Change Detection fundamentals and applications to learning in nonstationary environments

Giacomo Boracchi

June 27th, 2024

Ulsan National Institute of Science and Technology

giacomo.boracchi@polimi.it

Tutorial Outline

- Learning in Nonstationary Environments (NSE): the General Picture
  - Fraud Detection
- Problem Formulation Concept Drift and Learning in NSE
- Major Approaches in Learning in NSE
- Concept Drift Detection by monitoring
  - classification error
  - raw data distribution
- Adaptation Strategies:
  - Active Approaches
  - Passive Approaches
- Concluding Remarks

Disclaimer

This tutorial is meant to illustrate the major challenges and the basic principles for learning in NSE.

The major expected outcome is to cast LNSE problems in a standard (statistical) framework, providing you the tools for possibly understanding other solutions not illustrated here

I will present a few methods, but these are not an exhaustive (nor updated) survey on the field

The General Picture

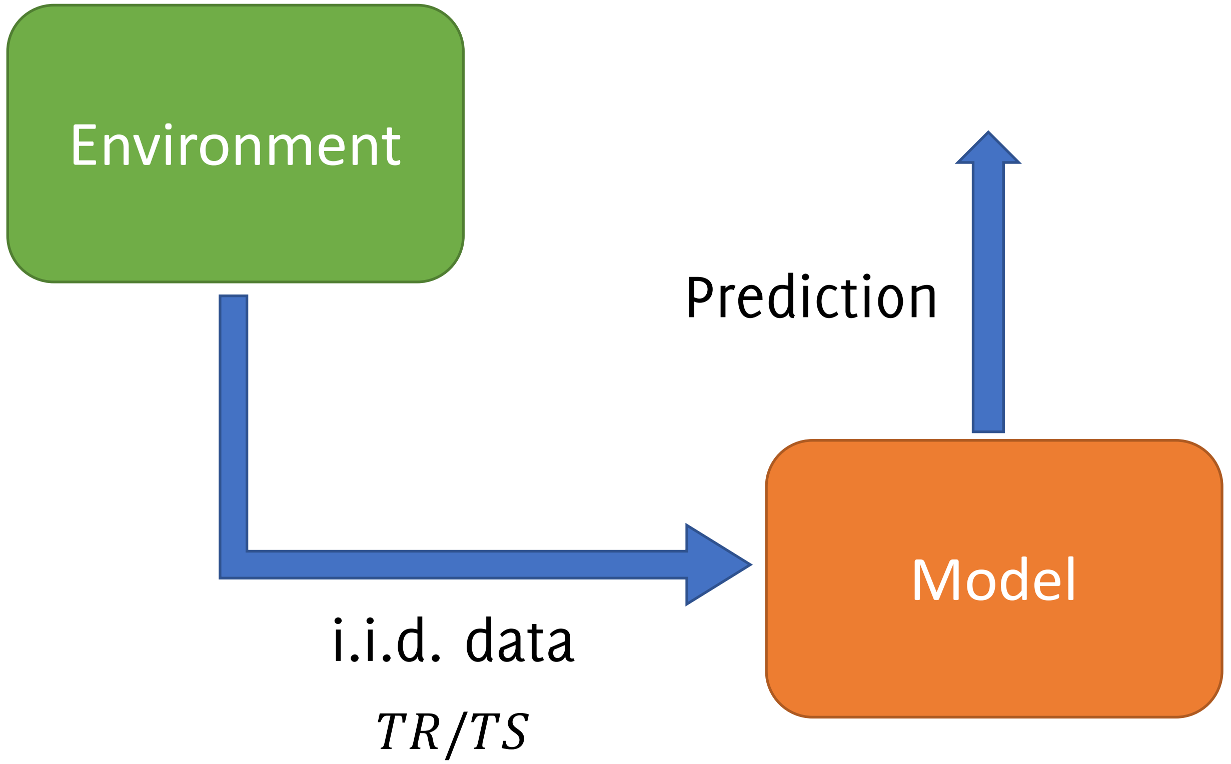
General ML Framework

Typical assumption in ML:

Incoming data (both training or testing) are ***independent and identically distributed*** *(i.i.d.) realizations of an unknown process*

The major focus is towards making data-driven models able to extract information out of training data (TR) to perform inference on test data (TS)

Rmk: Predictions **does not influence** the environment, nor TR/TS



This Course Framework

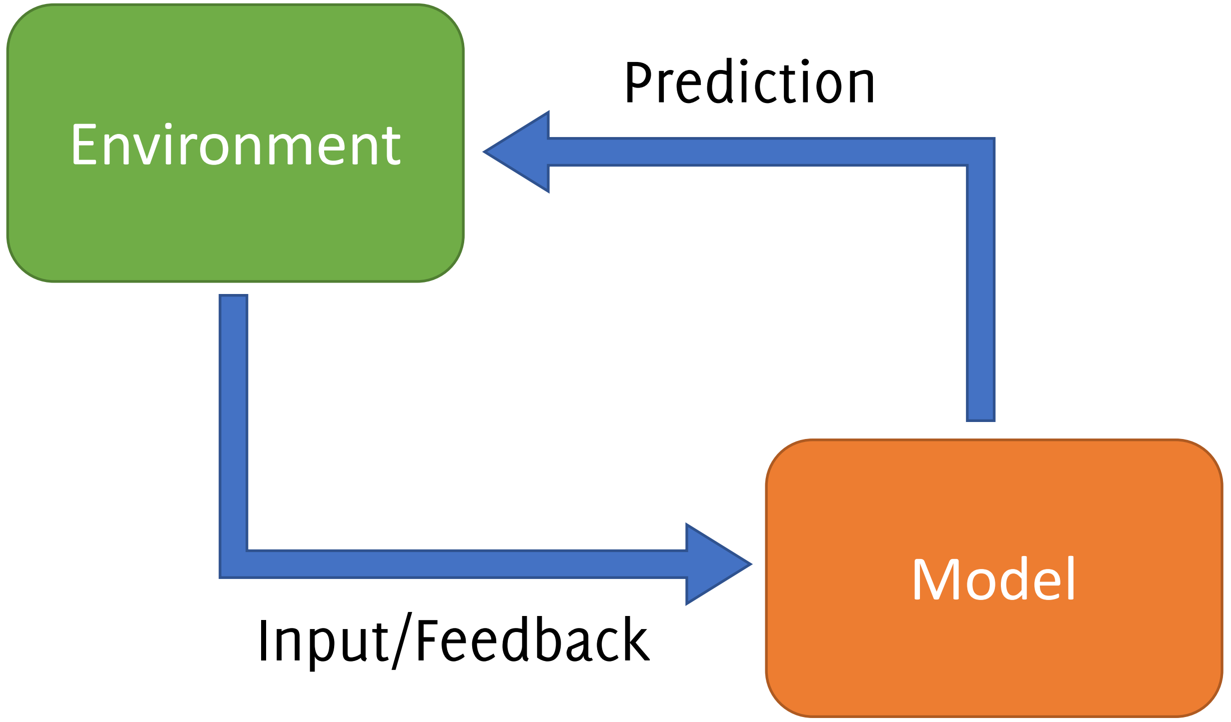
In a **streaming** scenario i.i.d. assumption often does not hold:

- the **environment** might be **changing or adversarial**
- It is not possible to ignore the **model-environment interactions,** since model outcomes are influencing the environment, or the way supervision is provided (feedback)

These settings call for:

- Techniques to **learn-adapt** the data-driven **model**
- Techniques to **monitor** the **model-environment interaction**

Rmk: Predictions **might influence** the environment or TR



Not i.i.d.

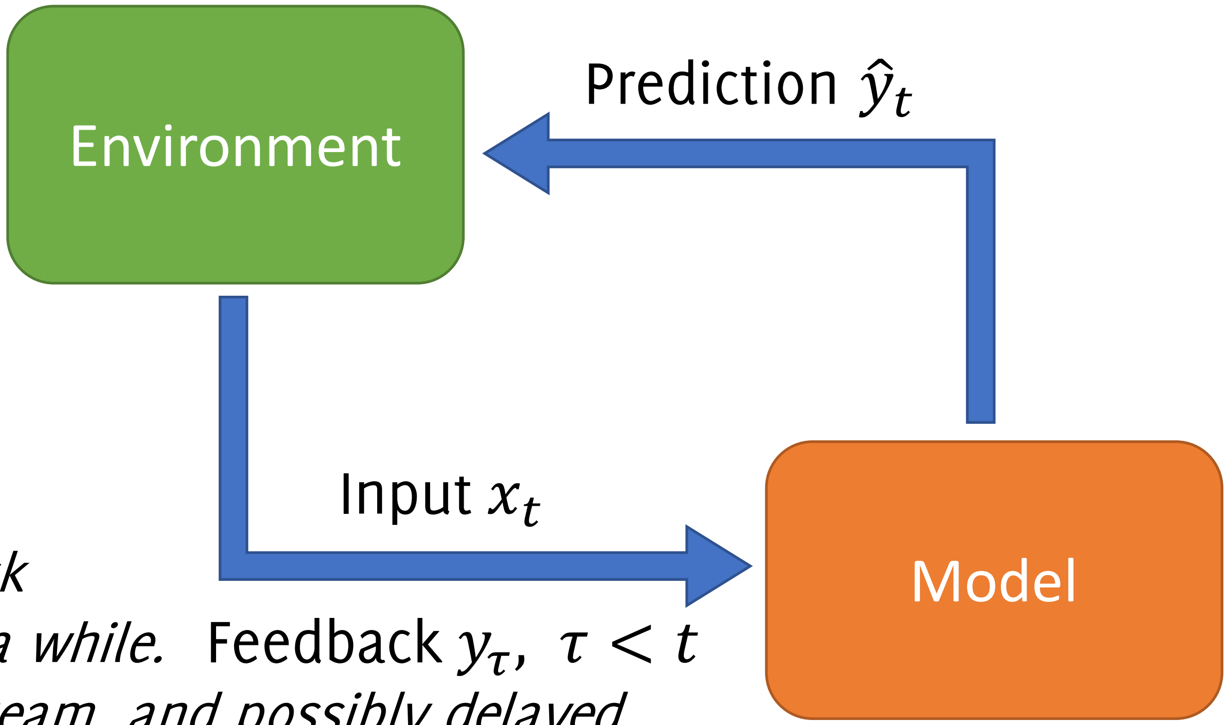
Learning in Non-Stationary Environment

At each time instant t

- we get an input x_t from a stream
- we generate a prediction $\hat{y}_t$
- **we get feedback** y_τ $(\tau < t)$
- we update the model as the learning problem might change

Example: **Fraud Detection**

You classify each transaction x_t assigning a label $\hat{y}_t$ (genuine/fraudulent), investigators check only those labels and return a feedback y_t after a while.
Feedbacks are not representative of the entire stream, and possibly delayed



Boracchi

Tutorial Overview

Typical assumption in ML:

Training and incoming data are i.i.d.

This course:

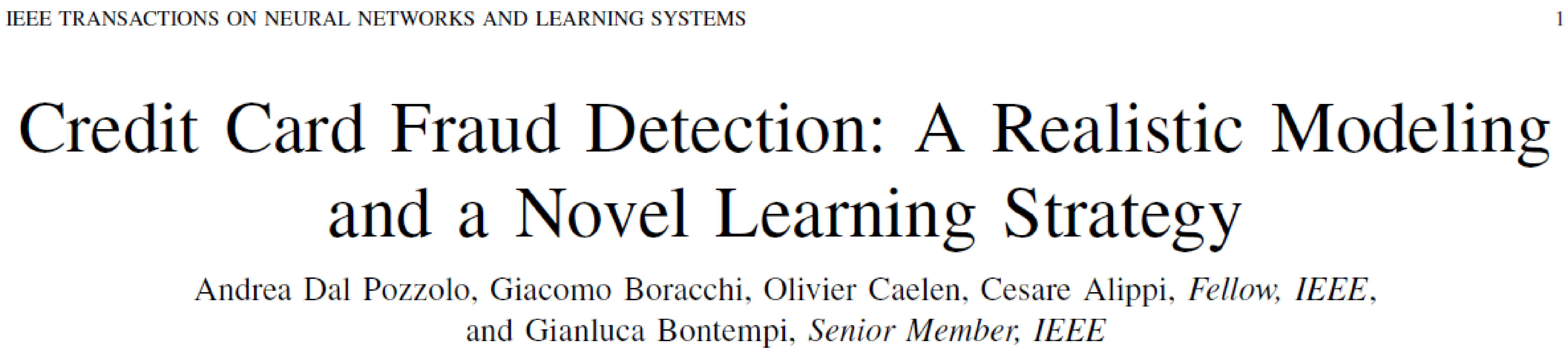
Training and incoming are either nonstationary or chosen by an adversarial

These settings are often encountered in **real-world applications on streaming data,** e.g., to detect frauds in credit card transaction.

The course provides an **overview of techniques to employ data-driven models** in these **streaming settings**

Fraud Detection

A Cool Example for Learning in NSE



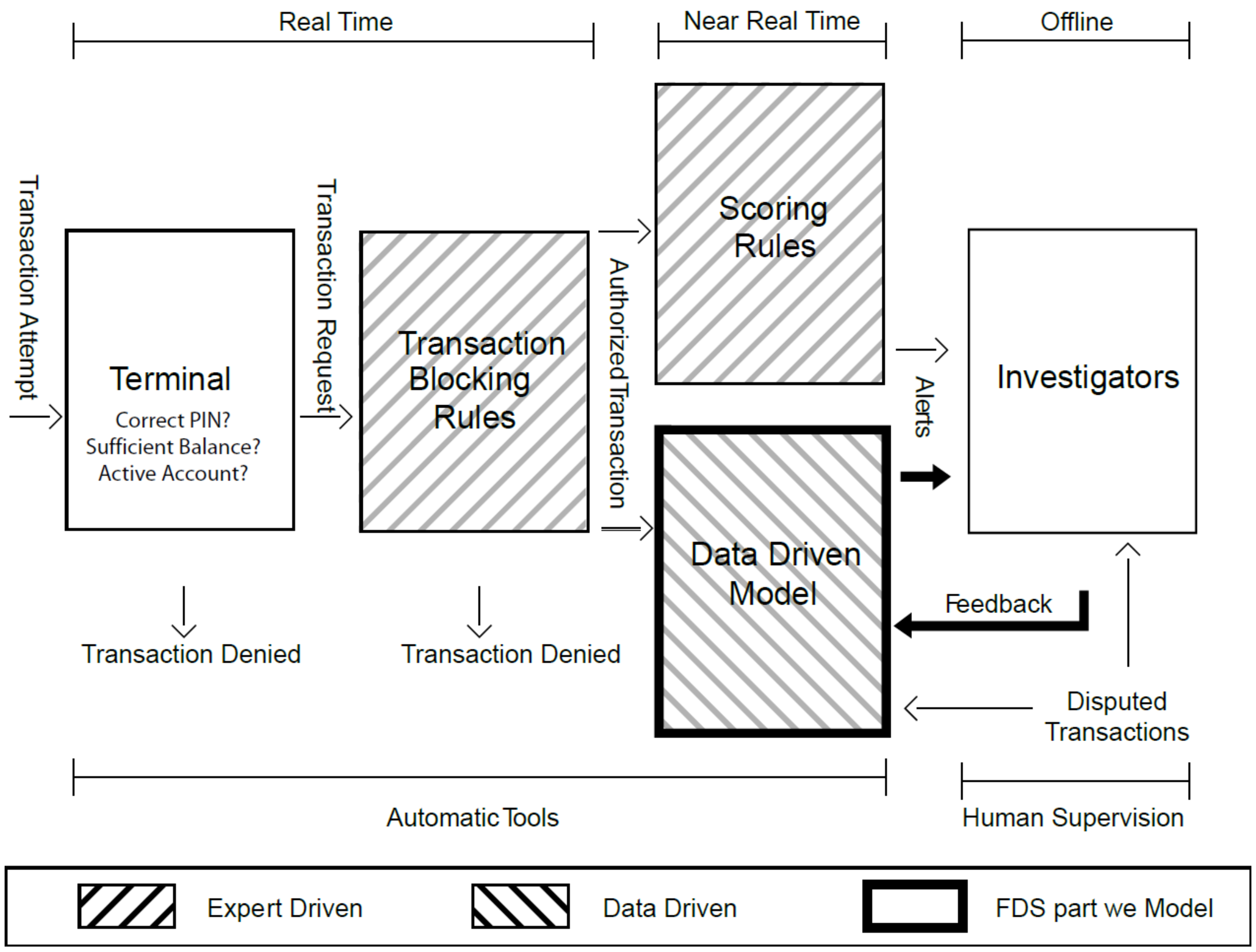
IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS 1

Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy

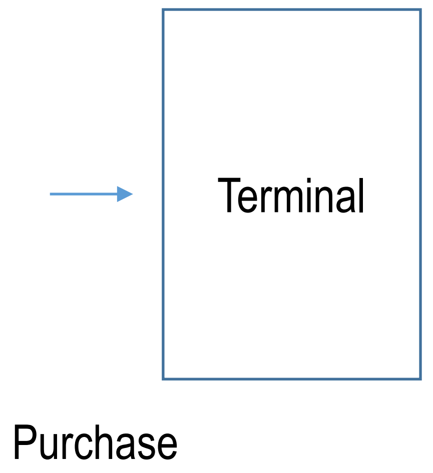
Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, *Fellow, IEEE*, and Gianluca Bontempi, *Senior Member, IEEE*

Dal Pozzolo A., Boracchi G., Caelen O., Alippi C. and Bontempi G., *"Credit Card Fraud Detection: a Realistic Modeling and a Novel Learning Strategy"* , IEEE TNNLS 2017

Fraud Detection



The Terminal



The Terminal

Acceptance checks like:

- Correct PIN
- Number of attempts
- Card status (active, blocked)
- Card balance / availability

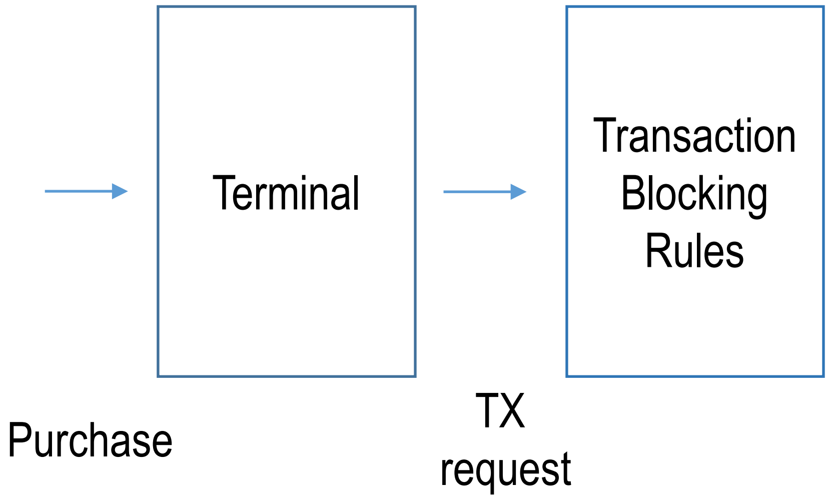
are immediately performed.

These checks are done in **real time**, and **preliminary filter** our purchases: when these checks are not satisfied, the card/transaction can be blocked.

Otherwise, a **transaction request** is entered in the system that include information of the actual purchase:

- *transaction amount, merchant id, location, transaction type, date time, …*

Blocking rules



Transaction Blocking Rules

Association rules (if-then-else statements) like*****

IF Internet transactions AND compromised website THEN deny the transaction

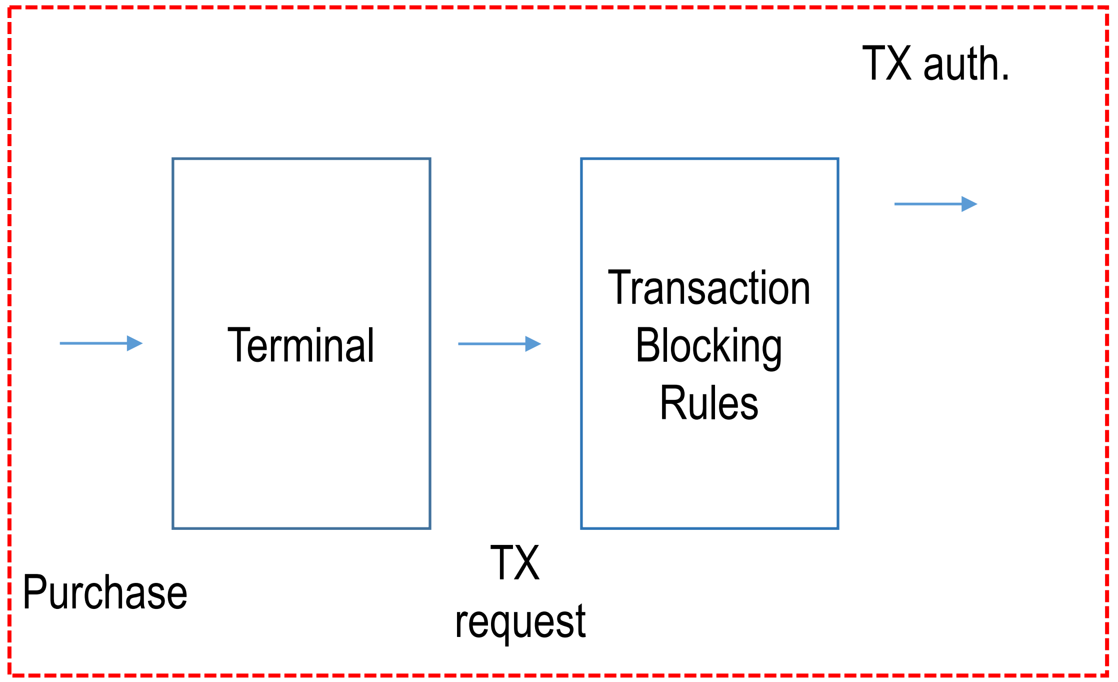
These rules:

- are **expert-driven**, designed by investigators
- involves quite simple expressions with a few data
- are easy to interpret
- have always «deny the transaction» as statement (otherwise the transaction is accepted)
- are executed in real time

All the transaction RX passing these rules are **authorized transactions** and further analyzed by the FDS

(*****) Transaction blocking rules are confidential and this is just a likely example

Near Real Time Processing



Real time

Feature Augmentation

A feature vector $\boldsymbol{x}$ is associated to each authorized transaction.

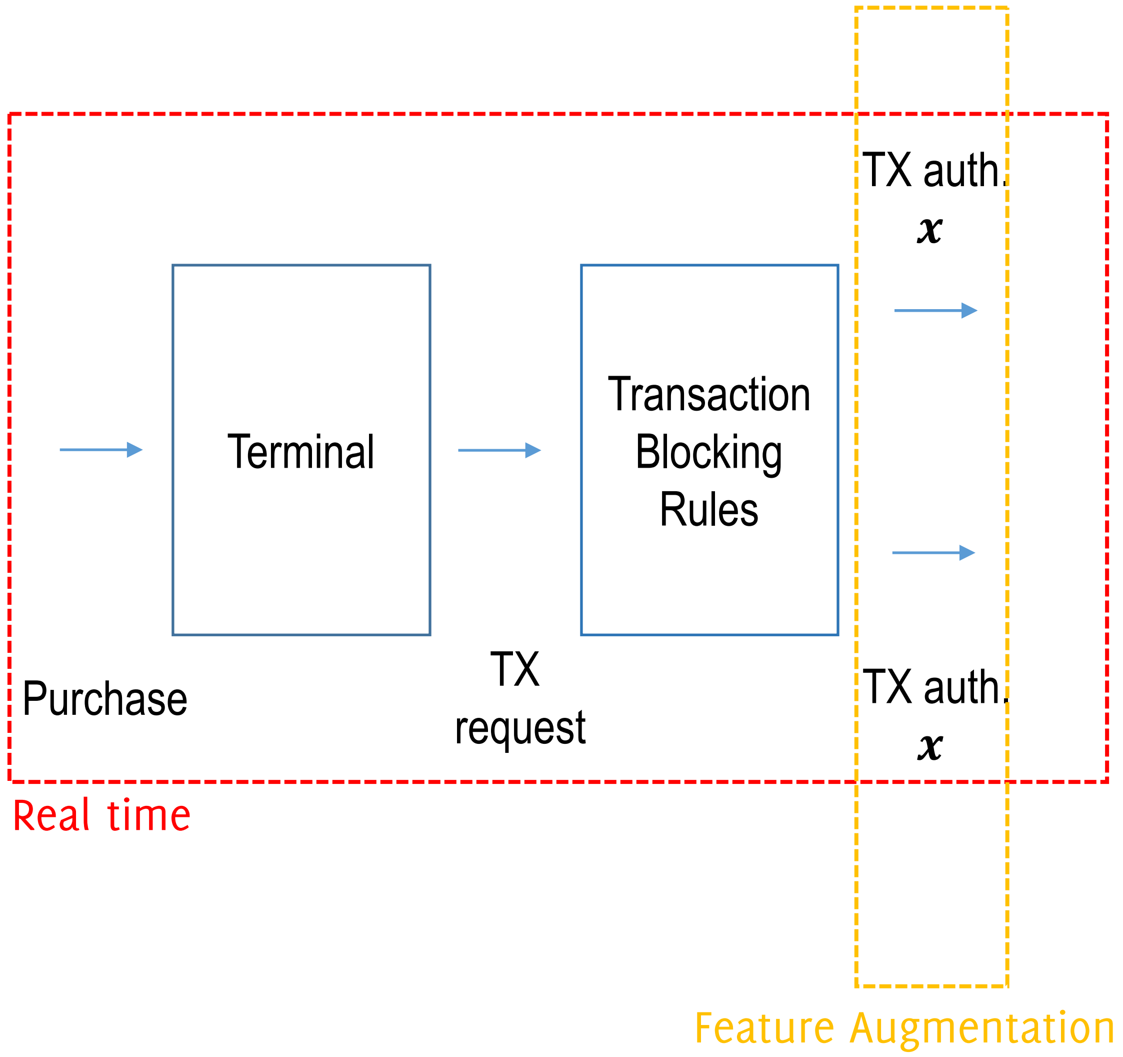
The components of $\boldsymbol{x}$ include data about the current transaction and customary shopping habits of the cardholder, e.g.:

- the average expenditure
- the average number of transactions per day
- the cardholder age
- the location of the last purchases
- …

and are **very informative** for fraud-detection purposes

Overall, about 40 features are extracted in near-real time.

Near Real Time Processing

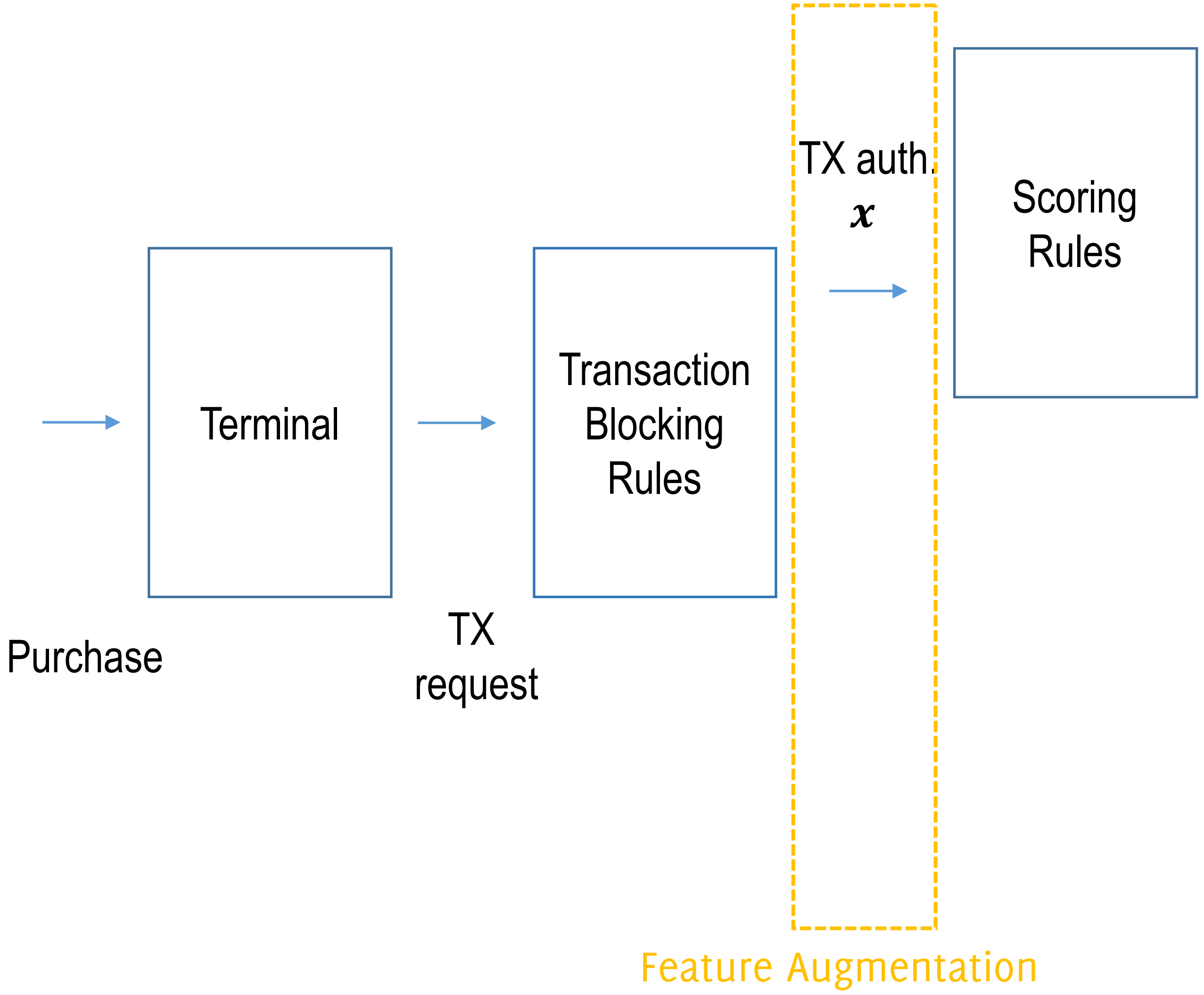


Transaction data

e.g. average amount

Augmented data

Scoring Rules



Scoring Rules

Scoring rules are if-then-else statements that:

- are being processed in near-real time
- are **expert-driven**, designed by investigators.
- Operate on augmented features (components of $\boldsymbol{x}$)
- Assign a **score**: the larger the score the riskier the transaction. The score can be seen as an estimate of the probability for $\boldsymbol{x}$ to be a fraud, according to investigator expertise.
- Feature vector receiving large scores are alerted
- Are easy to interpret and are designed by investigators

Scoring Rules

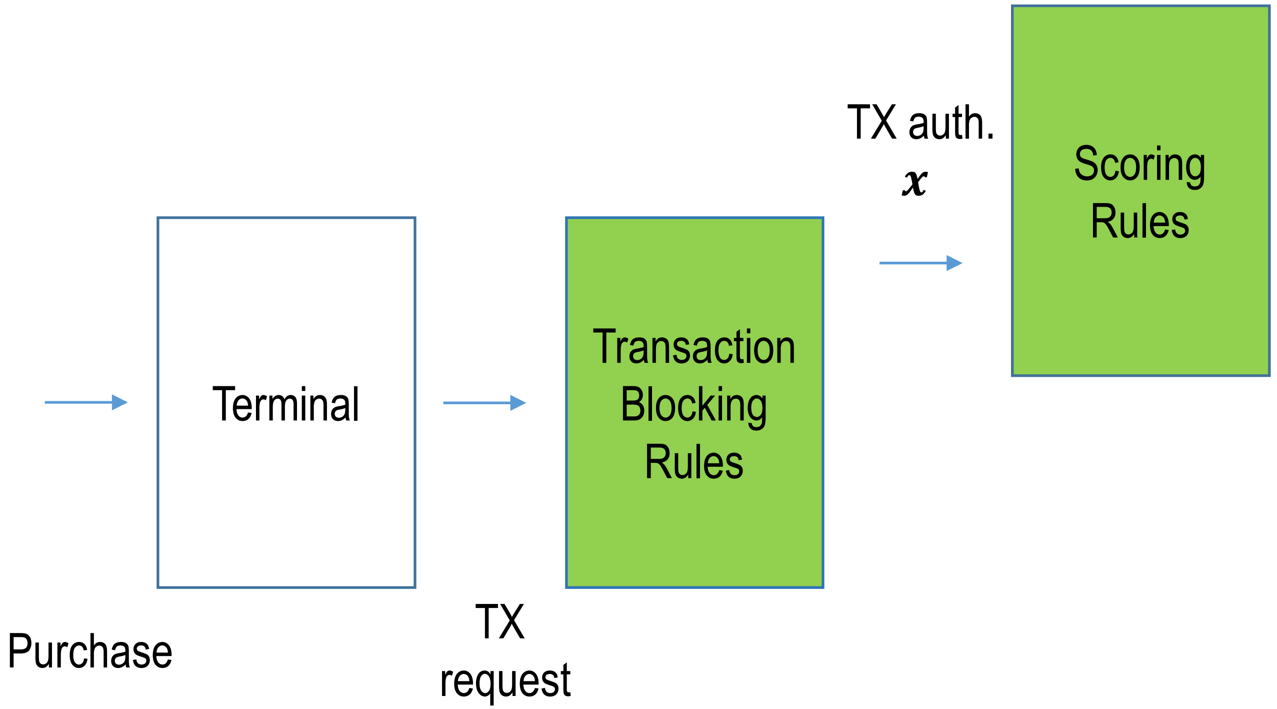
Examples* of scoring rules might be:

- *IF previous transaction in a different country AND less than 2 hours since the previous transaction, AND operation using PIN THEN fraud score = 0.95*
- *IF amount > average of transactions* $+ 3\sigma$ *AND country is a fiscal paradise AND customer travelling habits low THEN fraud score = 0.75*

(*) Scoring rules are confidential and these are just likely examples

Expert-Driven Models in fraud detection

🟩 Expert-driven



Interpretable rules
enable interaction/adjustment

Expert-Driven vs Data-Driven models

Scoring rules are an **expert-driven model**, thus:

- Can detect **well-known / reasonable** frauds
- Involve **few components** of the feature vector
- **Difficult** to **exploit correlation** among features

Expert-Driven vs Data-Driven models

Scoring rules are an **expert-driven model**, thus:

$\boldsymbol{x} =$ $K(\boldsymbol{x}) = \begin{cases} \text{fraud} \\ \text{genuine} \end{cases}$

- Can detect **well-known / reasonable** frauds
- Involve **few components** of the feature vector
- **Difficult** to **exploit correlation** among features

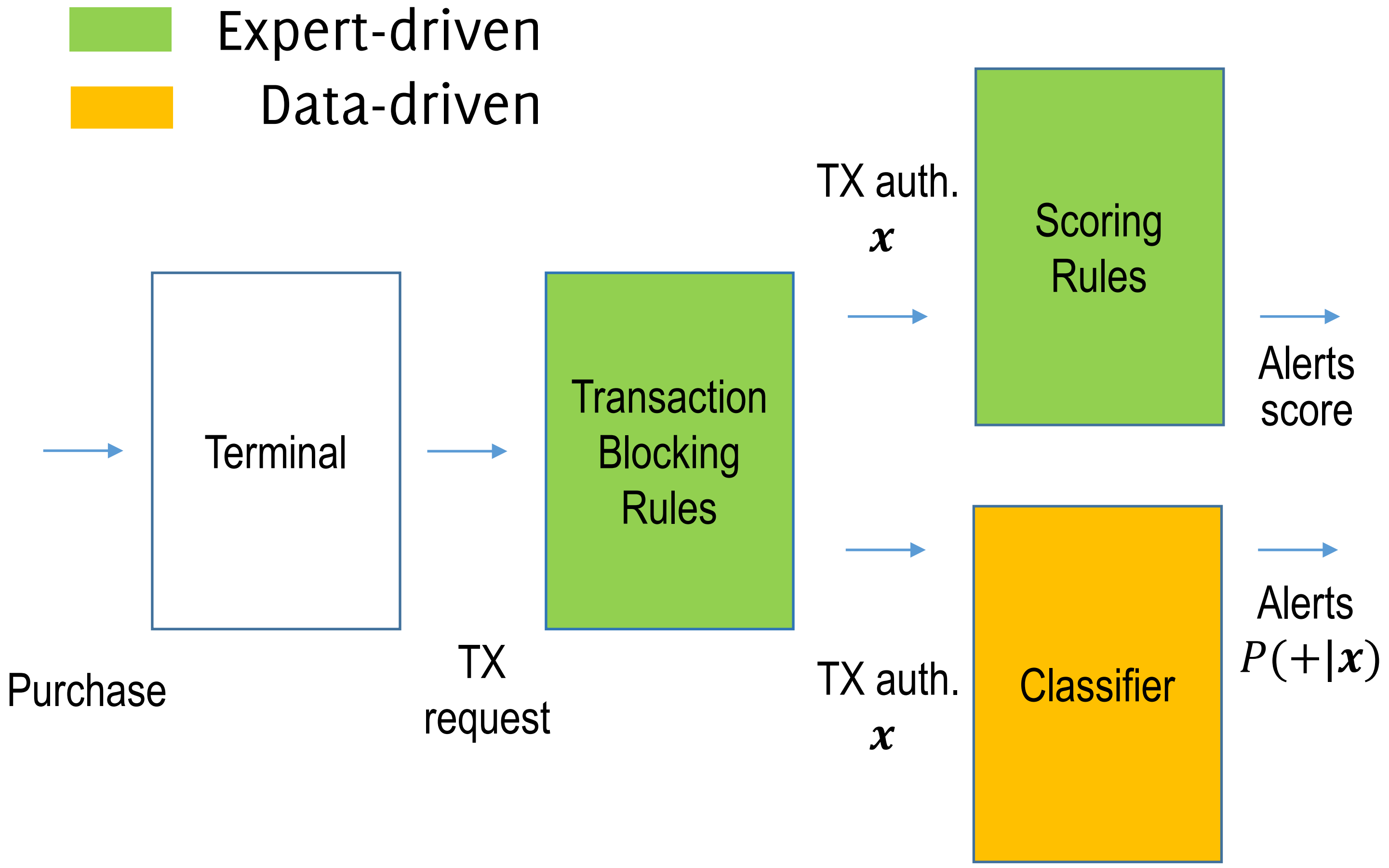
Fraudulent patterns can be directly **learned from data**, by means of a **data-driven model**.

This has the potential to:

- Simultaneously analyze **several components** of the feature vector
- Uncover **complex relations among features** that cannot be identified by investigator

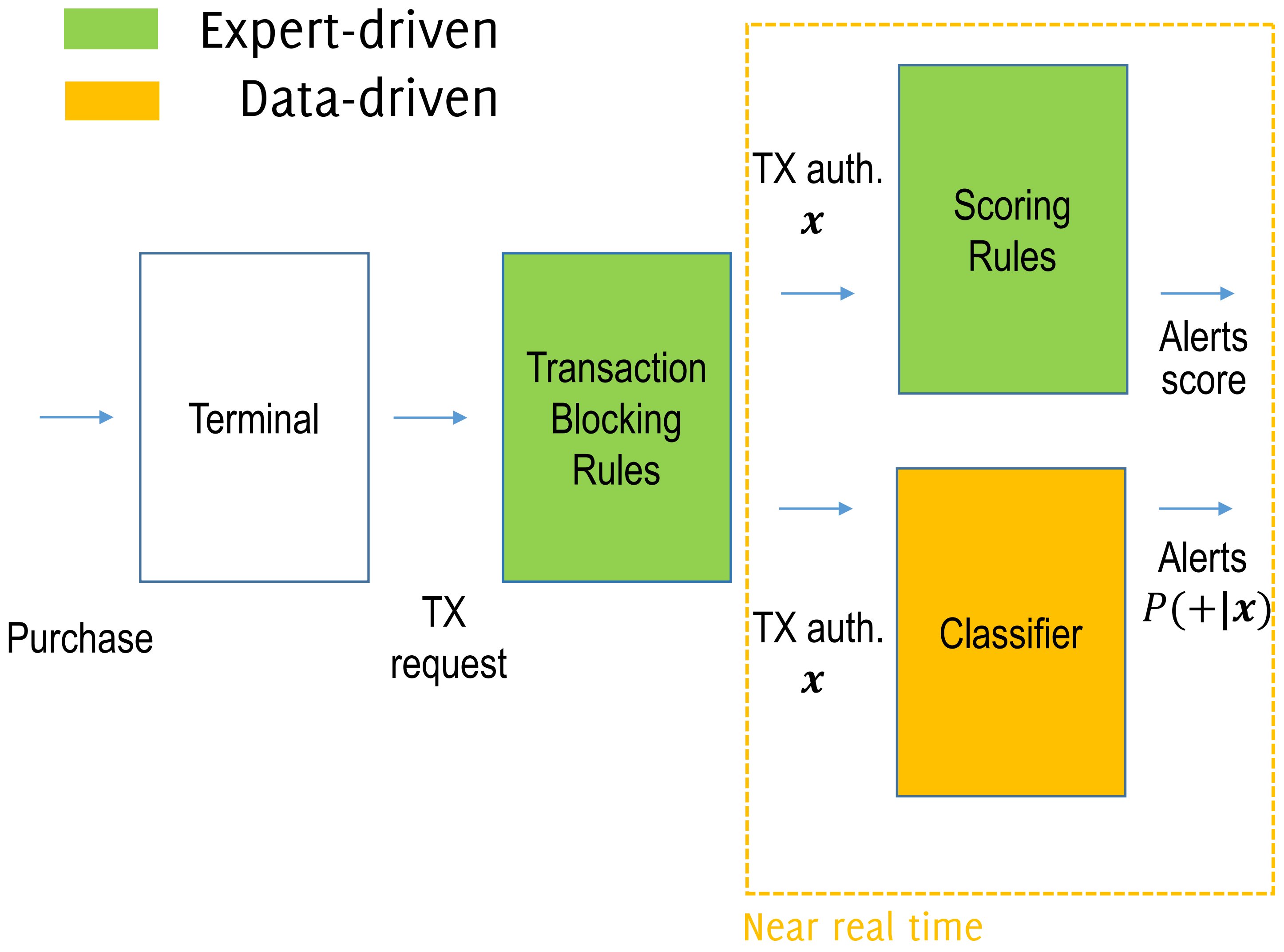
These relations can be meaningful for separating frauds from genuine transactions

Data-driven models in fraud detection



🟩 Expert-driven
🟨 Data-driven
Purchase
Terminal
TX request
Transaction Blocking Rules
TX auth. $\boldsymbol{x}$
Scoring Rules
Alerts score
TX auth. $\boldsymbol{x}$
Classifier
Alerts $P(+|\boldsymbol{x})$

Data-driven models in fraud detection



🟩 Expert-driven
🟨 Data-driven
Purchase
Terminal
TX request
Transaction Blocking Rules
TX auth. $\boldsymbol{x}$
Scoring Rules
Alerts score
TX auth. $\boldsymbol{x}$
Classifier
Alerts $P(+|\boldsymbol{x})$
Near real time

Classifiers in Fraud Detection

In practice, the classifier K then can assign a label where the label $\hat{y} \in \{+,-\}$ i.e., $\{«fraud», «genuine»\}$ to each incoming feature vector $\boldsymbol{x}$

$$\begin{tikzcd} \boldsymbol{x} \arrow[r] & \boxed{K} \arrow[r] & K(\boldsymbol{x}) \in \{+,-\} \end{tikzcd}$$

K considers transactions **labeled as ‘$+$’ as frauds**

Classifiers in Fraud Detection

It is not feasible to alert all transactions labeled as frauds.

Only few transactions that are **very likely** to be frauds can be alerted.

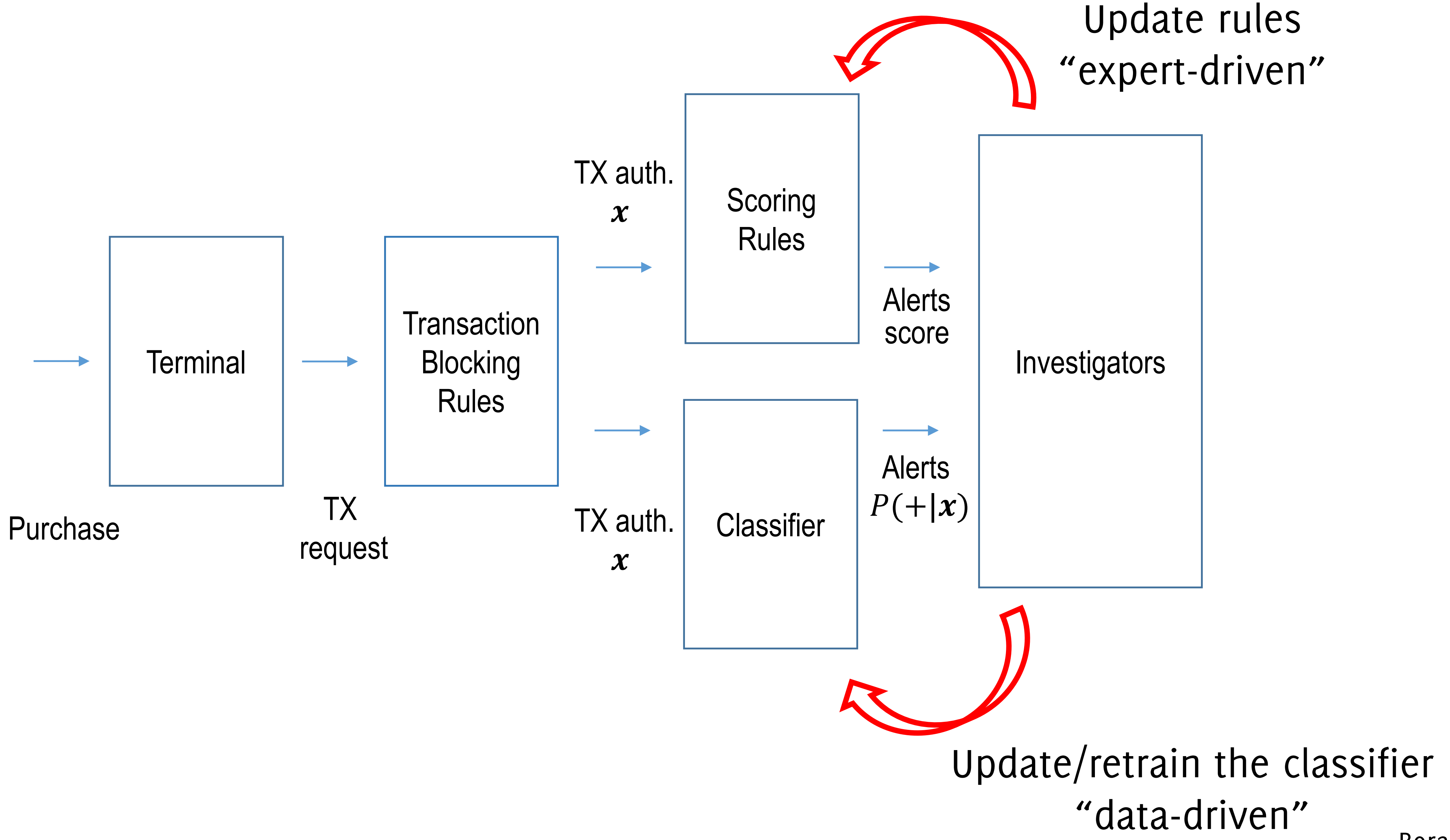
Thus, the FDS typically consider $p_K(+|\boldsymbol{x})$, **an estimate of the probability** for $\boldsymbol{x}$ to be a fraud according to K

Since this is a binary classification problem $p_K(-|\boldsymbol{x}) = 1 - p_K(+|\boldsymbol{x})$

$$\begin{tikzcd} \boldsymbol{x} \arrow[r] & \boxed{K} \arrow[r] & p_K(+|\boldsymbol{x}) \end{tikzcd}$$

and only transactions yielding $p_K(+|\boldsymbol{x}) \approx 1$ raise an alert

Investigators Provide Feedbacks



Investigators

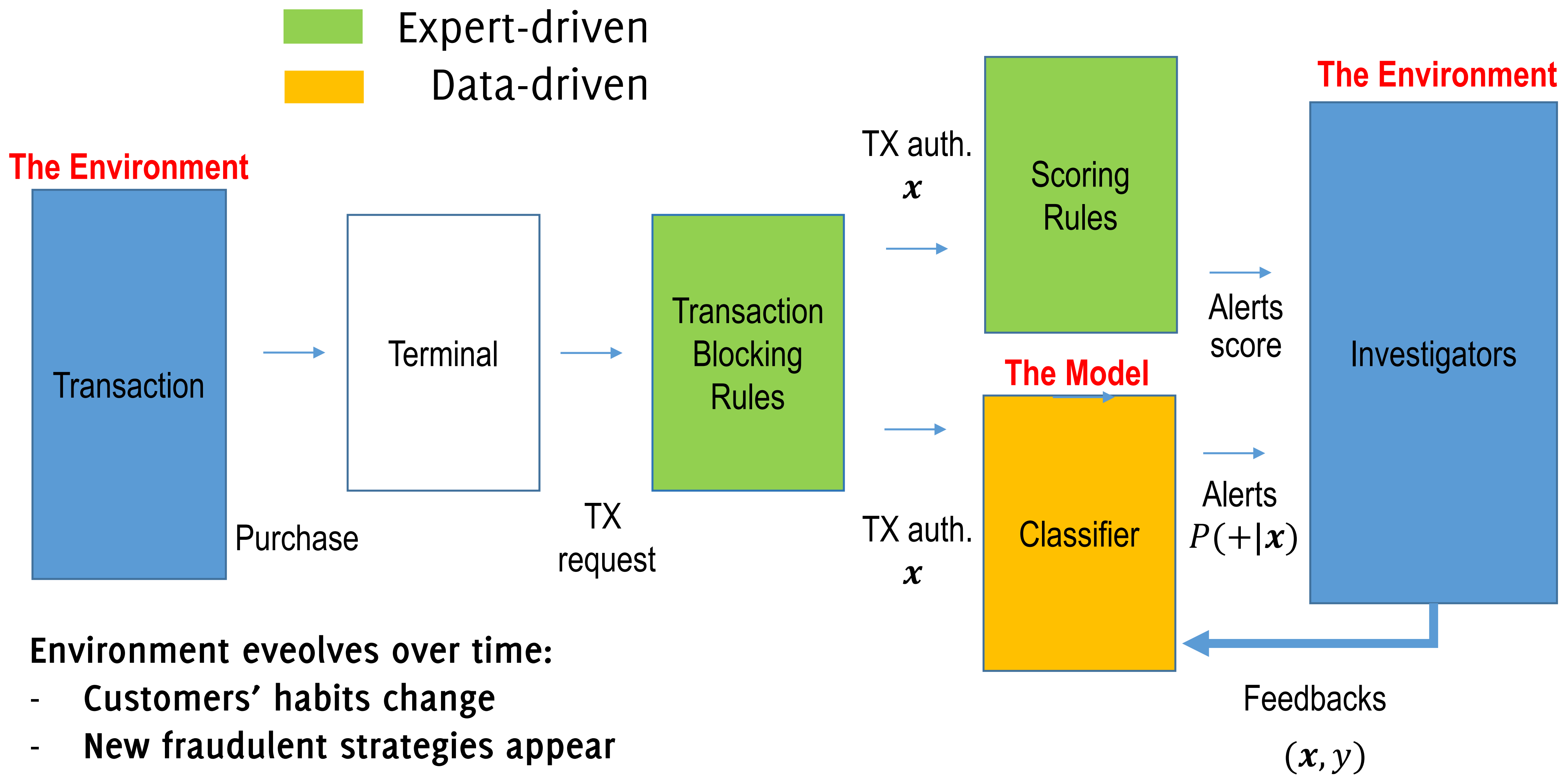
Investigators are **professionals** that are experienced in analyzing credit card transactions:

- they **design blocking/scoring rules**
- they **call cardholders** to check whether alerts correspond to frauds
- as soon as they detect a fraud, they block the card
- they annotate the **true label** of checked transactions

The labels associated to transactions comes in the form of **feedbacks** and can be used to re-train/update K

Given the limited number of investigators, the large number of transactions, the multiple sources of alerts, etc … it is important to provide **very precise alerts**

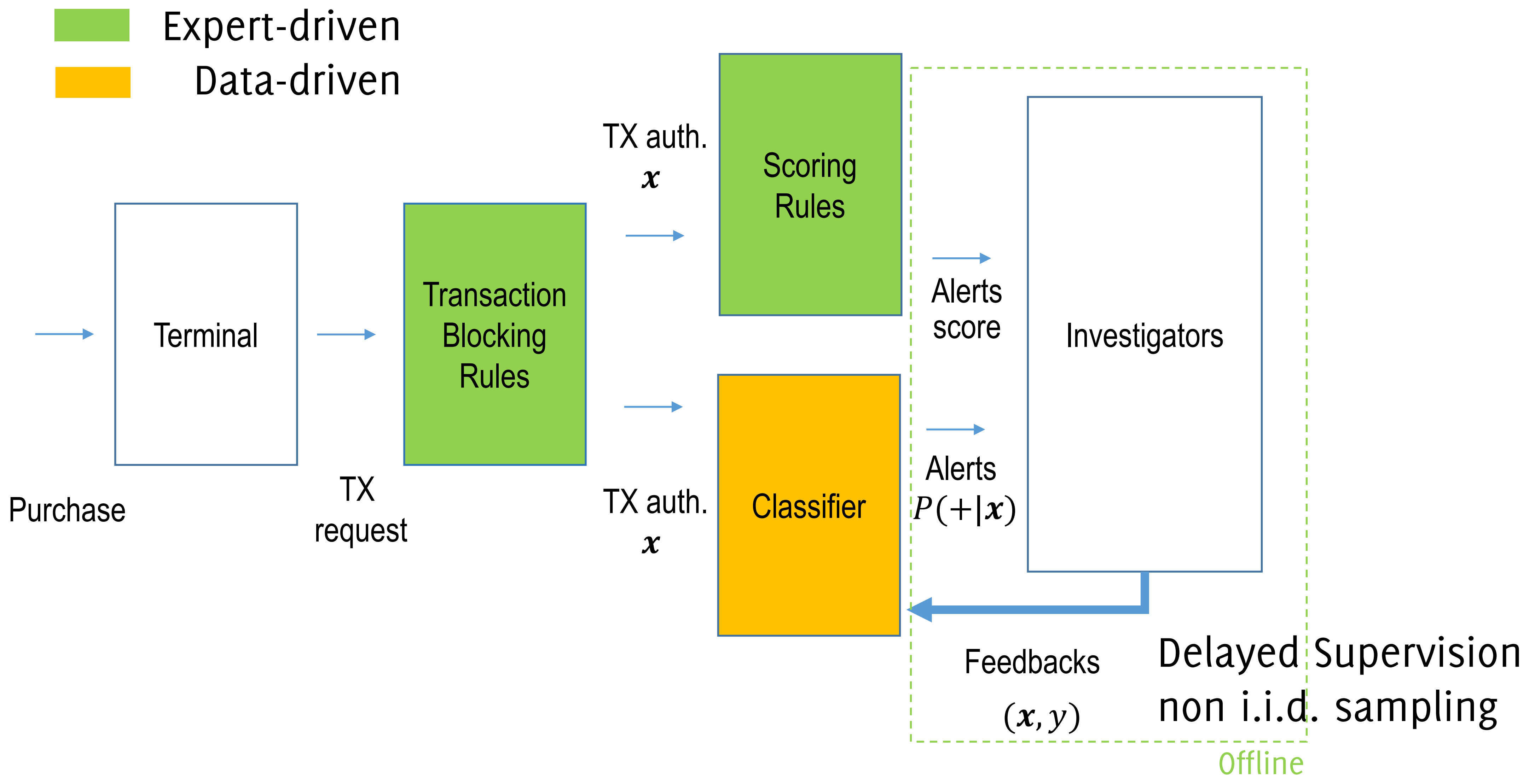
Investigators' feedback: Supervised Information



Environment eveolves over time:

- **Customers' habits change**
- **New fraudulent strategies appear**

Investigators' feedback: Supervised Information



Problem Formulation

Classification over Datastreams

Classification Over Datastreams

The problem: classification over a potentially infinitely long **stream of data**

$$X = \{\boldsymbol{x_0}, \boldsymbol{x_1}, \dots, \}$$

Data-generating process $\mathcal{X}$ generates tuples $(\boldsymbol{x}_t, y_t) \sim \phi_{\boldsymbol{x},y}$

- $\boldsymbol{x}_t$ is the observation at time t (e.g., $\boldsymbol{x}_t \in \mathbb{R}^d$)
- y_t is the associated label which is (often) unknown ($y_t \in \Lambda$)

Classification Over Datastreams

Typical assumptions:

- Inputs are independent and identically distributed (**i.i.d.**)

$$(\boldsymbol{x_t}, y_t) \sim \phi_{\boldsymbol{x},y}$$

- An **initial training set** $TR = \{(\boldsymbol{x}_0, y_0), \dots, (\boldsymbol{x}_n, y_n)\}$ is provided for learning K
- TR contains data generated in **stationary conditions**

The classifier is trained (i.e. its parameters are estimated) by optimizing some loss function (e.g. binary cross-entropy, hinge loss, …) over TR.

A **stationary condition of $\mathcal{X}$** is denoted as **concept.**

Classification error

A classifier estimates for each input $\boldsymbol{x}$ a label $\hat{y}$ **(*)**

$$\hat{y} = K(\boldsymbol{x})$$

And – hopefully – it often happens that $\hat{y} = y$.

Here, we consider the **classification error** to measure how good a learned model K matches the distribution $\phi_{\boldsymbol{x},y}$, namely

$$p = \#\{\widehat{y_i} \neq y_i, i \in R\}$$

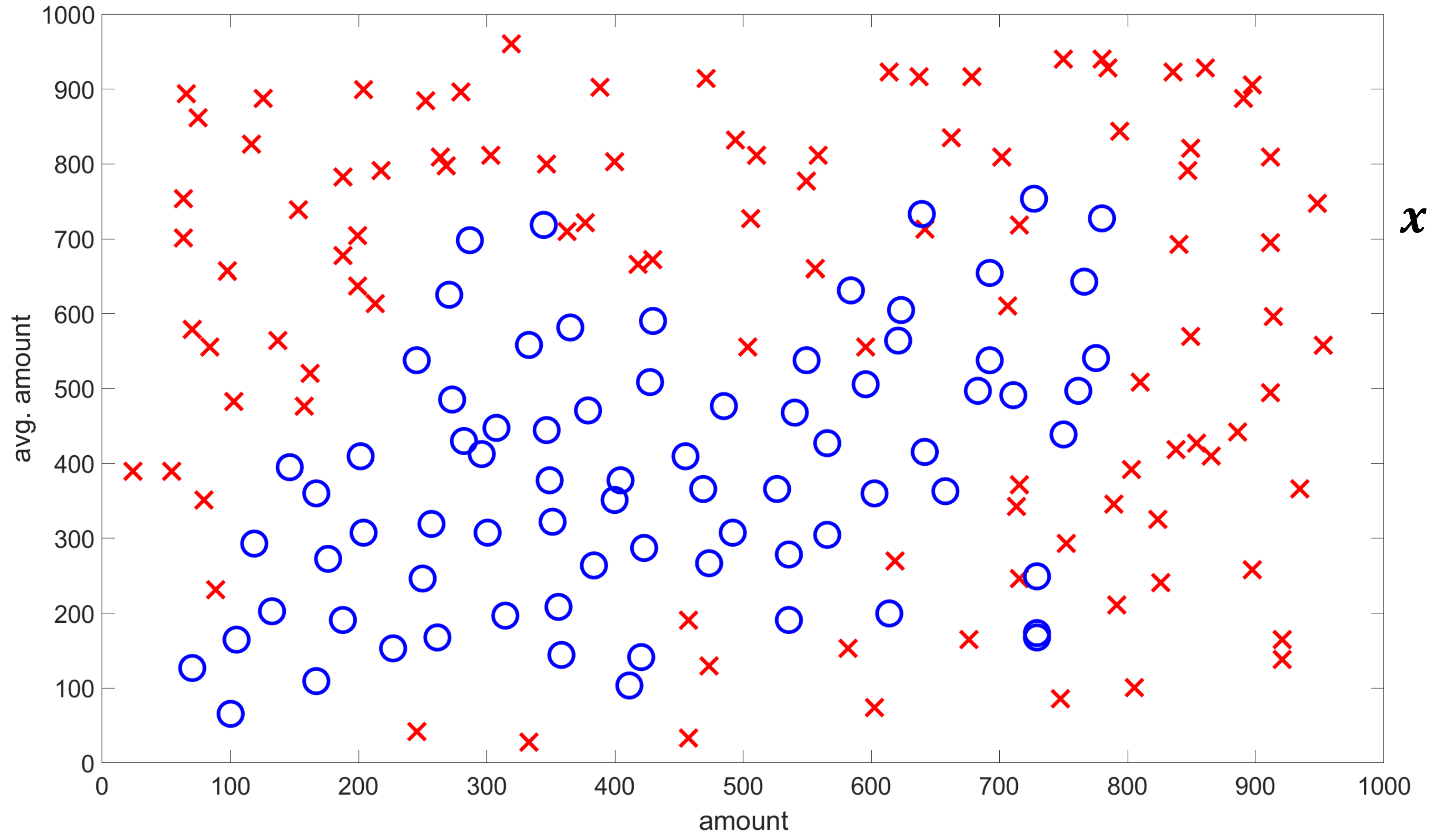
being R «a reference set» for assessing the error

(*) Classifiers typically return the posterior probability of each class

Training the Classifier

$$\begin{tikzcd}
\mathcal{X} \arrow[d] & \text{Data generating process } \mathcal{X} \sim \phi_{x,y} & \\
TR = \left\{(\boldsymbol{x}, y)_i, i = 1, \dots, N, \ (\boldsymbol{x}, y)_i \sim \phi_{x,y}\right\} \arrow[dr] & & \\
& \boxed{\text{Training}} \arrow[r] & K
\end{tikzcd}$$

Training Set



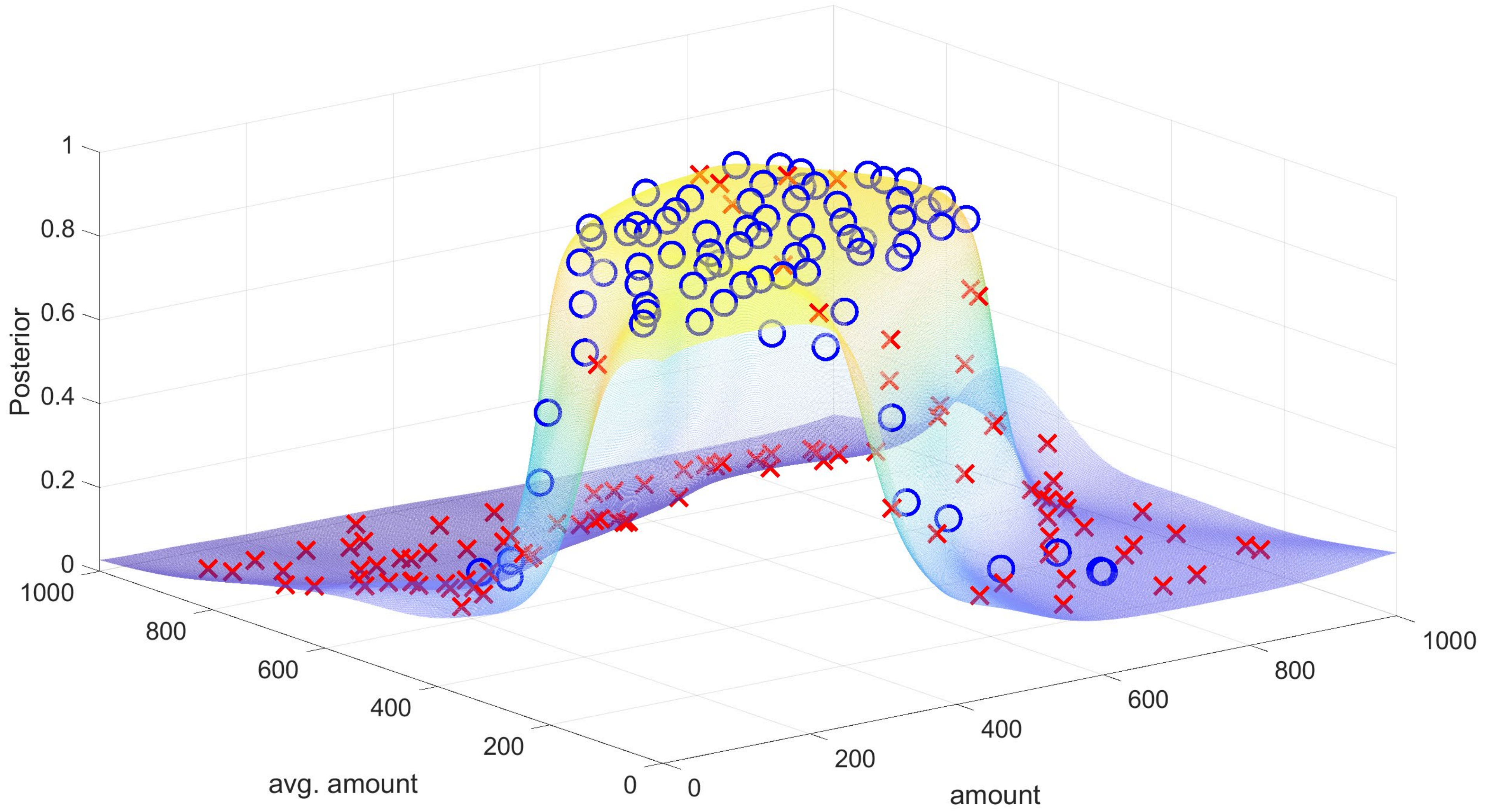
× fraud

○ genuine

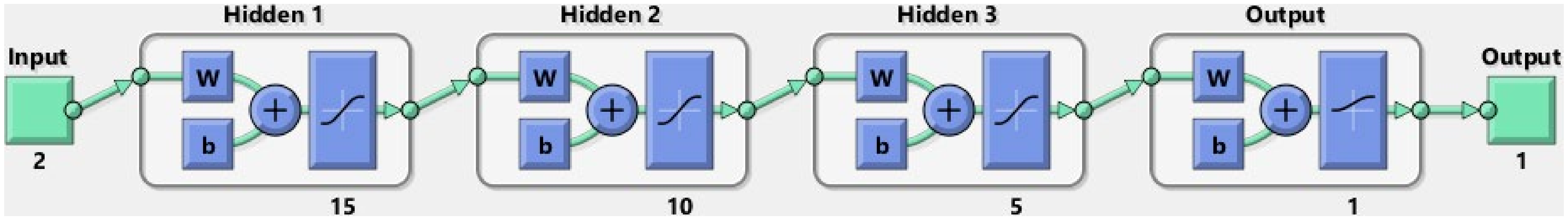
$$\boldsymbol{x} = \begin{bmatrix} \text{avg. month amount} \\ \text{transaction amount} \end{bmatrix}$$

Boracchi

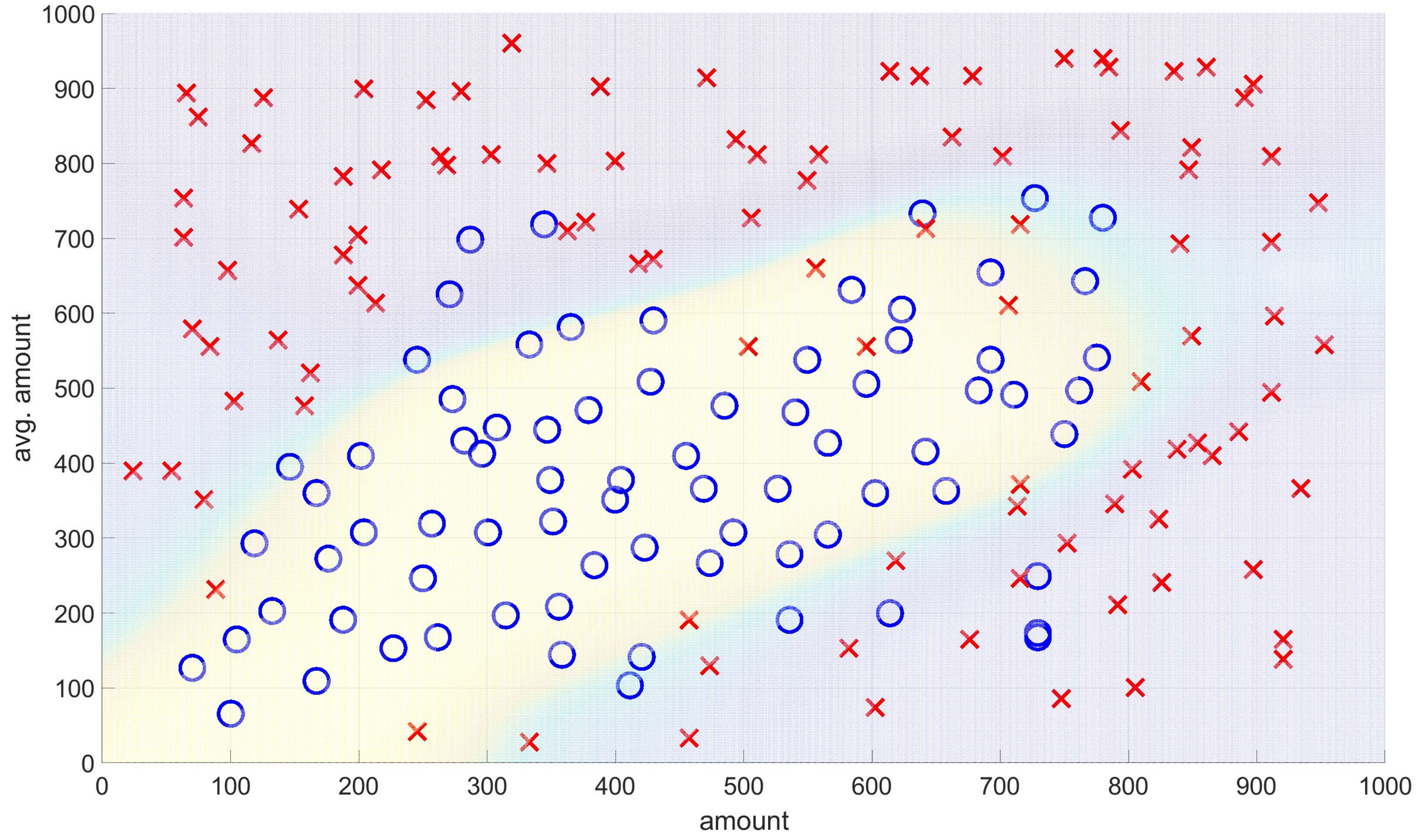
The output of the classifier



Btw… that was a Neural Network

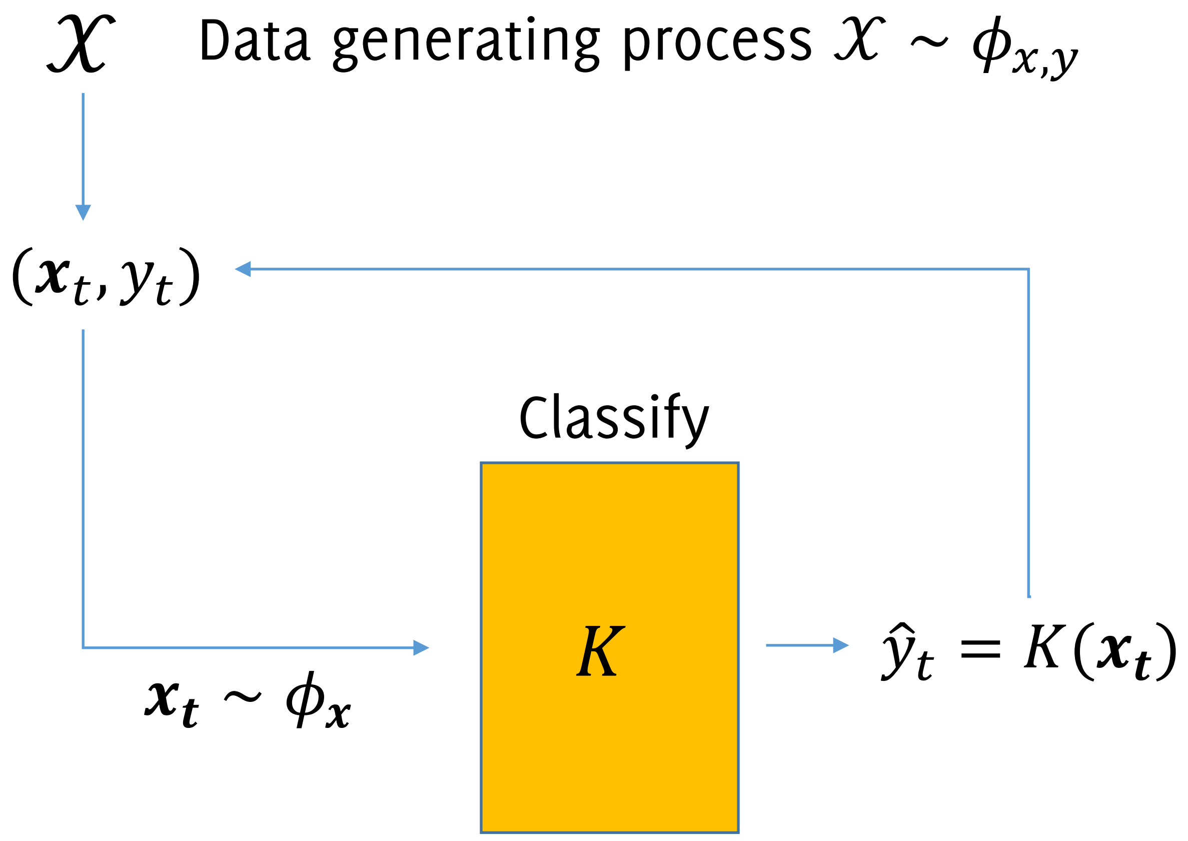


The output of the classifier



Boracchi

Classification (Inference)


$\mathcal{X}$
Data generating process $\mathcal{X} \sim \phi_{x,y}$
$(\boldsymbol{x}_t, y_t)$
Classify
K
$\boldsymbol{x_t} \sim \phi_{\boldsymbol{x}}$
$\hat{y}_t = K(\boldsymbol{x_t})$

Supervised Information (performance assessment)

Gather all the supervised information $\{(\boldsymbol{x_t}, y_t) \sim \phi_{\boldsymbol{x},y}\}$

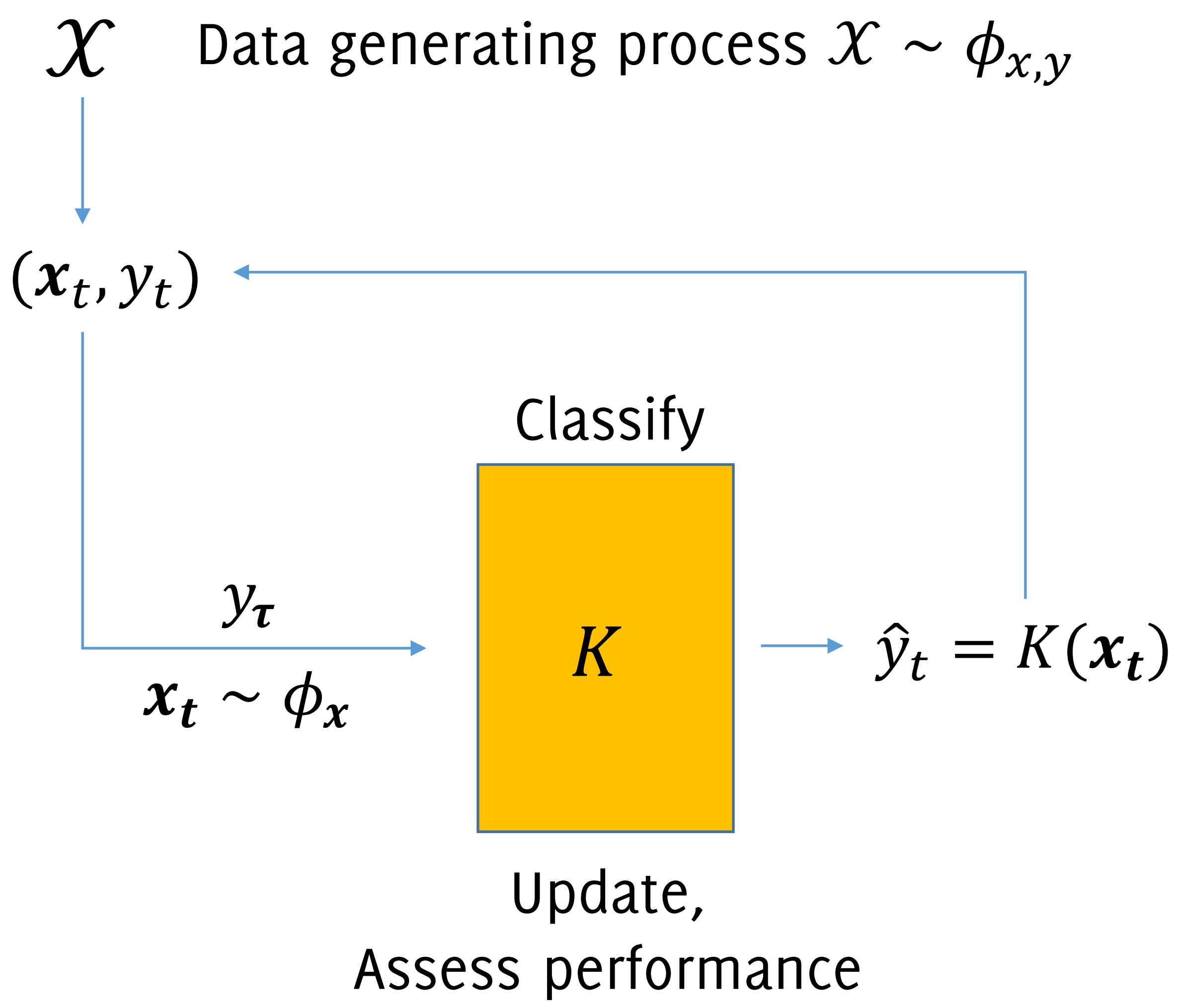
These are very useful for:

- assessing performance of K

$$p(T) = \frac{1}{T}\sum_t e_t$$

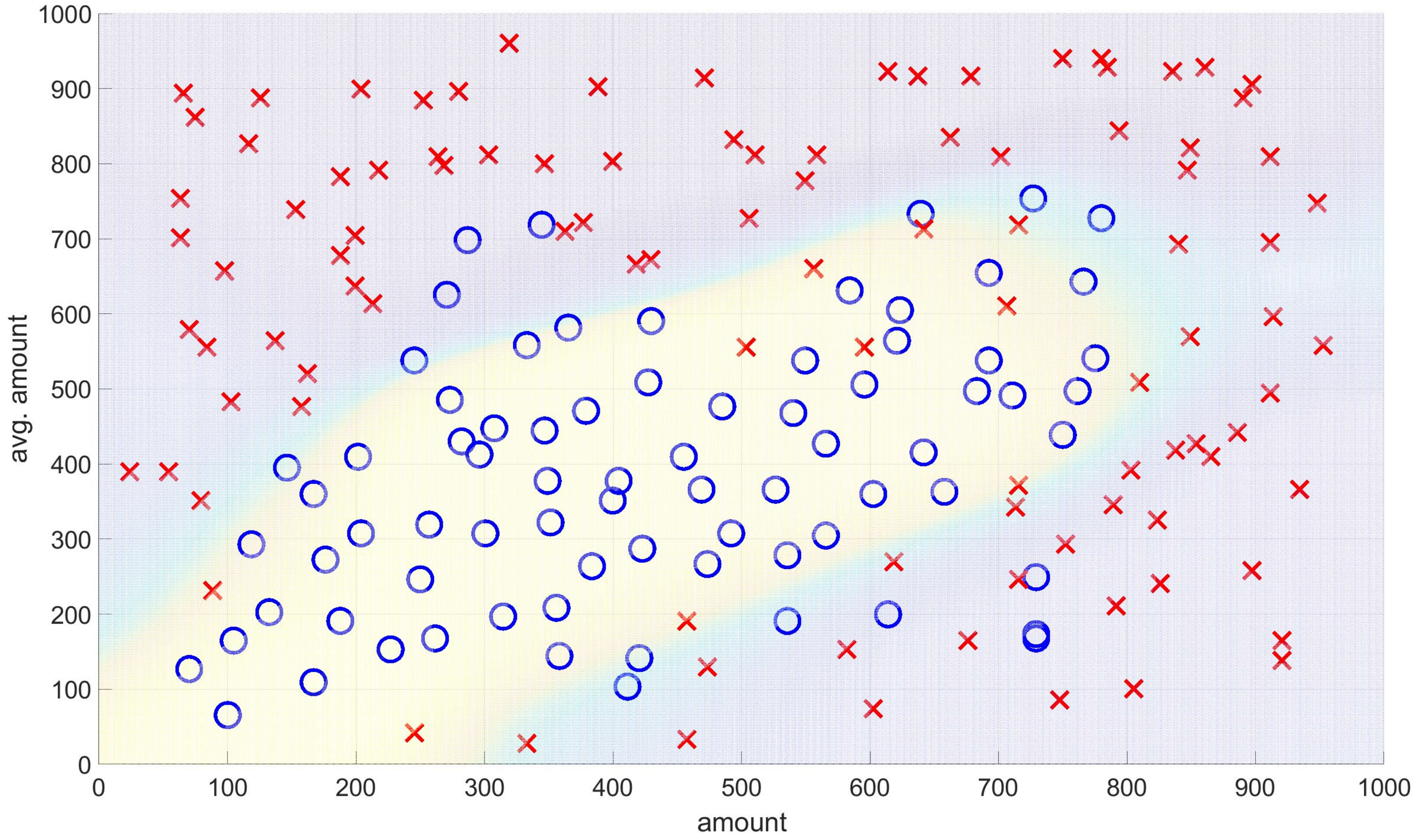
$$\text{where } \; e_t = \begin{cases} 0, & \text{if } \hat{y}_t = y_t \\ 1, & \text{if } \hat{y}_t \neq y_t \end{cases}$$

- updating K



$\mathcal{X}$
Data generating process $\mathcal{X} \sim \phi_{x,y}$
$(\boldsymbol{x}_t, y_t)$
$y_{\boldsymbol{\tau}}$
$\boldsymbol{x_t} \sim \phi_{\boldsymbol{x}}$
Classify
K
$\hat{y}_t = K(\boldsymbol{x_t})$
Update,
Assess performance

The output of the classifier



Learning in Nonstationary (Streaming) Environments

The Problem Formulation

Concept Drift

Unfortunately, in **the real world**, datastream $\mathcal{X}$ might **change unpredictably** during operation.

The data generating process is then modeled as:

$$(\boldsymbol{x}_t, y_t) \sim \phi_t(\boldsymbol{x}, y)$$

We say that **concept drift** occurs at time t if

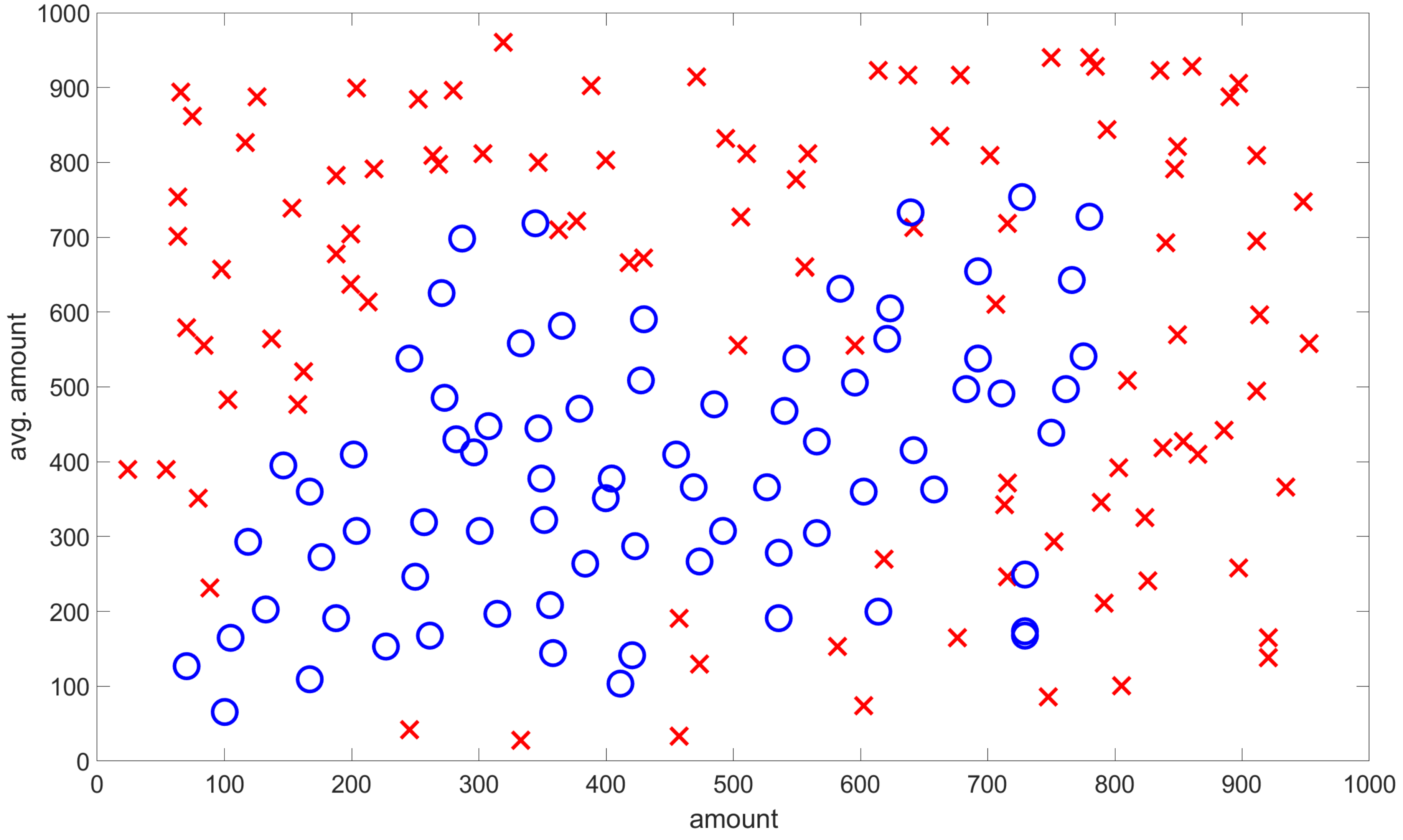
$$\phi_t(\boldsymbol{x}, y) \neq \phi_{t+1}(\boldsymbol{x}, y)$$

We also say $\mathcal{X}$ becomes **nonstationary**.

TR is always assumed i.i.d.

After the change (e.g. 10 days), data are no more i.i.d. because data are not identically distributed after

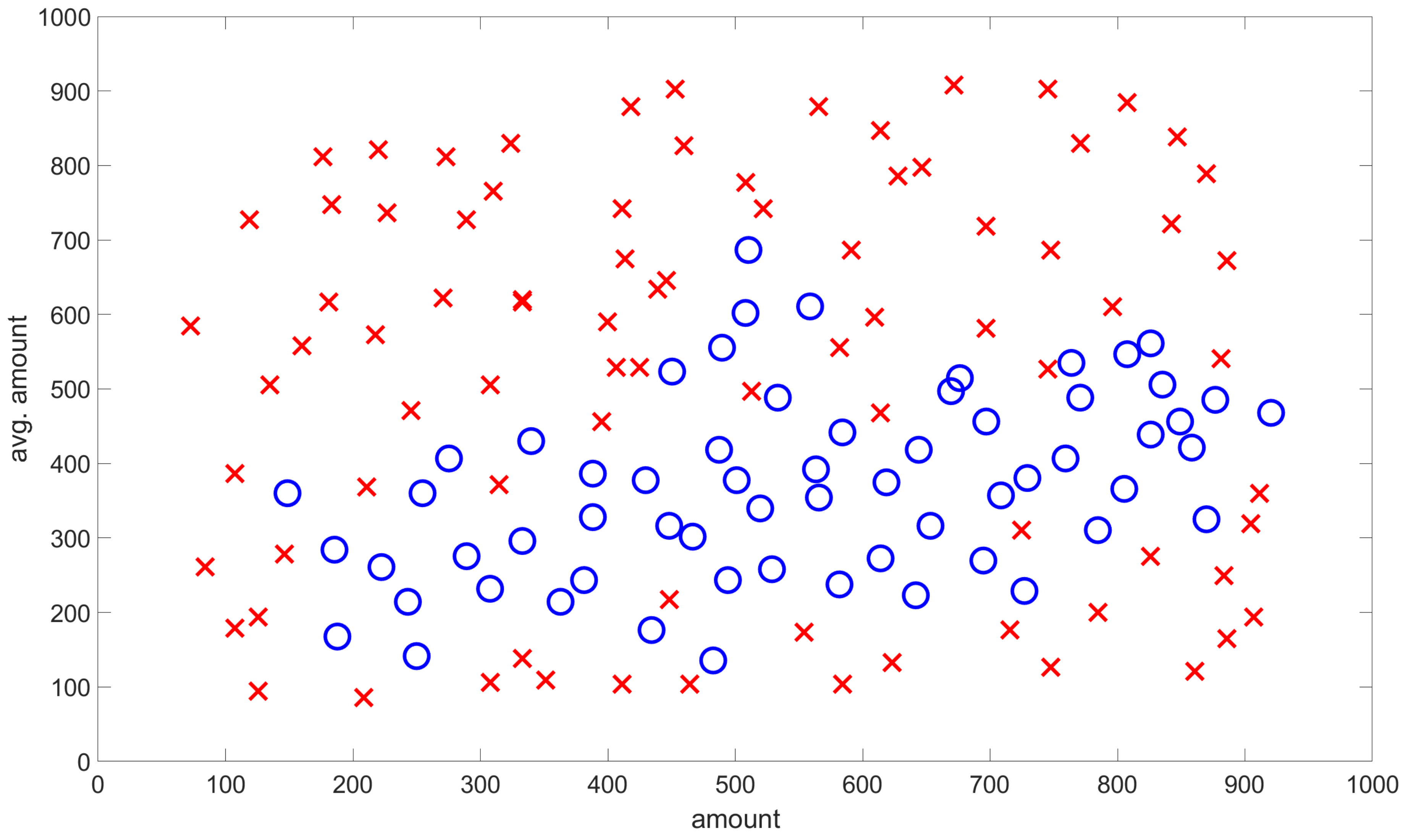
Distribution Changes



$\phi_{x,y}^{0}$

Boracchi

Distribution Changes

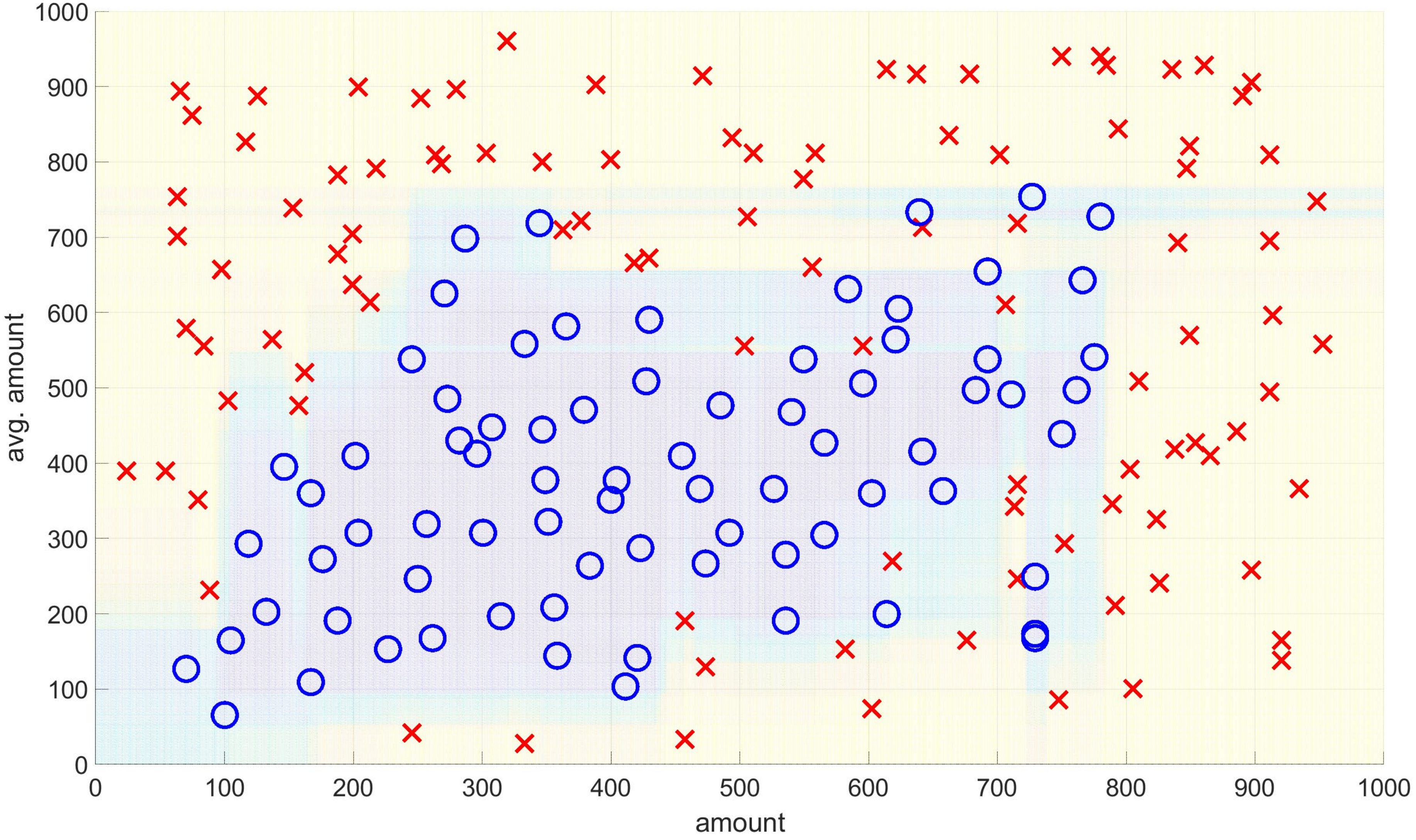


$\phi^1_{x,y}$

Boracchi

What happens when $\phi^0_{x,y} \rightarrow \phi^1_{x,y}$?

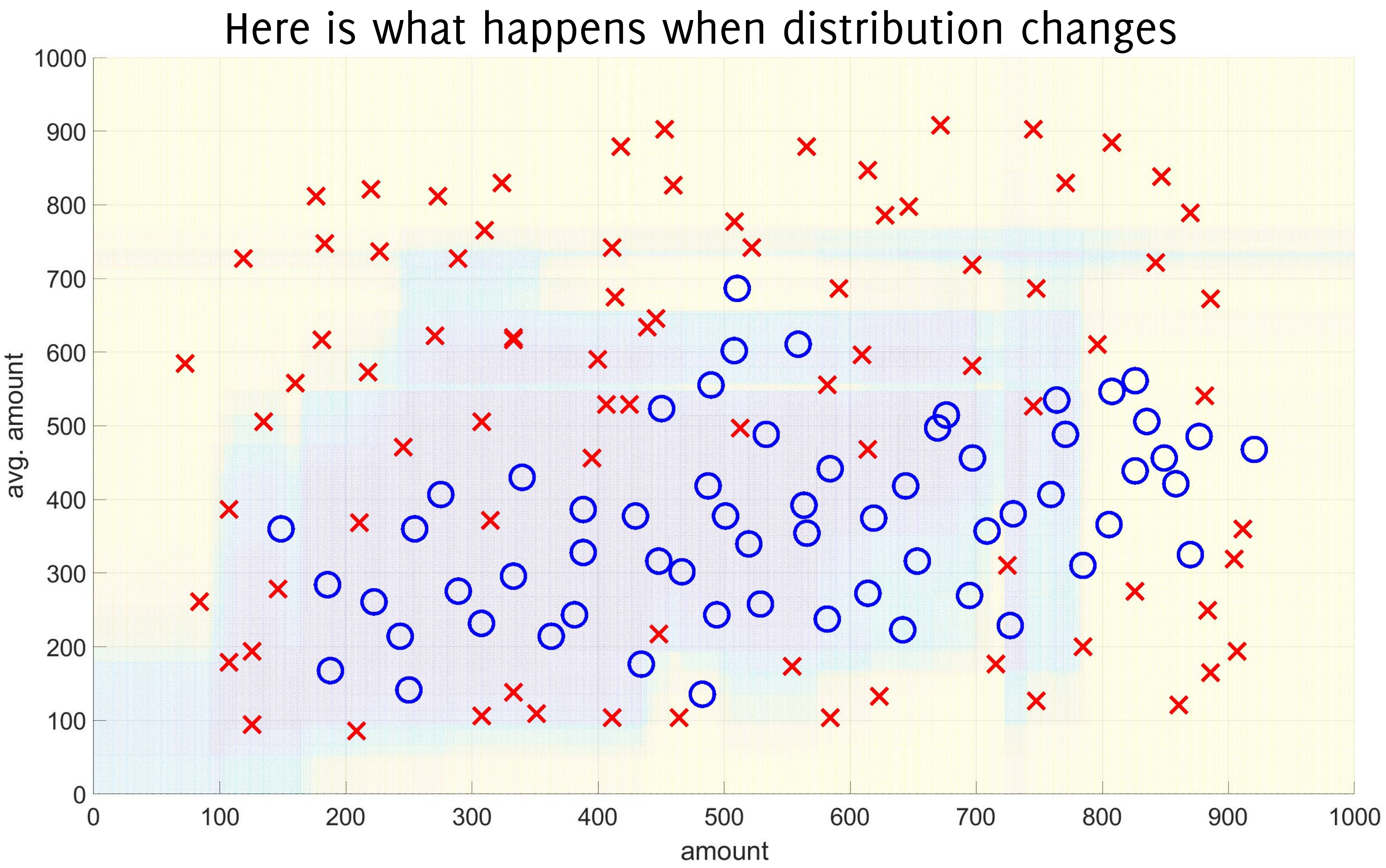
Classifier output over training data



$\phi^0_{x,y}$

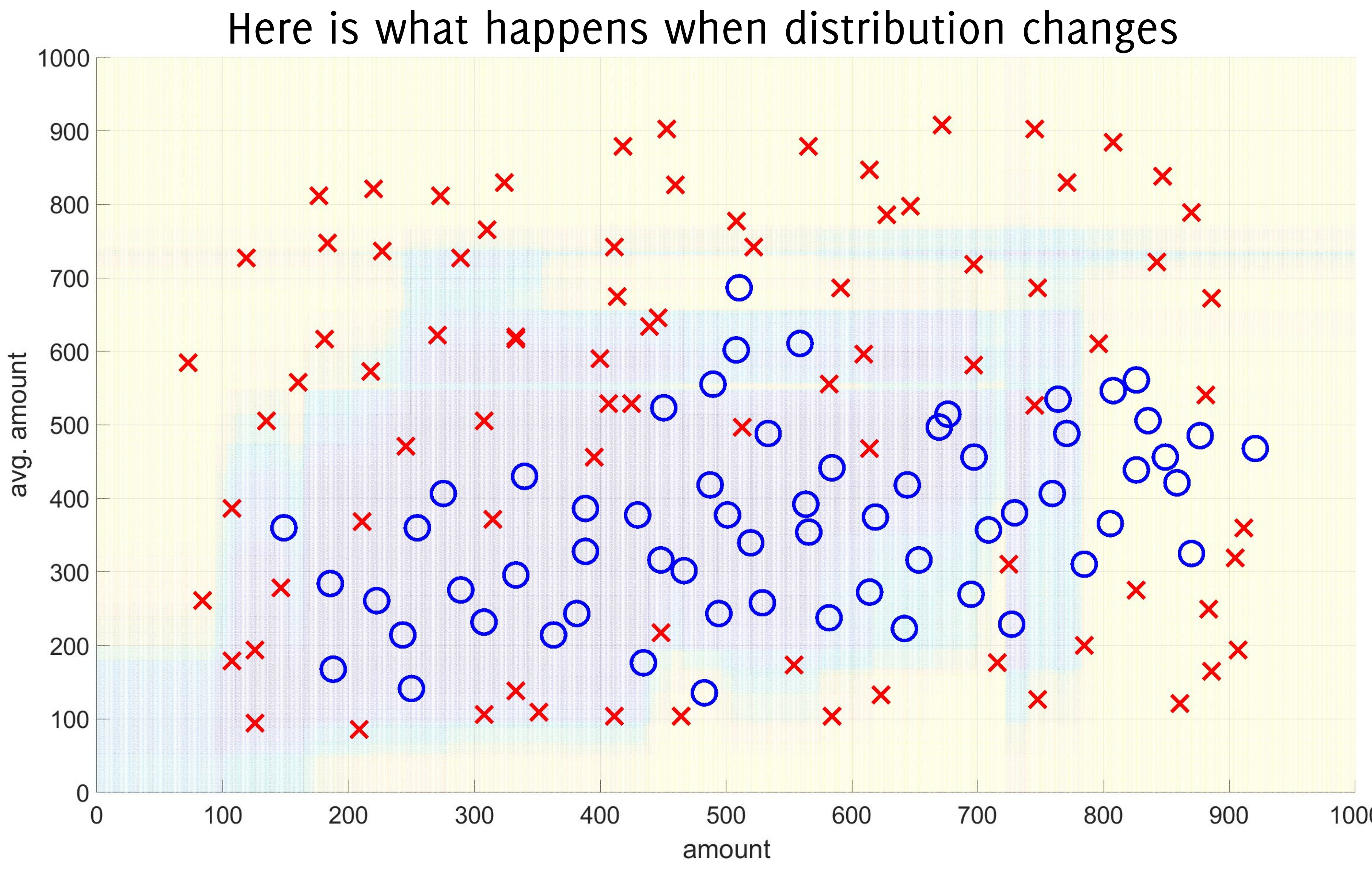
Boracchi

What happens when $\phi^0_{\boldsymbol{x},y} \rightarrow \phi^1_{\boldsymbol{x},y}$?



$\phi^1_{\boldsymbol{x},y}$

What happens when $\phi_{x,y}^0 \to \phi_{x,y}^1$?

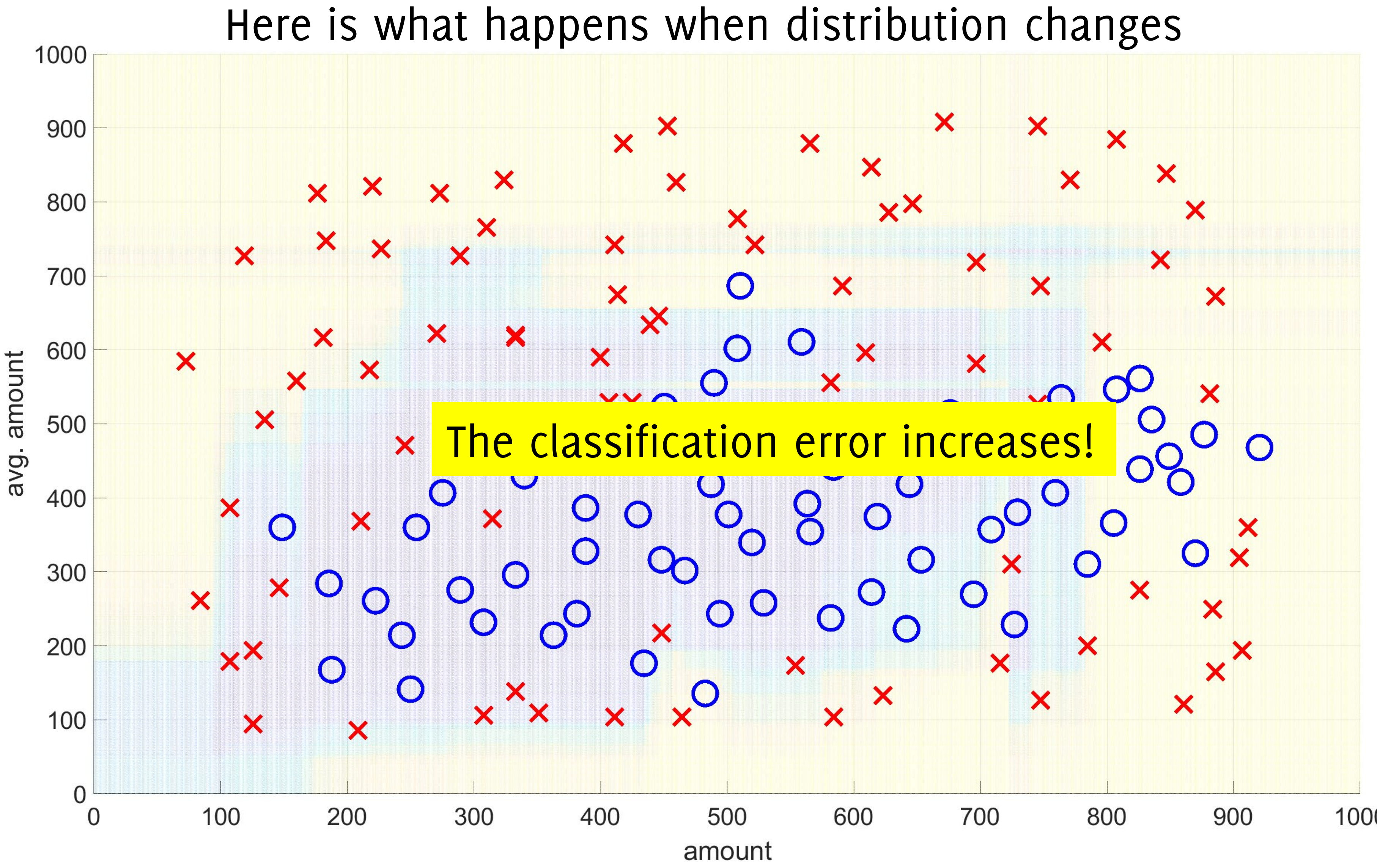


$\phi_{x,y}^1$

Classification errors

Boracchi

What happens when $\phi^0_{x,y} \to \phi^1_{x,y}$?



The classification error increases!

$\phi^1_{x,y}$

Boracchi

Problem formulation learning in NSE

The task: learn an **adaptive classifier** K_t to predict labels

$$\hat{y}_t = K_t(\boldsymbol{x}_t)$$

in an **online manner** having a low **classification error** over time:

$$\frac{1}{T}\sum_{t=1}^{T} e_t \text{ , where } e_t = \begin{cases} 0, & \text{if } \hat{y}_t = y_t \\ 1, & \text{if } \hat{y}_t \neq y_t \end{cases}$$

This classifier should also operate when the distribution generating the input data changes.

- **Measuring and monitoring the classification error**
- **Updating the classifier**

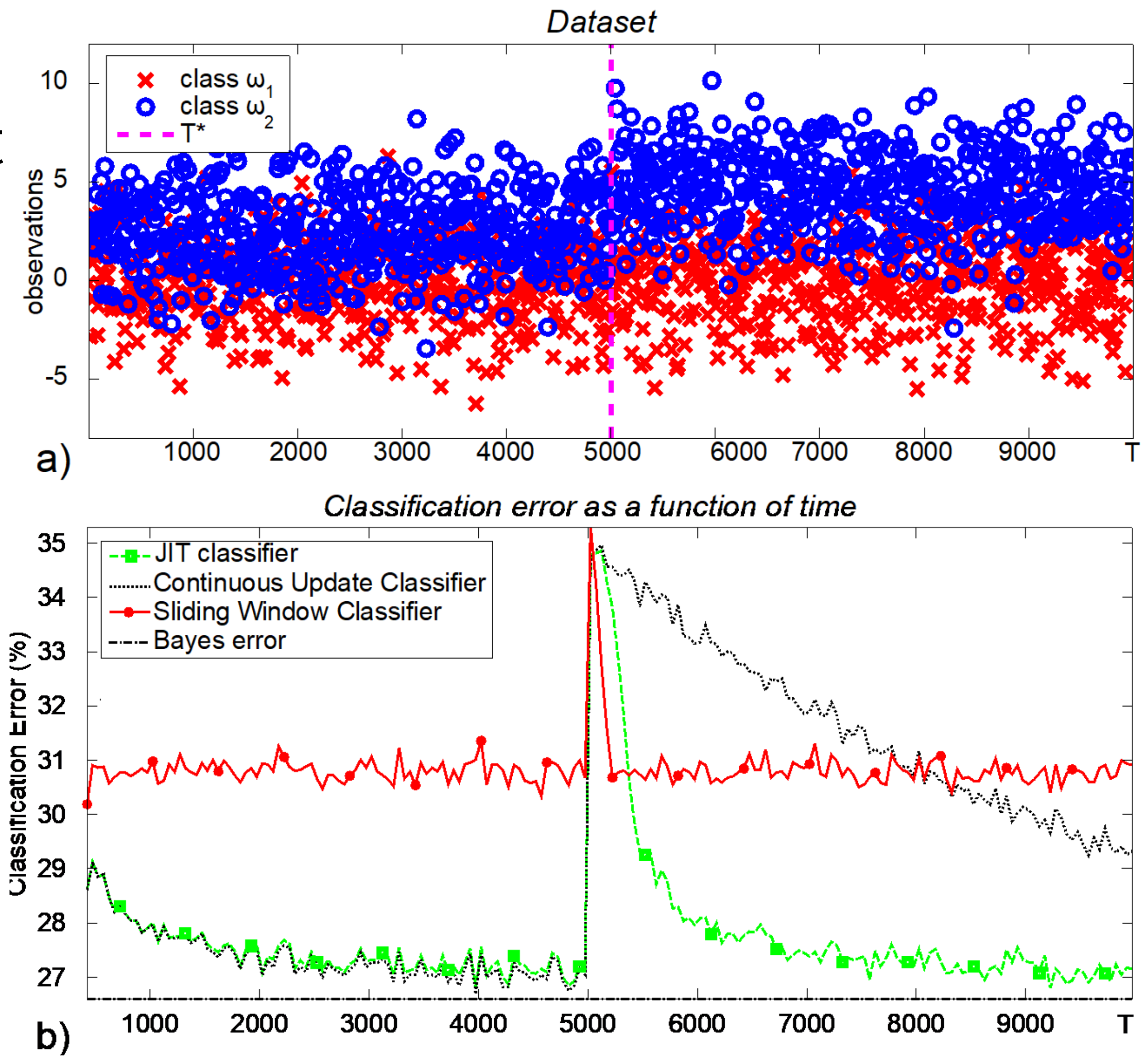
Adaptation

Do we Really Need Smart Adaptation Strategies?

Simple Adaptation Strategies

Consider two simple adaptation strategies and a simple concept drift

- **Incremental:** continuously update K_t using all supervised couples
- **Sliding Window:** Train K_t using only the last δ supervised couples

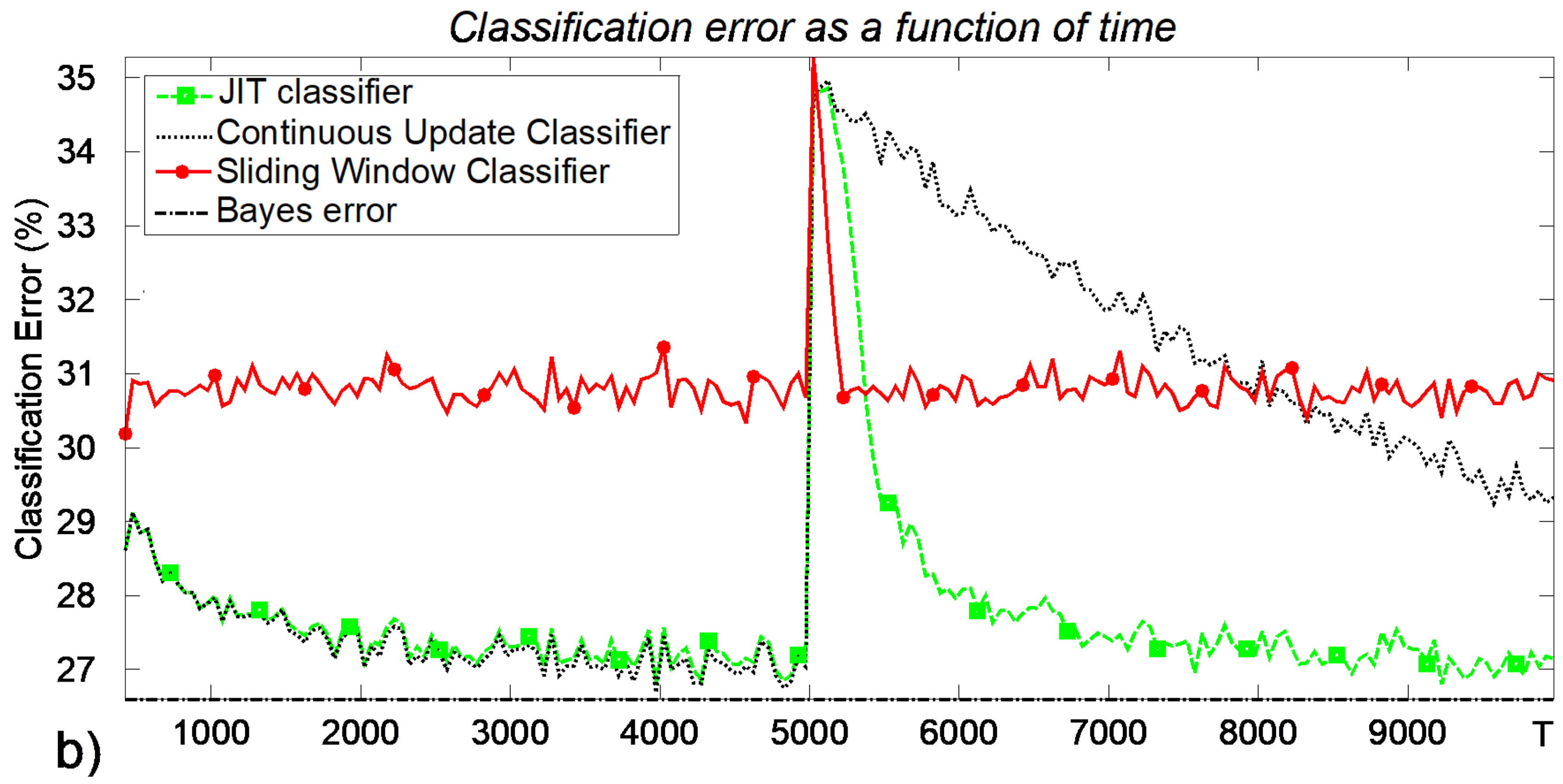


Boracchi

Simple Adaptation Strategies

Classification error of two simple adaptation strategies

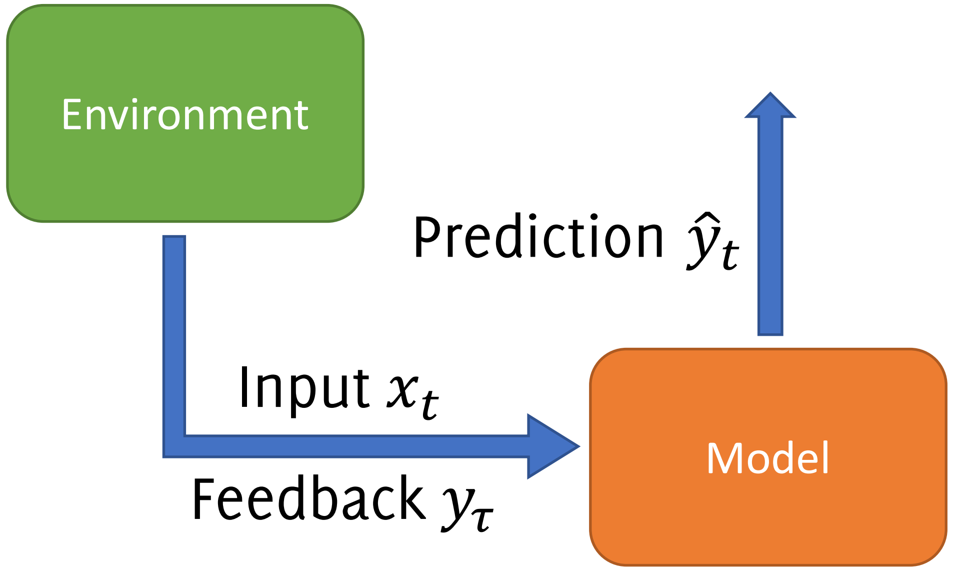
- Black dots: K_t uses all supervised couples at time t
- Red line: K_t uses only the last δ supervised couples



Alippi, C., Boracchi, G., Roveri, M. (2013). Just-in-time classifiers for recurrent concepts. *IEEE TNNLS* 620-634.

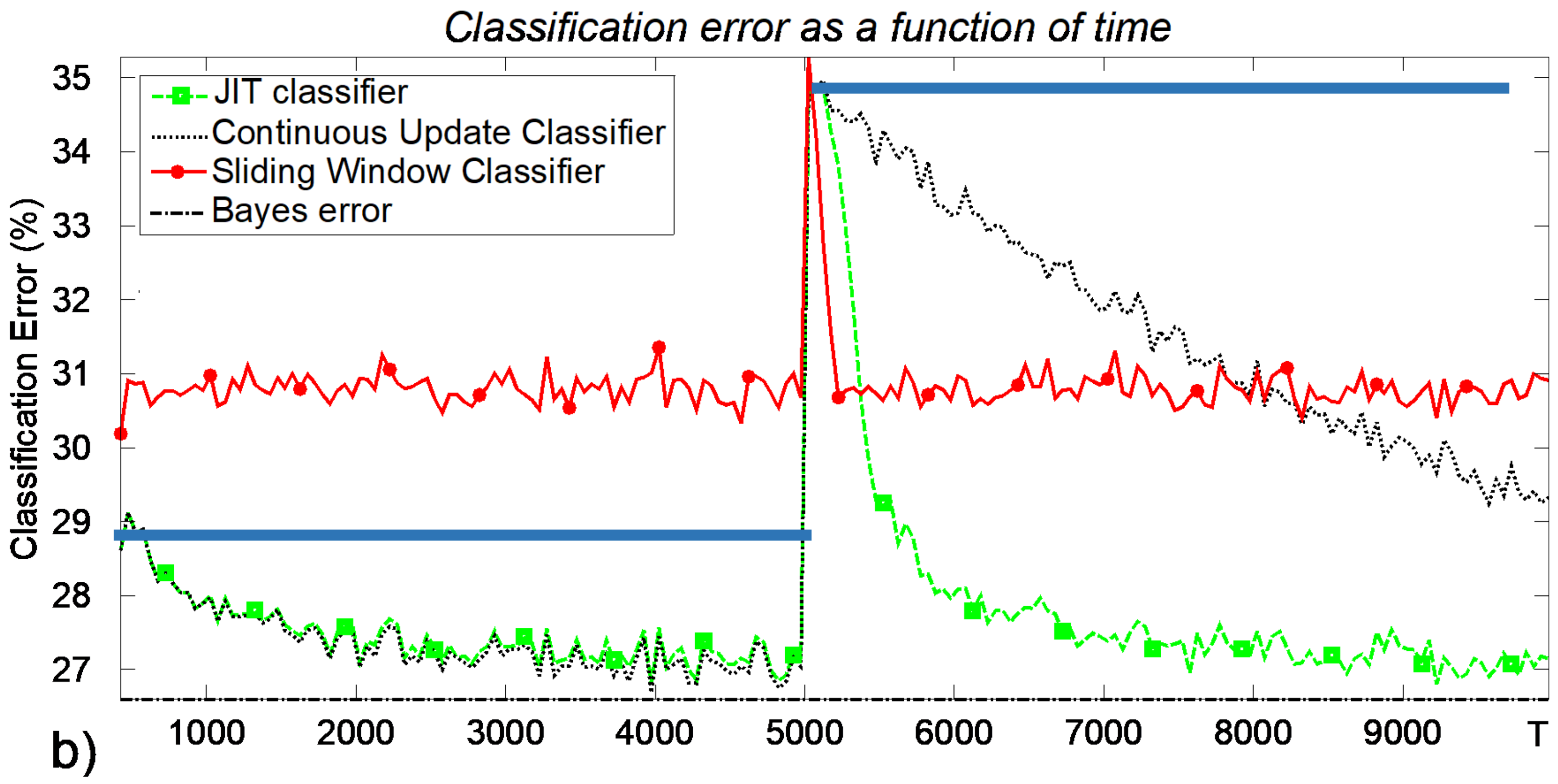
The LNSE loop

This are the standard (stationary/i.i.d.) ML settings



Classifier

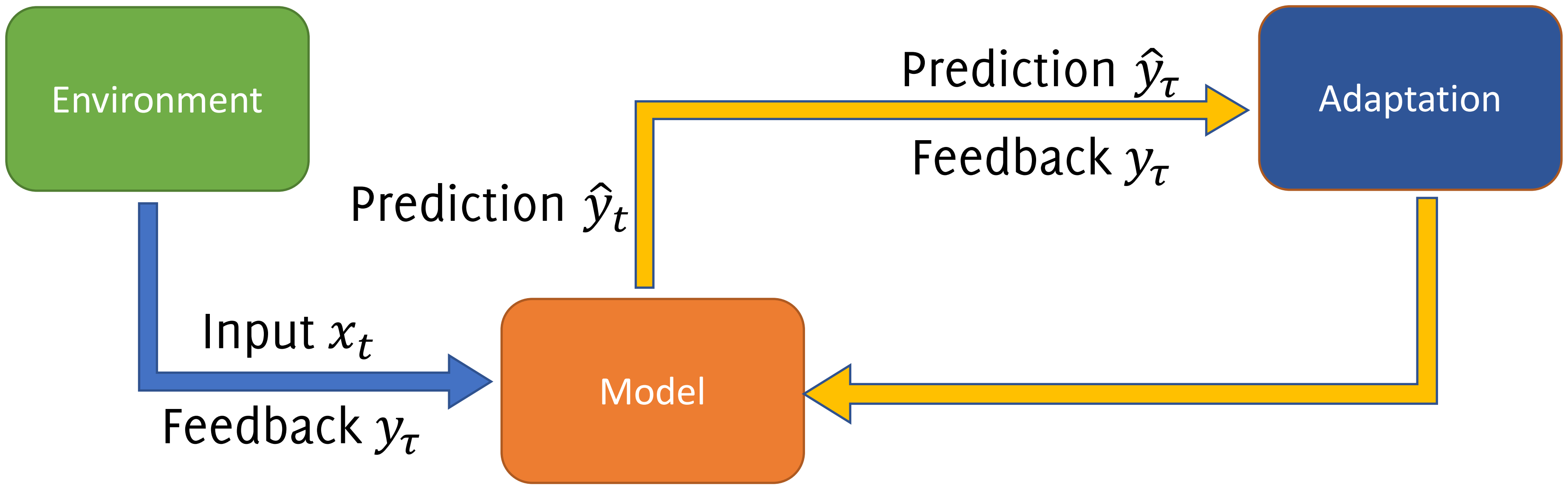
The blue solid line denotes the expected error of the classifier that is never updated. The classification error changes only because the classification problem is changing (concept drift).



Boracchi

The LNSE loop

This is the most simple pipeline in LNSE, which includes model adaptation

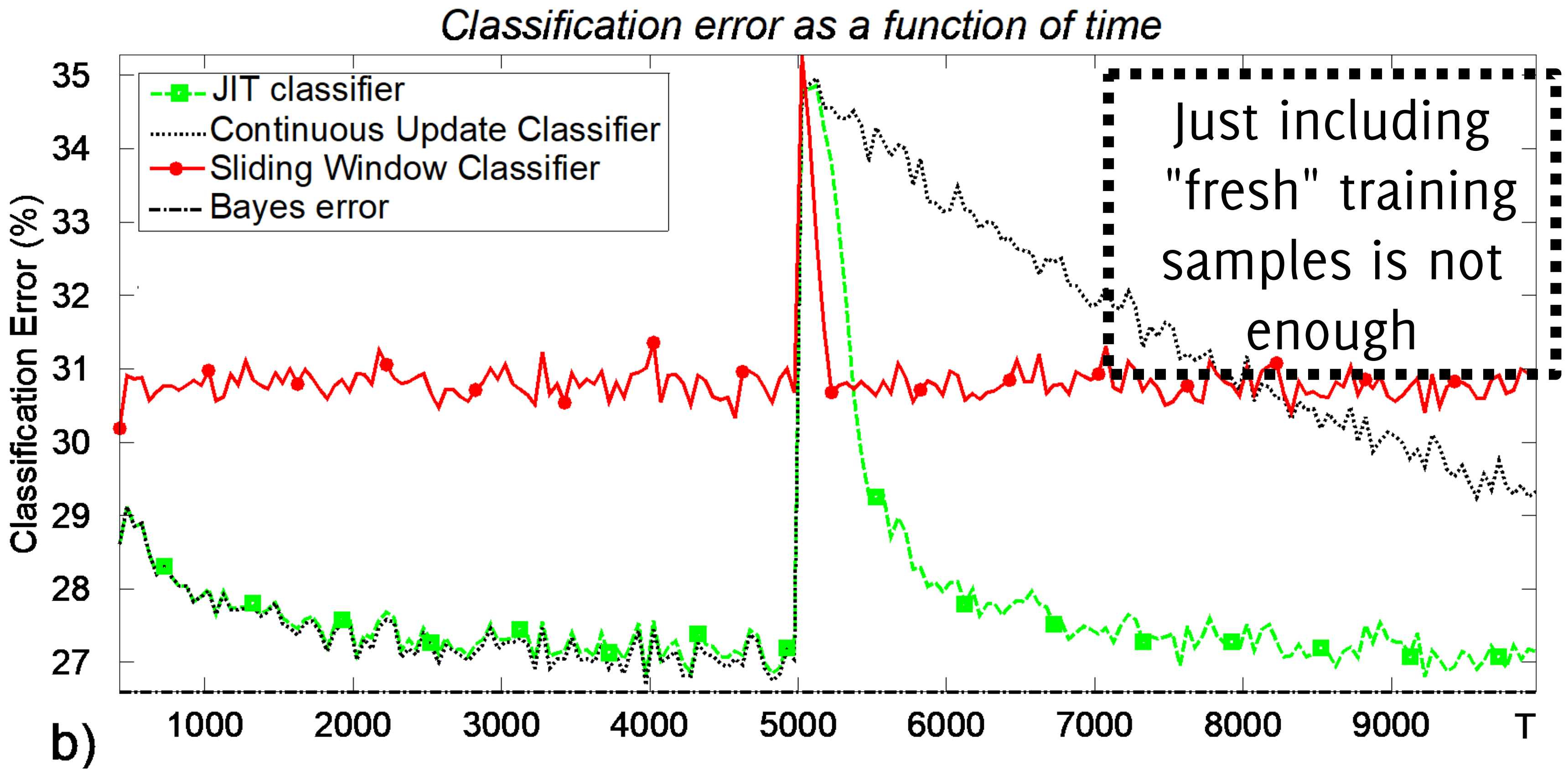


Different solutions follow different form of adaptation.

Simple Adaptation Strategies

Classification error of two simple adaptation strategies

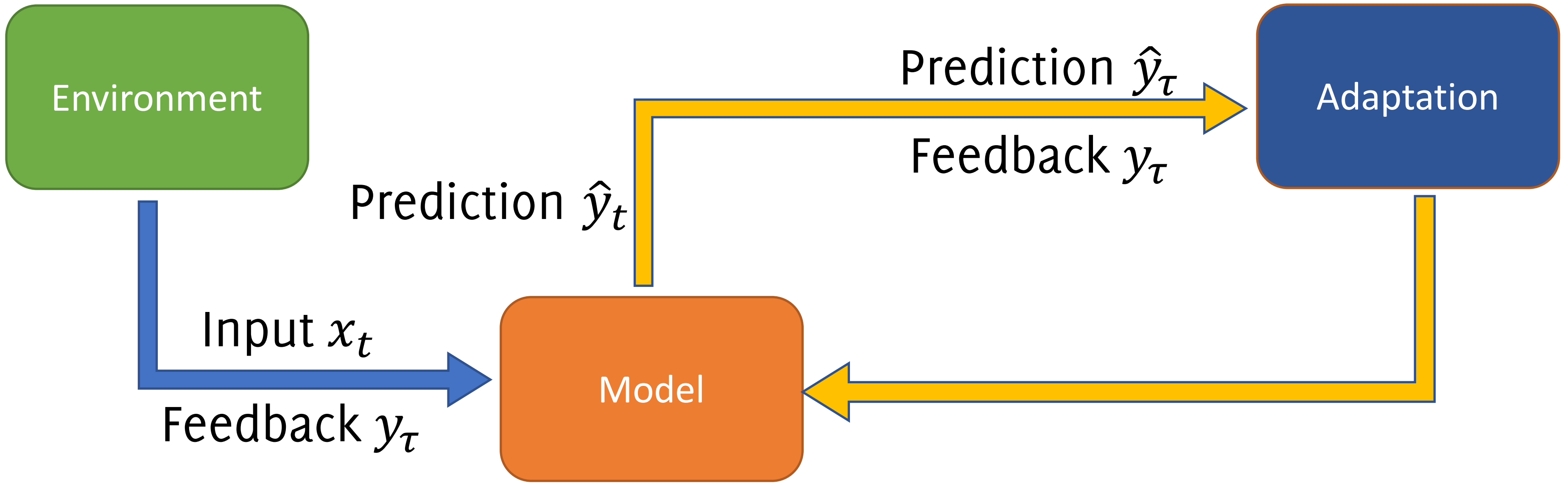
- Black dots: K_t uses all supervised couples at time t
- Red line: K_t uses only the last δ supervised couples



Just including "fresh" training samples is not enough

The LNSE loop

This is the most simple pipeline in LNSE, continuous adaptation

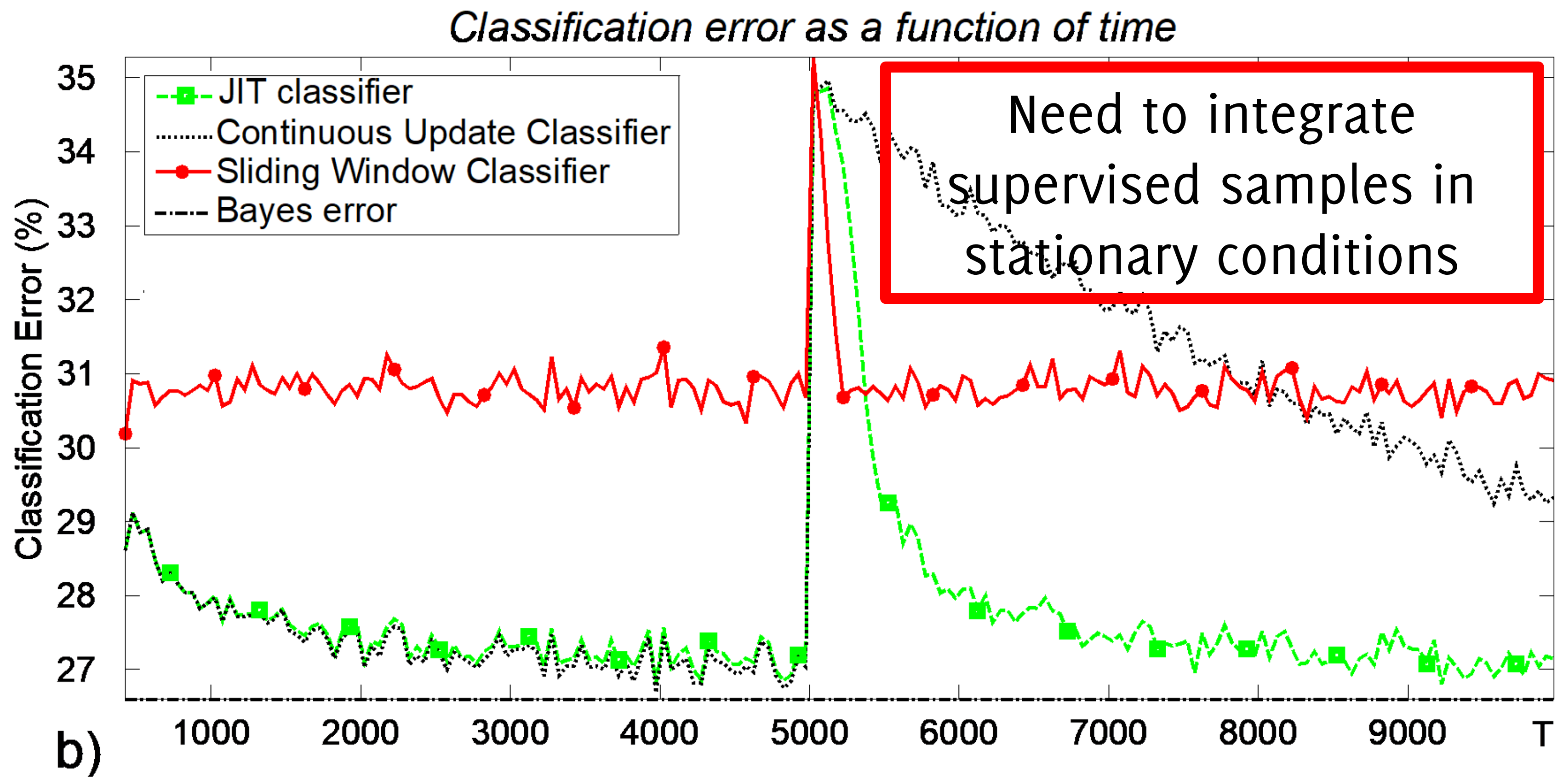


The incremental classifier (the black dashed line), appends the new label y_t to TR and forces retraining over the entire TR

Simple Adaptation Strategies

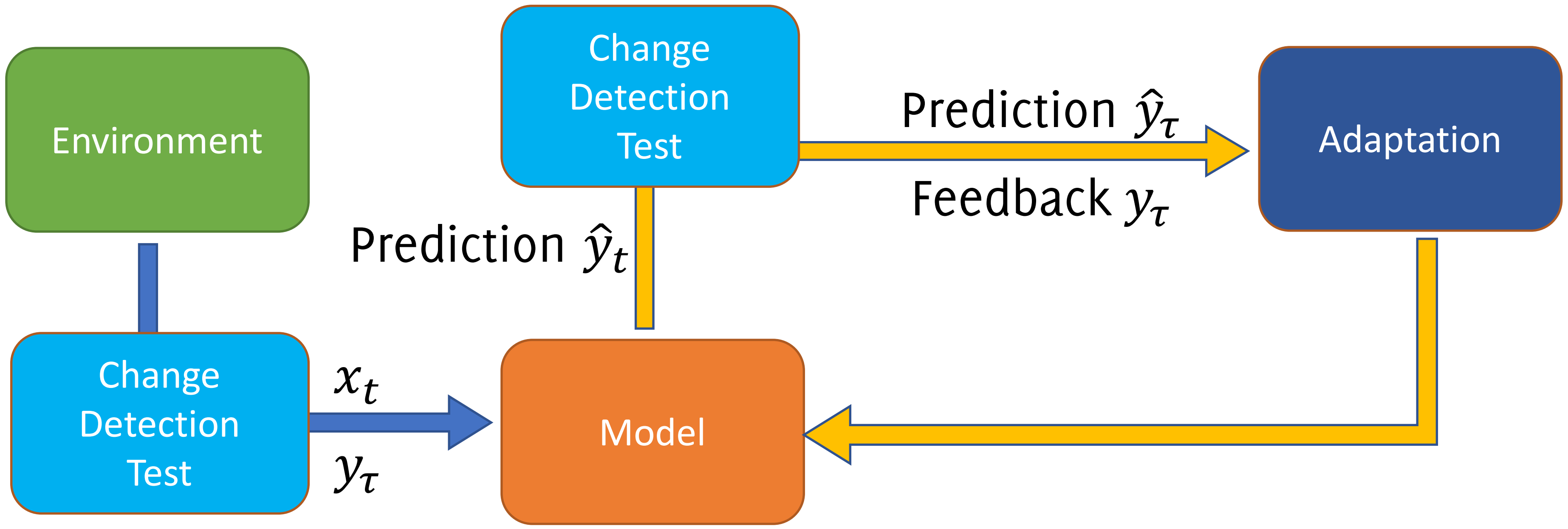
Classification error of two simple adaptation strategies

- Black dots: K_t uses all supervised couples at time t
- Red line: K_t uses only the last δ supervised couples



The LNSE loop

More sophisticated adaptation strategies are possible

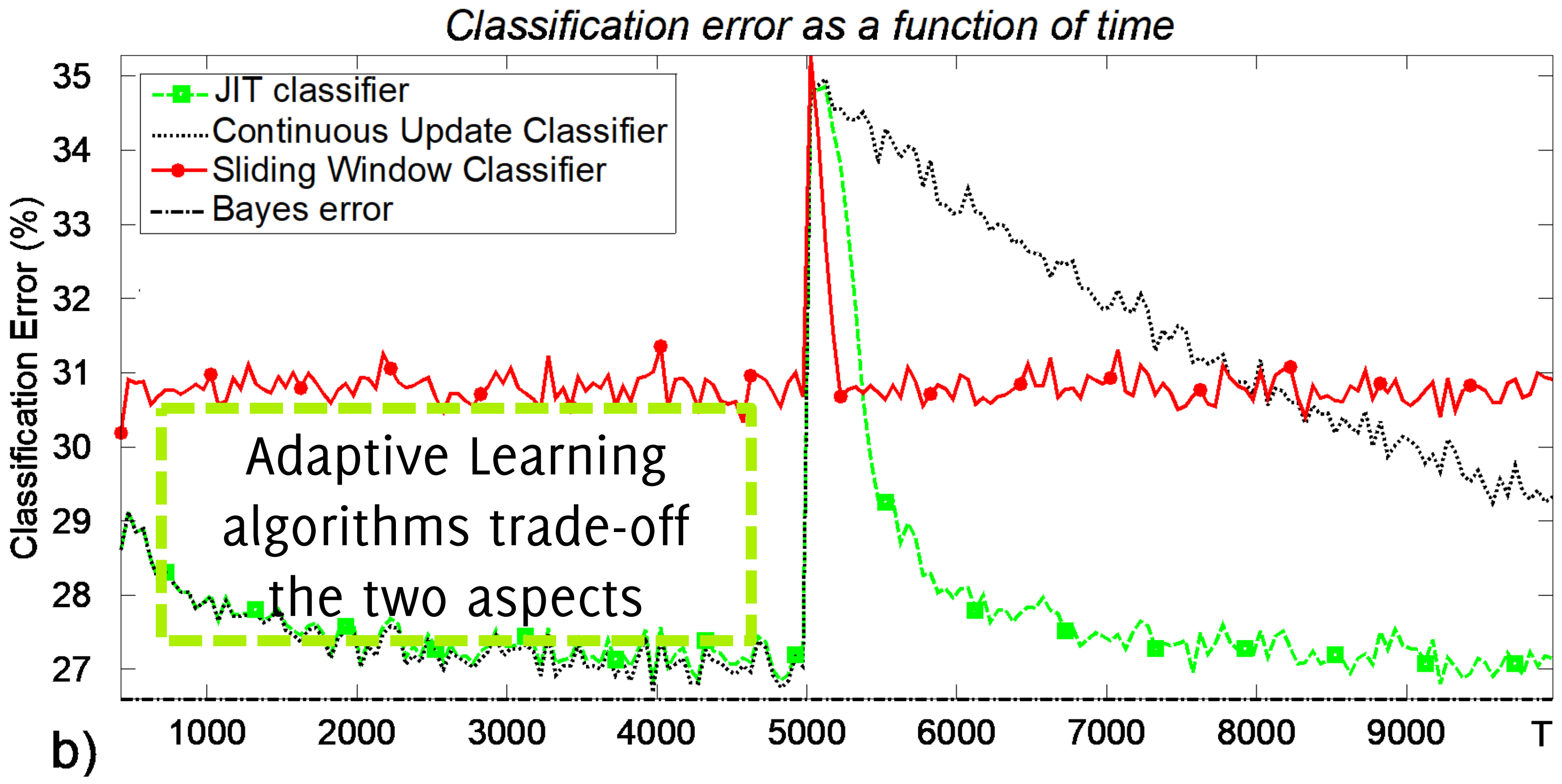


More

Simple Adaptation Strategies

Classification error of two simple adaptation strategies

- Black dots: K_t uses all supervised couples at time t
- Red line: K_t uses only the last δ supervised couples



Alippi, C., Boracchi, G., Roveri, M. (2013). Just-in-time classifiers for recurrent concepts. *IEEE TNNLS* 620-634.

Monitoring

Change
Detection
Test

Change Detection: Problem Formulation
.. In a statistical framework

Process Changes

Normal data are generated in **stationary** conditions, i.e. **are i.i.d. realizations** of a process $\mathcal{P}_N$

After the change, **data are** generated from a **different process** $\mathcal{P}_A \neq \mathcal{P}_N$, which **persists over time**

Examples:

- Quality inspection system: **faults** producing flawed components
- Environmental monitoring: **persistent changes in the morphology** of measured signals
- Change of **user interests** in on-demand platform

Change-Detection in a Statistical Framework

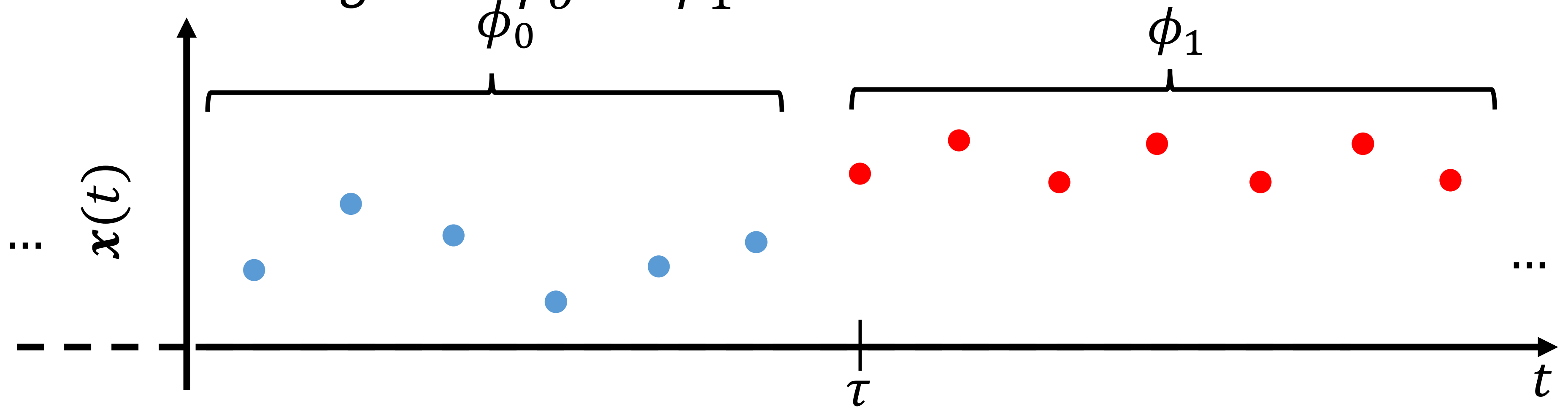
Often, the change-detection problem **boils down to:**

Monitor a **stream** $\{\boldsymbol{x}(t), t = 1, \dots\}$, $\boldsymbol{x}(t) \in \mathbb{R}^d$ of realizations of a **random variable**, and **detect the change-point** τ,

$$\boldsymbol{x}(t) \sim \begin{cases} \phi_0 & t < \tau \quad \text{in control state} \\ \phi_1 & t \geq \tau \quad \text{out of control state} \end{cases},$$

where $\{\boldsymbol{x}(t),\ t < \tau\}$ **are i.i.d.** and $\phi_0 \neq \phi_1$

We denote such change as: $\phi_o \to \phi_1$



Change-Detection in a Statistical Framework

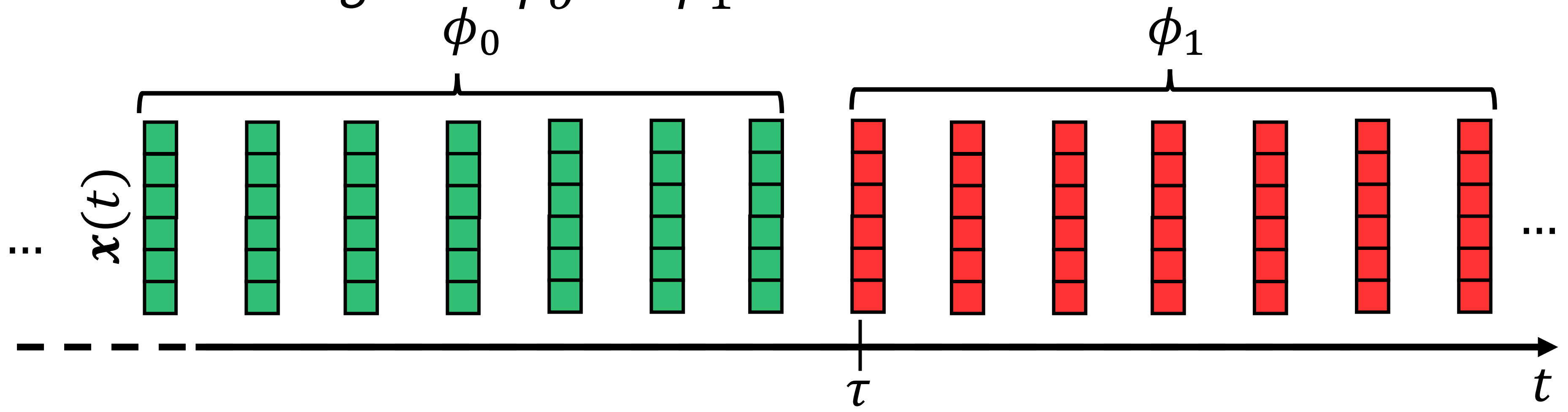
Often, the change-detection problem **boils down to:**

Monitor a **stream** $\{\boldsymbol{x}(t), t = 1, \dots\}$, $\boldsymbol{x}(t) \in \mathbb{R}^d$ of realizations of a **random variable**, and **detect the change-point** τ,

$$\boldsymbol{x}(t) \sim \begin{cases} \phi_0 & t < \tau \quad \text{in control state} \\ \phi_1 & t \geq \tau \quad \text{out of control state} \end{cases},$$

where $\{\boldsymbol{x}(t),\ t < \tau\}$ **are i.i.d.** and $\phi_0 \neq \phi_1$

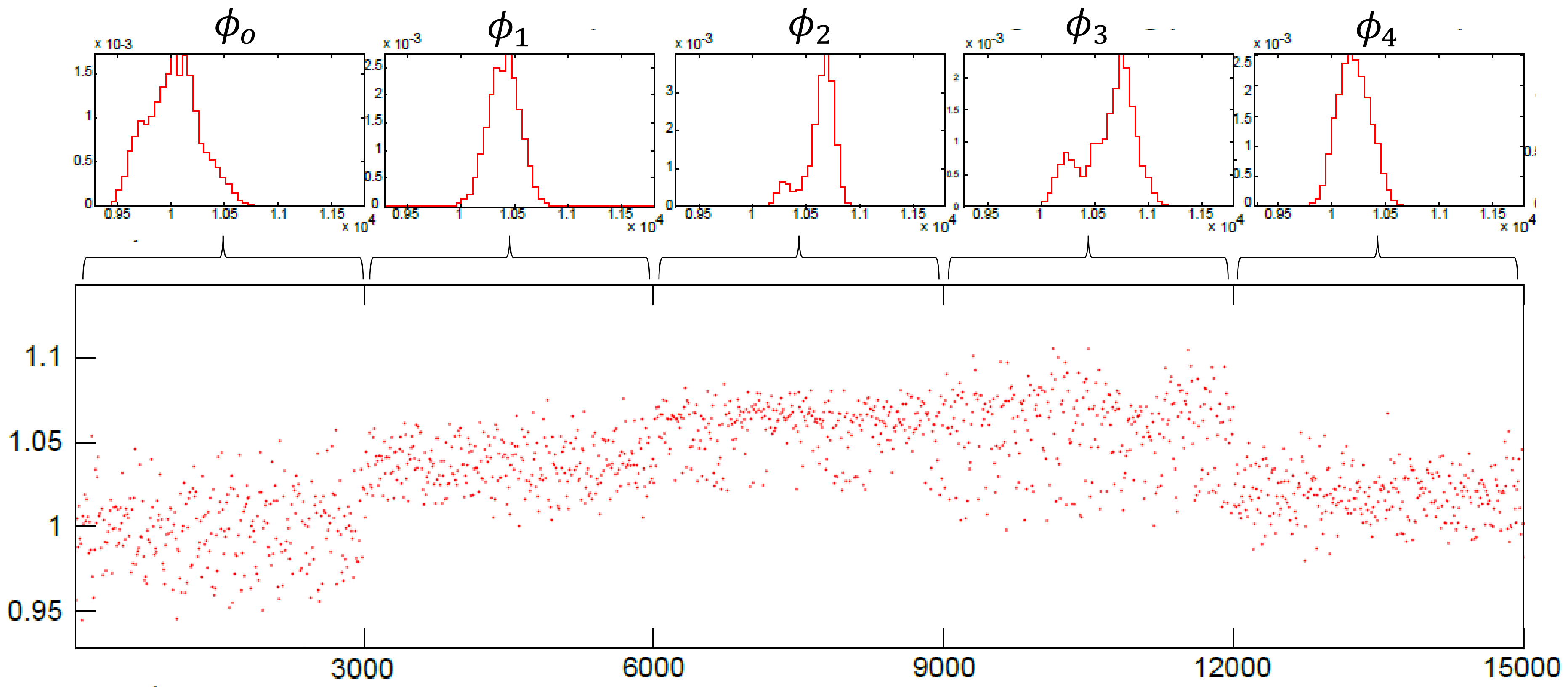
We denote such change as: $\phi_o \to \phi_1$



Change-Detection in a Statistical Framework

Here are data from an X-ray monitoring apparatus.

There are 4 changes $\phi_o \rightarrow \phi_1 \rightarrow \phi_2 \rightarrow \phi_3 \rightarrow \phi_4$ corresponding to different monitoring conditions and/or analyzed materials



Change Detection Questions

Change-detection question:

Given the previously estimated model, the arrival of new data invites the question: "Is yesterday's model capable of explaining today's data?"

Detecting process changes is important to understand the monitored phenomenon

V. Chandola, A. Banerjee, V. Kumar. "*Anomaly detection: A survey*". ACM Comput. Surv. 41, 3, Article 15 (2009), 58 pages.

C. J. Chu, M. Stinchcombe, H. White "*Monitoring Structural Change*" Econometrica Vol. 64, No. 5 (Sep., 1996), pp. 1045-1065.

The Typical Solution

Most algorithms are composed of:

- A **statistic** that has a **known response to normal data** (e.g., the average, the sample variance, the log-likelihood, the confidence of a classifier, an “anomaly score”…)
- A **decision rule** to analyze the statistic (e.g., an adaptive threshold, a confidence region)

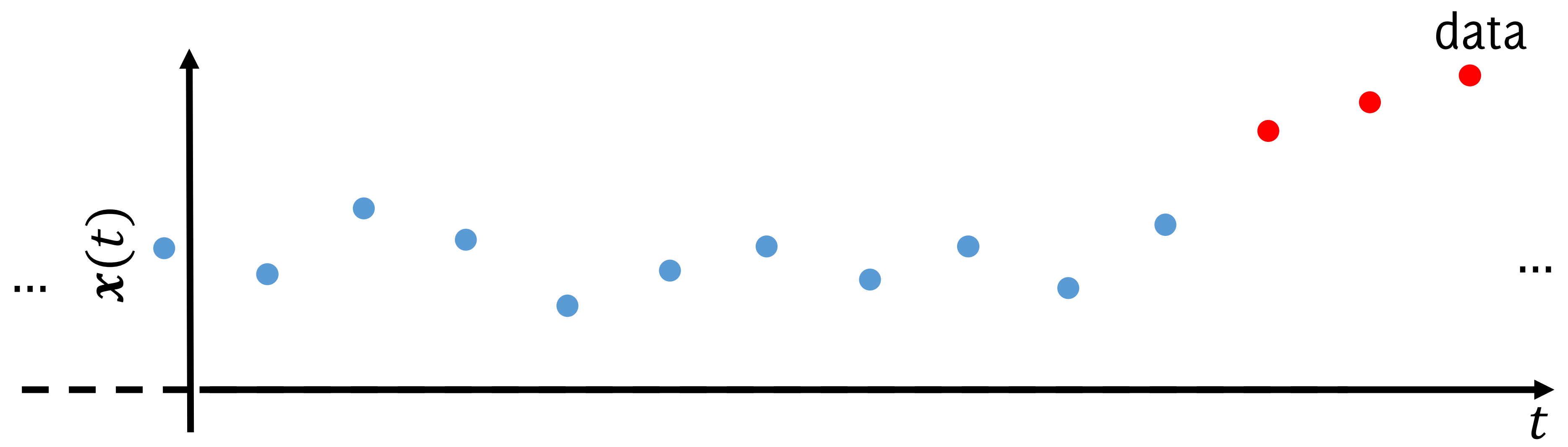
Statistics and Decision Rules

Change-detection algorithms:

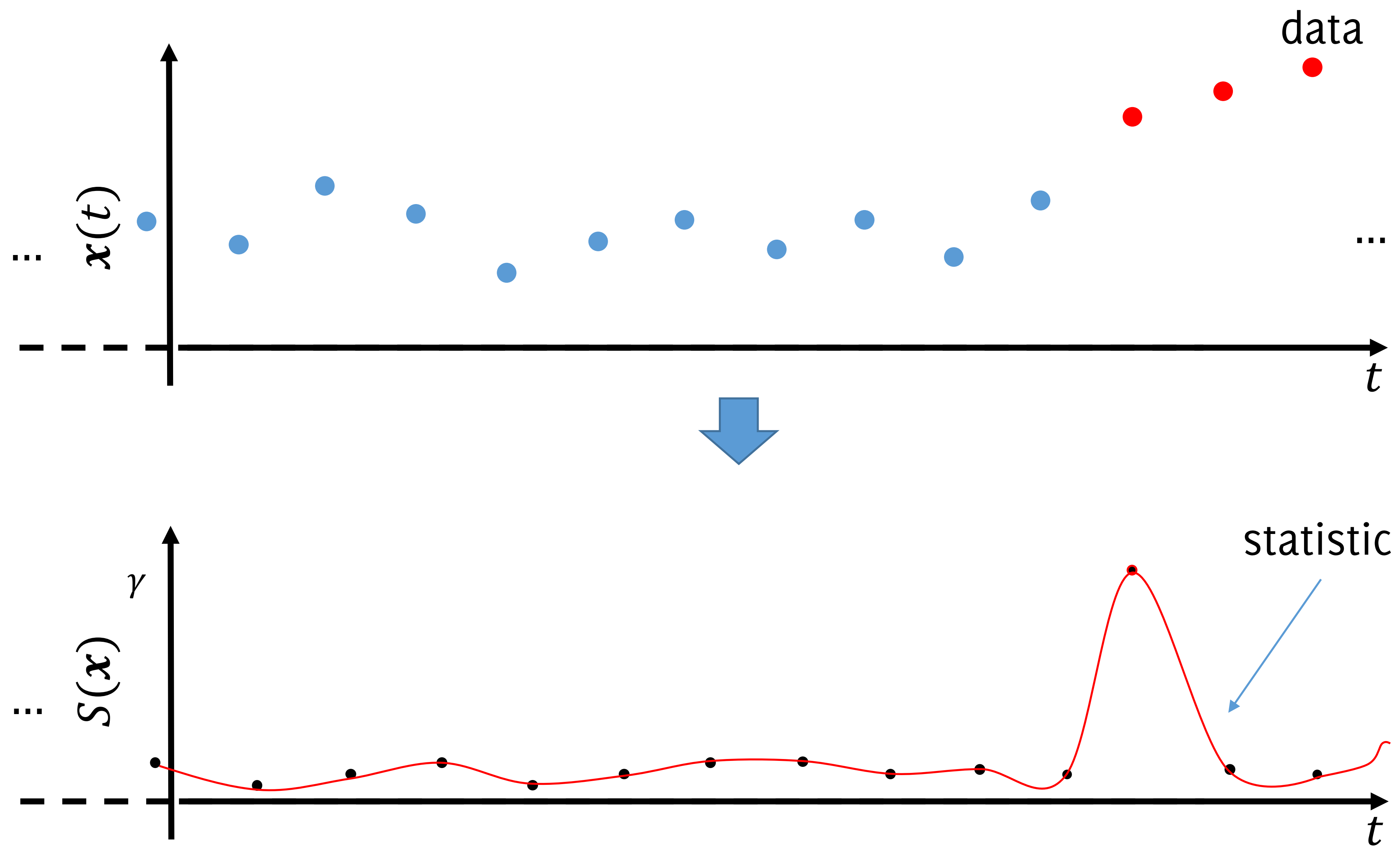
Statistics and decision rules are **sequential**, as they make a decision considering --in principle-- all the data received so far. Integrating information over time makes these algorithms able to detect subtle changes as well.

E.g.: The cumulative average of all the points

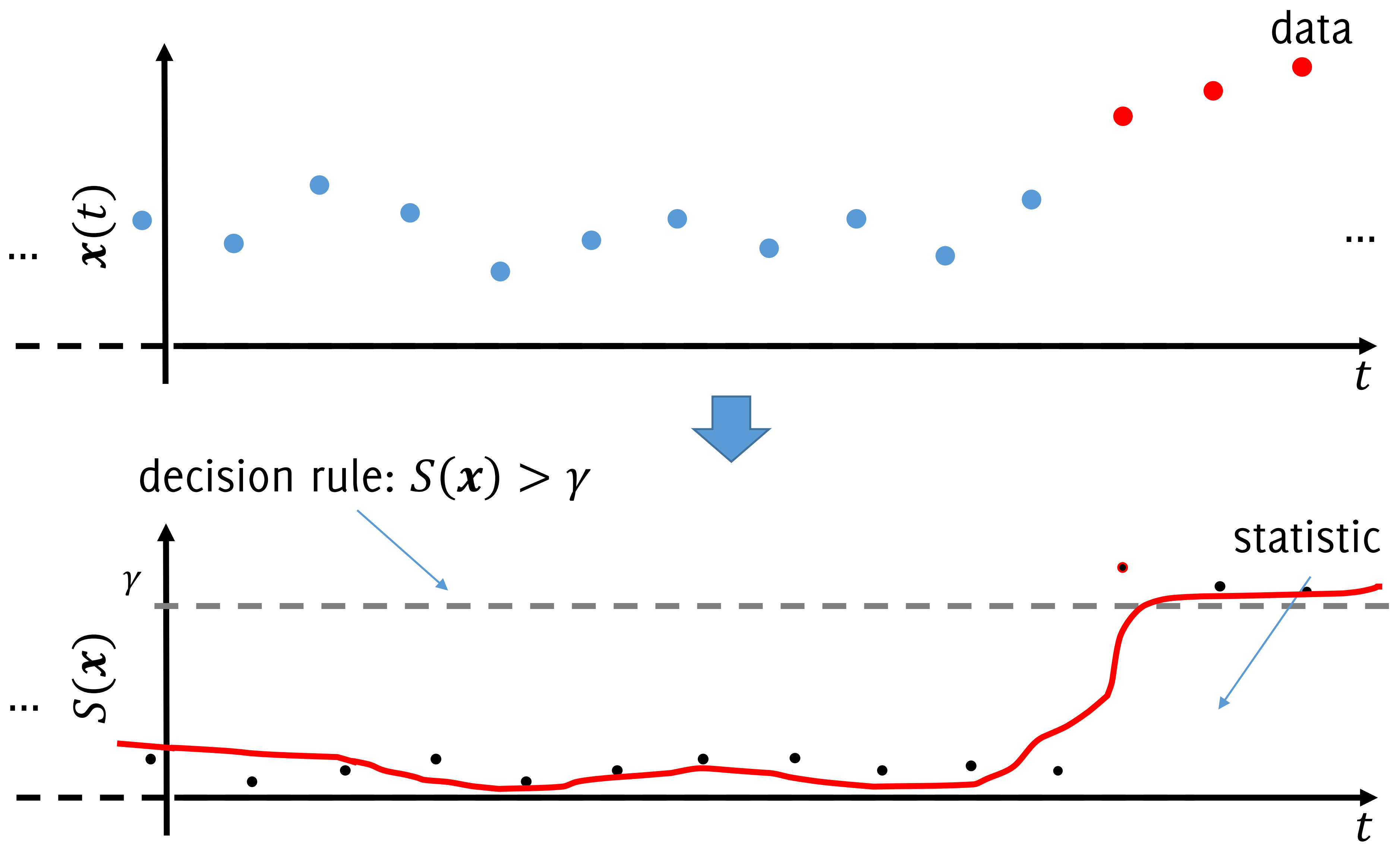
The Typical Solution



The Typical Solution

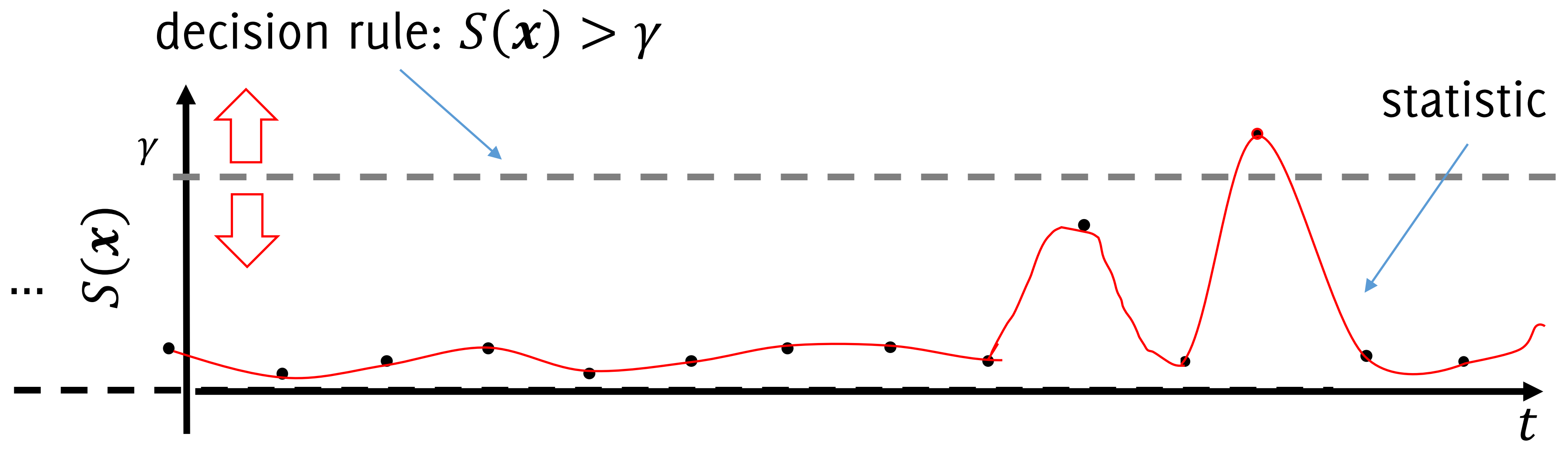


The Typical Solutions



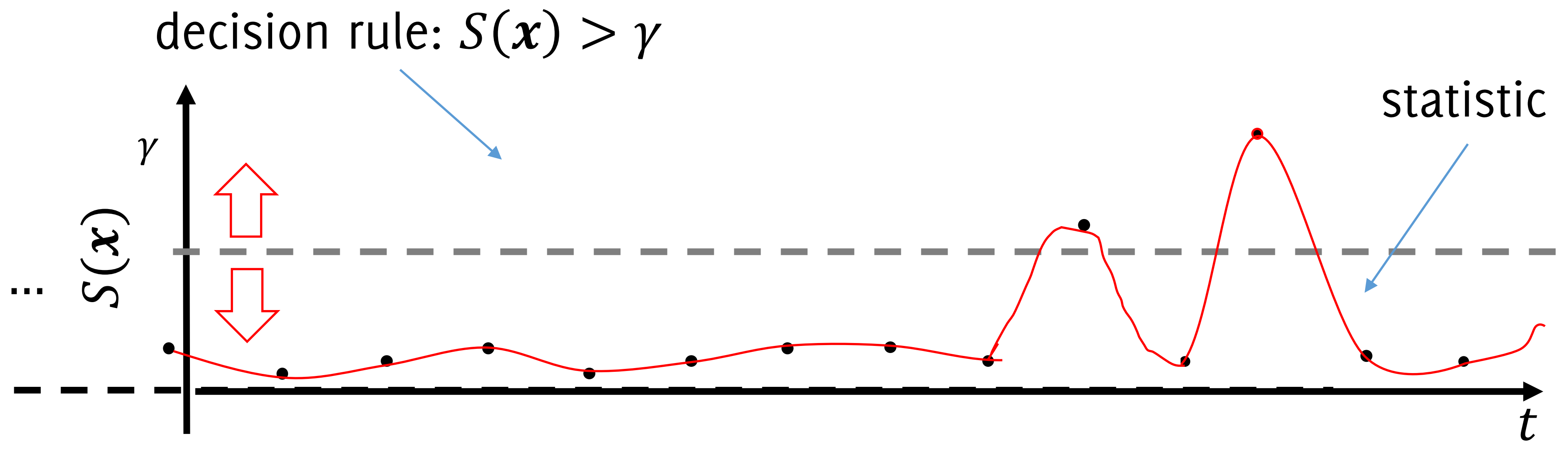
The Typical Solutions

By changing γ it is possible to achieve different detection performance (e.g. more true positive, more false positives)



The Typical Solutions

By changing γ it is possible to achieve different detection performance (e.g. more true positive, more false positives)



Statistics and Decision Rules

Detection rules often **rely on thresholds,** namely γ

In both these cases: It is of primary concern to control *false positives*, namely “how often” a change/anomaly is detected within stationary data.

Controlling False Positives in Change Detection

In change-detection false positives are controlled by the Average Run Length ARL_0 :

$$ARL_0 = \mathrm{E}_{\boldsymbol{x}}[\hat{\tau} | \boldsymbol{x} \sim \phi_0]$$

Thus **denotes the expected time between false positive detections**

Controlling False Positives in Change Detection

A good change-detection test is accompanied with a table/rule/formula that defines, for a target value of ARL_0, the corresponding threshold γ

$$\gamma = \gamma(ARL_0)$$

Watch out: thresholds depend on the statistics S**, which in turn might depend on the distribution of the monitored data** ϕ_x^0

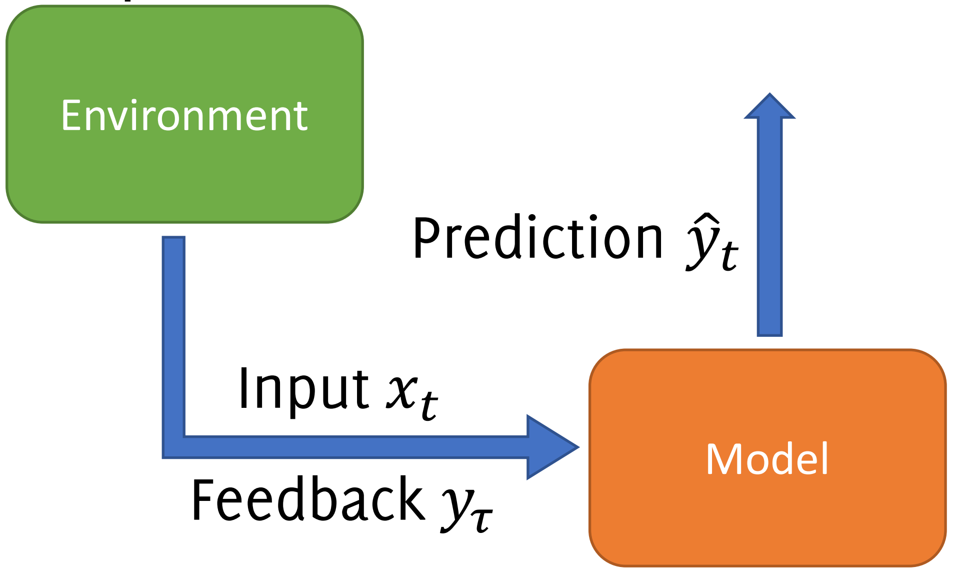
Threshold computation for change-detection algorithm **is more complicated than in anomaly-detection** algorithm since bootstrap procedure has to **consider temporal evolution** of the analysis

Learning in NSE by Monitoring the Classification Error

…when ϕ_0 and ϕ_1 are unknown

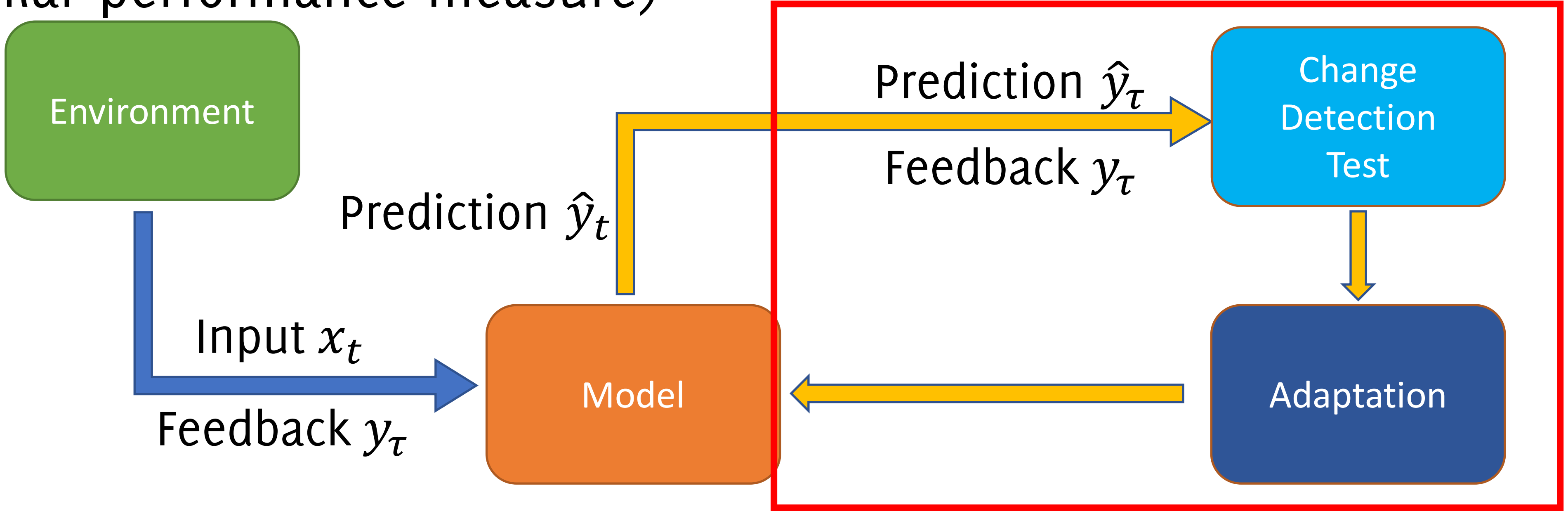
Monitoring the Classification Error

The simplest approach consist in monitoring the classification error (or similar performance measure)



Monitoring the Classification Error

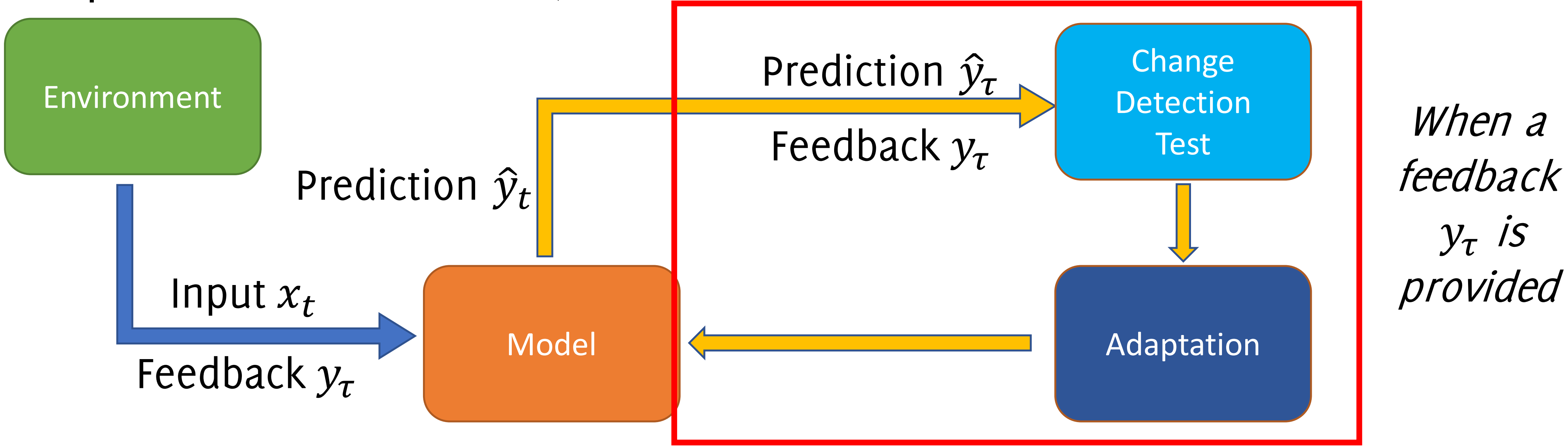
The simplest approach consist in monitoring the classification error (or similar performance measure)



When a feedback y_τ is provided

Monitoring the Classification Error

The simplest approach consist in monitoring the classification error (or similar performance measure)



Pro:

- The classification error is the most straightforward figure of merit to monitor
- Changes prompts adaptation only when performance are affected

Cons:

- CD detection from supervised samples only

Monitoring the Classification Error

The element-wise classification error e_t follows a Bernoulli pdf

$$e_t \sim \text{Bernulli}(p_0)$$

Which is a discrete probability distribution of a random variable which:

- Takes the value 1 with probability p_0
- Takes the value 0 with probability $1 - p_0$

Where π_0 is the expected classification error when $\boldsymbol{x} \sim \phi_0$

$$\text{expect}(\text{Bernulli}(p_0)) = p_0, \qquad \text{variance}(\text{Bernulli}(p_0)) = p_0(1 - p_0)$$

Monitoring the Classification Error

The sum of errors e_t in a **window of $\boldsymbol{\nu}$ samples** follows a Binomial pdf

$$\sum_{t=T-\nu}^{T} e_t \sim \mathcal{B}(p_0, \nu)$$

which is also a discrete distribution

$$\text{expect}\big(\mathcal{B}(p_0,\nu)\big) = \nu p_0, \qquad \text{variance}\big(\mathcal{B}(p_0,\nu)\big) = p_0$$

Gaussian approximation holds when ν is sufficiently large

$$\frac{1}{\nu}\,\mathcal{B}(p_0,\nu) \approx \mathcal{N}\left(p_0, \frac{p_0}{\nu}\right)$$

The average classification error **over disjoint windows of** ν samples $p_t = \frac{1}{\nu}\sum_{t=T-\nu}^{T} e_t$ can be approximated as a sequence of i.i.d. realization of a Gaussian distributed random value. **Overlaps among the windows drop the independence**.

Monitoring the Classification Error: DDM

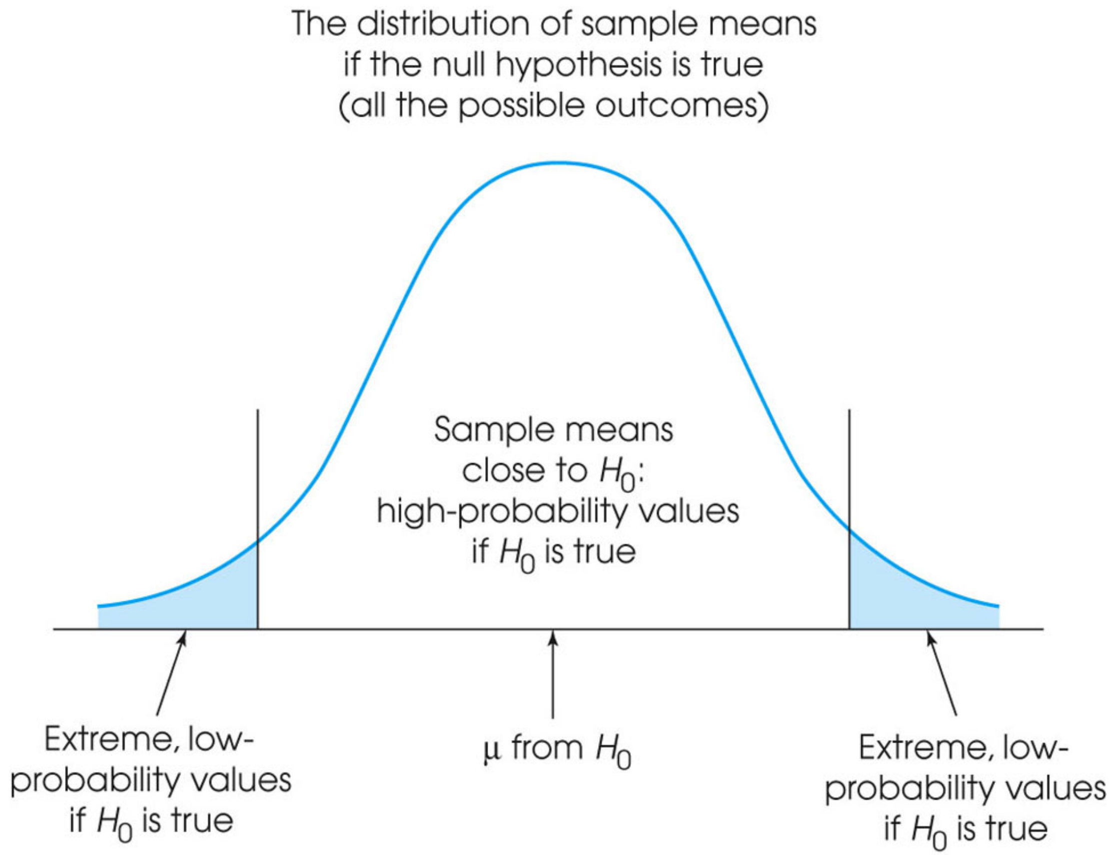
Basic idea behind Drift Detection Method (DDM):

J. Gama, P. Medas, G. Castillo, and P. Rodrigues. *"Learning with Drift Detection"* SBIA. Springer, Berlin, 286–295, 2004

Monitoring the Classification Error: DDM

Basic idea behind Drift Detection Method (DDM):

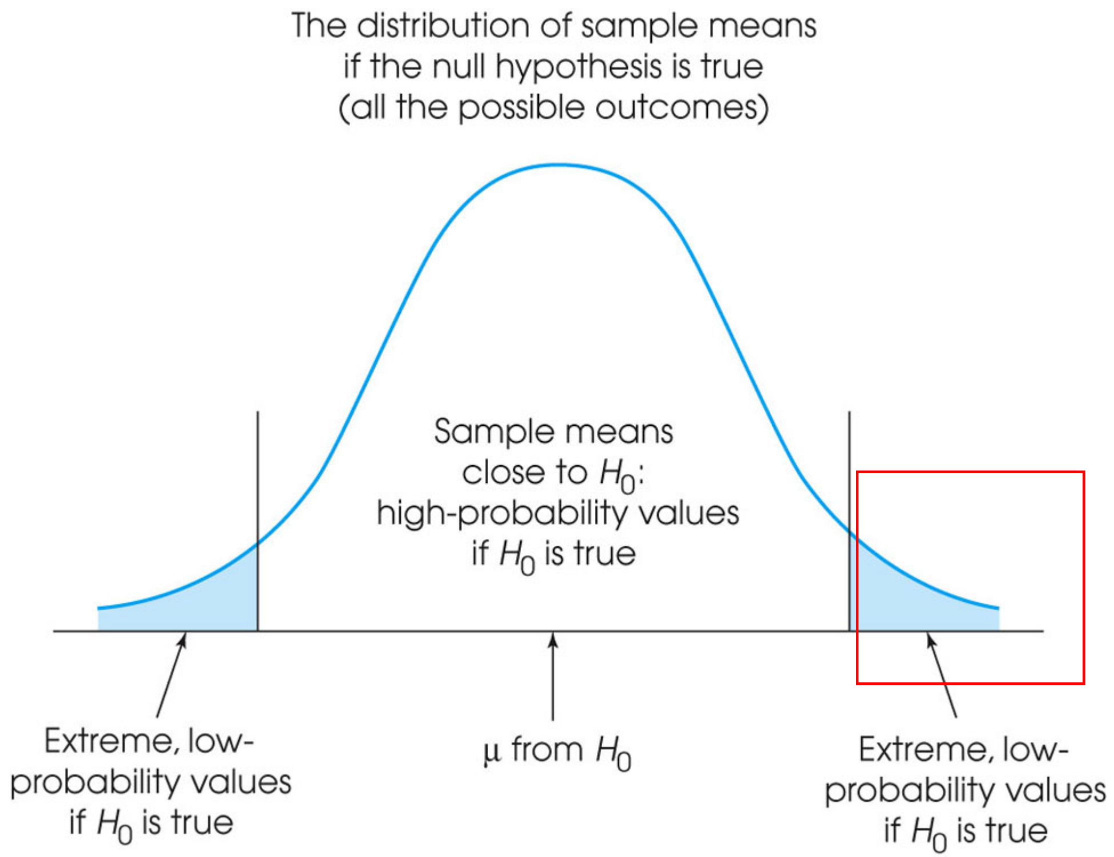
- Detect concept drift as an **outlier** in the classification error


The distribution of sample means if the null hypothesis is true (all the possible outcomes)
Sample means close to H_0: high-probability values if H_0 is true
Extreme, low-probability values if H_0 is true
μ from H_0
Extreme, low-probability values if H_0 is true

Monitoring the Classification Error: DDM

Basic idea behind Drift Detection Method (DDM):

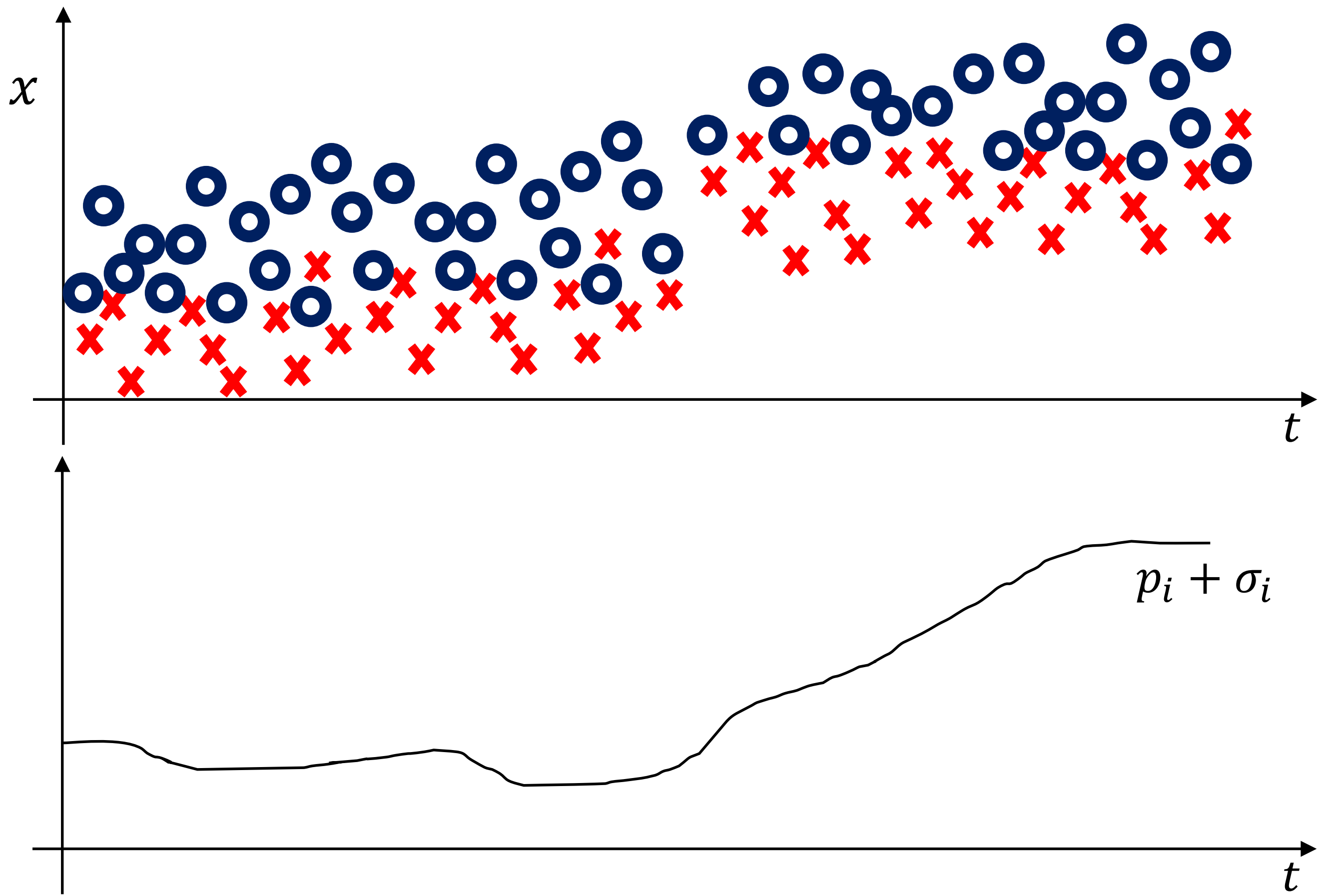
- Detect concept drift as an **outlier** in the classification error
- In stationary conditions error decreases, **look for outliers** in the right tail



Monitoring the Classification Error: DDM

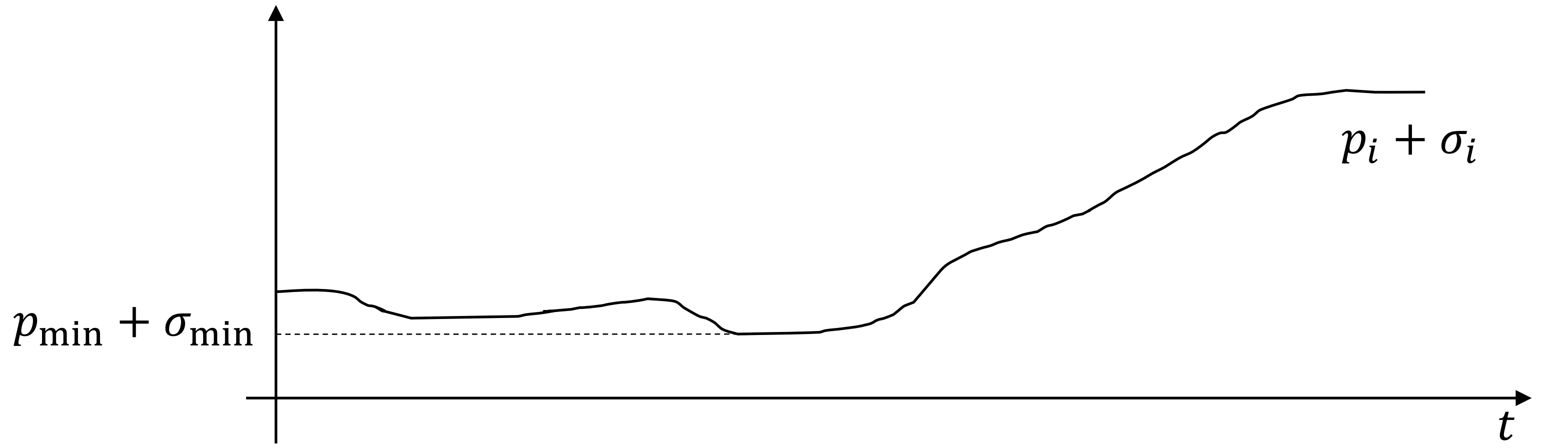
1. During monitoring, steadily compute p_i and $\sigma_i = \sqrt{\frac{p_i(1-p_i)}{i}}$

These are the sample estimates of the mean and standard deviation of the Gaussian of the average error over the first i samples



Monitoring the Classification Error: DDM

1. During monitoring, steadily compute p_i and $\sigma_i = \sqrt{\frac{p_i(1-p_i)}{i}}$
2. Let $p_{\min}$ be the minimum error before i and $\sigma_{\min} = \sqrt{\frac{p_{\min}(1-p_{\min})}{i}}$

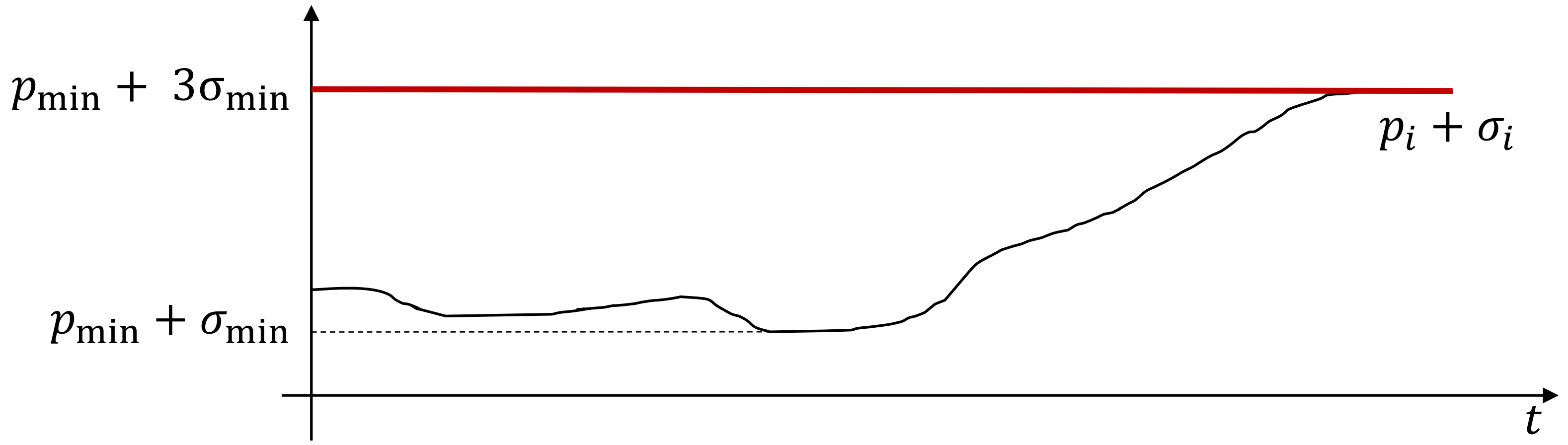


Boracchi

Monitoring the Classification Error: DDM

1. During monitoring, steadily compute p_i and $\sigma_i = \sqrt{\frac{p_i(1-p_i)}{i}}$
2. Let $p_{\min}$ be the minimum error before i and $\sigma_{\min} = \sqrt{\frac{p_{\min}(1-p_{\min})}{i}}$
3. Detect concept drift when $p_i + \sigma_i > p_{\min} + 3 * \sigma_{\min}$

This is an heuristic decision rule, that does not guarantee control over FPR



Boracchi

Adaptation Heuristic in DDM

Adaptation

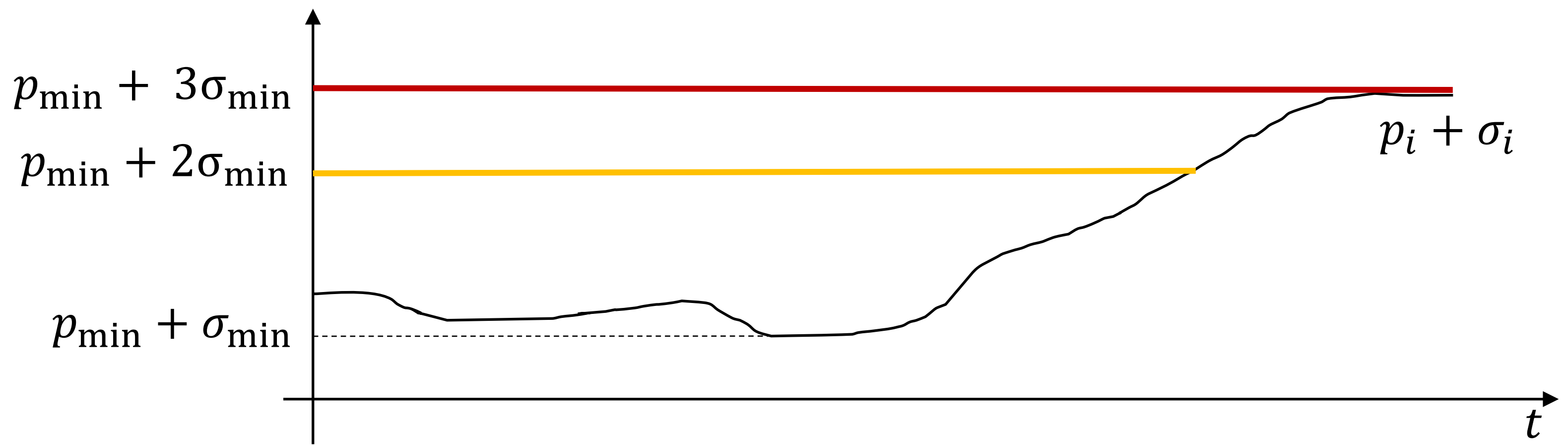
Monitoring the Classification Error: DDM

1. During monitoring, steadily compute p_i and $\sigma_i = \sqrt{\frac{p_i(1-p_i)}{i}}$

2. Let $p_{\min}$ be the minimum error before i and $\sigma_{\min} = \sqrt{\frac{p_{\min}(1-p_{\min})}{i}}$

3. Raise a "warning" when $p_i + \sigma_i > p_{\min} + 2 * \sigma_{\min}$

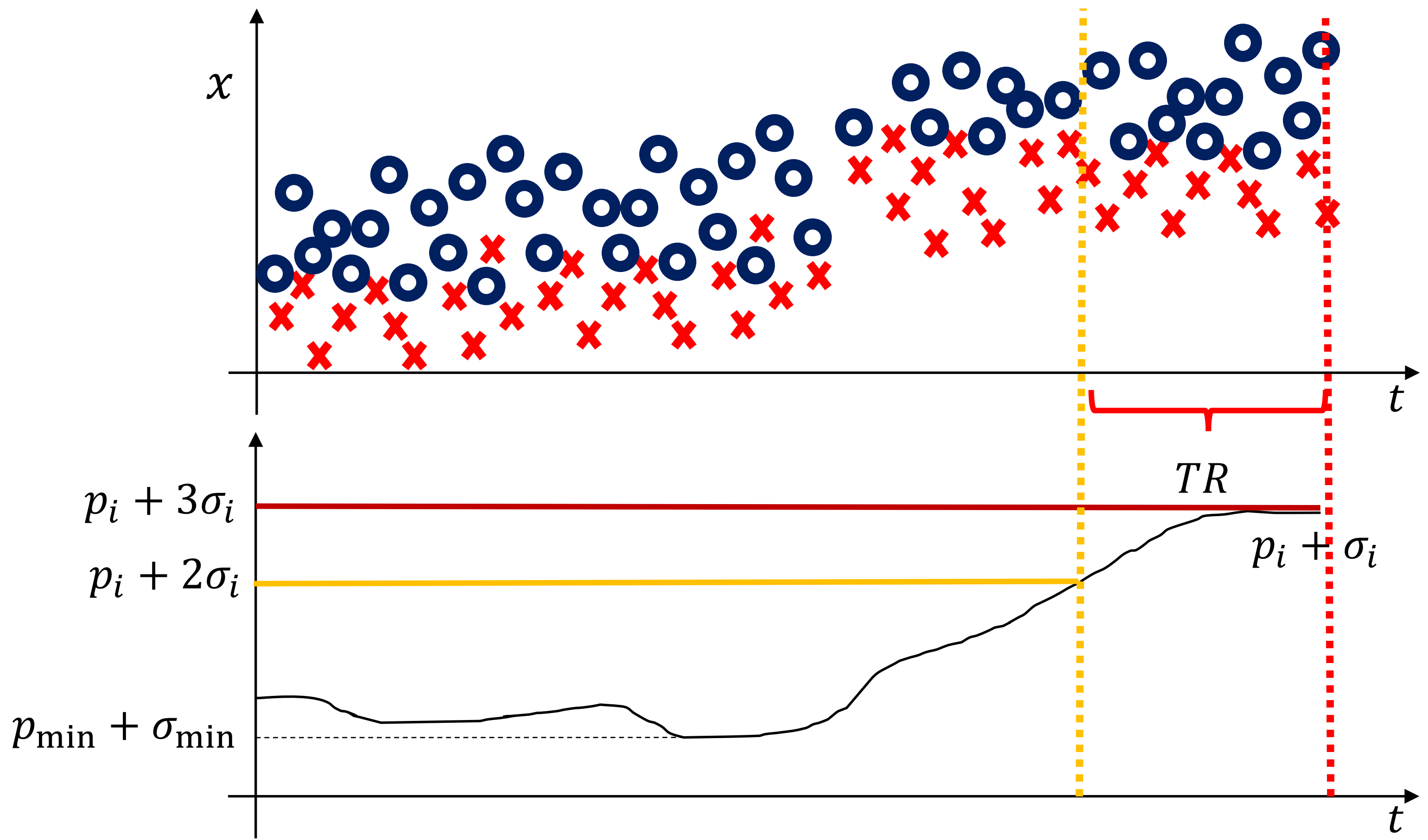
4. Detect concept drift when $p_i + \sigma_i > p_{\min} + 3 * \sigma_{\min}$



Boracchi

Post-detection Adaptation: DDM

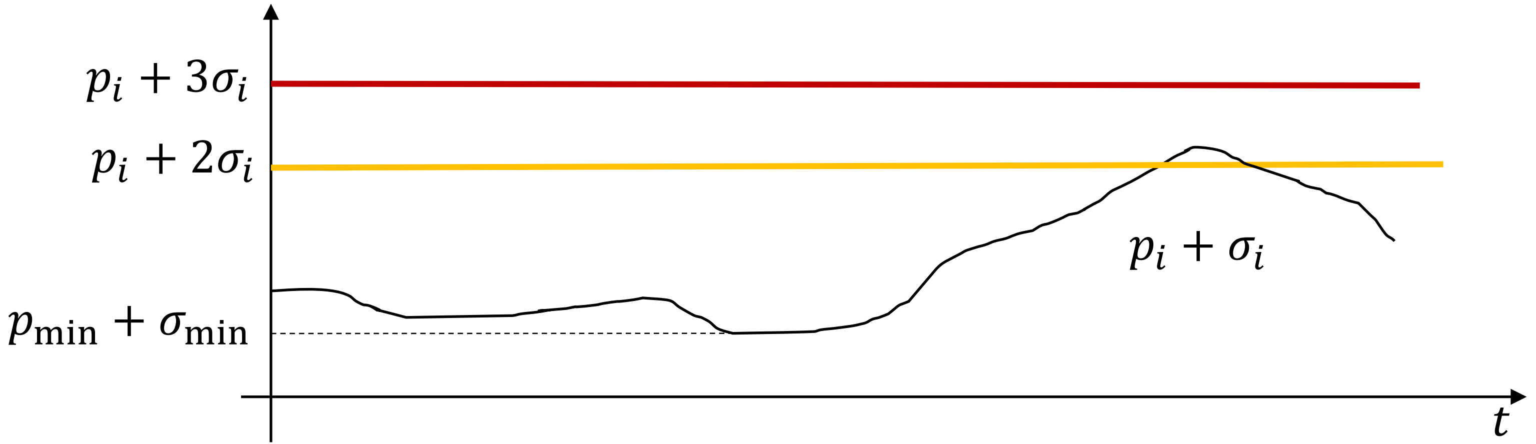
Use supervised samples in **between warning** and **drift** alert to **reconfigure** the classifier



Post-detection Adaptation: DDM

Use supervised samples in **between warning** and **drift** alert to **reconfigure** the classifier

Warning alerts non that are not followed by a drift alert are discarded and considered false-positive detections



Other Monitoring Solutions for the Classification Error

Adaptation

Monitoring the Classification Error: EDDM

Early Drift Detection Methods (EDDM) performs similarly but **monitors the average distance between misclassified samples**

- Average distance between two mis-classifier samples is expected to decrease under CD
- They aim at detecting gradual drifts

M. Baena-García, J. Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldá, R. Morales-Bueno. *"Early drift detection method"* In Fourth International Workshop on Knowledge Discovery from Data Streams (2006)

Monitoring the Classification Error: EWMA

Use the **Exponential Weighted Moving Average** (EWMA) as tests statistic
EWMA statistic, which is a convex combination of current error and average over the previous ones

$$Z_t = (1-\lambda)Z_{t-1} + \lambda\, e_t, \qquad Z_0 = 0$$

where $\lambda \in (0,1)$ is a configuration parameter, $e_t \in \{0,1\}$

G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand *"Exponentially Weighted Moving Average Charts for Detecting Concept Drift"* Pattern Recogn. Lett. 33, 2 (Jan. 2012), 191–198 2012

Monitoring the Classification Error: EWMA

Use the **Exponential Weighted Moving Average** (EWMA) as tests statistic
EWMA statistic, which is a convex combination of current error and average over the previous ones

$$Z_t = (1-\lambda)Z_{t-1} + \lambda\, e_t, \qquad Z_0 = 0$$

where $\lambda \in (0,1)$ is a configuration parameter, $e_t \in \{0,1\}$

Now, if you expand the expression

$$Z_t = (1-\lambda)\big((1-\lambda)Z_{t-2} + \lambda\, e_{t-1}\big) + \lambda\, e_t,$$

$$\dots$$

$$Z_t = \sum (1-\lambda)^{t-i} \lambda e_i$$

G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand *"Exponentially Weighted Moving Average Charts for Detecting Concept Drift"* Pattern Recogn. Lett. 33, 2 (Jan. 2012), 191-198 2012

Monitoring the Classification Error: EWMA

Use the **Exponential Weighted Moving Average** (EWMA) as tests statistic

EWMA statistic, which is a convex combination of current error and

average

where λ

Now, if

In stationary conditions Z_t is an estimate of p_0, since all e_i have the same expectation.

Since $\lambda \in (0,1)$, then

$(1-\lambda)^{t-i}\lambda$ decreases as i increases, thus recent samples have larger weights

…

$$Z_t = \sum (1-\lambda)^{t-i} \lambda e_i$$

G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand *"Exponentially Weighted Moving Average Charts for Detecting Concept Drift"* Pattern Recogn. Lett. 33, 2 (Jan. 2012), 191–198 2012

Monitoring the Classification Error: EWMA

Any change $\phi_0 \to \phi_1$ introduces a bias in Z_t, as it includes values in the statistic that are generated with expectation $p_1 > p_0$, the classification error after the change.

The Exponential Weighted Moving Average expression

$$Z_t = \sum (1-\lambda)^{t-i} \lambda e_i$$

Assigns much smaller weights to old samples $e_i \; i \ll t$, and is mostly influenced by recent classification errors $e_i, i \approx t$

The parameter λ (typically set in $[0.1, 0.3]$) regulates how fast the contribution of past observations decay and how quickly Z_t converges toward p_1 after the change

G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand *"Exponentially Weighted Moving Average Charts for Detecting Concept Drift"* Pattern Recogn. Lett. 33, 2 (Jan. 2012), 191–198 2012

Monitoring the Classification Error: EWMA

A natural choice for a decision rule in our settings consists in:

$$Z_t > p_0 + L\,\sigma_{Z_t}$$

Where σ_{Z_t} can corresponds to

$$\sigma_{Z_t} \; = \mathrm{std}[Z_t] = \sigma_0 \sqrt{\frac{\lambda}{2-\lambda}(1-(1-\lambda)^{2t})}$$

being σ_0 the **standard deviation of the classification error.**

This expression holds for a general EWMA monitoring scheme.

G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand *"Exponentially Weighted Moving Average Charts for Detecting Concept Drift"* Pattern Recogn. Lett. 33, 2 (Jan. 2012), 191–198 2012

Unfortunately…

Before adopting the EWMA detection rule in practice

$$Z_t > p_0 + L\,\sigma_{Z_t}$$

There are quite a few things to set

1. How to estimate p_0 and σ_0? These are typically unknown, while the monitoring scheme assume these are given!
2. How to set L to guarantee a certain ARL_0?

EWMA for Bernoulli Random Variables

The classifier error before the change has a constant expectation p_0 with the standard deviation $\sigma_0 = \sqrt{p_0(1-p_0)}$

Replace p_0 with its BLUE (Best Linear Unbiased Estimator) $\hat{p}_{0,t}$ at time t

$$\hat{p}_{0,t} = \frac{t}{t+1}\hat{p}_{0,t-1} + \frac{1}{t+1}e_t = \frac{1}{t}\sum e_i$$

Compute the corresponding variance of $\hat{p}_{0,t}$ from the formula for Bernoulli RV

$$\hat{\sigma}^2_{0,t} = \hat{p}_{0,t}(1-\hat{p}_{0,t})$$

plug this in the variance of the EWMA statistic (which indeed scales $\hat{\sigma}_{0,t}$)

$$\hat{\sigma}_{Z_t} = \hat{\sigma}_{0,t}\sqrt{\frac{\lambda}{2-\lambda}(1-(1-\lambda)^{2t})}$$

Stopping Rule for EWMA for Bernoulli

When replacing the true values p_0 and σ_0 by their estimates in the control chart we have that

$$Z_t > \hat{p}_{0,t} + L\hat{\sigma}_{Z_t}$$

To preserve a target ARL_0 **the control limit becomes time-dependent**

$$Z_t > \hat{p}_{0,t} + L_t\hat{\sigma}_{Z_t}$$

Defining the sequence $\{L_t\}_t$ is very complicated as these depend on $\hat{p}_{0,t}$.

A «simpler» problem to address via MonteCarlo simulation is, given a value L and p_0, to estimate the corresponding ARL_0

$$Montecarlo(L, p_0) \rightarrow ARL_0$$

It is also possible «to revert» this by setting up a suitable Montecarlo scheme such that, provided ARL_0 and p_0 one estimates L, as described in the paper.

Sparks, R.S., 2000. CUSUM charts for signalling varying location shifts. J. Qual. Technol. 32.

Stopping Rule for EWMA for Bernoulli

So, it is possible to estimate by Montecarlo simulations a function

$$f : (P_0, A_0) \rightarrow L$$

that returns L yielding $ARL_0 = \alpha_0$ over Bernoulli streams having $p_0 = P_0$.

This can be done by **polynomial fit in p_0 over the results of MonteCarlo simulations,** and yields **a function to be invoked at each iteration of the algorithm since $\widehat{p}_{0,t}$ does change**

$L(\hat{p}_{0,t})$

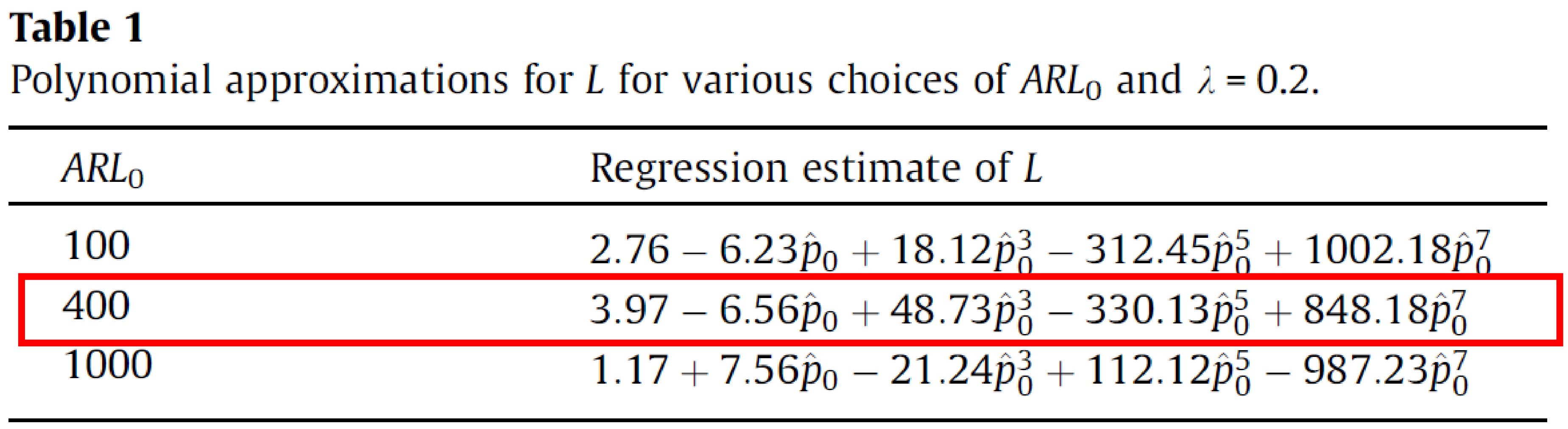


Table 1
Polynomial approximations for L for various choices of ARL_0 and $\lambda = 0.2$.

ARL_0	Regression estimate of L
100	$2.76 - 6.23\hat{p}_0 + 18.12\hat{p}_0^3 - 312.45\hat{p}_0^5 + 1002.18\hat{p}_0^7$
400	$3.97 - 6.56\hat{p}_0 + 48.73\hat{p}_0^3 - 330.13\hat{p}_0^5 + 848.18\hat{p}_0^7$
1000	$1.17 + 7.56\hat{p}_0 - 21.24\hat{p}_0^3 + 112.12\hat{p}_0^5 - 987.23\hat{p}_0^7$

Stopping Rule for EWMA for Bernoulli

Very Important: thresholds does not depend on ϕ_0

- The distribution of EWMA statistic does not depend on ϕ_0, as it monitors Bernoulli realizations that depends exclusively on p_0 ->
- The Montecarlo simulation has been done considering the above estimator of $\hat{p}_{0,t}$, use always this estimator
- Thresholds depend on the monitoring parameters ARL_0, λ

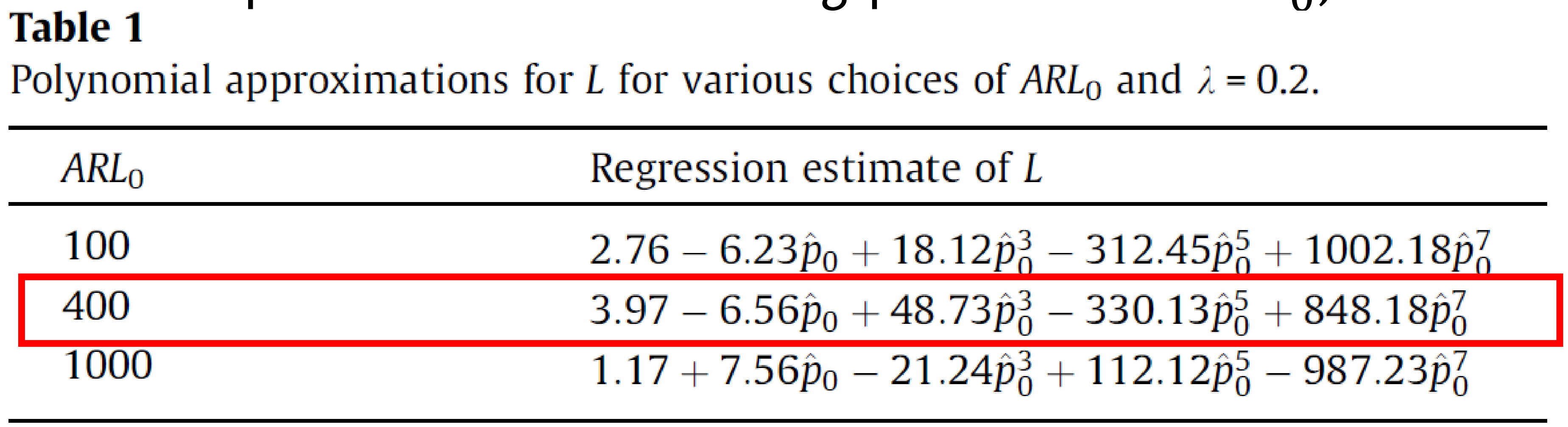


Table 1
Polynomial approximations for L for various choices of ARL_0 and $\lambda = 0.2$.

$L(\hat{p}_{0,t})$

ARL_0	Regression estimate of L
100	$2.76 - 6.23\hat{p}_0 + 18.12\hat{p}_0^3 - 312.45\hat{p}_0^5 + 1002.18\hat{p}_0^7$
400	$3.97 - 6.56\hat{p}_0 + 48.73\hat{p}_0^3 - 330.13\hat{p}_0^5 + 848.18\hat{p}_0^7$
1000	$1.17 + 7.56\hat{p}_0 - 21.24\hat{p}_0^3 + 112.12\hat{p}_0^5 - 987.23\hat{p}_0^7$

Ross, Adams, Tasoulis, Hand *"Exponentially Weighted Moving Average Charts for Detecting Concept Drift"* Pattern Recogn. Lett. 33, 2012

Adaptation in EWDMA

Like DDM, **classifier reconfiguration** is performed by monitoring Z_t also at a *warning level*

$$Z_t > p_{0,t} + 0.5\, L\left(\hat{p}_{0,t}\right)\sigma_t$$

Once CD is detected, the first sample raising a warning is used to isolate samples from the new distribution and retrain the classifier.

This is a heuristic criteria for defining a classifier update.

EWMA Monitoring for concept drift

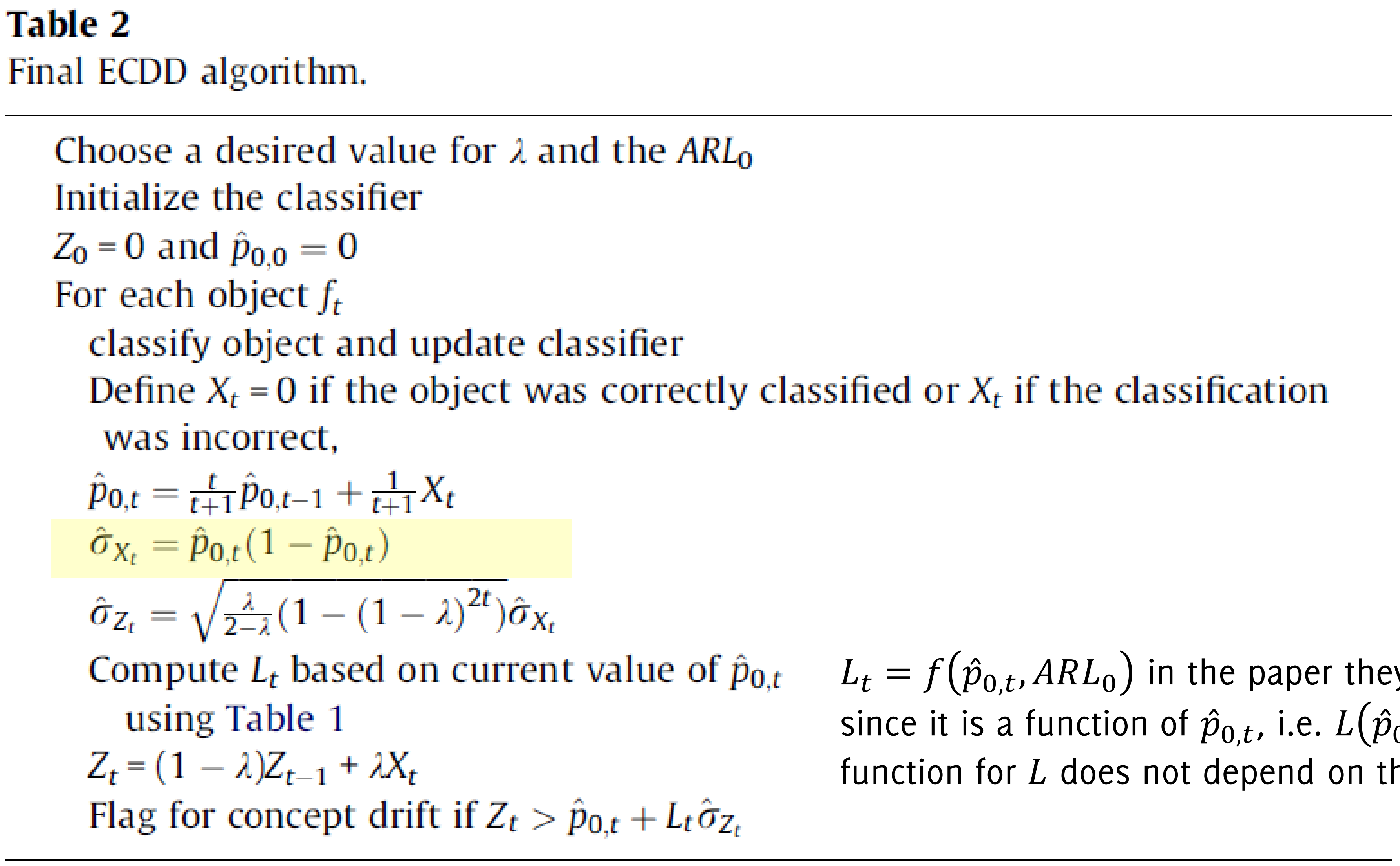


Table 2
Final ECDD algorithm.

```
Choose a desired value for $\lambda$ and the $ARL_0$
Initialize the classifier
$Z_0 = 0$ and $\hat{p}_{0,0} = 0$
For each object $f_t$
    classify object and update classifier
    Define $X_t = 0$ if the object was correctly classified or $X_t$ if the classification
     was incorrect,
    $\hat{p}_{0,t} = \frac{t}{t+1}\hat{p}_{0,t-1} + \frac{1}{t+1}X_t$
    $\hat{\sigma}_{X_t} = \hat{p}_{0,t}(1 - \hat{p}_{0,t})$
    $\hat{\sigma}_{Z_t} = \sqrt{\frac{\lambda}{2-\lambda}(1-(1-\lambda)^{2t})}\hat{\sigma}_{X_t}$
    Compute $L_t$ based on current value of $\hat{p}_{0,t}$
       using Table 1
    $Z_t = (1-\lambda)Z_{t-1} + \lambda X_t$
    Flag for concept drift if $Z_t > \hat{p}_{0,t} + L_t\hat{\sigma}_{Z_t}$
```

$L_t = f(\hat{p}_{0,t}, ARL_0)$ in the paper they refer to L_t since it is a function of $\hat{p}_{0,t}$, i.e. $L(\hat{p}_{0,t})$. The function for L does not depend on the time.

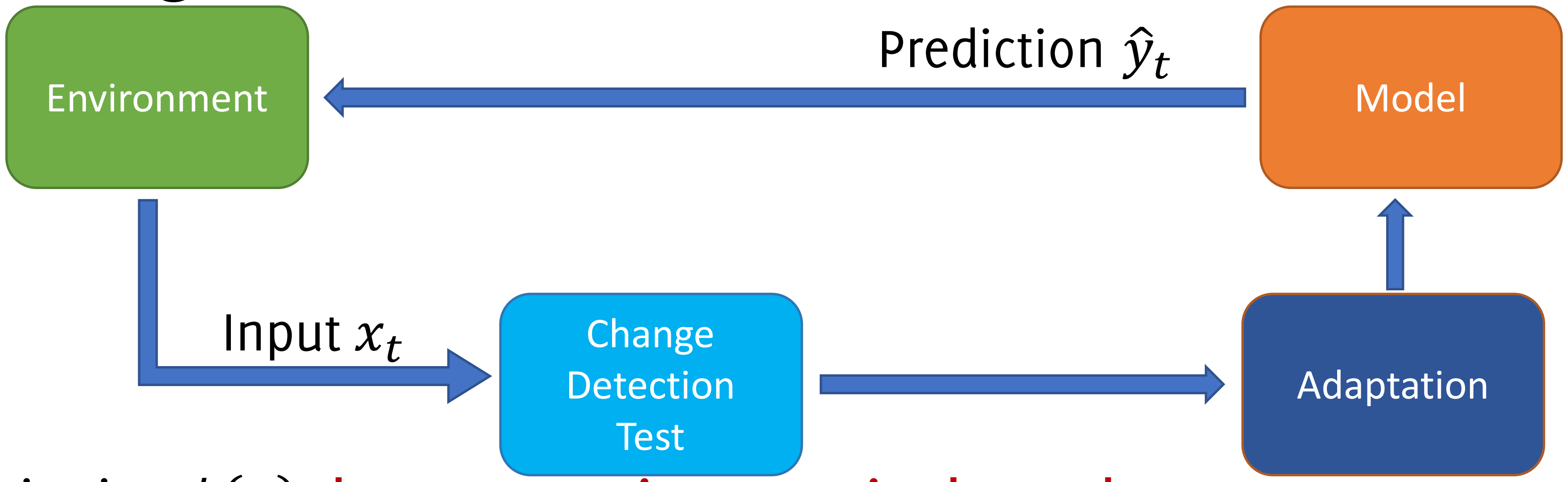
Ross, Adams, Tasoulis, Hand *"Exponentially Weighted Moving Average Charts for Detecting Concept Drift"* Pattern Recogn. Lett. 33, 2012

Monitoring the Input Distribution

… when ϕ_0 and ϕ_1 are both unknown

Change Detection Test

Monitoring Input Distribution



Pros:

- Monitoring $\phi(\boldsymbol{x})$ **does not require supervised samples**
- Enables the detection of both **real and virtual concept drift**
- **Detection before prediction**

Cons:

- CD that does not affect $\phi(\boldsymbol{x})$ are not perceivable (e.g. classes' swap)
- In principle, changes not affecting $\phi(y|\boldsymbol{x})$ do not require reconfiguration.
- Difficult to design **sequential detection tools** when streams are **multivariate** and drawn from an **unknown distribution**

Monitoring Input Distribution by Comparing Windows

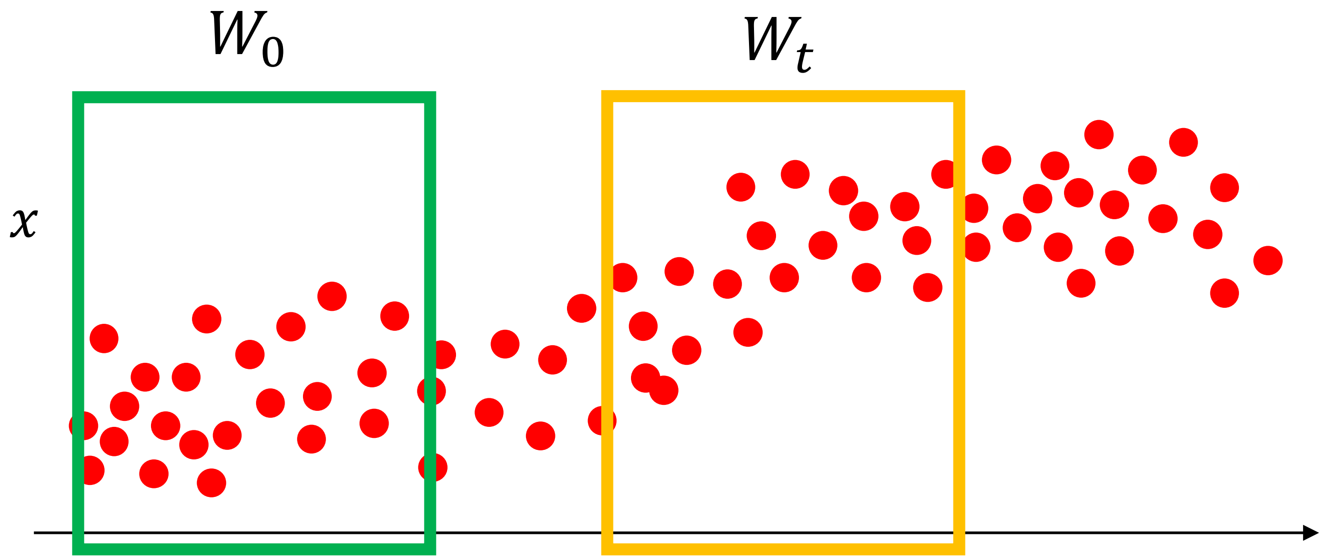
The Motivating Idea

Detect CD at time t by comparing two different windows.

In practice, one computes:

$$\mathcal{S}(W_0, W_t)$$

- W_0: reference window of past (stationary) data
- W_t: sliding window of recent (possibly changed) data
- $\mathcal{S}$ is a suitable statistic over the classification error

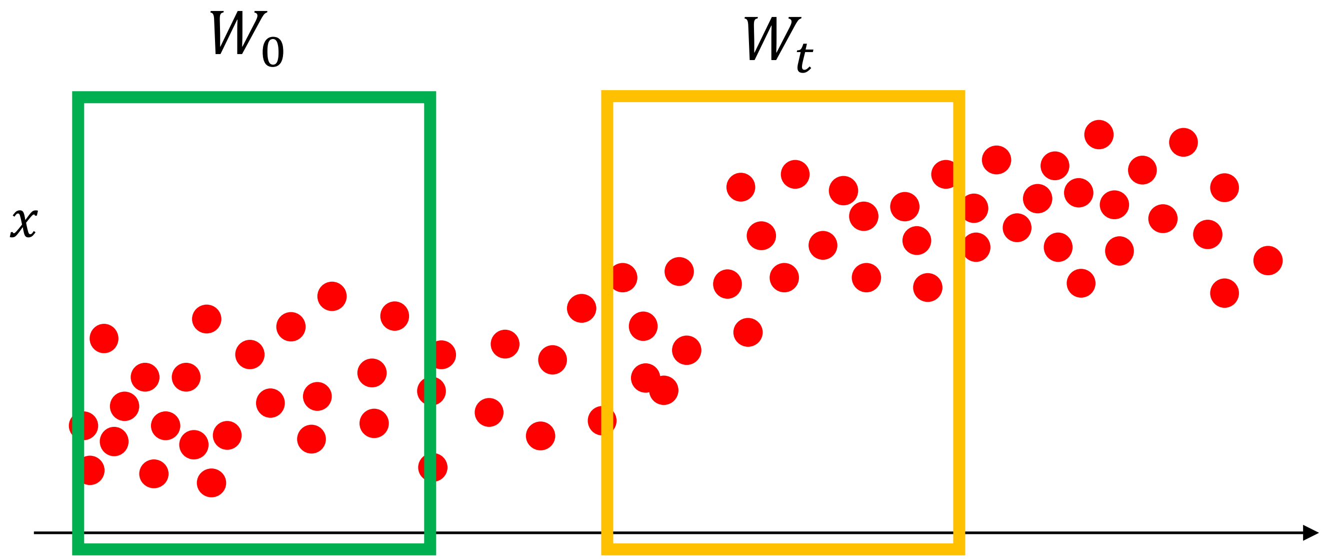


Window Comparison: Major Approaches

Hypothesis testing:

- Select W_0, a reference window from the initial concept: $W_0 \subset TR$
- As data arrives, crop a window W_t from the latest samples
- Detect concept drift by comparing an appropriate test statistic with γ

$$\mathcal{S}(W_0, W_t) \lesseqgtr \gamma$$



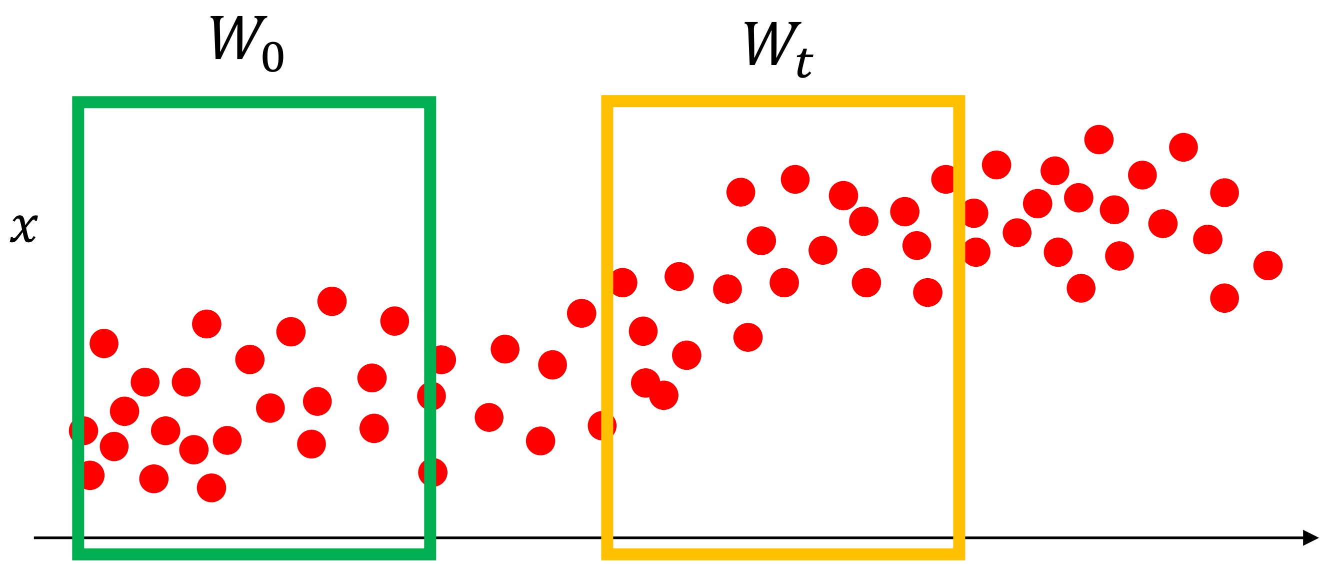
Window Comparison: Major Approaches

Hypothesi

- Select
- As dat
- Detect γ

Tackle the change-detection problem as anomaly detection on batches cropped from the datastream

Example of statistics: t-test (Hotelling t-square) statistics for detecting shifts in the batch expectation for scalar (multivariate) streams, respectively



Window Comparison: Major Approaches

ADWIN: Compare the averages of scalar inputs over two adjacent windows having increasing size.

Whenever two *"large enough"* subwindows of the stream exhibit *"distinct enough"* averages, detect a change and drop the old samples in W.

$W = 101010110111111$

ADWIN: ADAPTIVE WINDOWING

```
• Initialize Window $W$
• for each $t > 0$ do
    $W \leftarrow W \cup \{x_t\}$ (add $x_t$ to the head of $W$)
    • repeat Drop elements from the tail of $W$
        until $|\mu_0 - \mu_1| < \epsilon$ holds for every split
    of $W$ into $W = [W0, W1]$
• Output $\mu_W$
```

Window Comparison: Major Approaches

ADWIN: Compare the averages of scalar inputs over two adjacent windows having increasing size.

Whenever two *“large enough”* subwindows of the stream exhibit *“distinct enough”* averages, detect a change and drop the old samples in W.

ADWIN: ADAPTIVE WINDOWING

```
• Initialize Window $W$
• **for each** $t > 0$ **do**
    $W \leftarrow W \cup \{x_t\}$ (add $x_t$ to the head of $W$)
    • **repeat** Drop elements from the tail of $W$
        **until** $|\mu_0 - \mu_1| < \epsilon$ holds for every split
    of $W$ into $W = [W0, W1]$
• Output $\mu_W$
```

$$W = 101010110111111$$

$$W_0 = \boxed{\underset{\mu_0}{1,}} \qquad W_1 = \boxed{\underset{\mu_1}{01010110111111}}$$

Window Comparison: Major Approaches

ADWIN: Compare the averages of scalar inputs over two adjacent windows having increasing size.

Whenever two *"large enough"* subwindows of the stream exhibit *"distinct enough"* averages, detect a change and drop the old samples in W.

ADWIN: ADAPTIVE WINDOWING

```
• Initialize Window $W$
• **for each** $t > 0$ **do**
    $W \leftarrow W \cup \{x_t\}$ (add $x_t$ to the head of $W$)
    • **repeat** Drop elements from the tail of $W$
        **until** $|\mu_0 - \mu_1| < \epsilon$ holds for every split
    of $W$ into $W = [W0, W1]$
• Output $\mu_W$
```

$$W = 101010110111111$$

$$W_0 = 1, \quad W_1 = 01010110111111$$

$$W_0 = 10, \quad W_1 = 1010110111111$$

$$\dots$$

$$W_0 = 101010110, \quad W_1 = 111111$$

$$|\mu_0 - \mu_1| \geq \epsilon$$

ADWIN2: efficient variant reducing computation and memory footprint

Bifet A., Gavaldà R. "Learning from time-changing data with adaptive windowing" SIAM Int. Conference on Data Mining 2007

Window Comparison: Major Approaches

1. **Hypothesis testing**
2. **ADWIN**: Compare the averages of scalar inputs over two adjacent windows
3. Compute **empirical distributions** of raw data over W_0 and W_t and compare
   - The Kullback-Leibler divergence

T. Dasu, Sh. Krishnan, S. Venkatasubramanian, and K. Yi. *"An Information-Theoretic Approach to Detecting Changes in Multi-Dimensional Data Streams"*. In Proc. of the 38th Symp. on the Interface of Statistics, Computing Science, and Applications, 2006

Window Comparison: Major Approaches

1. **Hypothesis testing**
2. **ADWIN**: Compare the averages of scalar inputs over two adjacent windows
3. Compute **empirical distributions** of raw data over W_0 and W_t and compare
   - The Kullback-Leibler divergence
   - The Hellinger distance

G. Ditzler and R. Polikar, *"Hellinger distance based drift detection for nonstationary environments"* in Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2011 IEEE Symposium on, April 2011, pp. 41–48.

Window Comparison: Major Approaches

1. **Hypothesis testing**
2. **ADWIN**: Compare the averages of scalar inputs over two adjacent windows
3. Compute **empirical distributions** of raw data over W_0 and W_t and compare
   - The Kullback-Leibler divergence
   - The Hellinger distance
   - The density ratio over the two windows using kernel methods (to overcome curse of dimensionality problems when computing empirical distributions)

Kawahara, Y. and Sugiyama, M. *"Sequential change-point detection based on direct density-ratio estimation"*. Statistical Analysis and Data Mining, 5(2):114–127, 2012.

Other Schemes for Monitoring the Input Distribution

Change Detection Approaches

- The Change-Point Formulation
  - Parametric
  - Non-parametric
- Change-Detection by Monitoring Features / the Log-likelihood
- Change-Detection by Histograms

Change Detection in Parametric Settings: CPM

Change-Point Methods (CPM) are **sequential** monitoring schemes that **extend** traditional **parametric hypothesis tests.**

Parametric settings: ϕ_0 and ϕ_1 are known up to their parameters (θ_0 and θ_1), thus the change $\phi_0 \to \phi_1$ corresponds to a change $\theta_0 \to \theta_1$

Non-Parametric settings: Both ϕ_0 and ϕ_1 are unknown, the change $\phi_0 \to \phi_1$ is completely unpredictable

Pro: CPMs do not require training samples

Pro: They provide fixed ARL_0

Con: These nonparametric statistics are not meant for multivariate data.

Hawkins, D. M., and Zamba, K. D. *"Statistical process control for shifts in mean or variance using a changepoint formulation"* Technometrics 2005

Ross, G. J. "Sequential change detection in the presence of unknown parameters". Statistics and Computing, 24(6), 1017-1030, 2014

Change Detection in Parametric settings: CPM

In Statistical Process Control, monitoring is divided in two phases:

- **Offline / Phase I:** Given a sequence $\{x_t\}$, determine whether it contains a change point τ or not. This is "one-shot test"
- **Online / Phase II:** data arrive steadily, and decision has to be taken as data flows (online).

We illustrate first the basic CPM scheme in offline monitoring, then we show how a Phase I mechanism can be **iterated to perform online change detection** (sequential monitoring).

Hawkins, D. M., and Zamba, K. D. *"Statistical process control for shifts in mean or variance using a changepoint formulation"* Technometrics 2005

Ross, G. J. "Sequential change detection in the presence of unknown parameters". Statistics and Computing, 24(6), 1017-1030, 2014

The Changepoint Model for Statistical Process Control

DOUGLAS M. HAWKINS and PEIHUA QIU

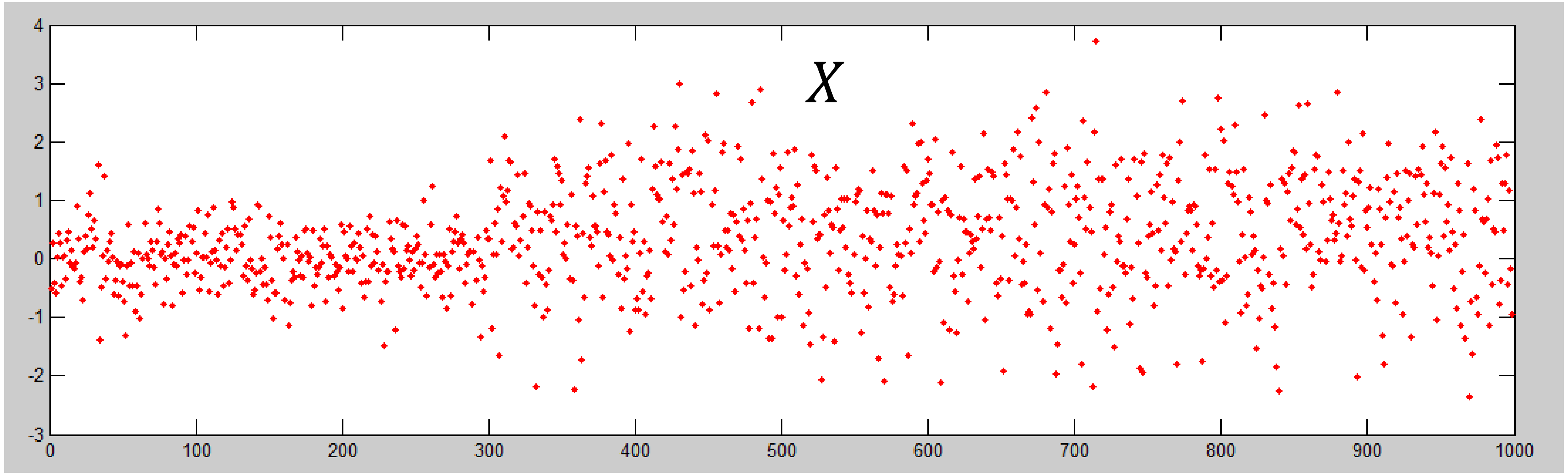
University of Minnesota, Minneapolis, MN 55455

CHANG WOOK KANG

Hanyang University, Seoul, Korea

D. M. Hawkins, P. Qiu, and C. W. Kang, “The changepoint model for statistical process control,” Journal of Quality Technology, vol. 35, No. 4, pp. 355–366, 2003.

The Change Point Method (CPM)

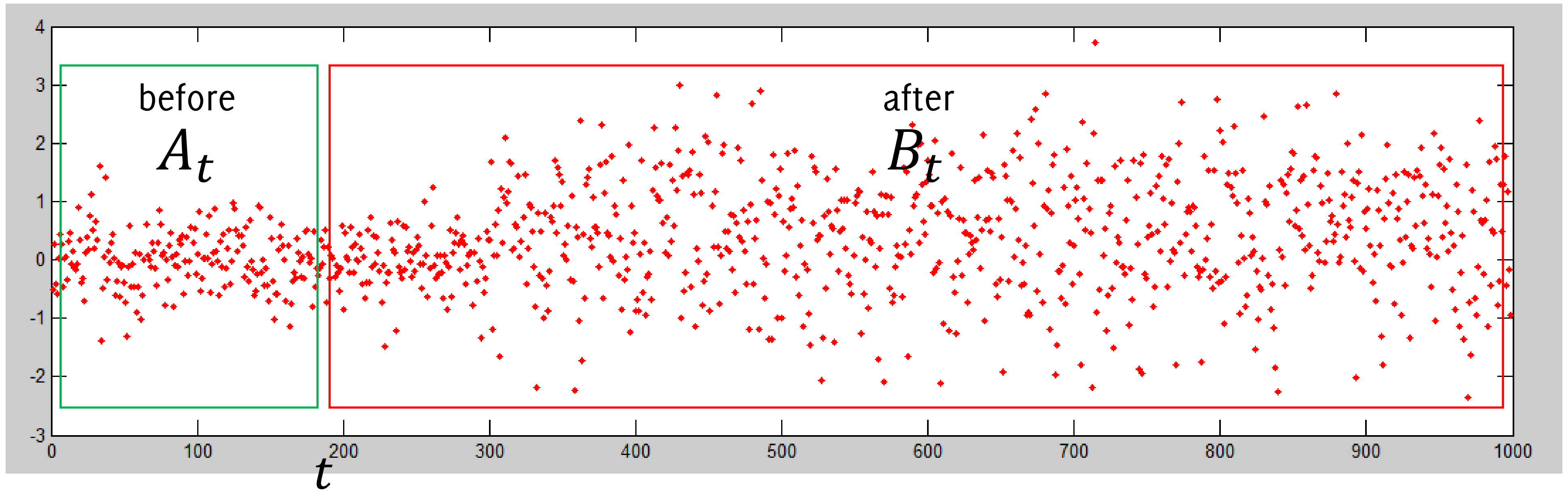


Assume a sequence X of 1000 points is given and we want to find the change point τ inside (offline analysis)

$$\boldsymbol{x}(t) \sim \begin{cases} \phi_0 & t < \tau \\ \phi_1 & t \geq \tau \end{cases}$$

Assume we are given a statistic $\mathcal{S}_t$ to compare two datasets $A_t, B_t \subset X$ coming before and after t

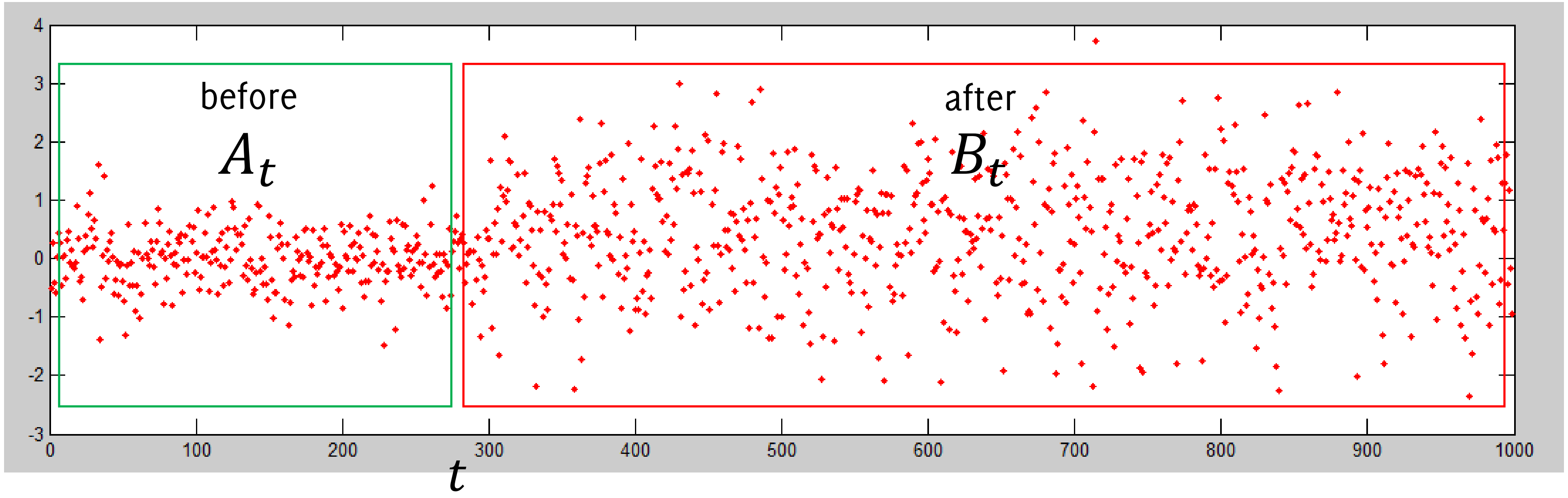
The Change Point Method (CPM)



- Test a single point t to be a change point
- Split the dataset in two sets $A_t, B_t \subset X$, namely samples «before» and «after» the putative change at t
- **Compute a test statistic** $\mathcal{S}_t = \mathcal{S}(A_t, B_t)$ to determine whether the two sets are from the same distribution (e.g. same mean)
- **Repeat the procedure** $t = \delta, \dots, 1000 - \delta$ and store the value of the statistic

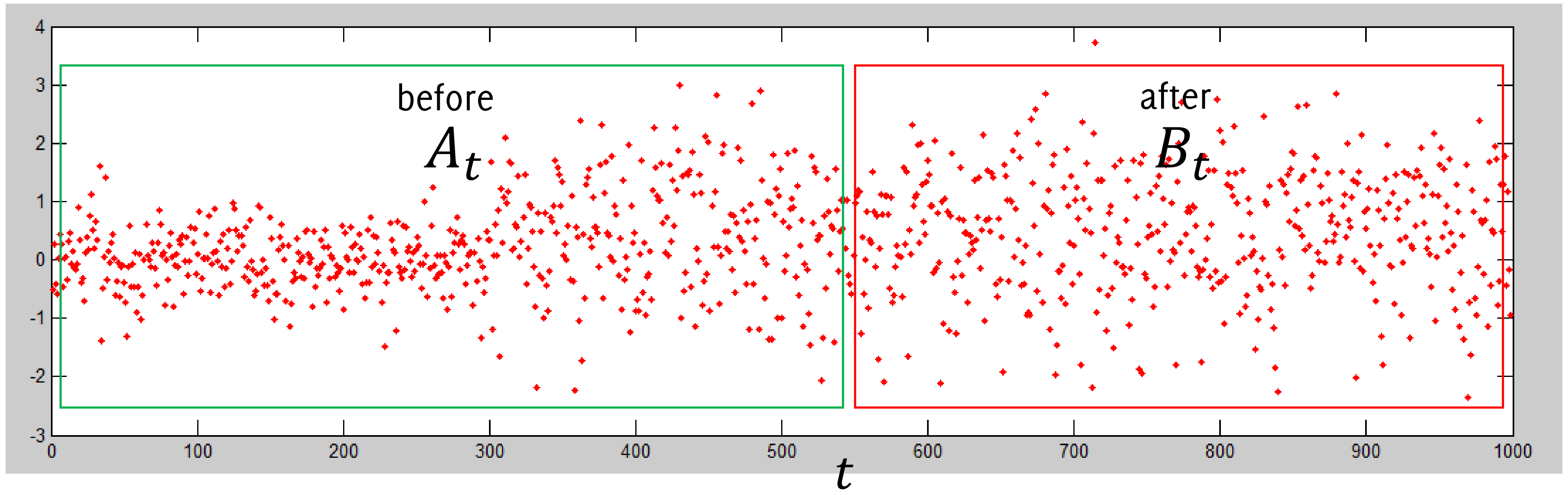
Boracchi

The Change Point Method (CPM)



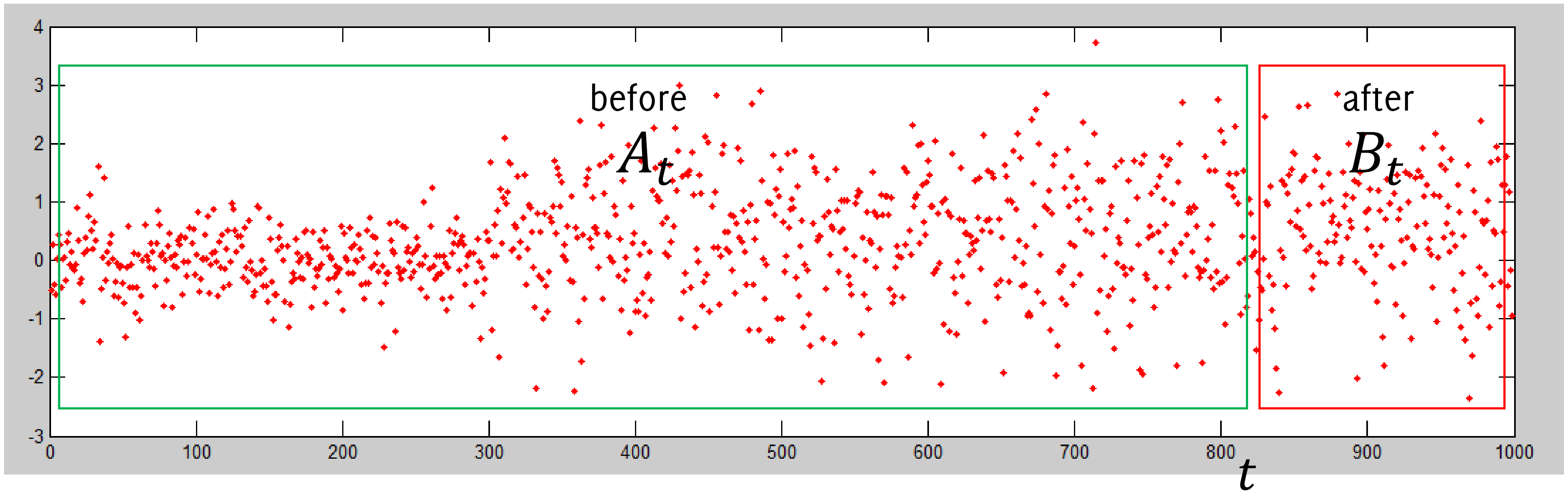
- Test a single point t to be a change point
- Split the dataset in two sets $A_t, B_t \subset X$, namely samples «before» and «after» the putative change at t
- **Compute a test statistic** $\mathcal{S}_t = \mathcal{S}(A_t, B_t)$ to determine whether the two sets are from the same distribution (e.g. same mean)
- **Repeat the procedure** $t = \delta, \dots, 1000 - \delta$ and store the value of the statistic

The Change Point Method (CPM)



- Test a single point t to be a change point
- Split the dataset in two sets $A_t, B_t \subset X$, namely samples «before» and «after» the putative change at t
- **Compute a test statistic** $\mathcal{S}_t = \mathcal{S}(A_t, B_t)$ to determine whether the two sets are from the same distribution (e.g. same mean)
- **Repeat the procedure** $t = \delta, \dots, 1000 - \delta$ and store the value of the statistic

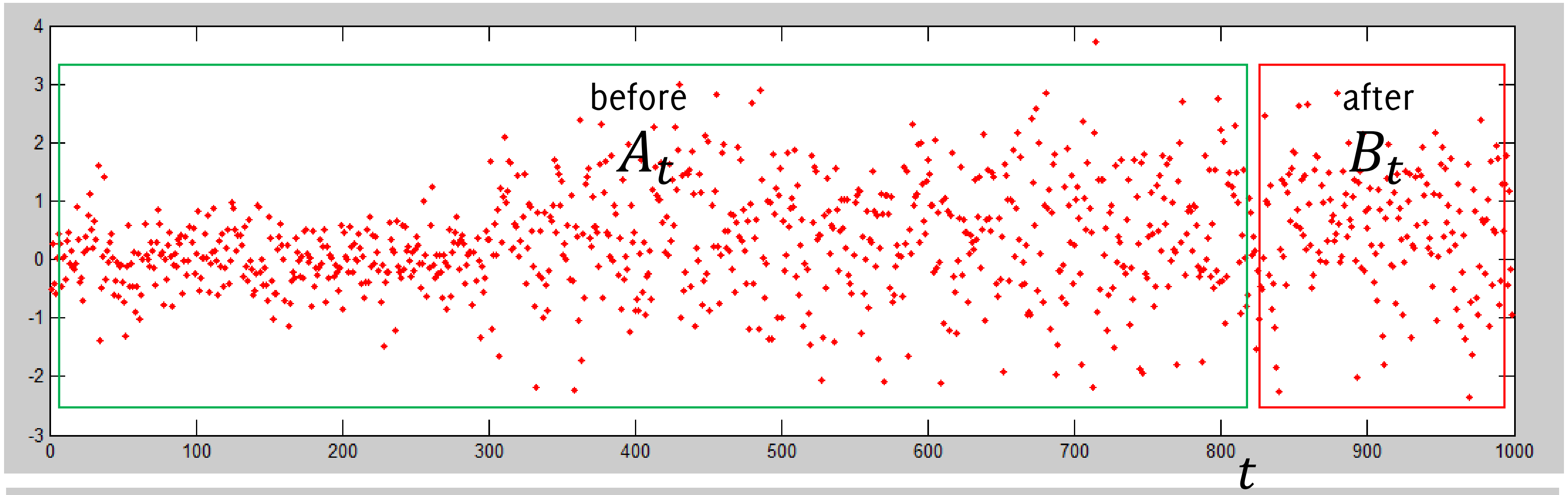
The Change Point Method (CPM)

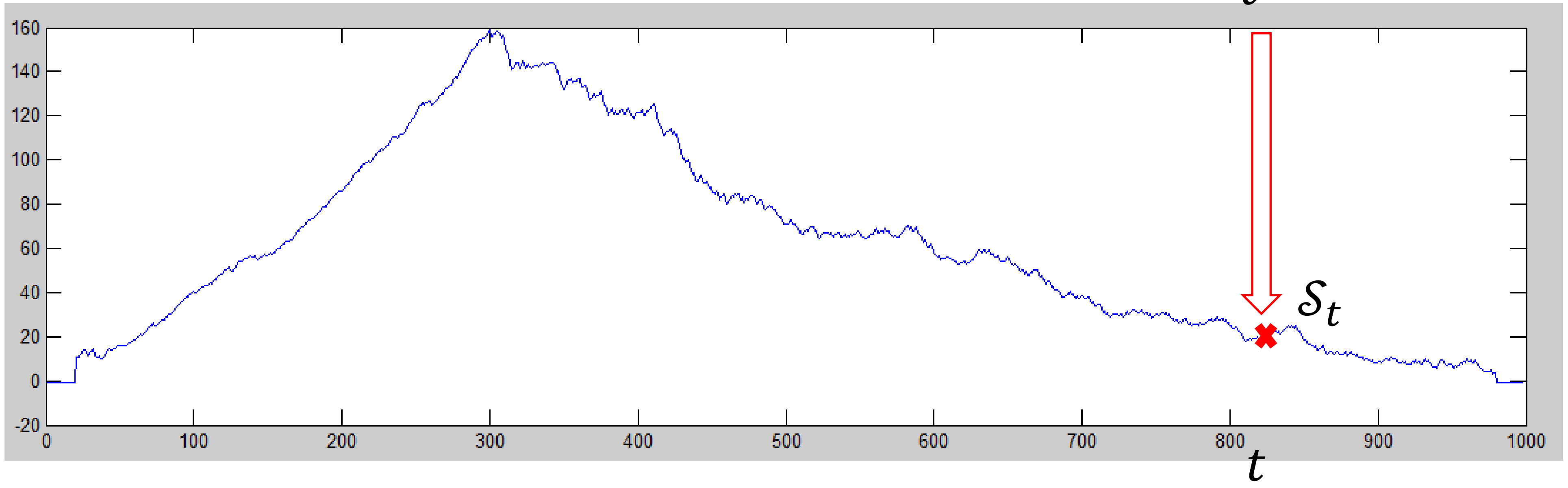


- Test a single point t to be a change point
- Split the dataset in two sets $A_t, B_t \subset X$, namely samples «before» and «after» the putative change at t
- **Compute a test statistic** $\mathcal{S}_t = \mathcal{S}(A_t, B_t)$ to determine whether the two sets are from the same distribution (e.g. same mean)
- **Repeat the procedure** $t = \delta, \dots, 1000 - \delta$ and store the value of the statistic

Boracchi

The Change Point Method (CPM)





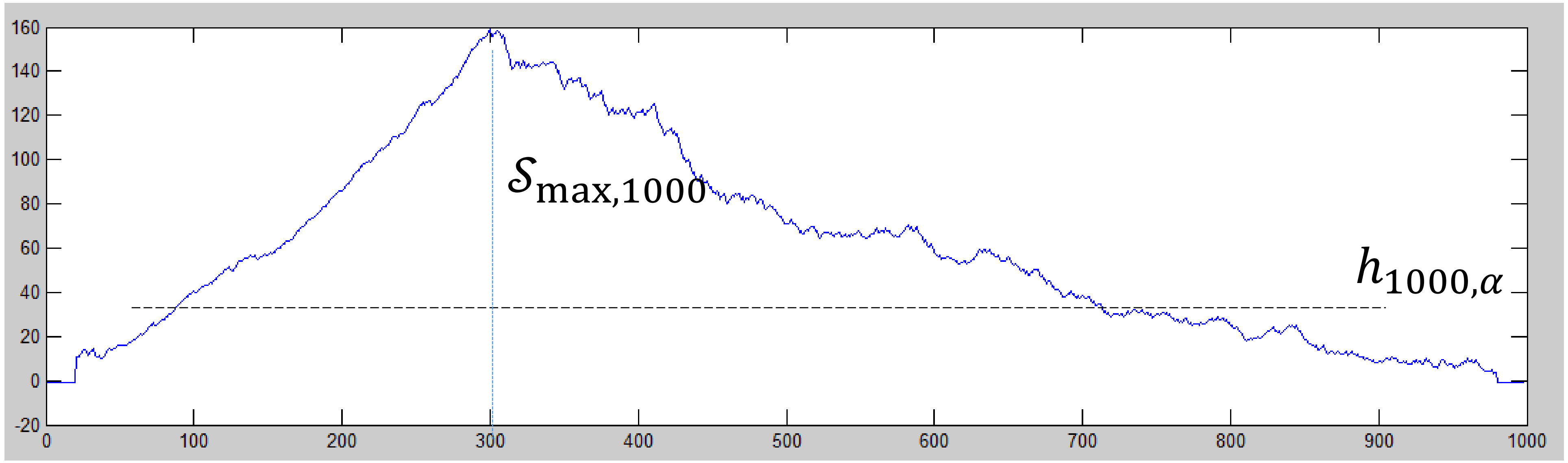
Boracchi

The Change Point Method (CPM)

The point where the **statistic achieves its maximum** is the most likely position of the change-point

As in hypothesis testing, it possible to **set a threshold** $h_{1000,\alpha}$ for $\mathcal{S}_{\max,1000}$ by setting to α the **probability of type I errors.**

The CPM framework can be extended to online monitoring, and in this case it is possible to control the ARL_0



The CPM Formulation

Phase I (offline): test all the possible splits $\forall\, t \in [1, N]$, being $N = \#X$

- Define $A_t = \{x(u), 0 \leq u < t\}$ and $B_t = \{x(u), t \leq u \leq N\}$
- Compute the test statistic

$$\mathcal{S}_t = \mathcal{S}(A_t, B_t)$$

- We claim that $\{x(t)\}_t$ contains a change point when

$$\mathcal{S}_{\max,N} = \max_t(\mathcal{S}_t) > \gamma_N$$

The threshold γ_N has to be set to control type I errors under $H_0 : X \sim \phi_0$

- The estimated change point location is

$$\hat{\tau} = \operatorname*{argmax}_t(\mathcal{S}_t) > \gamma_N$$

Threshold Computation (Offline Analysis)

Finding a threshold γ_N guaranteeing control over type I error is not trivial, as this depends on **the distribution of $\boldsymbol{\mathcal{S}_{max,N}}$** under $X \sim \phi_0$

Rmk: the distribution of $\mathcal{S}_{\max,N}$ is very complicated due to the **high correlation between the $\{\boldsymbol{\mathcal{S}_{t,N}}\}$ statistics**.

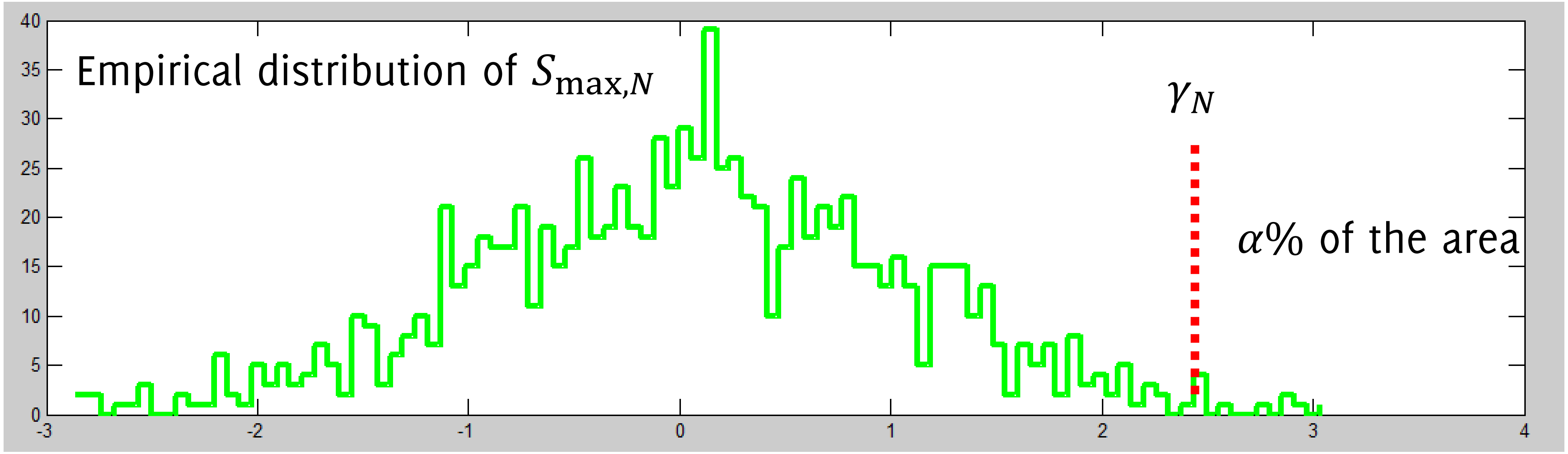
Other options:

- Bonferroni approximations for $\mathcal{S}_{max,N}$, but this is too loose an approximation: there are many comparisons (N), one per sample
- Asymptotic bounds for $\mathcal{S}_{max,N}$, but these are only available for certain statistics $\mathcal{S}$, thus wouldn't apply to all distribution ϕ_0 (and would possibly yield a coarse approximation at early monitoring stages)
- **Resort to MonteCarlo simulations**

Threshold Computation (offline analysis)

Therefore we resort to bootstrap.

- Draw many (?!) sequences $X \sim \phi_0$
- Compute the statistic $\mathcal{S}_{\max,N}$ for each sequence and store their values
- Set the threshold as the quantile of this empirical distribution

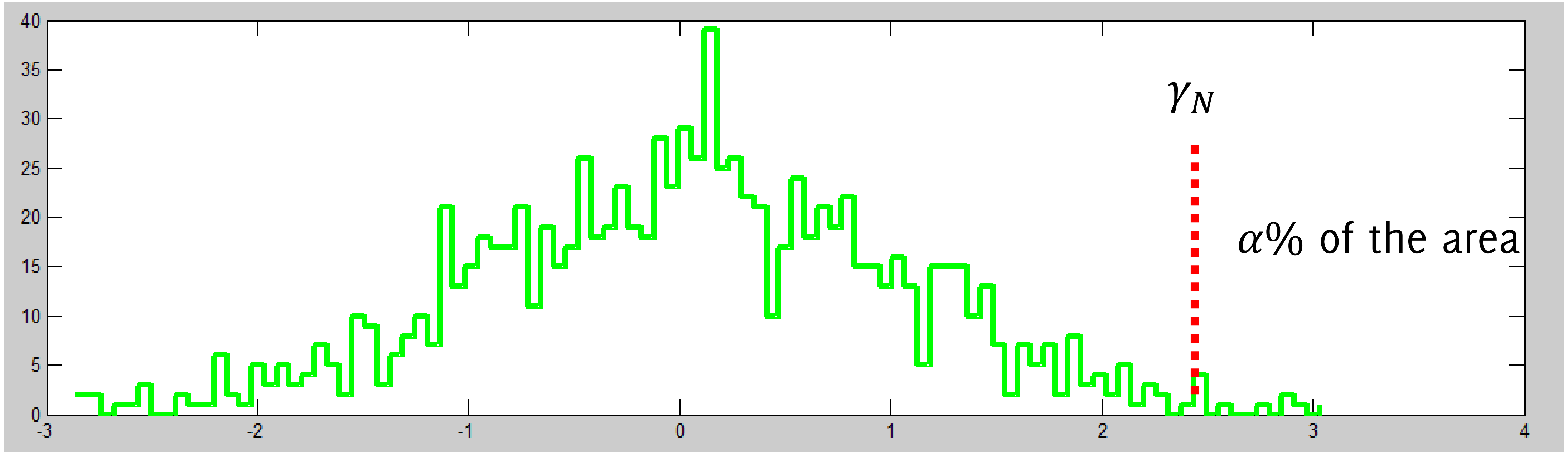


Threshold Computation

The computed thresholds depends on many factors:
- The distribution of input data ϕ_0
- The length of the sequence N
- The target FPR α

Therefore we resort to bootstrap.

- Draw many (?!) sequences $X \sim \phi_0$
- Compute the statistic $\mathcal{S}_{\max,N}$ for each sequence and store their values
- Set the threshold as the quantile of this empirical distribution



Threshold Computation

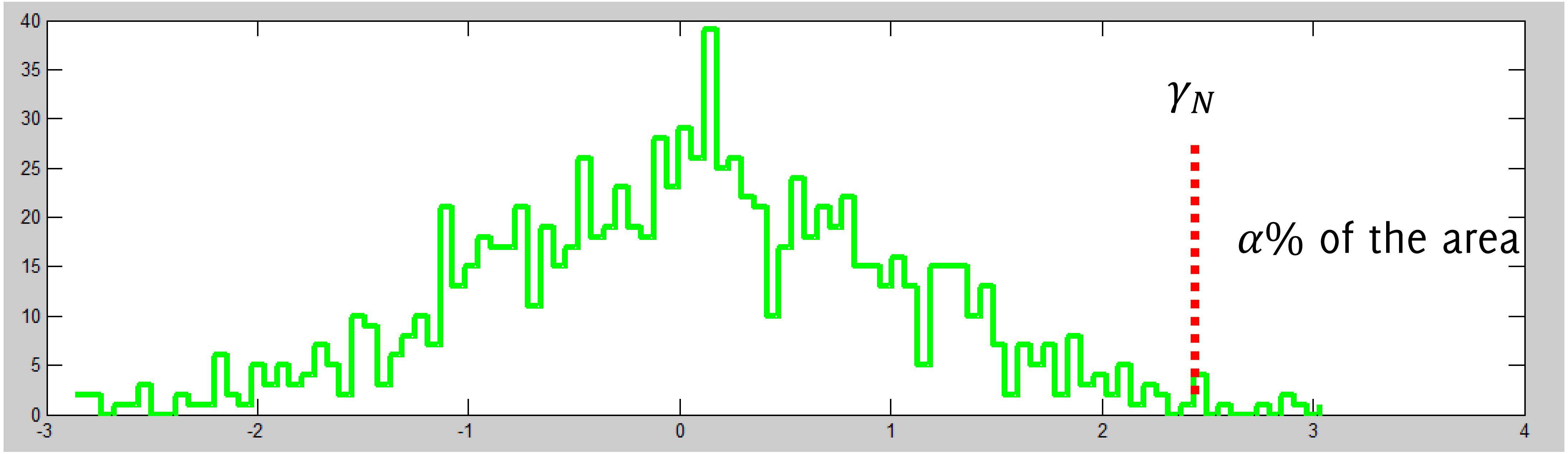
Therefore we resort to bootstrap.

- Draw many (?!) sequences $X \sim$
- Compute the statistic $\mathcal{S}_{\max,N}$ for
- Set the threshold as the quantile of this empirical distribution

The computed thresholds depends on many factors:

- The distribution of input data ϕ_0
- The length of the sequence N
- The target FPR α

The same bootstrap procedure has to be repeated for each ϕ_0 (TR) and N



Boracchi

Nonparametric Monitoring of Data Streams for Changes in Location and Scale

Gordon J. Ross, Dimitris K. Tasoulis, and Niall M. Adams

Department of Mathematics
Imperial College London
London, SW7 2AZ, U.K.
(*gordon.ross03@imperial.ac.uk*; *d.tasoulis@imperial.ac.uk*;
n.adams@imperial.ac.uk)

G. J. Ross, D. K. Tasoulis, and N. M. Adams, "Nonparametric monitoring of data streams for changes in location and scale," Technometrics, vol. 53, no. 4, pp. 379–389, 2011.

Nonparametric Monitoring of Data Streams for Changes in Location and Scale

Gordon J. Ross, Dimitris K. Tasoulis, and Niall M. Adams

Department of Mathematics
Imperial College London
London, SW7 2AZ, U.K.
(*gordon.ross03@imperial.ac.uk*; *d.tasoulis@imperial.ac.uk*;
n.adams@imperial.ac.uk)

G. J. Ross, D. K. Tasoulis, and N. M. Adams, “Nonparametric monitoring of data streams for changes in location and scale,” Technometrics, vol. 53, no. 4, pp. 379–389, 2011.

CPM in non-parametric settings

Any statistics for HT could be used in both online and offline change-point methods. A better option would be to adopt **nonparametric statistics**, like:

- Mann-Whitney,
- Mood,
- Lepage,
- Two sample Kolmogorov-Smirnov,
- Cramer von Mises,

which do not require **any information** about ϕ_0 or ϕ_1.

A relevant advantage: sequences for computing the threshold can be generated by an arbitrarily distribution ψ, as the test statistic S does not depend on ϕ_0. **Synthetic data generation rather than bootstrap**

Ross, G. J., Tasoulis, D. K., Adams, N. M. *"Nonparametric monitoring of data streams for changes in location and scale"* Technometrics, 53(4), 379-389, 2012.

CPM in non-parametric settings

The (two samples) **Kolmogorov Smirnov and Cramer Von Mises** are very general test statistics, as they assess variations in the empirical distribution of data.

However, these "omnibus" tests have **low power**, and it is better to focus on statistics detecting specific types changes in ϕ_0

- *Location Changes: i.e.,* $\phi_1(x) = \phi_0(x + \delta)$
- *Scale Changes: i.e.,* $\phi_1(x) = \phi_0(\delta x)$

In practice it is very unlikely that $\phi_1$and $\phi_0$would differ while having the same expectation and variance.

Nonparametric Statistics for Scale and Location

Most of nonparametric statistics ranks the observations

$$rk\big(x(i)\big) = \sum_{i \neq j} I(x(i) > x(j))$$

The **Mann-Whitney** statitic to assess location changes between two sets

The **Mood** statistic to assess scale changes between two sets

Both Mann-Withney and Mood statistics:

- Can be used to compare two sets A, B
- **Are independent from** ϕ_0 **the distribution of the observations** $x(t)$

Mann-Whitney Statistic for two sets A and B

The idea

When N i.i.d. samples are spread over two sets A and B, the expectation of the sum in in A of ranks computed over $[A, B]$, should be like "the average rank" over $[A, B]$

$$E\left[\sum_{x(t)\in A} r(x(t))\right] = \#A \ * \frac{(\#[A,B]+1)}{2}$$

The U statistic measures how much the sum in A of ranks over $[A, B]$

$$\sum_{x(t)\in A} r(x(t))$$

deviates from $\#A \ * \frac{(\#[A,B]+1)}{2}$

```
m = length(A(:)); n = length(B(:));
N = m + n;

% row vector containing both dataset
D = [A(:); B(:)]';
% labels,
L = [ones(1, m)  , zeros(1, n)];
[~, indx] = sort(D);
V = L(indx);
xx =[1 : size(D, 2)]; % ranks

% U: Wilcoxon / Mann-Whitney statistic
U = xx * V';

%% compute normalization terms
mu = m * (N + 1) / 2;
sigma =  m * n * (N + 1)  / 12;

%% compute the normalized test statistic
U = abs(U - mu) / sqrt(sigma);
```

Boracchi

Mann-Whitney Statistic for two sets A and B

The idea

When N i.i.d. samples are spread over two sets A and B, the expectation of the sum in in A of ranks computed over $[A, B]$, should be like “the average rank” over $[A, B]$

$$E\left[\sum_{x(t)\in A} r(x(t))\right] = \#A \ * \frac{(\#[A,B]+1)}{2}$$

The U statistic measures how much the sum in A of ranks over $[A, B]$

$$\sum_{x(t)\in A} r(x(t))$$

deviates from $\#A \ * \frac{(\#[A,B]+1)}{2}$

```
m = length(A(:)); n = length(B(:));
N = m + n;

% row vector containing both dataset
D = [A(:); B(:)]';
% labels,
L = [ones(1, m) , zeros(1, n)];
[~, indx] = sort(D);
V = L(indx);
xx =[1 : size(D, 2)]; % ranks

% U: Wilcoxon / Mann-Whitney statistic
U = xx * V';

%% compute normalization terms
mu = m * (N + 1) / 2;
sigma =  m * n * (N + 1)  / 12;

%% compute the normalized test statistic
U = abs(U - mu) / sqrt(sigma);
```

Boracchi

Mann-Whitney Statistic for two sets A and B

The idea

When N i.i.d. samples are spread over two sets A and B, the expectation of the sum in in A of ranks computed over $[A, B]$, should be like "the average rank" over $[A, B]$

$$E\Bigg[\sum_{x(t)\in A} r(x(t$$

When $A \sim \phi_0$ and $B \sim \phi_0(\cdot - \delta)$, the ranks of elements in B will be larger $(\delta > 0)$ or smaller $(\delta < 0)$ than those in A

The U statistic measures how much the sum in A of ranks over $[A, B]$

$$\sum_{x(t)\in A} r(x(t))$$

deviates from $\#A * \frac{(\#[A,B]+1)}{2}$

```
m = length(A(:)); n = length(B(:));
N = m + n;

% row vector containing both dataset
D = [A(:); B(:)]';
% labels,
L = [ones(1, m) , zeros(1, n)];
                                      ;

                                      ]; % ranks

% U: Wilcoxon / Mann-Whitney statistic
U = xx * V';

%% compute normalization terms
mu = m * (N + 1) / 2;
sigma =  m * n * (N + 1)  / 12;

%% compute the normalized test statistic
U = abs(U - mu) / sqrt(sigma);
```

Boracchi

Mood Statistic for two sets A and B

The idea

When N i.i.d. samples are divided in two sets A and B, then

$$E[r(x(t))] = \frac{N+1}{2}$$

the expected rank of each point under $H_0 =$ "both sets are identically distributed" is $(N+1)/2$

M measures the (squared) deviation of ranks of samples in A from this expectation

```
m = length(A(:)); n = length(B(:));
N = m + n;

% row vector containing both dataset
D = [A(:); B(:)]';

% compute the rank
[vs, vi] = sort(D);
[x, r] = sort(vi);

% Mood Statistic,
M = sum((r(1 : m) - (N + 1) / 2).^2);

% Expectation of Mood Stats
mu = m * (N^2 - 1) / 12;

% Standard deviation of Mood Stats
sigma =  m*n*(N + 1)*(N - 2)* (N+2) / 180;

%% compute the normalized test statistic
M = abs((M - mu)) / sqrt(sigma);
```

Mood Statistic for two sets A and B

The idea

When N i.i.d. samples are divided in two sets A and B, then

$$E[r($$ $$N+1$$

the expected r

$H_0 =$ "both se

distributed" is

M measures the (squared) deviation of ranks of samples in A from this expectation

When $A \sim \phi_0(\cdot)$ and $B \sim \phi_0(\delta\ \cdot)$, the ranks of elements in B will be more extreme $(\delta > 1)$ or condensed $(\delta < 1)$ than those in A.
This results in a larger/smaller variance of ranks, which corresponds to larger values of M statistics

```
m = length(A(:)); n = length(B(:));
N = m + n;

% row vector containing both dataset
D = [A(:); B(:)]';

% compute the rank

                                       (N + 1) / 2).^2);

                                     d Stats
mu = m * (N^2 - 1) / 12;

% Standard deviation of Mood Stats
sigma =  m*n*(N + 1)*(N - 2)* (N+2) / 180;

%% compute the normalized test statistic
M = abs((M - mu)) / sqrt(sigma);
```

Boracchi

How to monitor for both Location and Scale Changes?

In practice we don't know if ϕ_0 and ϕ_1 would differ because of location or scale changes

Using Mood and Mann-Whitney in parallel makes difficult to control the ARL_0 (or type I error in the offline scenario)

Better to monitor location and scale jointly: use the **Lepage Test statistic**

$$L = U^2 + W^2$$

CPM for Online Monitoring

Observations arrive steadily,

$$x(1), \dots, x(N), \dots$$

possibly forming an infinite stream

At each new arrival, a Change-Point Method (CPM) assesses if the distribution of the observations differs from the previous samples.

The primary issue is the **detection**, but the CPM monitoring scheme performs also the **estimation of change point location**, once the detection is signalled.

In fact any online CPM returns

- $\widehat{T}$, the time instant when the change is detected,
- $\hat{\tau}$, the estimate of the change time-instant

Two Issues in CPMs for Online Monitoring

In principle, one may **iterate the offline approach** presented before – at each new arrival.

Two issues:

- How to compute the thresholds?
- Iterating CPM becomes time and resources demanding..

Even if we compute γ_N for the offline analysis, these thresholds would not be appropriate for online analysis

Threshold Computation: Online CPM

Quantiles of test statistic $\mathcal{S}_{\max,t}$ used for offline analysis cannot be used, since γ_t **has to be set controlling the conditional probability** that

$$P\left(\mathcal{S}_{\max,t} > \gamma_t \mid \mathcal{S}_{\max,t-1} < \gamma_{t-1}, \dots, \mathcal{S}_{\max,1} < \gamma_1\right) < \alpha$$

Still, one may resort to numerical simulations to compute them **in a sequential manner**.

A few methods can set the false alarm probability (FAP) to be the same in each point, that is, $P(\mathcal{S}_{\max,t} > \gamma_t \mid x \sim \phi_0) = \alpha$ for all t,

then the ARL_0 relates to α as

$$\alpha = \frac{1}{ARL_0}$$

Threshold Computation: Online CPM

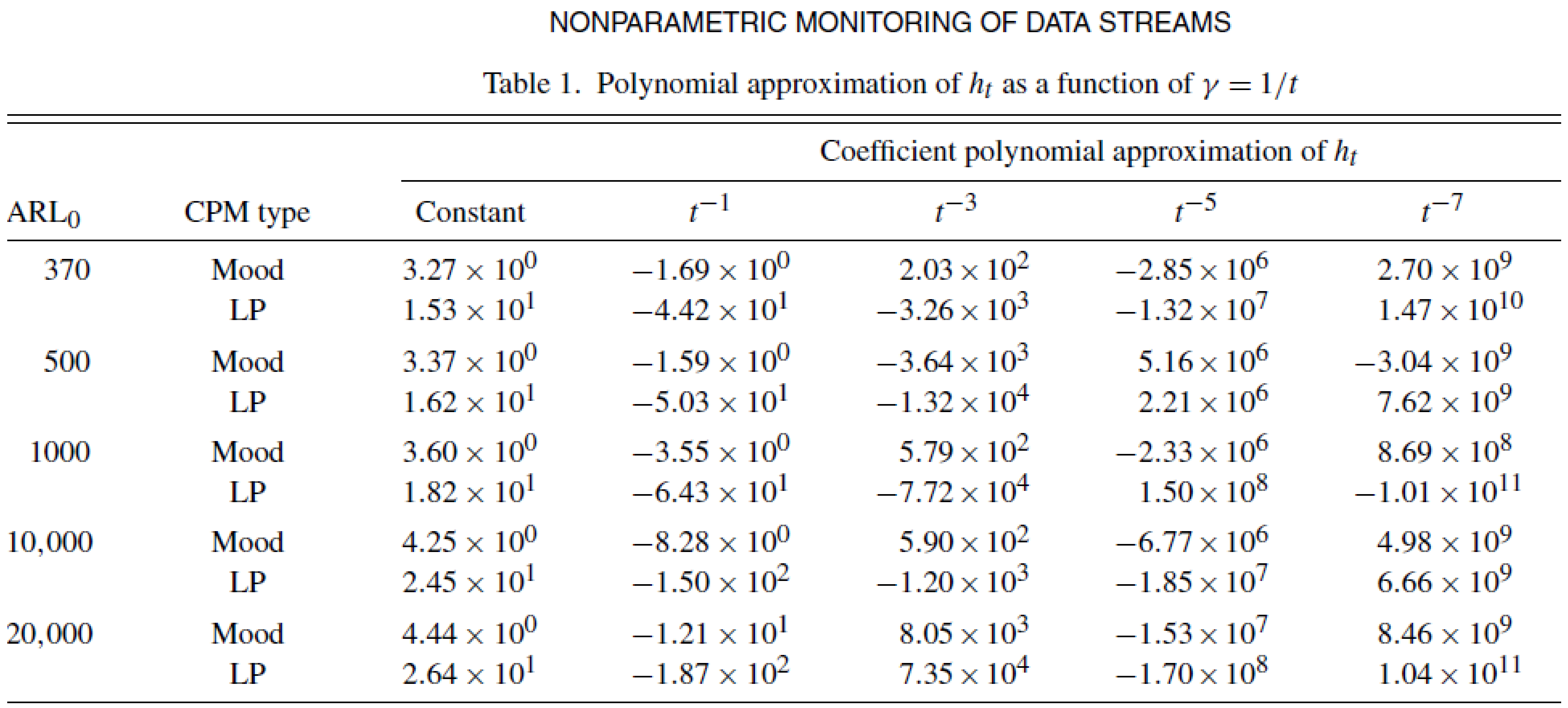
For each desired value of α:

Generate a dataset D of one million streams containing 5000 points drawn from an **arbitrary distribution** $\boldsymbol{\psi}$ (e.g. $N(0,1)$). This is feasible when $\mathcal{S}$ is a distribution-free statistic, as this does not depend on ϕ_0.

- For $t = 1, \dots$
  - Evaluate the statistics over each stream in D and compute $\mathcal{S}_{\max,t}$
  - Compute γ_t over sequences in D such that
    $$P\big(\mathcal{S}_{\max,t} > \gamma_t \mid \mathcal{S}_{\max,t-1} < \gamma_{t-1}, \dots, \mathcal{S}_{\max,1} < \gamma_1\big) < \alpha$$
  - Remove from D streams where $\mathcal{S}_{\max,t} > \gamma_t$
- Interpolate the values of γ_t by some parametric function of t, to “fill in possible gaps” and to smooth all the estimates.

Threshold Computation

Here is an example of polynomial cofficients modeling γ_t



NONPARAMETRIC MONITORING OF DATA STREAMS

Table 1. Polynomial approximation of h_t as a function of $\gamma = 1/t$

		Coefficient polynomial approximation of h_t				
ARL_0	CPM type	Constant	t^{-1}	t^{-3}	t^{-5}	t^{-7}
370	Mood	3.27×10^0	-1.69×10^0	2.03×10^2	-2.85×10^6	2.70×10^9
	LP	1.53×10^1	-4.42×10^1	-3.26×10^3	-1.32×10^7	1.47×10^{10}
500	Mood	3.37×10^0	-1.59×10^0	-3.64×10^3	5.16×10^6	-3.04×10^9
	LP	1.62×10^1	-5.03×10^1	-1.32×10^4	2.21×10^6	7.62×10^9
1000	Mood	3.60×10^0	-3.55×10^0	5.79×10^2	-2.33×10^6	8.69×10^8
	LP	1.82×10^1	-6.43×10^1	-7.72×10^4	1.50×10^8	-1.01×10^{11}
10,000	Mood	4.25×10^0	-8.28×10^0	5.90×10^2	-6.77×10^6	4.98×10^9
	LP	2.45×10^1	-1.50×10^2	-1.20×10^3	-1.85×10^7	6.66×10^9
20,000	Mood	4.44×10^0	-1.21×10^1	8.05×10^3	-1.53×10^7	8.46×10^9
	LP	2.64×10^1	-1.87×10^2	7.35×10^4	-1.70×10^8	1.04×10^{11}

Online Monitoring: Ranks Computation

Ranks computation requires storing all the data in memory

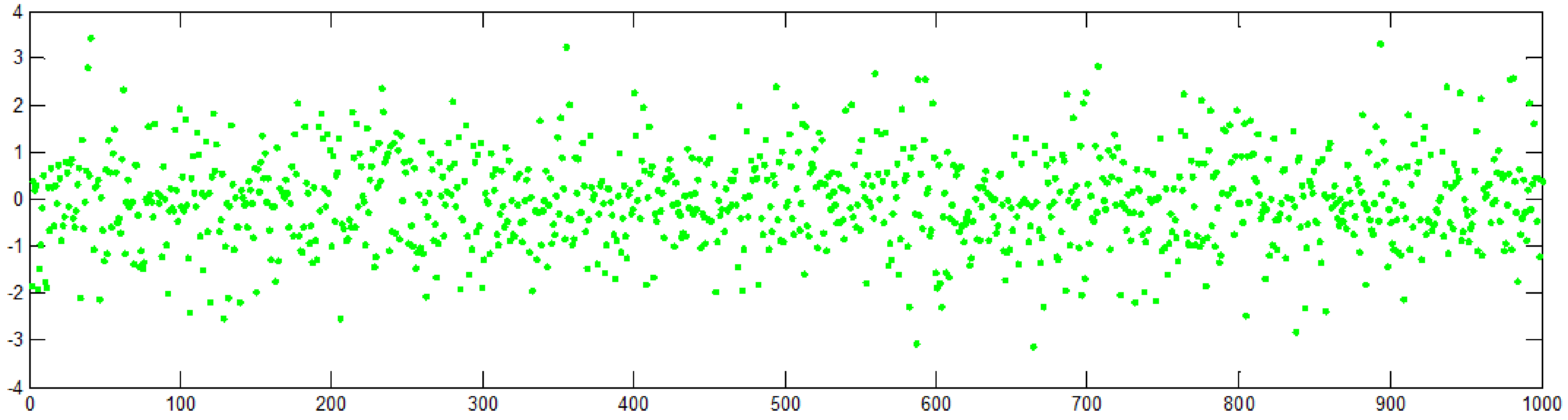
Also time requirement grows at each new observation

This is usually infeasible when working with data streams.

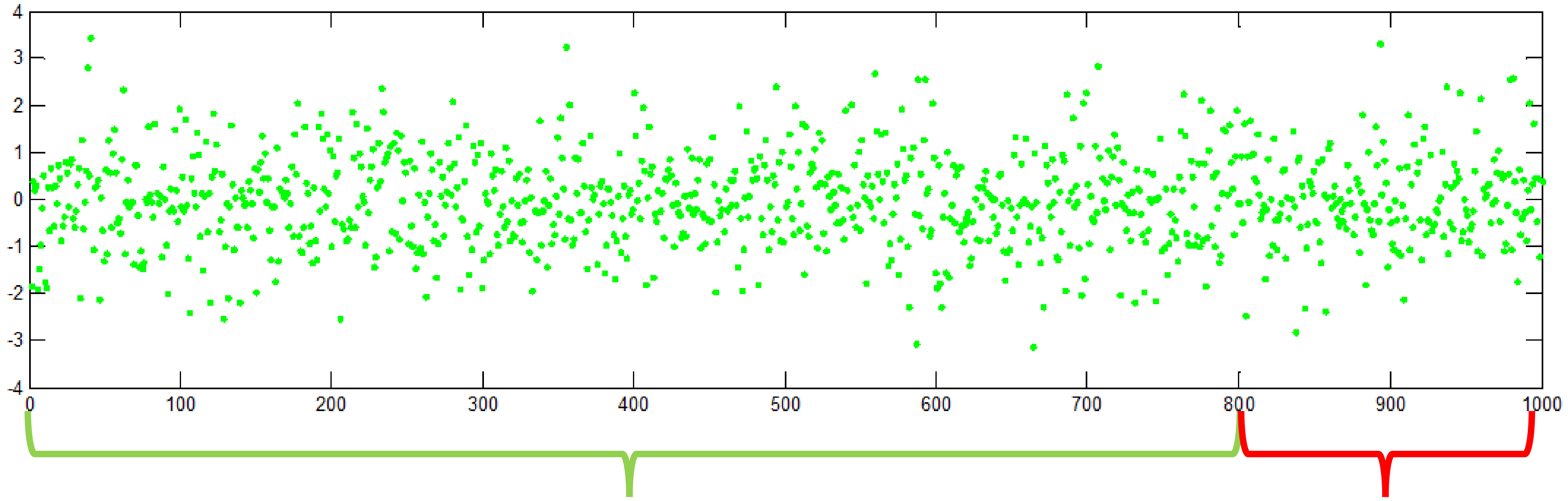
Solution: discretization of the older part of the stream

- Past data are stored in an histogram (ranks computed from quantized values)
- A window over the most recent data is kept to process these accurately
- Introduce an upper bound in memory and time requirements

Data Quantization



Data Quantization

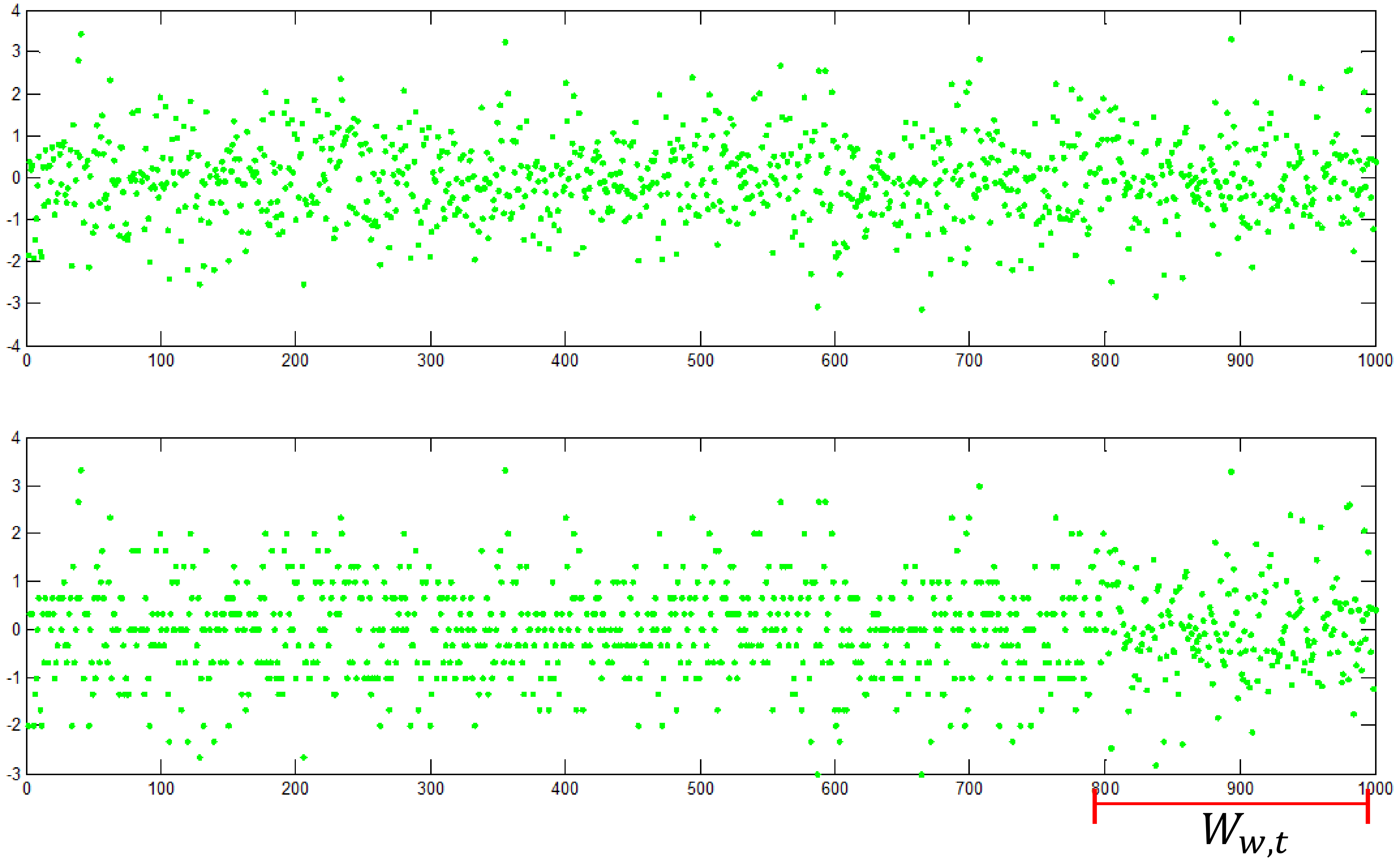


Quantize past data in m values (range is defined from a training sequence)

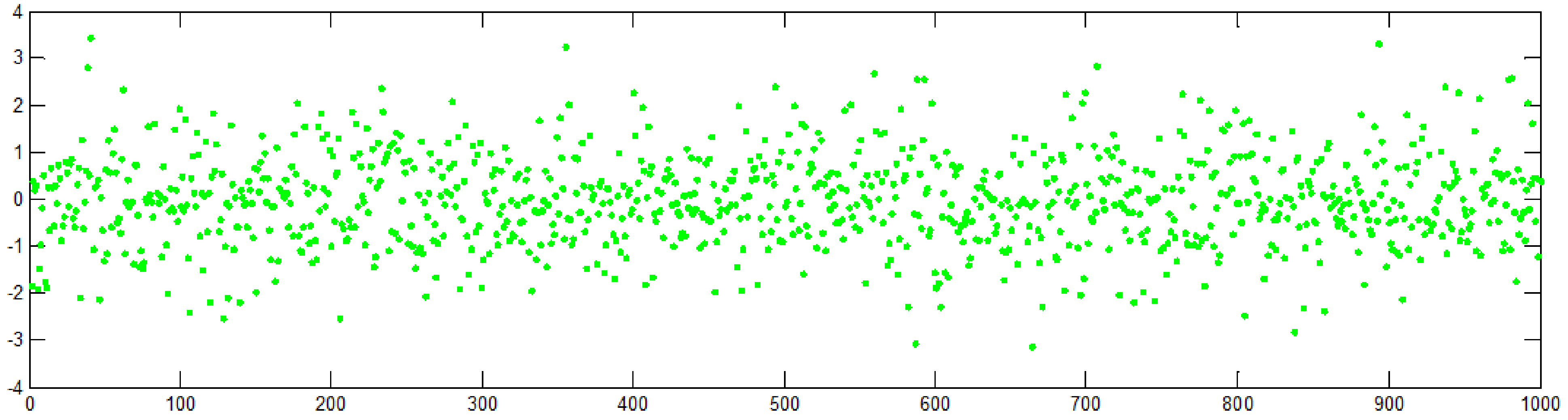
Sliding window $W_{w,t}$ over the stream

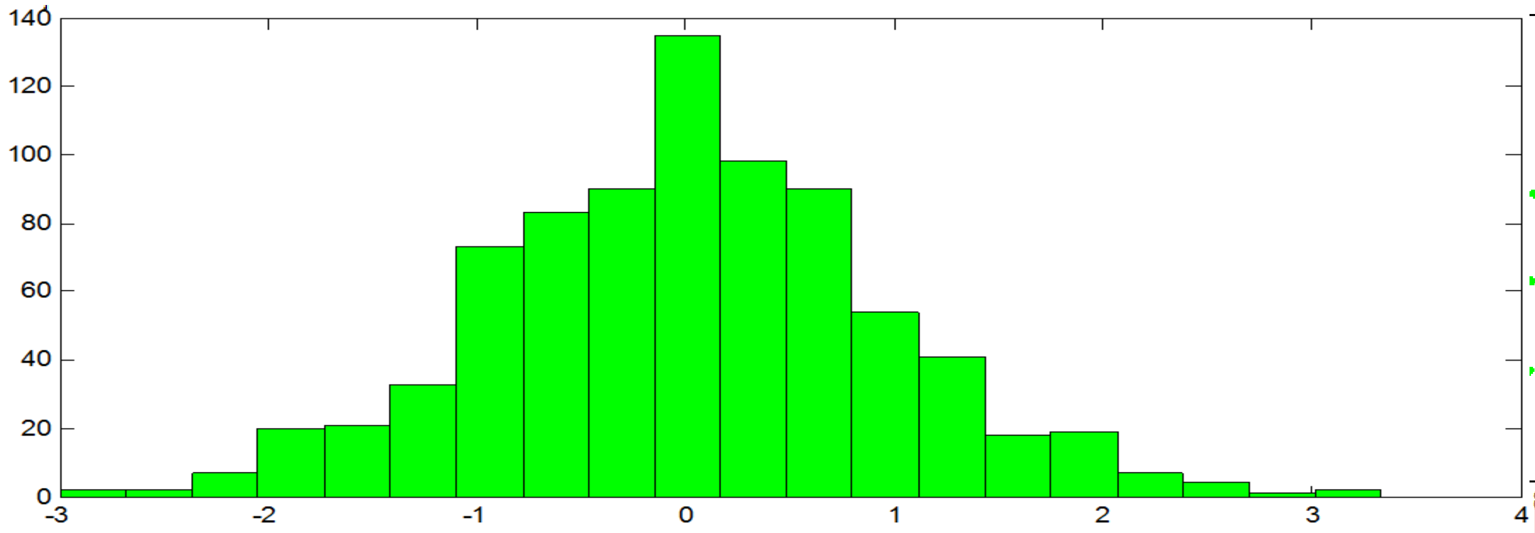
Boracchi

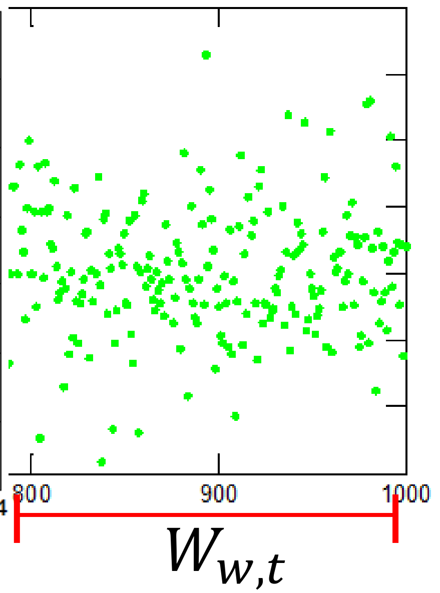
Data Quantization



Data Quantization

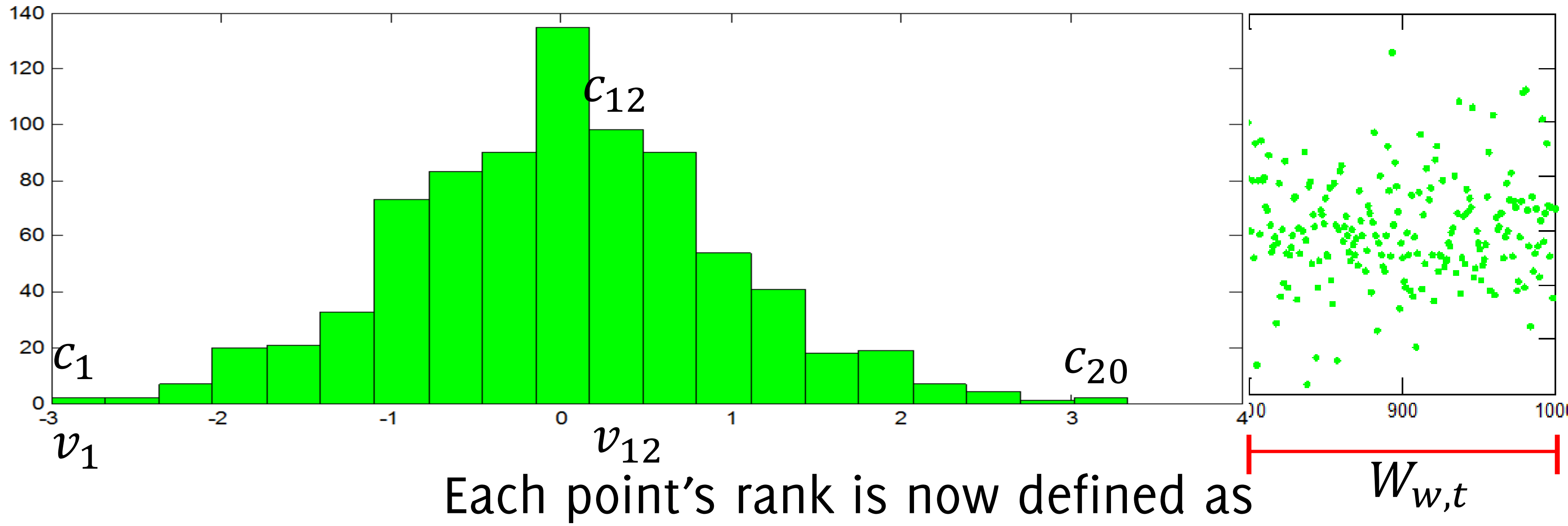






Boracchi

Ranks Computation



Each point's rank is now defined as

$$r(x_t) = r_w(x_t) + \sum_{i=1}^{m} c_i \, I\big(x_t > v_j\big) \; - 1$$

the sum over W plus the rank w.r.t the histogram

Data Quantization

Pros: When windowing is used, the maximum number of operation performed becomes constant (when $t > w$)

Cons: loss of accuracy in rank computation (std adjustement)

Cons: No post-detection diagnosis possible when τ falls before $W_{w,t}$

The change point outcomes is

$$\tau = \underset{t \in W_{w,t}}{\operatorname{argmax}} S_t$$

Change Detection Approaches

- The Change-Point Formulation
    - Parametric
    - Non-parametric
- Change-Detection by Monitoring Features / the Log-likelihood
- Change-Detection by Histograms

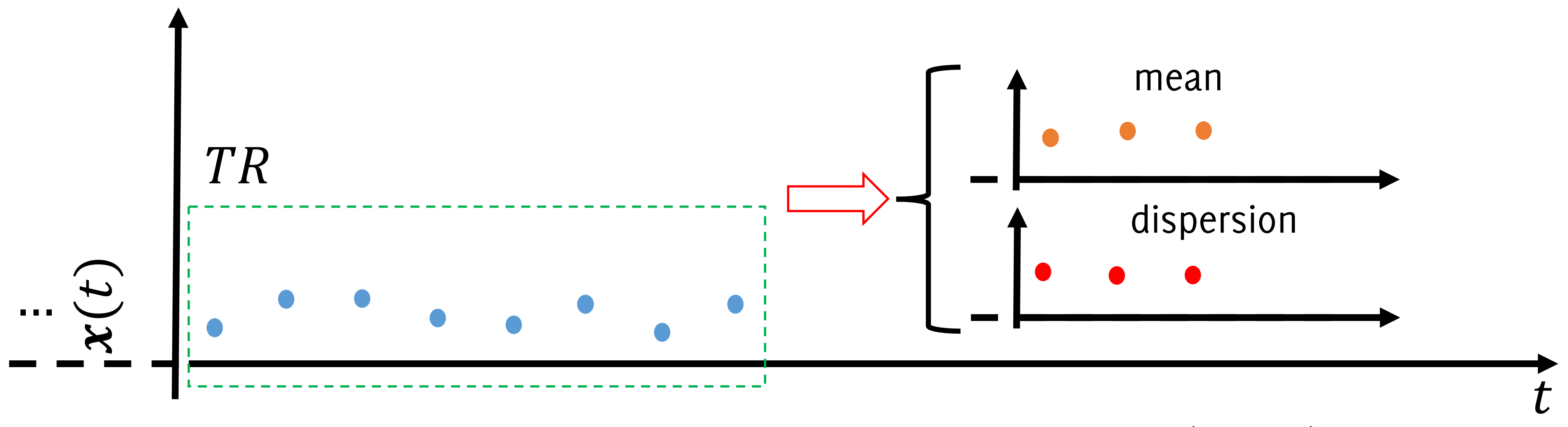
CMPs are nice, but statistics based on sorting holds for scalar sterams

Now we investigate solutions meant for multivariate data streams

Change Detection by Monitoring Features

Most often, **a training set** TR containing **stationary data** is provided, as in semi-supervised anomaly detection methods.

Extract indicators (features), which are **expected to change** when $\phi_0 \rightarrow \phi_1$ and which **distribution is known** under ϕ_0



Nonparametric settings: Sequential Monitoring

Examples of **decision rules** for features

- **CPM,** which can control the ARL_0
- **NP-CUSUM,** to detect changes in the data expectation
- **ICI rule**, to detect changes in the data expectation

Unfortunately **most** nonparametric statistics and the decision rules **do not apply to multivariate data**.

Different features are being monitored separately

Ross, G. J., Tasoulis, D. K., Adams, N. M. *"Nonparametric monitoring of data streams for changes in location and scale"* Technometrics, 53(4), 379-389, 2012.

Alippi, C., Boracchi, G., Roveri, M. *“Change detection tests using the ICI rule”* Proceedings of IJCNN 2010 (pp. 1-7).

Tartakovsky, A. G., Veeravalli, V. V. "*Change-point detection in multichannel and distributed systems*". Applied Sequential Methodologies: Real-World Examples with Data Analysis, 173, 339-370, 2004

Alippi C., Boracchi G. and Roveri M. *"Ensembles of Change-Point Methods to Estimate the Change Point in Residual Sequences"* Soft Computing, Springer, Volume 17, Issue 11 (2013)

Monitoring the Log-Likelihood: A Mainstream Change Detection Approach

Three ingredients

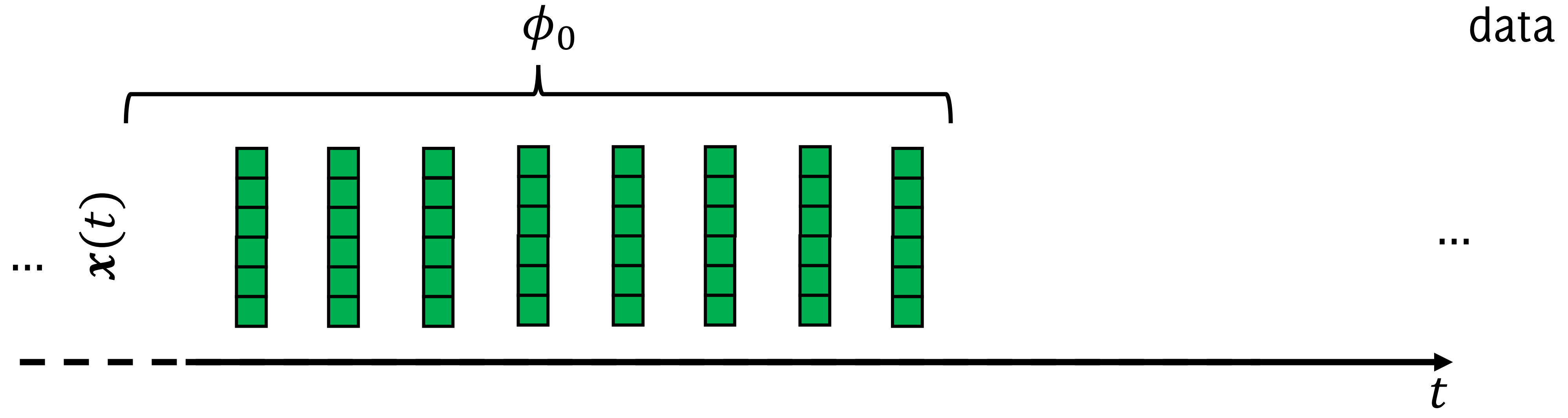
Most change-detection algorithm consists in

i. A model $\hat{\phi}_0$ describing ϕ_0

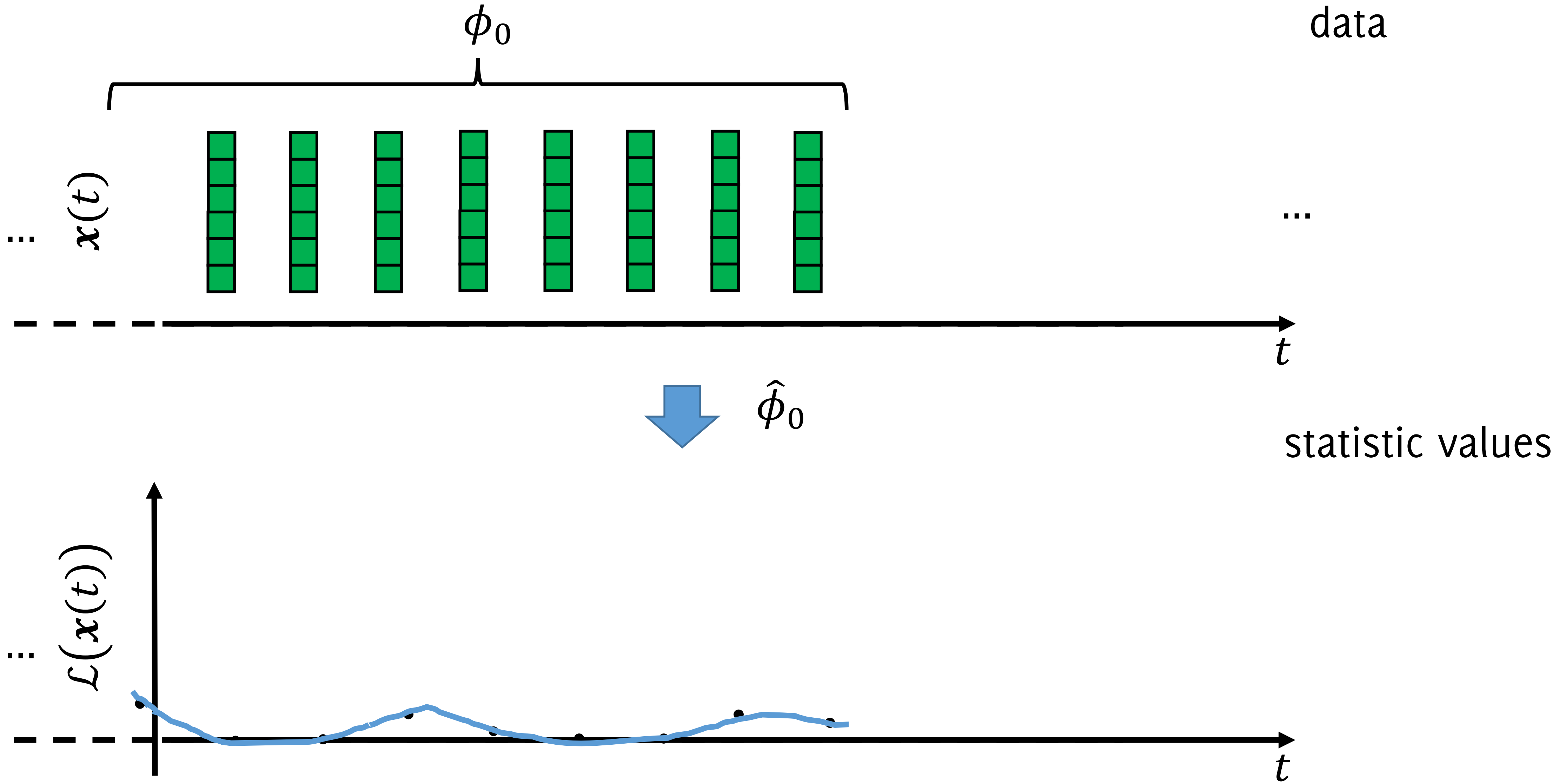
ii. A statistic $\mathcal{T}$ to test incoming data:

iii. A decision rule that monitors $\mathcal{T}$ to detect changes

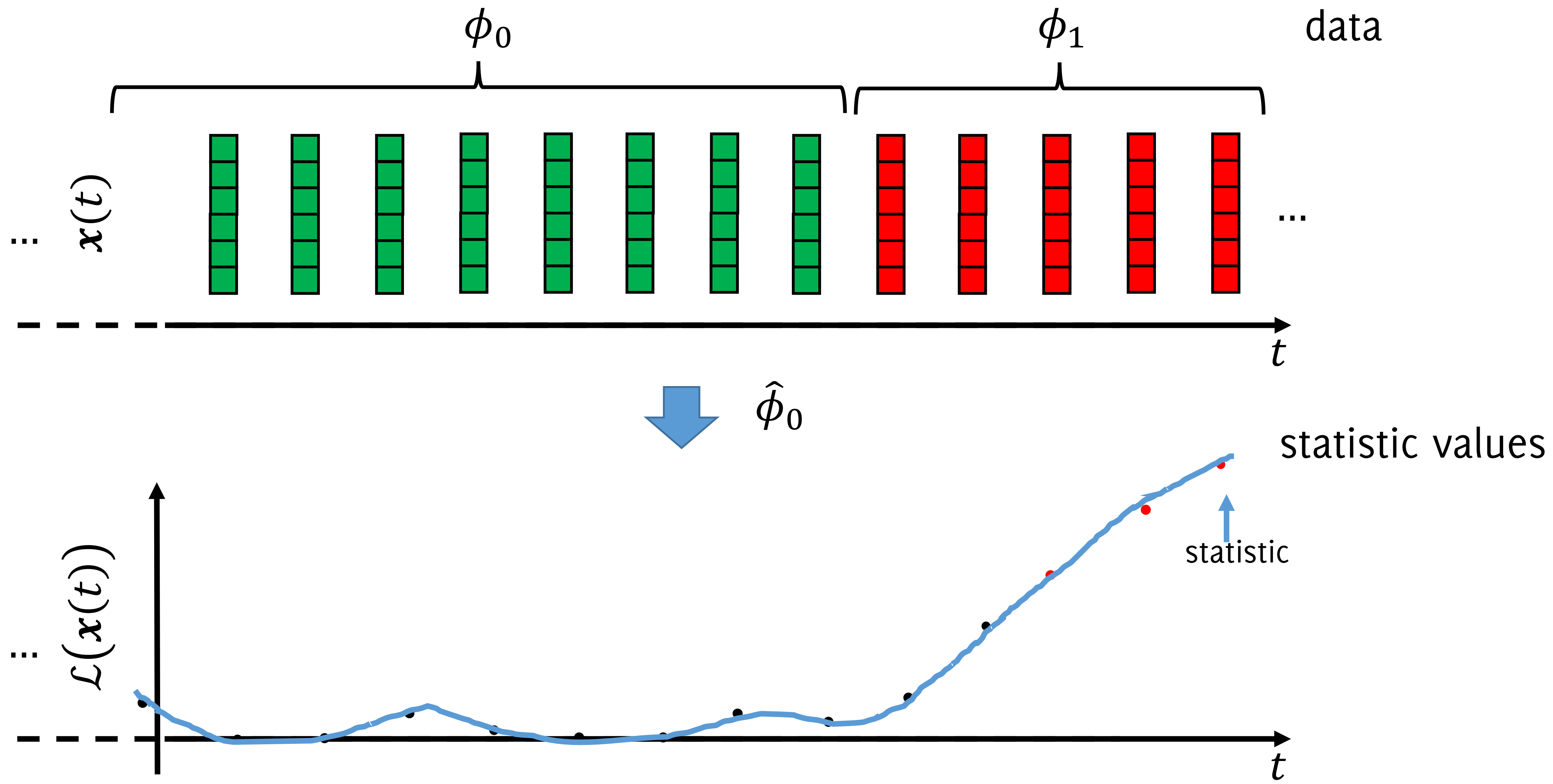
Illustration



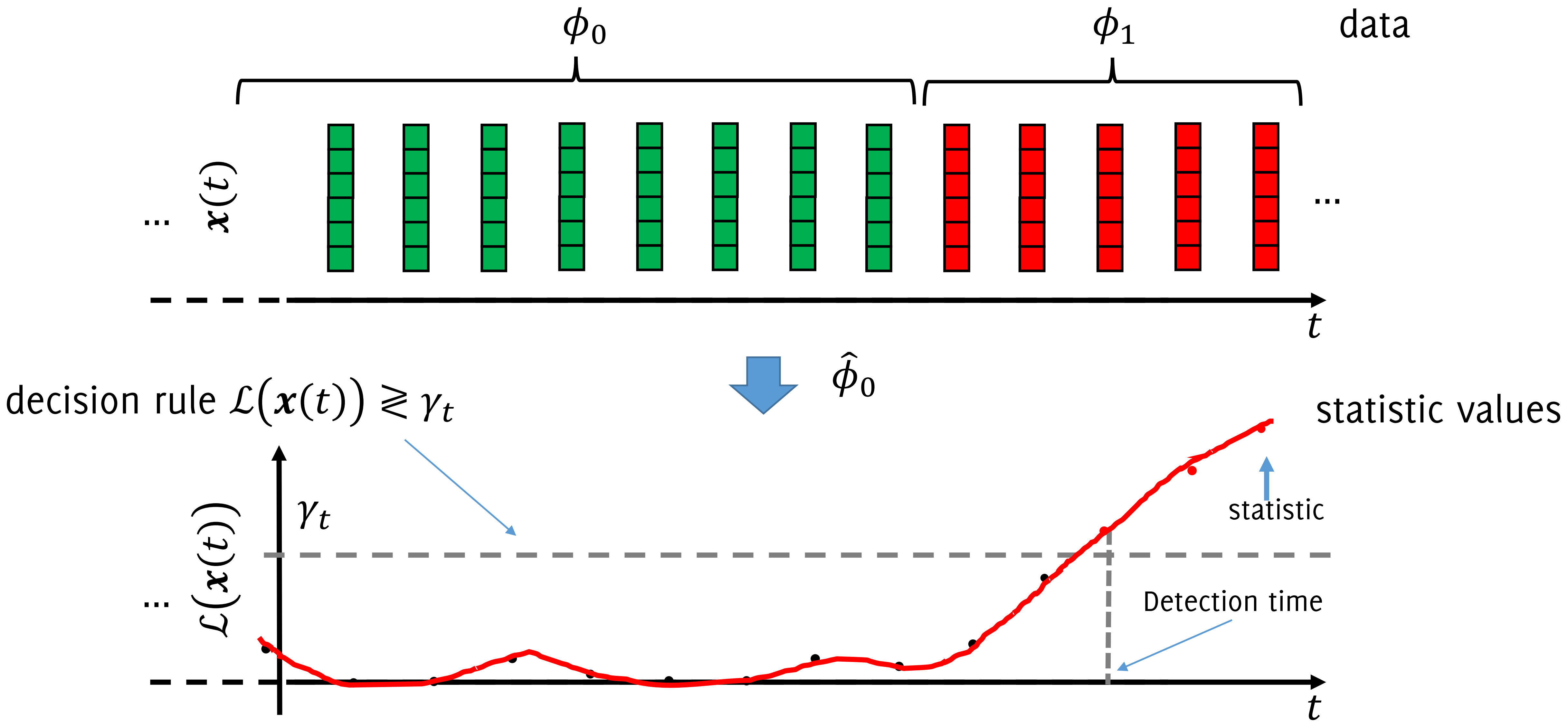
Illustration



Illustration



Illustration



Monitoring the log-likelihood

Fit a general density model $\hat{\phi}_0$ from TR

- Gaussian Mixtures
- Nonparametric Models (KDE)

Statistic to monitor:

$$\mathcal{L}\big(\boldsymbol{x}(t)\big) = -\log(\hat{\phi}_0(\boldsymbol{x}(t)))$$

Heuristic decision rule

$$\mathcal{L}\big(\boldsymbol{x}(t)\big) > \gamma$$

L. I. Kuncheva, *“Change detection in streaming multivariate data using likelihood detectors,"* IEEE Transactions on Knowledge and Data Engineering, 2013.

X. Song, M. Wu, C. Jermaine, and S. Ranka, *“Statistical change detection for multidimensional data,"* in Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD), 2007.

J. H. Sullivan and W. H. Woodall, *“Change-point detection of mean vector or covariance matrix shifts using multivariate individual observations,"* IIE transactions, vol. 32, no. 6, 2000.

Monitoring the log-likelihood

Fit a general density model $\hat{\phi}_0$ from TR

- Gaussian Mixtures
- Nonparametric Models (KDE)

Statistic to monitor:

$$\boxed{\mathcal{L}(\boldsymbol{x}(t)) = -\log(\hat{\phi}_0(\boldsymbol{x}(t)))}$$

Heuristic decision rule

$$\mathcal{L}(\boldsymbol{x}(t)) > \gamma$$

Computing the log prevents numerical errors in case of Gaussian densities. For Gaussian mixtures this can be approximated

L. I. Kuncheva, *"Change detection in streaming multivariate data using likelihood detectors,"* IEEE Transactions on Knowledge and Data Engineering, 2013.

X. Song, M. Wu, C. Jermaine, and S. Ranka, *"Statistical change detection for multidimensional data,"* in Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD), 2007.

J. H. Sullivan and W. H. Woodall, *"Change-point detection of mean vector or covariance matrix shifts using multivariate individual observations,"* IIE transactions, vol. 32, no. 6, 2000.

Sequential Monitoring the log-likelihood

A good baseline for truly sequential and non parametric monitoring:

1. Fit a general density model $\hat{\phi}_0$ from TR

$$\hat{\phi}_0 = \text{fit\_density\_model}(\{\boldsymbol{x}(t), t = 1, \dots, N\})$$

2. For each test sample $\boldsymbol{x}(t)$ compute the log-likeihood

3. Adopt a nonparametric CPM over the stream of likelihood values

$$L = \{-\log(\hat{\phi}_0(\boldsymbol{x}(t)))\,, t = 1, \dots, \}$$

Batch-wise anomaly-detection in the log-likelihood

1. **Fit a general density model** $\hat{\phi}_0$ from TR and **compute the log likelihood** from the last portion of R **training samples** (which have not been used to fit the density model $\hat{\phi}_0$):

$$\hat{\phi}_0 = \text{fit\_density\_model}(\{\boldsymbol{x}(t), t = 1, \dots, N - R\})$$
$$TR_1 = \left\{-\log\left(\hat{\phi}_0\big(\boldsymbol{x}(t)\big)\right), t = N - R + 1, \dots, N\right\}$$

2. Divide the incoming stream in batches and **compute the likelihood over each batch** W_t

$$TS = \left\{-\log\left(\hat{\phi}_0\big(\boldsymbol{x}(t)\big)\right), t \in W_t\right\}$$

3. Detect anomalies as a **left-tailed two-sample t-test** comparing the distributions of likelihood values over TR_1 and TS

CUSUM control chart (parametric case)

Fit a general density model $\hat{\phi}_0$ from TR

- Gaussian Mixtures
- Nonparametric Models (KDE)

Make a guess on $\hat{\phi}_1$

- Statistic to monitor:

$$\mathcal{S}(t) = \left(\log\left(\frac{\hat{\phi}_1(\boldsymbol{x}(t))}{\hat{\phi}_0(\boldsymbol{x}(t))}\right) + \mathcal{T}(t-1) \right)^+$$

- Decision rule

$$\mathcal{S}(t) > \gamma$$

Histograms in Change Detection

Histograms

An histogram h^0 defined over the input domain $\mathcal{X} \subset \mathbb{R}^d$ is

$$h^0(\mathcal{X}) = \{(S_k, p_k^0)\}_{k=1,\dots,K}$$

Where $\{S_k\}_k$ is a disjoint covering of $\mathcal{X}$, namely $S_k \subset \mathcal{X}$

$$\bigcup_k S_k = \mathcal{X} \text{ and } S_j \cap S_i = \delta_{i,j}$$

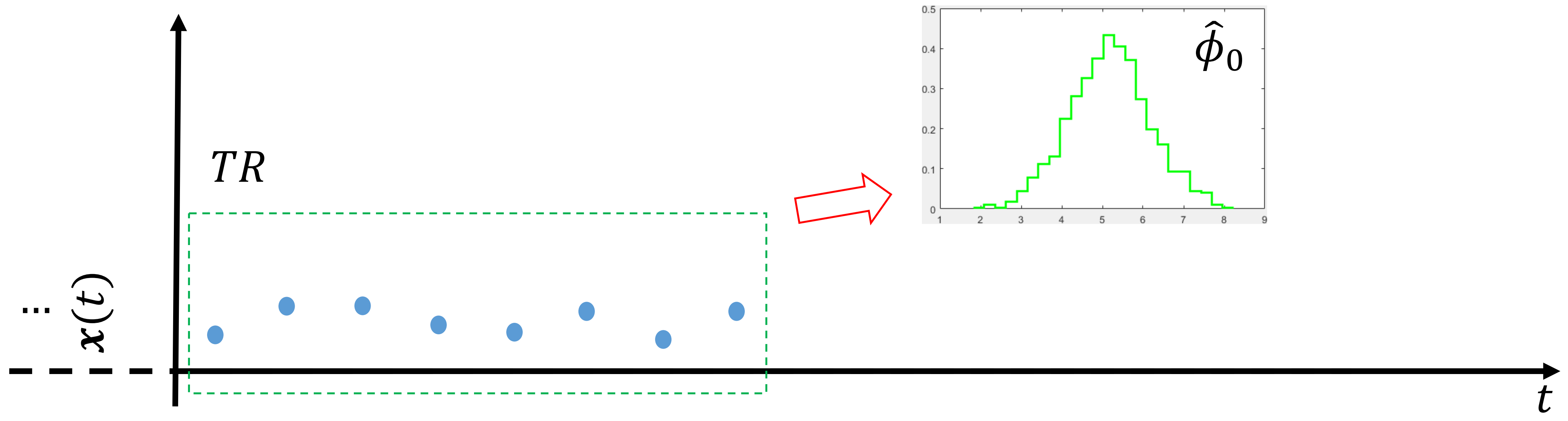
and $p_k^0 \in [0,1]$ is the probability (estimated from TR) for a sample drawn from ϕ_0 to fall inside S_k, i.e.

$$p_k^0 = \frac{m_k}{N}$$

and $N = \#TR$

Change Detection by Means of Histograms

The distribution of stationary data can be approximated by a histogram $\hat{\phi}_0$ estimated from a given training set TR containing stationary data



T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi. *"An information-theoretic approach to detecting changes in multi-dimensional data streams"*. Symposium on the Interface of Statistics, Computing Science, and Applications. 2006

R. Sebastião, J. Gama, P. P. Rodrigues, and J. Bernardes, *"Monitoring incremental histogram distribution for change detection in data streams,"* Lecture Notes on Computer in Knowledge Discovery from Sensor Data, 2017.

Monitoring Approaches

Two **major monitoring approaches** using histograms:

- **Likelihood-based** methods
- **Distance-based** methods

whose applicability also depends on the partitioning scheme

Log-likelihood – Based Monitoring Scheme

As in density-based methods, $\hat{\phi}_0$ can be used to compute the log-likelihood, which can be then monitored by univariate CDT

1. During training, estimate $\hat{\phi}_0 = \{(S_k, p_k^0)\}_{k=1,\dots,K}$ from TR
2. During testing, compute
$$\mathcal{L}(\boldsymbol{x}(t)) = \hat{\phi}_0(\boldsymbol{x}(t))$$
3. Monitor $\{\mathcal{L}(\boldsymbol{x}(t)),\ t = 1, \dots\}$ which is now discrete

Log-likelihood – Based Monitoring Scheme

As in density-based methods, $\hat{\phi}_0$ can be used to compute the log-likelihood, which can be then monitored by univariate CDT

1. During training, estimate $\hat{\phi}_0 = \{(S_k, p_k^0)\}_{k=1,\dots,K}$ from TR

2. During testing, compute
$$\boxed{\mathcal{L}(\boldsymbol{x}(t)) = \hat{\phi}_0(\boldsymbol{x}(t)) = p_k^0 \ \text{s.t.} \ \boldsymbol{x}(t) \in S_k}$$

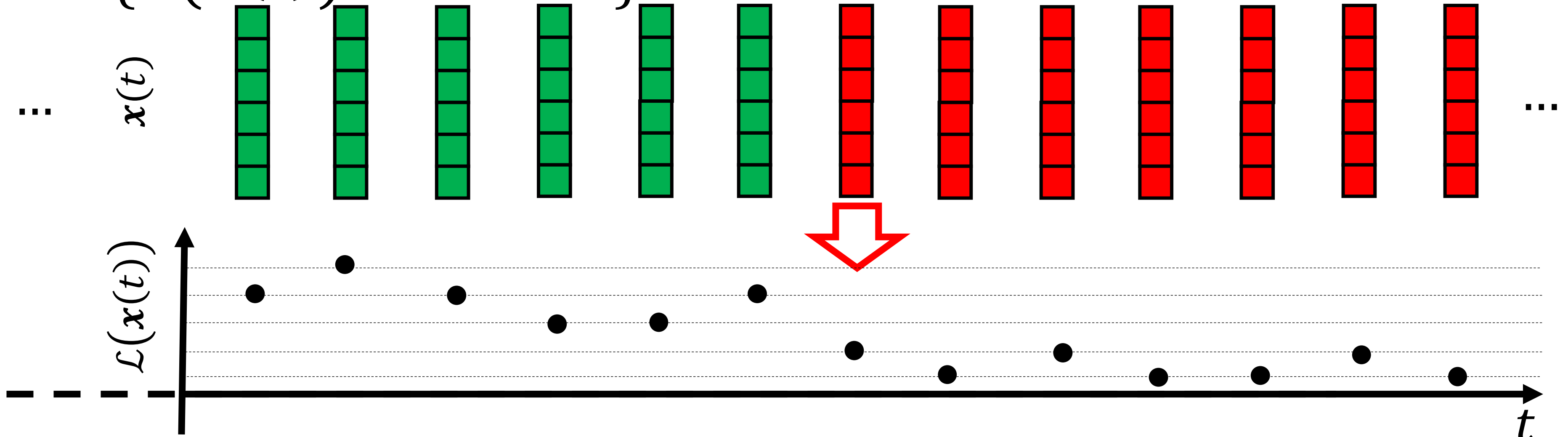
This is the problem of associating each incoming sample to the corresponding bin

3. Monitor $\{\mathcal{L}(\boldsymbol{x}(t)),\ t = 1, \dots\}$ which is now discrete

Log-likelihood – Based Monitoring Scheme

As in density-based methods, $\hat{\phi}_0$ can be used to compute the log-likelihood, which can be then monitored by univariate CDT

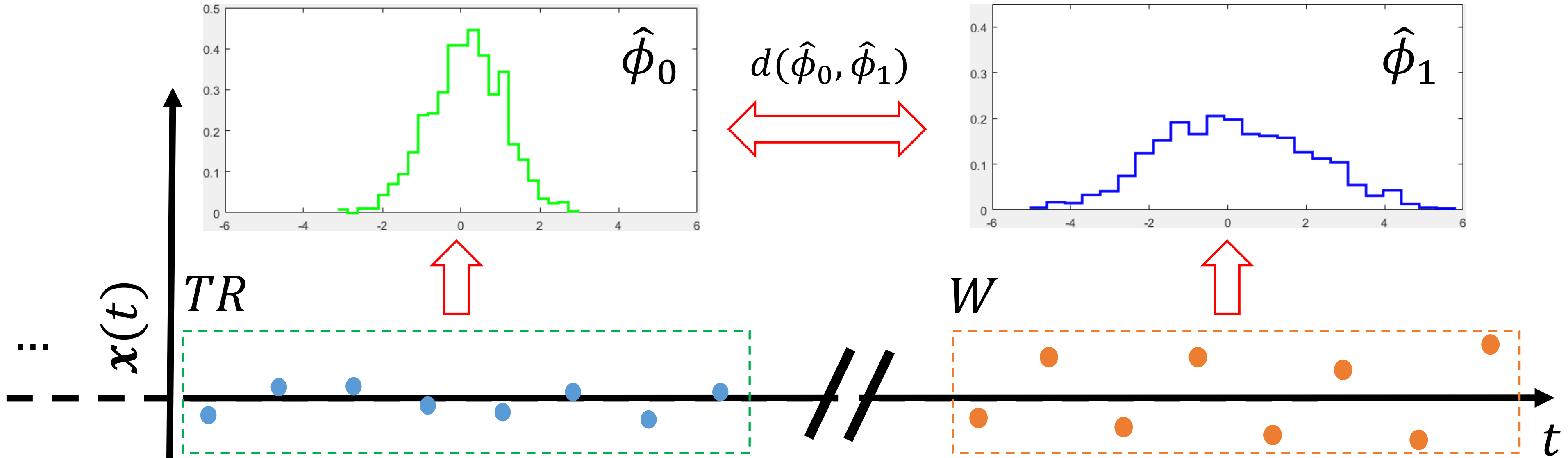
1. During training, estimate $\hat{\phi}_0 = \{(S_k, p_k^0)\}_{k=1,\dots,K}$ from TR
2. During testing, compute
$$\mathcal{L}(\boldsymbol{x}(t)) = \hat{\phi}_0(\boldsymbol{x}(t)) = p_k^0 \text{ s.t. } \boldsymbol{x}(t) \in S_k$$
3. Monitor $\{\mathcal{L}(\boldsymbol{x}(t)),\ t = 1, \dots\}$ which is now discrete



Distance-Based (or batch) Monitoring Scheme

$\hat{\phi}_0$ can be used to monitor the datastream window-wise:

- During training, estimate $\hat{\phi}_0 = \{(S_k, p_k^0)\}_{k=1,\dots,K}$ from TR
- Crop a window W over the most recent data
- Estimate $\hat{\phi}_1 = \{(S_k, p_k^1)\}_{k=1,\dots,K}$ from W
- Compare $\hat{\phi}_0$ and $\hat{\phi}_1$ by a distance d between distributions
- Monitor $d(\hat{\phi}_0, \hat{\phi}_1)$

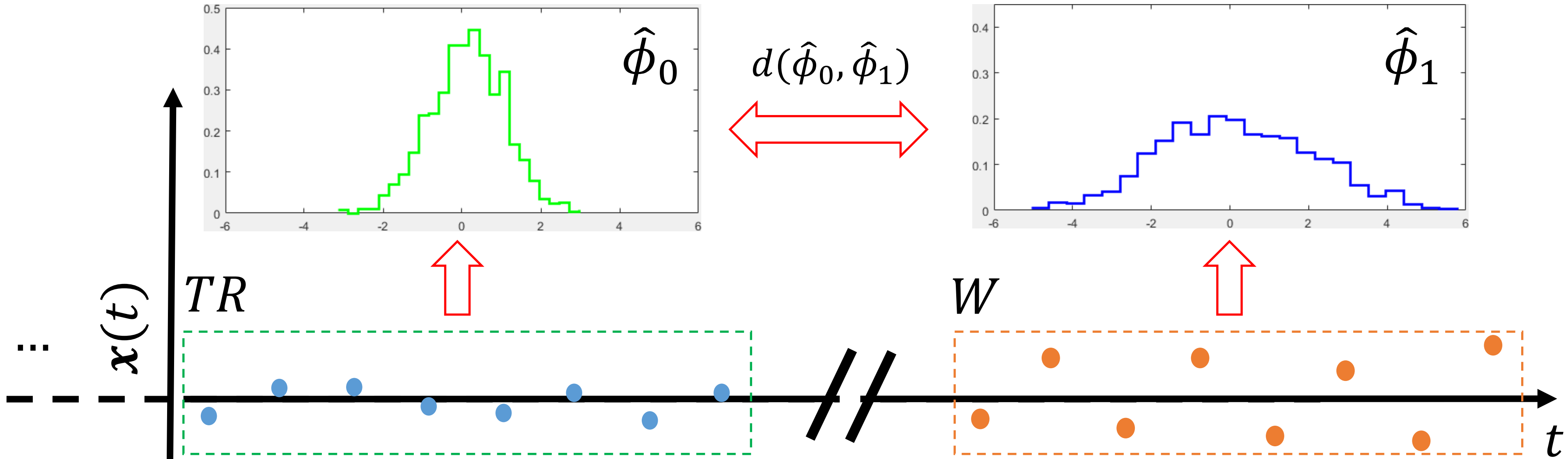


Distance-Based (or batch) Monitoring Scheme

$\hat{\phi}_0$ can be used to monitor the datastream window-wise:

- During training, estimate $\hat{\phi}_0 = \{(S_k, p_k^0)\}_{k=1,\dots,K}$ from TR
- Crop a window W over the most recent data
- Estimate $\boxed{\hat{\phi}_1 = \{(S_k, p_k^1)\}_{k=1,\dots,K} \text{ from } W}$
- Compare $\hat{\phi}_0$ and $\hat{\phi}_1$ by a distance d between distributions
- Monitor $d(\hat{\phi}_0, \hat{\phi}_1)$

Here bins are defined by $\hat{\phi}_0$, we just have to associate each sample to the corresponding bin



Distance-Based Monitoring scheme: Stopping Rule

Thresholding the distance is the typical stopping rule.

$$d(\hat{\phi}_0, \hat{\phi}_1) \gtrless \gamma$$

- γ defined from the empirical distribution of $d(\hat{\phi}_0, \hat{\phi}_1)$, which is computed through a **Bootstrap procedure**.
- γ given from **approximation of the statistic**, which typically **holds asymptotically**, as in case the of the Pearson statistics

Similar approaches can be used to compare features extracted in different data-windows.

Dasu, T., Krishnan, S., Venkatasubramanian, S., Yi, K. "*An information-theoretic approach to detecting changes in multi-dimensional data streams*". Symp. on the Interface of Statistics, Computing Science, and Applications, 2006.

Ditzler G., Polikar R., *"Hellinger distance based drift detection for nonstationary environments"*, IEEE SSCI 2011.

Boracchi G., Cervellera C., and Maccio D. "*Uniform Histograms for Change Detection in Multivariate Data*" IJCNN 2017

Sebastião R., Gama J. Mendonça T. "*Fading histograms in detecting distribution and concept changes*" IJDSA, 2017

Bu L., Alippi C., Zhao D. *"A pdf-free change detection test based on density difference estimation"* TNNLS 2016

S. Liu, M. Yamada, N. Collier, and M. Sugiyama, *"Change-point detection in time-series data by relative density-ratio estimation,"* Neural Networks, 2013

An example of distance-based monitoring scheme

1. Compute the probabilities for an incoming batch W over $\{S_k\}$

$$p_k^W = \frac{\#\{\boldsymbol{x}_i \in S_k \cap W\}}{\nu}$$

2. Compare h^0 and h^W by a suitable distance, e.g.

$$d_{TV}(h^0, h^W) = \frac{1}{2}\sum_k |p_k^0 - p_k^W| \quad \text{(total variation)}$$

or

$$d_{PS}(h^0, h^W) = \nu \sum_k \frac{\left(p_k^0 - p_k^W\right)^2}{p_k^0} \quad \text{(Pearson)}$$

3. Run an HT on d_{TV} (having estimated its p-values empirically) or d_P (this follows a χ-square distribution)

Pros and Cons of using histograms

Pros:

- Histograms are very **general and flexible models**.
- Some partitioning schemes can be associated with **a tree having splits along a single component (kd-trees, quantTrees)**. This enable very fast searches through the histogram.

Cons:

- When d increases, grids are not a viable option, since they require q^d bins.
- In general, the distribution of test statistic is unknown

Pros and Cons of using histograms

Pros:

- Histograms are very **general and flexible models**.
- Some partitioning schemes can be associated with **a tree having splits along a single component (kd-trees, quantTrees)**. This enable very fast searches through the histogram.

Cons:

- When d increases, grids are not a viable option, since they require q^d bins.
- In general, the distribution of test statistic is unknown

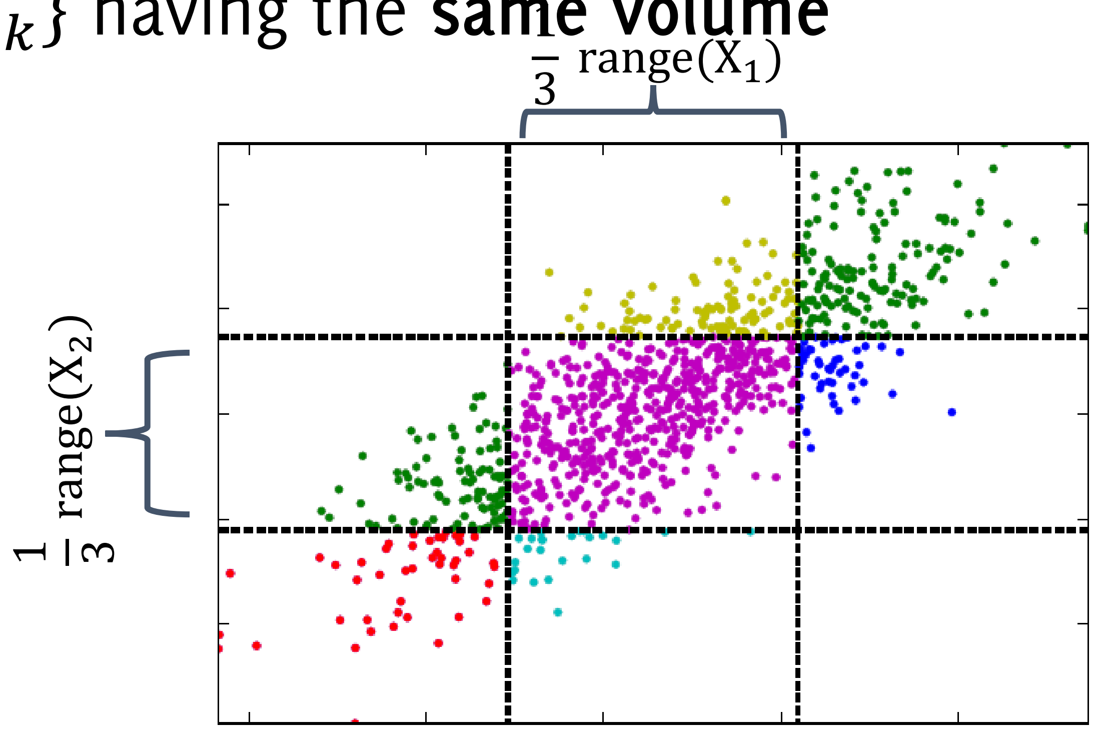
However, there is quite a lot of freedom in designing $\{S_k\}_k$

Histograms yielding uniform volume

"grids": the most common way of constructing histograms.

Build a tessellation of $\text{supp}(TR)$ by splitting each component in q equally sized parts.

This yields q^d hyper-rectangles $\{S_k\}$ having the **same volume**



An example of 2D histogram $q = 1/3$

Histograms yielding uniform volume

"grids": the most common way of constructing histograms.

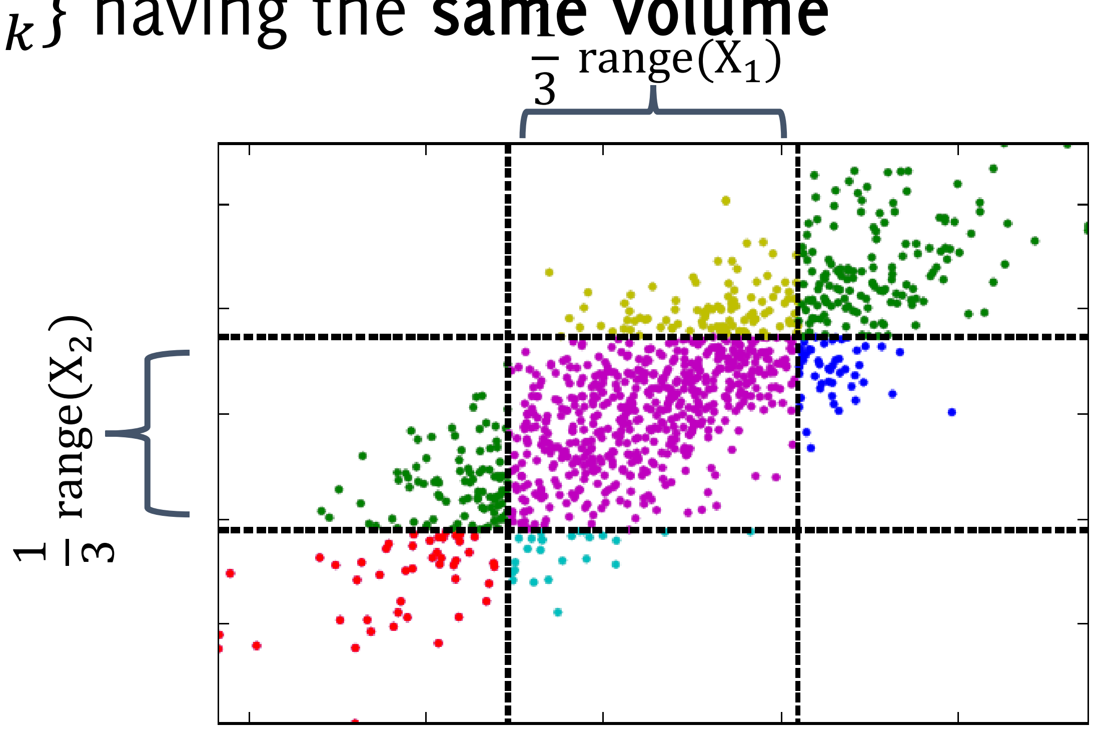
Build a tessellation of $\text{supp}(TR)$ by splitting each component in q equally sized parts.

This yields q^d hyper-rectangles $\{S_k\}$ having the **same volume**

Add to the histogram a region to gather points that during operation, won't fall in $\text{supp}(TR)$

$$S_K = \overline{TR}, p_K^0 = 0$$

being $K = q^d + 1$



An example of 2D histogram $q = 1/3$

Boracchi

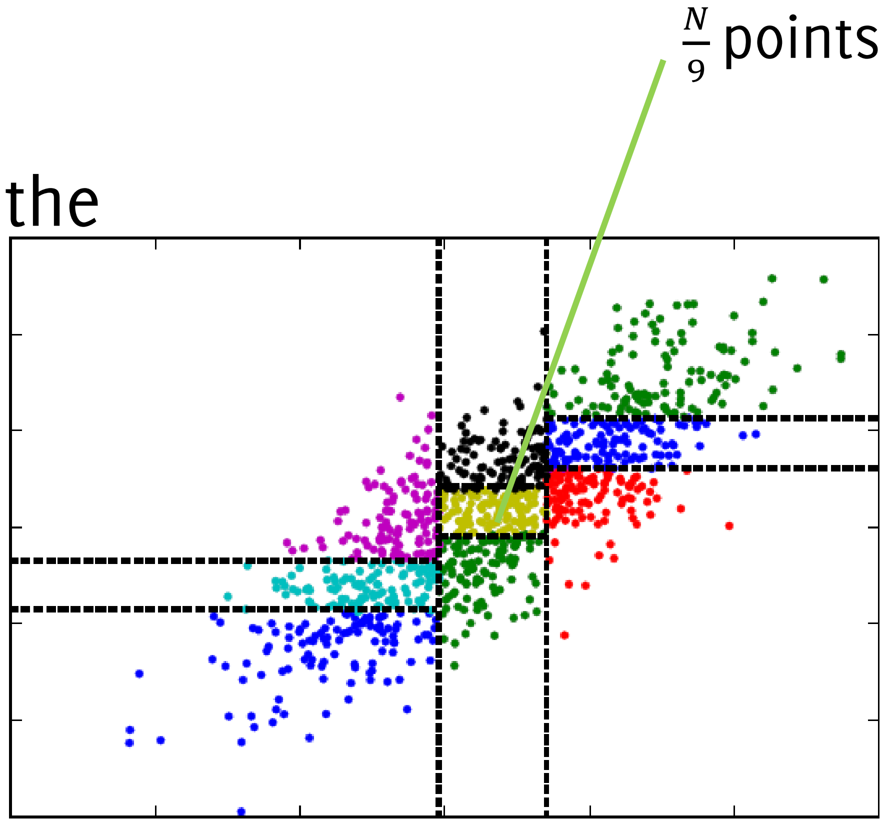
Histograms yielding uniform density

Define the partition $\{S_k\}_k$ in such a way that all the subsets have the **uniform density**, i.e.,

$$p_k^0 \approx \frac{1}{K}, k = 1, .., K$$

Such that each of the q^d hyper-rectangles contains the same number of points

No need to consider a bin for $\bar{X}$



An example of 2D histogram $q = 1/3$

Histograms yielding uniform density

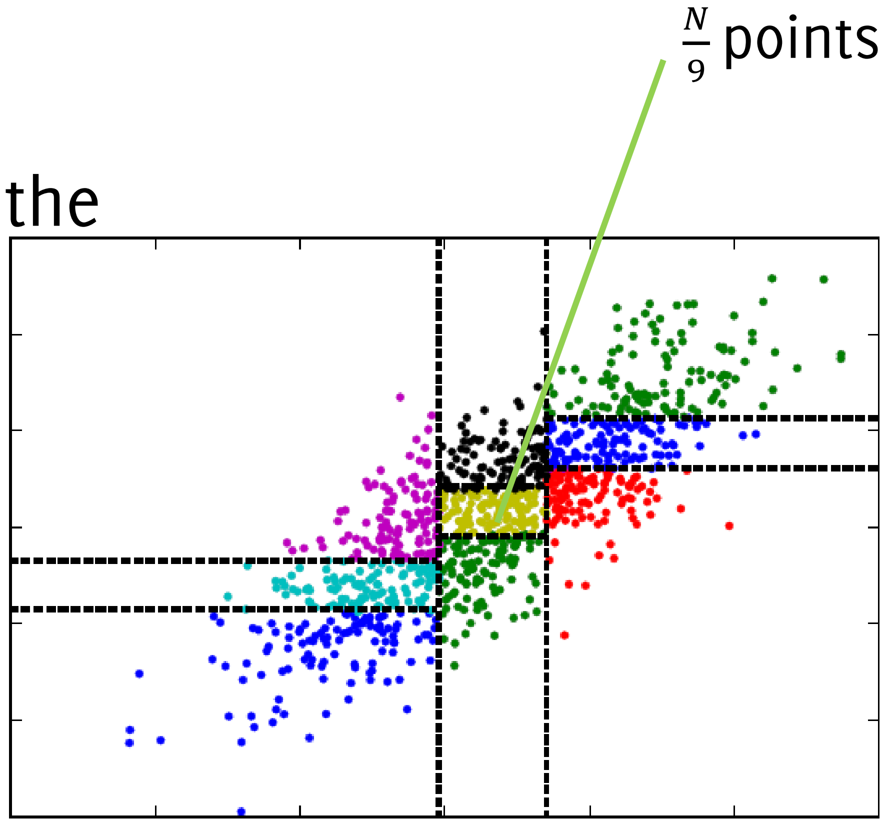
Define the partition $\{S_k\}_k$ in such a way that all the subsets have the **uniform density**, i.e.,

$$p_k^0 \approx \frac{1}{K}, k = 1, .., K$$

Such that each of the q^d hyper-rectangles contains the same number of points

No need to consider a bin for $\bar{X}$

This is an example of k-d tree, there are many alternatives…



An example of 2D histogram $q = 1/3$

Adaptation in NSE

Adaptation

Adaptation Strategies Under Concept Drift

Two main solutions in the literature:

- **Active**: the classifier K_t is combined with statistical tools **to detect concept drift and pilot the adaptation**
- **Passive**: the classifier K_t undergoes **continuous adaptation** determining every time which supervised information to preserve

Which is best depends on the expected change rate and memory/computational availability

Active Approaches

Peculiarities:

- Rely on an **explicit drift-detection mechanism**: such as an outlier detection or a change detection test (CDT)
- Specific **post-detection adaptation** procedures to isolate data generated after the change, which are coherent with the new concept

Pro:

- Also provide information that CD has occurred
- Can improve their performance in stationary conditions
- Alternatively, classifier adapts only after detection

Cons:

- Difficult to handle incremental and gradual drifts

Passive Approaches

Passive approaches:

- **Do not have an explicit** CD **detection** mechanism
- They are **aware** that $\phi_t(\boldsymbol{x}, y)$ *might* change at any time and at any rate
- **Perform continuous adaptation** of their model(s) parameters at each new arrival

They can be divided in:

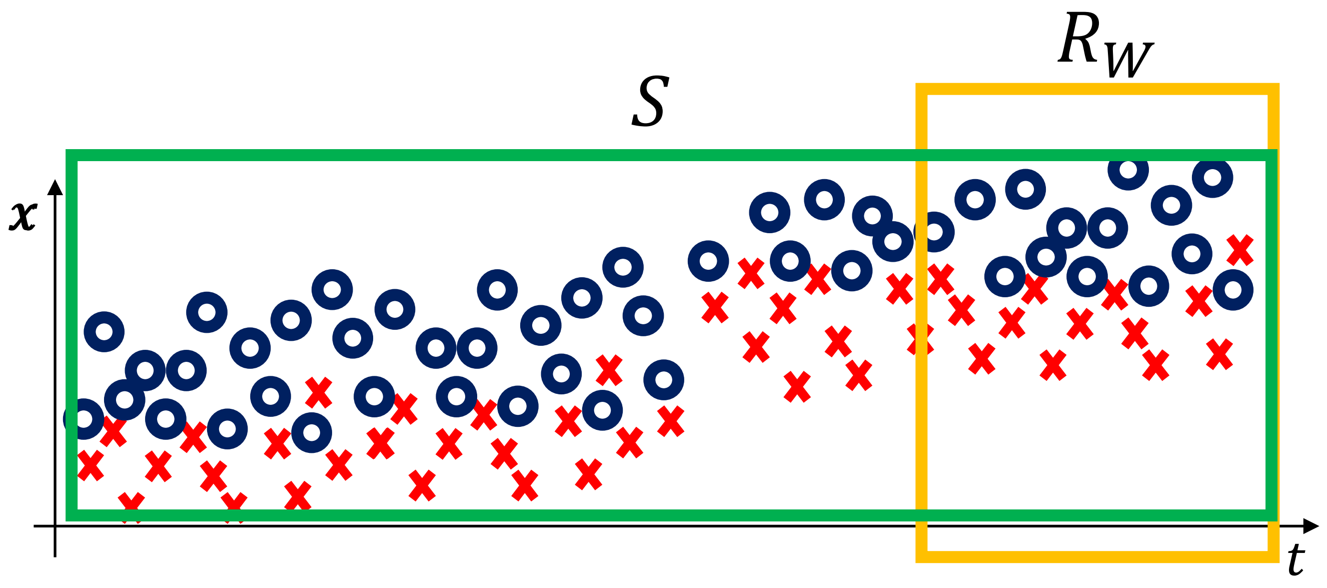
- **Single model** methods
- **Ensemble** methods

Adaptation in Active Approaches

Methods Based on Windows Comparison

Paired Learners

To cope with concept drift, we paired a stable online learner with a reactive one. A ***stable learner*** S *predicts based on all of its experience, whereas a* ***reactive learner*** R_W *predicts based on its experience over a short, recent window of time.*



Paired Learners

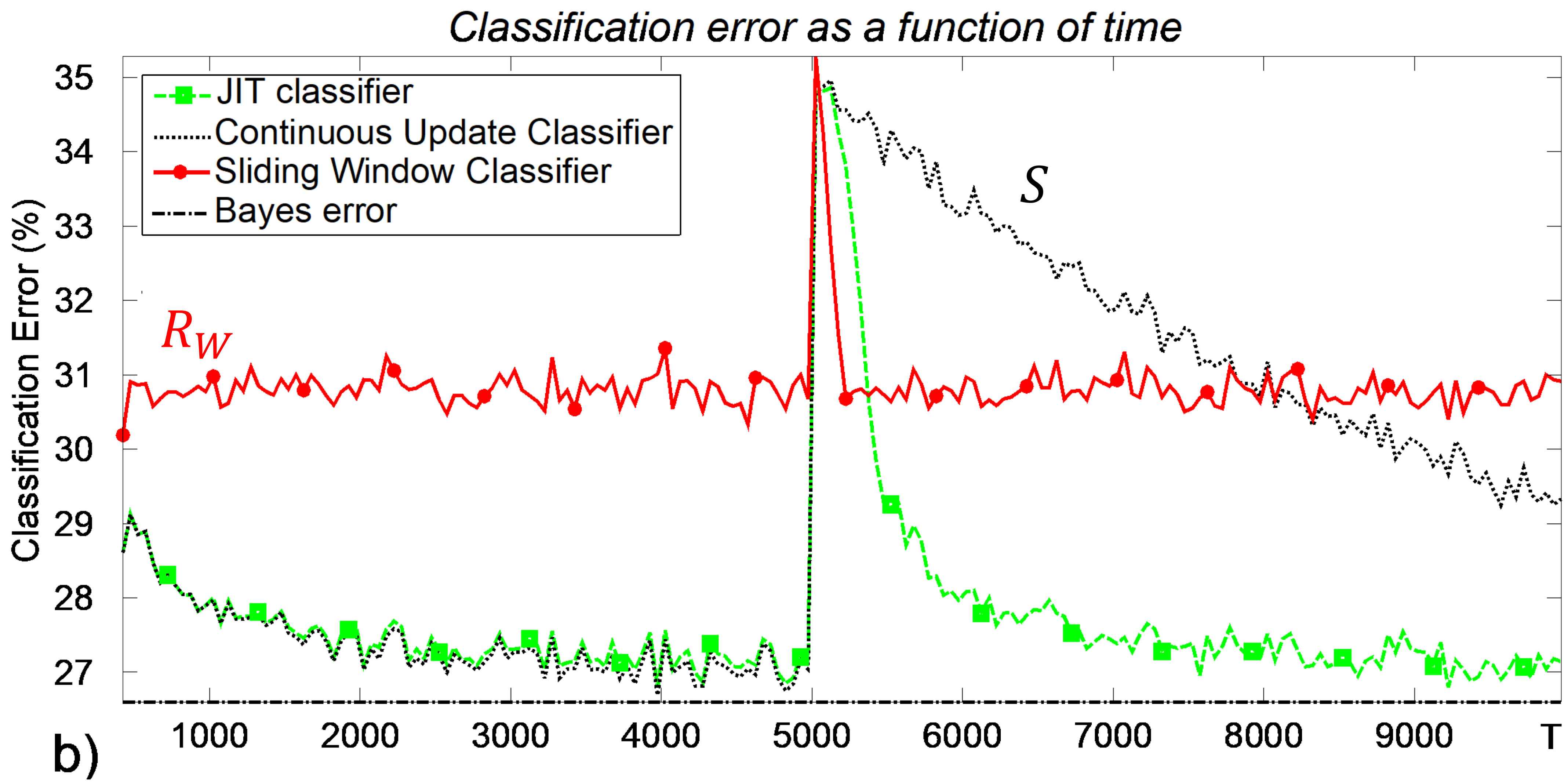
To cope with concept drift, we paired a stable online learner with a reactive one. A ***stable learner*** S *predicts based on all of its experience, whereas a* ***reactive learner*** R_W *predicts based on its experience over a short, recent window of time.*

Paired Learning copes with concept drift by:

- *Leveraging the interplay between reactive and stable learners*
- *Analyze the differences in their accuracy to cope with concept drift*

Bach, S.H.; Maloof, M., *"Paired Learners for Concept Drift"* ICDM '08.

Paired Learners



Paired Learners

Limitations of fixed window methods:

- A too reactive R_W may have difficulty acquiring *any* target concept
- A too stable learner S may be overly burdened by knowledge of a previous concept to learn a new one.

Strengths:

- $\boldsymbol{S}$ **outperforms** $\boldsymbol{R_W}$ when acquiring a stationary concept,
- $\boldsymbol{R_W}$ **outperforms** $\boldsymbol{S}$ **when** the concept changes.

Idea:

- Detect a change when R_W outperforms S over a short window of time
- Adapt to the new concept by replacing S with R_W
- Predictions are always provided by the stable classifier S

Paired Learners

Two classifiers are trained and steadily updated

- A **stable online learner** (S) that predicts based on all the supervised samples
- A **reactive** one (R_w) trained over a short sliding window

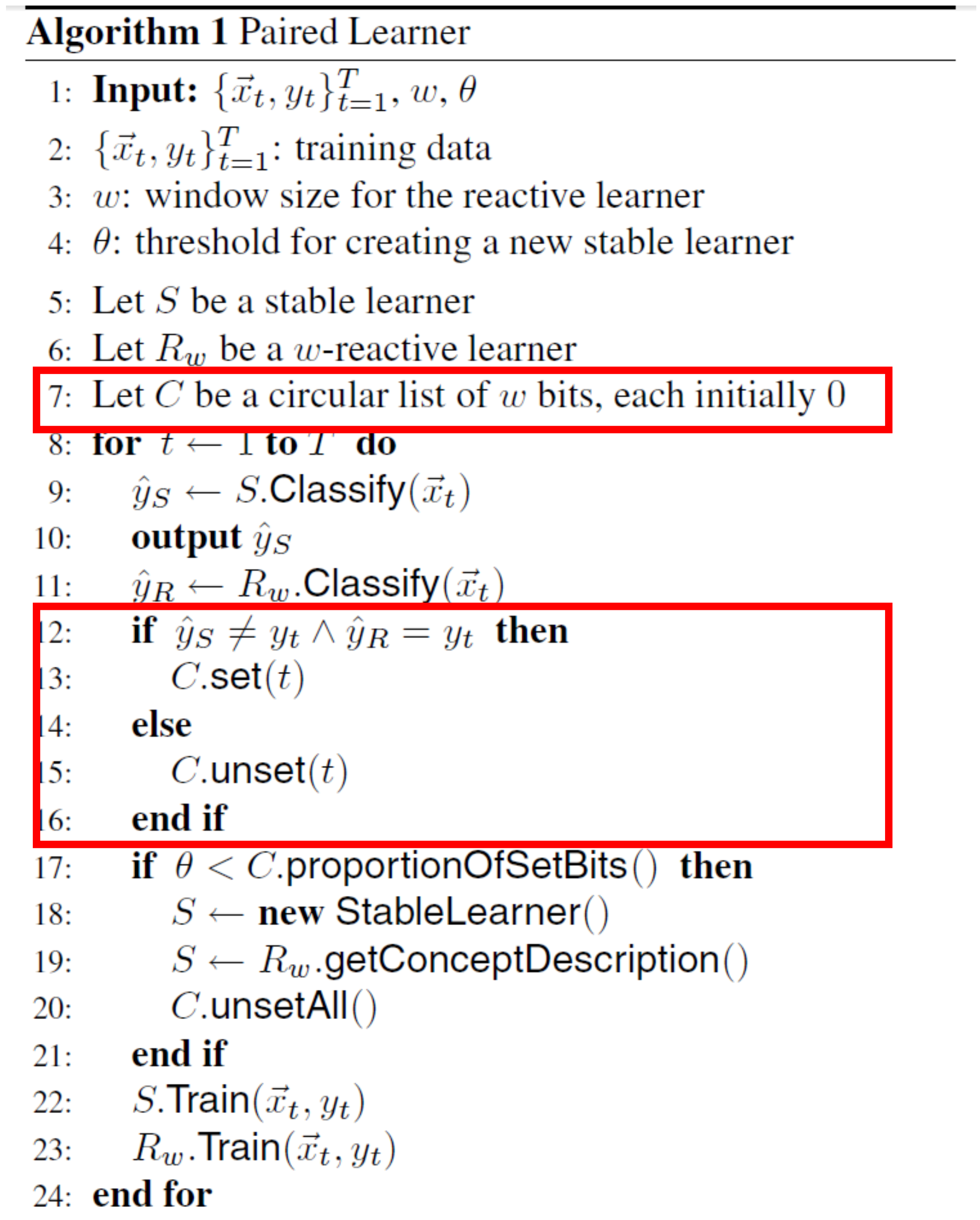
During operation

- Only S provides the outputs of the model
- Predictions of R_w are computed but not provided
- As soon as, on the most recent samples, $\boldsymbol{R_w}$ **outperforms** S over a test window of length w, then **detect CD**

Adaptation consists in **replacing** S by R_w

Paired Learners

This is to keep track of classification errors ϵ_t over the window w



Algorithm 1 Paired Learner

```
Input: ${\{\vec{x}_t, y_t\}}_{t=1}^{T}$, $w$, $\theta$
${\{\vec{x}_t, y_t\}}_{t=1}^{T}$: training data
$w$: window size for the reactive learner
$\theta$: threshold for creating a new stable learner
Let $S$ be a stable learner
Let $R_w$ be a $w$-reactive learner
Let $C$ be a circular list of $w$ bits, each initially $0$
for $t \leftarrow 1$ to $T$ do
    $\hat{y}_S \leftarrow S$.Classify$(\vec{x}_t)$
    output $\hat{y}_S$
    $\hat{y}_R \leftarrow R_w$.Classify$(\vec{x}_t)$
    if $\hat{y}_S \neq y_t \wedge \hat{y}_R = y_t$ then
        $C$.set$(t)$
    else
        $C$.unset$(t)$
    end if
    if $\theta < C$.proportionOfSetBits$()$ then
        $S \leftarrow$ new StableLearner$()$
        $S \leftarrow R_w$.getConceptDescription$()$
        $C$.unsetAll$()$
    end if
    $S$.Train$(\vec{x}_t, y_t)$
    $R_w$.Train$(\vec{x}_t, y_t)$
end for
```

Bach, S.H.; Maloof, M., *"Paired Learners for Concept Drift"* ICDM '08.

Paired Learners

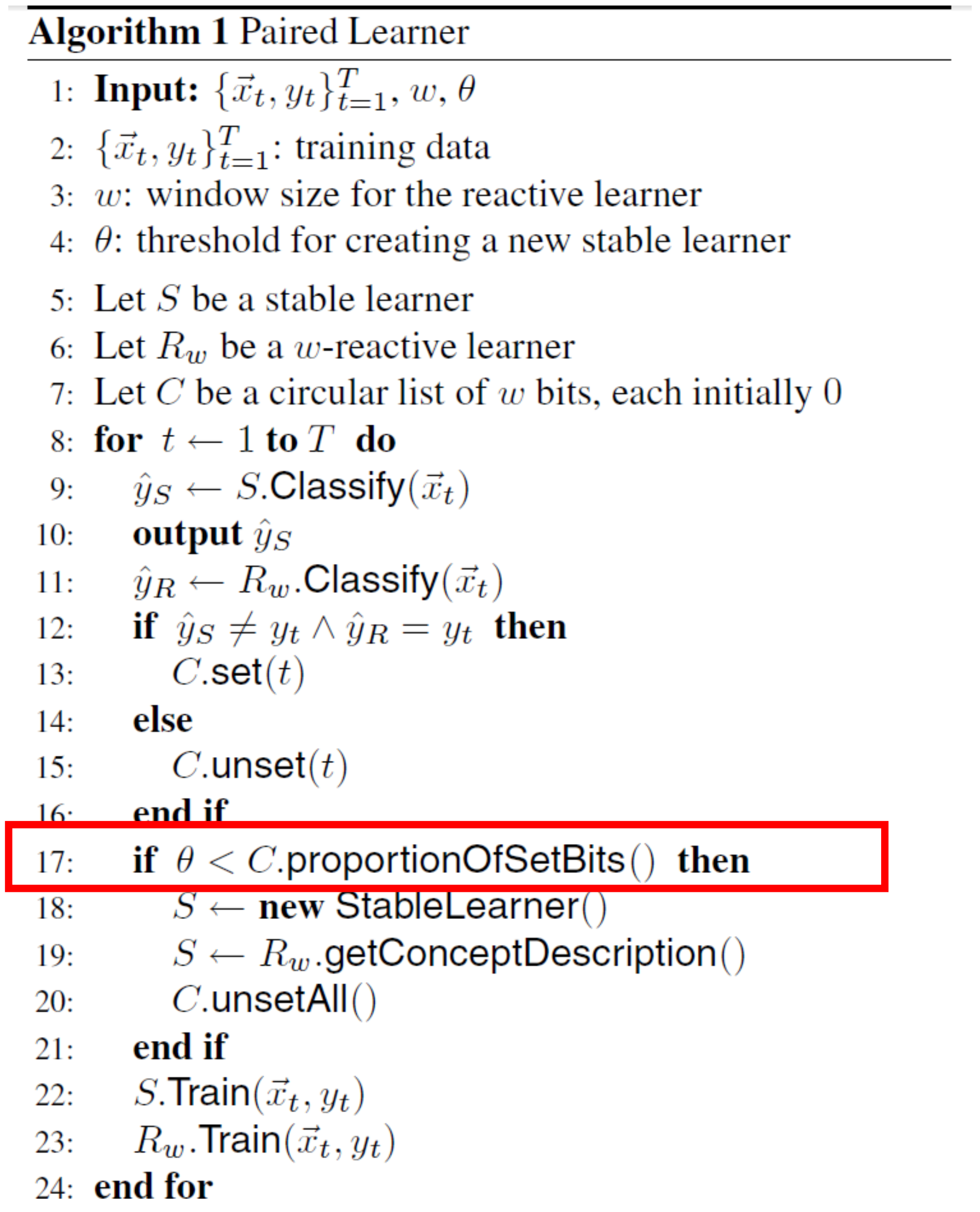
Predictions are only provided by S

Algorithm 1 Paired Learner

```
1: **Input:** $\{\vec{x}_t, y_t\}_{t=1}^{T}$, $w$, $\theta$
2: $\{\vec{x}_t, y_t\}_{t=1}^{T}$: training data
3: $w$: window size for the reactive learner
4: $\theta$: threshold for creating a new stable learner
5: Let $S$ be a stable learner
6: Let $R_w$ be a $w$-reactive learner
7: Let $C$ be a circular list of $w$ bits, each initially 0
8: **for** $t \leftarrow 1$ **to** $T$ **do**
9:     $\hat{y}_S \leftarrow S.\mathsf{Classify}(\vec{x}_t)$
10:    **output** $\hat{y}_S$
11:    $\hat{y}_R \leftarrow R_w.\mathsf{Classify}(\vec{x}_t)$
12:    **if** $\hat{y}_S \neq y_t \wedge \hat{y}_R = y_t$ **then**
13:        $C.\mathsf{set}(t)$
14:    **else**
15:        $C.\mathsf{unset}(t)$
16:    **end if**
17:    **if** $\theta < C.\mathsf{proportionOfSetBits}()$ **then**
18:        $S \leftarrow$ **new** $\mathsf{StableLearner}()$
19:        $S \leftarrow R_w.\mathsf{getConceptDescription}()$
20:        $C.\mathsf{unsetAll}()$
21:    **end if**
22:    $S.\mathsf{Train}(\vec{x}_t, y_t)$
23:    $R_w.\mathsf{Train}(\vec{x}_t, y_t)$
24: **end for**
```

Bach, S.H.; Maloof, M., *"Paired Learners for Concept Drift"* ICDM '08.

Paired Learners



Algorithm 1 Paired Learner

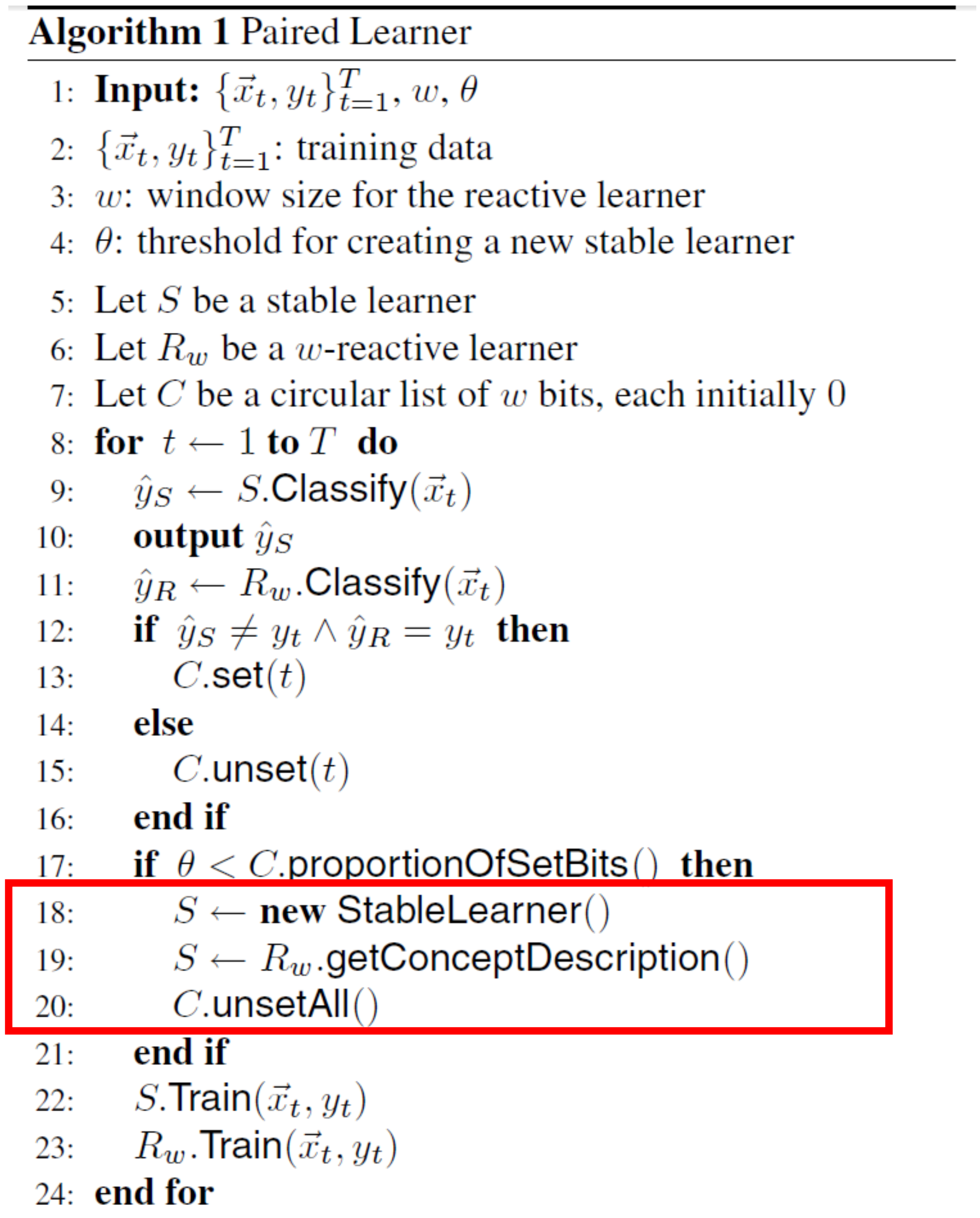
```
Input: ${\vec{x}_t, y_t}_{t=1}^T$, $w$, $\theta$
${\vec{x}_t, y_t}_{t=1}^T$: training data
$w$: window size for the reactive learner
$\theta$: threshold for creating a new stable learner
Let $S$ be a stable learner
Let $R_w$ be a $w$-reactive learner
Let $C$ be a circular list of $w$ bits, each initially 0
for $t \leftarrow 1$ to $T$ do
    $\hat{y}_S \leftarrow S$.Classify($\vec{x}_t$)
    output $\hat{y}_S$
    $\hat{y}_R \leftarrow R_w$.Classify($\vec{x}_t$)
    if $\hat{y}_S \neq y_t \wedge \hat{y}_R = y_t$ then
        $C$.set($t$)
    else
        $C$.unset($t$)
    end if
    if $\theta < C$.proportionOfSetBits() then
        $S \leftarrow$ new StableLearner()
        $S \leftarrow R_w$.getConceptDescription()
        $C$.unsetAll()
    end if
    $S$.Train($\vec{x}_t, y_t$)
    $R_w$.Train($\vec{x}_t, y_t$)
end for
```

Drift deteceted when more than θ times over the latest w samples, R_W provides a correct prediction while S does not

Bach, S.H.; Maloof, M., *"Paired Learners for Concept Drift"* ICDM '08.

Paired Learners

Adaptation: R_W replaces S and the error computation is reset
Detection suggests that samples before $t - w$ do not conform with the current status of the process



Algorithm 1 Paired Learner

```
Input: ${\vec{x}_t, y_t}_{t=1}^{T}$, $w$, $\theta$
${\vec{x}_t, y_t}_{t=1}^{T}$: training data
$w$: window size for the reactive learner
$\theta$: threshold for creating a new stable learner
Let $S$ be a stable learner
Let $R_w$ be a $w$-reactive learner
Let $C$ be a circular list of $w$ bits, each initially 0
for $t \leftarrow 1$ to $T$ do
    $\hat{y}_S \leftarrow S$.Classify($\vec{x}_t$)
    output $\hat{y}_S$
    $\hat{y}_R \leftarrow R_w$.Classify($\vec{x}_t$)
    if $\hat{y}_S \neq y_t \wedge \hat{y}_R = y_t$ then
        $C$.set($t$)
    else
        $C$.unset($t$)
    end if
    if $\theta < C$.proportionOfSetBits() then
        $S \leftarrow$ new StableLearner()
        $S \leftarrow R_w$.getConceptDescription()
        $C$.unsetAll()
    end if
    $S$.Train($\vec{x}_t, y_t$)
    $R_w$.Train($\vec{x}_t, y_t$)
end for
```

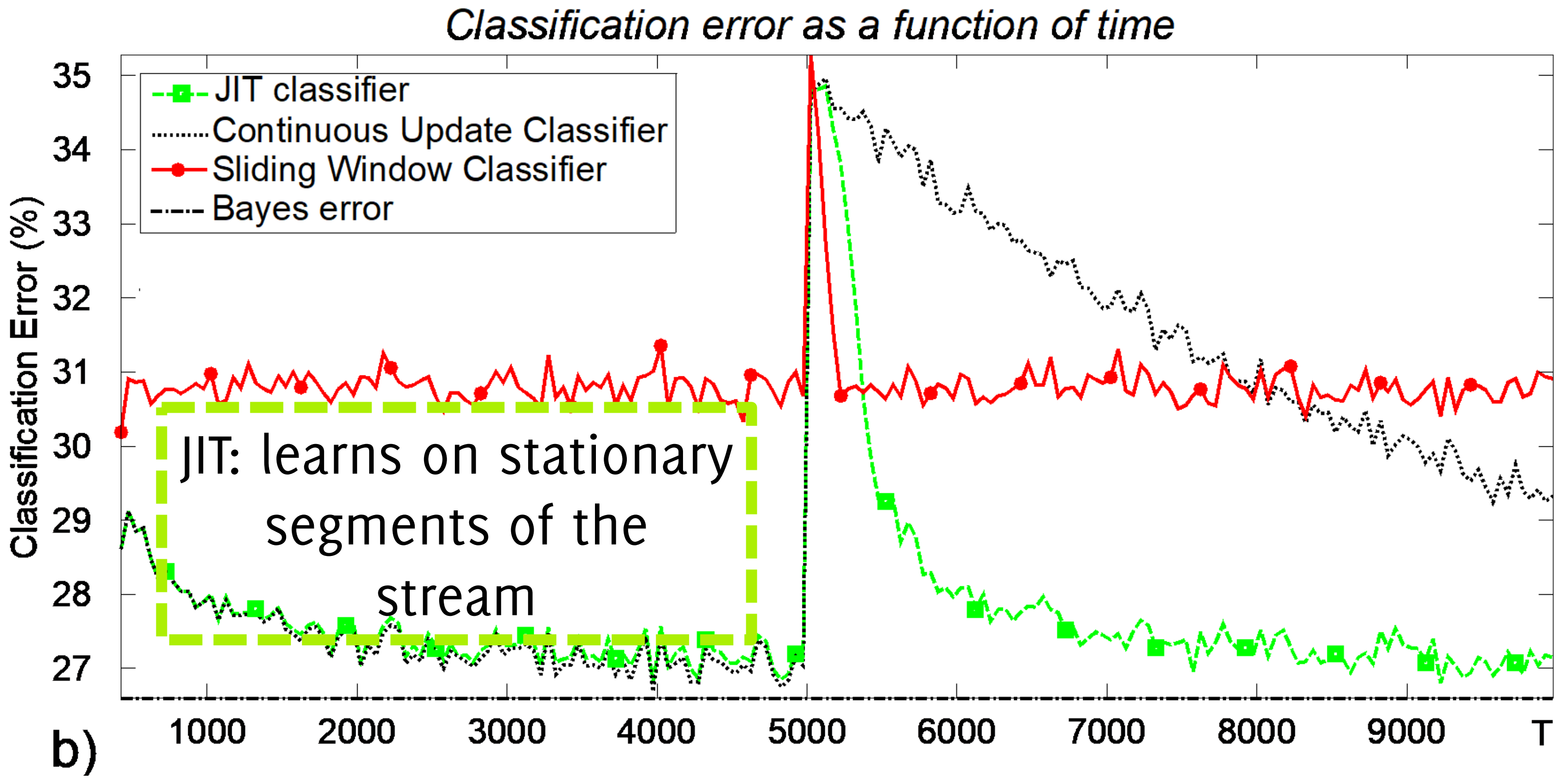
Bach, S.H.; Maloof, M., *"Paired Learners for Concept Drift"* ICDM '08.

JUST-IN-TIME Classifiers

JIT Classifiers

Idea:

- use different change-detection methods to segment the stream in stationary distributions
- After the change, recover useful knowledge from the past observed concepts



Boracchi

Just In Time Classifiers

JIT classifiers are described in terms of:

- **concept representations**
- **operators** for concept representations

JIT classifiers are **able to**:

- detect abrupt CD (both real or virtual)
- **identify a new training set** for the new concept and **exploit recurrent concepts**

JIT classifiers leverage:

- **sequential techniques to detect CD,** monitoring both classification error and raw data distribution
- **statistical techniques to identify the new concept and possibly recurrent ones**

An example of Concept Representations

$$C_i = (Z_i, F_i, D_i)$$

$Z_i = \{(\boldsymbol{x_0}, y_0), \dots, (\boldsymbol{x_n}, y_n)\}$: **supervised samples** provided during the i^{th} concept

F_i **features describing** $p(\boldsymbol{x})$ of the i^{th} concept. We take:

- the sample mean $M(\cdot)$
- the power-low transform of the sample variance $V(\cdot)$

  extracted from **non-overlapping sequences**

D_i **features for detecting** concept drift. These include:

- the sample mean $M(\cdot)$
- the power-low transform of the sample variance $V(\cdot)$
- the average classification error $p_t(\cdot)$

  extracted from **non-overlapping sequences**

In **stationary conditions** features are **i.i.d.**

JIT Classifiers: the Algorithm

```
1-  Build concept $C_0 = (Z_0, F_0, D_0)$ from the training sequence;
2-  $Z_{\text{rec}} = \emptyset$ and $i = 0$;
3-  while ($x_t$ is available) do
4-      $\mathcal{U}(C_i, \{x_t\}) \rightarrow C_i$;
5-      if ($y_t$ is available) then
6-          $\mathcal{U}(C_i, \{(x_t, y_t)\}) \rightarrow C_i$;
        end
7-      if ($\mathcal{D}(C_i) = 1$) then
8-          $i = i + 1$;
9-          $\Upsilon(C_{i-1}) \rightarrow (C_k, C_l)$;
10-         $C_i = C_l$;
11-         $C_{i-1} = C_k$;
12-         $Z_{\text{rec}} = \bigcup_{\substack{\mathcal{E}(C_i, C_j)=1 \\ 0 \leq j < i}} Z_j$;
        end
13-     if ($y_t$ is not available) then
14-         $\widehat{y}_t = K(Z_i \cup Z_{\text{rec}}, x_t)$.
        end
    end
```

Concept Representations

$$C = (Z, F, D)$$

- Z: set of supervised samples
- F: set of features for assessing concept equivalence
- D: set of features for detecting concept drift

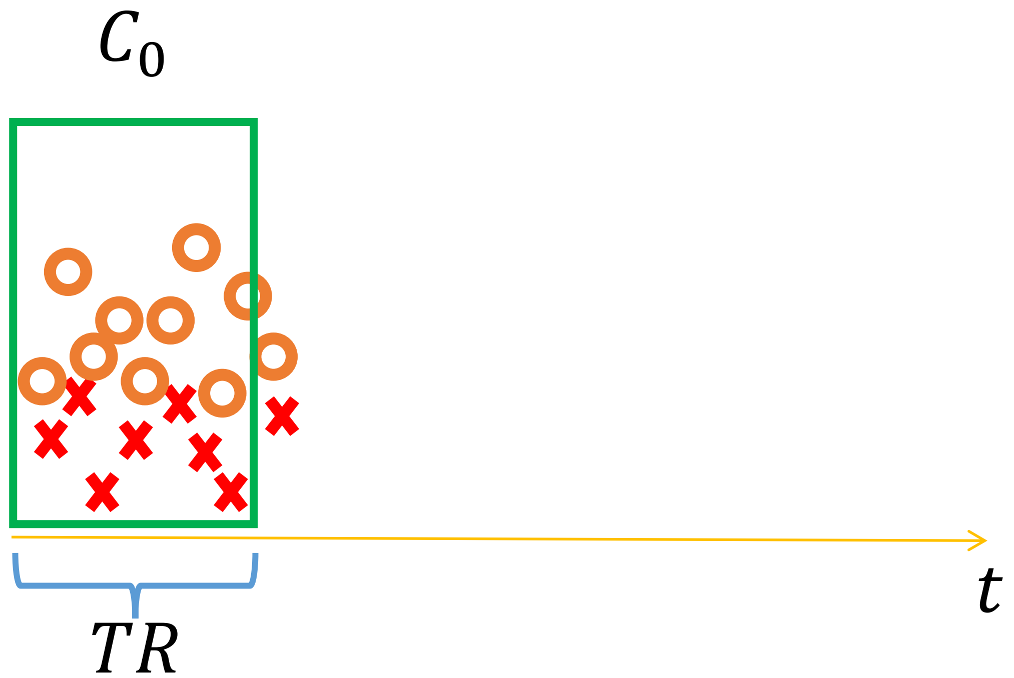
Initial Training

Use the initial training sequence to build the concept representation C_0

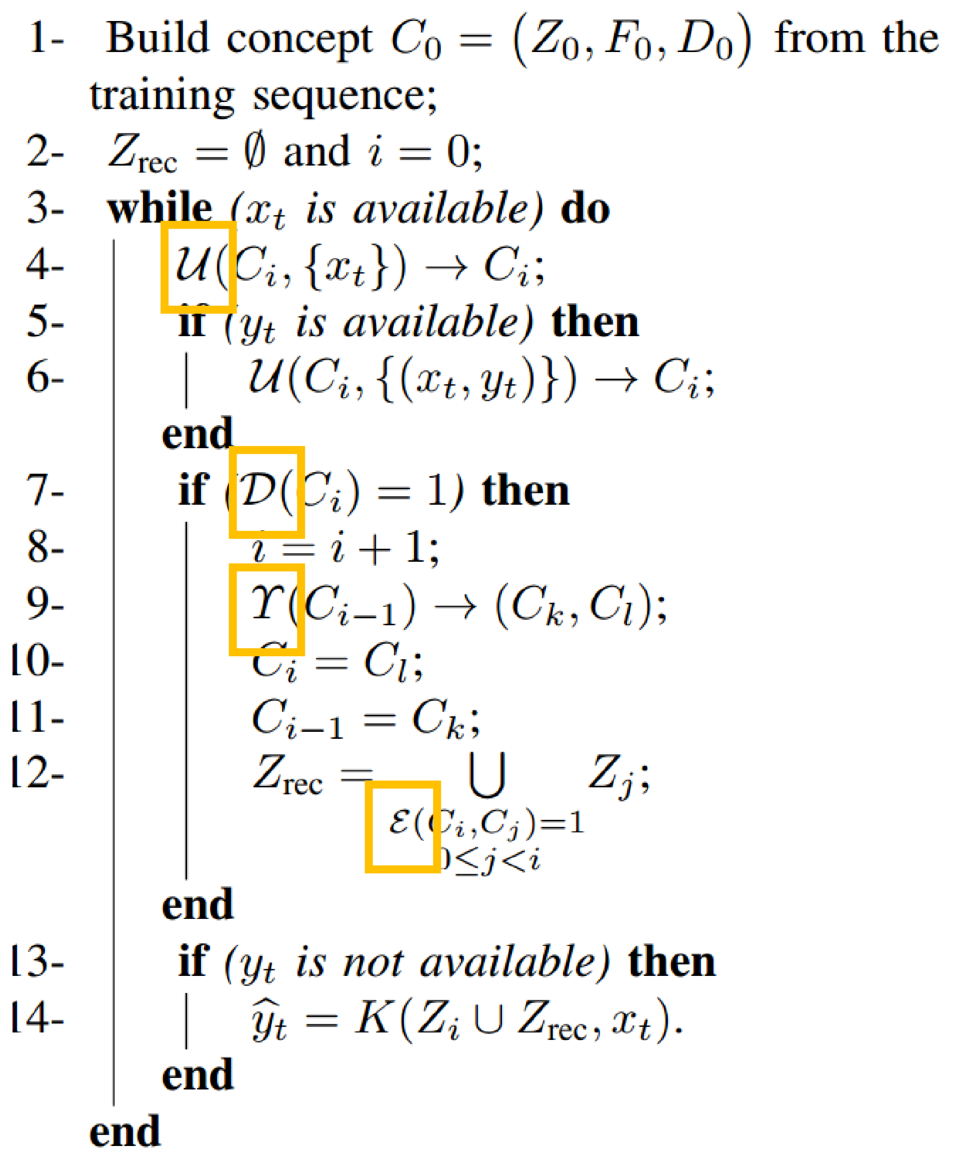
JIT Classifiers: Initial training

Build C_0, a **practical representation** of the **current concept**

- Characterize both $\phi(\boldsymbol{x})$ and $\phi(y|\boldsymbol{x})$ in stationary conditions



JIT Classifiers: the Algorithm



```
1-  Build concept $C_0 = (Z_0, F_0, D_0)$ from the training sequence;
2-  $Z_{\text{rec}} = \emptyset$ and $i = 0$;
3-  while ($x_t$ is available) do
4-      $\mathcal{U}(C_i, \{x_t\}) \rightarrow C_i$;
5-      if ($y_t$ is available) then
6-          $\mathcal{U}(C_i, \{(x_t, y_t)\}) \rightarrow C_i$;
        end
7-      if ($\mathcal{D}(C_i) = 1$) then
8-          $i = i + 1$;
9-          $\Upsilon(C_{i-1}) \rightarrow (C_k, C_l)$;
10-         $C_i = C_l$;
11-         $C_{i-1} = C_k$;
12-         $Z_{\text{rec}} = \bigcup_{\substack{\mathcal{E}(C_i, C_j) = 1 \\ 0 \leq j < i}} Z_j$;
        end
13-     if ($y_t$ is not available) then
14-         $\widehat{y}_t = K(Z_i \cup Z_{\text{rec}}, x_t)$.
        end
    end
```

Concept Representations

$$C = (Z, F, D)$$

- Z : set of supervised samples
- F : set of features for assessing concept equivalence
- D : set of features for detecting concept drift

Operators for Concepts

- $\mathcal{D}$ concept-drift detection
- Υ concept split
- $\mathcal{E}$ equivalence operators
- $\mathcal{U}$ concept update

Boracchi

JIT Classifiers: the Algorithm

```
1-  Build concept $C_0 = (Z_0, F_0, D_0)$ from the
    training sequence;
2-  $Z_{\text{rec}} = \emptyset$ and $i = 0$;
3-  while ($x_t$ is available) do
4-      $\mathcal{U}(C_i, \{x_t\}) \rightarrow C_i$;
5-      if ($y_t$ is available) then
6-          $\mathcal{U}(C_i, \{(x_t, y_t)\}) \rightarrow C_i$;
        end
7-      if ($\mathcal{D}(C_i) = 1$) then
8-          $i = i + 1$;
9-          $\Upsilon(C_{i-1}) \rightarrow (C_k, C_l)$;
10-         $C_i = C_l$;
11-         $C_{i-1} = C_k$;
12-         $Z_{\text{rec}} = \bigcup_{\substack{\mathcal{E}(C_i, C_j)=1 \\ 0 \leq j < i}} Z_j$;
        end
13-     if ($y_t$ is not available) then
14-         $\widehat{y}_t = K(Z_i \cup Z_{\text{rec}}, x_t)$.
        end
    end
```

Concept Update:

During operations, each input sample is analyzed to:

- Extract features that are appended to F_i
- Append supervised information in Z_i

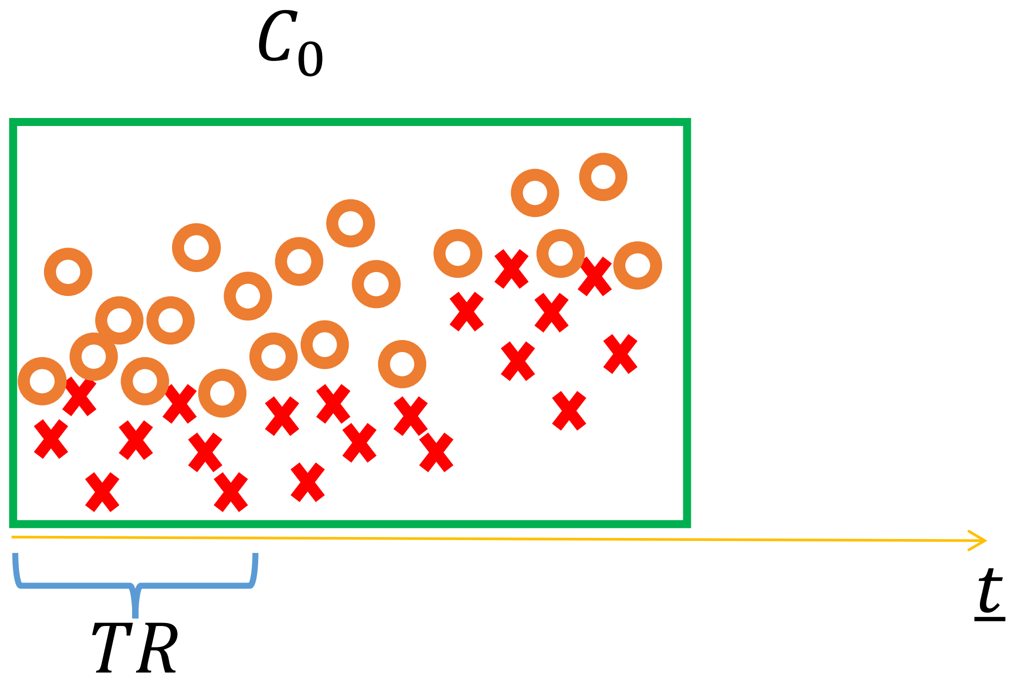
thus updating the current concept representation

Boracchi

JIT Classifiers: Concept Update

The **concept representation** C_0 is **always updated** during operation,

- Including supervised samples in Z_0 (to describe $p(y|\boldsymbol{x})$)
- Computing feature F_0 (to describe $p(\boldsymbol{x})$)
- Computing feature D_0



JIT Classifiers: the Algorithm

```
1-  Build concept $C_0 = (Z_0, F_0, D_0)$ from the
    training sequence;
2-  $Z_{\text{rec}} = \emptyset$ and $i = 0$;
3-  **while** *($x_t$ is available)* **do**
4-      $\mathcal{U}(C_i, \{x_t\}) \to C_i$;
5-      **if** *($y_t$ is available)* **then**
6-          $\mathcal{U}(C_i, \{(x_t, y_t)\}) \to C_i$;
        **end**
7-      **if** *($\mathcal{D}(C_i) = 1$)* **then**
8-          $i = i + 1$;
9-          $\Upsilon(C_{i-1}) \to (C_k, C_l)$;
10-         $C_i = C_l$;
11-         $C_{i-1} = C_k$;
12-         $Z_{\text{rec}} = \bigcup_{\substack{\mathcal{E}(C_i, C_j)=1 \\ 0 \le j < i}} Z_j$;
        **end**
13-     **if** *($y_t$ is not available)* **then**
14-         $\widehat{y}_t = K(Z_i \cup Z_{\text{rec}}, x_t)$.
        **end**
    **end**
```

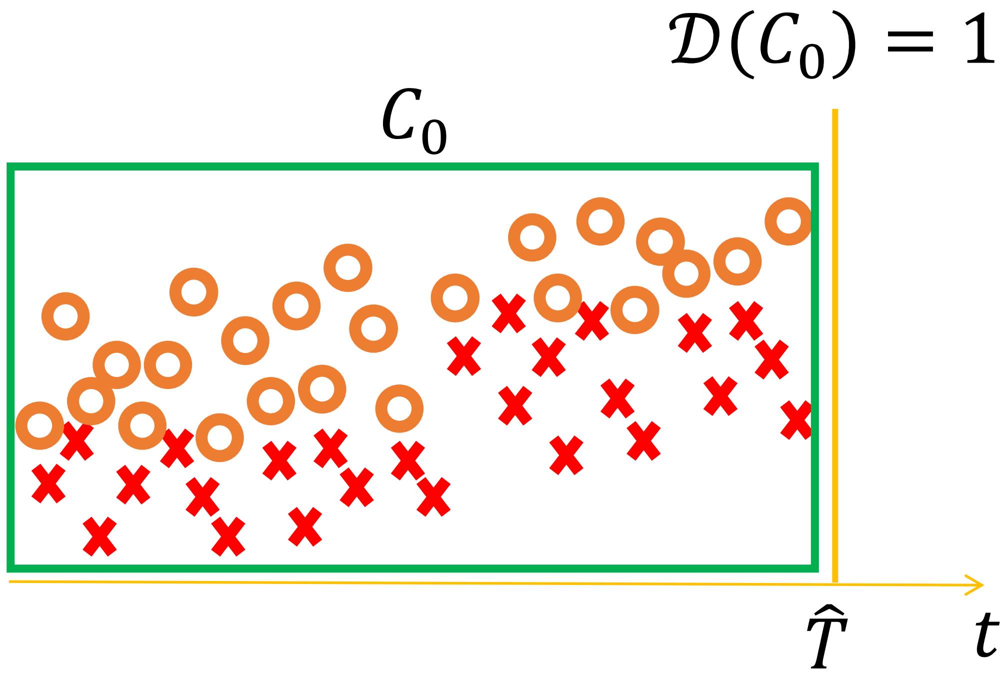
Concept Drift Detection:

The current concept representation is analyzed by $\mathcal{D}$ to determine whether concept drift has occurred

JIT Classifiers: Concept Drift Detection

Determine when **features in *D*** are no more stationary

- $\mathcal{D}$ monitoring the datastream by means of **online** and **sequential change-detection tests** (CDTs)
- Depending on features, both changes in $\phi(y|\boldsymbol{x})$ and $\phi(\boldsymbol{x})$ can be detected
- $\hat{T}$ is the detection time



An example of detection operator

$$\mathcal{D}(C_i) \in \{0,1\}$$

Implements **online** change-detection tests (**CDTs**) based on the **Intersection of Confidence Intervals** (ICI) rule

The ICI-rule is an adaptation technique used to define adaptive supports for polynomial regression

The ICI-rule determines when feature sequence (D_i) cannot be fit by a zero-order polynomial, thus **when D_i is non stationary**

ICI-rule requires **Gaussian**-distributed **features** but **no assumptions on the post-change distribution**

A. Goldenshluger and A. Nemirovski, *"On spatial adaptive estimation of nonparametric regression"* Math. Meth. Statistics,1997.

V. Katkovnik, *"A new method for varying adaptive bandwidth selection"* IEEE Trans. on Signal Proc, vol. 47, pp. 2567–2571, 1999.

JIT Classifiers: the Algorithm

```
1-  Build concept $C_0 = (Z_0, F_0, D_0)$ from the
    training sequence;
2-  $Z_{\text{rec}} = \emptyset$ and $i = 0$;
3-  while ($x_t$ is available) do
4-      $\mathcal{U}(C_i, \{x_t\}) \rightarrow C_i$;
5-      if ($y_t$ is available) then
6-          $\mathcal{U}(C_i, \{(x_t, y_t)\}) \rightarrow C_i$;
        end
7-      if ($\mathcal{D}(C_i) = 1$) then
8-          $i = i + 1$;
9-          $\Upsilon(C_{i-1}) \rightarrow (C_k, C_l)$;
10-         $C_i = C_l$;
11-         $C_{i-1} = C_k$;
12-         $Z_{\text{rec}} = \bigcup_{\substack{\mathcal{E}(C_i, C_j)=1 \\ 0 \leq j < i}} Z_j$;
        end
13-     if ($y_t$ is not available) then
14-         $\widehat{y}_t = K(Z_i \cup Z_{\text{rec}}, x_t)$.
        end
    end
```

Concept Split

Divide the current concept C_{i-1} into two concepts C_k and C_l, with the former being consistent with the current state of the process

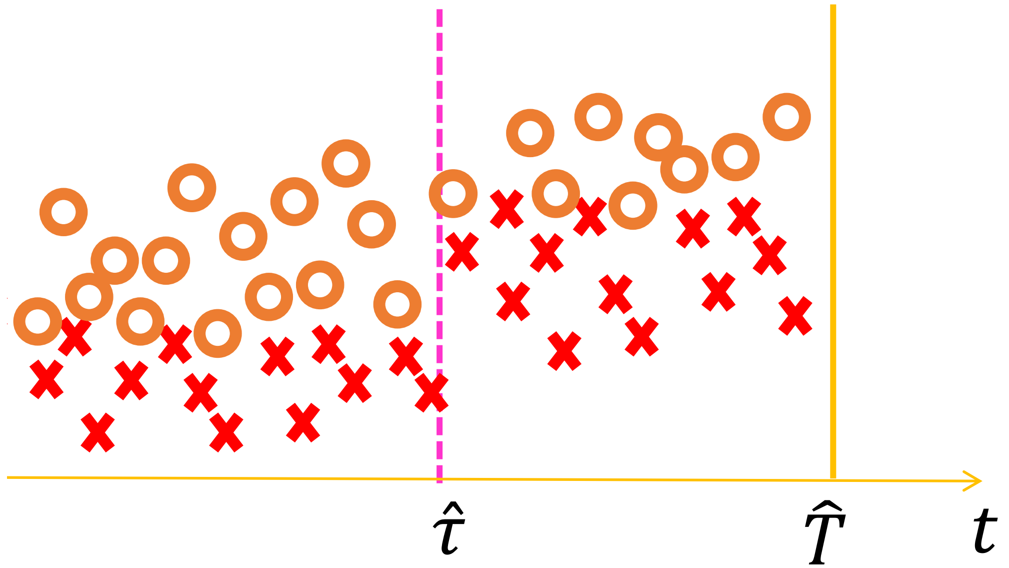
Boracchi

JIT Classifiers: Concept Split

Goal: estimating the change point τ (detections are always delayed). Samples in between $\hat{\tau}$ and $\hat{T}$

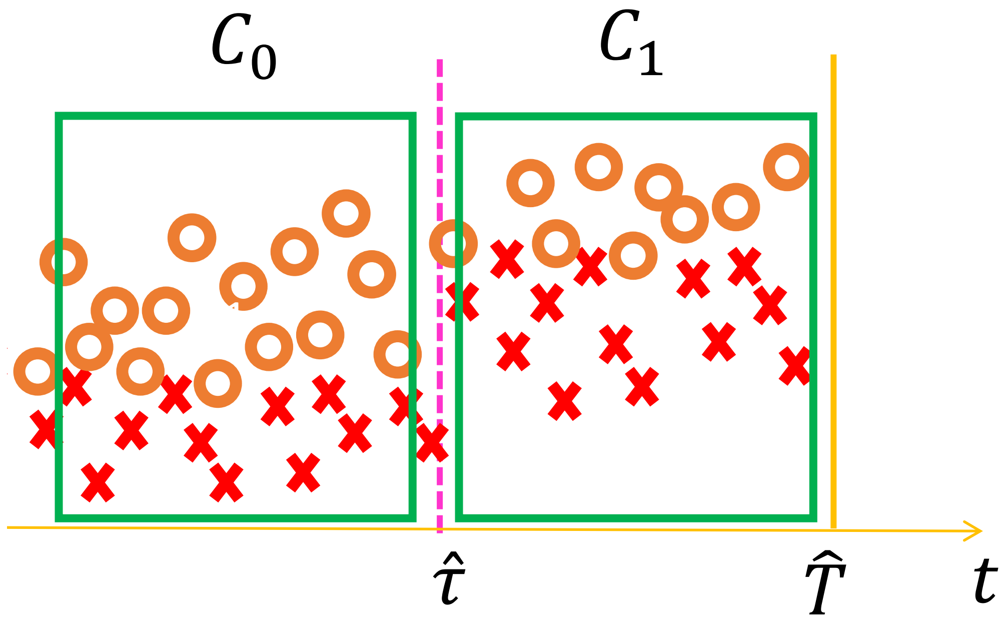
Uses statistical tools for performing an **offline** and **retrospective analysis** over the recent data, like:

- as hypothesis tests (HT)
- change-point methods (CPM)



JIT Classifiers: Concept Split

Given $\hat{\tau}$, two different concept representations are built



Examples of Concept Split Operator

$$\Upsilon(C_0) = (C_0, C_1)$$

It performs an **offline analysis** on F_i (just the feature detecting the change) to estimate **when concept drift has actually happened**

Detections $\hat{T}$ are delayed w.r.t. the actual change point τ

Change-Point Methods implement the following hypothesis test on the feature sequence:

$$\begin{cases} H_0: \text{"}F_i \text{ contains i.i.d. samples"} \\ H_1: \text{"}F_i \text{ contains a change point"} \end{cases}$$

testing all the possible partitions of F_i and determining the most likely to contain a change point

ICI-based CDTs implement a refinement procedure to estimate τ after having detected a change at $\hat{T}$.

JIT Classifiers: the Algorithm

```
1-  Build concept $C_0 = (Z_0, F_0, D_0)$ from the training sequence;
2-  $Z_{\text{rec}} = \emptyset$ and $i = 0$;
3-  while ($x_t$ is available) do
4-      $\mathcal{U}(C_i, \{x_t\}) \to C_i$;
5-      if ($y_t$ is available) then
6-          $\mathcal{U}(C_i, \{(x_t, y_t)\}) \to C_i$;
        end
7-      if ($\mathcal{D}(C_i) = 1$) then
8-          $i = i + 1$;
9-          $\Upsilon(C_{i-1}) \to (C_k, C_l)$;
10-         $C_i = C_l$;
11-         $C_{i-1} = C_k$;
12-         $Z_{\text{rec}} = \bigcup_{\substack{\mathcal{E}(C_i, C_j)=1 \\ 0 \leq j < i}} Z_j$;
        end
13-     if ($y_t$ is not available) then
14-         $\widehat{y}_t = K(Z_i \cup Z_{\text{rec}}, x_t)$.
        end
    end
```

Concept Equivalence

Look for concepts that are equivalent to the current one.

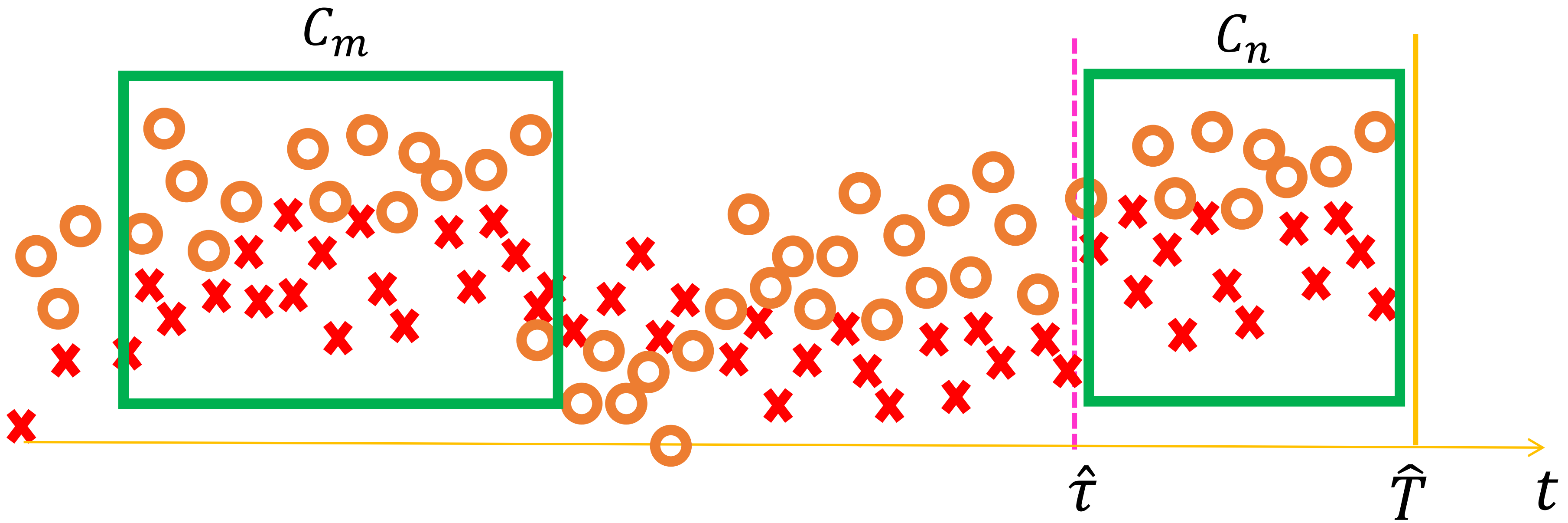
Gather supervised samples from all the representations C_j that refers to the same concept

JIT Classifiers: Comparing Concepts

Concept equivalence is assessed by

- comparing features F to determine whether $\phi(\boldsymbol{x})$ is the same on C_m and C_n using a **test of equivalence**
- comparing classifiers trained on C_m and C_n to determine whether $\phi(y|\boldsymbol{x})$ is the same

$$\mathcal{E}(C_m, C_n) = 1$$



Testing for Equivalence

Conventional HTs are meant to assess if two populations are different and **assume under H_0 that "they are the same"**

When there is not enough statistical evidence to conclude that the two populations are different, there is no hint on whether the two populations are the same.

We use Two One-Sided t-Test (TOST) to assess equivalence of F_0 and F_1.

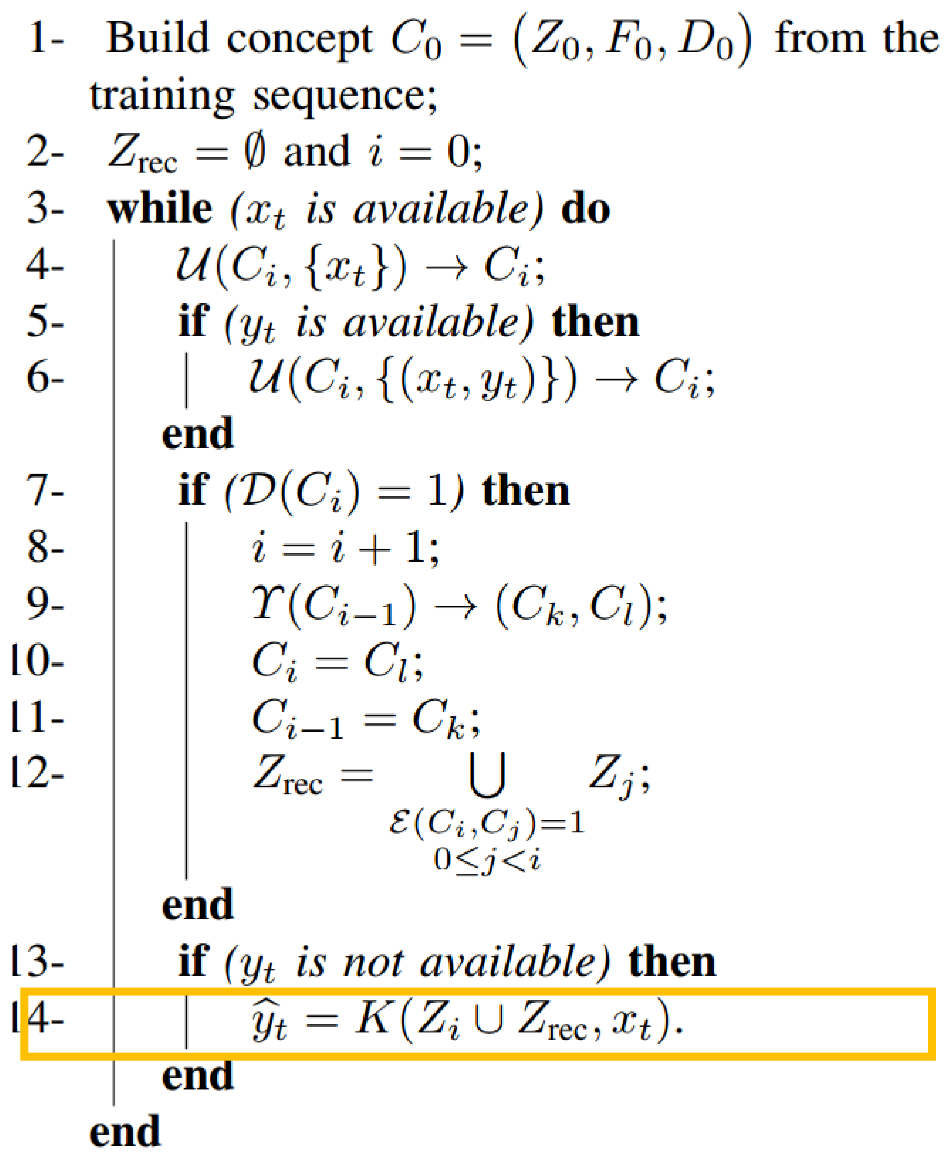
In TOST, H_0 corresponds to the non-equivalence of the two populations. Discarding H_0 implies accepting that the two populations are equivalent.

$$H_0 : \text{``}\left(\overline{F}_j^{\kappa} - \overline{F}_i^{\kappa}\right) < \theta_L \text{ or } \left(\overline{F}_j^{\kappa} - \overline{F}_i^{\kappa}\right) > \theta_U \text{,''}$$

$$H_1 : \text{``}\theta_L \leq \overline{F}_j^{\kappa} - \overline{F}_i^{\kappa} \leq \theta_U \text{,''}$$

[40] P. Bauer and M. Kieser, "A unifying approach for confidence intervals and testing of equivalence and difference," Biometrika, vol. 83, no. 4, pp. pp. 934–937, 1996. http://www.jstor.org/stable/2337298

JIT Classifiers: the Algorithm



```
1-  Build concept $C_0 = (Z_0, F_0, D_0)$ from the
    training sequence;
2-  $Z_{\text{rec}} = \emptyset$ and $i = 0$;
3-  while ($x_t$ is available) do
4-      $\mathcal{U}(C_i, \{x_t\}) \rightarrow C_i$;
5-      if ($y_t$ is available) then
6-          $\mathcal{U}(C_i, \{(x_t, y_t)\}) \rightarrow C_i$;
        end
7-      if ($\mathcal{D}(C_i) = 1$) then
8-          $i = i + 1$;
9-          $\Upsilon(C_{i-1}) \rightarrow (C_k, C_l)$;
10-         $C_i = C_l$;
11-         $C_{i-1} = C_k$;
12-         $Z_{\text{rec}} = \bigcup_{\substack{\mathcal{E}(C_i, C_j) = 1 \\ 0 \leq j < i}} Z_j$;
        end
13-     if ($y_t$ is not available) then
14-         $\widehat{y}_t = K(Z_i \cup Z_{\text{rec}}, x_t)$.
        end
    end
```

Label Prediction:

The classifier K is reconfigured using all the available supervised couples

Boracchi

The Passive Approach

Classifiers undergoing continuous adaptation

Passive Approach

Passive approaches:

- **Do not have an explicit** CD **detection** mechanism
- They are **aware** that $\phi_t(\boldsymbol{x}, y)$ *might* change at any time / any rate
- **Perform continuous adaptation** of their model(s) parameters at each new arrival

They can be divided in:

- **Single model** methods
- **Ensemble** method

Passive Approach

- Overcomes the potential disadvantage of active methods that can get false alarms or delays in the detection of drift.
- Potentially **better suited** for **slow concept drifts.**
- **Potential disadvantage** not addressing concept drift detection: **don’t inform** of **whether concept drift is occurring**.

Single Classifier Models

Lower computational costs

Mainly related to specific classifiers

- CVFDT: Concept-adapting Very Fast Decision Tree learner, and online decision tree algorithm that incrementally learns from a sliding window

P. Domingos and G. Hulton, *“Mining high-speed data streams”* in Proc. of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 71–80, 2000.

G. Hulten, L. Spencer, and P. Domingos, *“Mining time-changing data streams”* in Proc. of Conference on Knowledge Discovery in Data, pp. 97–106, 2001.

Single Classifier Models

Lower computational costs

Mainly related to specific classifiers

- CVFDT: Concept-adapting Very Fast Decision Tree learner, and online decision tree algorithm that incrementally learns from a sliding window
- OLIN: fuzzy-logic based approach that exploits a sliding window

L. Cohen, G. Avrahami-Bakish, M. Last, A. Kandel, and O. Kipersztok, *"Real-time data mining of non-stationary data streams from sensor networks"*, Information Fusion, vol. 9, no. 3, pp. 344–353, 2008.

Single Classifier Models

Lower computational costs

Mainly related to specific classifiers

- CVFDT: Concept-adapting Very Fast Decision Tree learner, and online decision tree algorithm that incrementally learns from a sliding window
- OLIN: fuzzy-logic based approach that exploits a sliding window
- An Extreme Learning Machine has been also combined with a time-varying NN

Y. Ye, S. Squartini, and F. Piazza, *"Online sequential extreme learning machine in nonstationary environments"*, Neurocomputing, vol. 116, no. 20, pp. 94–101, 2013

Ensemble methods

A Dilemma of Sorts

Stability

The ability of an algorithm to recall old information that it has learned in the past

Plasticity

The ability for an algorithm to learn new information when data are available

Sounds like we could have two opposing ideas!

Ensemble Methods

An **ensemble** of **multiple models** is preserved in memory

$$\mathcal{H} = \{h_0, \dots, h_N\}$$

Each **individual** $h_i, i = 1, \dots, N$ is typically trained from a different training set and could be from different models

Final prediction of the ensemble is given by (weighted) **aggregation of the individual predictions**

$$\mathcal{H}(\boldsymbol{x}_t) = \underset{\boldsymbol{y} \in \boldsymbol{\Lambda}}{\operatorname{argmax}} \sum_{\boldsymbol{h}_i \in \boldsymbol{\mathcal{H}}} \alpha_i \, [h_i(\boldsymbol{x}_t) = y]$$

Typically, one assumes data arrives in **batches** and each classifier is trained over a batch

Ensemble Methods

An **ensemble** of **multiple models** is preserved in memory

$$\mathcal{H} = \{h_0, \dots, h_N\}$$

The weight α_i encodes how reliable the prediction from h_i is **at the current time**.

Different methods set different weighting schemes, which are typically based on the posterior of h_i or the accuracy of h_i over recent data

$$\mathcal{H}(\boldsymbol{x}_t) = \underset{\boldsymbol{y}\in\boldsymbol{\Lambda}}{\text{argmax}} \sum_{\boldsymbol{h}_i\in\boldsymbol{\mathcal{H}}} \alpha_i\, [h_i(\boldsymbol{x}_t) = y]$$

Typically, one assumes data arrives in **batches** and each classifier is trained over a batch

Ensemble Methods and Concept Drift

Each individual h_i implicitly refers to a component of a mixture distribution characterizing a **concept**

Often, ensemble methods assume data (supervised and unsupervised) are provided in batches

Adaptation can be achieved by:

- **updating each individual**: either in batch or online manner
- **dynamic aggregation**: adaptively defining weights $\alpha_i(t)$
- **structural update**: including new (pruning old) individuals in the ensemble, possibly recovering past ones that are useful in case of recurrent concepts

Kuncheva, L. I. "*Classifier ensembles for changing environments*" In Workshop on Multiple Classifier Systems. MCS. 1–15 2004.

Ensemble Methods and Concept Drift

Ensemble based approaches provide a **natural fit** to the problem **of learning in nonstationary settings**,

- Ensembles tend to be more accurate than single classifier-based systems due to **reduction in the variance of the error**
- **Stability:** flexible to easily incorporate new data into a classification model, simply by **adding new individuals** to the ensemble and by updating each individual
- **Plasticity:** provide a natural mechanism **to forget irrelevant knowledge**, by simply **removing** old **individual(s)** from the ensemble
- They can operate in **continuously drifting environments**

Adaptive strategies can be applied to add/remove classifiers by on individual classifier and the ensemble error

Kuncheva, L. I. *"Classifier ensembles for changing environments"* In Workshop on Multiple Classifier Systems. MCS. 1–15 2004.

Streaming Ensemble Algorithm: SEA

An **ensemble of a fixed number** of individuals $\mathcal{H}$ performs

- **batch learning**
- **structural update** to adapt to concept drift

Two additional classifiers are stored h_t and h_{t-1}

- h_t is being trained on the current batch
- h_{t-1} is the classifier trained on the previous batch

W. N. Street and Y. Kim, *"A streaming ensemble algorithm (SEA) for large scale classification"*, in Proceedings to the 7th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 377–382, 2001

Streaming Ensemble Algorithm: SEA

When a new batch $S = \{(\boldsymbol{x}_0^t, y_0^t), (\boldsymbol{x}_1^t, y_1^t), \dots, (\boldsymbol{x}_B^t, y_B^t)\}$ arrives

- train h_t on S
- test h_{t-1} on S
- If the ensemble is not full $(\#\mathcal{H} < N)$, **add** h_{t-1} to $\mathcal{H}$
- Otherwise, **remove** $h_i \in \mathcal{H}$ that is **less accurate** on S (as far as this is worst than h_{t-1})

Classifier h_t is never added as its performance on the current batch are affected by overfitting

Adaptation to concept drift is performed by replacing individuals (no update of each individual instead)

Pruning the ensemble to improve the overall performance

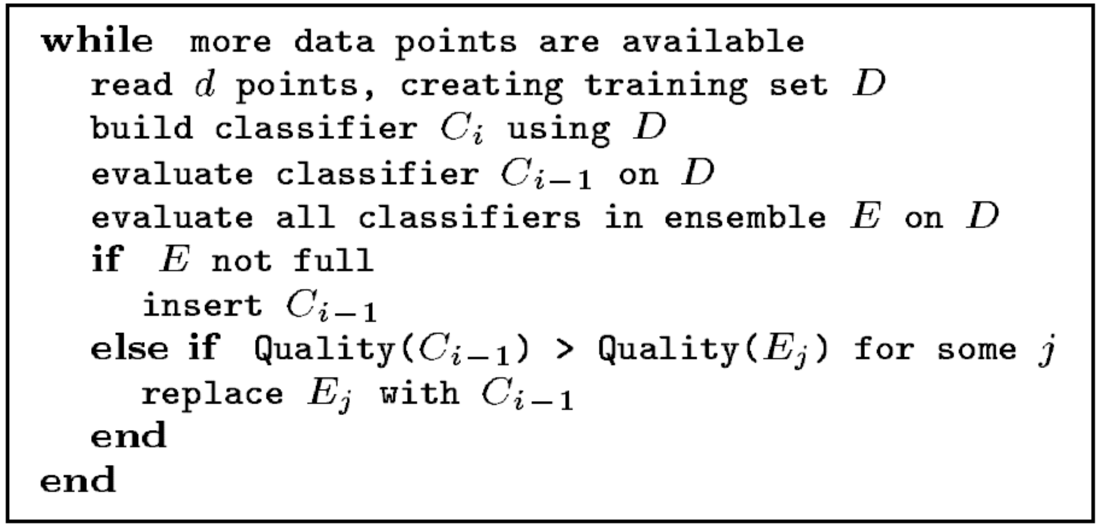
W. N. Street and Y. Kim, "*A streaming ensemble algorithm (SEA) for large scale classification*", in Proceedings to the 7th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 377-382, 2001

Streaming Ensemble Algorithm: SEA

The **individuals are decision trees**, which enables fast processing

Majority voting as aggregation strategy over the ensemble

“Quality” of an individual is an indicator to **favor individuals that correctly classify recent samples in *S* where the ensemble was “undecided”** providing a score close to 0.5 (in two class problems)



```
while more data points are available
  read $d$ points, creating training set $D$
  build classifier $C_i$ using $D$
  evaluate classifier $C_{i-1}$ on $D$
  evaluate all classifiers in ensemble $E$ on $D$
  if $E$ not full
    insert $C_{i-1}$
  else if Quality($C_{i-1}$) > Quality($E_j$) for some $j$
    replace $E_j$ with $C_{i-1}$
  end
end
```

Figure 1: Pseudocode for streaming ensembles. E_j **represents the** j**th tree in ensemble** E**.**

W. N. Street and Y. Kim, *"A streaming ensemble algorithm (SEA) for large scale classification"*, in Proceedings to the 7th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 377–382, 2001

Dynamic Weighted Majority: DWM

Dynamic weighted majority (DWM) is an ensemble where:

- **Individuals** are trained on different batches of data and **regularly updated** at a pre-defined frequency
- Each **individual is associated to a weight** $\{\alpha_i\}$
- Predictions are made by **weighted majority voting**
- **Weights** α_i **are decreased** to individuals that are **not accurate** on the samples of the current batch
- Individuals having **low weights are dropped**
- When **the ensemble makes a wrong guess**, a **new individual** is added
- **The ensemble size is also dynamic as it might vary over time**

Kolter, J. and Maloof, M. *"Dynamic weighted majority: An ensemble method for drifting concepts"*. Journal of Machine Learning Research 8, 2755–2790. 2007

Learns++ .NSE

Batch-learning algorithm performing predictions based on a **weighted majority voting** scheme:

- Two different **weighting schemes** for **individuals** and **training samples**
- **Misclassified training samples** receive **large weights**: samples from new concepts are often misclassified, thus receive large weights.
- **Weights of the individuals** depends on the **time-adjusted errors** on current and past batches: old individuals can be recovered in case of recurrent concepts
- Old individuals are not discarded

Elwell R. and Polikar R., "Incremental Learning of Concept Drift in Nonstationary Environments" IEEE TNNLS , vol. 22, 2011.

Concluding Remarks

Comments from my personal experience

In Learning problems the **classification error is typically the most important figure of merit**.

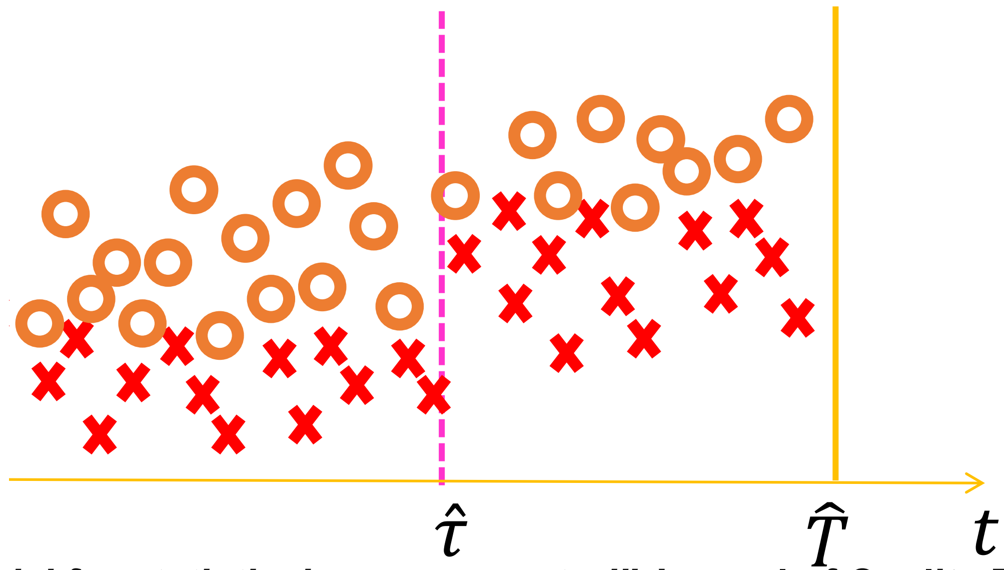
- In this scenario, **in general, false positives hurt less than detection delays**
- Things might change on class unbalance

Active approaches might be penalized due to their detection delay, while passive approaches might start adaptation earlier

Comments from my personal experience

Providing enough i.i.d. samples for **reconfiguration seems more critical.**
When estimating the change-time:

- Overestimates of τ provide too few samples
- Underestimates of τ provide non i.i.d. data
- Worth using accurate SPC methods like change-point methods (CPMs)



D. M. Hawkins, P. Qiu, and C. W. Kang, *"The changepoint model for statistical process control"* Journal of Quality Technology, 2003.

Comments from my personal experience

Exploiting recurrent concept is important

- Providing additional samples could make the difference
- Mitigate the impact of false positives

Comments from my personal experience

- Ensemble classifier approaches have had more success that single classifier implementations for nonstationary environments
- Hybrid approaches (active & passive) can be beneficial!
- In practice, a weighted majority vote is a better strategy as long as we have a reliable estimate of a classifiers error

Comments from my personal experience

We have combined

- a JIT classifier using recurrent concepts
- a sliding window classifier

As in paired learners,

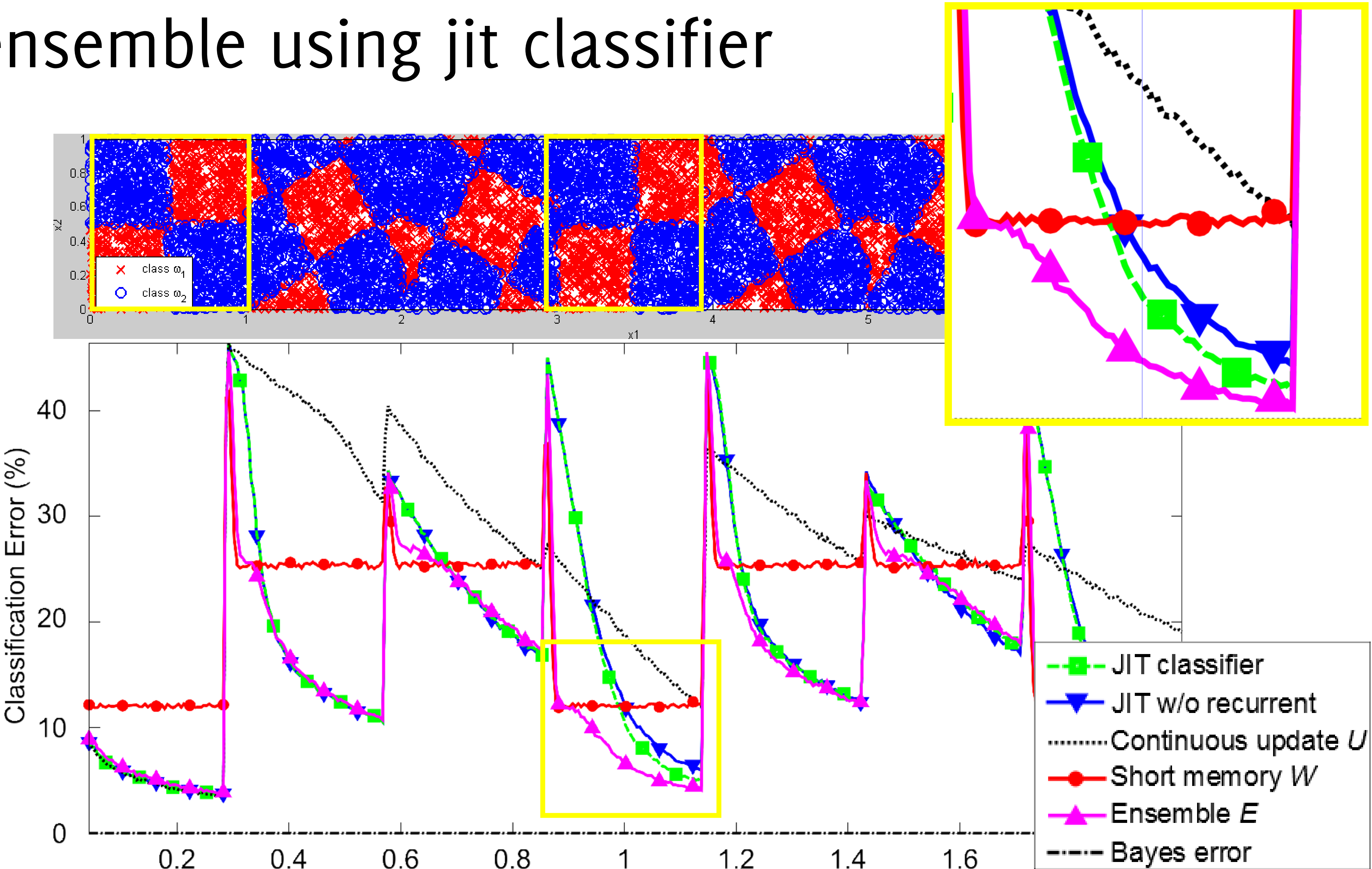
- JIT is meant to provide the best post-detection adaptation and best performance in a stationary state
- The sliding window classifier is meant to provide the quickest reaction to CD

We used a simple aggregation *"Predictions are made by the most accurate classifier over the last 20 samples"*

Actually, this ensemble performed very well, combining the advantages of the two classifiers

C. Alippi, G. Boracchi and M. Roveri, "*Just In Time Classifiers for Recurrent Concepts*" IEEE Transactions on Neural Networks and Learning Systems, 2013. vol. 24, no.4, pp. 620 -634

The ensemble using jit classifier


class ω1
class ω2
x1
x2
Classification Error (%)
JIT classifier
JIT w/o recurrent
Continuous update U
Short memory W
Ensemble E
Bayes error