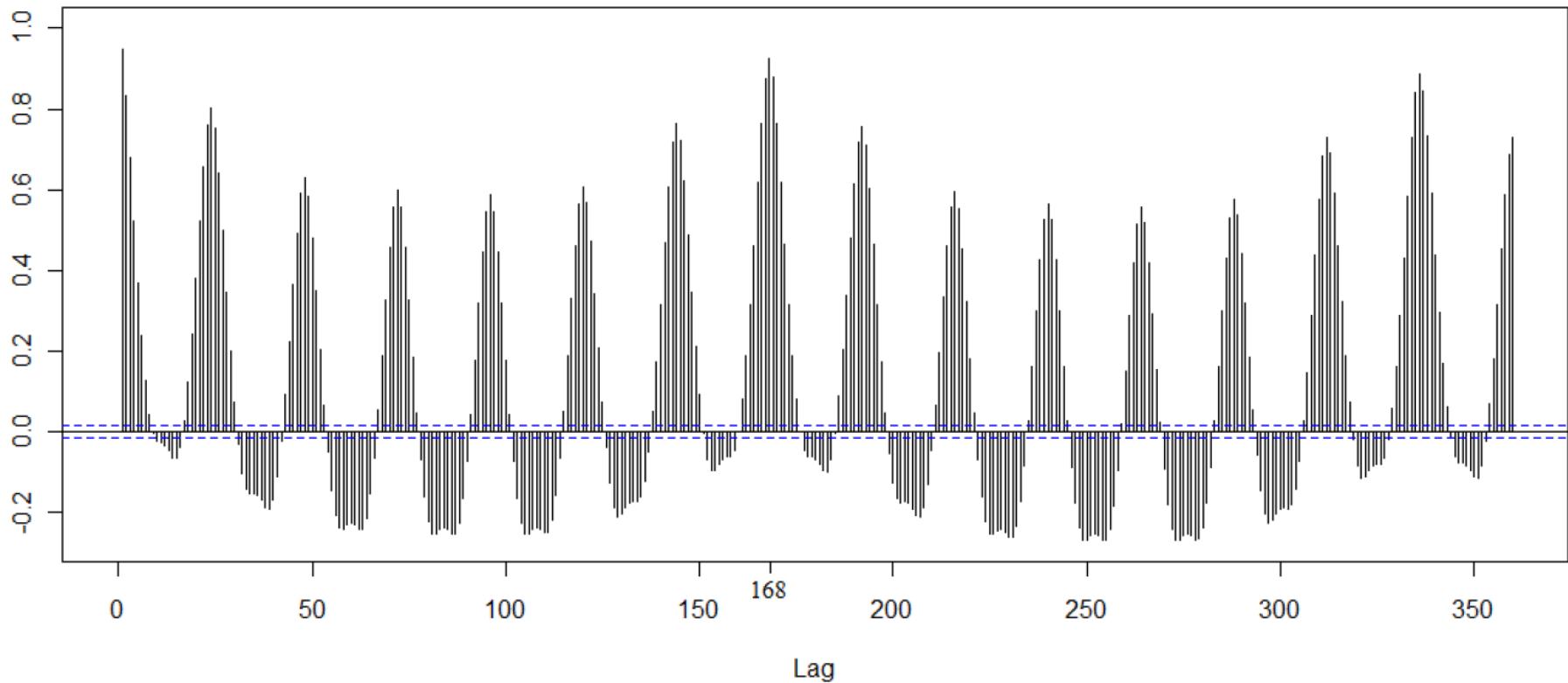
Correlation types



<https://www.geo.fu-berlin.de/en/v/soga/Basics-of-statistics/Descriptive-Statistics/Measures-of-Relation-Between-Variables/Correlation/index.html>

Power system data correlogram

Autocorrelation (1 Hour – 2 Weeks)



Gama J. Knowledge discovery in data streams. CRC Press. 2010.

Seasonality and autocovariance

- DEF: Covariation is a measure of the joint variability of two random variables

$$cov_{x,y} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N - 1},$$

x, y: random variables

\bar{x}, \bar{y} : means of x and y

N: number of values

- Positive covariance → greater values of variable x correspond to greater values of variable y
- Negative covariance → greater values of variable x correspond to smaller values of variable y
- Note:** Both autocorrelation and autocovariance are useful statistics to detect periodic signals

SIMILARITY

Motivation

- **Similarity measures** are necessary for most time series analysis types
 - Assess distance between time series → form clusters
 - Measure distance from classes → assign to classes
 - Lack of (any) similarity → might signify an anomaly
- **Similarity criteria** in time series analysis can be based on
 - Raw data similarity
 - Time series feature similarity
 - Similarity between the underlying (data) generating processes (i.e. model)

Euclidean distance

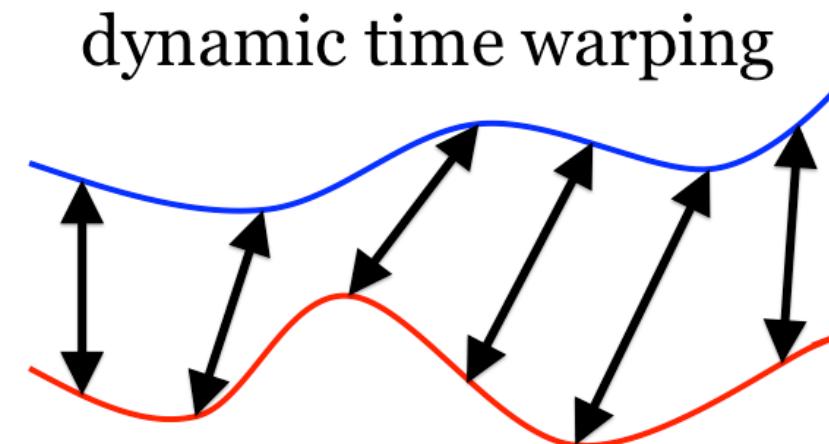
- **DEF:** The Euclidean distance between two time series is the square root of sum of the squared distances between each pair of points between 2 time series
 - Time series alignment is necessary

$$D(C, Q) = \sqrt{\sum_{i=1}^n (q_i - c_i)^2}$$

- Satisfies the 4 properties of distance:
 - Identity: $D(Q, Q) = 0$
 - Non-negative: $D(C, Q) \geq 0$
 - Symmetric: $D(C, Q) = D(Q, C)$
 - Satisfies the triangular inequality: $D(Q, C) + D(C, T) \geq D(Q, T)$

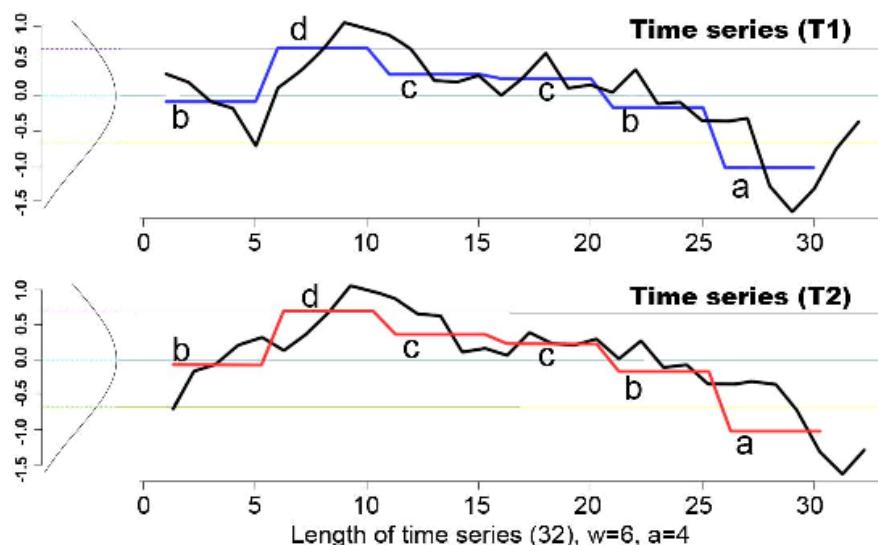
Dynamic Time Warping (DTW)

- Dynamic Time Warping (DTW) is a distance measure for comparing two temporal sequences, which may vary in speed → no alignment needed
- Time complexity: $O(N^2)$
- **Note:** does not allow time scaling of segments
- **Trivia:** A well-known use case is speech recognition with different speaking speeds



<https://www.mathworks.com/matlabcentral/fileexchange/43156-dynamic-time-warping-dtw>

Symbolic Aggregate Approximation (SAX)

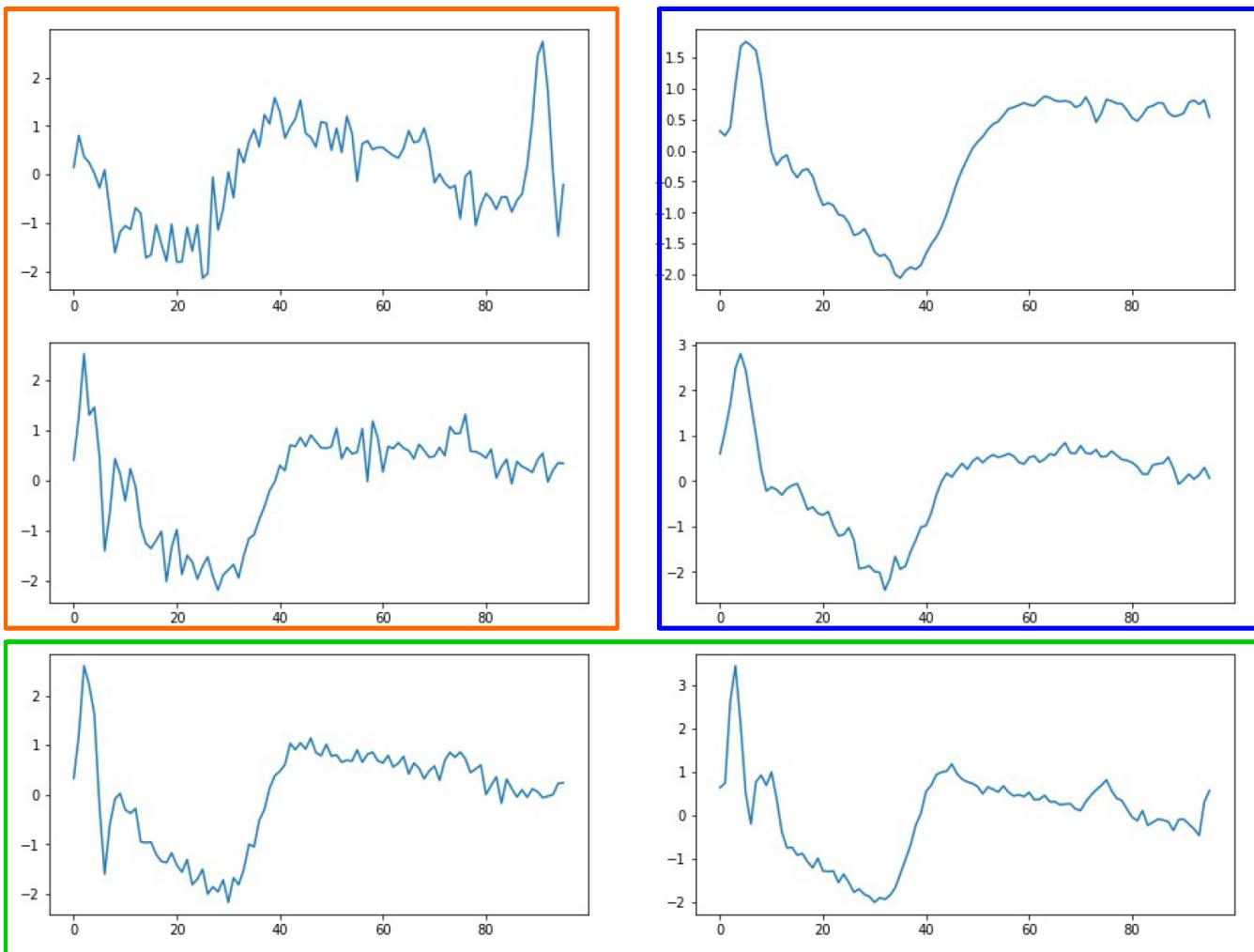


- DEF: Symbolic Aggregate Approximation (SAX) transforms a time series into a string of characters
- Complexity: $O(N)$
- Steps:
 - Piecewise Aggregate Approximation (PAA)
 - Symbolic Discretization
 - Distance Measure
- SAX use cases:
 - Motifs = previously unknown frequent patterns
 - Discords = the most unusual time series sub-sequence

<https://www.semanticscholar.org/paper/An-improved-symbolic-aggregate-approximation-based-Zan-Yamana/bf8267be7a70b1f9df982155f12c5786451c7756>

CLUSTERING

Clustering primer

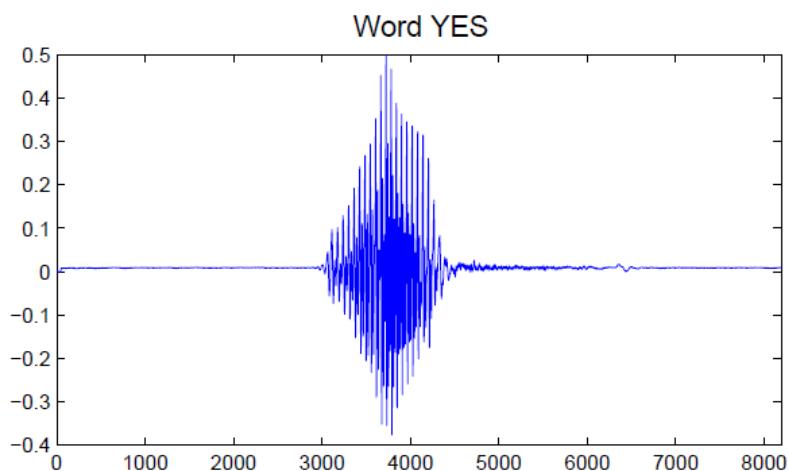
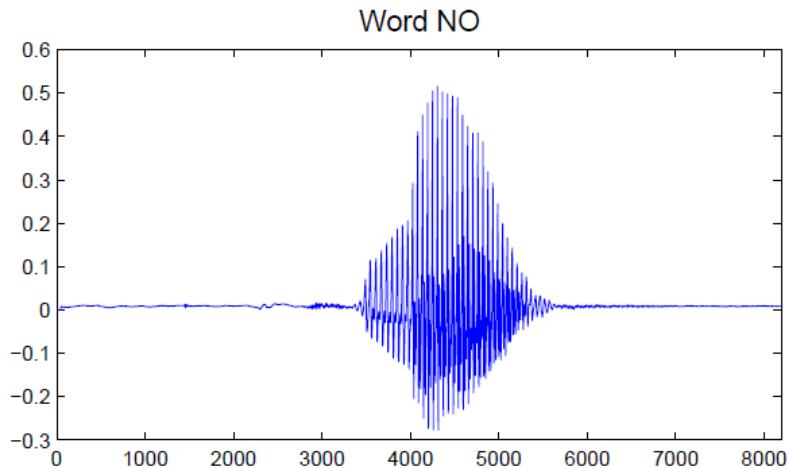


<http://www.biointelligence.hu/pdf/timeseriestutorial.pdf>

Problem definition and approaches

- **DEF:** In time **series clustering problems** multiple time series (or slices of a single time series) are analyzed with the goal to group them or their subsets into different clusters
- Latest application domains: finance, medicine, seismology, meteorology, etc.
- Approaches
 - Raw data clustering → direct
 - Clustering by features → indirect, based on features
 - Model-based clustering → indirect, based on a model
- Key time series clustering reference: Andrés M. Alonso's slides from 2019 (unless otherwise stated)

Raw data classification



- DEF: Raw data clustering measures the **element-wise distance** between two (or more) time series

$$D(x_i, x_j) = d(x_i - x_j)$$

- The series need to be **perfectly aligned** → hard to achieve in real-life use cases
- Other raw data approaches: autocorrelation, extreme value

Feature-based clustering

- **DEF:** **Feature-based clustering** relies on derived statistical features of the time series
 - Assumption: a finite set of statistical measures can be used to capture the global nature of the time series
- **Common** time series features: mean, standard deviation, skewness, periodicity
- **Less common** features: kurtosis, energy, entropy
 - TSFEL: Time Series Feature Extraction Library (60 features)
- Feature-based clustering **advantages**:
 - Reduced dimensionality of the original time series
 - Lower sensitivity to missing data
 - Ability to handle different lengths of time series

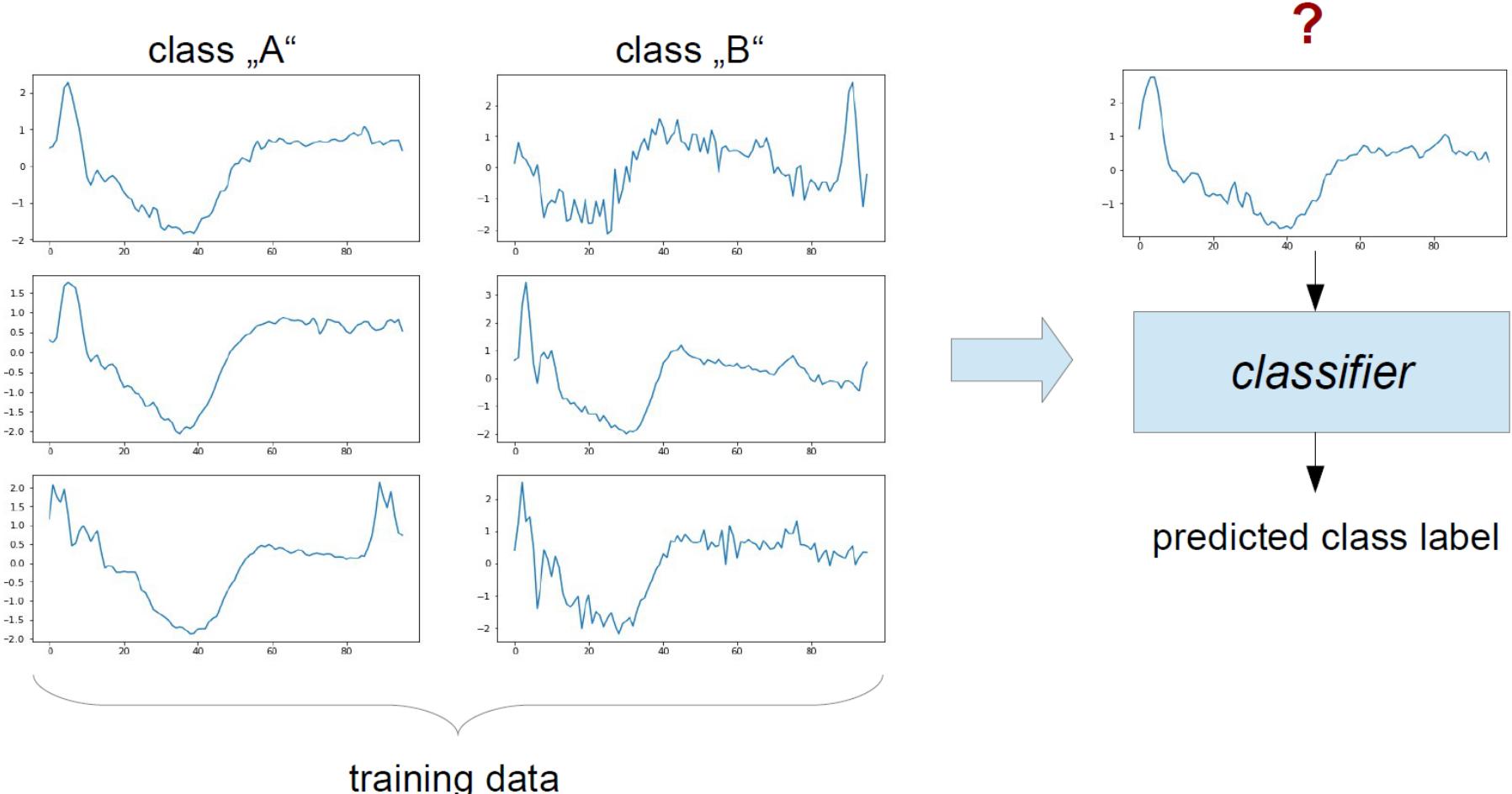
https://tsfel.readthedocs.io/en/latest/descriptions/feature_list.html

Model-based clustering

- DEF: **Model-based clustering** assumes that the data were generated by a model and tries to **recover the original model** from the data
- The model recovered from the data defines the clusters
 - In a K-means approach the model is a set of centroids which (are supposed to have) generated the data
- Advantages:
 - Low computational cost (if the model-matching is ‘cheap’)
- Disadvantages:
 - It might be challenging to derive a correct model
- Source: Stanford NLP Group

CLASSIFICATION

Classification primer



Buza K. Time Series Classification and its Applications, 8th International Conference on Web Intelligence, Mining and Semantics. June 25 – 27 2018, Novi Sad, Serbia.

Classification techniques

- **Similarity-based classification**, e.g. nearest neighbor, hubness-aware classifiers
 - Classification based on characteristic local patterns, e.g. motif-based, shapelet-based
- **Feature-based classification**
 - Feature extraction + a standard classifier such as SVM, Naive Bayes, decision tree...
 - Possible features: min, max, avg, std, etc.
- **Other** techniques:
 - Hidden Markov Models
 - Deep learning with CNN

Evaluation

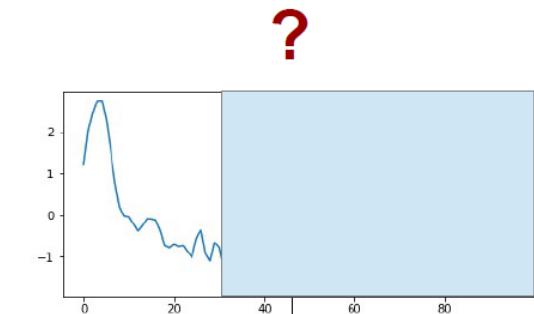
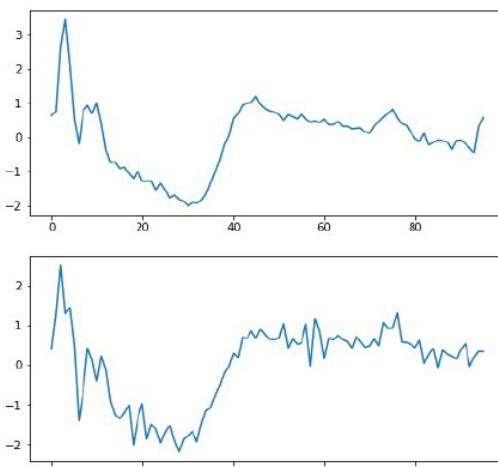
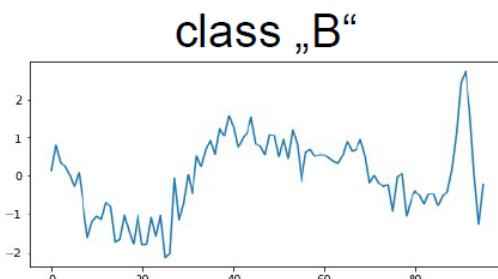
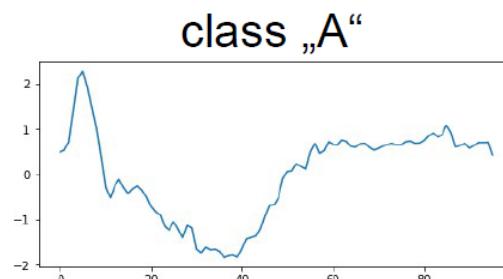
Evaluation protocol

- Simulate real-life applications and data as much as possible → why train a classifier it will not be used?!
- Independent test set
- Cross-validation

Evaluation metrics

- Accuracy, AUC, precision, recall, F-measure, AUPR
- Standard deviation, statistical significance tests
- Note: Be careful when evaluating any solution on unbalanced data

Stream mining challenge



predicted class label

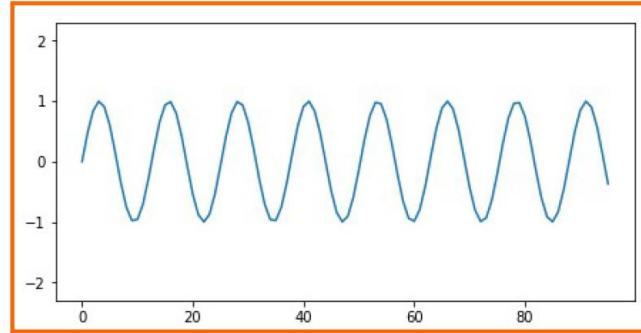
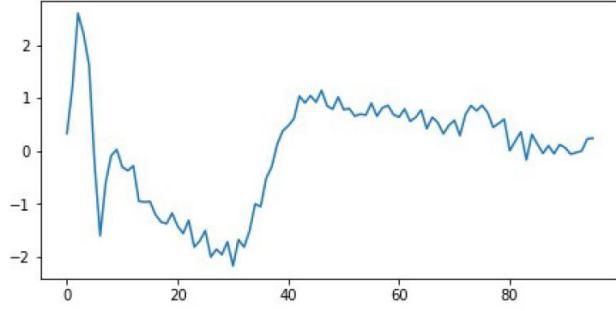
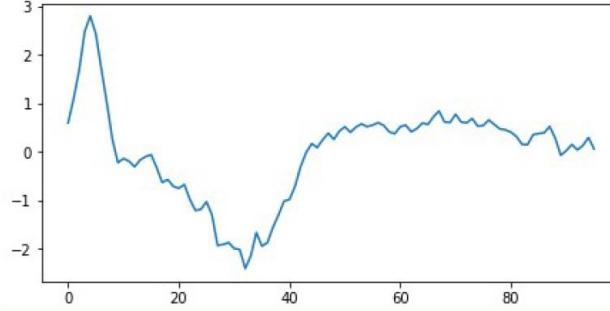
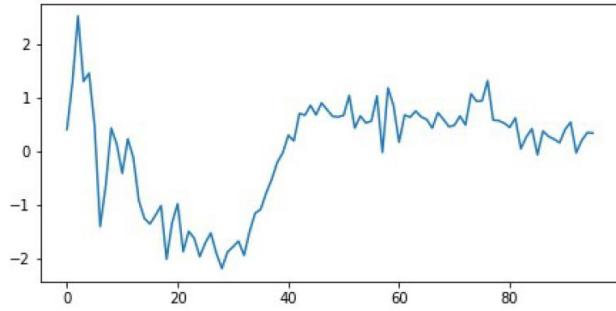
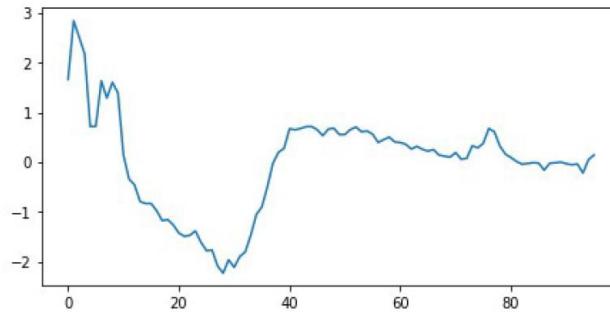
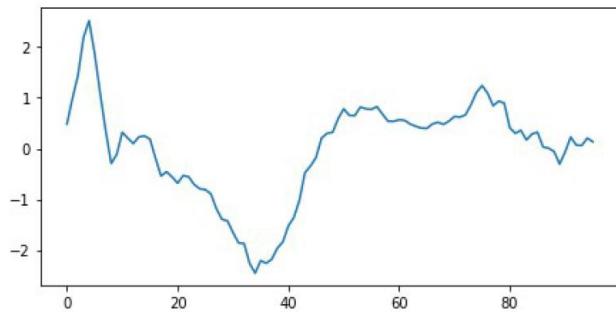
training data

Additional references

- Buza, Schmidt-Thieme (2009): **Motif-based** classification of time series with Bayesian networks and SVMs, Advances in Data Analysis, Data Handling and Business Intelligence. Springer, Berlin, Heidelberg, pp. 105-114
- Hills et al. (2014): Classification of time series by **shapelet** transformation, Data Mining and Knowledge Discovery, 28(4), pp. 851-881

ANOMALY DETECTION

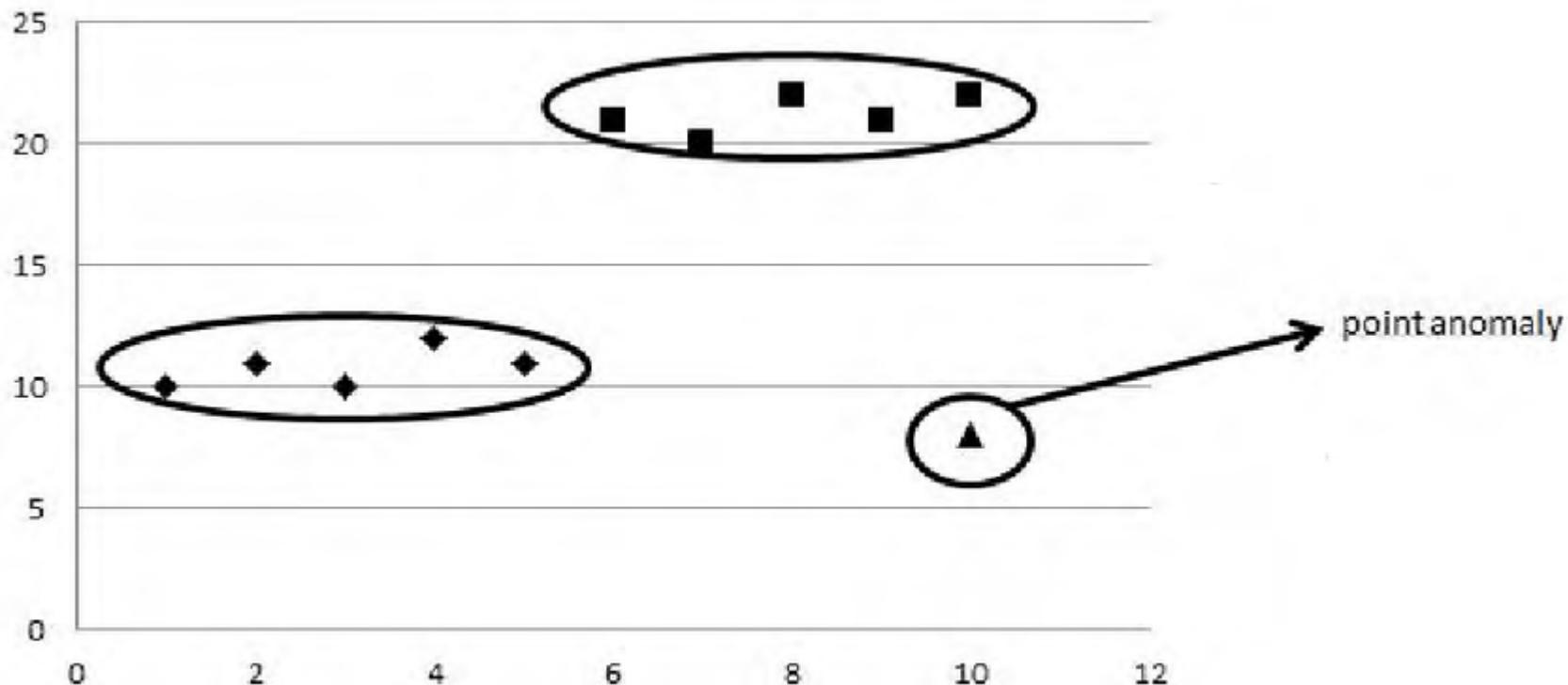
Anomaly detection primer



Buza K. Time Series Classification and its Applications, 8th International Conference on Web Intelligence, Mining and Semantics. June 25 – 27 2018, Novi Sad, Serbia.

Type #1: Point anomalies

- DEF: In a **point anomaly** an individual data instance is anomalous with respect to its surroundings



Baddar, S. W. A. H., Merlo, A., & Migliardi, M. (2014). Anomaly Detection in Computer Networks: A State-of-the-Art Review. *JoWUA*, 5(4), 29-64.

Type #2: Contextual anomalies

- DEF: In **contextual anomalies** a data instance is anomalous in specific context, but otherwise might be normal in other contexts.

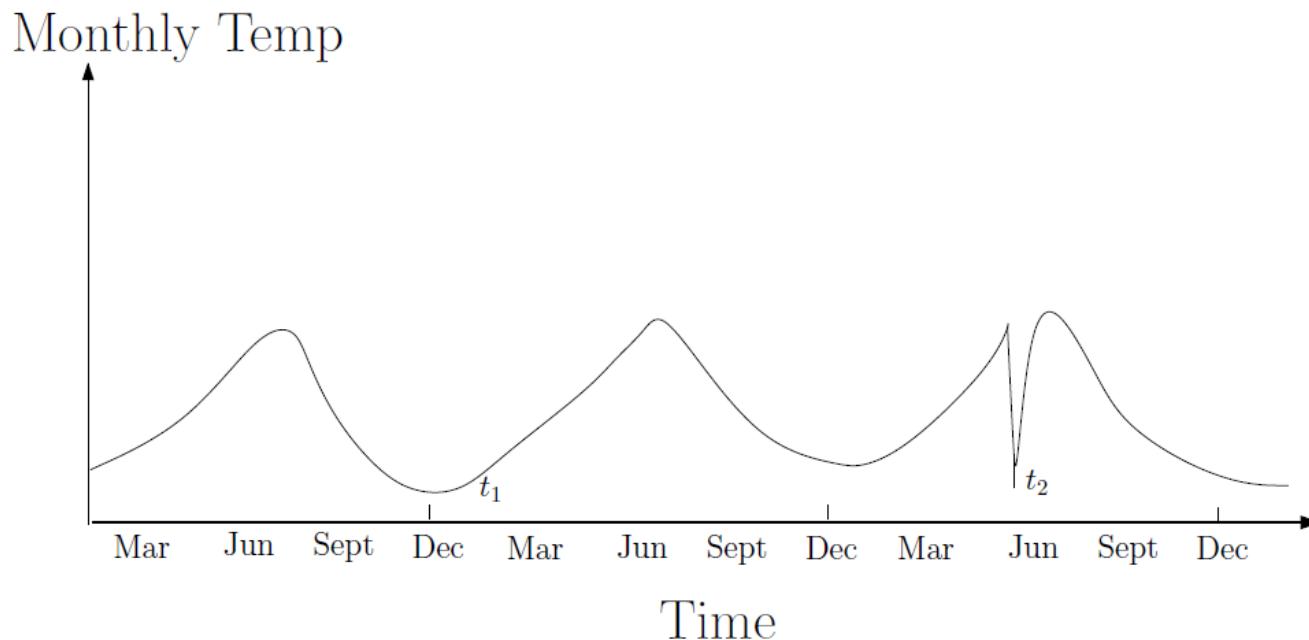


Fig. 3. Contextual anomaly t_2 in a temperature time series. Note that the temperature at time t_1 is same as that at time t_2 but occurs in a different context and hence is not considered as an anomaly.

Chandola V., Banerjee A., Kumar V., "Anomaly Detection: A Survey", Technical Report TR 07-017, 2007

Type #3: Collective anomalies

- DEF: **Collective anomalies** are collections of data instances anomalous in relation to the entire data set

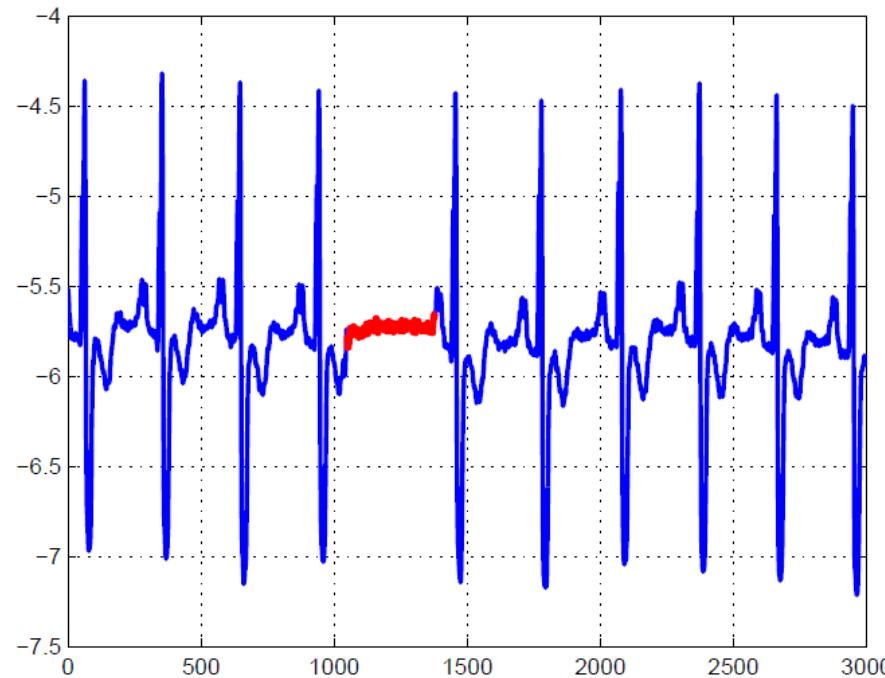


Fig. 4. Collective anomaly corresponding to an *Atrial Premature Contraction* in an human electrocardiogram output.

Chandola V., Banerjee A., Kumar V., "Anomaly Detection: A Survey", Technical Report TR 07-017, 2007

Anomaly detection techniques

- **Seasonal and Trend decomposition using Loess (STL)**
 - split time series into (season, trend, residue)
 - The residue element contains the anomalies
- **Classification** → applicable if there is labeled data → no class means outlier/anomaly
- **Auto Regressive Integrated Moving Average (ARIMA)**
 - predict future points → detect discrepancies
 - Several points in the past used to forecast next point + noise
- **Long short-term memory (LSTM)**
 - Malhotra, Pankaj; Vig, Lovekesh; Shroff, Gautam; Agarwal, Puneet (April 2015). "Long Short Term Memory Networks for Anomaly Detection in Time Series". ESANN 2015.
- + many other methods

Summary

- Introduction
- Time series categories
- Trends & seasonality
- Similarity
- Clustering
- Classification
- Anomaly detection



Common references

- Aggarwal, C. C. (2015). Data mining: the textbook. Springer.
 - Note: chapter “Mining time series data”
- Aghabozorgi, S., Shirkhorshidi, A. S., & Wah, T. Y. (2015). Time-series clustering – a decade review. *Information Systems*, 53, 16-38.
- Buza K. Time Series Classification and its Applications, 8th International Conference on Web Intelligence, Mining and Semantics. June 25 – 27 2018, Novi Sad, Serbia.
<http://www.biointelligence.hu/pdf/timeseriestutorial.pdf>
- Esling, P., & Agon, C. (2012). Time-series data mining. *ACM Computing Surveys (CSUR)*, 45(1), 1-34.
- Gama J. Knowledge discovery in data streams. CRC Press. 2010.
- Holan, S. H., & Ravishanker, N. (2018). Time series clustering and classification via frequency domain methods. *Wiley Interdisciplinary Reviews: Computational Statistics*, 10(6), e1444.
- Kotsakos, D., Trajcevski, G., Gunopulos, D., & Aggarwal, C. C. (2013). Time-Series Data Clustering.
- Maharaj, E. A., D'Urso, P., & Caiado, J. (2019). Time series clustering and classification. CRC Press.

Thank you for your attention!



SKETCHES, FREQUENT PATTERNS AND SEQUENCES

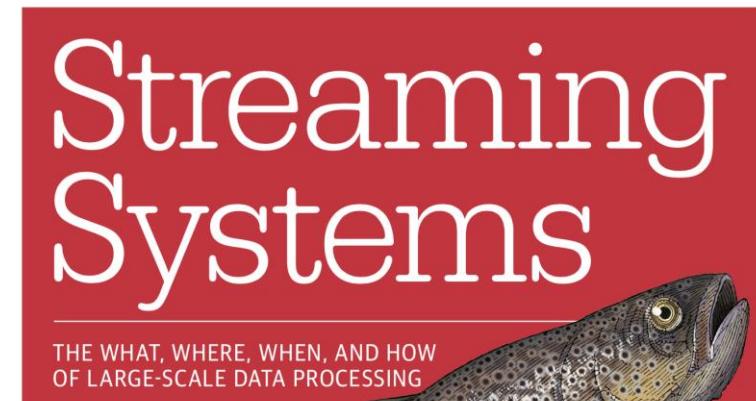
Stream mining (SM)

Imre Lendák, PhD, Associate Professor

Chandresh Maurya, PhD, Assistant Professor

Stream mining course status update

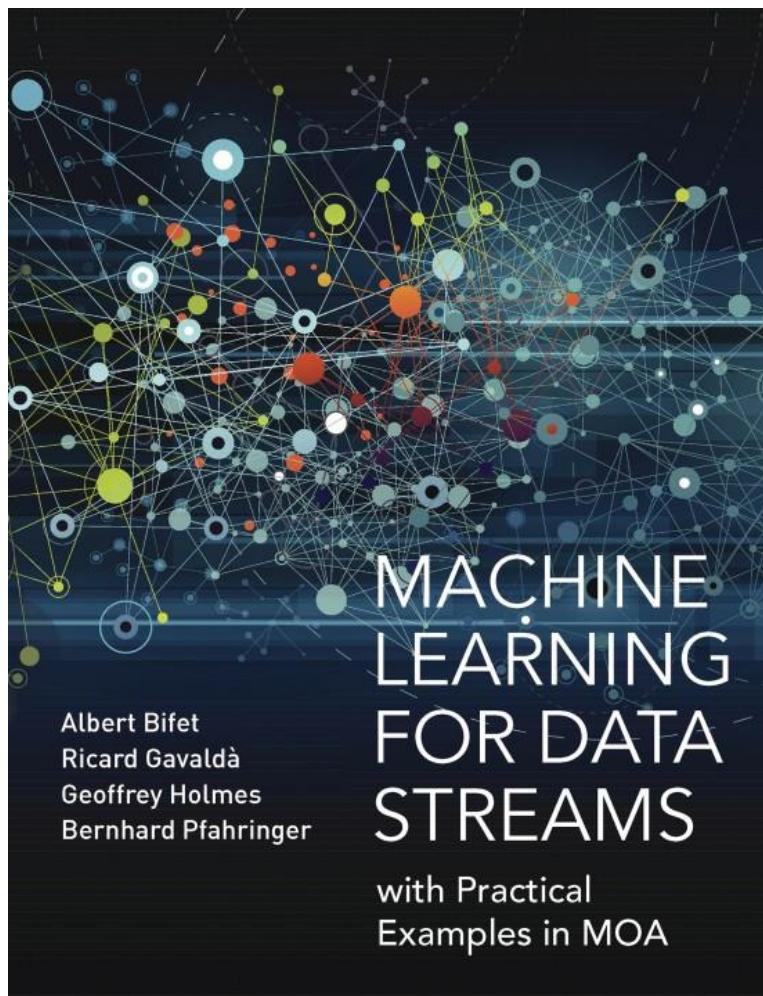
O'REILLY®



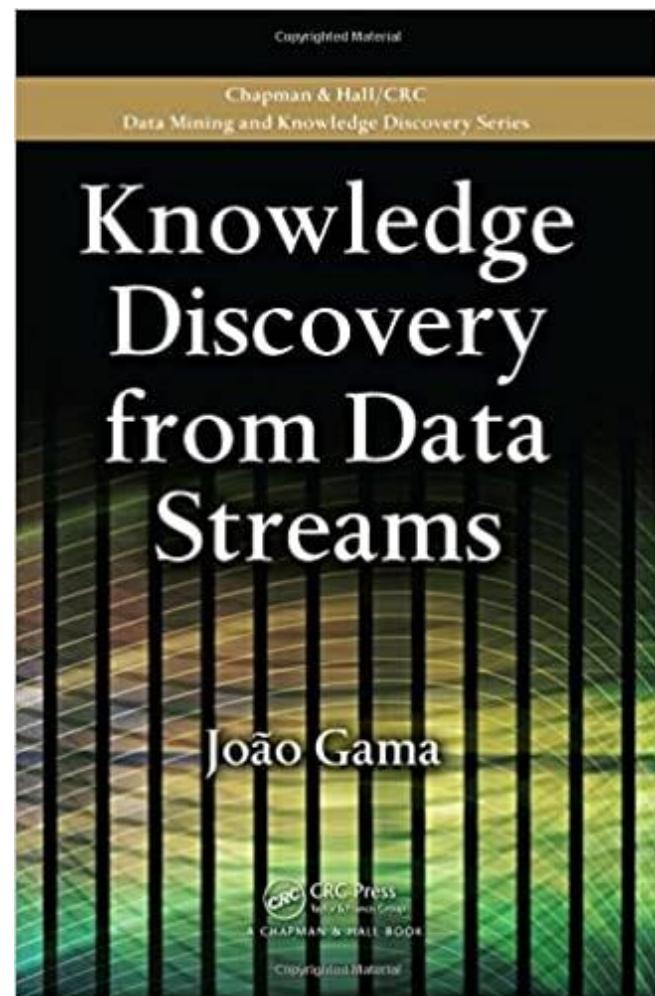
Tyler Akidau, Slava Chernyak
& Reuven Lax

- Started with the Streaming Systems book by Google
- In the stream mining domain we covered:
 1. Sampling and filtering
 2. Distinct elements and moments
 3. Clustering
 4. Classification
- What comes next (maybe in different sequence)
 1. Frequent patterns
 2. Time series
 3. Novelty detection
 4. Change detection

Additional sources



<https://mitpress.mit.edu/books/machine-learning-data-streams>



<https://www.amazon.com/Knowledge-Discovery-Streams-Chapman-Mining/dp/1439826110>

SKETCHES

Stream mining axioms (re-iterated)

- **One pass** – each item in the data stream is observed only once
- **Low processing time** – per item processing times must be short
- **Low memory use** – per item memory use must be low
- **Anytime query execution** – the stream mining solution must be capable to answer queries at any moment and with limited latency
- **Non-stationary data streams** – stream mining solutions must be designed with input data change in mind

Sketches defined

- DEF: Set I consists of all records in a data stream
 - Data streams are unbounded data structures
 - It is (often) not feasible to store all elements of I in limited memory, neither random access, nor storage.
- DEF: An **item/record** is an element of set I
 - An item is stored in X units of memory
- DEF: A **sketch** is a (data structure, algorithm) pair which read a stream and store sufficient information to be able to answer one or more predefined queries about the data stream

Sketch creation and use

- **Initialize** → initialize the data structure and constants necessary for the sketch
- **Update** → called when a new item is observed in the data stream
 - Optionally modifies the underlying data structure of the sketch
- **Query** → return the current value of the pre-defined query
 - The return value is calculated based on the items read
 - A single sketch can implement multiple queries → multiple useful pieces of information can be extracted from the data structure maintained in the Update phase

Accuracy and confidence (re-visited)

- For accuracy value ε we define
 - Absolute or additive ε -approximation as
$$\forall x: |f(x) - g(x)| < \varepsilon$$
 - Relative or multiplicative ε -approximation as
$$\forall x: |f(x) - g(x)| < \varepsilon|f(x)|$$
- Confidence is defined as $1 - \delta$
- An (ε, δ) -approximation requires that an ε -approximation holds with confidence $1 - \delta$
 - For $(0.1, 0.01)$ the approximation error should be less than 0.1 for 99% of calls

Sampling

RESERVOIR SAMPLING

```
1  Init( $k$ ):  
2      create a reservoir (array[0... $k - 1$ ]) of up to  $k$  items  
3      fill the reservoir with the first  $k$  items in the stream  
4       $t \leftarrow k$     ▷  $t$  counts the number of items seen so far  
5  Update( $x$ ):  
6      select a random number  $r$  between 0 and  $t - 1$   
7      if  $r < k$   
8          replace the  $r$ th item in the reservoir with item  $x$   
9       $t \leftarrow t + 1$   
10 Query():  
11     return the current reservoir
```

- **Name:** Reservoir Sampling
- **Init:** create data structure for k items
- **Update:** replace the r -th element in the reservoir with x (i.e. the new item)
- **Query:** return the contents of the reservoir reservoir

Counting total items – 1

MORRIS'S COUNTER

```
1 Init():  
2      $c \leftarrow 0$   
3 Update(item):  
4     increment  $c$  with probability  $2^{-c}$   
5     ▷ and do nothing with probability  $1 - 2^{-c}$   
6 Query():  
7     return  $2^c - 1$ 
```

- **Name:** Morris's counter
- **Init:** set count c to 0
- **Update:** increment c with a probability
- **Query:** return the estimation

Counting distinct items – 1

- Counting distinct elements → maintain a count of distinct (i.e. different) items observed in the data stream so far

LINEAR COUNTING

```
1 Init( $D, \rho$ ):  
2     ▷  $D$  is an upper bound on the number of distinct elements  
3     ▷  $\rho > 0$  is a load factor  
4      $s \leftarrow D/\rho$   
5     choose a hash function  $h$  from items to  $\{0, \dots, s - 1\}$   
6     build a bit vector  $B$  of size  $s$   
7 Update( $x$ ):  
8      $B[h(x)] \leftarrow 1$   
9 Query():  
10    let  $w$  be the fraction of 0s in  $B$   
11    return  $s \cdot \ln(1/w)$ 
```

- **Name:** Linear Counting
- **Init:** choose hash function, init vector B
- **Update:** set $B[h(x)] = 1$
 - Calculate hash of x
 - Use hash as index
 - Set value at index to 1
- **Query:** calculate & return current count

Counting distinct items – 2

PROBABILISTIC COUNTER

```
1 Init( $D$ ):  
2      $\triangleright D$  is an upper bound on the number of distinct elements  
3      $L \leftarrow \log D$   
4     choose a hash function  $h$  from items to  $\{0, \dots, L - 1\}$   
5      $p \leftarrow L$   
6 Update( $x$ ):  
7     let  $i$  be such that  $h(x)$  in binary starts with  $0^i 1$   
8      $p \leftarrow \min(p, i)$   
9 Query():  
10    return  $2^p / 0.77351$ 
```

Flajolet-Martin

- **Name:** Flajolet-Martin
- **Init:** set up the necessary constants
- **Update:**
 - Divide stream in $m = 2^b$ substreams
 - Use first $b = \log(m)$ bits of $h(x)$ to choose substream
- **Query:** calculate & return value

Frequency problems

- Often it is necessary to separately count different types of items
- **DEF:** The **absolute frequency** of item x over (time) period t is the number of times x appeared (within period t)
- **DEF:** The **relative frequency** of item x over period t is the number of times it appeared divided by t , i.e. $\text{count}(x)/t$
- The **heavy hitter** problem: Given a threshold ϵ the set of heavy hitters consists of items whose relative frequency exceeds ϵ
- Naïve solution for the heavy hitter problem: use sampling and calculate relative frequency for the sample

Frequency problems

SPACE SAVING

```
1  Init( $k$ ):  
2      create an empty set  $S$  of pairs (item,count) able to hold up to  $k$  pairs  
3  Update( $x$ ):  
4      if item  $x$  is in  $S$   
5          increase its count by 1  
6      else if  $S$  contains less than  $k$  entries  
7          add  $(x, 1)$  to  $S$   
8      else  
9          let  $(y, count)$  be an entry in  $S$  with lowest count  
10         replace the entry  $(y, count)$  with  $(x, count + 1)$  in  $S$   
11  Query():  
12      return the list of pairs  $(x, count(x))$  currently in  $S$ 
```

- **Init:** Creates empty data structure for a set of k (item, count) pairs
- **Update** (when item x is received):
 - Add $(x, 1)$ if below capacity
 - Increment $(x, count)$ by one if x already ‘in’
 - Evict element $(y, count)$ with lowest count and add $(x, count+1)$
- **Query:** return list $(x, count(x))$

Exponential histogram for sliding windows

- **DEF:** Sliding windows in the stream mining context allow the algorithm to consider only the last W items where W is the window size
 - Windows are usually created by item count W (i.e. the number of elements), but it is feasible to consider items received in time period T , where T can be a second, hour, day or other time period
- Naïve window implementation: keep a circular buffer for the last W items
 - When a new item is received, evict the oldest element → circular buffer
 - Uses memory complexity W
- The exponential histogram approach optimizes memory use to $O(\log W)$ memory

Exponential histogram

| | | | | | | | | | | | | |
|------------|---|----------|--|----------|--|---------|--|----|--|----|--|------|
| Bucket: | <table border="1"><tr><td>1011100</td></tr></table> | 1011100 | <table border="1"><tr><td>10100101</td></tr></table> | 10100101 | <table border="1"><tr><td>100010</td></tr></table> | 100010 | <table border="1"><tr><td>11</td></tr></table> | 11 | <table border="1"><tr><td>10</td></tr></table> | 10 | <table border="1"><tr><td>1000</td></tr></table> | 1000 |
| 1011100 | | | | | | | | | | | | |
| 10100101 | | | | | | | | | | | | |
| 100010 | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | |
| 1000 | | | | | | | | | | | | |
| Capacity: | 4 | 4 | 2 | 2 | 1 | 1 | | | | | | |
| Timestamp: | $t - 24$ | $t - 14$ | $t - 9$ | $t - 6$ | $t - 5$ | $t - 3$ | | | | | | |

EXPONENTIAL HISTOGRAMS

```
1  Init( $k, W$ ):  
2       $t \leftarrow 0$   
3      create a list of empty buckets  
4  Update( $b$ ):  
5       $t \leftarrow t + 1$   
6      if  $b = 1$      $\triangleright$  do nothing with 0s  
7          let  $t$  be the current time  
8          create a bucket of capacity 1 and timestamp  $t$   
9           $i \leftarrow 0$   
10         while there are more than  $k$  buckets of capacity  $2^i$   
11             merge the two oldest buckets of capacity  $2^i$  into a  
12                 bucket of capacity  $2^{i+1}$ : the timestamp of the new bucket  
13                 is the largest (most recent) of their two timestamps  
14              $i \leftarrow i + 1$   
15         remove all buckets whose timestamp is  $\leq t - W$   
16  Query():  
17      return the sum of the capacities of all existing buckets  
18          minus half the capacity of the oldest bucket
```

Mergeability

- **DEF:** A sketch is **mergeable** if sketches built from data streams D_1, D_2, \dots, D_n can be combined into a sketch of the same type which can correctly answer queries about the interleavings of the input streams
- Item frequency sketches are trivial to merge → the answer should be the same for all interleavings
- Item counting sketches are also trivial to merge → the integer counts can be added
- Other possible operators for merging sketches:
 - OR for linear counters
 - MIN for Cohen and Flajolet-Martin

FREQUENT PATTERNS

Introduction

- DEF: Frequent pattern mining is the identification of the most frequent patterns in a collection of data

| Transactions | Itemsets |
|--------------|----------------------------|
| t_1 | milk, bread, butter |
| t_2 | milk, beer, diper |
| t_3 | diper, shampoo coke |
| t_4 | beer, diper, milk, bread |
| t_5 | beer, diper, coke |
| t_6 | milk, diper, shampoo, beer |

Definitions

- A collection of **items** is denoted by $I = \{i_1, i_2, \dots, i_m\}$
 - E.g. items you buy from supermarket, pages you visit over internet in one session, etc.
- Any subset S of items I is called an **itemset**
$$S \subseteq I$$
- A set of **transactions** is denoted by $T = \{t_1, t_2, \dots, t_n\}$ and it is called the transaction database
- The **support** of an itemset is defined as the number of transactions containing it and denoted by σ
- NOTE: Bifet re-defines graphs and trees in the frequent pattern mining chapter → we will refrain from that

Frequent itemset mining defined

- **DEF:** Given a set of items $I = \{i_1, i_2, \dots, i_m\}$ and a vector of transactions $T = \{t_1, t_2, \dots, t_n\}$ and minimum support σ_{min} the **frequent itemset mining problem** aims to find the set of frequent itemsets

$$FI = \{S \subseteq I | \sigma(S) \geq \sigma_{min}\}$$

- **Anti-monotone** property: If $X, Y \subseteq S$ are two itemsets such that $X \subseteq Y \Rightarrow \sigma(Y) \leq \sigma(X)$
 - If an itemset is infrequent, then all its supersets will also be infrequent.
- Question: How many itemsets are possible?

Patterns

Patterns and Support

- **DEF:** Patterns are entities whose presence or absence, or frequency, indicate ways in which data deviates from randomness
- **Sub-pattern** and **super-pattern** correspond to the notions of subset and superset
 - A sub-pattern is a subset of items identified as a pattern
 - A super-pattern is a superset of items identified as a pattern
- **DEF:** A transaction t **supports** a pattern p if p is a sub-pattern of the pattern in transaction t

Closed and Maximal

- **DEF:** A pattern is **closed** for a dataset D if it has higher support in D than every one of its superpatterns
- **DEF:** A pattern is **maximal** if it is frequent and none of its superpatterns are frequent
 - Note: Every maximal pattern is closed.

Itemset types

| ID | Transaction | | | | | |
|--------------|-------------|---------|-------------------|----------------------------|--------|------|
| | | Support | Frequent | Gen | Closed | Max |
| t1 | abce | | | | | |
| t2 | cde | | | | | |
| t3 | abce | | | | | |
| t4 | acde | | | | | |
| t5 | abcde | 6 | t1,t2,t3,t4,t5,t6 | c | c | c |
| t6 | bcd | 5 | t1,t2,t3,t4,t5 | e,ce | e | ce |
| Transactions | | 4 | t1,t3,t4,t5 | a,ac,ae,ace | a | ace |
| | | 4 | t1,t2,t3,t5 | b,bc | b | bc |
| | | 4 | t2,t4,t5,t6 | d,cd | d | cd |
| | | 3 | t1,t3,t5 | ab,abc,abe, be,bce,abce | ab,be | abce |
| | | 3 | t2,t4,t5 | de,cde | de | cde |

Itemset characteristics for the above set of transactions

Batch algorithms

- **Naïve pattern mining:** one-pass the dataset, keep track of every (pattern, support) and output those with support > threshold
 - Challenge: the number of possible patterns grows too fast with dataset size
- **FP-growth algorithm:** two-passes, FP-Tree data structure, no candidate generation phase
 - Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.
- **Eclat algorithm:** depth-first search, collect transactions supporting each itemset & intersect them
 - Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In Proceedings of the Third ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 97), Newport Beach, California, USA, August 14–17, 1997, pages 283–286, 1997.

Challenges and some solutions

- Frequent itemset mining in data streams is challenging due to several reasons:
 - Low memory
 - Single pass
 - Excessively large search space
 - Change → previously infrequent item become frequent and vice versa
- Three stream mining approaches for frequent patterns:
 - Approaches which don't distinguish old and new itemsets
 - Approaches which give more weights to recent items
 - Itemset mining over multiple time granularity

Approaches and criteria for streams

- **All or closed/maximal:** Mine all frequent patterns or only the closed or maximal ones.
- **Recent or all:** Consider the pattern frequency from the beginning of the stream, or give more importance to recent items, either via sliding windows or some importance decay scheme.
- **Fast update:** Update the set of frequent items after each item arrives, or perform batch updates: that is, collect a batch of stream items, then update the data structure that represents frequent patterns using the batch. The choice of batch size is key to performance.
- **Exact or approximate:** return exactly the set of patterns that satisfy the frequency condition, or just some approximation of the set. Approximate algorithms may have false positives (infrequent patterns reported as frequent), false negatives (frequent patterns not reported), or both.

Coresets of Closed Patterns

- **DEF:** A coresset of a set C is a small subset such that solving the problem on the coresset provides an approximate solution for the problem on C
 - $\text{supp}(p)$ – absolute support for pattern p
 - $\sigma(p)$ – relative support of pattern p
- p is δ -closed in D if every super-pattern of p has support less than $(1 - \delta) \text{supp}_D(p)$
 - Note: Closed patterns are the 0-closed patterns and maximal patterns are the 1-closed patterns.
- (σ, δ) -coreset of D is dataset $D' \subseteq D$ such that, for every pattern p ,
 - every pattern p occurring in D' is σ -frequent and δ -closed in D , and
 - if $p \in D'$ then p occurs as many times in D' as in D .
- (σ, δ) -coresets are lossy, compressed summaries
 - Minimum support σ excludes infrequent patterns, and
 - δ -closure excludes patterns whose support is very similar to that of some subpattern.

Heavy hitters

- In frequent pattern mining **heavy hitters** are those frequent itemsets whose counts exceed a (pre-defined) threshold
 - Re-visit: Heavy hitters in sketches calculate items with relative frequency above a threshold ϵ
- Naïve heavy hitter solution for frequent itemsets: group stream items into batches, identify frequent itemsets and memorize them in an adequately chosen data structure
- Other solutions which can be adapted for the heavy hitter itemset problem: Lossy Counting, SpaceSaving

Moment

- Moment is an exact closed frequent itemset miner operating on a sliding window
- Uses the Closed Enumeration Tree data structure to store all itemsets needed at the moment
- Distinguishes node types:
 - **Infrequent** gateway node: Contains an itemset x that is not frequent, but whose parent node contains an itemset y that is frequent and has a frequent sibling z such that $x = y \cup z$.
 - **Unpromising** gateway node: Contains an itemset x that is frequent and such that there is a closed frequent itemset y that is a superset of x , has the same support, and is lexicographically before x .
 - **Intermediate** node: A node that is not an unpromising gateway and contains a frequent itemset that is non-closed but has a child node that contains an itemset with the same support.
 - **Closed** node: Contains a closed frequent itemset.

Yun Chi, Haixun Wang, Philip S. Yu, and Richard R. Muntz. Catch the moment: Maintaining closed frequent itemsets over a data stream sliding window. *Knowl. Inf. Syst.*, 10(3):265–294, 2006.

Moment: Closed Enumeration Tree

- Moment adds and removes transactions from the tree while maintaining these five properties:
 1. All supersets of itemsets in infrequent gateway nodes are infrequent.
 2. All itemsets in unpromising gateway nodes and all their descendants are non-closed.
 3. All itemsets in intermediate nodes are non-closed but have some closed descendant.
 4. When adding a transaction, closed itemsets remain closed.
 5. When removing a transaction, infrequent items remain infrequent and non-closed itemsets remain non-closed.

Yun Chi, Haixun Wang, Philip S. Yu, and Richard R. Muntz. Catch the moment: Maintaining closed frequent itemsets over a data stream sliding window. *Knowl. Inf. Syst.*, 10(3):265–294, 2006.

FP-Stream

- Tilted-time window types:
 - Natural tilted-time window: Maintains the most recent 4 quarters of an hour, the last 24 hours, and the last 31 days. It needs only 59 counters to keep this information.
 - Logarithmic tilted-time window: Maintains slots with the last quarter of an hour, the next 2 quarters, 4 quarters, 8 quarters, 16 quarters, and so on. As in Exponential Histograms, the memory required is logarithmic in the length of time being monitored.
- A pattern is **subfrequent** if its support is more than σ' but less than σ , for a value $\sigma' < \sigma$.
- FP-STREAM maintains a global FP-Tree structure and processes transactions in batches. Every time a new batch is collected, it computes a new FP-Tree and adds it to the global FP-Tree.
 - FP-STREAM computes frequent and subfrequent patterns.

C. Giannella, J. Han, J. Pei, X. Yan, and P. Yu. Mining frequent patterns in data streams at multiple time granularities. In *Proceedings of the NSF Workshop on Next Generation Data Mining*, pages 191–212, 2002.

IncMine

- The IncMine algorithm was designed to identify frequent closed itemsets (FCIs) in windows
 - Besides FCIs, it stores candidate itemsets as well → maintains set C of semi-FCIs
 - Stream elements are collected in batches B of size b
 - Batch B is analyzed and identified FCIs are merged into set C
 - $w = \text{number of batches stored in the window} \rightarrow \text{represents } wb \text{ transactions}$
 - Number of itemsets is minimized by early dropping pattern p if it does not occur at least σ_{ib} times in the first i batches and with minimum support σ
- Pro: considerably faster than Moment and FP-STREAM
- Contra: approximate solution, allows false negatives

James Cheng, Yiping Ke, and Wilfred Ng. Maintaining frequent closed itemsets over a sliding window. *J. Intell. Inf. Syst.*, 31(3):191–215, 2008.

SEQUENTIAL PATTERN MINING

Introduction

- **DEF:** A **sequence** is an ordered list of items. Formally, a sequence of items $A = \{a_1 a_2, a_3, \dots, a_n\}$ is such that items in the set are ordered according to some rule. Rule could be timestamp based on arrival, or values of the items.
- Transactions vs sequences (they are not the same!)

| ID | Transaction |
|----|-------------|
| t1 | abce |
| t2 | cde |
| t3 | abce |
| t4 | acde |
| t5 | abcde |
| t6 | bcd |

| ID | Sequence(s) |
|----|-------------------|
| s1 | <a(abc)(ac)d(cf)> |
| s2 | <(ad)c(bc)(ae)> |
| s3 | <(ef)(ab)(df)> |
| s4 | <eg(af)> |
| s5 | <af> |
| s6 | <(eg)> |

Sequence mining defined

- DEF: **Sequential pattern mining** aims to find the complete set of frequent subsequences given a set of sequences and support threshold

| ID | Sequence(s) |
|----|-------------------|
| s1 | <a(abc)(ac)d(cf)> |
| s2 | <(ad)c(bc)(ae)> |
| s3 | <(ef)(ab)(df)> |
| s4 | <eg(af)> |
| s5 | <af> |
| s6 | <(eg)> |

A sequence → <a(abc)(ac)d(cf)>



An element may contain a set of items. Items within an element are unordered and we list them alphabetically.

Sub-sequence → <a(abc)(ac)>

- Given **support** threshold min_sup =2, <(ab)c> is a sequential pattern

Challenges

- A huge number of possible sequential patterns are hidden in datasets and especially in unbounded data streams
- A sequential pattern mining algorithms should therefore
 - find the complete set of patterns, when possible, satisfying the minimum support (frequency) threshold
 - be highly efficient, scalable, involving only a small number of database scans
 - be able to incorporate various kinds of user-specific constraints

Applications

- Sequences of commands frequently used by users of a software product
- Shopping of items: first buy computer, then pen-drive, then keyboard over a period of time.
- Some event sequence: earthquake → tsunami → medical treatment → (optionally) fission core melts
- DNA/protein sequences etc.

Summary

- Sketches
- Frequent patterns and itemsets
- Sequential pattern mining



Thank you for your attention!



CHANGE DETECTION

Stream mining (SM)

Imre Lendák, PhD, Associate Professor

Chandresh Maurya, PhD, Assistant Professor

SM course project clarifications

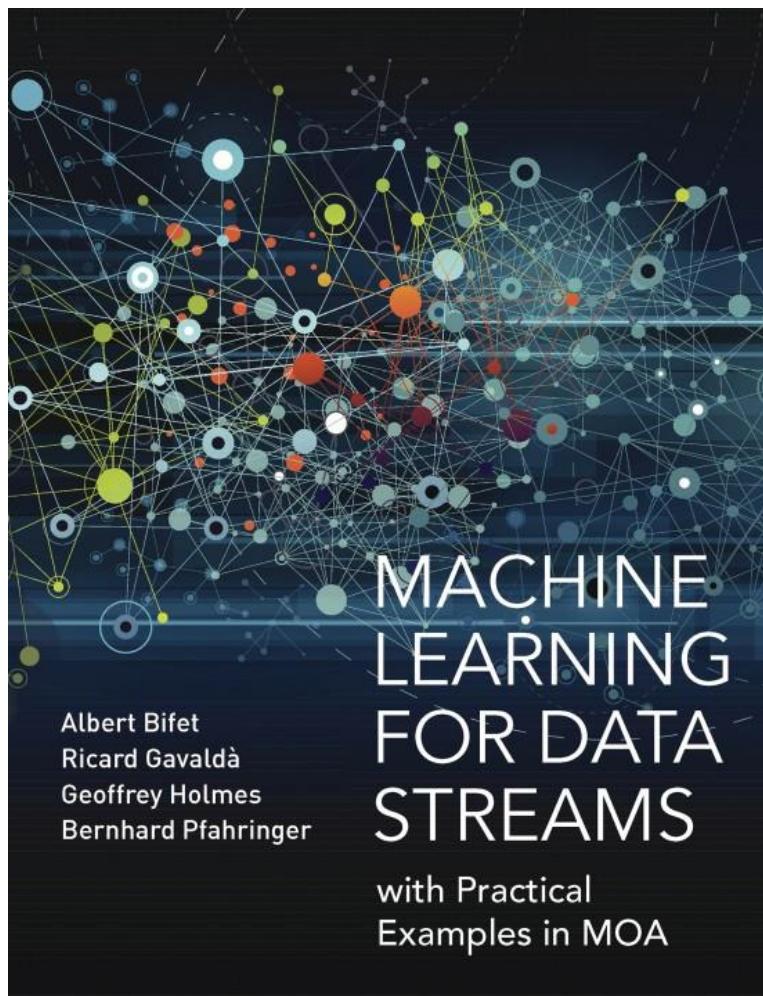
- Each project team member is required to **actively participate** in the streaming system element of the project
 - The separation of duties within the stream mining portion of project should also be completely clear
- If only **simple operations** (e.g. counting and filtering) are used, then the SM project grade will be lower or the lowest
 - The lectures held by Peter and Imre, i.e. everything after Part I: Streaming Systems contains ideas for possible stream processing stages, e.g. sketches, frequent pattern mining, anomaly detection
- **Note:** the use of online systems in the **public cloud** and the upload of course project data into those systems is **strictly forbidden**
 - Any indication of uploading the data in the public cloud or making it available via public & direct links will result in banning the student from the oral exam in school year 2020/2021

Overview & lecture topics

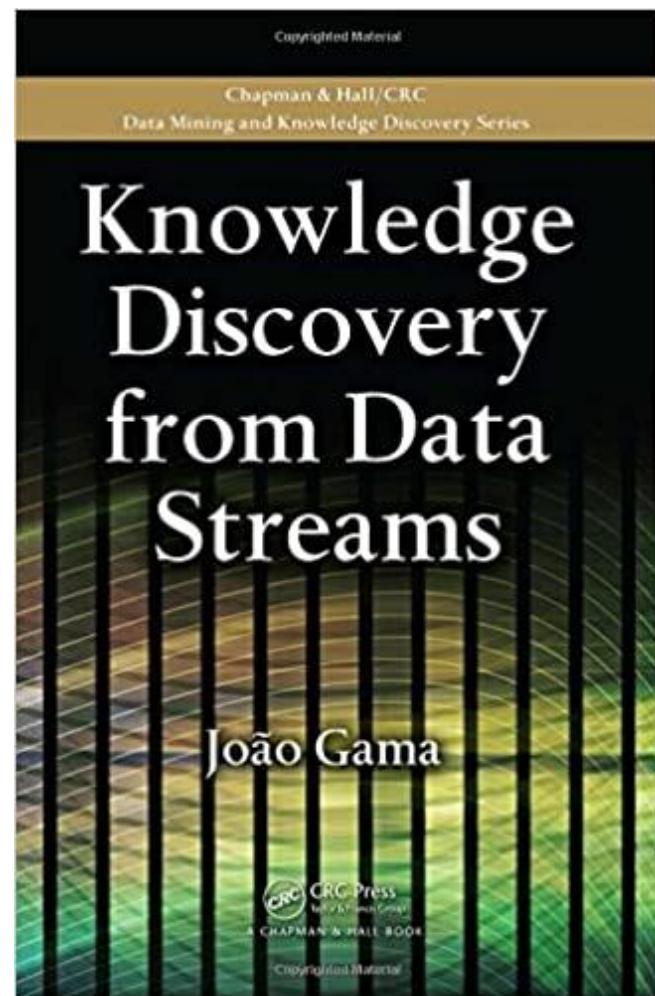


- Background and terminology
- Responding to change
- Change detection model
 - Detect
 - Adapt
 - Manage models
 - Evaluate
- Use cases
- References

Key sources (but many others as well)



<https://mitpress.mit.edu/books/machine-learning-data-streams>

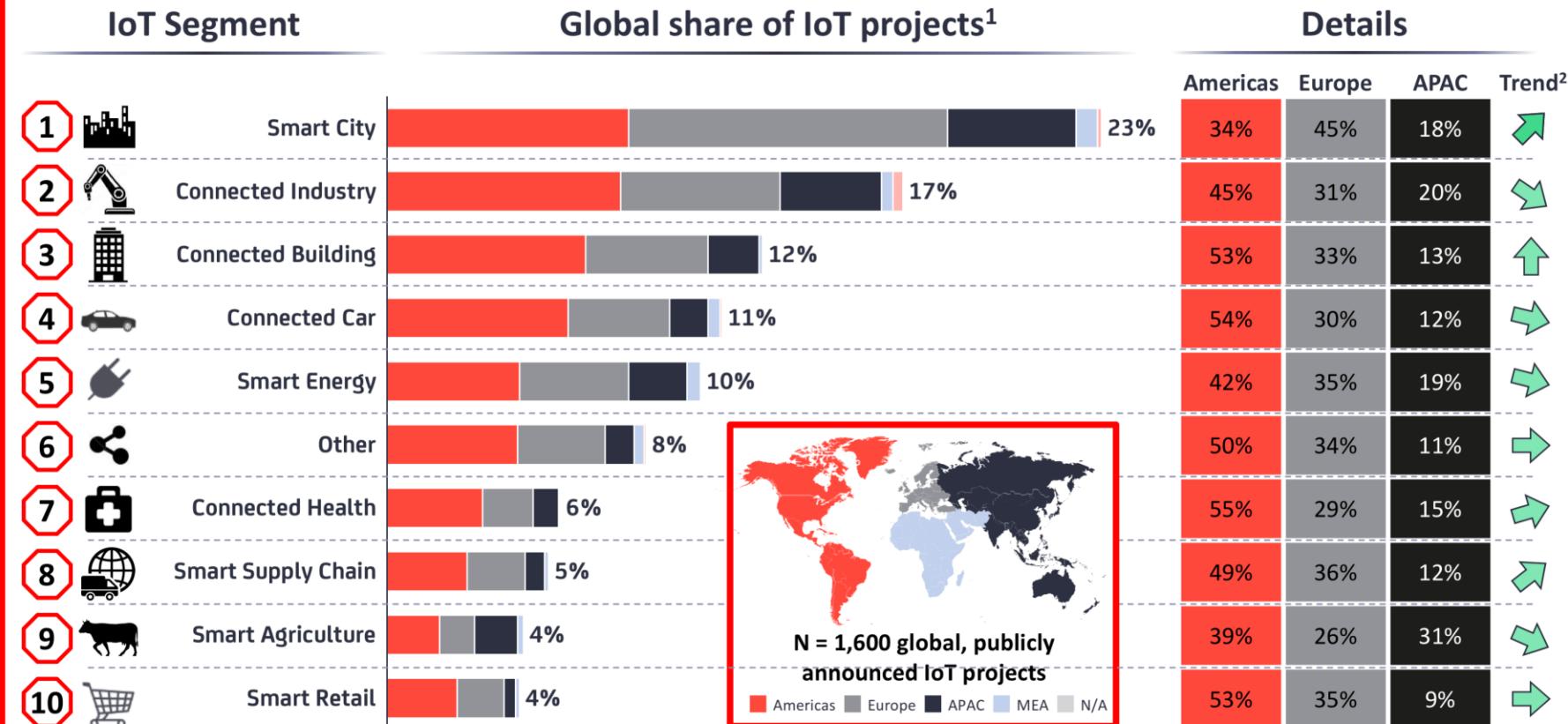


<https://www.amazon.com/Knowledge-Discovery-Streams-Chapman-Mining/dp/1439826110>

Background

- Historically, most machine learning works assumed that data records used for training are **generated at random** and according to **stationary probability distribution**
- As soon as the 1990s, scientists identified real-life problems in which change detection was highly relevant
 - **User behavior modeling**, e.g. happy employees turn into disgruntled employees
 - **Industrial process monitoring**, e.g. the quality of chemicals used improves over time
 - **Fault detection**, e.g. gradual equipment status deterioration due to material fatigue or other reasons
 - ...

Motivation



1. Based on 1,600 publicly known enterprise IoT projects (Not including consumer IoT projects e.g., Wearables, Smart Home). 2. Trend based on comparison with % of projects in the 2016 IoT Analytics Enterprise IoT Projects List. A downward arrow means the relative share of all projects has declined, not the overall number of projects. 3. Not including Consumer Smart Home Solutions. Source: IoT Analytics 2018 Global overview of 1,600 enterprise IoT use cases (Jan 2018)

Source: IoT Analytics, Jan 2018

<https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/>

A world in movement

- The new characteristics of data in the IoT (and other) settings:
 - **Time and space:** the objects of analysis exist in time and space. Often they are able to move.
 - **Dynamic environment:** the objects exist in a dynamic and evolving environment.
 - **Information processing capability:** the objects have limited information processing capabilities.
 - **Locality:** the objects know only their local spatio-temporal environment.
 - **Distributed Environment:** objects will be able to exchange information with other objects.
- Main goal: **real-time analysis** with decision models which **evolve** in correspondence with the evolving environment.

Gama J. Data Sciences: where are we going? Plenary speech @ IDEAL 2020

Challenges of real-time data mining

- Switch from **one-shot learning** to continuously learning dynamic models that evolve over time.
- **Finite training sets, static models, and stationary distributions** will have to be completely thought anew.
- **Computational resources are finite** → algorithms will have to use limited computational resources (in terms of computations, memory, space and time, communications).
- Additional examples:
 - Internet: traffic logs, user queries, email, financial.
 - Telecommunications: phone calls, SMS,
 - Astronomical surveys: optical, radio.
 - Sensor networks: many more observation points.

From static to real-time

Stream mining yesterday

- In **real-time stream mining** we face continuous data flows generated at high-speed in dynamic, time-changing environments
- The **usual approaches** for querying, clustering and prediction use batch procedures and cannot cope with this streaming setting.
- Machine Learning **algorithms assume:**
 - Instances are independent and generated at random according to some probability distribution D.
 - It is required that D is stationary
- **Practice:** finite training sets, static models

Stream mining today

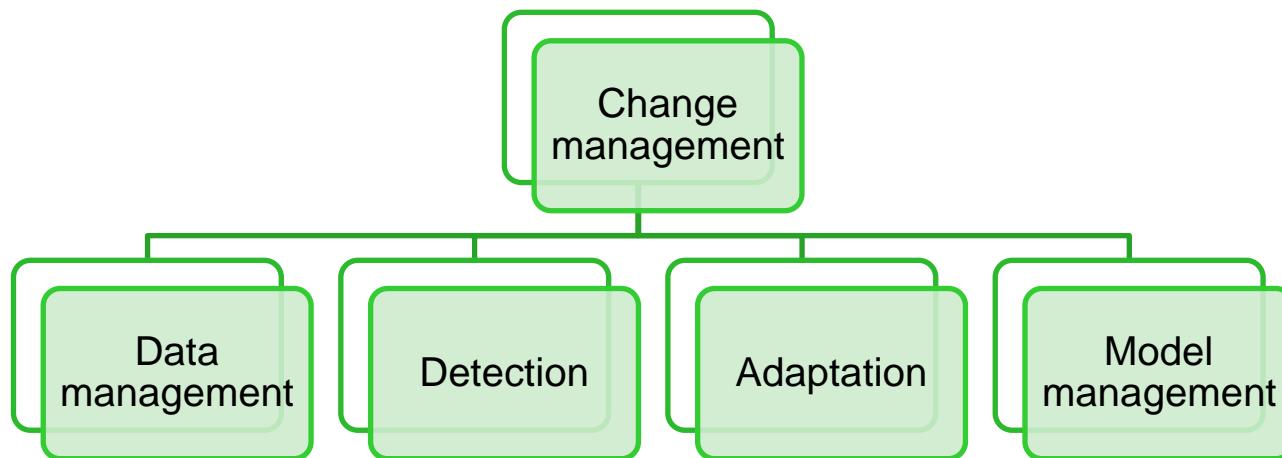
- We need to maintain decision models in real time.
- Learning algorithms must be capable of:
 - **Incorporate new information** at the speed data arrives
 - **Detect changes** and adapt decision models to the most recent information
 - **Forget outdated information**
- **Latest practice:** Unbounded training sets, dynamic models.

Terminology

- **DEF:** The **goal of change detection** is to monitor, detect and respond to changes in the systems which are modeled and in which decisions are made by computer-based models
 - **Change detection in streaming systems** is additionally complicated by the unbounded nature of the inputs and limited storage and processing power and the necessity to produce outputs in a timely (often real-time) manner
 - Change detection solutions must be able to differentiate **noise vs change**
 - **Persistence** → there is a consistent set of records following the changed distribution
- **DEF:** **Drift** is gradual change

RESPONDING TO CHANGE

Traditional change management

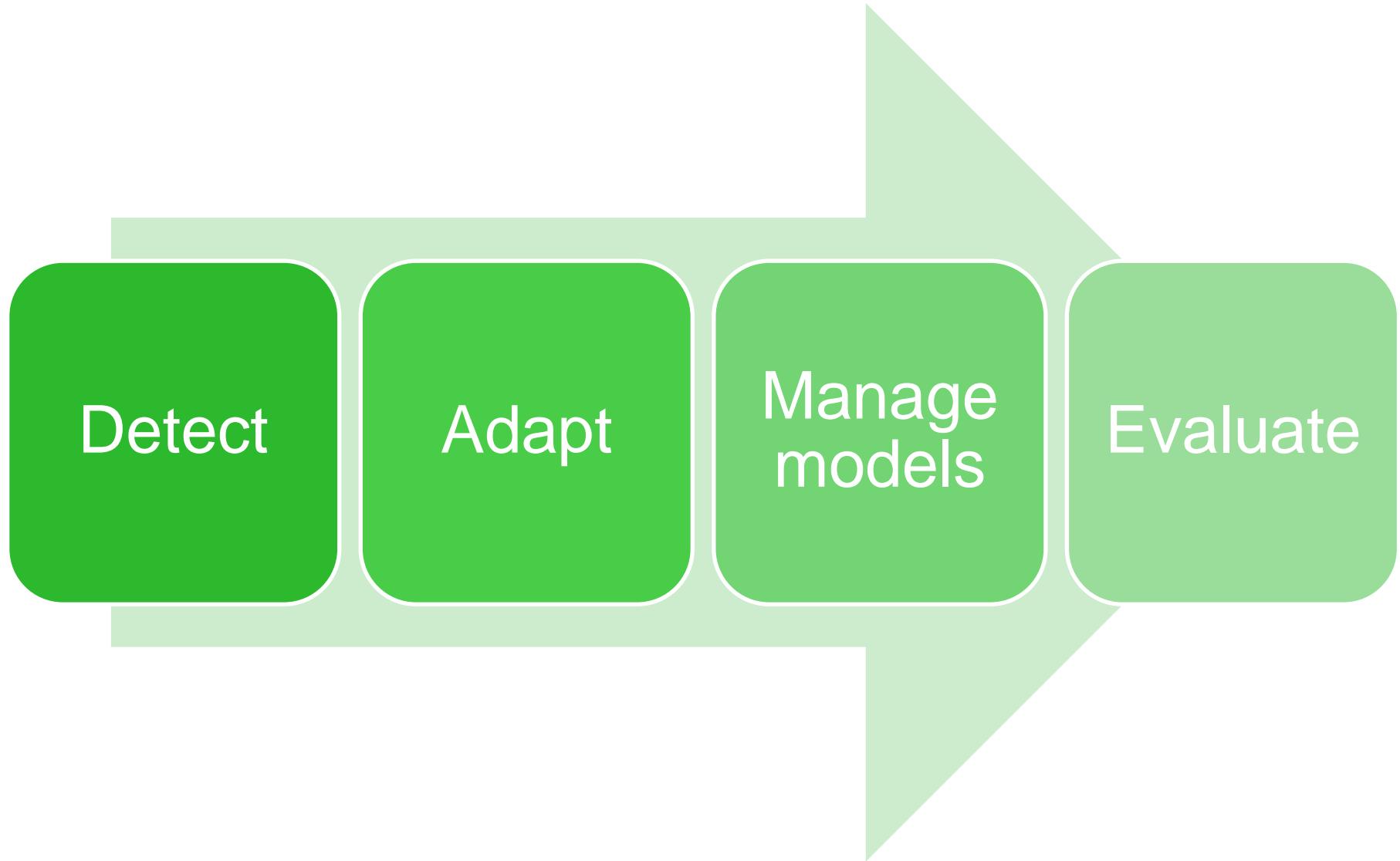


- Machine learning solutions try to find function f that maps input x to output y
$$y = f(x)$$
- Such a function is assumed to be stationary, i.e. distribution generating the data is fixed (but unknown).
- Real-life datasets (and especially data streams) are non-stationary and evolving

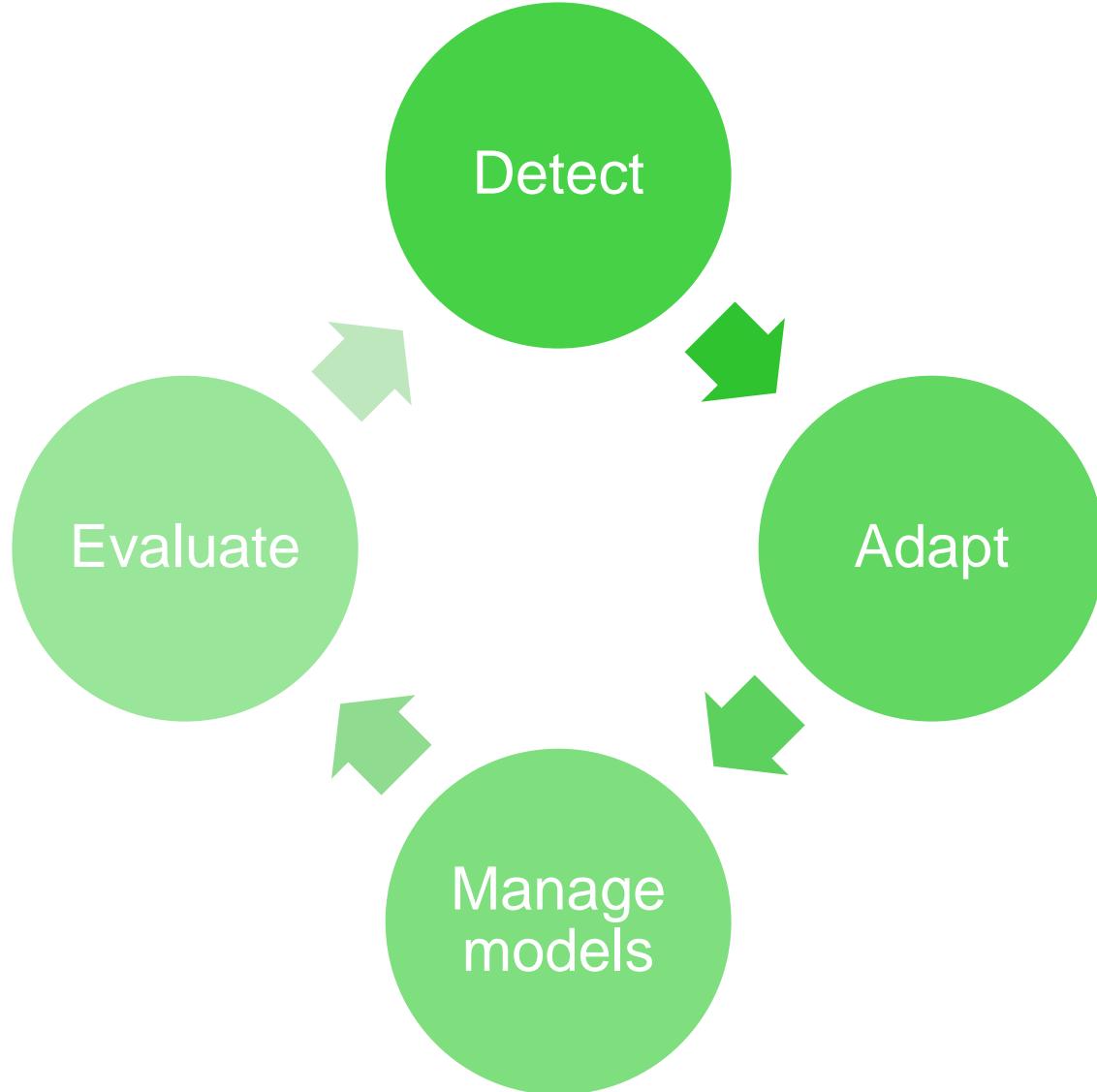
Change management concepts

- **DEF:** **Data management** techniques select and store the optimum type and amount of data and data aggregates for efficient change management
- **DEF:** **Change detection** algorithms monitor input data streams and machine learning solution performance with the goal to detect sub-optimal system operation
- **DEF:** **Adaptation** techniques enable the data analysis system to react (or preempt) changes
- **DEF:** **Model management** techniques allow the data analysis system to handle multiple models, train or re-train, retire
- **DEF:** Change detection **evaluation** techniques allow the data analysis system to measure its change management process → additional improvements, detect bugs

Change management process v1



Change management process v2

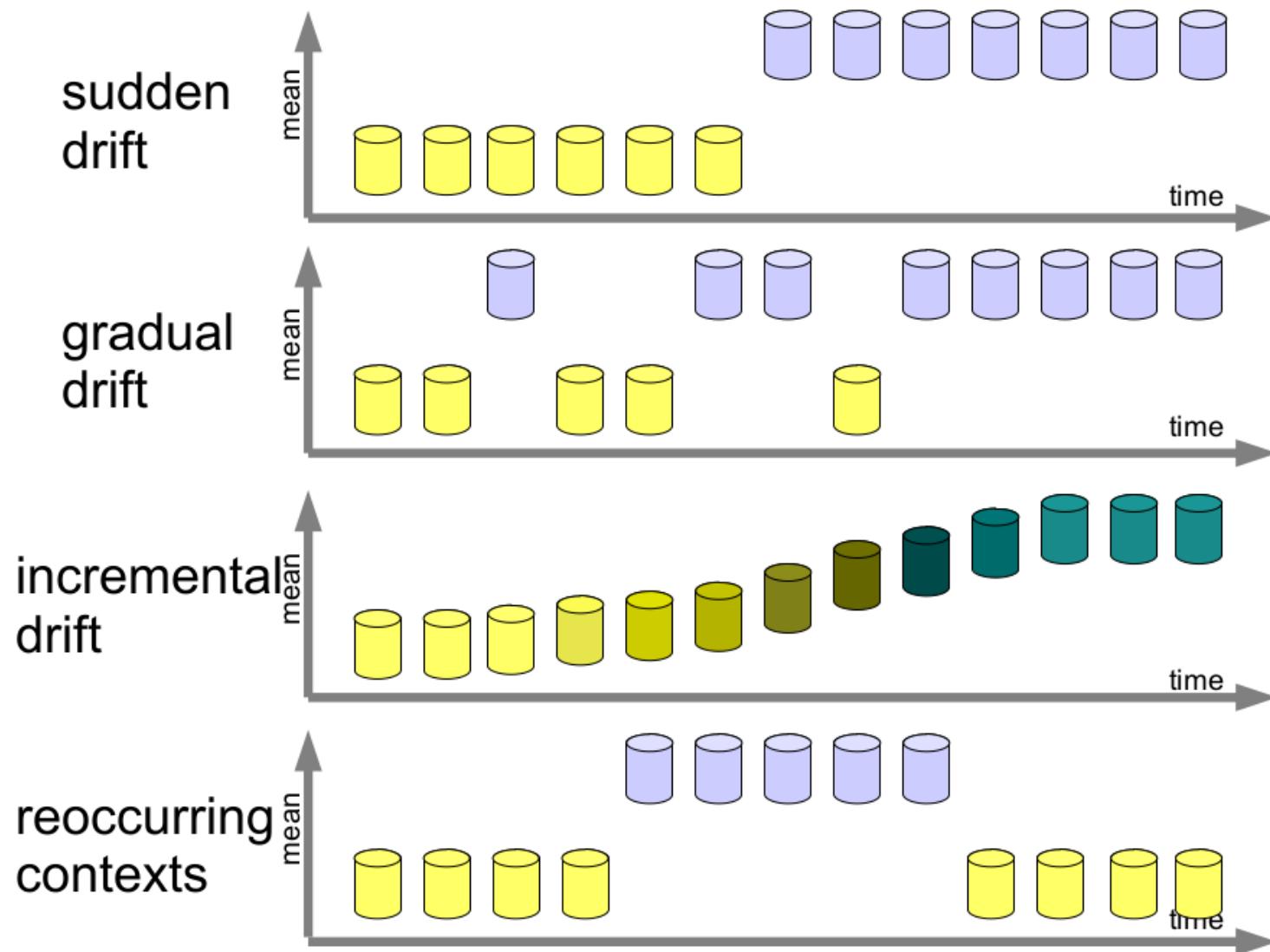


PHASE I: DETECT

Causes of change

- Changes due to modifications in the **context of learning**
 - The world changes → our assumption and system models need to be updated
 - New, improved techniques appear → change model
- Changes in **hidden variables**
 - ML algorithms learn from observations described by a finite set of attributes.
 - In real world problems, there can be important properties of the domain that are not observed → hidden variables
 - Hidden variables may change over time (!) → such changes can invalidate otherwise well-trained models
- Changes in the **characteristic properties** in the observed variables
 - The observed variables change in time → re-train necessary

Rate of change



Rate of change in words

- **Concept shift** usually refers to abrupt changes → usually ‘easier’ to detect
 - Re-occurring contexts are essentially periodically or otherwise repeated sudden drifts
- **Concept or gradual drift** is usually associated with gradual change in the observed concept(s)
 - Usually more challenging to detect compared to sudden drift
 - The initial phase of gradual change can be mistaken for noise in the incoming data
 - Resilience to noise can be obtained via observing more records
 - Incremental drift is gradual change with a ‘smooth’ transition
- **Note:** if the rate of change is larger than the ability to learn, then it is not possible to set up a proper machine learning solution

Types of change

Change Detection

37

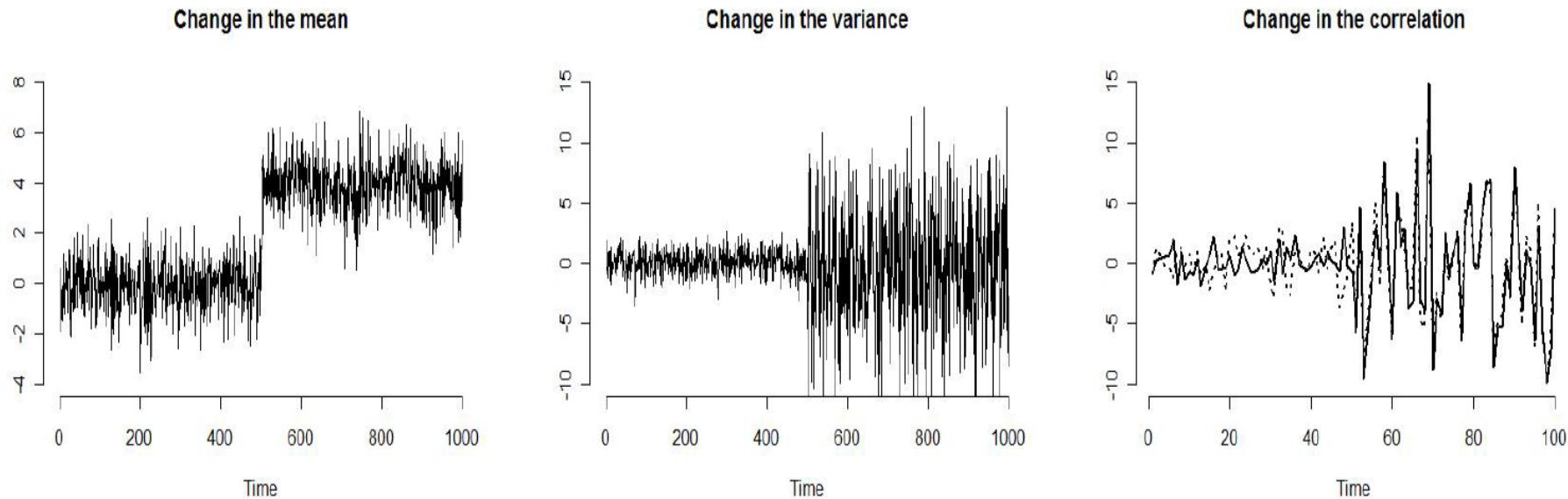


Figure 3.1: Three illustrative examples of change: (a) Change on the mean; (b) Change on variance; (c) Change on correlation

Gama J. (2010). Knowledge discovery in data streams. CRC Press.

Data management approaches

- Change detection techniques rely on data
- The data management approaches used by such techniques can be categorized into
 - **Full memory** → store statistics over all examples → their implementation in streaming systems can be challenging
 - Usually rely on some form of sketches (stream aggregates)
 - Example: weighted average over all observations
 - **Partial memory** → store only the most recent observations
 - **Fixed window size** → store in memory a fixed amount of the most recent observations
 - **Adaptive window size** → usually works with ‘longer’ windows in normal operation and with ‘shorter’ windows when change is detected
 - Example: 20 SMA during normal operation and 10 SMA when change is suspected to come
- Weighting observations in the data management solution allow gradual forgetting of ‘older’ observations

Monitoring in detection

- Monitoring the evolution of **performance indicators** of the decision model
 - Usually monitoring accuracy, recall and precision over time and raise the red flag (i.e. signal change) when the values of these indicators are outside some predefined (or varying?) bound(s)
 - Most change detection approaches fall into this category
- Monitoring data distributions on **two different time-windows**
 - Usually summarize and compare past information and the most recent observations
 - One approach is to examine observations drawn from two (or more?) probability distributions and decide whether they are different

Cumulative sum (CUSUM)

- CUSUM (CUmulative SUM) algorithm is a change detection algorithm which monitors the cumulative sum to detect a change.
- **DEF:** Let S_t be the current cumsum and m_t the current min value of S_t , the cumsum compares this difference with a threshold.

$$z_t = (x_t - \mu) / \sigma$$

$$S_0 = 0$$

$$S_t = \max(0, S_{t-1} + z_t - k)$$

Declare change if

$$S_t > h$$

Reset CUSUM by:

$$S_0 = 0, \text{reset } \mu, \sigma$$

- **Challenge:** how to choose values k & h ?
 - Guideline for k : set it to half the value of the change to be detected (measured in standard deviations)
 - Guideline for h : set $h = \ln(\frac{1}{\delta})$ where δ is the acceptable false alarm rate

Page-Hinkley (PH) test

- The Page-Hinkley test is essentially a variant of CUSUM

$$z_t = (x_t - \mu) / \sigma$$

$$s_0 = 0$$

$$s_t = s_{t-1} + z_t - k$$

$$S_t = \min\{s_t, S_{t-1}\}$$

Declare change if

$$g_t - G_t > h$$

Reset PH to

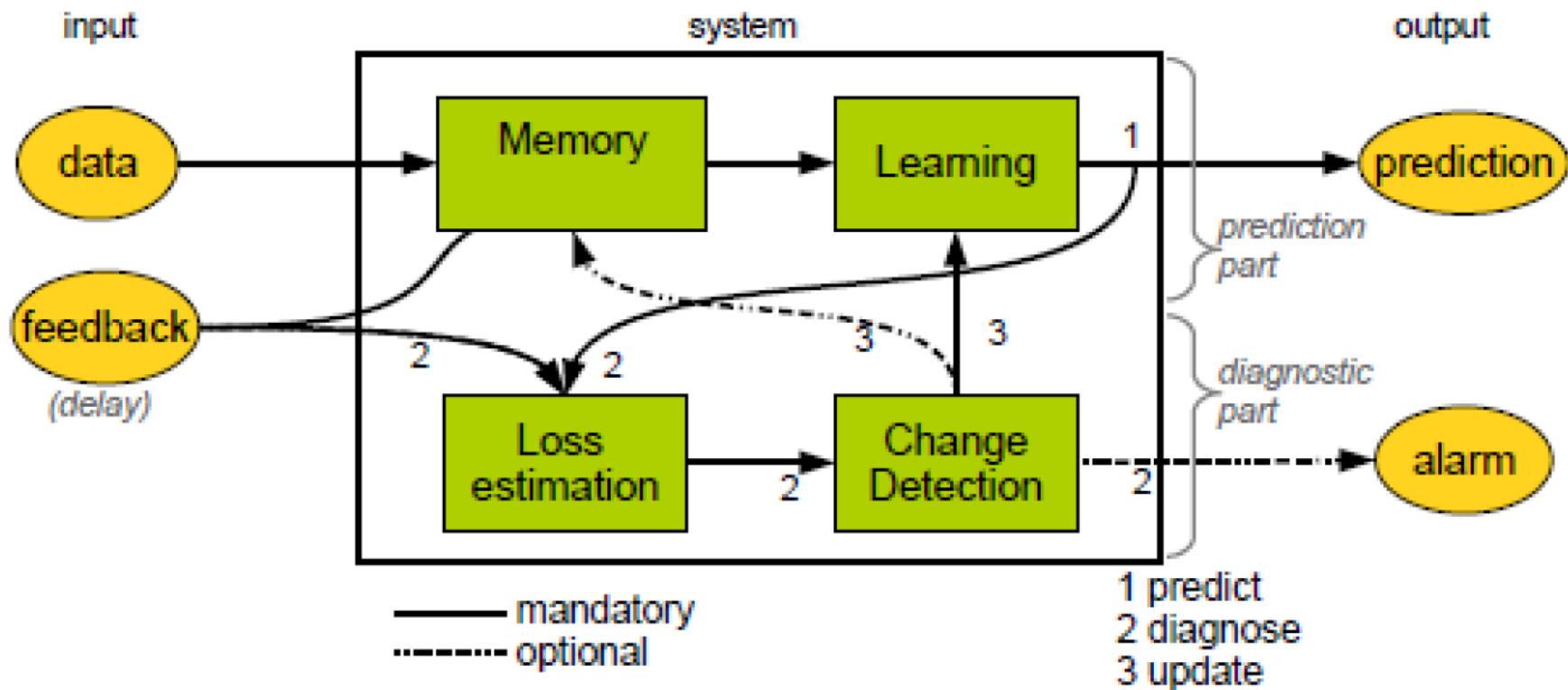
$$S_0 = 0, \text{reset } S_t, \mu, \sigma$$

PHASE II: ADAPT

Adaptation methods

- **Blind methods** adapt (i.e. modify) decision models at regular intervals without considering whether changes have really occurred.
 - Examples include methods that weight the examples according to their age and methods that use time-windows of fixed size
 - **Pro:** no detection is necessary
 - **Contra:** re-train might not be necessary → wasted resources
- **Informed methods** only modify the decision model after a change was detected.
 - They are used in conjunction with a detection model
 - **Pro:** optimized for resource use → not run when not needed
 - **Contra:** might miss changes due to noise or delays

Adaptive learning solution



A generic schema for an online adaptive learning algorithm.

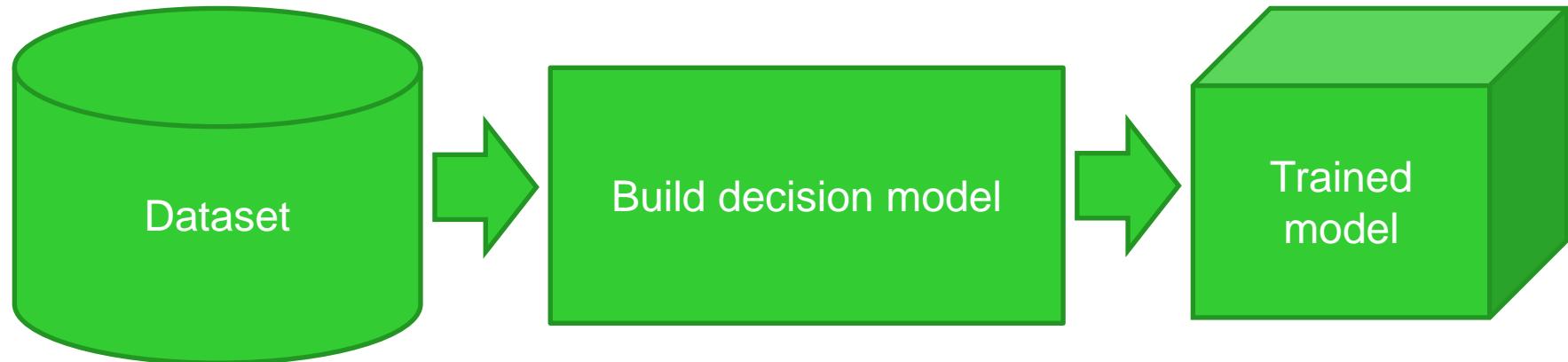
(*A survey on concept drift adaptation*, J.Gama et al, ACM-CSUR 2014)

PHASE III: MANAGE MODELS

Manage models

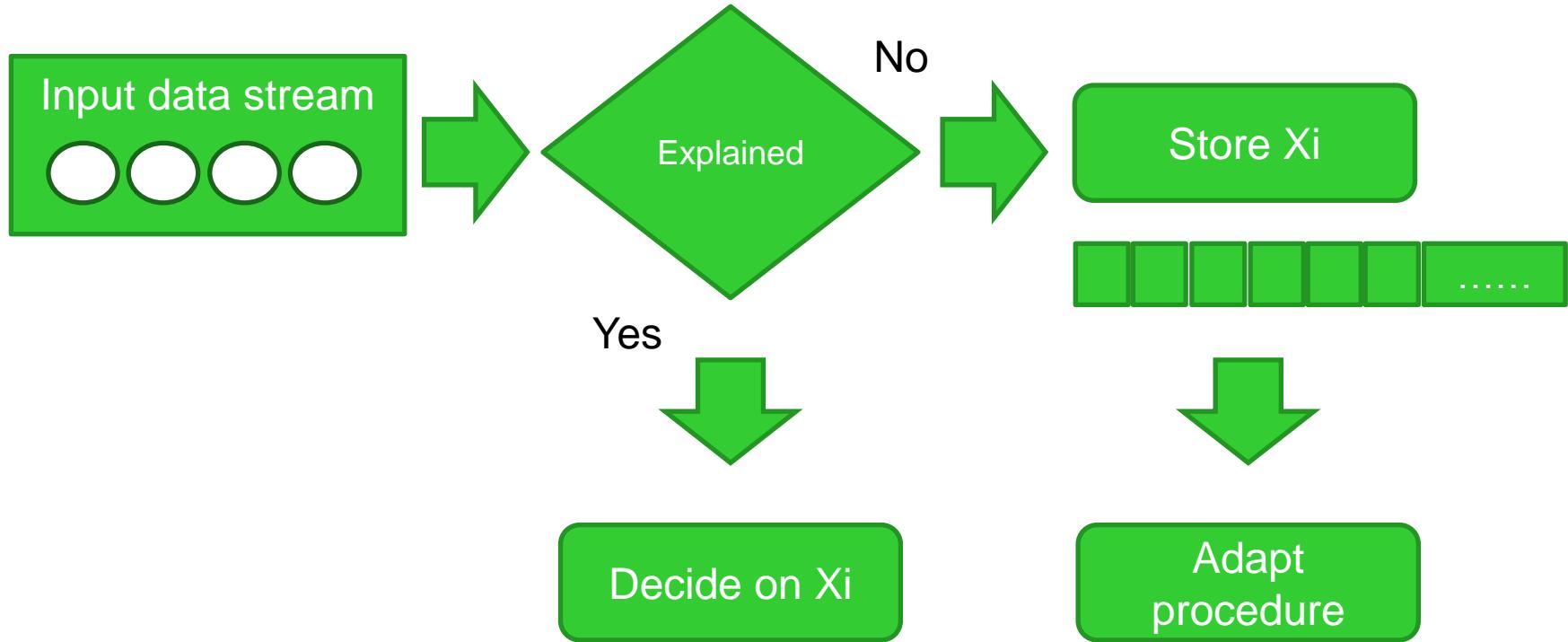
- Instead of maintaining a single decision model often it is optimal to use multiple decision models
 - This is different from ensemble models used to obtain optimal decisions in different ML use cases (!)
- **DEF:** The task of the **model management phase** is to store (in memory or in storage), archive and retire models
 - The model building is triggered in the detect phase
 - The models are built in the adapt phase
- Model management phases:
 - Phase I (offline): A model is built offline
 - Phase II (online): The detect and adapt phases yield new models as well as models which should be retired

Offline training phase



- Build a decision model for known normal patterns
- The MM phase is tasked to store the trained model – at the right in the above figure

MM in the online adaptation phase



- **Step #1:** Detect observations or sets of observations for which a consistent decision cannot be made → store them in short-term memory
- **Step #2:** Short-term memory full → initiate adaptation procedure
- **Step #3:** Manage models, e.g. add newly trained and retire old model

PHASE IV: EVALUATE

Evaluate change detection

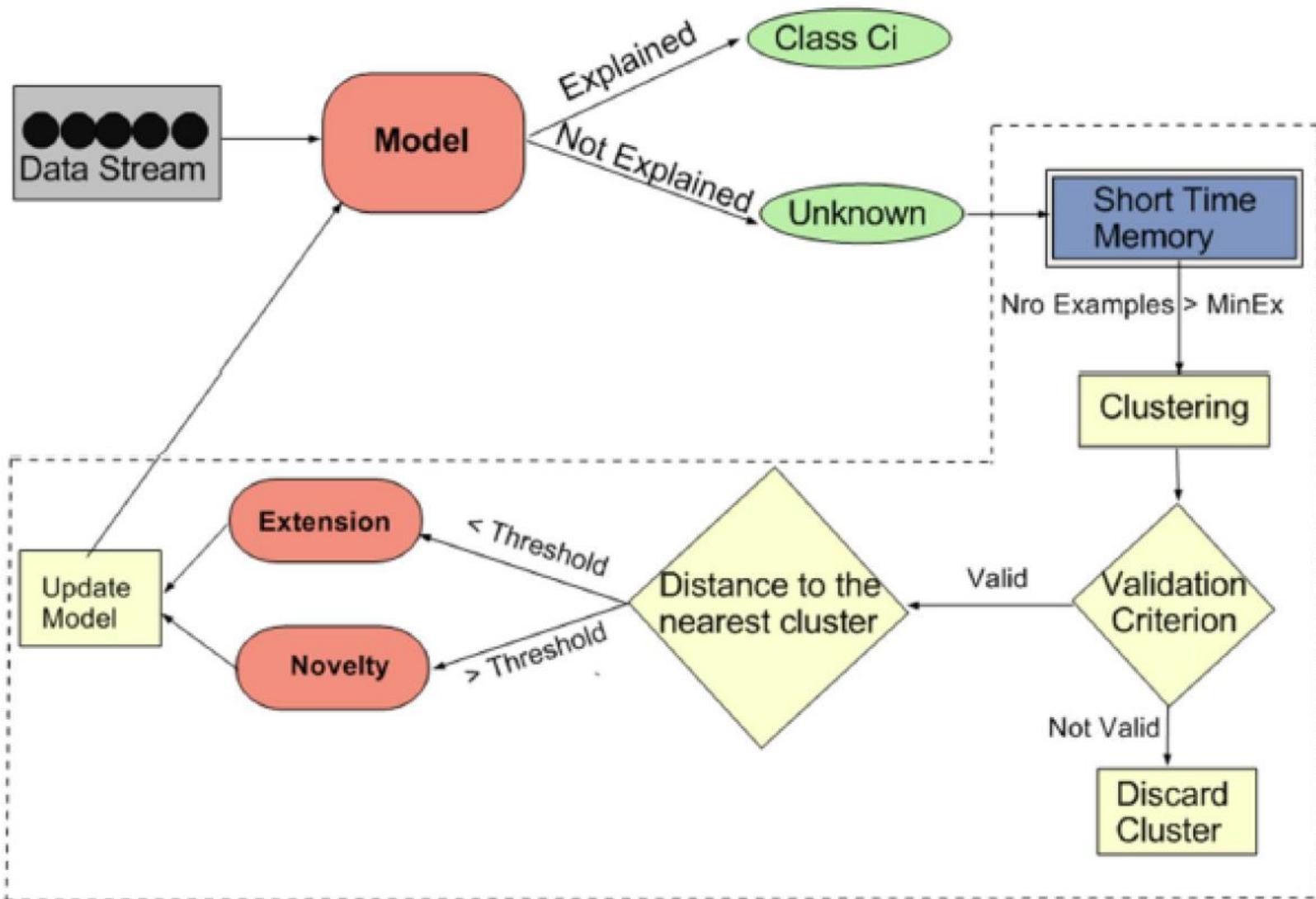
- Drift detection methods are evaluated on the following metrics:
 - **Error rate** (Number of mistakes made so far) → possible if there is (delayed) feedback about changes, i.e. they are labeled
 - **Probability of true detection** or TPR → capacity to detect and react to change
 - **Probability of false alarm** or FPR → does not signal when there is no change in the observed concept
 - **Delay in detection** → usually measured as the number of examples required to detect a change after the occurrence of a change
 - Other possible performance indicators: precision, recall, AUC etc.

CHANGE DETECTION USE CASES

Change detection use cases

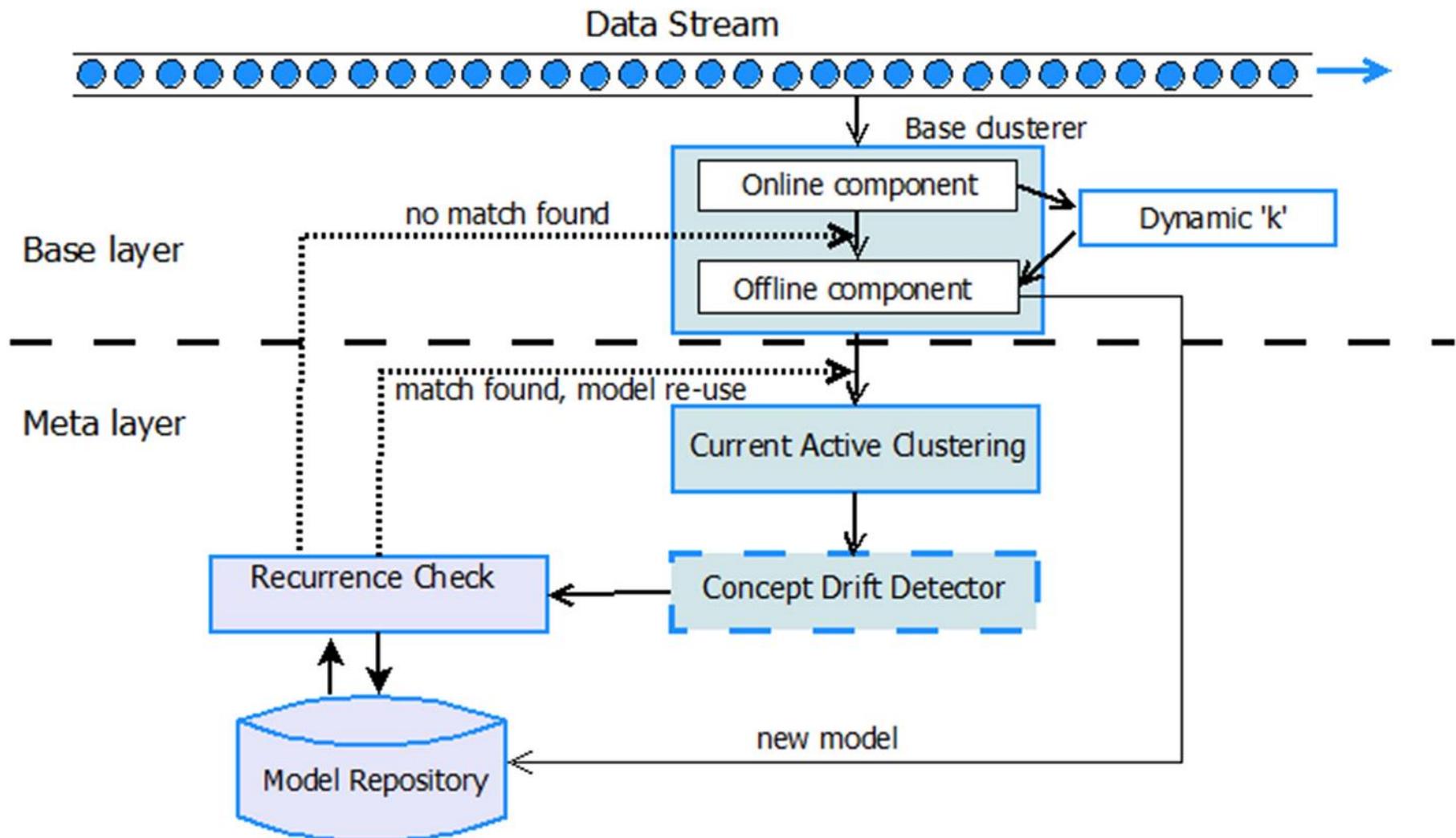
- **Social media** → sentiment analysis, e.g. via detecting changes in Twitter feeds
→ the analysis of data streams consisting of short texts
- **Information security**
 - Changing behavior of employees → more complex insider threat monitoring
 - Changing behavior of adversaries → frequent adaptation to the latest threats, e.g. based on security data feeds, intra-industry collaboration
- **Financial systems**
 - Macro-level market changes → re-visit models used for forecasting market movement(s)
 - Typical residential and commercial customer behavior shifts → use different solutions to predict churn, upsell, analyze (credit) risk
- **Health infrastructure**
 - Changing habits of the population → changing health of average male and female patients
- **Electric power systems**
 - Residential, commercial and industrial blocks are re-purposed, customers install energy saving tech → changing power consumption patterns

Classification in the presence of change



Gama J. Data Sciences: where are we going? IDEAL 2020.

Clustering in the presence of change



Namitha K., Santhosh Kumar G. (2020). Learning in the presence of concept recurrence in data stream clustering, vol 7 (75).

Prediction in the presence of change

- The Drift Detection Method (DDM) is applicable in the context of predictive models
 - The method monitors the number of errors produced by a model learned on the previous stream items
 - When DDM observes that the prediction error increases, it takes this as evidence that change has occurred
$$s_t = \sqrt{p_t(1 - p_t)/t},$$
p_t is the error rate
- DDM stores the smallest error rate p_{min} observed up until moment t and performs the following checks
 - Issue warning if $p_t + s_t \geq p_{min} + 2 \cdot s_{min}$
 - Declare change if $p_t + s_t \geq p_{min} + 3 \cdot s_{min}$
- **DDM drawback:** p_t is computed based on all observations since the last change → DDM might be slow to respond

Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004, September). Learning with drift detection. In Brazilian symposium on artificial intelligence (pp. 286-295).

Anomaly detection in the presence of change

- Open discussion with students
- Example questions for discussion:
 - Which datasets are we analyzing?
 - What do we want to find out, i.e. what are the anomalies (outliers) we are interested in?
 - How to detect change?
 - Classification vs clustering? Or other techniques?
 - Are motifs and discords useful in the changing anomaly detection scenario in unbounded data streams?
 - Is windowing necessary? Event time vs processing time windowing?

REFERENCES

References

- Barddal J.P., Gomes H.M., Enembreck F. (2015). Advances on Concept Drift Detection in Regression Tasks Using Social Networks Theory, International Journal of Natural Computing Research, vol 5(1).
- Bifet, A., Gavaldà, R., Holmes, G., & Pfahringer, B. (2018). Machine learning for data streams: with practical examples in MOA. MIT Press.
- Bifet, A., Holmes, G., Pfahringer, B., & Gavaldà, R. (2011, August). Mining frequent closed graphs on evolving data streams. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 591-599).
- Demšar, J., & Bosnić, Z. (2018). Detecting concept drift in data streams using model explanation. Expert Systems with Applications, 92, 546-559.
- Gama J. Data Sciences: where are we going? (plenary talk) (2020). 21st International Conference on Intelligent Data Engineering and Automated Learning - IDEAL'2020 4-6 November Guimaraes, Portugal
- Gama J. (2010). Knowledge discovery in data streams. CRC Press.
- Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004, September). Learning with drift detection. In Brazilian symposium on artificial intelligence (pp. 286-295). Springer, Berlin, Heidelberg.
- Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. ACM computing surveys (CSUR), 46(4), 1-37.
- Janardan, S.M. (2017). Concept drift in Streaming Data Classification: Algorithms, Platforms and Issues. Procedia Computer Science, vol 122, pp. 804-811.
- Namitha K., Santhosh Kumar G. (2020). Learning in the presence of concept recurrence in data stream clustering, vol 7 (75).
- Page E.S. (1954). Continuous inspection schemes. Biometrika, 41(1/2):100–115.

Thank you for your attention!