Streaming Machine Learning

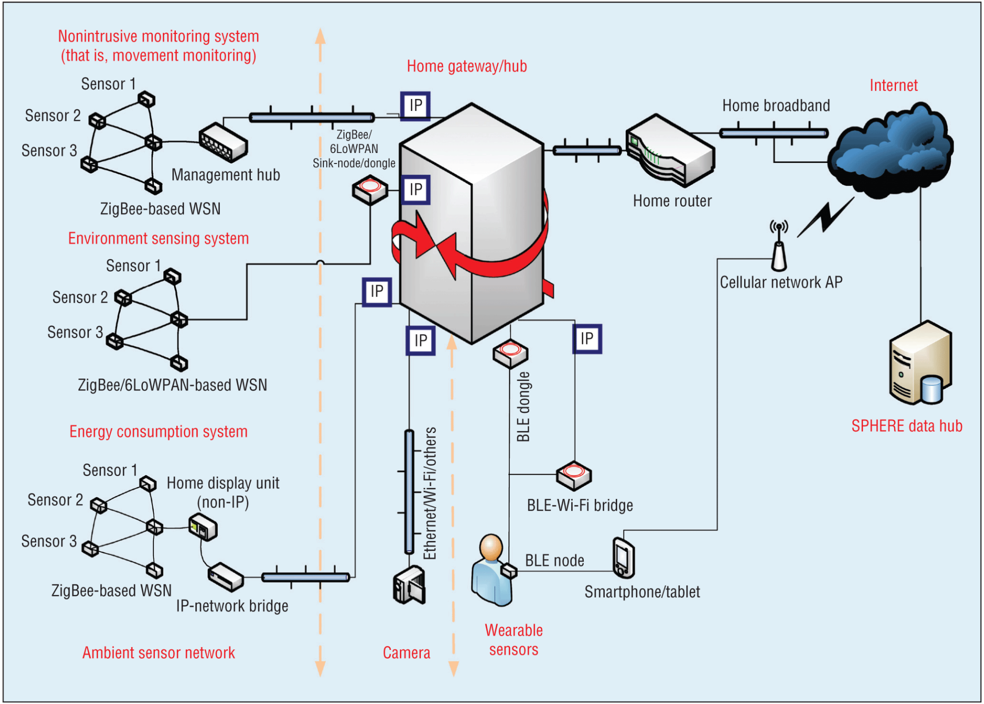
Tom Diethe
tdiethe@amazon.com

University of Bristol, December 2017

Outline

Stream Processing

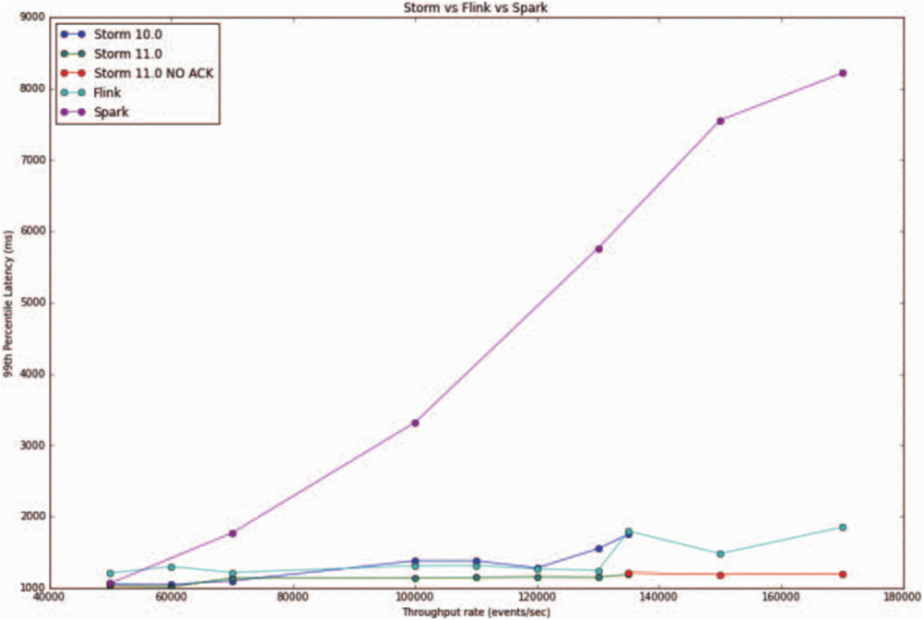
- Suitable for applications that exhibit certain characteristics:
  - Compute Intensity: the number of arithmetic operations per I/O or global memory reference. In many signal processing applications today it is well over 50:1 and increasing with algorithmic complexity.
  - Data Parallelism: the same function is applied to all records and records can be processed simultaneously without waiting for results from previous records.
  - Data Locality: a specific type of temporal locality common in signal processing applications where data is produced once, read once or twice later in the application, and never read again. Intermediate streams passed between kernels as well as intermediate data within kernel functions can capture this locality directly using the stream processing programming model.


Nonintrusive monitoring system
(that is, movement monitoring)
Sensor 1
Sensor 2
Sensor 3
Management hub
ZigBee-based WSN
ZigBee/
6LoWPAN
Sink-node/dongle
Home gateway/hub
IP
Home broadband
Internet
Home router
Environment sensing system
Sensor 1
Sensor 2
Sensor 3
ZigBee/6LoWPAN-based WSN
Cellular network AP
SPHERE data hub
Energy consumption system
Sensor 1
Sensor 2
Sensor 3
Home display unit
(non-IP)
ZigBee-based WSN
IP-network bridge
Ethernet/Wi-Fi/others
BLE dongle
BLE-Wi-Fi bridge
BLE node
Smartphone/tablet
Ambient sensor network
Camera
Wearable
sensors

What's out there

- Batch File Based Processing
  - Emulates "full" stream processing
    - ★ Apache Kafka
    - ★ Apache Flink
    - ★ Apache Storm
    - ★ Apache Apex
    - ★ Apache Spark
  - Typically requires beefy hardware - unsuitable for small deployments (e.g. IoT)
- Stream Processing Services:
  - Cost-effective processing at scale
    - ★ Amazon Web Services - Kinesis
    - ★ Google Cloud - Dataflow
    - ★ Microsoft Azure - Stream Analytics
    - ★ IBM - Streaming Analytics
  - Requires data to be pushed to the cloud continuously

Speed Comparison



(1)

Motivation

- want models that take into account recent history when it makes its predictions
  - e.g. weather: If it has been sunny and 25 degrees the last two days, it is unlikely that it will be -5 and snowing the next day

Motivation

- want models that take into account recent history when it makes its predictions
  - e.g. weather: If it has been sunny and 25 degrees the last two days, it is unlikely that it will be -5 and snowing the next day
- want a model that is updatable - evolve as data streams through their infrastructure
  - e.g. retail sales model that remains accurate as the business gets larger

Motivation

- want models that take into account recent history when it makes its predictions
  - e.g. weather: If it has been sunny and 25 degrees the last two days, it is unlikely that it will be -5 and snowing the next day
- want a model that is updatable - evolve as data streams through their infrastructure
  - e.g. retail sales model that remains accurate as the business gets larger
- Central question: is the underlying source generating the data changing? (stationary vs non-stationary)

Motivation

- want models that take into account recent history when it makes its predictions
  - e.g. weather: If it has been sunny and 25 degrees the last two days, it is unlikely that it will be -5 and snowing the next day
- want a model that is updatable - evolve as data streams through their infrastructure
  - e.g. retail sales model that remains accurate as the business gets larger
- Central question: is the underlying source generating the data changing? (stationary vs non-stationary)
  - weather forecasting: No: Given the weather from the previous few days you can usually make a pretty good guess at the weather for the next day, and your guess, given recent history will be roughly the same from year to year. The same model for last year will work for this year.

Motivation

- want models that take into account recent history when it makes its predictions
  - e.g. weather: If it has been sunny and 25 degrees the last two days, it is unlikely that it will be -5 and snowing the next day
- want a model that is updatable - evolve as data streams through their infrastructure
  - e.g. retail sales model that remains accurate as the business gets larger
- Central question: is the underlying source generating the data changing? (stationary vs non-stationary)
  - weather forecasting: No: Given the weather from the previous few days you can usually make a pretty good guess at the weather for the next day, and your guess, given recent history will be roughly the same from year to year. The same model for last year will work for this year.
  - business, Yes: the business is growing, and so your guess of the sales given the previous few days of sales is probably going to be different from last year. So last year's data, when the business was small, is really not relevant to this year, when the business is large. We need to update the model (or scrap it completely and retrain) to get something that works.

Solutions

- Incremental Algorithms:
  - incremental versions of batch algorithms
  - model is updated each time it sees a new training instance
  - incremental versions of Support Vector Machines (adatron) and Neural networks. Bayesian models lend themselves naturally to this setting (use posterior as new prior)
  - most require multiple passes through the data!
- "True" online learning:
  - algorithms developed specifically for the online setting
- Periodic Re-training with a batch algorithm:
  - more straightforward solution
  - simply buffer the relevant data and retrain our model "every so often"

Issues

- Data Horizon:
  - How quickly do you need the most recent datapoint to become part of your model?
  - Does the next point need to modify the model immediately, or is this a case where the model needs to behave conditionally based on that point?

Issues

- Data Horizon:
  - How quickly do you need the most recent datapoint to become part of your model?
  - Does the next point need to modify the model immediately, or is this a case where the model needs to behave conditionally based on that point?
- Data Obsolescence:
  - How long does it take before data should become irrelevant to the model?
  - Is the relevancy somehow complex? Are some older instances more relevant than some newer instances? Is it variable depending on the current state of the data? e.g. economics:
    - generally, newer data instances are more relevant
    - in some cases data from the same month or quarter from the previous year are more relevant than the previous month or quarter of the current year
    - In a recession, previous recessions more relevant than a different part of the economic cycle

Issues

- Data Horizon:
  - How quickly do you need the most recent datapoint to become part of your model?
  - Does the next point need to modify the model immediately, or is this a case where the model needs to behave conditionally based on that point?
- Data Obsolescence:
  - How long does it take before data should become irrelevant to the model?
  - Is the relevancy somehow complex? Are some older instances more relevant than some newer instances? Is it variable depending on the current state of the data? e.g. economics:
    - generally, newer data instances are more relevant
    - in some cases data from the same month or quarter from the previous year are more relevant than the previous month or quarter of the current year
    - In a recession, previous recessions more relevant than a different part of the economic cycle
  - Incremental learners all have an (implicit) assumption that controls the relevancy of old data: may or may not be modifiable and the relationship may be complex

Issues

- Data Horizon:
  - How quickly do you need the most recent datapoint to become part of your model?
  - Does the next point need to modify the model immediately, or is this a case where the model needs to behave conditionally based on that point?
- Data Obsolescence:
  - How long does it take before data should become irrelevant to the model?
  - Is the relevancy somehow complex? Are some older instances more relevant than some newer instances? Is it variable depending on the current state of the data? e.g. economics:
    - ★ generally, newer data instances are more relevant
    - ★ in some cases data from the same month or quarter from the previous year are more relevant than the previous month or quarter of the current year
    - ★ In a recession, previous recessions more relevant than a different part of the economic cycle
  - Incremental learners all have an (implicit) assumption that controls the relevancy of old data: may or may not be modifiable and the relationship may be complex
  - Batch retraining $\longrightarrow$ flexible: easy to select data for retraining, filter by relevant criteria, even weight the data according to some relevancy function using one of the many batch training algorithms that take weighting into account. Or use concept drift detection? (5)
- New "tasks": modify ML models so they can remember old tasks when learning a new one: see "Continual Learning" (3, 4)

HyperStream

- developed at the University of Bristol for the SPHERE project (2)
- python package for processing streaming data with workflow creation capabilities.
- interfaces to execute complex nesting, fusion, and prediction both in online and offline forms

https://github.com/IRC-SPHERE/HyperStream

Building Blocks

- Streams
- Tools
- Channels
- Stream IDs (Meta-data)
- Plates
- Nodes
- Factors
- Workflows
- Sessions
- Plugins

Streams

Stream

A Stream is a (possibly infinite) collection of documents

- A stream is defined using a StreamID object, which consists of a name and meta_data

```
{
    "name": "video",
    "meta_data": ((("house", "1"), ("location", "kitchen")),)
}
{
    "name": "wearable",
    "meta_data": ((("house", "1"), ("resident", "A")),)
}
```

- Each individual document in a stream is a StreamInstance object, consisting of:
  - A UTC timestamp
  - A value: any arbitrary python object that can be converted to BSON

```
StreamInstance(
    timestamp=datetime(2017, 7, 27, 10, 33, 45, tzinfo=<UTC>),
    value=42))
```

Tools

Tool

A tool is the element of computation, that operates on source streams and produces output streams. Tools have parameters which are fixed for the life of the tool.

- Examples:

Tool	Sources	Sinks	Parameters
Clock	{}	{ticks}	first, stride
Apply*	{objects}	{func(objects)}	func
Splitter	{dicts}	{value 1, value 2, ...}	element, mapping
CSV Reader	{}**	{rows}	filename

* not the same as map() ** not a stream

Tools

Tool

A tool is the element of computation, that operates on source streams and produces output streams. Tools have parameters which are fixed for the life of the tool.

- Examples:

Tool	Sources	Sinks	Parameters
Clock	{}	{ticks}	first, stride
Apply*	{objects}	{func(objects)}	func
Splitter	{dicts}	{value 1, value 2, ...}	element, mapping
CSV Reader	{}**	{rows}	filename

* not the same as map() ** not a stream

- Bonus fact: Tools are streams too!

Channels

Channel

A channel defines where the data is stored. Some channels are read-only.
Channels define the manifestation of streams, along with any specific processing required to read and write the streams, which abstracts away the specifics of interacting with different data sources.

- Read/Write channels
  - Memory channel: Volatile channel for storing intermediate streams
  - Database channel: uses HyperStream's native mongodb
- Read only channels
  - File channel
  - Module channel. A file channel for python modules
  - Tool channel. A specific type of module channel for storing Tools
  - Assets channel. For storing static assets

Workflows

Workflow

A workflow defines a graph of nodes connected by factors, which can be surrounded by plates.

- Workflows can have multiple time ranges, which will cause the streams contained in the nodes to be computed on all of the ranges given.
- Workflows can be defined to be operable in offline-only mode, or also available to the HyperStream online engine, which will cause the workflow to be executed continuously.
- Workflows are serialised to MongoDB by HyperStream for ease of deployment.

Simple analysis

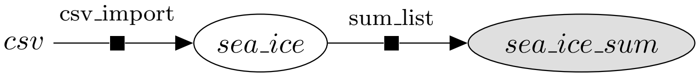


Figure: Example chain of computations. The filled (grey) node indicates that the *sea_ice_sum* stream is stored in the database rather than memory.

Sleep prediction

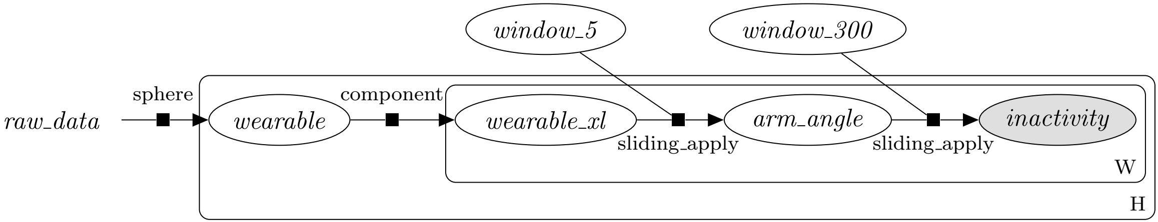


Figure depicts an example workflow for prediction of sleep, showing nested plates, nodes and factors. Here the raw data comes from the SPHERE deployment houses, which are on the **H** plate. The wearable data is then split by its unique identifier (since there is more than one wearable per house) onto the **W** plate, which is nested inside the **H** plate. Two `sliding_apply` tools are then executed for each wearable in each house with differing length sliding windows (5s and 300s) to first compute windowed arm angles and then a windowed inactivity estimate, which is stored in the database channel and subsequently used as part of a sleep prediction algorithm.

Conclusion

Machine Learning presents special challenges for streaming:

- How to update models online
- When to retrain
- What to store

Conclusion

Machine Learning presents special challenges for streaming:

- How to update models online
- When to retrain
- What to store

HyperStream was developed to tackle these:

- Workflow management
- Online and offline executions modes
- Simple deployment (few dependencies)

Conclusion

Machine Learning presents special challenges for streaming:

- How to update models online
- When to retrain
- What to store

HyperStream was developed to tackle these:

- Workflow management
- Online and offline executions modes
- Simple deployment (few dependencies)

https://irc-sphere.github.io/HyperStream/

Tutorials: http://nbviewer.jupyter.org/github/IRC-SPHERE/HyperStream/blob/master/examples/index.ipynb

Questions?

Selected References

[1] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng, et al. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, pages 1789–1792. IEEE, 2016.

[2] Tom Diethe, Meelis Kull, Niall Twomey, Kacper Sokol, Hao Song, Emma Tonkin, and Peter Flach. IRC-SPHERE/HyperStream: First public pre-release version, January 2017. URL https://doi.org/10.5281/zenodo.242227.

[3] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, page 201611835, 2017.

[4] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[5] Indre Å¡liobaite, Mykola Pechenizkiy, and Joao Gama. An overview of concept drift applications. In *Big Data Analysis: New Algorithms for a New Society*, pages 91–114. Springer, 2016.

tdiethe@amazon.com