Stream mining. Machine learning methods for data streams. Part III: concept drift

Maciej Grzenda, PhD, DSc
Maciej.Grzenda@pw.edu.pl

Faculty of Mathematics and Information Science
Warsaw University of Technology, Poland

Warszawa, 2025

Introduction

Changes in data streams

It is natural to expect that x instances arriving with time will vary i.e. will include different values, even if we assume the same dimensionality.

Data we observe is the result of a random process that generates an item according to probability distribution. Hence, as observed in [1], two cases may occur:

- this probability distribution is the same at any time i.e. the distribution remains stationary
- change(s) may occur i.e. the underlying distribution changes. When a change occurs, the underlying distribution varies from one step to the next.

Learning in nonstationary conditions

Learning in nonstationary environments attract major attention, which is best illustrated by the fact that it gave a name for one of key survey papers [4].

Beyond IID data

As observed in [1], almost all developments in data mining, and particularly classification, assume that data are **independently and identically distributed (IID)**.

Thus, IID data are assumed to come from:

- stationary distribution i.e. distribution that does not change with time
- the distribution randomly producing data in no particular order

In a streaming setting, none of the two assumptions are typically met i.e.:

- we frequently observe labels to be correlated over some periods of time e.g. periods of intrusion attempts [1] or device failures
- and distribution changes over time e.g. due to seasonality or ageing of sensors [4].

Designing ML methods which exploit autocorrelations is one of the challenges in stream mining domain.

Concept drift

In line with [4], let P stand for a process, which provides a data stream $\{(\mathbf{x}_t, y_t)\}$ sampled from unknown distribution $p_t(\mathbf{x}, y)$.

For some processes, the probability $p_t(y|\mathbf{x})$ changes at some period(s). This means learning is performed in nonstationary environment and *concept drift* occurs.

Two cases can be distinguished [4]:

- *real (concept) drift* [4] also referred to as *real change* [1], *actual drift* [8], *class drift* [12] or *prior probability shift* [12] i.e. when $p_t(y|\mathbf{x})$ varies over time
- *virtual (concept) drift* [4] also referred to as *virtual change* [1], *covariate drift* [12] i.e. when $p_t(\mathbf{x})$ varies over time, but without changing the probability of classes $p_t(y|\mathbf{x})$

Concept drift types I

Many different types of concept drift have been identified. These types partly focus on different aspects. Hence, a distinction can be made between:

- *sudden (abrupt)* change [1, 3] - when distribution remains unchanged for a long time, then changes in a few steps to a significantly different one. Such change is also called *shift*
- *gradual* [1, 4, 8] change when a new concept gradually replaces the old one
- *incremental* change [1, 8] - when there is a minor change at each time step, but accumulated changes become significant
- *global* [1] or *partial* change [1] - depending whether entire instance space or just a part of it is affected

Concept drift types II

- *recurrent concepts* [1, 4] also referred to as *reoccurring concepts* [8] when distributions from the past appear later again (after some change period)
- *permanent* or *transient* drift [4] depending on whether distribution change is not limited in time, or the effect of drift disappears after some time.

The duration of concept drift

Finally please note, that concept drift may happen at an exact time stamp, but also last for a long time [8].

Drift types illustrated

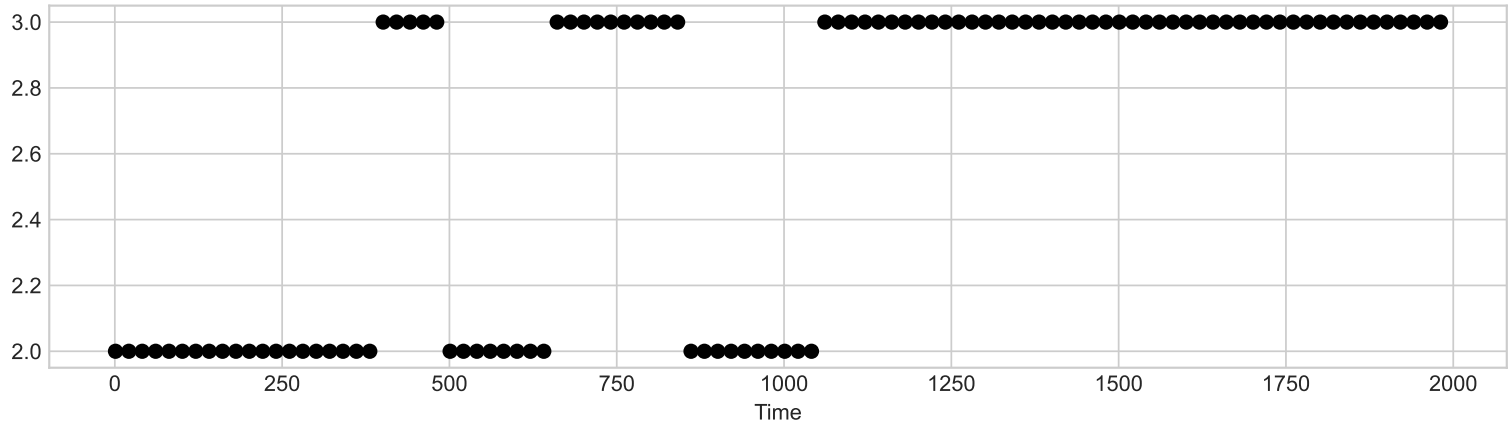

Figure: Gradual drift example

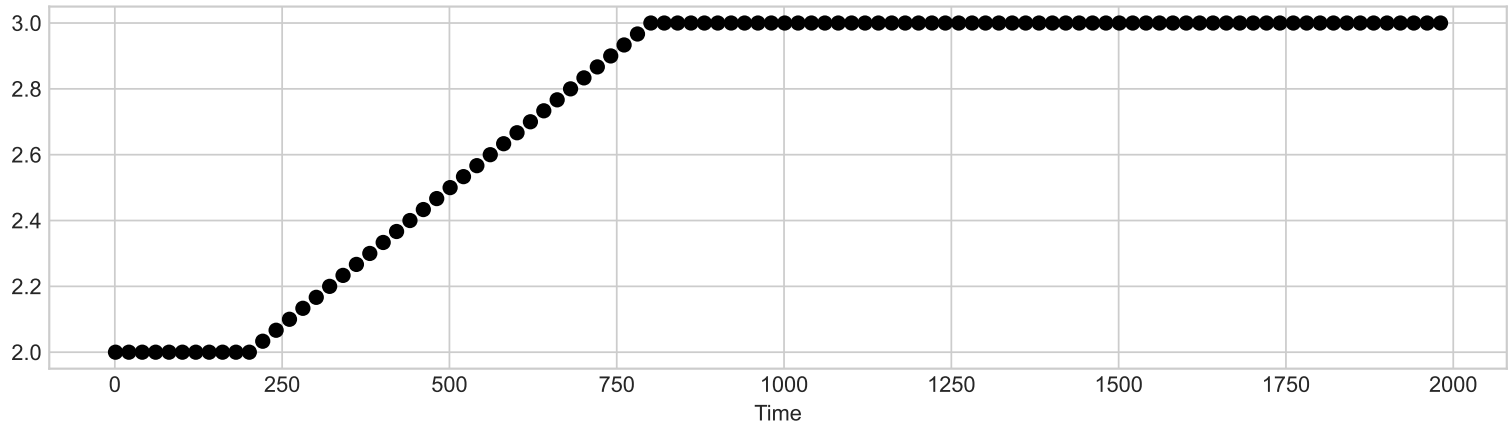

Figure: Incremental drift example

Drift types illustrated

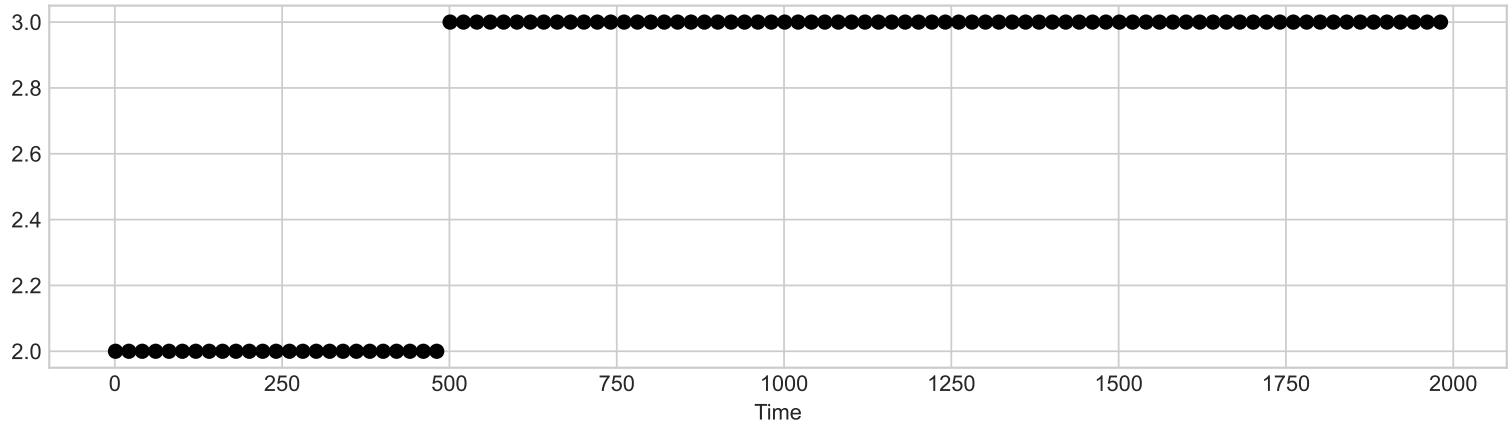

Figure: Sudden drift example

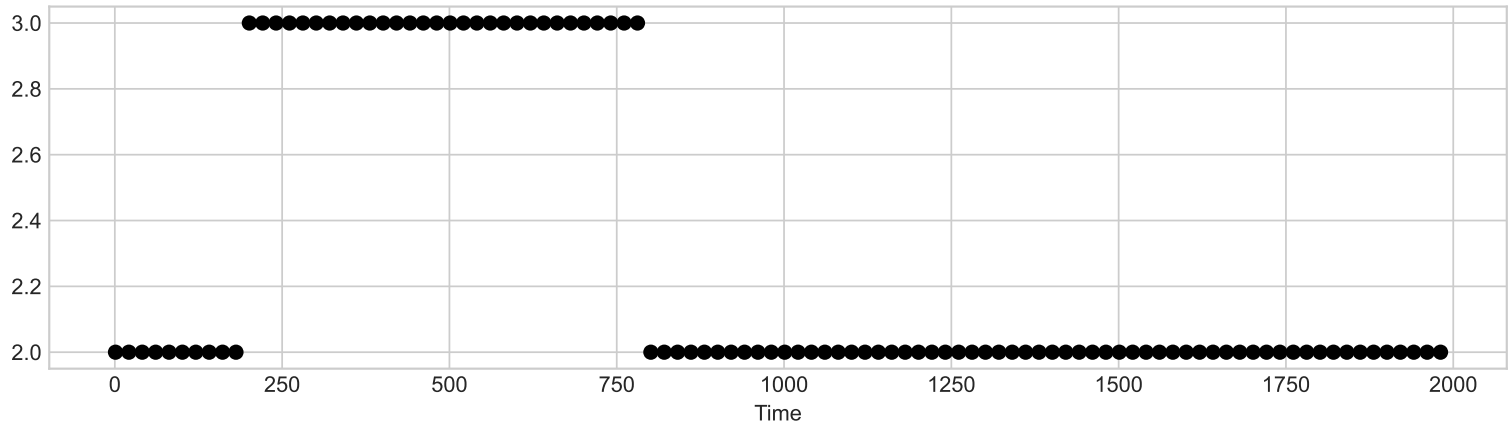

Figure: Reoccurring concepts example

The sources of concept drift

Concept drift occurs at time stamp $t+1$ when $\exists t : P_t(X, y) \neq P_{t+1}(X, y)$ [8] i.e. when joint probability changes.

This can be decomposed as $P_t(X, y) = P_t(X) \times P_t(y|X)$

Hence, this means one of two cases occurred [8]:

- $P_t(X)$ has changed while $P_t(y|X)$ has not. This is referred to as *feature space drift*
- $P_t(y|X)$ has changed, which may be accompanied by a change of $P_t(X)$ or not. Still, this means *decision boundary drift*.

Drift consequences

Which of the two cases listed above necessitates an update of a classifier?

Concept evolution

One more aspect of changes in data streams is *concept evolution* [1]. This phenomenon occurs when new classes appear in evolving data streams.

Examples include: new topics discussed on Twitter, or new types of attack observed in intrusion detection systems.

Some methods have an explicit support for concept evolution. As an example streaming version of kNN will simply start considering new class labels once they are observed among instance neighbours.

Sample stream changes - CoMobility use case

Mobility choices data

In the CoMobility project we analysed what makes citizens select different means of transport such as walking, taking a bike, using private car or public transport.

The data used come from surveys, in which respondents document their journeys and transport modes used for them. This is extended with other data and provides labelled $(\mathbf{x}, y)$ instances, where y is the transport mode selected for the journey.

Funding

The mobility choices data comes from the CoMobility project. The CoMobility benefited from a 2.05 million€ grant from Iceland, Liechtenstein and Norway through the EEA Grants. The aim of the project is to provide a package of tools and methods for the co-creation of sustainable mobility in urban spaces. See `https://streamudatasys.mini.pw.edu.pl` for details of our works.

Changes in distances if made by walking

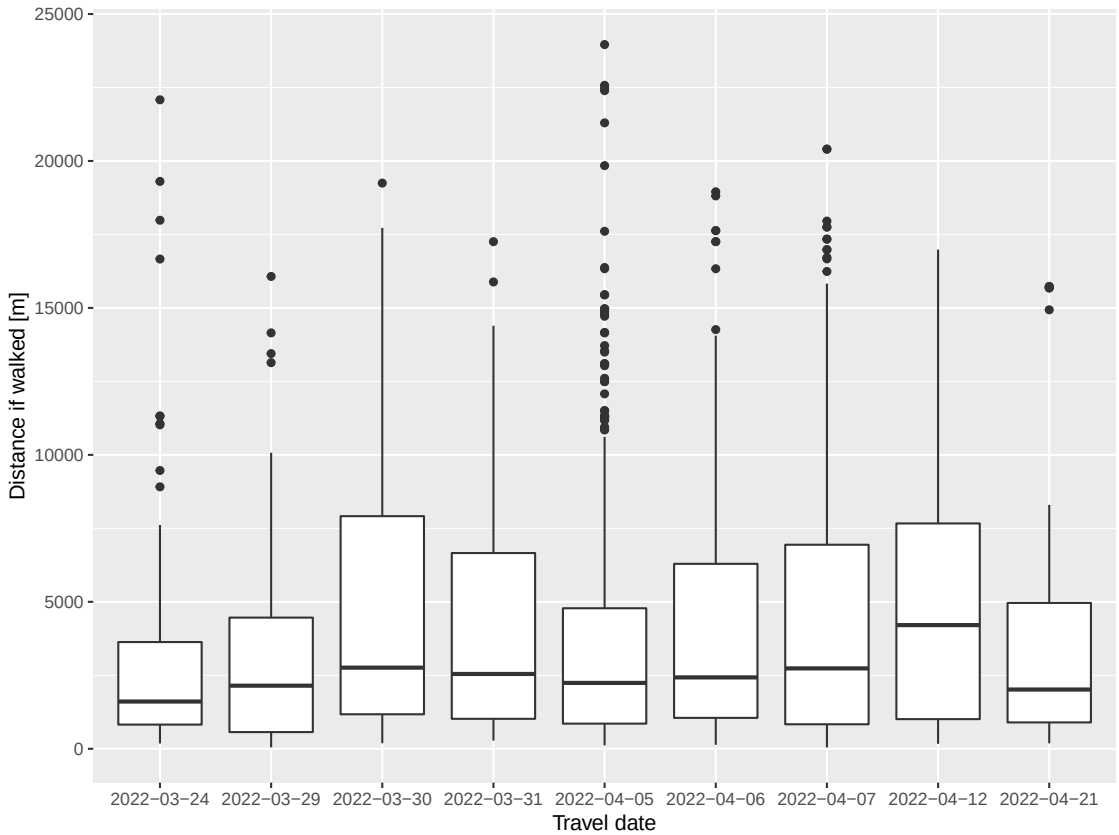

Figure: Distances of journeys made over individual days described by respondents, if made by walking

For individual real trips reported by respondents, we calculated distance by walking.

Changes in duration of journeys if made by walking

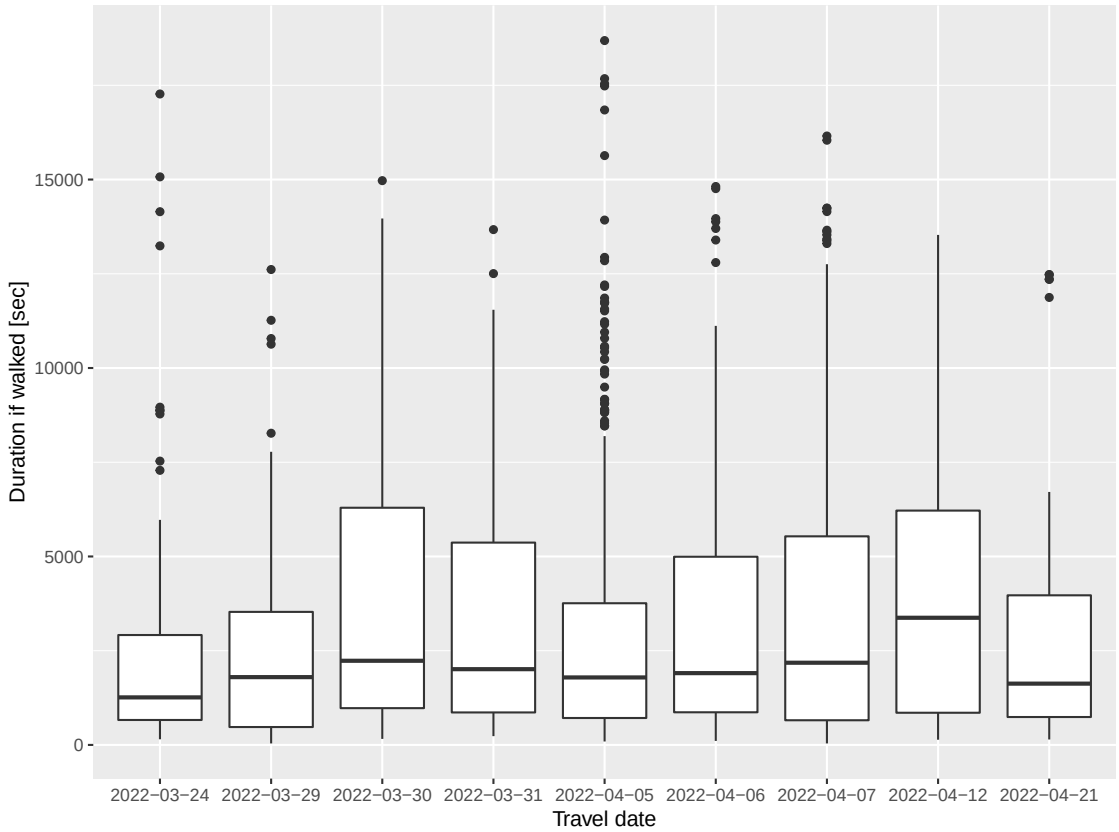

Figure: Duration of journeys made over individual days described by respondents, if made by walking

Changes in the number of connections available for each journey

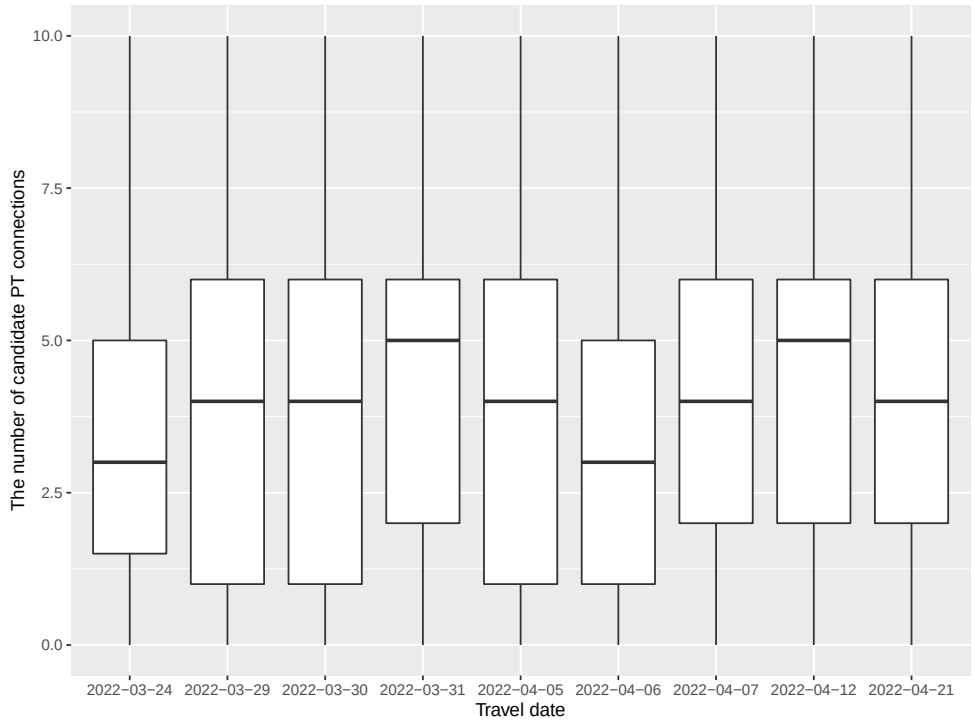

Figure: The number of available public transport connections

The question arises whether changes in histograms of individual features f over different windows of instances $\mathbf{x}_i^f, \ldots, \mathbf{x}_{i+\delta}^f$ are due to random fluctuations or distribution changes

Concept drift detectors

Introduction

Detecting both real and virtual drift is fundamental for learning from data streams e.g. training online classifiers.

A classifier h_t not matching $P_t(X, y)$ may yield largely wrong predictions $\hat{y} = h_t(x)$.

The question arises how to detect changes in $P(X)$ and $P(y|X)$.

This is why many different concept drift detection algorithms were proposed. Such algorithms can be a part of online learning methods.

Hence, the key requirements for these detectors are similar to the requirements laid out for stream mining methods i.e.:

- operating with a stream as an input rather than a data set
- limited performance overhead
- limited (bounded) memory use, irrespective of the volume of stream instances

The taxonomy of drift detection algorithms I

A recent survey on learning under a concept drift [8] identifies three method groups:

- **Error rate-based drift detection methods** - these methods detect drift by tracking changes in the error of classifiers.
  As an example, in Drift Detection Method (DDM) proposed by Gama et al. in [7], when the error rate within the time window increases significantly, first a warning, then after exceeding even higher threshold, a change is reported [1, 8]. This is used to build a new learner (after the warning point), and replace the old one with the new one (after the change point).

The taxonomy of drift detection algorithms II

- **Data distribution-based drift detection methods** - in this case, differences between distributions of historical data and new data are quantified. This is typically done based on some distance function. The differences found to be statistically significant typically trigger changes in the models. Most frequently, the calculation of dissimilarities between distributions relies on two sliding windows - one for historical data and one for new data.
- **Multiple hypothesis test drift detection methods** - in this group of methods, multiple hypothesis tests are used to detect drift. This can be done in two ways:
  - parallel multiple hypothesis tests
  - hierarchical hypothesis tests

  As an example, one test can be used for detection and the next one for validation of this detection [8].

ADWIN detector

One of the popular detectors is **ADWIN** proposed in [2].

In ADWIN, the adaptive window approach is used for streaming data and applied to detect changes in the average value of a stream of bits or real-valued numbers.

The size of sliding windows used for change detection is not constant and defined a priori, but depends on the rate of detections. Thus for stationary data, the window is growing, but in the case of detection, the window is narrowing to discard historical data.

The only parameter of the detector is a confidence value $\delta \in (0, 1)$, controlling the sensitivity of the detection, i.e. influencing the ratio of false positives. A change is detected with a probability of at least $1 - \delta$.

ADWIN is considered an example of *error rate-based drift detection* in [8], as it is frequently used to track changes in errors of online classifiers. See `setInput(double intEntrada, double delta)` of `moa.classifiers.core.driftdetection.ADWIN` class in MOA for final ADWIN algorithm.

KSWIN detector

Another recently proposed approach to concept drift detection relies on the Kolmogorov-Smirnov test. The test is applied to sliding windows populated with recent data instances from a data stream [10].

The parameters for the **KSWIN** detector are:

- the probability α of the test statistic
- the sizes of two sliding windows used for the detection of a difference between the distributions of data present in these windows.

Concept change is detected when the distance between the empirical cumulative distribution functions (eCDFs) of the two differently sized windows exceeds the α-dependant threshold.

KSWIN is an example of data distribution-based drift detection methods. It is a recent proposal. Hence, it could not be listed in [8], which is an earlier study.

HDDM detectors

Research into change detectors is largely inspired by the need to detect when an update of the learning model is needed to adapt the model to concept drift.

A family of methods monitoring the mean estimated from real values with an explicit focus on monitoring the values of performance measures of learning models was proposed in [6].

The methods rely on **HDDM** algorithms proposed in the study and use Hoeffding's inequality to report warnings and actual drifts based on two confidence levels – the parameters of the change detector.

HDDM methods are considered an example of *error rate-based drift detection* in [8], as they were proposed to monitor the performance of learning models.

HDDM_A and HDDM_W detectors

Both detectors take a stream of bits as an input. By default the meaning of the bit was whether error occurred, i.e. 1 denotes prediction error, while 0 denotes correct prediction.

HDDM_A uses moving **a**verage as an estimator and input and is suggested for abrupt drifts. HDDM_W relies on **w**eighted moving average for the same and is recommended for gradual drifts. Both detectors can first report a warning that a drift is likely to occur.

Parameter	Default	Description
drift_confidence	0.001	Confidence for the drift detections.
warning_confidence	0.005	Confidence for the warnings about possible drift.
lambda_val	0.05	The weight given to recent data. Smaller values mean less weight is given to recent data. Only for HDDM_W
two_sided_test	False	If True, will monitor error increments and decrements (two-sided). By default, it will only monitor increments.

Table: HDDM_A and HDDM_W parameters. Developed based on `https://riverml.xyz/0.22.0/api/drift/binary/HDDM-A/` and `https://riverml.xyz/0.22.0/api/drift/binary/HDDM-W/`

Drift detection examples

Sample drift detection code

```
1
2  from river import drift
3  import numpy as np
4
5  dataStreamIncremental = np.concatenate([np.full(200,2),np.
       arange(2,3,1/600),np.full(1200,3)])
6  adwin = drift.ADWIN()
7  data_stream=dataStreamIncremental
8
9  plt.figure(figsize=(12,3))
10 plt.plot(instanceIndexes[::20], data_stream[::20], 'o', color=
       'black');
11 plt.xlabel('Time')
12
13 # adapted from river example
14 for i, val in  enumerate(data_stream):
15     _ = adwin.update(val)
16     if adwin.drift_detected:
17         print("Change detected at example {i}, input value at
               which change was detected: {val}")
18         plt.axvline(x=i,color='r')
19 plt.savefig('ADWIN.pdf')
```

The output of drift detection: incremental drift

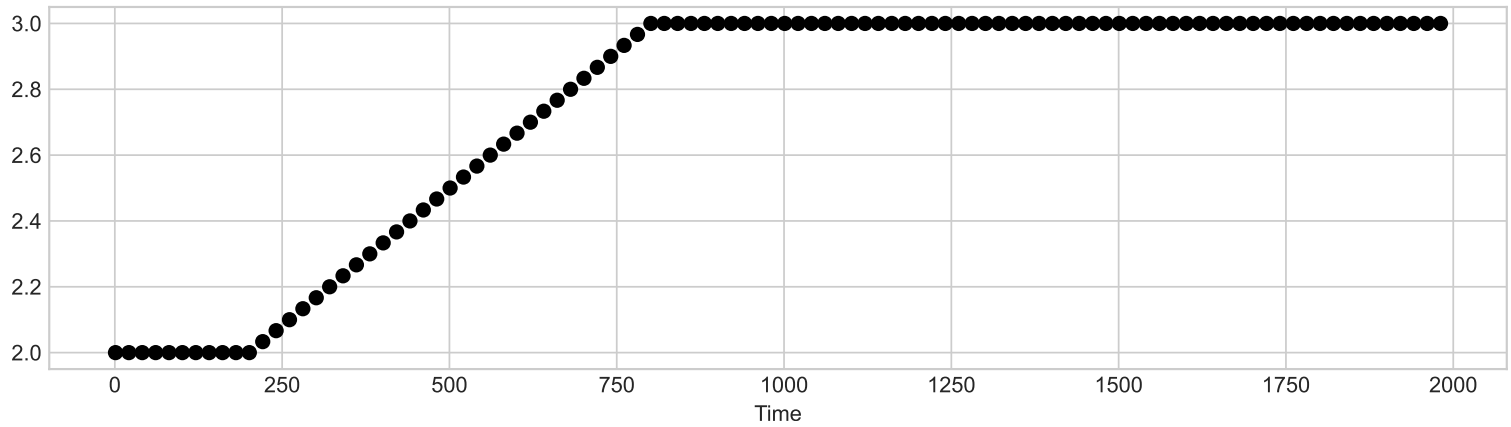

Figure: Incremental drift example

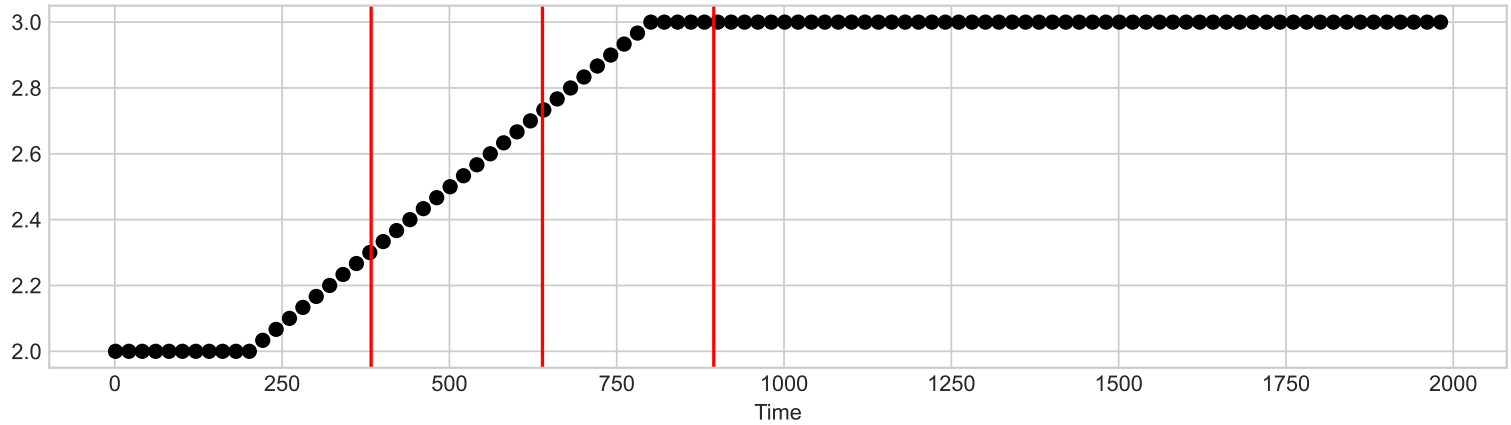

Figure: Drift detection performed by ADWIN. The moments of drift detection are shown in red. Default ADWIN settings were applied.

The output of drift detection: gradual drift

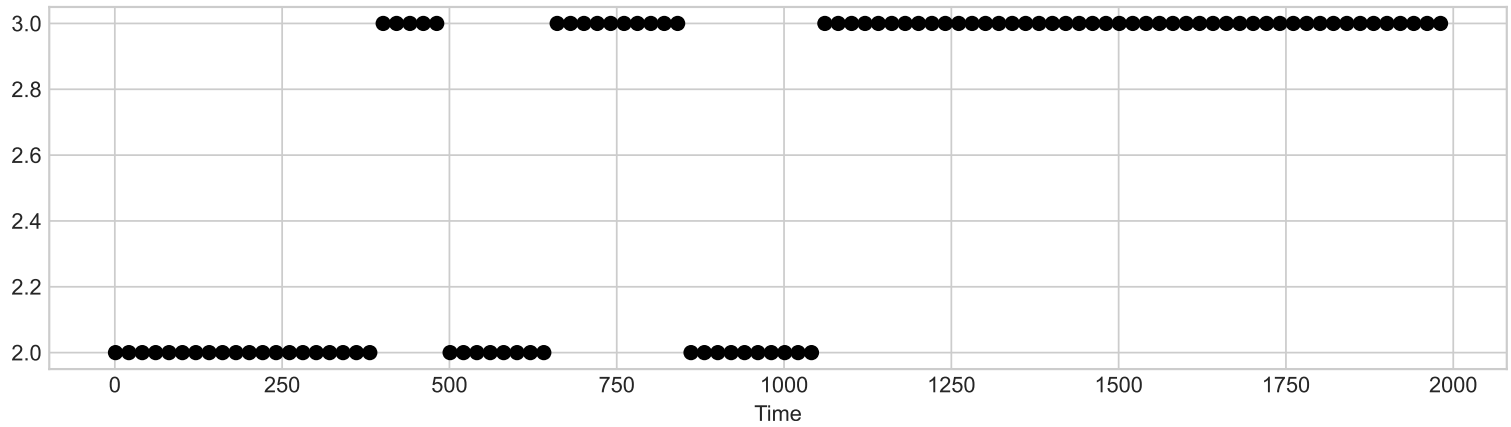

Figure: Gradual drift example

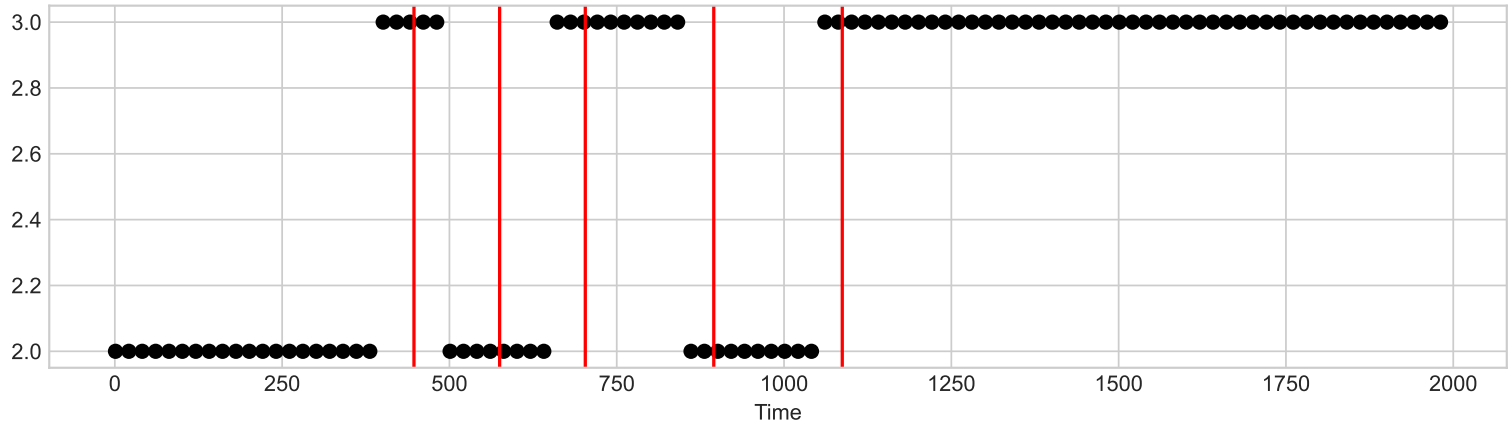

Figure: Drift detection performed by ADWIN. The moments of drift detection are shown in red. Default ADWIN settings were applied.

The output of drift detection: sudden drift

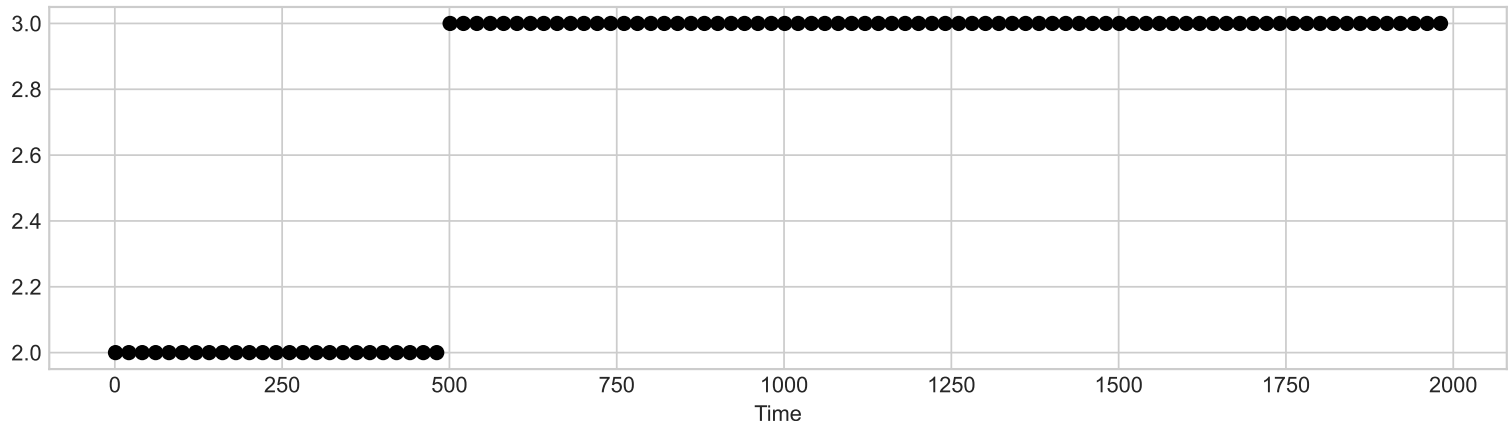

Figure: Sudden drift example

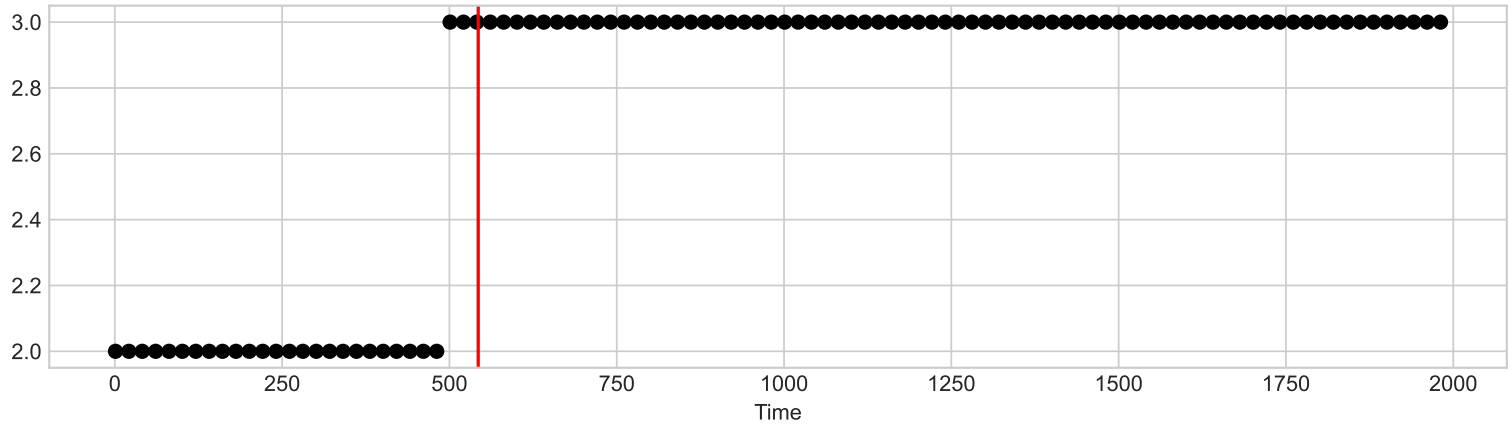

Figure: Drift detection performed by ADWIN. The moment of drift detection is shown in red. Default ADWIN settings were applied.

The output of drift detection: reoccurring concepts

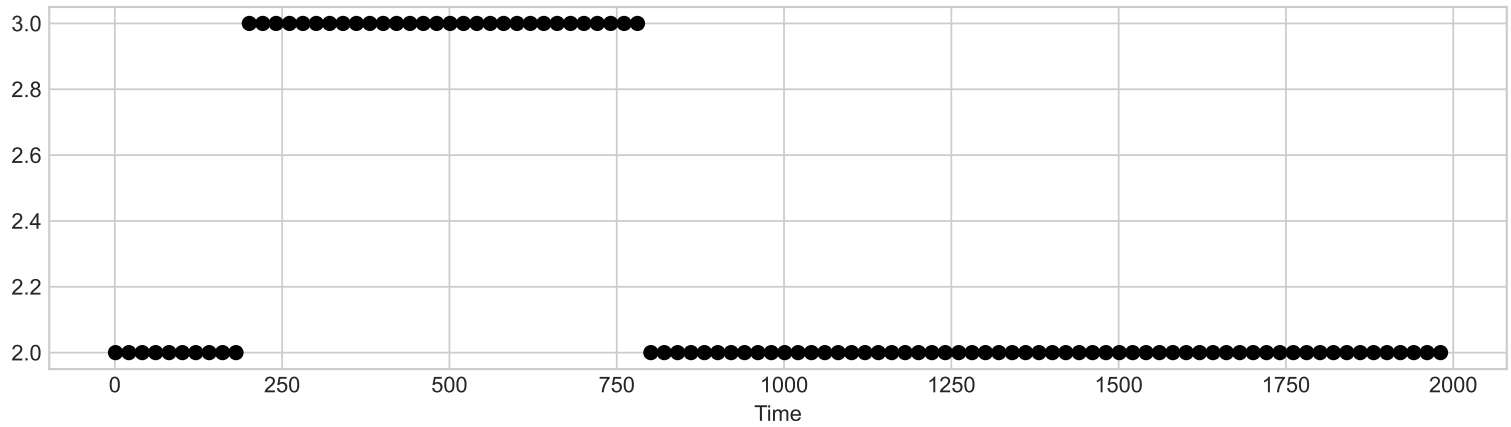

Figure: Reoccurring concepts example

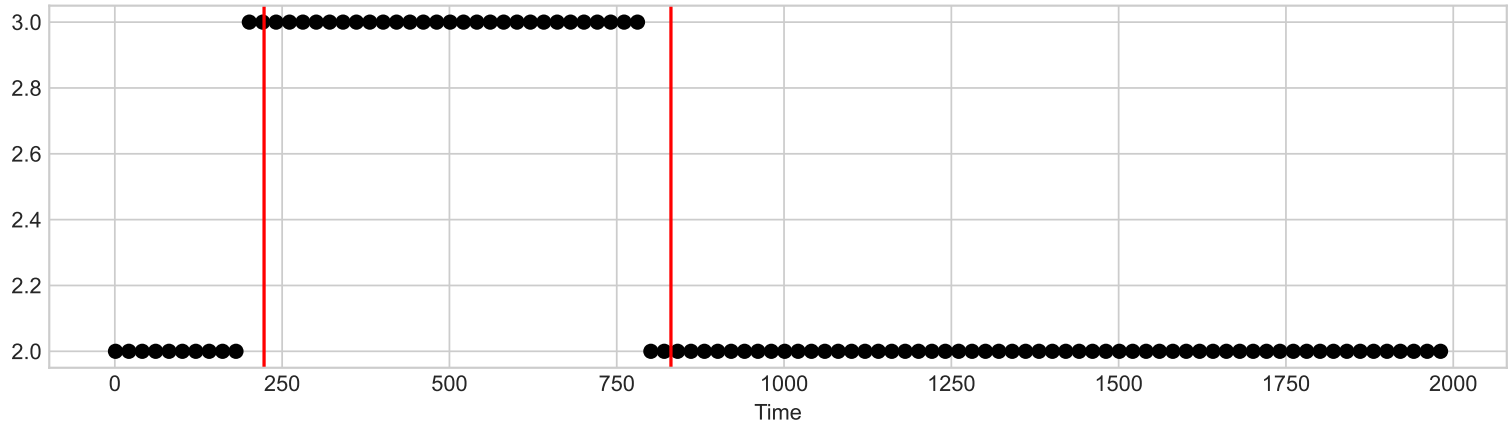

Figure: Drift detection performed by ADWIN. The moment of drift detections are shown in red. Default ADWIN settings were applied.

Discussion

Drift detection is inevitably delayed.

This is to avoid detecting drift, while just random fluctuations occurred in the data.

Drift detector settings can be used to control:

- detection latency
- computational overhead
- significance i.e. control false positive rate

Sample drift detection code - setting detector parameters

```
  from    river import drift
# default value of delta=0.002
adwin = drift.ADWIN()
adwinModified = drift.ADWIN(delta=0.1)

data_stream=dataStreamIncremental
plt.figure(figsize=(12,3))
plt.plot(instanceIndexes[::20], data_stream[::20], 'o', color=
    'black');
plt.xlabel('Time')

for i, val in  enumerate(data_stream):
    adwin.update(val)
    adwinModified.update(val)
    if adwin.drift_detected:
        plt.axvline(x=i,color='r')
    if adwinModified.drift_detected:
          plt.axvline(x=i,color='b')
```

The output of drift detection: the impact of significance setting

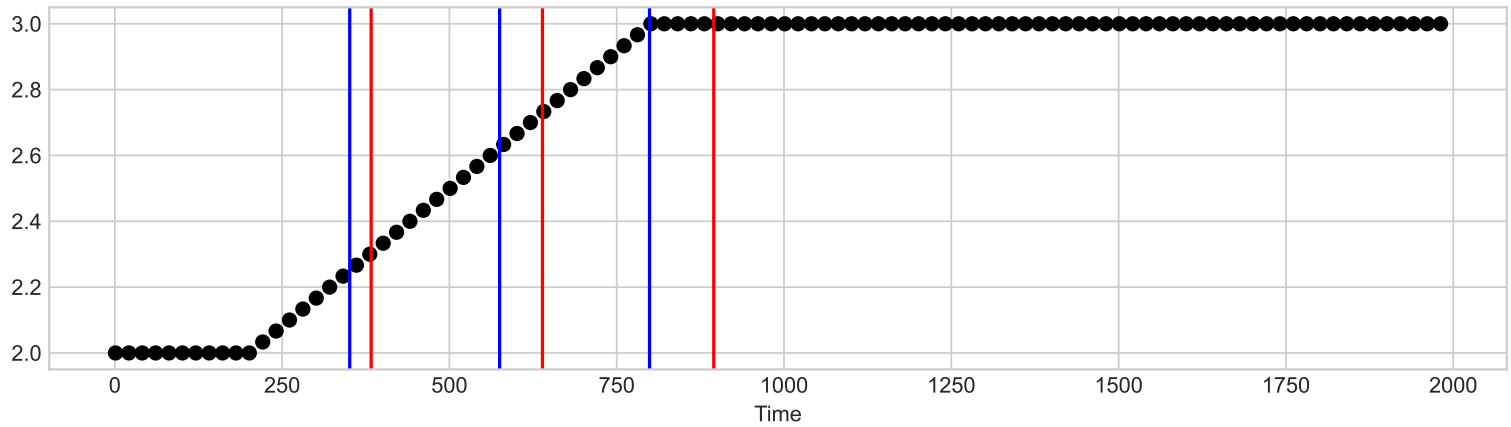

Figure: Drift detection performed by ADWIN. The moment of drift detections for $\delta = 0.002$ and $\delta = 0.01$ are shown in red and blue, respectively.

Which δ setting was better? Is it always the case?

The output of drift detection: random data

Let us see what happens when ADWIN is applied to random data.

Let us generate random data sampled from normal distribution $\mu = 3, \sigma = 0.1$. Hence, both mean value and standard deviation are constant in time.

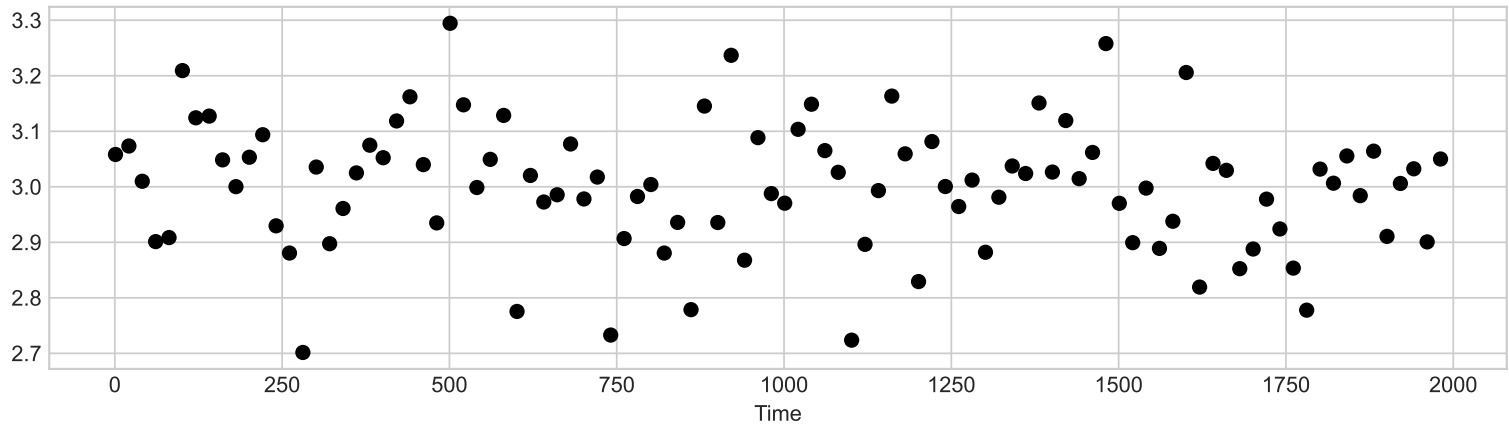

Figure: Drift detection performed by ADWIN. The moments of drift detection are shown in red. Default ADWIN settings were applied.

Is this behaviour of ADWIN i.e. no drift detections correct?

The output of drift detection: different detectors

Let us compare change detections provided by different detectors, each supplied with the same data stream.

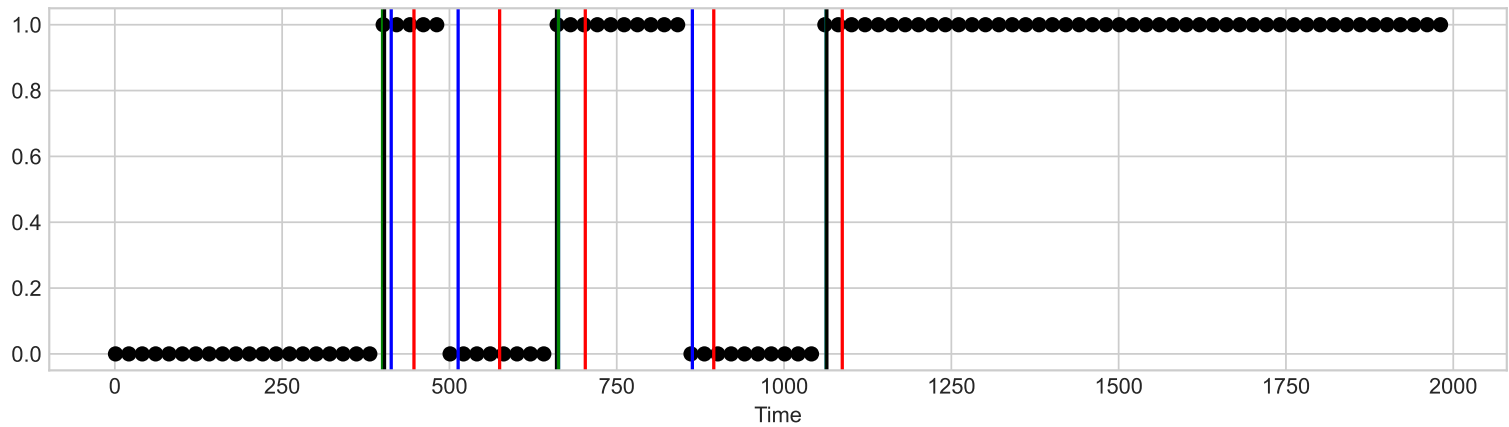

Figure: Drift detection performed by ADWIN, KSWIN, HDDM_A and HDDM_W. The moments of drift detections are shown in red, blue, green respectively. Default settings of all detectors were applied.

How to explain these differences?

Generating data with concept drift

Concept drift stream generator

	Description
Idea	Create concept drifting stream by merging two streams with the same attributes and classes
Details	As an example the two merged streams can be two Agrawal streams in which different labelling functions were used. The merging is driven by $f(t) = \frac{1}{1+e^{\frac{-4(t-p)}{w}}}$. If a random number is greater than $f(t)$ than the instance is taken from the original stream, otherwise from the second stream.
# features	the same as the number of features in source streams
# classes	the same as the number of classes in source streams
Key other parameters	• original stream • the second stream i.e. the one that will introduce concept drift • central position p of the concept drift change (in the number of instances) • the width of drift change w period (in the number of instances)
Drift	Yes
Source and further details	[1], `https://riverml.xyz/0.14.0/api/datasets/synth/ConceptDriftStream/`

Sample mixing of two data streams

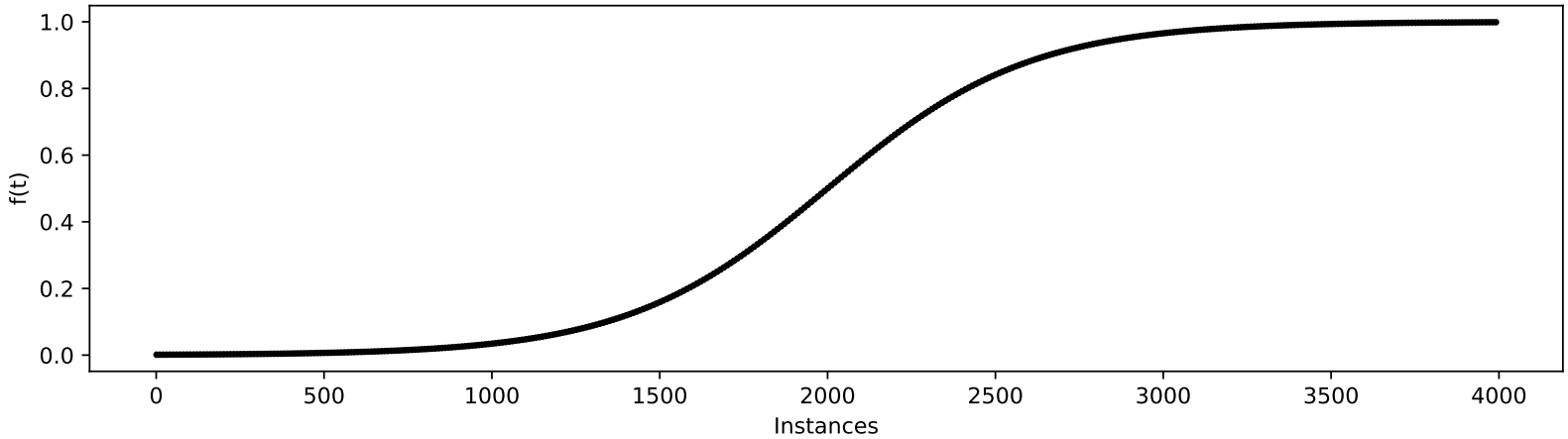

Figure: The $f(t)$ function for drift position p set to 2000 and width $w = 1200$

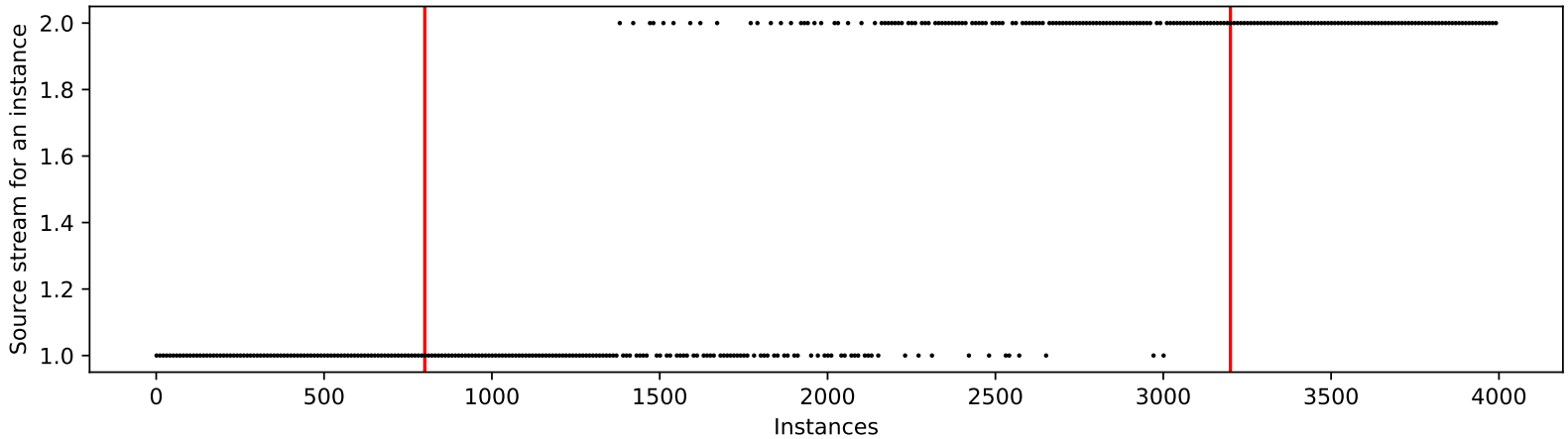

Figure: Sample selection of stream instances from two participating streams for the same function. Red lines placed at $p - w$ and $p + w$ instances

Merging streams - discussion

Two or more streams can be merged.

This makes it possible to implement multiple concept drift periods of different widths i.e. speed of transition from one concept to another.

Merging can be used to:

- generate data with real concept drift by merging streams with different decision boundaries and/or different noise levels i.e. different probabilities $p(y|x)$ as in the case of hyperplane data
- generate data with virtual concept drift by merging data with different shares of instances of different types

Note that generators explicitly introducing drift into their base versions also exist. As an example, `LEDDrift` generator makes it possible to define the number of attributes the values of which will be swapped

Feature selection in stream mining

Feature selection in batch learning

There are two classical approaches to feature selection [9]:

- *feature ranking*. In methods of this type, we determine the ranking of the importance of features and on its basis we make a decision about the selection of features. This approach is referred to as the filter approach [5, 13], as it means selecting features before training the model.
- *subset selection*. In these methods, a group of features is assessed simultaneously. The method of obtaining a subset of features is typically the so-called wrapper approach [5, 13]. The selection of features takes into account the results of model training with this subset.

Discussion

Because of the links *feature ranking* - *filtering approach* and *subset selection* -*wrapper approach*, the above division is often replaced by the filter/wrapper division [5]. Which approach sounds more suitable for stream mining?

Pearson's correlation coefficient

One of the basic methods of feature evaluation is the calculation of the correlation coefficient.

In the case of regression and binary classification problems, one of the solutions is to use the Pearson correlation coefficient to assess the significance of a feature. The value of this coefficient is the absolute value of the cosine of the angle between the vector of the values of variable X_j and the vector of the values of dependent variable Y.

$$C(j) = \frac{\sum_{i=1}^{n}(X_{i,j}-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_{i,j}-\bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i-\bar{Y})^2}}$$

Discussion

Pearson's coefficient illustrates well the limitation of filter-based methods. It may result in a low score for a feature that affects the value of the Y variable, but it is not a linear effect. Let use note that such impact could be successfully identified by e.g. a regression tree.

Feature selection in data stream preprocessing

An extensive survey of data preprocessing methods for data streams is provided in [11].

In the case of feature selection methods (FS), the key findings are as follows [11]:

- most of FS methods are naturally incremental methods designed for offline filtering methods. These filters rely on cumulative functions.
- feature selection methods differ in assumptions they make about data streams and what can change in data streams with time

Let us note that online FS methods relying on filtering approach have to calculate scores for individual features. This has to be done incrementally.

The non-trivial aspects of FS for data streams

The optimal subset of features may change with time due to concept drift. The phenomenon of changing relevance of features over time is known as *feature drift* and can be treated as a special case of real concept drift [11]. However, other changes also can be considered [11]:

- features can be assumed to arrive one-by-one, but feature vectors are initially available which is referred to as *streaming features*
- instances can be assumed to arrive sequentially, but the set of available features may change with time, which requires *online FS*. As an example, new tests can be made for patients with time.
- new classes may arrive with time

[11] observe that the scenario in which both new instances and new features can arrive with time matches best real-world problems and provide a survey of FS methods. Most of surveyed methods rely on filter approach, just two of them on the wrapper approach.

The consequences of feature set evolution

A side effect of allowing new features to occur in a data stream with time is that [11]:

- the dimensionality of consecutive instances may change with time. As an example, some $x_i \in \mathbb{R}^{S_1}$, while $x_j \in \mathbb{R}^{S_2}$ where $i < j, S_1 < S_2$
- conversion between feature spaces may have to be applied to map test instances to train space of a different (e.g. lower) dimensionality

Homogenisation between spaces based on padding with zeros missing feature in the other set can be a solution to this problem [11].

Even the natural scenario of adding new features to the instances of our data stream (results of new tests, readings form extra sensors) necessitates substantially different methods than for batch FS.

Summary

Preparing machine learning methods addressing concept drift is of fundamental importance in stream mining.

Hence, concept drift detectors have been proposed.

Different types of detectors are available. They can detect changes in errors made by online models, and changes in data distribution. Some of them may use multiple hypothesis tests to detect drift.

Moreover, ML methods capable of detecting concept drift and adapting their models are developed.

References I

[1] Albert Bifet, Ricard Gavaldà, Geoff Holmes, and Bernhard Pfahringer. *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press, 2018. https://moa.cms.waikato.ac.nz/book/.

[2] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 443–448. Society for Industrial and Applied Mathematics, 4 2007.

[3] Dariusz Brzezinski and Jerzy Stefanowski. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):81–94, 2014.

[4] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.

[5] Peter Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, USA, 2012.

References II

[6] Isvani Frias-Blanco, Jose del Campo-Avila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustin Ortiz-Diaz, and Yaile Caballero-Mota. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27:810–823, 3 2015.

[7] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In Ana L. C. Bazzan and Sofiane Labidi, editors, *Advances in Artificial Intelligence – SBIA 2004*, pages 286–295, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[8] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2019.

[9] Robert Nisbet, John Elder, and Gary Miner. *Handbook of Statistical Analysis and Data Mining Applications*. Academic Press, 2009.

[10] Christoph Raab, Moritz Heusinger, and Frank-Michael Schleif. Reactive soft prototype computing for concept drift streams. *Neurocomputing*, 416:340–351, 2020.

References III

[11] Sergio Ramírez-Gallego, Bartosz Krawczyk, Salvador García, Michał Woźniak, and Francisco Herrera. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing*, 239:39–57, 2017.

[12] Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30:964–994, 7 2016.

[13] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.