Chapter 9
Learning in Nonstationary and Evolving Environments

Previous chapters have developed methods and methodologies for solving specific aspects involving intelligent processing on embedded systems and presented techniques for their performance assessment. However, if we look carefully at those methods, we can observe that we have commonly assumed that the process generating the data acquired by our sensors was not changing with time (stationarity or time invariance assumption).

We say that a data-generating process is *stationary* when generated data are i.i.d. realizations of a unique random variable whose distribution does not change with time. Thus, stationarity applies to stochastic processes. We say that a process is *time invariant* when its outputs do not explicitly depend on time. Less formally, in the former case the parameters characterizing the pdf do not change over time, in the latter, the transfer function of the-possibly dynamic-system does not have an explicit time dependency.

In some cases, nonstationarity and time invariance are related concepts. For instance, we will see in the chapter that inspection for time variance in some cases can be achieved by extracting features from the transfer function and verifying a potential change in stationarity. Stationarity was requested either directly by requiring i.i.d data streams or features, or indirectly, by requesting that the application or the model learned from the data was fixed before being implemented in the embedded system. Stationarity/time invariance is requested by performance assessment methods, e.g., the PACC framework, where the given Lebesgue measurable function, albeit affected by uncertainty, is fixed. The same holds for a robustness analysis that initially operates on a stationary/time invariant processing flow.

All in all, we mainly assume stationarity/time-invariance in our applications but are aware that such an assumption represents a first order, yet in many cases, a reasonable hypothesis for the data-generating process.

However, real-world processes are often affected by *concept drift*, i.e., changes in their inherent structure, which results in having the process departing from its initial, stationary (or time invariant) conditions. Concept drift could be due, for example, to a natural evolution of the environment, changes in the operational schema of a system, aging effects (e.g., structural changes in the transduction mechanism of a sensor), as well as faults affecting a cyber-physical system (e.g., abrupt or slowly developing drift). Nonstationarity and time variance can then be modeled as instances of concept drift. We have *gradual concept drift* when concept drift continuously evolves with time (e.g., a drift type of evolution). Conversely, we have an *abrupt concept drift* when concept drift is characterized by a sudden abrupt type of change (e.g., an abrupt type of evolution).

As an example, assume that the data-generating process admits a specific parametric expression $f(\theta, x) \in Y \subset \mathbb{R}$, $\theta \in \Theta \subset \mathbb{R}^d$, $x \in X \subset \mathbb{R}^l$, which can be either the transfer function of the system or the pdf in case of random variables. Let us consider the situation where the parameter vector is affected by a slowly developing concept drift, that shifts a given θ_0 toward a perturbed state, characterized by parameter vector $\theta_0 + \delta\theta$ belonging to the neighborhood of θ_0. By expanding with Taylor differentiable $f(\theta, x)$ in θ_0 we have that

$$f(\theta_0 + \delta\theta, x) = f(\theta_0, x) + \frac{\partial f}{\partial \theta}^T |_{\theta_0} \delta\theta + O(\delta^2\theta) \tag{9.1}$$

with $\frac{\partial f}{\partial \theta}$ and θ column vectors. The stationarity/time invariance hypothesis assumes that no perturbations affect the system or that the perturbation is negligible (the gradient term is negligible compared with the driving term $f(\theta_0, x)$). Expansion given in (9.1) has been written for a slowly developing concept drift continuously affecting the parameters. However, nothing changes if concept drift introduces an abrupt type of perturbation of small magnitude on θ_0.

Clearly, if we want to design effective intelligent embedded systems they must be able to deal with time invariant/nonstationary situations to guarantee good performance also in situations where the system or the environment where they operate in evolves with time. We name *learning in a nonstationary or evolving environments* all those aspects involving learning mechanisms for evolving environments.

The literature addressing learning in nonstationary or evolving environments classifies existing approaches as passive or active depending on the learning mechanism adopted to deal with the process evolution, e.g., [77]. We say that the approach is *passive* when the application undergoes a continuous training without explicitly knowing whether concept drift has occurred or not. Differently, within an *active* approach a triggering mechanism, e.g., a Change Detection Test (CDT), is considered to detect a change in the process generating the data and the application evolves and adapts only when the change has been detected.

This chapter presents passive and active methods for learning in an evolving environment. At first we introduce the learning approaches, detailing afterwards the key elements needed for a successful adaptation.

9.1 Passive and Active Learning

Learning in an evolving environment is specialized in the literature according to the chosen learning method, the way available data instances are used in the training process, and the type of envisioned application. Most learning methods follow either the active or the passive approach, depending on the way the available data instances are used both in the training phase and in the operational life. In nonstationary environments, both passive and active methods can be pursued to cope with concept drift: the most suitable method typically depends on the type of envisioned application.

9.1.1 Passive Learning

Since in passive approaches neither a priori nor derived information is available about the potential concept drift, we are completely blind about the fact a concept drift had, has, or will happen. Adaptation strategies must then be compulsive and carried out passively, without taking advantage of the information provided by incoming data. In fact, as new data come the application is reconfigured, adapted, or relearned depending on its nature and constraints. For instance, if a model M_1 is initially built from data, then a sequence of models $M_2, \ldots M_t$ is generated over time as data come during the operational life.

Passive methods can now be classified depending on the way incoming data are processed:

- *Online learning*. In online learning the new model M_t is obtained by updating the previous model M_{t-1} with data acquired at time t. To derive the online learning procedure we consider at first the traditional, offline, training mechanism given in Sect. 3.4.1. There the model parameters in θ are estimated over a training set Z_N by minimizing the empirical risk

  $$V_N(\theta, Z_N) = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, f(\theta, x_i)\right)$$

  leading to the estimate

  $$\hat{\theta} = \arg\min_{\theta\in\Theta} V_N(\theta, Z_N).$$

  Without any loss in generality, assume that function minimization is carried out by considering the straight backpropagation procedure implementing a simple gradient descent algorithm. Parameter θ at iteration $i+1$, i.e., θ_{i+1} can be expressed as

  $$\theta_{i+1} = \theta_i - \eta\frac{\partial V_N(\theta, Z_N)}{\partial\theta}|_{\theta_i}$$

where η is the learning rate. Training stops when some terminal conditions are met.
In online learning, the application-model is continuously trained by exploiting new instances and $V_N(\theta, Z_N)$ simplifies as

$$V_N(\theta, \{(x_i, y_i)\}) = L\,(y_i, f(\theta, x_i))$$

where (x_i, y_i) is the running supervised sample provided at time i. Parameters update becomes

$$\theta_{i+1} = \theta_i - \eta \frac{\partial L\,(y_i, f(\theta, x_i))}{\partial \theta}|_{\theta_i} \tag{9.2}$$

and η is a sufficiently small positive scalar in the simplest version of backpropagation. Many variants exist; the interested reader can refer to [100] for further details. Not rarely, the loss function is the squared function $L(y_i, f(\theta, x_i)) = (y_i - f(\theta, x_i))^2$. It should be noted that, at each time instant $i+1$, a new couple (x_{i+1}, y_{i+1}) is given and the procedure is reiterated. The training algorithm is shown to converge to the optimal value of parameters minimizing the empirical risk, provided that the loss function is quadratic and a sufficiently small η is given.
Since a sample-by-sample training method can be a time-consuming operation, a batch modality can be introduced to mitigate such an issue and stabilize the learning procedure. Thus, model parameters are updated asynchronously, at specific *time events* when a batch of n data is made available. All data instances in the batch are considered to have the same relevance and could be gathered from—possibly overlapping—n-dimensional data windows, even if the overlapping mechanism does not find any justifiable reason. If we denote by $Z_{n,i} = \{(x_i, y_i), (x_{i-1}, y_{i-1}), \ldots (x_{i-n+1}, y_{i-n+1})\}$ the batch of n data at time event i, then the learning procedure following the (9.2) becomes

$$\theta_{i+1} = \theta_i - \eta \frac{\partial V_N(\theta, Z_{n,i})}{\partial \theta}|_{\theta_i}. \tag{9.3}$$

- *Ensemble learning*. In ensemble learning, several individual models are activated at the same time instant, and the output of the ensemble is obtained by aggregating the outputs of each individual model. There are no specific restrictions about the individual models, which could be trained and updated during the operational life according to an online learning scheme. We comment that models to be selected neither need to belong to the same model family (the dimension of the parameter vector θ may vary) nor the same model hierarchy (models are generic entities not constrained to belong to the same model class).
  Most often, the aggregation consists in averaging the models' outputs, namely the ensemble provides a weighted average of the outputs of the individual models. However, several different aggregation mechanisms have been considered in the rich literature, which does not necessarily address the issue of learning in evolving

environments. In fact, the use of ensemble of models is shown to be beneficial in many circumstances and this claim has theoretical justifications, e.g., [240], where it is proven that an ensemble of models behaves better than the single generic model, even though not necessarily better than the best performing one (which is however hard to identify in a noisy environment).
Let us denote by $M = \{M_i(\cdot), i = 1, \ldots, k\}$ the set of individual models composing the ensemble. Since the simplest aggregation mechanism in ensemble learning consists in a weighted average, the output of the ensemble in correspondence with instance x is

$$y(x) = \sum_{i=1}^{k} w_i M_i(x)$$

where $\{w_i, i = 1, \ldots, k\}$ are suitably chosen weights typically yielding a linear convex combination of the individual models' outputs.
A viable option to cope with concept drift is to assign larger weights to the individual models that have been more recently trained or updated. This method is effective in developing gradual concept drift where the time locality property surely holds. Differently, if we have an abrupt type of concept drift, then we introduce a spurious effect due to the "step" introduced by the abrupt change. The spurious effect vanishes after k time events if each time event is associated with a new model (after k time events models will be associated with data affected by the concept drift only and, therefore are coherent). In the specific case where each individual model M_i is trained at time event i in an online learning manner, the ensemble may act as a windowing over the k most recent models, mitigating or discarding those models older than k events.
Weights aggregation can be set in different ways, depending on the a priori information we have about the application or the developing class of concept drift. Within an instance selection framework, it is possible to select at first the individual models to be aggregated. For instance, we might select only $l < k$ models which are better suited for describing the current observation. The ensemble would provide output

$$y(x) = \sum_{i \in \mathscr{A}} w_i M_i(x),$$

where $\mathscr{A}$ is a set of cardinality l containing the indexes of the individual models selected. In this case we are able to better deal with an abrupt type of concept drift, still dealing with developing ones.
In other situations, we might set the weights depending on the accuracies of the individual models evaluated on a common validation or test set. For instance, if model $M_i(x)$ is characterized by a mean squared error in validation or test σ_i^2 then the weight w_i can be chosen as

$$w_i = \frac{\frac{1}{\sigma_i^2}}{\sum_{j=1}^{k} \frac{1}{\sigma_j^2}}.$$

When all the individual models need to be equally treated or when there is no a priori information about the effectiveness of each individual model, all weights are naturally set to $\frac{1}{k}$.
Weights can undergo adaptation following the evolution of the environment.

9.1.2 Active Learning

Active learning is a learning paradigm that assumes interaction between the learner and an oracle or some other information source. In the case of learning in nonstationary environments the oracle can be an automatic triggering mechanism able to detect concept drift. Such a triggering mechanism typically operates on features extracted from the data, that are assumed to be stationary when the data-generating process is stationary or time invariant, but are expected to propagate the effects of concept drift once a change arises. Typically, such triggering methods are change-detection tests (CDT) or Change-Point Methods (CPM), which will be described in detail in the rest of the chapter.

Once concept drift is detected the application/model/service must be retrained. We consider, as an illustrative example, the embedded system setup of Fig. 9.1 where the data stream provided by sensors feeds the application/service and is inspected by either a CDT or a CPM. When the trigger detects a change in the data stream, the application/service is reconfigured/retrained by means of cognitive mechanisms, possibly exploiting additional data coming from nearby sensors, when these are inserted in a network. We refer to this paradigm as *detect and react*.

For instance, if an agent detects that the temperature sensor of my mobile is no more accurate (as a consequence of concept drift) an information exchange modality can be activated: the App will inspect nearby weather stations or other smartphones composing the local network to provide an estimate of the correct temperature. Calibration and compensation mechanisms are introduced as a reaction aspect on our unit. Also, the triggering mechanism might need to be retrained since its configuration might be obsolete having been configured in an old state. If no change is detected no reconfiguration is requested at the triggering mechanism and the application level.

When a network of distributed embedded systems is available, the situation is that depicted in Fig. 9.2. In such a case, intelligence can be present both at the embedded system and at the upper management level of the distributed system, where the information is collected and processed for decision making. If this is the case, communication among the components of the intelligent system must be present and activated to deliver relevant information.

The most simple triggering mechanisms are deterministic and based on fixed thresholds. When the features exceed the thresholds, a change is detected.

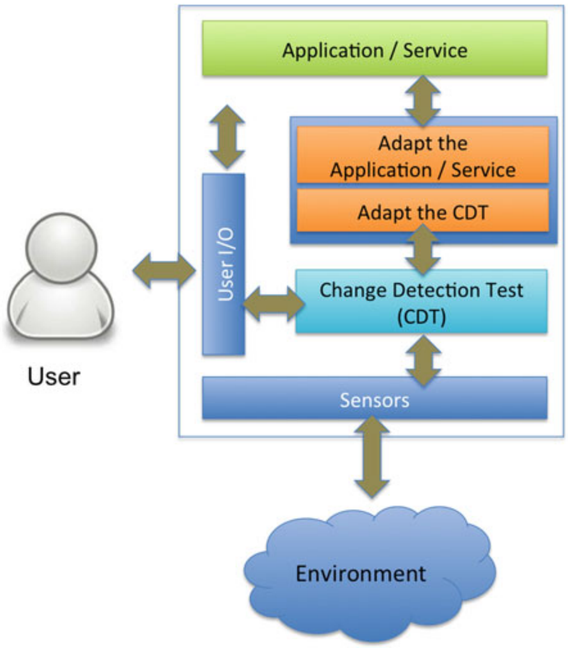

Fig. 9.1 The overall active detect and react methodology for an intelligent embedded system. A triggering mechanism inspects for concept drift (e.g., *through a CDT*). When concept drift is detected the application in execution must be adapted to track the change and, consequently, the CDT is reconfigured on the new operational state

Example: A Fixed Threshold

Consider a scalar feature x which is distributed as an i.i.d. random variable having expectation μ_x and variance σ_x^2. The feature could be the average classification error computed over time in a classification scenario, a measurement, or the result of an uncertainty-affected computation. Changes could be detected in a straightforward manner by setting a threshold $T = \lambda\sigma_x$, which, according to the Tchebychev inequality, implies that the probability of x exceeding T in stationary conditions is less than $\frac{1}{\lambda^2}$. All comments and derivations given in Chap. 3 also apply here.

That said, when $|x - \mu_x| > \lambda\sigma_x$ the threshold is violated and the trigger provides a positive response by detecting a change. The situation is that given in Fig. 9.3 where a threshold (the dashed line) is set. Once instances (samples) are above the threshold a change is detected.

However, since the realizations of x are independent, the probability of having false positives after n observations is $1 - (1 - \frac{1}{\lambda^2})^n$, which rapidly tends to 1 as n grows. By referring to Fig. 9.4 false positives are those data instances above the threshold line not associated with a true change.

We comment that a false positive might be an unpleasant—but not dramatic event—within a "detect and react" mechanism where the detection of concept drift is followed by a reaction that forces the application to undergo an adaptation phase. In fact, when false positives arise the activated reaction will introduce an unnecessary and undue computation with the effect that a new model/application/service will be

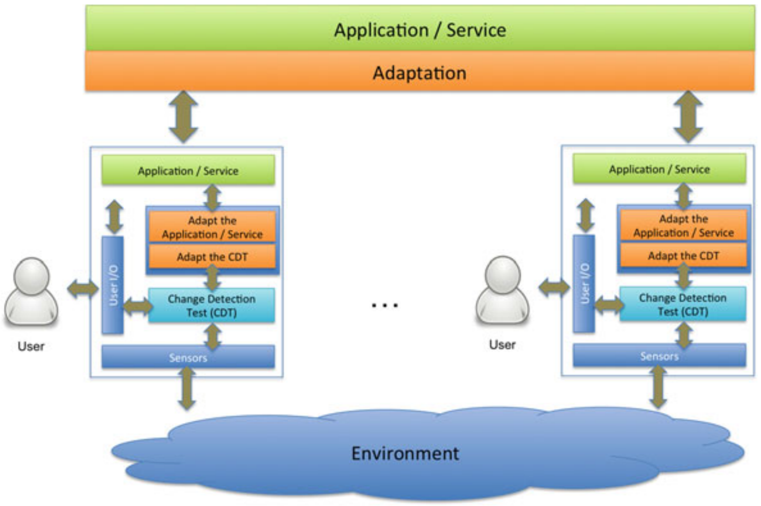

Fig. 9.2 The overall active detect and react methodology for a distributed intelligent system. The application/service is distributed and takes advantage of the information provided by the units composing the distributed platform. Adaptation can operate, with a simple strategy, at the embedded systems level and, at the same time, at the distributed network layer where more sophisticated algorithms can be executed. The outcome of the algorithm introduces adaptation at the distributed application layer and, thanks to remote reprogrammability, to the embedded distributed units

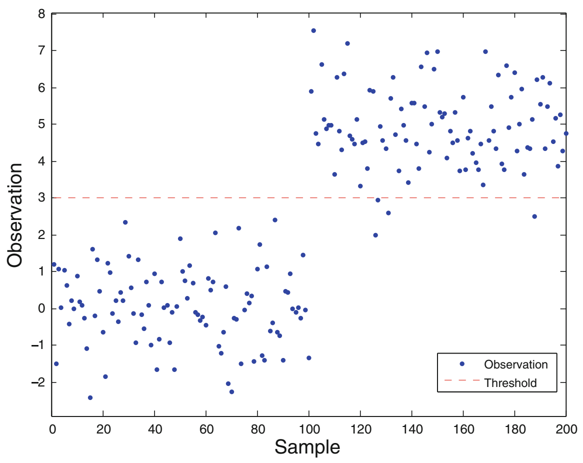

Fig. 9.3 Triggering with a fixed threshold: a change is detected when observation x is above a threshold value, here set to 3

configured even though it was not necessary. Running such unnecessary computation might be not appealing in embedded systems driven by strict real-time execution constraints and/or whenever energy consumption is an issue. As such, we should try to keep the false positive rate as small as possible.

In some other applications, a false positive might be a strongly unpleasant event. For instance, think of the case where a vision system detected in an airport a face

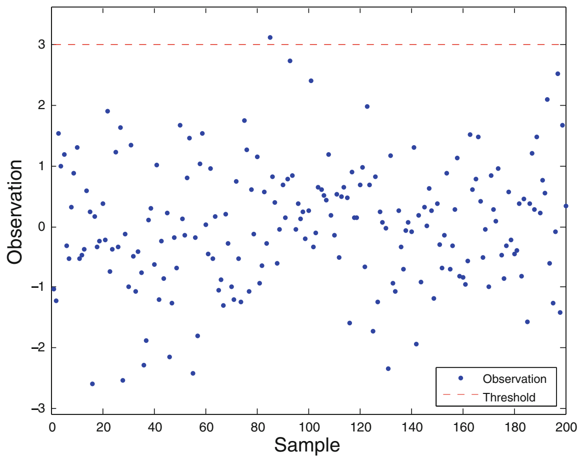

Fig. 9.4 Triggering with a fixed threshold: a false positive arises in correspondence to the sample over the threshold

of a “dead or alive wanted” person that was you! Another situation where false positives are not welcome is associated with fault detection. Here, false positives will erroneously claim that concept drift is associated with a fault in the plant, in a sensor, in a system module when the device is working properly. For the sensitivity and relevance of the issue we will address the fault diagnosis aspect in Chap. 10 where intelligence will play a main role.

To mitigate the above problem we might decide to introduce filters, e.g., a median, at the outputs of the triggering mechanism, hence implicitly assuming that if concept drift arises it will be of permanent and not of transient type. Although this solution is simple and might be effective in some applications and under specific assumptions about the nature of the fault, for others we might need a more accurate statistic-based triggering mechanism reducing false positives and negatives.

More sophisticated stochastic triggering mechanisms of CPM and CDT type will be presented in Sect. 9.2. We anticipate that the main difference between the two triggering methods resides in the way data are processed for decision making. CPMs operate on a fixed set of data to take a decision about the presence of concept drift, although some extensions have been proposed to mitigate the problem so as to address a sequential analysis. As such they are mostly inadequate to process streams of data. Moreover, the CPM computational cost might become prohibitive, making it hardly usable in embedded systems. On the other hand, CPMs are very effective in detecting concept drift, the false positive rate can be controlled at design time and latency in detection is low (CPMs show to be very responsive). Conversely, CDTs are able to operate at the data stream level, their computational cost is contained and, hence, suitable for intelligent embedded systems. The cost we have to pay is associated with an increased latency and the difficulty to guarantee a fixed false positive rate.

Tables 9.1 and 9.2 list the main stochastic triggering methods suggested in the related literature by classifying them according to the parametric/non parametric feature, respectively. Parametric tests require knowledge of the probability density

Table 9.1 Parametric triggering mechanisms for concept drift detection

Name	Test family	Change (A/D)	Entity under test	Type	Notes
Z-test	Statistical hypothesis test	Abrupt	Mean	1D	Assumes normality and known variance [89]
t-test	Statistical hypothesis test	Abrupt	Mean	1D	Assumes normality [89]
F-test	Statistical hypothesis test	Abrupt	Variance	1D	Assume normality [89]
Hotellings T-square statistic	Statistical hypothesis test	Abrupt	Mean	ND	Assumes normality [92]
SPRT	Sequential hypothesis Test	Abrupt	Pdf	1D	Minimizes the stopping time, nonparametric extensions are available [88]
CUSUM	Sequential change-point detection	Abrupt	Pdf	ND	Minimizes the worst detection latency [87]
Parametric CPM	Sequential change-point detection	Abrupt	Depends on the statistics used	1D/ND	Sequential version of a change-point method [93]

function and/or prior information about the concept drift, whereas nonparametric tests are more flexible and require little—mostly reasonable from the application point of view—hypotheses.

In detail, the tables present the family to which the method belongs, either a statistical hypothesis test designed on a given data set or sequential and, as such, suitable to address data stream-based applications. The "change" column shows which type of concept drift the method has been designed for while the "entity under test" column presents the key features used by the test to operate. "Type" refers to the nature of numerical data the test can receive, which is either scalar (univariate test, 1D) or multidimensional (multivariate test, ND). Finally, key references to the test as well as comments are given in the last column to complete the overview.

9.2 Change Point Methods

Change point methods inspect a given data sequence to check its stationary, i.e., whether the samples composing the sequence are independent realizations of a unique random variable or not. The problem is solved by checking if the sequence contains a change point, i.e., a specific time location beyond which the data distribution has changed.

Table 9.2 Non parametric triggering mechanisms for concept drift detection

Name	Test family	Change (A/D)	Entity under test	Type	Notes
Mann-Whitney U test	Statistical Hypothesis test	Abrupt	Median	1D	Rank Test Error based [186]
Kolmogorov-Smirnov test	Statistical Hypothesis test	Abrupt	Pdf	1D	Also goodness of fit test [90]
Mann Whitney Wilcoxon test	Statistical Hypothesis test	Abrupt	Pdf	1D	Rank-based [186]
Kruskal-Wallis test	Statistical Hypothesis test	Abrupt	Median	1D	Mann-Whitney based [91]
Pearson's chi-squared test	Statistical Hypothesis test	Abrupt	Pdf	1D	Goodness of fit and test of independence [80]
Distribution-Free CUSUM	Sequential change-point detection	Abrupt	Median	1D	Nonparametric extension of the CUSUM test [86]
Mann Kendall	Sequential change-point detection	Abrupt	Mean	1D	Designed to analyze climate change [79]
Multi-chart detection algorithm	Sequential change-point detection	Abrupt	Median	1D / ND	Detection of intrusion systems [85]
CI-CUSUM	Sequential change-point detection	Abrupt, Drift	PDF, sample moments	1D/ND	Computational intelligence based [84]
ICI change detection test	Sequential change-point detection	Abrupt, Drift	Mean and variance	1D	Exploits the Intersection of Confidence Interval (ICI) rule [83, 94]
Hierarchical change detection test	Sequential change-point detection	Abrupt, Drift	Mean and variance	1D	Based on a hierarchy of change detection tests [82]
Shiryaev-Robert Extension	Sequential change-point detection	Abrupt	Median	1D	Nonparametric extension of the Shiryaev-Robert test [81]
Mood	Statistical Hypothesis test	Abrupt	Dispersion	1D	Based on ranks [93]
Lepage	Statistical Hypothesis test	Abrupt	Location and dispersion	1D	sum of Mann-Whitney and Mood statistic [93]
Nonparametric CPM	Sequential change-point detection	Abrupt	Depends on the statistic used	1D	Sequential version of a change-point method [93]

9.2.1 Change Points

We say that given data sequence

$$\mathscr{X} = \{x(t), t = 1, \ldots, n\},$$

contains a change point at time/sample $\tau < n$ if subsequences

$$\begin{aligned} \mathscr{A}_\tau &= \{x(t),\ t = 1, \ldots, \tau\}, \\ \mathscr{B}_\tau &= \{x(t),\ t = \tau + 1, \ldots, n\}, \end{aligned} \tag{9.4}$$

are distinct i.i.d. realizations of two different unknown random variables distributed as $\mathscr{F}_0$ and $\mathscr{F}_1$. The problem detection can be rewritten as

$$\tau \text{ is a change point if } x(t) \sim \begin{cases} \mathscr{F}_0, & \text{for } t < \tau \\ \mathscr{F}_1, & \text{for } t \geq \tau \end{cases}, \tag{9.5}$$

The change point problem is then converted into an equivalent problem asking if $\mathscr{A}_\tau$ and $\mathscr{B}_\tau$ are sets generated from the same or different distributions.

9.2.2 Set Dissimilarity

A straightforward solution to determine whether a given τ is a change point or not consists in formulating a two-sample hypothesis test on the subsequences $\mathscr{A}_\tau$ and $\mathscr{B}_\tau$. In the hypothesis test, the null (H_0) and the alternative (H_1) hypotheses are composed as

$$H_0 : x(t) \sim \mathscr{F}_0 \ \forall t \tag{9.6}$$

$$H_1 : x(t) \sim \begin{cases} \mathscr{F}_0, & \text{if } t < \tau \\ \mathscr{F}_1, & \text{if } t \geq \tau \end{cases}. \tag{9.7}$$

To test the above hypothesis we need a statistic $\mathscr{T}$, assessing the dissimilarity between $\mathscr{A}_\tau$ and $\mathscr{B}_\tau$ defined in (9.4). Denote by $\mathscr{T}_\tau$ the value of such statistic $\mathscr{T}$

$$\mathscr{T}_\tau = \mathscr{T}(\mathscr{A}_\tau, \mathscr{B}_\tau), \tag{9.8}$$

in comparing $\mathscr{A}_\tau$ and $\mathscr{B}_\tau$. Following a standard hypothesis testing procedure, H_0 can be rejected when the value of $\mathscr{T}_\tau$ exceeds a suitable threshold $h_{n,\alpha}$, corresponding to a given confidence level α and depending on n. In this case, it is possible to claim that $\mathscr{A}_\tau$ and $\mathscr{B}_\tau$ are generated from different distributions (and $\mathscr{X}$ is, hence, not stationary), taking into account the percentage α of false positives.

Example: Evaluating the Dissimilarity of Two Sets

Consider, as an example, the case where data in $\mathscr{A}_\tau$ and $\mathscr{B}_\tau$ are Gaussian distributed with the same variance and we aim at investigating if they share the same expected value. We choose as test statistic D the standardized difference between the two sample means, which leads to a *two-sample t test*. The test statistic is

$$D_\tau = \sqrt{\frac{\tau(n-\tau)}{n}}\frac{\bar{\mathscr{A}_\tau} - \bar{\mathscr{B}_\tau}}{S_\tau} \tag{9.9}$$

where $\bar{\mathscr{A}_\tau}$ and $\bar{\mathscr{B}_\tau}$ denote the sample means evaluated on $\mathscr{A}_\tau$ and $\mathscr{B}_\tau$ respectively and S_τ is the pooled sample variance evaluated on $\mathscr{A}_\tau$ and $\mathscr{B}_\tau$. The threshold $h_{n,\alpha}$ for the statistic D is provided by the Student t distribution with $n-2$ degrees of freedom.

9.2.3 *The Change Point Formulation*

When the test statistic corresponding to a specific partitioning of $\mathscr{X}$ does not provide enough statistical evidence to reject H_0 we can only claim that the particular τ is not considered as a change point at the given confidence level, hence implying that no change in stationarity happened at sample τ. All other points composing the sequence need to be checked for being potential change points by considering all possible partitions of $\mathscr{X}$. The change point formulation provides a rigorous framework for testing the presence of a change point in a sequence $\mathscr{X}$. Within the CPM framework, e.g., [183], the null and alternative hypotheses for change point method are formulated as

$$H_0 : \forall t,\ x(t) \sim \mathscr{F}_0 \tag{9.10}$$

$$H_1 : \exists\, \tau\ x(t) \sim \begin{cases} \mathscr{F}_0, & \text{if } t < \tau \\ \mathscr{F}_1, & \text{if } t \geq \tau \end{cases}. \tag{9.11}$$

Each feasible time location in $\mathscr{X}$ has to be considered as a candidate change point. More in detail, for each candidate change point $s \in \{2, \ldots, n-1\}$,[1] the sequence $\mathscr{X}$ is partitioned into two nonoverlapping sets $\mathscr{A}_s = \{x(t),\ t = 1, \ldots, s\}$

[1] The actual range of s depends on the minimum number of samples needed to compute $\mathscr{T}$ from $\mathscr{A}_s$ and $\mathscr{B}_s$.

and $\mathscr{B}_s = \{x(t),\ t = s+1, \dots, n\}$. Set dissimilarity is measured as recommended in Sect. 9.2.2 by means of a suitable test statistic $\mathscr{T}$, which is evaluated for each change point candidate, yielding $\{\mathscr{T}_s, s = 2, \dots, n-1\}$. The most likely change point for sequence $\mathscr{X}$ is finally the one maximizing the statistic

$$M = \underset{s=2,\dots,n-1}{\operatorname{argmax}} \left(\mathscr{T}_s\right). \tag{9.12}$$

corresponding to the value $\mathscr{T}_M$ of $\mathscr{T}$

$$\mathscr{T}_M = \max_{s=2,\dots,n-1} \left(\mathscr{T}_s\right). \tag{9.13}$$

To finalize the test, $\mathscr{T}_M$ has to be compared with a predefined threshold $h_{n,\alpha}$, which guarantees a controlled rate α of false positives. Besides α, the threshold depends on the statistic $\mathscr{T}$ and the cardinality n of $\mathscr{X}$.

When $\mathscr{T}_M$ exceeds $h_{n,\alpha}$, the CPM rejects the null hypothesis, and $\mathscr{X}$ is claimed to contain a change point at M, the location maximizing (9.13). In these circumstances, besides claiming that $\mathscr{X}$ is not stationary, the CPM also provides M, an estimate of the change point instant τ. Conversely, when $\mathscr{T}_M < h_{n,\alpha}$, there is not enough statistical evidence to reject the null hypothesis, and no change point is identified within $\mathscr{X}$. The above can be formalized in the final outcome of the CPM test

$$\begin{cases} \text{The estimated change point in } \mathscr{X} \text{ is } M & \text{if } \mathscr{T}_M \geq h_{n,\alpha} \\ \text{No change point identified in } \mathscr{X}, & \text{if } \mathscr{T}_M < h_{n,\alpha} \end{cases}. \tag{9.14}$$

It is important to comment that, often, the major issue for a CPM is the definition of the thresholds $\{h_{n,\alpha}\}$. In fact, even when the distribution of the statistic $\mathscr{T}$ is known for any partitioning of $\mathscr{X}$, the distribution of its maximum $\mathscr{T}_M$ is hard to be computed. Asymptotic results are available for some test statistics which, however, are often inaccurate at low sample size. We comment that also when it is possible to provide an approximation for the distribution of the maximum, the outcome might not be appropriate. For instance, as discussed in [183], the Bonferroni approximation tends to be over-conservative as n grows. For these reasons, thresholds are often computed with the Monte Carlo method or, even better, with randomized algorithms.

Example: The CPM

Figure 9.5 illustrates the operating principle of a CPM relying on the Student t statistic D (9.9). Figure 9.5a presents a sequence $\mathscr{X}$ composed of 500 data. A change point is injected at $\tau = 350$, so that

$$x(t) \sim \begin{cases} \mathscr{N}(0,1), & \text{if } t < 350 \\ \mathscr{N}(-1,1), & \text{if } t \geq 350 \end{cases}. \tag{9.15}$$

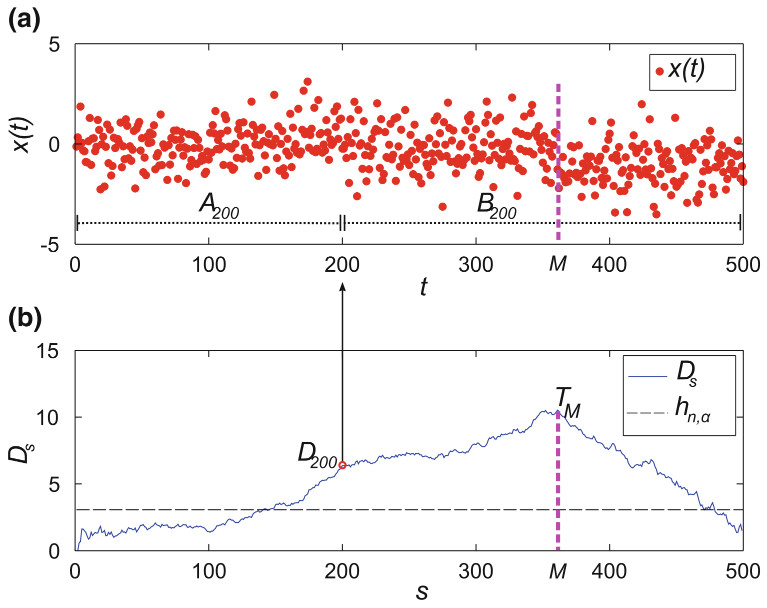

Fig. 9.5 An example of a CPM based on a Student t statistic. Data in (**a**) are distributed according to (9.15). The values assumed by the corresponding test statistic $\{D_s, s = 2, \dots, 499\}$ are reported in (**b**). The estimated change point M, and the corresponding value of the test statistic $\mathscr{T}_M$ are also reported. For illustrative purpose the figure also shows the partition of $\mathscr{X}$ in $\mathscr{A}_s$ and $\mathscr{B}_s$ when $s = 200$ together with the corresponding value of the statistic $D_{s=200}$

Figure 9.5b illustrates the values of the statistics D_s as a function of $s = 2, \dots, 499$.

The threshold corresponding to $\alpha = 0.05$, i.e., $h_{500,0.05} = 3.225$ was provided by the `CPM` package [184], implemented in the statistical `R` language. Other CPMs can be designed to detect shifts in the mean of a Gaussian random variable, e.g., [183].

9.2.4 Test Statistics Used in CPMs

Very often the test statistic $\mathscr{T}$ measures the dissimilarity between two sets by comparing the estimates for both expected value (sample mean) and variance (sample variance). This choice is motivated by the fact that, in practice, a change in the distribution as per (9.5) would also affect its first moments [185]. It is also preferable to employ nonparametric test statistic since, often, the distribution (even before the change point) is unknown.

Several nonparametric statistics are based on the rank computation, such as the Mann–Whitney [186] (to assess changes in the location), the Mood [187] (to assess changes in the scale), and the Lepage ones [188] (to assess both changes affecting the location and the scale). A CPM based on the Mann–Whitney statistic was introduced in [189] together with a CPM for Bernoulli random variables.

A different approach consists in locating change points by comparing the empirical distributions over two sets of data, as in the CPMs [190] that are based on the Kolmogorov–Smirnov and the Cramer Von Mises [191] statistics. So far, we mentioned only test statistics for scalars, however, the change point formulation can be used to analyze multivariate data, such as the CPM in [192], which relies on the Hotelling T^2 statistic.

9.2.5 Extensions Over the Basic Scheme

The change point formulation was originally presented as an offline processing tool. However, the methods have recently gained a lot of attention and CPM solutions for online and data streams have been provided. Such extensions basically consist in iterating the CPM at each new sample arrival [205]. It comes out that the computational complexity of such CPMs would endlessly grow, hence pushing the research toward the proposal of variants keeping in mind the computational complexity and the memory requirement of the methods [185,190]. In particular, a streaming adaptation is required when the test statistic $\mathscr{T}$ is computationally demanding (such as in test statistics based on the rank computation).

Another relevant issue is how to set the thresholds for online CPMs. First of all, it does not make any sense to control the probability of a false positive as in a hypothesis test. In fact, the test has to be iterated as a new sample arrives and the control of false positives has to be intended within a sequential scenario. Therefore, the thresholds have to be set to guarantee a fixed Average Run Length (ARL) of the test [87], namely the expected number of samples before the test yields a false positive during the operational life. Second, the probability that $\mathscr{T}_M$ exceeds $h_{n,\alpha}$ at the n-th sample has to be here conditioned to the fact that $\mathscr{T}$ never exceeded the threshold in previous $n-1$ samples. For these reasons, thresholds $\{h_{n,\alpha}, n > 0\}$ have to be computed numerically, through simulations, as in [183]. The `CPM` package [184] implements several CPMs based on different test statistics and provides also the thresholds $\{h_{n,\alpha}\}$ for both offline (traditional) CPMs and their online versions. Such thresholds could be loaded in a LUT as we move the CPM to an embedded system.

Any CPM (9.10) requires that data are either i.i.d. in the whole sequence (stationarity) or before and after the change point (nonstationarity). This may seem a restrictive assumption, since in real applications data are often characterized by different forms of dependence. When this happens, and we wish to operate with a CPM, it is necessary to move in a feature space where the i.i.d. assumption is met. A possibility is to operate in the model space by designing suitable models, e.g., of regression or predictive type, to fit the observations, and then analyze the residuals, see [193]. However, the residuals may not be i.i.d. since, quite often, the obtained model has a model bias component: in this case it can be convenient to aggregate several CPMs in an ensemble, as described in [204].

In what follows Change Detection Tests (CDTs) will be presented as statistical techniques designed having in mind online and sequential monitoring.

9.3 Change Detection Tests

There exists a large literature for concept drift detection mostly based on statistical hypothesis tests which, generally, require knowledge of the probability density function of the process generating the data and/or priors about the structure of the concept drift, e.g., a fault or a change in the environment. Again, the reference is that of Tables 9.1 and 9.2. In the parametric class of CDTs we find classic textbook tests such as the Student t-test and the Fisher f-test, addressing changes affecting the mean and the variance of the extracted features, respectively.

Nonparametric tests are more flexible tools, which require weaker assumptions, mostly tolerable at the application level. For instance, the Mann–Whitney U-test and the Wilcoxon test are nonparametric tests designed to detect a single change point and cannot support a sequential use, as sensing datastreams require. Differently, MannKendall and CUSUM are widely used tests adequate for a sequential analysis as the recently introduced ICI-based and hierarchical tests. In the section we present and detail three CDTs representing effective sequential nonparametric solutions to be implemented in embedded systems.

9.3.1 The CUSUM CDT Family

Complex and effective nonparametric tests generally require a configuration phase to fix test parameters at design time. The traditional CUmulative SUM control chart (CUSUM) is a sequential analysis technique designed for change detection that guarantees an appreciable change detection accuracy when a priori information about concept drift and the process generating the data are available. We present in the sequel two CDT methods that extend the traditional CUSUM by relaxing some of its restrictive assumptions. The first test extends the CUSUM by allowing the designer to automatically identify the configuration of the test parameters (adaptive CUSUM). The change detection ability of the adaptive CUSUM is based on the analysis of the evolution over time of the mean and the variance of some features extracted from the data-generating process. The second test, named Computational Intelligence CUSUM (CI-CUSUM), extends the first one by considering a richer set of features to improve efficiency in detecting changes in stationarity.

9.3.1.1 The Adaptive CUSUM CDT

Let $X = \{x(t), t = 1, \ldots, N\}, x(t) \in \mathbb{R}$ be a sequence of instances coming from the data generating process ruled by probability density function $f_\theta(x)$, which we assume to be unknown and parameterized in the parameter vector $\theta \in \mathbb{R}^n$.

Assume that the stochastic process changes its statistical behavior at unknown time T^o. This is generally modeled by considering a transition from parameter vector θ_0 to θ_1, associated with the pdfs $f_{\theta_0}(x)$ and $f_{\theta_1}(x)$, respectively. As with CUSUM, we evaluate the discrepancy between the two pdfs at time t, by computing the log-likelihood ratio

$$s(t) = \ln \frac{f_{\theta_1}(x(t))}{f_{\theta_0}(x(t))} \text{ for each } t = 1, \ldots, N$$

and the cumulative sum

$$S(t) = \sum_{\tau=1}^{t} s(\tau).$$

CUSUM identifies a change in X at time $\hat{T}$ when $g(t) = S(t) - m(t)$, the difference between the value of the cumulative sum $S(t)$ and its current minimum value $m(t) = \min_{\tau=1,\ldots,t} S(\tau)$ exceeds a given threshold h, namely

$$\hat{T} \text{ is the earliest time when } g(t) \geq h$$

CUSUM assumes that key parameters θ_0, θ_1 and h are available at design time. The assumption is generally hard to be satisfied but parameters can be estimated with the following procedure. Generate at first the cumulative sequence $Y = \{y(1), y(2), \ldots, \}$, where each s-th instance $y(s)$ represents the value of the sample mean estimated over a sliding nonoverlapping window of width n taken from X

$$y(s) = \frac{1}{n} \sum_{t=s(n-1)+1}^{sn} x(t)$$

From the central limit theorem the distribution of Y can be approximated with a Gaussian distribution provided that n is large enough. The basic CUSUM can then be applied to sequence Y. The first K configuration instances of X constitute the configuration set that is used to generate the training set of Y, whose cardinality is K/n (K is conveniently selected among the multiples of n). The whole procedure is depicted in Fig. 9.6. The parameters θ_0 characterizing the Gaussian distribution are the mean and variance of Y, i.e., $\theta = [\mu, \sigma^2]$, estimated on the train set. The parameters θ_1 are obtained through the identification of a neighborhood confidence for θ_0.

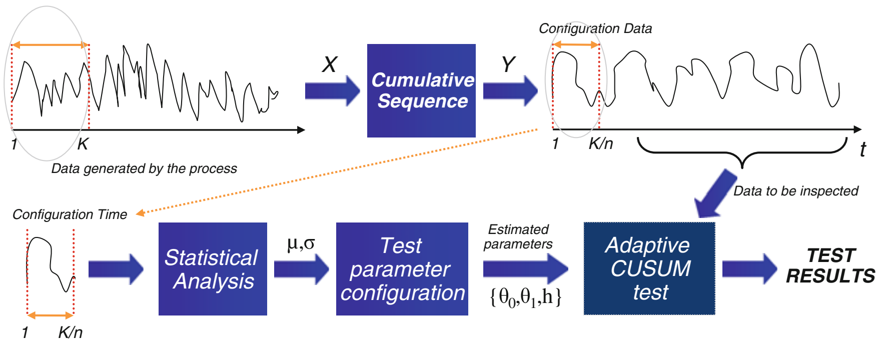

Fig. 9.6 The operational procedure for the adaptive CUSUM test. Data stream X undergoes a sequential windowing as data come in. When n samples are available, a data window is completed and ready to be averaged to generate the transformed instance $y(s)$. The distribution of $y(s)$ is approximately Gaussian, thanks to the central limit theorem, provided n is large enough. The basic CUSUM test can be applied with parameters $\theta = [\mu, \sigma^2]$. Needed parameters θ_0, θ_1 and threshold h are estimated on the training set

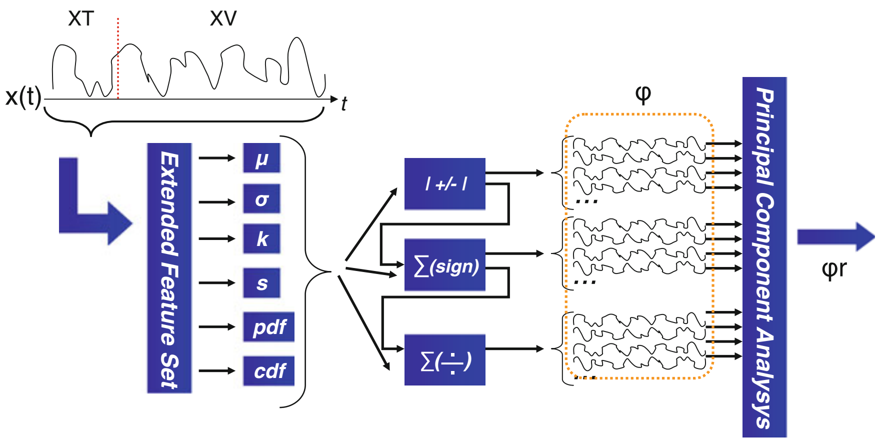

Fig. 9.7 The feature extraction and reduction phases of the CI-CUSUM. A rich set of features is extracted from the input signal to compose the feature set φ. Features extracted from the operational set XV are contrasted with those evaluated on the training configuration set XT. A PCA technique yields the reduced feature vector φ_r

9.3.1.2 The CI-CUSUM CDT

The CI-CUSUM represents an interesting extension of the adaptive CUSUM and turns to be much more powerful than the basic CUSUM and the Adaptive CUSUM since any feature can be extracted from the data stream to take advantage of different sensitivities in concept drift detection. The reference figure is Fig. 9.7.

Features φ are selected to be sensitive to concept drift. In particular, since data extracted from the training set XT are assumed to be i.i.d., the considered features are evaluated in a differential way to amplify the discrepancy between the current feature and the reference one associated with the training stationary state.

Envisaged features contain some well-known moments such as the mean μ, the variance σ^2 (to assess changes in the mean and variance of the distribution), the kurtosis $kurt$ and skewness $skew$ indexes (measuring how the distribution is peaked or flat and the lack of symmetry of a distribution, respectively), as well as information derived from the pdf and cumulative density function (cdf) of the signal. The running index is then contrasted with the corresponding one evaluated on the training set and features aim at amplifying the discrepancy between the two. For instance, feature $\varphi_1(t) = |\mu_0 - \mu_V|$ aims at amplifying discrepancies in the mean value. μ_0 is the value of the mean evaluated on the training set XT and subscript V refers to the test set, i.e., the index must be evaluated on data up to the running one (training data excluded). The basic features are

$$\varphi_1(t) = |\mu_0 - \mu_V|, \varphi_2(t) = |\sigma_0 - \sigma_V|, \varphi_3(t) = |kurt_0 - kurt_V|, \varphi_4(t) = |skew_0 - skew_V|$$

$$\varphi_5(t) = \int_x |pdf_0(x) - pdf_V(x)|dx, \varphi_6(t) = \int_x |cdf_0(x) - cdf_V(x)|dx$$

$$\varphi_{7\le j\le 12}(t) = \left\{\sum_{v=1}^{t-1} sgn\left(\varphi_{j-6,v+1} - \varphi_{j-6,v}\right)\}, \varphi_{13\le j\le 24}(t) = \{\sum_{v=1}^{t-1}\left(\frac{\varphi_{j-12,v+1}}{\varphi_{j-12,v}}\right)\right\}.$$

In particular, features $\varphi_5(t)$ and $\varphi_6(t)$ evaluate the discrepancy between the running pdf and cdf and that induced by the training set, respectively.

Features $\varphi_7(t)$ to $\varphi_{12}(t)$ investigate changes in the sequence of signs in consecutive elements and $\varphi_{13,t}$ to $\varphi_{24,t}$ the cumulative sum of the ratio of consecutive elements. To reduce the complexity of the feature space we performed a PCA on φ which provides a transformed feature φ_r. Since the pdf of φ_r is not a priori available, we operate as in the adaptive CUSUM case. In detail, we take the average of φ_r over nonoverlapping windows and invoke the central limit theorem which provides an approximated multivariate Gaussian distribution for the transformed variable φ' characterized by mean M and covariance matrix C. The mean M_0 and covariance matrix C_0 of φ_r are estimated on the training set and provide the nominal reference configuration $\theta_0 = [M_0, C_0]$. The adaptive CUSUM procedure is invoked that computes the alternative hypothesis for the change detection test $\theta_1 = [M_1, C_1]$. The CI-CUSUM is now configured and assesses over time $\varphi'(t)$ by checking whether it belongs to distribution $\mathcal{N}(M_0, C_0)$ or not by measuring the discrepancy between the two multivariate probability density functions at time t through the log-likelihood ratio mechanism

$$s(t) = \ln \frac{\mathcal{N}_{M_0,C_0}(\varphi'(l))}{\mathcal{N}_{M_1,C_1}(\varphi'(l))} \text{ for each } l = 1, \cdots, t.$$

The adaptive CUSUM test can now be applied and either returns detection of concept drift or claims that concept drift is not present.

9.3.2 The Intersection of Confidence Intervals CDT Family

The Intersection of Confidence Intervals (ICI) CDT and its evolutions [94] detect concept drift affecting a data stream by monitoring the evolution of suitable features extracted from incoming data. Features must be i.i.d. and Gaussian distributed, at least before concept drift occurs. The assumptions might appear strong and far from any engineering reality, in particular the i.i.d. one. However, this is not the case in many real applications provided that suitable transformations are invoked.

For instance, the method can be used to inspect sequences of residuals, e.g., associated with the discrepancy between a predictive model describing the data stream and the real data as they are acquired. When the test detects a change in the residual then concept drift is detected. This issue will be further addressed in the sequel. We now present the principal features of the ICI-CDT family.

9.3.2.1 The ICI-CDT

In the ICI-CDT, features are extracted by windowing the available data in disjoint subsequences composed of n instances. For each subsequence we compute the sample mean and the sample variance which are Gaussian distributed thanks to the central limit theorem for the former and and ad hoc transformation [95] the latter. More in detail, named s the s-th subsequence, the extracted features are

$$M(s) = \frac{\sum\limits_{t=(s-1)n+1}^{ns} x(t)}{n}, \text{ and } V(s) = \left(\frac{\left(\sum\limits_{t=(s-1)n+1}^{ns} (x(t) - M(s))^2 \right)}{n-1} \right)^{h_0}, \tag{9.16}$$

The parameter h_0 is the exponent of the power-law transform devised in [95] to generate an approximated Gaussian distribution for the sample variance. h_0 is estimated from the sample cumulants computed on training data O_{T_0}.

The ICI-CDT is configured on the two sequences of features $\{M(s), s = 1, \ldots, S_0\}$ and $\{V(s), s = 1, \ldots, S_0\}$, being $S_0 = T_0/n$ extracted from O_{T_0}, We compute the means $\hat{\mu}^M_{S_0}$, $\hat{\mu}^V_{S_0}$ and the standard deviations $\hat{\sigma}^M_{S_0}$, $\hat{\sigma}^V_{S_0}$ of the two features over the training set, i.e.,

$$\hat{\mu}^M_{S_0} = \frac{\sum_{s=1}^{S_0} M(s)}{S_0}, \quad \text{and} \quad \hat{\sigma}^M_{S_0} = \sqrt{\frac{\sum_{s=1}^{S_0} (M(s) - \hat{\mu}^M_{S_0})^2}{S_0 - 1}}. \tag{9.17}$$

and

$$\hat{\mu}^V_{S_0} = \frac{\sum_{s=1}^{S_0} V(s)}{S_0}, \quad \text{and} \quad \hat{\sigma}^V_{S_0} = \sqrt{\frac{\sum_{s=1}^{S_0} (V(s) - \hat{\mu}^V_{S_0})^2}{S_0 - 1}}. \tag{9.18}$$

These estimates define the confidence intervals for the mean and standard deviation features that, under the stationary condition, are defined as

$$\begin{aligned} \mathscr{I}^M_{S_0} &= [\hat{\mu}^M_{S_0} - \Gamma\hat{\sigma}^M_{S_0}, \hat{\mu}^M_{S_0} + \Gamma\hat{\sigma}^M_{S_0}], \\ \mathscr{I}^V_{S_0} &= [\hat{\mu}^V_{S_0} - \Gamma\hat{\sigma}^V_{S_0}, \hat{\mu}^V_{S_0} + \Gamma\hat{\sigma}^V_{S_0}], \end{aligned} \tag{9.19}$$

with $\Gamma > 0$ controlling the amplitude of the confidence interval and, then, the probability that features belong to the interval under the stationary assumption.

Once training is perfected the CDT becomes operational and can be used to assess changes in stationarity in the data stream. Every time n data are made available, a new sequence s is created and features extracted to populate the $\mathscr{I}^M_s$ and $\mathscr{I}^V_s$.

The intersection of confidence intervals rule (ICI-rule) [96] can then be applied. The ICI-rule verifies whether the new feature instance can be intended as a realization of the existing Gaussian distribution. If not, concept drift is detected in the data stream.

From the operational point of view, the sample mean of all the feature values is computed, together with the confidence interval of the corresponding estimator which is expressed as (9.19). As soon as the intersection of all the confidence intervals up to the current one results in an empty set, the basic ICI-CDT detects a change. Thus, we detect a concept drift in the subsequence $\hat{s}$ if

$$\begin{gathered} \bigcap_{s<\hat{s}} \mathscr{I}^M_s \neq \emptyset \text{ and } \bigcap_{s\leq\hat{s}} \mathscr{I}^M_s = \emptyset \text{ or} \\ \bigcap_{s<\hat{s}} \mathscr{I}^V_s \neq \emptyset \text{ and } \bigcap_{s\leq\hat{s}} \mathscr{I}^V_s = \emptyset \end{gathered} \tag{9.20}$$

and the detection time $\hat{T} = n\hat{s}$ corresponds to the rightmost term of the subsequence $\hat{s}$.

Concept drift is associated with those feature(s) that yielded the empty intersection. Figure 9.8 illustrates how the ICI-rule operates. To reduce the computational

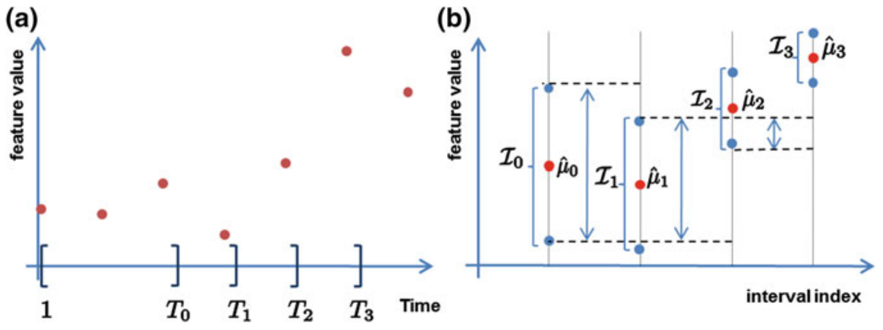

Fig. 9.8 An illustrative example of the ICI rule in the setting used for change detection: **a** feature values and the set of intervals $\{[1, T_0], [1, T_1], [1, T_2], [1, T_3]\}$, **b** the corresponding polynomial zeroth-order estimates and their confidence intervals. The ICI rule selects the interval $[1, T_2]$, since $\mathcal{I}_0 \cap \cdots \cap \mathcal{I}_2 \neq \emptyset$ and $\mathcal{I}_0 \cap \cdots \cap \mathcal{I}_3 = \emptyset$. The brackets in (**b**) represent the confidence intervals; the arrows their intersections

load, the average feature and the intersection of confidence intervals are computed incrementally, and each feature is separately processed.

The whole procedure is summarized in Algorithm 20. As pointed out in [94] the basic ICI-CDT is particularly effective but introduces a structural limitation inducing a structural false positive when time passes.

Despite the fact that this problem can be tolerated in many detect and react mechanisms, it is important to design a test that does not introduce structural false positives as time passes. This problem can be solved by considering a second test, built on the top of the basic ICI-CDT that verifies, once activated, if a false positive has been generated by the first CDT or the raised alarm should be considered a proper concept drift. For its layered structure the test is named Hierarchical CDT.

9.3.2.2 The H-CDT

The Hierarchical CDT (H-CDT) has been designed to mitigate the structural problem posed by the ICI-CDT, for which false positives are generated as time passes. The H-CDT is a hierarchical sequential change detection test structured into two processing levels. The first level is composed of the ICI-CDT test and the second is a statistical test validating/rejecting the change hypothesis. The ICI-CDT operates sequentially as presented in the previous subsection and, when it detects a change in the sequence $x(t)$ at time $\hat{T}$, it activates the upper test to validate the detection by checking if the data sets before and after the estimated $\hat{T}$ are consistent with the change hypothesis.

For such change validation purposes, we need to acquire a set of N additional data $O_{\hat{T}} = \{x(t), t = \hat{T}, \ldots, \hat{T} + N\}$ generated after $\hat{T}$, which are considered to have been potentially generated from the new state of the data-generating process, namely, after concept drift. The adjective "potentially" is appropriate since a false

Algorithm 20: The basic ICI-CDT

```
1  Compute $\{M(s), s = 1, \dots, S_0\}$, being $S_0 = T_0/n$;
2  $\hat{\mu}^M_{S_0} = \sum_{s=1}^{S_0} \frac{M(s)}{S_0}$;
3  $\hat{\sigma}^M = \sqrt{\sum_{s=1}^{S_0} \frac{\left(M(s)-\hat{\mu}^M_{S_0}\right)^2}{S_0-1}}$, $\hat{\sigma}^M_{S_0} = \frac{\hat{\sigma}^M}{\sqrt{S_0}}$;
4  Define $\mathscr{I}^M_{S_0} = \left[\hat{\mu}^M_{S_0} - \Gamma\hat{\sigma}^M_{S_0}, \hat{\mu}^M_{S_0} + \Gamma\hat{\sigma}^M_{S_0}\right]$;
5  Compute $h_0$;
6  Compute $\{V(s), s = 1, \dots, S_0\}$;
7  $\hat{\mu}^V_{S_0} = \sum_{s=1}^{S_0} \frac{V(s)}{S_0}$;
8  $\hat{\sigma}^V = \sqrt{\sum_{s=1}^{S_0} \frac{\left(V(s)-\hat{\mu}^V_{S_0}\right)^2}{S_0-1}}$, $\hat{\sigma}^V_{S_0} = \frac{\hat{\sigma}^V}{\sqrt{S_0}}$;
9  Define $\mathscr{I}^V_{S_0} = \left[\hat{\mu}^V_{S_0} - \Gamma\hat{\sigma}^V_{S_0}; \hat{\mu}^V_{S_0} + \Gamma\hat{\sigma}^V_{S_0}\right]$;
10 Set $s = S_0$;
11 while ($\mathscr{I}^M_s \neq \emptyset$ and $\mathscr{I}^V_s \neq \emptyset$) do
12     Set $s = s + 1$;
13     Wait for $n$ observations, until a new subsequence is populated;
14     Compute $M(s)$ and $V(s)$ from observations in the subsequence according to 9.16;
15     Compute $\hat{\mu}^M_s = \frac{(s-1)\hat{\mu}^M_{s-1}+M(s)}{s}$ and $\hat{\sigma}^M_s = \frac{\hat{\sigma}^M}{\sqrt{s}}$;
16     Compute $\hat{\mu}^V_s = \frac{(s-1)\hat{\mu}^V_{s-1}+V(s)}{s}$ and $\hat{\sigma}^V_s = \frac{\hat{\sigma}^M}{\sqrt{s}}$;
17     $\mathscr{I}^M_s = \left[\hat{\mu}^M_s - \Gamma\hat{\sigma}^M_s; \hat{\mu}^M_s + \Gamma\hat{\sigma}^M_s\right] \cap \mathscr{I}^M_{s-1}$;
18     $\mathscr{I}^V_s = \left[\hat{\mu}^V_s - \Gamma\hat{\sigma}^V_s; \hat{\mu}^V_s + \Gamma\hat{\sigma}^V_s\right] \cap \mathscr{I}^V_{s-1}$;
   end
19 Concept drift detected at $s = \hat{s}$ within time interval $\left[(\hat{s} - 1)n + 1, \hat{s}n\right]$,i.e., $\hat{T} = n\hat{s}$
```

positive might arise and, hence the information present in the $O_{\hat{T}}$ set is compatible with that provided by the training set O_{T_0} onto which the method was configured (T_0 refers to the last time instant associated with the training set).

We comment that, if estimate $\hat{T}$ is highly accurate, then we might expect to improve the accuracy of the method by considering the whole set $\{x(t), t < \hat{T}\}$ instead of O_{T_0}. The reason behind the last statement is associated with the fact that if $\hat{T}$ is a good estimate for the concept drift time then, the presence of data associated with new state in $\{x(t), t < \hat{T}\}$ is likely to be negligible. In what follows, we rather prefer to be conservative and consider O_{T_0} instead: this choice goes also in the direction of operating with a reduced computational load, which is a relevant issue for embedded systems.

Statistical affinity between sets O_{T_0} and $O_{\hat{T}}$ should be evaluated with a proper statistical test, e.g., Kolmogorov–Smirnov test or other hypothesis tests such as those given in Table 9.2. However, the required computational load of the Kolmogorov–Smirnov is too high and unfeasible for a large class of embedded systems where high MIPS cannot be provided. The problem of comparingthe data distribution over O_{T_0}

and $O_{\hat{T}}$, can be conveniently simplified to the problem of comparing the expected values of the features (9.16) of the ICI-CDT over O_{T_0} and $O_{\hat{T}}$ by means of an Hotelling's test.

In particular, the Hotelling's test is a multivariate hypothesis test, which we apply to compare the values of the features (9.16) arranged in two-dimensional vectors $F = [M(s), V(s)]$. These feature vectors are extracted from the time interval O_{T_0} (onto which the ICI-CDT was configured) and on the interval $O_{\hat{T}}$ (which are expected to describe the new state of the process). From each of these sets, the sample means $\overline{F}(O_{T_0})$ and $\overline{F}(O_{\hat{T}})$ and the pooled sample covariance matrix are computed. The null hypothesis H_0 is formulated as

$$H_0 : \overline{F}(O_{T_0}) - \overline{F}(O_{\hat{T}}) = \underline{0} \tag{9.21}$$

where $\underline{0}$ represents the two-dimensional vector of null components. Finally, the Hotelling T^2 test [241] can be executed to reject the null hypothesis at a predefined confidence level α. If the Hotelling test rejects the equivalence hypothesis then a change is considered to be present in the feature set and the change hypothesis raised by the ICI-CDT is validated. In turn, the hierarchical test detects concept drift.

Conversely, if there is not enough statistical evidence to reject the null hypothesis, then the ICI-CDT introduced a false positive and must be retrained on the original stationary state O_{T_0}. The Hotelling's test applied to the the features (9.16) shows to be a particularly effective solution to assess changes detected by the ICI-CDT.

The H-CDT is therefore an adaptive test which reacts when false positives are introduced by the ICI-CDT and shows to be a great test to be used in embedded systems. The H-CDT algorithm describing from a high-level perspective is given in Algorithm 21.

Interestingly, if we want the hierarchical test to operate in a sequential manner, after each change validation, we are able to retrain the ICI-ICT and we also update the reference set O_{T_0} used in the Hotelling test. In fact, if the change has been validated at time $\hat{T}$, the set $O_{\hat{T}}$ contains instances associated with the new state of the data-generating process, thus the inspection for concept drift proceeds. The algorithm is summarized in Algorithm 22.

9.3.2.3 An Improved Estimate for the Concept Drift Detection Time

The H-CDT described in the previous section has the main drawback of having to wait for N observations after $\hat{T}$ before proceeding with the change-validation phase. This is of course not appealing in an online monitoring scenario, also because the detection $\hat{T}$ is typically characterized by a structural delay (correct detections provided by most of CDTs come typically after the unknown change-time instant T^o). The idea is hence to improve the estimate of T^o once the change has been detected, to recover part of the samples between T^o and $\hat{T}$ for improving change-validation efficiency and possibly CDT reconfiguration. Therefore, the improved estimate of T^o, which we denote by $\overline{t}$, is expected to satisfy $T^o \leq \overline{t} \leq \hat{T}$. Once $\overline{t}$ has been

Algorithm 21: The hierarchical change detection test H-CDT. The test is initially configured on the training set O_{T_0}. Once concept drift is detected by the ICI-CDT, Hotelling test is activated. If the Hotelling test validates the concept-drift hypothesis then an alarm is raised by the H-CDT and concept drift is validated. When the ICI-CDT detection is not validated, a false positive is found, no concept drift alarm is raised, and the ICI-CDT needs to be reconfigured on the initial training data.

```
1 Train the ICI-CDT on $O_{T_0}$;
2 while (1) do
3     Extract features $M(s)$ and $V(s)$ out of the data stream;
4     if (ICI-CDT detects a change in the features) AND (Hotelling test validates the change)
      then
5         Conceptdrift= true;
6         retrain ICI-CDT onto $O_{T_0} = O_{\hat{T}}$;
      else
7         false positive: retrain ICI-CDT onto $O_{T_0}$
      end
  end
```

Algorithm 22: The H-CDT within the active learning modality. When concept drift is validated at time $\hat{T}$ the application is reconfigured and the H-CDT retrained on the new instances.

```
1 Train the ICI-CDT on $O_{T_0}$;
2 while (1) do
3     Extract features $M(s)$ and $V(s)$ from the data stream;
4     if (ICI-CDT detects a change in the features) AND (Hotelling test validates the change)
      then
5         Conceptdrift= true;
6         React on concept drift at the application level;
7         retrain ICI-CDT onto $O_{T_0} = O_{\hat{T}}$;
      else
8         false positive: retrain ICI-CDT onto $O_{T_0}$
      end
  end
```

computed, we could use the observations $\{x(t), t = \bar{t}, \ldots, \hat{T}\}$ to define $O_{\hat{T}}$ without delaying the validation procedure

$$O_{\hat{T}} = \{x(t), t = \bar{t}, \ldots, \hat{T}\}$$

that contains more than n samples, thus increasing the significance during validation and configuration w.r.t. the approach described in the previous section. However, when the ICI-CDT is very quick $O_{\hat{T}} = \{x(t), t = \bar{t}, \ldots, \hat{T}\}$ may not contain enough samples, and it would be preferable to wait for at least N sample before activating the change-validation and reconfiguration procedures.

Algorithm 23: The refinement procedure leading to the improved estimate for the change time instant $\bar{t}$.

```
1  Provide $\hat{T}$;
2  Compute $T_1 = T_0 + (\hat{T} - T_0)/\lambda$;
3  $i = 1$; continue = *true*;
4  **while** *(continue = true)* **do**
5      Apply the ICI-CDT to $[0, T_0] \cup [T_i, \hat{T}]$, detecting at $\hat{T}_i$;
6      Compute $T_{i+1} = T_i + (\hat{T} - T_i)/\lambda$;
7      Define $T_{\min} = \min\left(\hat{T}_j\right), j = 1, \ldots, i$;
8      **if** $(T_{min} < T_{i+1})$ **then**
9          continue = *false*;
       **end**
10     $i = i + 1$;
   **end**
11 Define $\bar{t} = T_{\min}$.
```

The key point of the proposed solution is that the ICI-CDT introduces a structural detection latency that increases as time passes [94]. This undesirable behavior can be exploited to design a post-detection procedure that, starting from $\hat{T}$ yields a better estimate $\bar{t}$ as illustrated in Algorithm 23.

The algorithm operates as follows. Given concept drift detection from the ICI-CDT at time instant $\hat{T}$ split the interval $[T_0, \hat{T}]$ in two intervals $[T_0, T_1]$ and $[T_1, \hat{T}]$, with $T_1 = T_0 + (\hat{T} - T_0)/\lambda$ defined according to the user-set parameter $\lambda > 1$ (line 2). Apply the ICI-CDT to dataset $[0, T_0] \cup [T_1, \hat{T}]$ (line 5), leading to a detection at time $\hat{T}_1$. Note that $\hat{T}_1$ is a more accurate estimate of the change time T_0, since the test operates on a shorter sequence w.r.t. the one which provided the initial detection $\hat{T}$. Interval $[T_1, \hat{T}]$ is further split into two intervals $[T_1, T_2]$ and $[T_2, \hat{T}]$ where $T_2 = T_1 + (\hat{T} - T_1)/\lambda$ (line 6). If $T_2 > \hat{T}_1$, the procedure stops, and $\bar{t} = \hat{T}_1$.

Otherwise, the procedure iterates: at the i-th iteration, the ICI-CDT is executed on $[0, T_0] \cup [T_i, \hat{T}]$, providing the estimate $\hat{T}$ (line 5). The interval $[T_i, \hat{T}]$ is then split by point $T_{i+1} = T_i + \frac{\hat{T} - T_i}{\lambda}$ (line 6). The procedure ends when T_{i+1} is larger than $T_{\min}$, the earliest detection identified during the iteration of the procedure (line 7). Finally, $T_{\min}$ is the best estimate of T^{o} obtainable according to this procedure, The improved final estimate is hence $\bar{t} = T_{\min}$.

The refinement procedure is visualized in Fig. 9.9.

Comments

The estimate $\bar{t}$, which is provided by all the ICI-CDTs, makes it particularly appealing for active (detect and react) learning frameworks, since they provide set $O_{\hat{T}}$ that contains instances associated with the new state of the process generating the data. These instances, in the data stream domain, can now be used to reconfigure the appli-

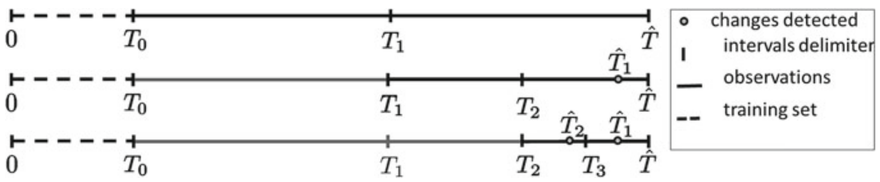

Fig. 9.9 The ICI-based time change estimate refinement procedure: an example with $\lambda = 2$. Initially, (*first line*) a change is detected by the ICI-CDT in correspondence with time $\hat{T}$ and the procedure starts by computing $T_1 = T_0 + \frac{\hat{T}-T_0}{\lambda}$. The ICI-CDT is then executed onto interval $[0, T_0] \cup [T_1, \hat{T}]$, resulting in a detection at $\hat{T}_1$ (*second line*). This procedure iterates by computing $T_2 = T_1 + \frac{\hat{T}-T_1}{\lambda}$ and the test is executed on interval $[0, T_0] \cup [T_2, \hat{T}]$. The procedure terminates when $T_3 > \hat{T}_2$, being $T_2 = \min\{\hat{T}_j\}$. The output is $\bar{t} = \hat{T}_2$, and $[\hat{T}_2, \hat{T}]$ is assumed to be generated by the process in the novel state, i.e., after concept drift

cation, besides the CDT itself. Moreover, the H-CDT shows to be computationally lighter than CI-CUSUM [101] and in most applications involving embedded systems should be preferred. For this reason the code of the hierarchical CDT has been made freely available and can be downloaded from the link given in [102].

We recall that insurgence of false positives introduces a processing load, since it leads to unnecessary reconfiguration, and that this might also reduce the performance of the application. In fact, if we mistakenly abandon the a priori rich training set O_{T_0} for the new one $O_{\hat{T}}$ following the false positive we should expect to end up with a consistent data set of lower cardinality.

At the same time, the presence of a false negative is also critical, since when no concept drift is detected, the adaptation mechanism is not activated.

As a last note we investigate the effects induced by slowly developing gradual concept drift. It is expected that this concept drift will not be detected in its early stages, but most probably later, when the influence of the concept drift on the features level grants detection. However, latency in detection is the cost we have to pay in correspondence to slowly developing gradual concept drift. Moreover, given the type of CDTs we are considering, a slowly developing concept drift results in a sequence of concept drift detections, a detection profile being symptomatic of a gradual concept drift evolution.

The literature has proposed CDTs specifically designed to manage situations with slowly developing concept drift under some assumptions about the evolution model for the concept drift, mostly following polynomial functions of a fixed order. The interested readers can refer to [97].

9.3.3 Amygdala—VM-PFC: The H-CDT

The H-CDT is a pure example of a cognitive mechanism. There, the lower processing level based on an ICI-CDT quickly processes the input stimuli like in the amygdala and provides a first reaction outcome. A threat (perceived change) is immediately detected (automatic process) and actions are promptly taken (e.g., we immediately react when we see a gun oriented toward us irrespective of other extra information). Afterwards, the emotional state is passed to the VM-PFC which perfects the taken action with a more articulated processing and comes back to the amygdala with a new control action (e.g., when we realize that the gun is indeed a fake water gun, held by a kid). The counterpart of VM-PFC corresponds to the higher level of the H-CDT, where the Hotelling test either validates or rejects the concept drift hypothesis raised by the lower level CDT. When the change hypothesis is rejected, the action invoked by the ICI-CDT is aborted, the state rolled back, and the ICI-CDT is reconfigured after the false positive detection. There is no evidence that the VM-PFC levels reconfigure the amygdala even though a negative feedback is likely to be provided.

9.4 The Just-in-Time Learning Framework

Availability of a CDT within a sequential framework allows us for designing applications characterized by an active learning modality. The chosen CDT detects concept drift (detection modality) and the application reacts accordingly (reaction modality) by adapting to the new state. This active learning modality is known in the literature as *Just in Time* (JIT) learning meaning that the application reconfiguration to track changes in the environment is activated exactly when needed, i.e., in correspondence with concept drift detection, in contrast with passive solutions where learning is always enabled.

We instance the JIT mechanism to an application to ease the presentation and to the classifier case for its relevance in applications. In JIT classifiers a CDT identifies concept drift affecting incoming data and the classifier-based application undergoes a reconfiguration phase to track the change in stationarity. A unique characteristic of JIT classifiers compared with other classifiers following the active learning modality is that, when no change is detected, the classifier continues integrating new supervised information made available to improve the classification accuracy.

A high level description of the JIT adaptive classification framework is given in Algorithm 24. The framework is very general and can host any type of CDT and classifier. Clearly, we should consider effective low complexity CDTs and classifiers having in mind, as final target, embedded systems.

The JIT framework is very general and can deal with any type of concept drift from *abrupt concept drift* to *gradual concept drift*. In the abrupt case we need to release obsolete data used for training the classifier and replace them with novel supervised instances characterizing the new operational condition, then the classifier is trained on the new training set, e.g., [94]. Differently, a frequent management activity involving both the update of the training set and retraining is needed when we encounter gradual concept drift, which are seen as a sequence of abrupt concept drifts following the detection mechanism. Extensions of the mechanism meant to deal with gradual concept drift has been suggested in [97].

Algorithm 24: The JIT adaptive classifier. New data instances are integrated in the classifier as they come until concept drift is detected. When the CDT detects a change the supervised instances associated with the $O_{\hat{T}}$ data set are used to reconfigure the classifier.

```
1  Configure the JIT classifier and the CDT;
2  while (true) do
3      input receive new data;
4      if (CDT detects concept drift) then
5          Characterize the new process state;
6          Configure the JIT classifier and the CDT on the new process state;
       else
7          integrate available extra information in the JIT;
       end
8      Classify the new input samples;
   end
```

In the sequel we focus at first on the core JIT for abrupt concept drift and address afterwards the gradual concept drift case. We use, as reference CDT the ICI-based family although any CDT can be adopted.

9.4.1 Observation Model

Consider, for sake of simplicity, a two-class classification problem. The operational framework can be formalized as follows.

Let $x \in X \subset \mathbb{R}^d$ be an i.i.d. random variable and $y \in \{\omega_1, \omega_2\}$ the associated binary classification output. The pdf of the inputs at time t

$$p(x|t) = p(\omega_1|t)p(x|\omega_1, t) + p(\omega_2|t)p(x|\omega_2, t), \tag{9.22}$$

depends on the pdfs of the outputs $p(\omega_1|t)$ and $p(\omega_2|t) = 1 - p(\omega_1|t)$ and the conditional probability distributions $p(x|\omega_1, t)$ and $p(x|\omega_2, t)$. In general, these distributions are unknown.

Let $O_T = \{x(t), t = 1, , T\}$ be the data sequence at time T and $Z_T = \{(x(t), y(t)),\ t \in I_T\}$ the knowledge base of the classifier at time T, which contains the supervised couples $(x(t), y(t))$, i.e., $y(t)$ is the classification label associated with the observation $x(t)$, and I_T is the set containing the arrival times of supervised samples up tp time T.

We further assume that the samples acquired before T_0 have been generated in stationary conditions. The set O_{T_0} is then used to train the CDT, while $Z_0 = \{(x(t), y(t)),\ t \in I_0\}$ represents the initial knowledge base (KB) of the classifier, being I_0 the set of supervised samples in O_{T_0}. Assume that at time instant $T^{\text{o}} > T_0$ a change in stationarity occurs with a subsequent change in the distribution of x: also the distribution after the change is unknown.

In the JIT framework the CDT inspects the process by operating on the data sequence O_T and, in some of its variations, by also exploiting supervised sample information.

9.4.2 The JIT Classifier

With reference to Algorithm 24, the JIT classifier undergoes an adaptation phase whenever the CDT detects concept drift. Otherwise, it integrates available new information in the training set to improve the classification accuracy over time.

9.4.2.1 React to the Change: Updating the Classifier

Retraining the adaptive JIT classifier requires learning the model of the data generating process after the change. Thus, the set of features $Z_{s|t>\bar{t}} = Z_{s|[\bar{t},\hat{T}]}$, i.e., the data observations in time interval $[\bar{t}, \hat{T}]$, represent the new state of the process generating the data following concept drift. Such instances must be used to retrain the classifier (in the time domain t) and the ICI-CDT (in the s domain).

Any consistent classifier, where consistency requires as necessary sufficient that the model family is a universal function approximator, can be considered in the JIT framework. However, if we have embedded systems in mind, then not all classifiers are equally valid. Computational complexity and memory usage must be taken into account when designing the application also for the indirect effect on power consumption in energy-aware applications. Feedforward neural networks, k-NN classifiers, Radial basis function neural networks are examples of consistent classifiers [100], SVM and regularized kernel classifiers are consistent classifiers depending on the particular choice of the loss function and the implementation algorithm [99]. However, the training phase is a costly operation for most classifiers, hence becoming most likely to be prohibitive for embedded systems, in particular if big data are involved. As shown in [101] k-NN classifier is a particularly appealing solution since its training phase is immediate and reduces to the insertion of supervised couples in a table representing the KB of the classifier.

We recall that the k-NN classifier provides a label to a new instance to be classified as the label majority of the k closest instances. The figure of merit evaluating the affinity between two instances—mostly based on an euclidean distance in the input space—the inspection of instances in the KB and score ranking to identify the k closest neighbors are the main computationally demanding parts of the algorithm. If the cardinality of the KB is N the k-NN classifier is consistent [103] provided that

$$\frac{k}{N} \to 0 \text{ as } k \to \infty, N \to \infty.$$

However, k-NN classifiers are memory eager solutions—this is the price we have to pay for a computational negligible training phase—since all N instances need to be stored in the memory, despite the fact that efficient solutions can be envisaged to keep under control the memory request, e.g., those based on condensing or editing techniques. Both condensing and editing techniques aim at reducing the cardinality of the training set yet preserving the maximum classification accuracy. In particular, condensing techniques, e.g., Condensed Nearest Neighbor (CNN) [105], aim at keeping in the training set only those samples fundamental to shape the decision boundary. Differently, editing techniques, e.g., the Wilson Editing Rule (WER) [106], intervene on the training set by removing particularly noisy samples and request the Bayes's decision boundary to be smooth.

A better solution would involve thresholding the cardinality of KB by keeping a maximum of N_M instances. If they are in stationary conditions and N increases, N will be buffered to N_M supervised couples, for instance by keeping the most recent N_M instances. A simple circular buffer would solve the problem.

We detail the JIT classifier based on a k-NN structure and the H-CDT; the reference is Algorithm 25.

The initial knowledge base of the k-NN classifier is $Z_0 = \{(x(t), y(t)),\ t \in I_0\}$ (line 1), while the value of k is set to k_{LOO}, estimated by means of the Leave-One-Out (LOO) technique applied to Z_0 (line 2). The H-CDT is configured on the initial training set O_{T_0} (line 3). After this configuration phase, the algorithm works online by classifying upcoming samples as they arrive and by introducing, whenever available (line 7), new supervised information $(x(t), y(t))$ into the knowledge base of the classifier KB. In this case, the algorithm stores in I_T the time instant t when the sample has been received (line 8), includes the pair $(x(t), y(t))$ in Z_T (line 9), and updates the parameter k so that consistency is granted. The reader should be aware that k cannot be freely chosen as N increase to satisfy the consistency conditions; an effective computational-aware method to estimate the appropriate k is given in [104] and relies on the LOO performance evaluation method.

In stationary conditions, the classification accuracy always increases by introducing additional supervised samples during the operational life [103] but when available $x(t)$ is not supervised, I_t and Z_t sets are not updated (lines 11-12).

When the H-CDT notifies concept drift in the subsequence containing $x(t)$ (line 13), the refinement procedure also provides $\bar{t}$ (line 15). The H-CDT is then reconfigured on features s associated with the new state of the process, i.e., those in time interval $[\bar{t}, \hat{T}]$ (line 16).

The $\bar{t}$ information allows the JIT for removing those training samples acquired before $\bar{t}$ both from I_t and Z_t (lines 17, 18). The new value of k_{LOO} is then estimated by means of the LOO procedure on the new knowledge-base (line 19) and k set to it. Finally, $x(t)$ is classified by relying on the updated knowledge-base Z_t, and the current value of k (line 20).

9.4.2.2 Example: JIT Learning in a Classification Systems

The experiment refers to a synthetic monodimensional classification problem with two equiprobable classes $\{\omega_1, \omega_2\}$ each of which ruled by a Gaussian distribution

Algorithm 25: H-CDT-based JIT Adaptive Classifier

```
1  $I_0 = \{1, \dots, T_0\}$ , $Z_0 = \{(x(t), y(t)),\ t \in I_0\}$,$O_{T_0}$;
2  Estimate $k_{\text{LOO}}$ by means of LOO on $Z_0$ and set $k = k_{\text{LOO}}$;
3  Configure the ICI-CDT part of H-CDT using $O_{T_0}$;
4  $Z_t = Z_0$,$I_t = I_0$,$t = T_0 + 1$;
5  while (1) do
6      Acquire $x(t)$ at time $t$;
7      if (supervised information $y(t)$ on $x(t)$ is available) then
8          $I_t = I_{t-1} \cup \{t\}$;
9          $Z_t = Z_{t-1} \cup \{(x(t), y(t))\}$;
10         update $k$ as in [104];
       else
11         $I_t = I_{t-1}$;
12         $Z_t = Z_{t-1}$;
       end
13     if (H-CDT detects concept drift on the sequence containing $x(t)$) then
14         Let $\hat{T}$ be the concept-drift detection time;
15         Extract $\bar{t}$ from H-CDT (Algorithm 23);
16         Configure ICI-CDT on $[\bar{t}, \hat{T}]$ and Hotelling on feature sequence $s|t > \bar{t}$;
17         $I_t = \left\{t \in T_t, t > \bar{t}\right\}$;
18         $Z_t = \{(x(t), y(t)), t \in I_t\}$;
19         Estimate $k_{\text{LOO}}$ by means of LOO on $Z_t$ and set $k = k_{\text{LOO}}$ ;
       end
20     Classify $x(t)$ as $k - NN(x(t), k, Z_t)$;
21     $t = t + 1$;
   end
```

$p(x|\omega_1) = \mathscr{N}(0, 4)$ and $p(x|\omega_2) = \mathscr{N}(2.5, 4)$. The experiment is composed of $N = 10,000$ scalar observations. An abrupt concept drift affects both classes at time $T^{\text{o}} = 5000$ by modifying the pdfs as $p(x|\omega_1) = \mathscr{N}(2, 4)$ and $p(x|\omega_2) = \mathscr{N}(4.5, 4)$. Figure 9.10a shows the data instances for the two classes over time.

The following adaptive classification frameworks have been considered for comparison:

- the proposed JIT classifier (green dashed line with a square marker).
- A classifier trained on all available data every time a new supervised couple is provided (dotted black line). This classifier guarantees the best performance in stationary conditions.
- A short memory classifier trained on a sliding window open over the latest 40 supervised samples (solid red line with circle marker).

All the considered adaptive classification frameworks rely on the k-NN classifier as a base classifier.

A supervised sample out of $m = 5$ observations is provided to the classifiers.

The classification accuracy on unsupervised samples is the figure of merit used to assess the performance of the considered adaptive classification framework. In

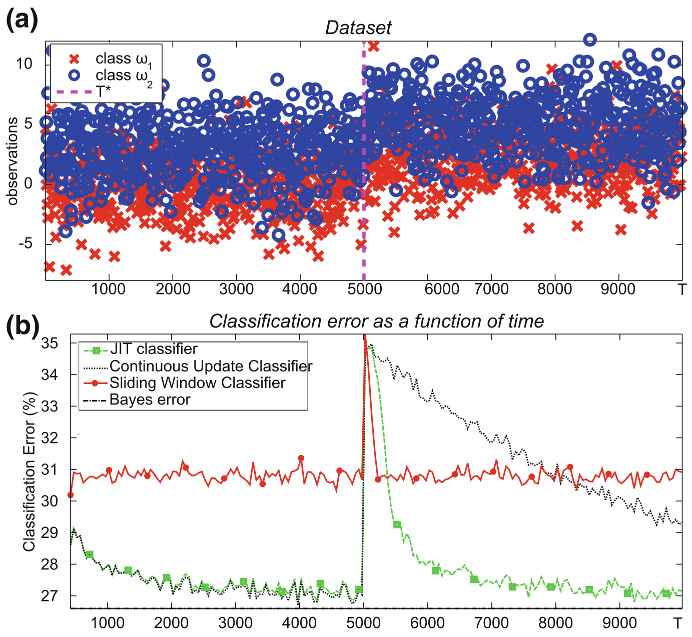

Fig. 9.10 An example of the just-in-time learning mechanism applied to a classifier. Data instances associated with the two classes are subject to a concept drift of abrupt type at T^{o} =5,000 that affects the mean of the distribution of class ω_2 (*upper plot*). The classification performance of the classifier are then compared with a short memory classifier implementing a batch online passive learning mechanism, the JIT classifier and the optimal Bays classifier (*lower plot*)

particular, Fig. 9.10b shows, at each time instant, the percentage of misclassified samples in 2,000 runs, averaged over a sliding window of 40 samples.

The JIT classifier tends to the Bayes error both before and after the change thanks to its ability to integrate fresh supervised samples during the operational life and to remove obsolete samples after a detected change. In fact, before T^{o} =5,000, the JIT classifier guarantees performance in line with those provided by the classifier trained on all available data (i.e., the black line) that, in stationary conditions, is able to guarantee the best performance.

After the change, the JIT classifier is able to promptly react to the change and adapt to the new working conditions thanks to its active detection/reaction approach. On the contrary, the classifier trained on all available data requires much more samples to adapt to the new working conditions since it is not endowed with a mechanism to remove obsolete samples.

Interestingly, the short memory classifier guarantees the best performance after the change since it is naturally able to remove obsolete samples through the windowing mechanism. Unfortunately, it is not able to improve its accuracy in stationary conditions since the sliding window is open over a fixed amount of samples (and this does not allow the base classifier to achieve the Bayes error).

9.4.3 Gradual Concept Drift

The proposed JIT grants asymptotic optimality when the process generating the data is affected by a sequence of abrupt concept drift [84] in the sense that, after concept drift is detected, the classifier's performance increases during operational life with provided additional supervised samples. The classic example is that of a quality inspection process where a supervisor is invoked time by time to provide an external quality evaluation which is the fresh information (supervised couple) that the JIT benefits to recover automatically from concept drift. Clearly, if the process is characterized by a sequence of concept drift that are too close in time, then the performance of the JIT might stay low even though the JIT classifier does its best to keep the highest accuracy possible compatible with the circumstances. In this case a passive-based classifier where k-NN is trained solely on the last N fixed data might provide better performance and be simpler from the complexity point of view.

This situation might also arise with gradual concept drift, obviously seen as a sequence of abrupt type of concept drift. Again, a passive solution might be preferable to the adaptive one if concept drift is fast, in the sense that its gradient over time is relevant (high developing concept drift). To address the gradual concept drift [97] proposes an extended JIT classifier introducing

- a modification of the ICI-CDT outlined in Sect. 9.3.2 that makes the CDT able to deal with a process whose expectation follows a polynomial trend.
- an adaptive classifier able to handle gradual concept drift affecting the process expectation. The classifier integrates an index estimating the evolution dynamics to improve classification accuracy.

Intuitively, the proposed extended classifier copes with gradual concept by estimating the concept drift trend, detrending the data and consider now the process as exhibiting a stationary state.

We model the gradual concept drift according to the formulation of equation (9.22). In particular, we focus on gradual concept drift that is represented by a possibly slow-time-varying stochastic process, whose expectation $\mathrm{E}[p(x|t)]$ follows a piecewise polynomial function $f_\theta(t)$. The parametric description of $f_\theta(t)$ is given by $\{(\theta_i, U_i)\}$ where θ_i is a parameter vector characterizing the polynomial $f_{\theta_i}(t)$ defined on the i-th time interval U_i (i.e., a subsequence of consecutive time instants). The expectations of the conditional probability distributions can be expressed as

$$\mathrm{E}[p(x|\omega_1, t)] = f_{\theta_i}(t) + q_{1,i} \tag{9.23}$$

$$\mathrm{E}[p(x|\omega_2, t)] = f_{\theta_i}(t) + q_{2,i} \tag{9.24}$$

where $t \in U_i$ and $q_{1,i}$ and $q_{2,i}$ are the expectations of the two classes ω_1 and ω_2 in stationary conditions. The process generating observations $x(t)$ at time t becomes

$$x(t) = \begin{cases} f_{\theta_i}(t) + \phi_{1,i}, & \text{if } y(t) = \omega_1 \\ f_{\theta_2}(t) + \phi_{2,i}, & \text{otherwise} \end{cases} \tag{9.25}$$

where $\phi_{1,i}$ and $\phi_{2,i}$ are random variables ruled by the pdfs characterizing the distributions of their respective classes ω_1 and ω_2 in stationary conditions with $\mathrm{E}[\phi_{1,i}] = q_{1,i}$ and with $\mathrm{E}[\phi_{2,i}] = q_{2,i}$.

We further assume that the probabilities $p(x|\omega_1, t)$ and $p(x|\omega_2, t)$ do not change within each interval defining the piecewise polynomial function, thus, the pdf of $x(t)$ is

$$p(x|t) = p_i(\omega_1)p(x|\omega_1, t) + p_i(\omega_2)p(x|\omega_2, t), \ t \in I_i.$$

The pdf of the inputs, the conditional distributions and the output distributions are unknown. The piecewise-polynomial function within each interval U_i, i.e., $f_{\theta_i}(t)$, is also unknown, but common between the two classes, as expressed in (9.24). We comment that the considered framework is an extension of the traditional one assuming constant $f_{\theta_i}(t)$.

9.4.4 JIT for Gradual Concept Drift

The key point of the proposed approach is to extend the observation model traditionally assumed in classification problems by allowing the expectation of the conditional probability density functions to evolve over time as a piecewise polynomial function, as expressed in (9.24). Under such a hypothesis, we can develop a CDT to assess variations in the (polynomial) trend of the process under monitoring, rather than in the value of its expectation. If the test does not detect variations, we perform a polynomial regression of the input samples and use the regression coefficients to modify online the knowledge base of an adaptive classifier. Differently, when a change is detected, the obsolete samples are removed from the knowledge base and the change detection test is restarted.

Since the ICI-CDT is natively able to deal with polynomial trends in the process under monitoring, it can be applied with minor modification at the feature level w.r.t. what is presented in Sect. 9.3.2. A detailed description can be found in [97].

Differently, the k-NN classifier has to be slightly modified to coherently adapt to gradual concept drift. Algorithm 26 presents the k-NN classifier able to deal with

Algorithm 26: Adaptive k-NN classifier for Gradual Concept drift

```
1- $N = |Z_T|$;
2- $i = 1$;
3- **while** $(i < N)$ **do**
4-     $d_i = \Big(\big(x(t) - f_{\hat{\theta}(t)}(t)\big) - \big(x(t_i) - f_{\hat{\theta}(t)}(t_i)\big)\Big)$;
5-     $i = i + 1$;
   **end**
6- Identify the nearest k training samples according to the distances $\{d_i\}_{i=1,\ldots,N}$;
7- Classify $x(t)$ as the majority of labels in the $k$ nearest training samples;
```

gradual concept drift. It is easy to see that the only difference w.r.t. the traditional k-NN classifier is the computation of the distance between the input sample and the training samples (line 4). Here, the parameter vector $\hat{\theta}(t)$ represents the coefficients of the best polynomial fit for the data during gradual concept drift. These coefficients can be estimated from the observations using any regression technique.

The polynomial fit represents the gradual concept drift, and is used to correct each term as function of the distance between the data and the fitted polynomial. In particular, the distance between the current sample $x(t)$ and the training sample $x(t_i)$ is computed after subtracting the values of the (estimated) polynomial having coefficients $\hat{\theta}(t)$ in their corresponding time instants (i.e., $f_{\hat{\theta}(t)}(t)$ and $f_{\hat{\theta}(t)}(t_i)$).

By replacing the CDT and the k-NN classifier it is then possible to formulate the JIT classifier for gradual concept drift following a similar scheme of Algorithm 25. Note that the proposed JIT for gradual concept drift is indeed an extension of Algorithm 25 as in absence of gradual concept drift, the higher order coefficients of the polynomial approach zero and the whole JIT operates as in stationary conditions.

9.4.5 Amygdala—VM-PFC—LPAC- ACC: The JIT Approach

The just-in-time learning framework is an example of a complex mechanism that founds its psychological roots in Piaget's theory of childhood learning. Detecting concept drift and reacting to it is aligned with Piagets psychological theory of human cognition [157], where learning is described as a constant effort to maintain or achieve balance between prior and new knowledge. As pointed out in [77], when new knowledge cannot be accommodated under existing schema because of severe conflict (i.e., nonstationarity), the need is to restructure the application to create new schemata that supplement or replace the prior knowledge base. While the former detection issue is addressed by the Amygdala—VM-PFC mechanism the need to supplement or replace the prior knowledge base (reaction process) is carried out by the LPAC-ACC layers.