Online Machine Learning in Streaming Applications

Stavros Kontopoulos (Principal Engineer, Lightbend)
@s_kontopoulos
&
Debasish Ghosh (Principal Engineer, Lightbend)
@debasishg

{stavros.kontopoulos, debasish.ghosh}@lightbend.com

Lightbend

Agenda

- Data Streams meet Machine Learning. How they differ from static data sets.
- Learning systems for data streams
- Adaptive learning model
- Evaluation metrics
- The ADWIN algorithm and the complete cycle of training streaming systems

Formalizing Data Streams

- Difference with batch learning and static data sets
- The data stream model
- Challenges of stream based learning

Batch Learning

- Static data set to generate an output hypothesis
- One shot data analysis - fixed stable feature space
- Assume stationarity of data
- Centralized algorithm (1 algorithm being used for the entire learning process)

Batch and Online Learning

- Static data set to generate an output hypothesis
- One shot data analysis - fixed stable feature space
- Assume stationarity of data
- Centralized algorithm (1 algorithm being used for the entire learning process)

- Learner operates on a (potentially infinite) sequence of data streams
- Evolving feature space - new learning paradigm *Feature Evolvable Streaming Learning*
- Non stationary data set - leading to concept drift
- Incremental retraining (possibly) using different algorithms

Formalizing Data Streams

- Data streams are ordered sequences of data elements: S = $(\mathbf{s_1, s_2, \ldots, s_n})$ where **n** can potentially be infinite

- The 3 main features of data streams which make them inapplicable to standard data mining algorithms in this field

    - Very large (potentially infinite number of data elements) - think sublinear space
    - High rate of data arrival at the system - may need to think of sampling
    - Changes in data distribution during stream processing - concept drift

Challenges of Stream-based Learning

- Very large (potentially infinite) number of data elements - think sublinear space
- High rate of data arrival at the system - may need to think of ***sampling***
- Changes in data distribution during stream processing, which can affect prediction - ***concept drift***

Learning Systems for Data Streams

- Dimensions of learning
- Classification problems
- Concept drift

Learning Systems for Data Streams

Desirable properties of learning systems for efficiently mining continuous, high-volume, open ended data streams (Hulten and Domingoes, 2001[1]):

- Require small constant time per data example
- Use fixed amount of main memory, irrespective of the number of examples
- Build a decision model using a single scan over the training data
- Generate an anytime model independent of the order of the examples
- Ability to deal with concept drift

[1] *Hulten, G., & Domingos, P. (2001). Catching up with the data: research issues in mining data streams. In Proc. of workshop on research issues in data mining and knowledge discovery, Santa Barbara, USA.*

Dimensions of Learning

- **Space** - the available memory is fixed
- **Learning Time** - process incoming examples at the rate they arrive
- **Generalization Power** - how effective the model is at capturing the true underlying *concept*

Classification Problems

$$P(y)$$

Beliefs before the start
of the experiment

Prior probabilities of
the class labels

Legend:
X : input examples
y : class labels

Lightbend

Classification Problems

$$P(y)$$

Beliefs before the start of the experiment

Prior probabilities of the class labels

$$p(X|y)$$

Probability function of X given the class label y

Class conditional probability density functions

Legend:
X : input examples
y : class labels

Classification Problems

$$P(y)$$

Prior probabilities of the class labels

$$p(X|y)$$

Class conditional probability density functions

$$P(y|X) = \frac{p(y)p(X|y)}{p(X)}$$

Posterior - Probability that the sample to be classified is a y, given the data set

$$p(X) = \sum_{y=1}^{c} p(y)p(X|y)$$

Legend:
X : input examples
y : class labels

Concept

(Prior)

$$P(y)$$

(Class conditional)

$$P(X|y)$$

The joint distribution gives us the stochastic definition of a Concept

$$P(X, y)$$

For unsupervised learning where we don't have the class labels, Concept is

$$P(X)$$

Concept Drift

Because data is expected to evolve over time, especially in dynamically changing environments, where ***non stationarity*** is typical, the underlying distribution can change dynamically over time.

Concept drift between time point $\mathbf{t_0}$ and time point $\mathbf{t_1}$ can be defined as:

$$\exists X : p_{t_0}(X, y) \neq p_{t_1}(X, y)$$

How does Concept Drift Affect Classification Problems

- Classification decision is made based on the posterior probabilities of the classes

$$P(y|X) = \frac{p(y)p(X|y)}{p(X)}$$

How does Concept Drift Affect Classification Problems

- Classification decision is made based on the posterior probabilities of the classes

may change *may change* *may change*

$$P(y|X) = \frac{p(y)p(X|y)}{p(X)}$$

How does Concept Drift Affect Classification Problems

Virtual Concept Drift

Real Concept Drift

changes in data distribution without knowing the class labels

affects predictive decision

$$P(y|X) = \frac{p(y)p(X|y)}{p(X)}$$

Real and Virtual Concept Drifts

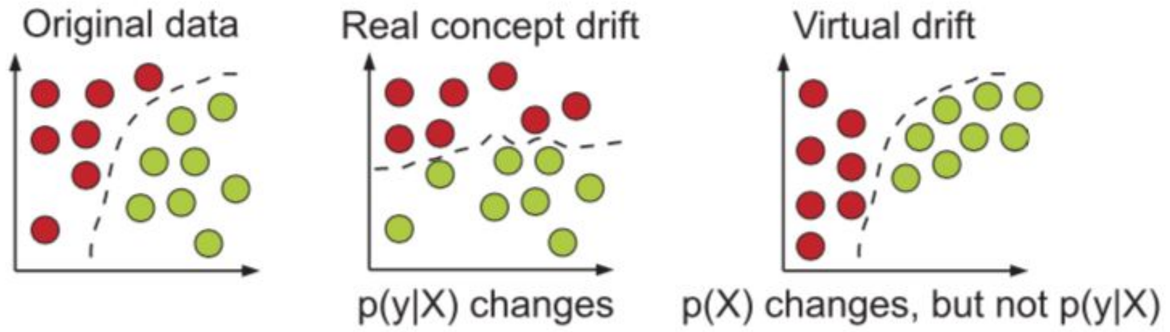


Fig. 1. Types of drifts: circles represent instances; different colors represent different classes.

(Source) A Survey on Concept Drift Adaptation - Joao Gama et al. ACM Computing Surveys, 46, 4, March 2014
DOI: http://dx.doi.org/10.1145/2523813

Classification Example

Assume the click prediction problem for a given e-commerce site, where we have a vector of features X for every user profile and a clicked product/advertisement Y. Things can change in many ways:

P(Y|X) <- suddenly users may change preference, thus affecting prediction.

P(Y) <- suddenly there is more demand for a specific product. Among the products

how likely is Y to be clicked?

P(X|Y) <- Given a product, the profiles of people who choose it may change. Audience may vary suddenly.

Adaptive Learning Model

- how to adapt to evolving data over time
- detecting concept drift and adapting to it

Adaptive Learning Model

- Detect and adapt to evolving data over time

- Adapt decision model to take care of concept drift
  - Detect drift
  - Adapt
  - Operate in less than example arrival time and
  - Use not more than a fixed amount of memory for any storage.

Adaptive Learning Model

Step 1: Predict

$$X_t \longrightarrow \boxed{L_t} \longrightarrow \hat{y}_t$$

Legend:
X : input examples
y : class labels
L : decision model for prediction
based on a learning algorithm y = L(X)

Adaptive Learning Model

Step 1: Predict

Step 2: Diagnose / Compute Loss Function

$$f\left(\hat{y}_t, y_t\right)$$

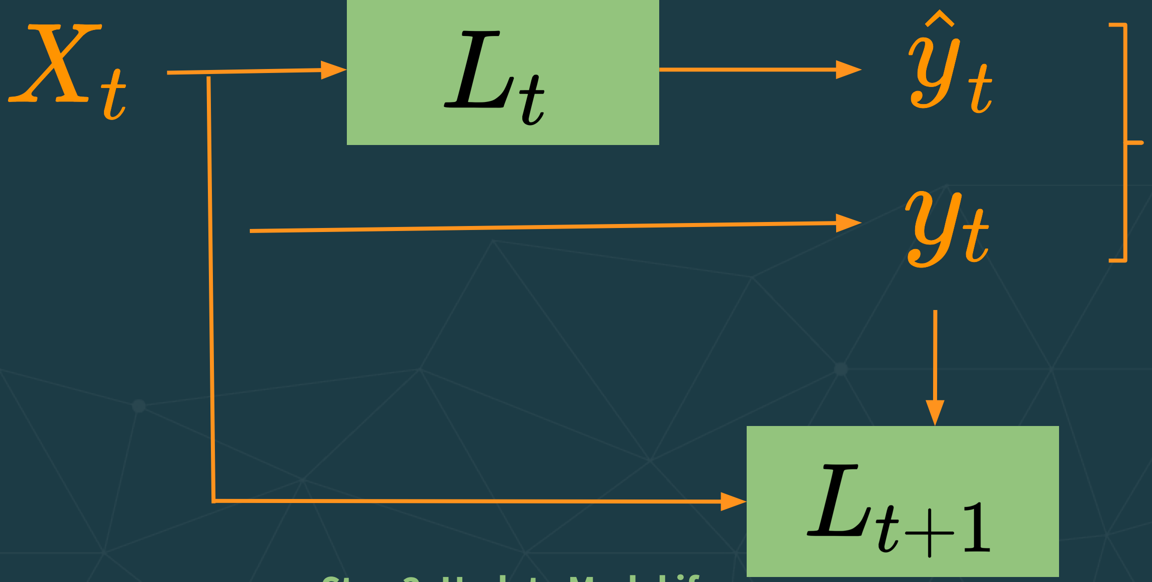
(does change detection as well)

Legend:
X : input examples
y : class labels
L : decision model for prediction
based on a learning algorithm y = L(X)

Adaptive Learning Model

Step 1: Predict

Step 2: Diagnose / Compute Loss Function



$$f\left(\hat{y}_t, y_t\right)$$

(does change detection as well)

Step 3: Update Model if necessary

Legend:
X : input examples
y : class labels
L : decision model for prediction
based on a learning algorithm y = L(X)

Lightbend

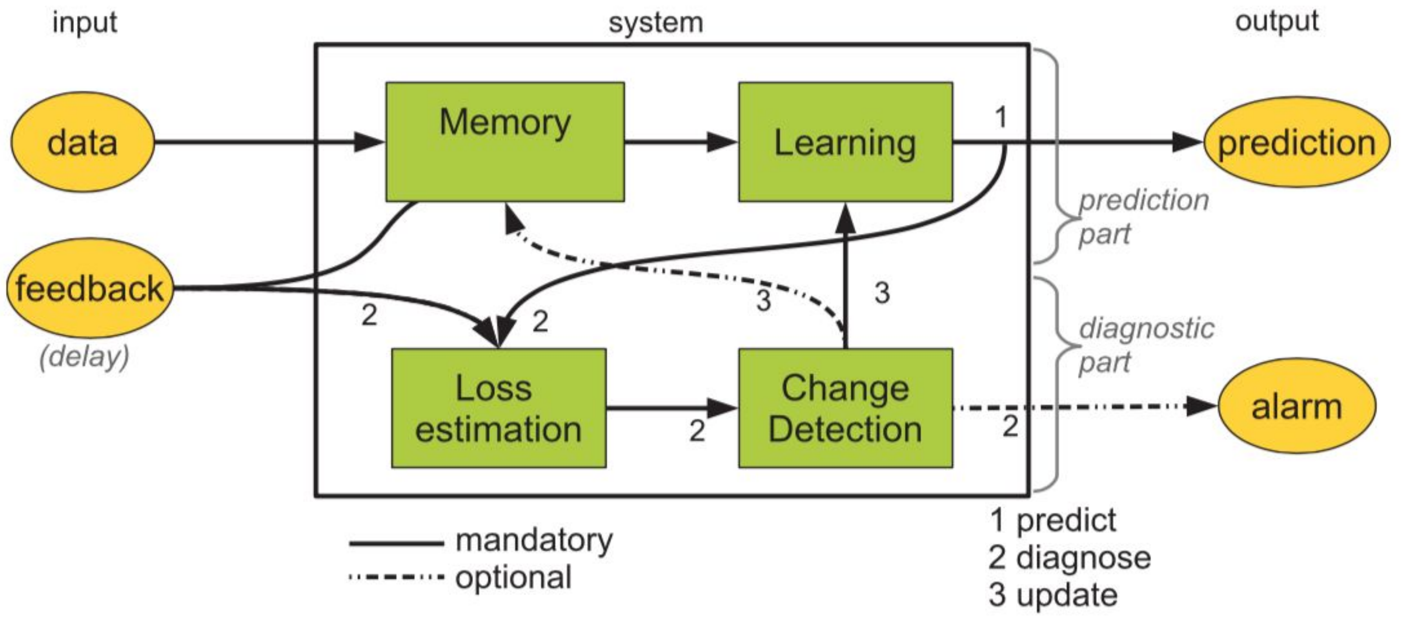


Fig. 3. A generic schema for an online adaptive learning algorithm.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. ACM Comput. Surv. 46, 4, Article 44 (March 2014), 37 pages. DOI: https://doi.org/10.1145/2523813

Lightbend

Evaluation Metrics

- Difference from batch mode
- Prequential evaluation model (test-then-train)

Evaluation Metrics - Batch

Learning Model
(Train)

$$(x_i, y_i) \longrightarrow \hat{y}_i$$

Examples of the form of pairs

Decision Model output

$$\text{True} \ (\hat{y}_i = y_i)$$

$$\text{False} \ (\hat{y}_i \neq y_i)$$

Error!

Error rate is the probability of observing $\text{False} \ (\hat{y}_i \neq y_i)$

Evaluation Metrics - Streaming

- Samples are analysed sequentially in order of arrival and they become immediately inaccessible
- Each sample serves 2 purposes - first we test our model on the sample (prediction) and then we train the model with the sample
- Testing the model on samples that we have not yet seen

$$x_i \xrightarrow{\text{Learning Model (Predict)}} \hat{y}_i$$

$$(x_i, y_i) \xrightarrow{\text{Learning Model (Train)}}$$

Evaluation Metrics - Streaming

- Now we can define the *Prequential Error*, computed at time t, based on a accumulated sum of a loss function between prediction and observed values

$$P_e(t) = \frac{1}{t} \sum_{k=1}^{t} L\left(y_k, \hat{y}_k\right)$$

Prequential Evaluation

- Evolution of learning as a process
- The model becomes better and better as we see more and more examples
- Recency is important - compute the prequential error using a forgetting mechanism

Forgetting mechanism for error estimation

- Prequential accuracy ***over a sliding window*** of a specific size with the most recent observations

- Fading factors that weigh observations ***using a decay factor*** $\boldsymbol{\alpha}$

The ADWIN Algorithm

An Adaptive Windowing Algorithm with forgetfulness

Learning from Time-Changing Data with Adaptive Windowing *

Albert Bifet   Ricard Gavaldà
Universitat Politècnica de Catalunya
{abifet,gavalda}@lsi.upc.edu

17 October 2006

The ADWIN Algorithm

- Windows of varying size (recomputed online)
- Automatically grows the window when no change occurs and shrinks it when data changes
- Whenever two “large enough” sub-windows exhibit “distinct enough” averages
  - We can conclude that the corresponding “expected values” are different
  - The older portion of the window is dropped

The ADWIN Algorithm - Notations and Settings

- a (possibly infinite) sequence of real values $x_1, x_2, .. , x_t, ..$
- a confidence value $\delta \in (0, 1)$
- the value of x_t is available only at time t
- each x_t is generated according to some distribution D_t independent of every t
- μ_t and σ_t^2 denote the expected value and variance of x_t when it is drawn according to D_t
- x_t is always in $[0, 1]$

Tail

Most recently added item

1 0 1 0 1 0 1 1 0 1 1 1 1 1 1

1

Window W

$\hat{\mu}_W$: observed average of elements in W

n : length of the window

Tail

Most recently added item

1 0 1 0 1 0 1 1 0 1 1 1 1 1 1

1

Window W

$\hat{\mu}_W$: observed average of elements in W

n : length of the window

2

Split into subwindows of lengths n_0 and n_1 such that $n_0 + n_1 = n$

1 | 0 1 0 1 0 1 1 0 1 1 1 1 1 1

W_0 W_1

1 0 | 1 0 1 0 1 1 0 1 1 1 1 1 1

W_0 W_1

1 0 1 | 0 1 0 1 1 0 1 1 1 1 1 1

W_0 W_1

Lightbend

Tail — Most recently added item

1 0 1 0 1 0 1 1 0 1 1 1 1 1 1

1

Window W

$\hat{\mu}_W$: observed average of elements in W

n : length of the window

2

Split into subwindows of lengths n_0 and n_1 such that $n_0 + n_1 = n$

1 | 0 1 0 1 0 1 1 0 1 1 1 1 1 1

W_0 W_1

1 0 | 1 0 1 0 1 1 0 1 1 1 1 1 1

W_0 W_1

1 0 1 | 0 1 0 1 1 0 1 1 1 1 1 1

W_0 W_1

3

For each of the subwindows check if

$$\left| \hat{\mu}_{W_0} - \hat{\mu}_{W_1} \right| \geq \epsilon_{cut}$$

$\hat{\mu}_{W_0}$: average of elements in W_0

$\hat{\mu}_{W_1}$: average of elements in W_1

ϵ_{cut} : threshold depending on n, n_0, n_1 and the confidence level of the algorithm

Tail

Most recently added item

1 0 1 0 1 0 1 1 0 1 1 1 1 1 1

2 Split into subwindows of lengths n_0 and n_1 such that $n_0 + n_1 = n$

1 | 0 1 0 1 0 1 1 0 1 1 1 1 1 1

W_0 W_1

1 0 | 1 0 1 0 1 1 0 1 1 1 1 1 1

W_0 W_1

1 0 1 | 0 1 0 1 1 0 1 1 1 1 1 1

W_0 W_1

Lightbend

1 Window W

$\hat{\mu}_W$: observed average of elements in W

n : length of the window

3 For each of the subwindows check if

$$\left|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}\right| \geq \epsilon_{cut}$$

$\hat{\mu}_{W_0}$: average of elements in W_0

$\hat{\mu}_{W_1}$: average of elements in W_1

ϵ_{cut} : threshold depending on n, n_0, n_1 and the confidence level of the algorithm

4 Whenever this happens, drop W_0 from W and the window compresses

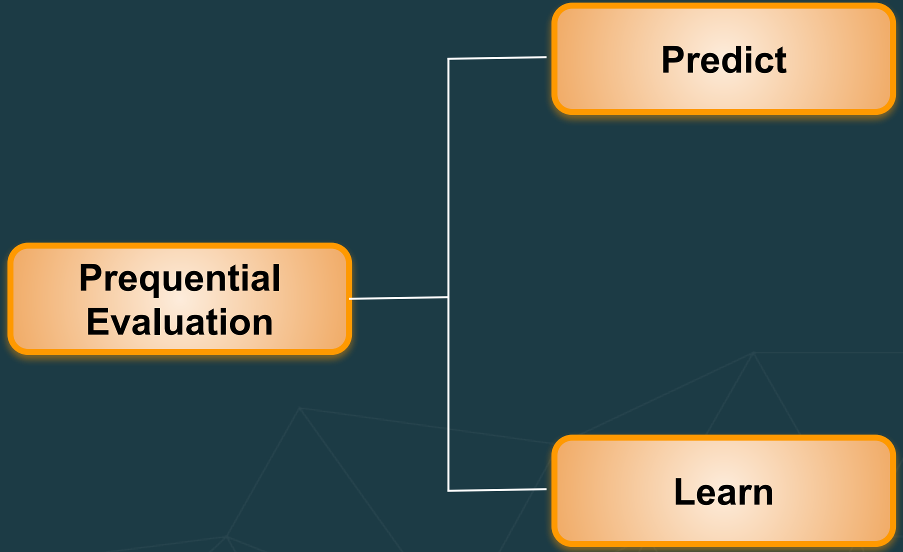
The ADWIN Algorithm

- When observed average in both subwindows differs by more than the threshold, the old part is discarded
- The new part gives the new correct mean
- ADWIN is not just a heuristic algorithm (unlike many of its predecessors), it comes with theoretical guarantees on the rates of false positives and false negatives and the size related to the rate of change

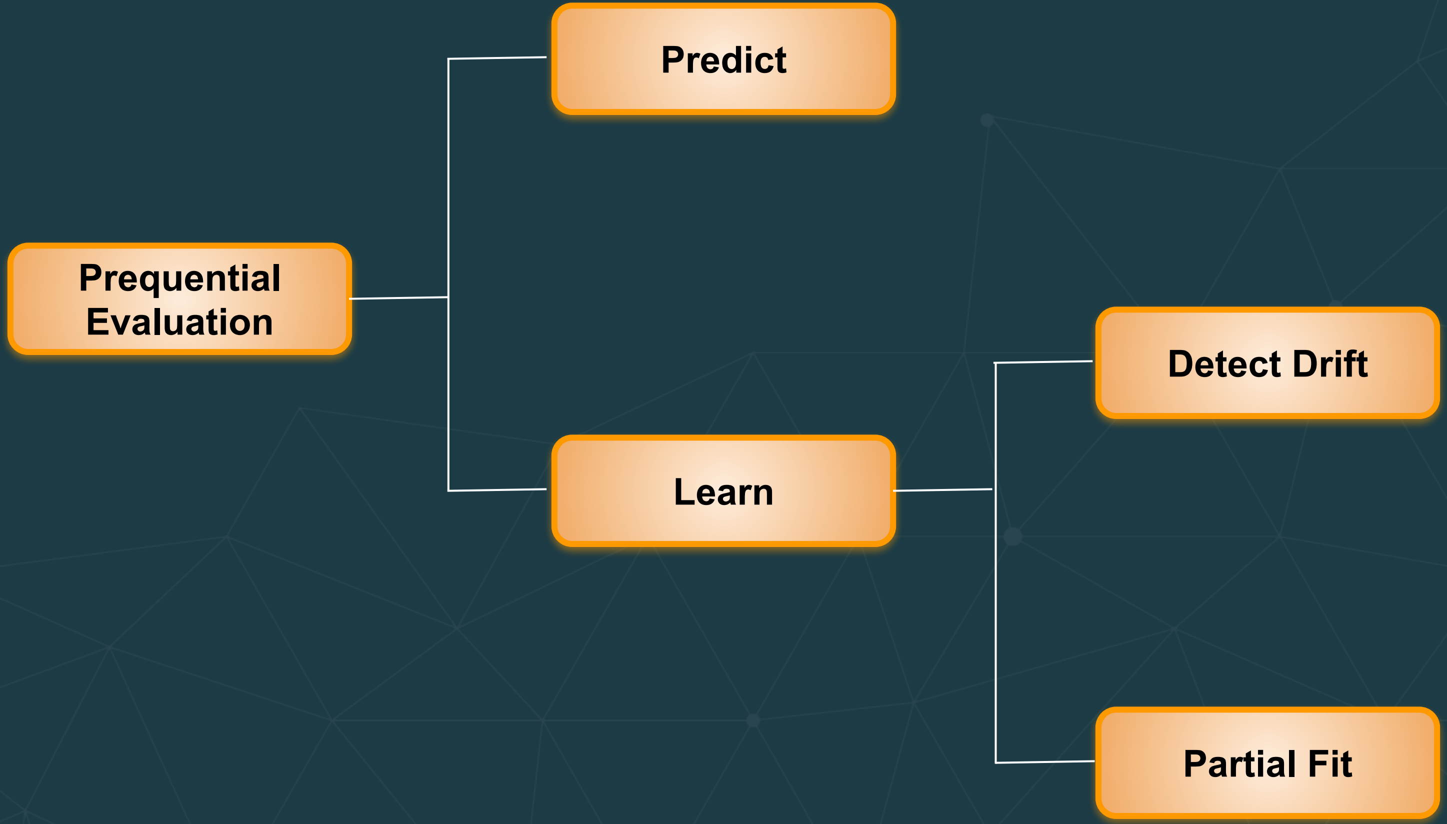
Concept Drift Detection and Retraining

Prequential Evaluation

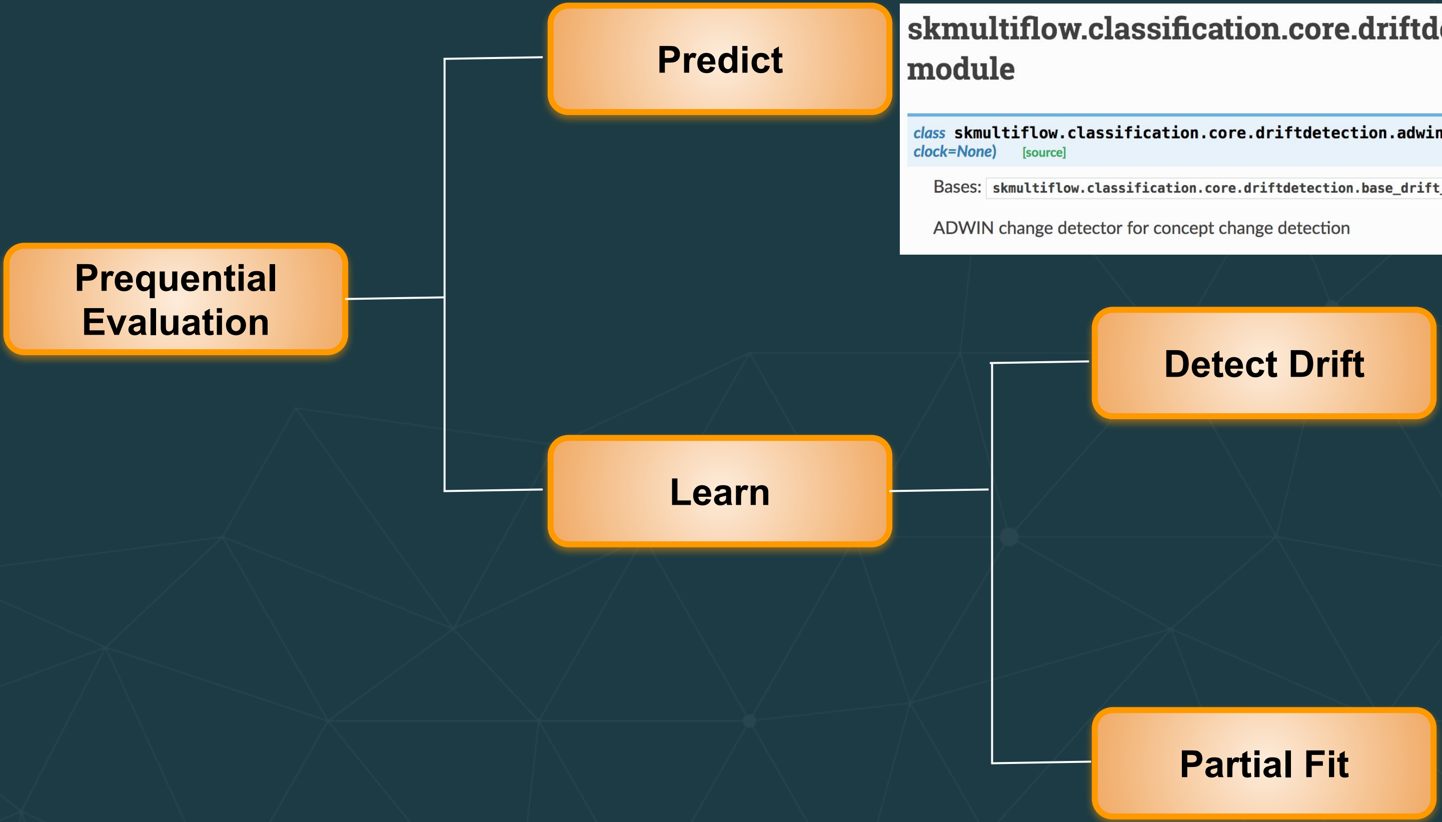
Concept Drift Detection and Retraining



Concept Drift Detection and Retraining



Concept Drift Detection and Retraining



skmultiflow.classification.core.driftdetection.adwin module

class `skmultiflow.classification.core.driftdetection.adwin.ADWIN`(*delta=0.002*, *clock=None*) [source]

Bases: `skmultiflow.classification.core.driftdetection.base_drift_detector.BaseDriftDetector`

ADWIN change detector for concept change detection

Concept Drift Detection and Retraining

- Prequential Evaluation
  - Predict
  - Learn
    - Detect Drift
    - Partial Fit

skmultiflow.classification.core.driftdetection.adwin module

class **skmultiflow.classification.core.driftdetection.adwin.ADWIN**(*delta=0.002, clock=None*) [source]

Bases: `skmultiflow.classification.core.driftdetection.base_drift_detector.BaseDriftDetector`

ADWIN change detector for concept change detection

partial_fit (*self, X, y, classes=None, sample_weight=None*) [source]

Incremental fit on a batch of samples.

This method is expected to be called several times consecutively on different chunks of a dataset so as to implement out-of-core or online learning.

This is especially useful when the whole dataset is too big to fit in memory at once.

sklearn.naive_bayes.GaussianNB

class `sklearn.naive_bayes.` **GaussianNB** (*priors=None*, *var_smoothing=1e-09*) [source]

Gaussian Naive Bayes (GaussianNB)

Can perform online updates to model parameters via `partial_fit` method. For details on algorithm used to update feature means and variance online, see Stanford CS tech report STAN-CS-79-773 by Chan, Golub, and LeVeque:

http://i.stanford.edu/pub/cstr/reports/cs/tr/79/773/CS-TR-79-773.pdf

Methods

`fit` (self, X, y[, sample_weight])	Fit Gaussian Naive Bayes according to X, y
`get_params` (self[, deep])	Get parameters for this estimator.
`partial_fit` (self, X, y[, classes, sample_weight])	Incremental fit on a batch of samples.
`predict` (self, X)	Perform classification on an array of test vectors X.

Demo

Learning from Production

Production Issues

- Model is no more an immutable function in the on-line ML case
  - Model is part of the ML pipeline's runtime state
- How do I deal with failures?
  - Model should be checkpointed along with the data.

Lightbend

Production Issues

- Many things can go wrong and affect training.

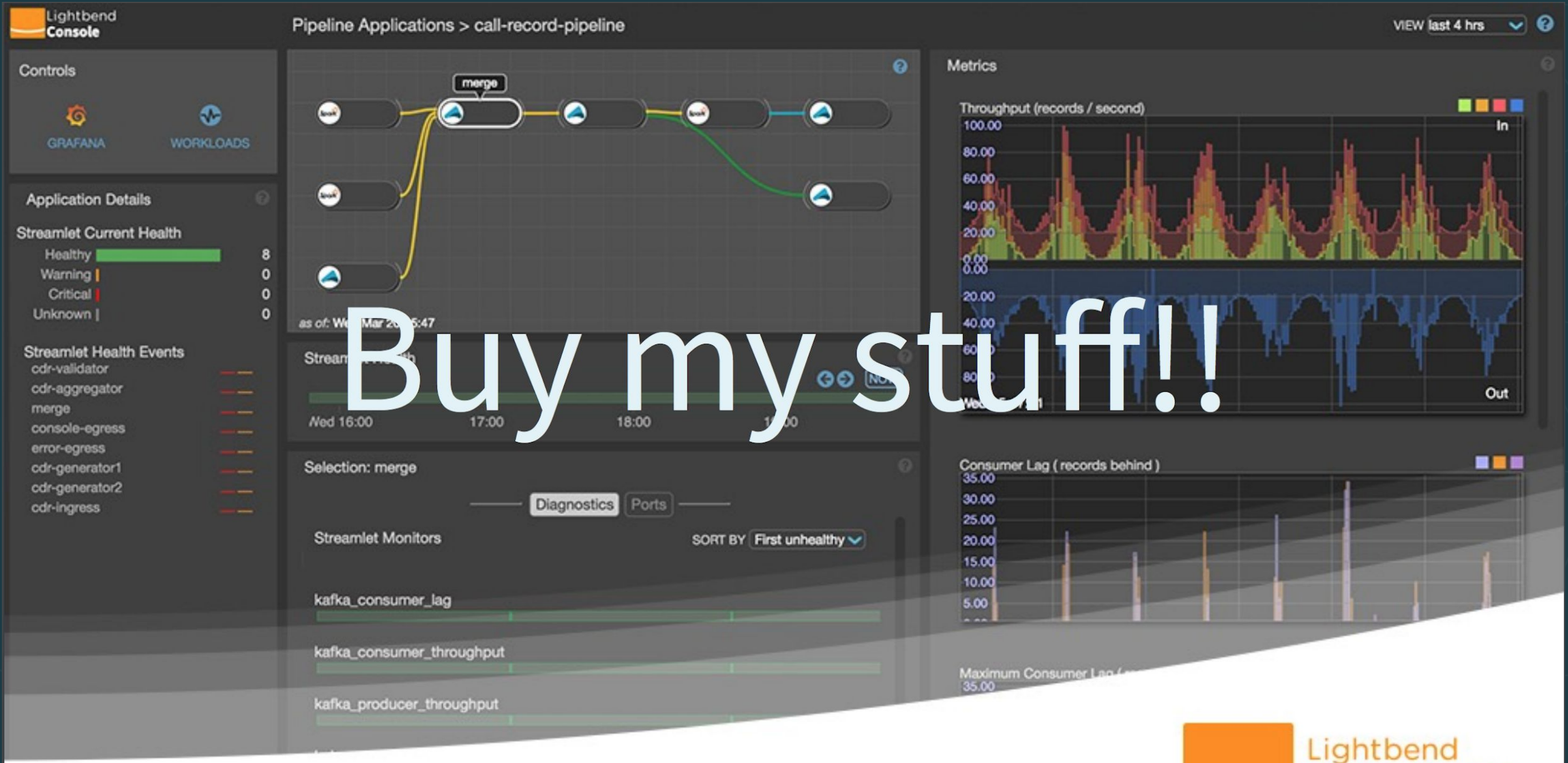
    - You need to protect your training process eg. remove outliers
    - Quality of data can change,
    - Monitor performance eg. quality of results and quality of input data
    - Measure response time
    - Numerical stability of algorithms. Eg, what is the best approach to calculate on-line statistics? Eg. Welford algorithm for stdv or moving average. How do I do a simple sum with a stream of values without losing precision eg. Kahan algorithm.

Production Issues

- Scaling?
  - Scale up best option eg. IoT use cases, one model per sensor, installation etv. Scale out is possible in certain cases eg. distributed on-line k-means.
  - Resource management?

- Model interpretability as a function of time
  - on-line partial dependence plots?
- Model Security
  - Data governance

Buy my stuff!!



What we're up to at Lightbend…

lightbend.com/lightbend-pipelines-demo

Lightbend Pipelines

Lightbend

Thank You

Stavros Kontopoulos
stavros.kontopoulos@lightbend.com
@s_kontopoulos

Debasish Ghosh
debasish.ghosh@lightbend.com
@debasishg