On Sequences of Different Adaptive Mechanisms in Non-Stationary Regression Problems

Rashid Bakirov, Bogdan Gabrys and Damien Fay
Data Science Institute, Bournemouth University, United Kingdom
Email: {rbakirov,bgabrys,dfay}@bournemouth.ac.uk

***Abstract*—Existing adaptive predictive methods often use multiple adaptive mechanisms as part of their coping strategy in non-stationary environments. These mechanisms are usually deployed in a prescribed order which does not change. In this work we investigate and provide a comparative analysis of the effects of using a flexible order of adaptive mechanisms' deployment resulting in varying adaptation sequences. As a vehicle for this comparison, we use an adaptive ensemble method for regression in batch learning mode which employs several adaptive mechanisms to react to the changes in data. Using real world data from the process industry we demonstrate that such flexible deployment of available adaptive methods embedded in a cross-validatory framework can benefit the predictive accuracy over time.**

I. Introduction

The data streams generated in real world are often changing in terms of their underlying distributions which focuses attention on adaptive models designed for such scenarios. This is because such processes often break the data stationarity assumptions used in static models, which leads to degraded performance. Retraining of static models is an option, but in this case older data must either be excluded (losing the information therein) or appropriately weighted (which is in itself a simple form of adaptation). Therefore model adaptation is essential to maintain predictive performance. Here we define *adaptation* as changes in model parameters and or structure in response to changes in the data stream.

There are typically several possible ways or *adaptive mechanisms* (AMs) to adapt a given model. Often their deployment order is fixed at the design phase. The core aim of this paper is to examine the merits of such fixed orders and to investigate whether flexible deployment of adaptation mechanisms can provide higher accuracy over time.

This research specifically examines the batch prediction scenario, where data arrives in large segments called batches. For our experiments we have constructed a regression algorithm which uses an ensemble of locally weighted experts to make a prediction. The local experts approach is a well known (described for example in [1]) method of aggregation of multiple models' predictions. Its appeal for this research is that it allows us to investigate a number of adaptive mechanisms, such as adding/removing experts and changing experts' local weights. In addition, the local expert models used may take any form and may include their own adaptation mechanisms. Later, when the true values become available, the model may be adapted.

In the sections below we review the related approaches, describe the algorithm, identify all of its adaptive mechanisms, and test its different configurations. As a flexible AM selection method, we propose the strategy which uses that AM algorithm which, *a posteriori*, performed best on the previous batch and deploys it on the current batch. The main finding in this paper is that this strategy empirically outperforms a range of alternatives, including deploying any single AM or all of them repeatedly on every batch. The algorithm itself can also be considered as a secondary original contribution of the paper.

II. Related Works

Recently, especially for industrial processes, many regression methods which explicitly consider the adaptation of the model, such as [2]–[10], have been proposed. Adaptivity is usually achieved by building a predictive model using a) the latest historical data and/or b) the historical data which is the most similar to the current data. Adaptive methods often combine outputs of multiple models for final prediction, examples being [2]–[7]. These are known as ensemble methods. Most of the time ensemble members, or experts, are built on the subsets of historical data which represent different degrees of relation to the current data. The experts are then weighted appropriately.

Ensemble methods date as far back as 1960's, when it was shown that combining multiple predictive models may give better results than using single models [11]. One of the advantages of ensemble methods is the ability to model local dependencies in the data, a classical example being [1]. This is achieved by weighting the models' predictions on a data instance by the location of this instance in the input space. Local ensemble learning was applied to regression ensembles in [2], [12], [13]. These methods first identify the *receptive fields* which are disjoint segments of the historical input space where the process produced outputs described by a common model. Then they build a model for each receptive field using Partial Least Squares (PLS) [14]. The models therefore describe different stages of the process. The final prediction is a weighted average of all of the experts. Here, for each new data instance, the weights of experts are calculated online, depending on the location of the observed instance and the prediction. The AM used in [12] is based on change of models' local weights depending on their error. This model was extended in [2] to include adaptation of the base models according to [15]. [13] further extends the model to include

creation of additional experts. Another local ensemble method, with a moving window and weights change AMs is described in [3].

Also popular in the literature are global regression ensembles [4]–[6]. These typically assign weights to experts based on their general performance, not considering the local aspects of data. Global ensemble methods use similar AMs. For instance, [4] adapts to changes by creating new experts and changing their weights. [6] includes AMs such as adaptation of base models via a moving window strategy, changing experts weights and adding new experts. [5] additionally employs instance weighting. Both [5] and [6] may remove experts as well. A method which uses an ensemble of univariate regressors for multivariate regression is described in [7]. It includes weighting of models and forgetting factor AMs. [16] uses *time difference* ensemble based on the distance between the current input and historical inputs. Method can use either moving window or just-in-time (local data window) approaches for adaptation.

The algorithms described above are examples of incremental learning. Adaptivity in batch learning mode has not been explored extensively in the regression literature. However it has been featured quite prominently for classification, for example in [17]–[20] for applications such as image recognition, network intrusion detection, credit card fraud identification etc.

In this section we have reviewed relevant models with multiple AMs which are based on ensemble methods. Each of them usually includes several AMs which are deployed in a fixed order. Most AMs, such as adapting the combination weights or the parameters of the base models, are deployed at every time step. Some works offer basic flexibility for some AMs. For instance, [13] creates new experts when existing ones are not built on the relevant data, [5], [6] create new experts when the predictive error on an instance is above a set threshold. In [16] the predictive accuracy is assessed to switch between two predictive models.

What is common about the above discussed selection of methods and algorithms is that they all use multiple AMs but in a fixed order and manner. Only very few works consider the possibility of adaptation without any fixed order. One of these, [21] presents a plug and play architecture for preprocessing, adaptation and prediction which foresees the possibility of using different adaptation methods modularly, but does not address the method of AM selection. In this paper we investigate of how changing the order and the type of adaptation affects the system characteristics and performance.

III. Formulation

We assume that the data is generated by a process which can be formulated as

$$y = \psi(\boldsymbol{x}) + \epsilon, \tag{1}$$

where ψ is an unknown function, ϵ is unknown error, $\boldsymbol{x} = \{x_1, ..., x_m\}$ is an input data instance and y is the output value. Then we consider the predictive method at a time t as

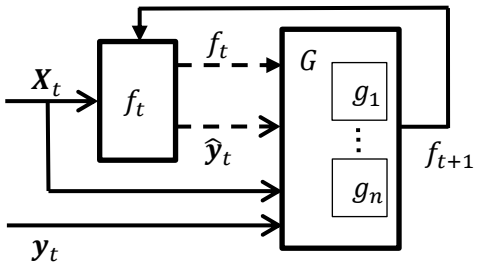

Fig. 1. Adaptation with multiple AMs. Optional inputs are shown with dashed lines.

a function

$$\hat{y} = f_t(\boldsymbol{x}, \Theta_f), \tag{2}$$

where $\hat{y}$ is the predicted output and Θ_f set of parameters of f.

We assume that we are operating in the batch streaming scenario, and receive a new input dataset, also called batch, $\boldsymbol{x}_t$, at time t. The predictive model f_t generates prediction vector $\hat{\boldsymbol{y}}_t$. Subsequently true values, $\boldsymbol{y}_t$, become available, so that $\boldsymbol{V}_t = \{\boldsymbol{x}_t, \boldsymbol{y}_t\}$. Then we consider an *adaptive mechanism* as an operator which generates an updated prediction function based on $\boldsymbol{V}_t$ and other optional inputs. This can be written as

$$g(f_t, \hat{\boldsymbol{y}}_t, \boldsymbol{V}_t, \Theta_g) : f_t \rightarrow f_{t+1}. \tag{3}$$

Here f_t and $\hat{\boldsymbol{y}}_t$ are optional arguments and Θ_g is the set of parameters of g.

We assume that we have multiple, different AMs $\{\emptyset, g_1, ..., g_h\} = G$ available. Datasets $\boldsymbol{V}_1, ..., \boldsymbol{V}_T$ arrive at timesteps $t = 1, ...T$. At each timestep any combination of the AMs $G_t \subset G$ can be executed. We call a sequence $G_1, ..., G_T$ an *adaptation sequence*. Note we also include the option of applying no adaptation at any particular stage denoted by the null set. In this work we limit ourselves to the scenario where $||G|| = 1$, meaning that at most one AM per data batch can be deployed. Figure 1 illustrates the proposed adaptation scheme.

IV. Algorithm

To perform experiments, we are using an adaptive method, Simple Adaptive Batch Local Ensemble (SABLE), which is inspired by the ILLSA method [2]. ILLSA is a local learning ensemble for regression, which uses Recursive PLS [15] as its base learner. While it predicts and learns/adapts incrementally on single instances, SABLE operates on batches of data. An expert can be created from each batch of data. The experts are trained using PLS (see Section VI) as a base method. PLS was chosen because it is widely used for predictions in chemical processes where our datasets originate from, features inherent dimension reduction and has an updatable version. The final prediction is the weighted sum of prediction of all experts where the weights are calculated as described below. We list the main parts and adaptive mechanisms of SABLE in the following sections.

A. Building of Experts' Descriptors

The relative (to each other) performance of experts varies in different parts of the input/output space. In order to quantify this a *descriptor* is used. Descriptors of experts are distributions of their weights with the aim to describe the area of expertise of the particular local expert. They describe the mappings from a particular input, x^m, and output, y to a weight, denoted $D_{i,m}(x^m, y)$, where m is the m-th input feature[1] and i is the i-th expert. The descriptor is constructed using a two-dimensional Parzen window method [22],

$$D_{i,m} = \frac{1}{||\boldsymbol{V}_i^{tr}||} \sum_{j=1}^{||\boldsymbol{V}_i^{tr}||} w(\boldsymbol{x}_j)\Phi(\mu_j^m, \boldsymbol{\Sigma}) \tag{4}$$

where $\boldsymbol{V}_i^{tr}$ is the training data of i-th expert, $||\boldsymbol{V}_i^{tr}||$ is the number of instances it includes, $w(\boldsymbol{x}_j)$ is the weight of sample point's contribution which is defined below, $\boldsymbol{x}_j$ is the jth sample of $\boldsymbol{V}_i^{tr}$, $\Phi(\mu_j^m, \boldsymbol{\Sigma})$ is two-dimensional Gaussian kernel function with mean value $\mu = (x_j^m, y_j)$ and variance matrix $\boldsymbol{\Sigma} \in \Re^{2\times 2}$ with the kernel width, σ, at the diagonal positions. σ, is unknown and must be estimated as a hyperparameter of the overall algorithm[2].

The weights $w(\boldsymbol{x}_j)$ for the construction of the descriptors (see Eq. 4) are proportional to the prediction error of the respective local expert:

$$w(\boldsymbol{x}_j) = \exp(-(\hat{y}_j - y_j)^2) \tag{5}$$

Finally, considering that there are M input variables and I models, the descriptors may be represented by a matrix, $\mathcal{D} \in \Re^{M\times I}$ called the *descriptor matrix*.

B. Combination of Experts' Predictions

During the run-time phase, SABLE must make a prediction of the target variable given a batch of new data samples. This is done using a set of trained local experts $\mathcal{F}$ and descriptors $\mathcal{D}$. Each expert makes a prediction $\hat{y}_i$ for a data instance $\boldsymbol{x}$ The final prediction $\hat{y}$ is the weighted sum of the local experts' predictions:

$$\hat{y} = \sum_{i=1}^{I} v_i(\boldsymbol{x}, \hat{y}_i)\hat{y}_i \tag{6}$$

where $v_i(\boldsymbol{x}, \hat{y}_i)$ is the weight of the i-th local expert's prediction. The weights are calculated using the descriptors, which estimate the performance of the experts in the different regions of the input space. This can be expressed as the posterior probability of the i-th expert given the test sample $\boldsymbol{x}$ and the local expert prediction $\hat{y}_i$:

$$v_i(\boldsymbol{x}, \hat{y}_i) = p(i|\boldsymbol{x}, \hat{y}_i) = \frac{p(\boldsymbol{x}, \hat{y}_i|i)p(i)}{\Sigma_{j=1}^{I} p(\boldsymbol{x}, \hat{y}_j|j)p(j)}, \tag{7}$$

where $p(i)$ is the *a priori* probability of the i-th expert, $\Sigma_{j=1}^{I} p(\boldsymbol{x}, \hat{y}_j)p(j)$ is a normalisation factor and $p(\boldsymbol{x}, \hat{y}_i|i)$ is the likelihood of $\boldsymbol{x}$ given the expert, which can be calculated by reading the descriptors at the positions defined by the sample $\boldsymbol{x}$ and prediction $\hat{y}_i$:

$$p(\boldsymbol{x}, \hat{y}_i|i) = \prod_{m=1}^{M} p(x^m, \hat{y}_i|i) = \prod_{m=1}^{M} D_{i,m}(x^m, \hat{y}_i). \tag{8}$$

Eq. 8 shows that the descriptors D_m are sampled at the position which are given on one hand by the scalar value x^m of the m-th feature of the sample point $\boldsymbol{x}$ and on the other hand by the predicted output $\hat{y}_i$ of the local expert corresponding to the ith receptive field. Sampling the descriptors at the positions of the predicted outputs may be potentially ineffective because the predicted value is not necessarily similar to the correct target value. However the correct target values are not available at the time of the prediction. The rationale for this approach is that the local expert is likely to be more accurate if it generates a prediction which conforms with an area occupied by a large number of true values during the training phase.

In practice our descriptors have a finite support as we set their value to 0 if x^m falls outside the interval $[min_{\boldsymbol{x}_{tr}^m}\ max_{\boldsymbol{x}_{tr}^m}]$ or $\hat{y}_i$ falls outside the interval $[min_{\boldsymbol{y}_{tr}}\ max_{\boldsymbol{y}_{tr}}]$ where $\boldsymbol{x}_{tr}$ is the training input data and $\boldsymbol{y}_{tr}$ is the training values of the expert. If the weights for all the experts in the ensemble are set to 0 in this fashion, the most recent expert is given the weight of 1 as it is assumed that it reflects the nature of the current data better than the older ones.

To reduce the number of redundant experts, at time t those that deliver similar predictions on batch $\boldsymbol{V}_t$ are merged, making use of the linear base model. If the base model is not linear and merging is not straightforward, then a pruning strategy as in [12] can be considered. There, the weight vectors of all experts on a batch of data are pairwise compared and if their similarity is higher than the defined threshold, one of the experts is removed. Prediction vectors can also be used to measure the similarity between different experts.

V. Adaptive Mechanisms

The SABLE algorithm allows the use of adaptive mechanisms. AMs are deployed as soon as the true values for the batch are available, before predicting on the next batch. It is also possible that none of them are deployed. The AMs are described in the following sections.

A. Batch Learning

The simplest AM augments existing data with the data from the new batch and retrains the model. Given predictions of each expert $f_i \in \mathcal{F}$ on $\boldsymbol{V}$, $\hat{Y} = \{\hat{\boldsymbol{y}}_1, ..., \hat{\boldsymbol{y}}_I\}$ and measurements of the actual values, $\boldsymbol{y}$, $\boldsymbol{V}$ is partitioned into subsets in the following fashion:

$$k = \underset{i}{argmin}(|\hat{y}_{i,j} - y_j|),\ i = 1 \cdots I,\ [\boldsymbol{x_j}, y_j] \in \boldsymbol{V}_k \tag{9}$$

for every instance $[\boldsymbol{x}_j, y_j] \in \boldsymbol{V}$. This creates subsets $\boldsymbol{V}_i, i = 1...I$ such that $\cup_{i=1}^{I} \boldsymbol{V}_i = \boldsymbol{V}$. Then each expert is updated using the respective dataset $\boldsymbol{V}_i$. This process updates

[1]For the base methods which transform the input space, such as PLS, the transformed input arguments are used instead of original ones.

[2]In this research we assume an isotropic kernel is sufficient for simplicity and also to reduce the number of parameters to be estimated.

experts only with the instances where they achieve the most accurate predictions, thus encouraging the specialisation of experts and ensuring that a single data instance is not used in the training data of multiple experts. This AM will be denoted as **AM1** in the description of the experiments below.

B. Batch Learning With Forgetting

This AM is similar to one described in Section V-A but includes a forgetting factor (see Section VI) which reduces the weight of the experts historical training data, making the most recent data more important. This AM will be denoted as **AM2**.

C. Recalculation of Descriptors

This AM recalculates the local descriptors, $\mathcal{D}$, using the new batch as described in the Section IV-A. The previous weights are discarded. This AM will be denoted as **AM3**.

D. Creation of New Experts

New expert f_{new} is created from the new batch at time t. Then it is checked if any of the experts from $\mathcal{F}_{t-1} \cup f_{new}$, where $\mathcal{F}_{t-1}$ is the experts pool at the time $t-1$, can be merged or pruned. Finally the descriptors of all resulting experts are calculated (Section IV-A). This AM will be denoted as **AM4**.

VI. Recursive Partial Least Squares

SABLE is conceived as an algorithm which can function with any base prediction model. In our experiments we use Recursive Partial Least Squares (RPLS) [15] as a base algorithm. The advantages of this algorithm are that it can be updated without requiring the historical data and that the merging of two models can be easily realised. RPLS is extension of the Partial Least Squares algorithm, both of which are popular in chemical process modelling. PLS projects the scaled and mean centered multidimensional input data $\boldsymbol{X} \in \mathcal{R}^{n \times m}$ and output data $Y \in \mathcal{R}^{n \times p}$, where n is the number of data instances, m is the number of input variables and p is the number of output variables, to separate latent variables,

$$\boldsymbol{X} = \boldsymbol{T}\boldsymbol{P}^T + \boldsymbol{E} \tag{10}$$

$$\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{Q}^T + \boldsymbol{F}. \tag{11}$$

Here $\boldsymbol{T} \in \mathcal{R}^{n \times l}$ ($l \leq m$ as the number of latent variables) and $\boldsymbol{U} \in \mathcal{R}^{n \times l}$ are the score matrices, $\boldsymbol{P} \in \mathcal{R}^{m \times l}$ and $\boldsymbol{Q} \in \mathcal{R}^{p \times l}$ are the corresponding loading matrices, and $\boldsymbol{E}$ and $\boldsymbol{F}$ are the input and output data residuals. Then the score matrices $\boldsymbol{T}$ and $\boldsymbol{U}$ consist of so called latent vectors:

$$\boldsymbol{T} = [\boldsymbol{t}_1, ..., \boldsymbol{t}_l], \text{ where } \boldsymbol{t}_i \in \mathcal{R}^{n \times 1} \tag{12}$$

$$\boldsymbol{U} = [\boldsymbol{u}_1, ..., \boldsymbol{u}_l], \text{ where } \boldsymbol{u}_l \in \mathcal{R}^{n \times 1}. \tag{13}$$

where the column vectors $\boldsymbol{p} \in \mathcal{R}^{m \times 1}$ and $\boldsymbol{q} \in \mathcal{R}^{m \times 1}$ of the loading matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$ represent the contributions of the input and output variables to the mutually orthonormal latent vectors $\boldsymbol{t}$ and $\boldsymbol{u}$, respectively. Equations 12 and 13 constitute the PLS outer model. Afterwards a regression model, which is also called the PLS inner model, between the latent scores is constructed:

$$\boldsymbol{U} = \boldsymbol{T}\boldsymbol{B} + \boldsymbol{R}, \tag{14}$$

where $\boldsymbol{B} \in \mathcal{R}^{l \times l}$ is a diagonal matrix of regression weights which minimizes the regression residuals $\boldsymbol{R}$. Then the estimates $\hat{\boldsymbol{Y}}$ of $\boldsymbol{Y}$ are:

$$\hat{\boldsymbol{Y}} = \boldsymbol{T}\boldsymbol{B}\boldsymbol{Q}^T, \tag{15}$$

There are different methods to calculate the required vectors $\boldsymbol{t}$, $\boldsymbol{p}$, $\boldsymbol{u}$, $\boldsymbol{q}$ and $\boldsymbol{b}$. One of the most popular ones, NIPALS [23], updates latent vectors in an iterative way. After each iteration, the explained covariance is removed from the data:

$$\boldsymbol{X}_{i+1} = \boldsymbol{X}_i - \boldsymbol{t}_i \boldsymbol{p}_i^T \tag{16}$$

$$\boldsymbol{Y}_{i+1} = \boldsymbol{Y}_i - \boldsymbol{u}_i \boldsymbol{q}_i^T. \tag{17}$$

The subsequent $(i+1)$-th vectors are calculated by the resulting new input and output data $\boldsymbol{X}_{i+1}$ and $\boldsymbol{Y}_{i+1}$. Recursive PLS, which uses NIPALS, updates the matrices $\boldsymbol{P}$, $\boldsymbol{T}$, $\boldsymbol{Q}$, $\boldsymbol{U}$ and $\boldsymbol{B}$ when the new data becomes available, on either sample-by-sample (incremental) or batch basis. In this work we are using batch adaptation. It works by applying PLS on the new batch and constructs new input and output matrices as follows:

$$\boldsymbol{X}_{new} = \begin{bmatrix} \lambda \boldsymbol{P}_0^T \\ \boldsymbol{P}_1^T \end{bmatrix} \tag{18}$$

$$Y_{new} = \begin{bmatrix} \lambda \boldsymbol{B}_0 \boldsymbol{Q}_0^T \\ \boldsymbol{B}_1 \boldsymbol{Q}_1^T \end{bmatrix}, \tag{19}$$

where the matrices $\boldsymbol{P}_0$, $\boldsymbol{B}_0$ and $\boldsymbol{Q}_0$ describe the old model and $\boldsymbol{P}_1$, $\boldsymbol{B}_1$ and $\boldsymbol{Q}_1$ the new one created from the most recent batch. $0 \leq \lambda \leq 1$ is the forgetting factor which determines how much influence the historic data will have, with $\lambda = 0$ meaning zero influence and $\lambda = 1$ meaning that the historical data has the same influence as the new batch. After constructing the new input and output data matrices, PLS is applied on them to get the updated matrices. The condition for this update is that the number of latent variables must be equal to the rank of $\boldsymbol{X}$. This condition can be practically met by finding a number of latent variables a for which the error on the training data is less than the defined threshold close to 0.

VII. Experimental Methodology

The experiments were performed on 3 datasets from the process industry which will be described later. As explained in [24], the first of these datasets (catalyst) requires strong adaptation, the second (drier) does not and the third (oxidizer) lies somewhere in between. Choosing these datasets will allow us to test AM sequences on data which exhibits different behaviours. With our experiments we aim to determine whether deploying some of AMs in certain order on the incoming batches of data is significantly different than deploying them in another order, or deploying them all on every batch. We also would like to know the order of deployment which minimizes the total average error, and whether using cross-validation on the current batch to choose the AM for the next batch provides

improvement in comparison to other methods. In detail, we assess the following:

- Performance of AMs separately from each other. For this purpose a single AM is deployed after every new batch of data,
- Whether the historical data can help in selecting AM for the next batch of data. For this purpose, after receiving a data batch with true values $\boldsymbol{V}_t = [\boldsymbol{X}_t, \boldsymbol{y}_t]$ at time t, all of the available AMs, $g_1, ..., g_h$ are deployed on it and the mean average errors after adaptation $e_{t,1}, ..., e_{t,h}$ are measured using a 10-fold cross-validation. The AM with the least mean absolute error (MAE) is then deployed on $\boldsymbol{V}_t$:

$$G_t = g_k \text{ where } k = \underset{i}{argmin}(e_{t,i}),\ i = 1...h, \quad (20)$$

- What is the adaptation sequence which minimizes the error on each subsequent batch[3]. For this purpose, after receiving a data batch with true values $\boldsymbol{V}_t = [\boldsymbol{X}_t, \boldsymbol{y}_t]$ at time t, all of the available AMs $g_1, ..., g_h$ are deployed on it. Then the resulting models $f'_1, ..., f'_h$ are applied to the next data batch $\boldsymbol{V}_{t+1} = [\boldsymbol{X}_{t+1}, \boldsymbol{y}_{t+1}]$ getting in prediction vectors $\hat{\boldsymbol{y}}_{t+1,1}, ...\hat{\boldsymbol{y}}_{t+1,h}$ and MAE values:

$$e_{t+1,1} = \frac{\Sigma_{j=1}^{n} |y_{t+1}^{j} - \hat{y}_{t+1,1}^{j}|}{n}, ..., \quad e_{t+1,h} = \frac{\Sigma_{j=1}^{n} |y_{t+1}^{j} - \hat{y}_{t+1,m}^{j}|}{n}, \quad (21)$$

where n is the number of instances in $\boldsymbol{V}_{t+1}$. The AM which provides the least error is then deployed on $\boldsymbol{V}_t$:

$$G_t = g_k \text{ where } k = \underset{i}{argmin}(e_{t+1,i})\ i = 1...h. \quad (22)$$

For the purposes above, we run configurations of SABLE, shown in the Table I on each of the datasets. On each batch one of the AMs described in the Section V or none of them (which is denoted by **AM0**) can be deployed.

For the simplicity reasons we look at the case where only one AM can be used on each batch, however the approach can be extended beyond this restriction. We use the AM configurations presented in Table I for AM selection.

To calculate the significance of differences between the predictions of different configurations, the significance test of difference of two estimators' errors relying on the sample covariance ([25], Section 3.2) was used. Kernel width $\sigma = 1$ was chosen for all experiments. The batch sizes for each dataset were set to ensure that the linear expert has enough data instances to be trained. In general, the size of the batch plays a decisive role in the results, as small batches can lead to experts trained with insufficient data, and large batches may span through different stages of process which would also lead to ineffective adaptation.

[3]Note that this is not applicable in the real life situations, as at the time t, the data $\boldsymbol{V}_{t+1}$ is not yet known. This configuration's shows achievable results using one step ahead optimal AM and will be used as a benchmark.

TABLE I
EXPERIMENTAL CONFIGURATIONS OF SABLE.

Configuration name	Description
NoAdapt	Using AM0 on every batch. This means that only the first batch of data is used to create an expert, which then predicts on all subsequent batches.
Sequence1	Using AM1 on every batch.
Sequence2	Using AM2 on every batch.
Sequence4	Using AM4 on every batch.
Joint	Deploy AM2 and AM4 together on every data batch.
Crossvalidation	Selecting AM based on the current data batch using the cross-validatory approach described above.
Optimal	Selecting AM based on the future data batch as described before. Used for benchmarking.
Random200	Randomly select AM for each batch. We report the average errors from 200 random runs.
Retrain	A new model is trained from the current batch and the old one is discarded. No data partitioning is used.

VIII. EXPERIMENTAL RESULTS

A. Catalyst Activation Dataset

This data set was used for the NiSIS 2006 competition[4]. It includes 14 sensor measurements like flows, concentrations and temperatures from a real process. The target variable is the simulation of catalyst's activity inside the reactor. The description of the reaction speed is taken from literature[4] showing a strong non-linear dependency on temperature. Further complicated processes like cooling and catalyst decay contribute to changes in the data. The data set covers one year of operation of the plant. Many of the features exhibit high co-linearity and contain high number of outliers. The data includes 5,867 data samples and is presented to an algorithm in batches of 100 samples. We have removed two features with mostly missing and 0 values during the preprocessing. Number of latent vectors for PLS was experimentally set to 12.

MAE and RMSE values resulting from application of the different configurations of SABLE to the catalyst data and rest of the used datasets can be found in Table II. Table III shows the AM sequences identified by ***Optimal*** and ***Crossvalidation***.

As expected, the benchmark configuration ***Optimal*** provides the best results. Among the realistic SABLE configurations, ***Crossvalidation***, ***Retrain*** and ***Sequence2*** are the most accurate ones whereby ***Crossvalidation***'s error is significantly less than the rest of configurations. These high accuracy of ***Retrain*** shows that the catalyst dataset requires very strong adaptation, and that data from older batches is much less important than the data from the last batch.

[4]http://www.nisis.risk-technologies.com/%28S%28ftalbreakikhjt34pd2xggfu%29%29/filedown.aspx?file=125

TABLE II
RESULTS ON ALL DATASETS. THE LOWEST ERROR MEASURES OF REALISTIC CONFIGURATIONS ARE HIGHLIGHTED WITH ITALIC BOLD.

	Catalyst		Drier		Oxidizer	
Configuration	***MAE***	***RMSE***	***MAE***	***RMSE***	***MAE***	***RMSE***
Optimal	*0.024*	*0.047*	$3.42 * 10^{-6}$	$1.27 * 10^{-5}$	*0.448*	*0.860*
Crossvalidation	***0.026***	***0.050***	$1.07 * 10^{-5}$	$3.06 * 10^{-5}$	***0.477***	1.110
Sequence2	0.031	0.067	$2.06 * 10^{-5}$	$1.16 * 10^{-4}$	0.490	***0.838***
Retrain	0.028	0.058	$2.59 * 10^{-5}$	$1.54 * 10^{-4}$	0.499	0.854
Joint	0.032	0.063	$2.31 * 10^{-5}$	$1.30 * 10^{-4}$	0.519	0.860
Sequence4	0.030	0.065	$1.18 * 10^{-5}$	$2.83 * 10^{-5}$	0.521	0.860
Sequence1	0.147	0.260	$\boldsymbol{8.09 * 10^{-6}}$	$\boldsymbol{2.04 * 10^{-5}}$	0.640	0.998
Random200	0.075	0.138	$3.92 * 10^{-5}$	$1.06 * 10^{-4}$	0.651	1.041
NoAdapt	0.279	0.313	$5.41 * 10^{-4}$	$5.97 * 10^{-4}$	0.760	1.181

TABLE III
AM SEQUENCES IDENTIFIED BY ***Optimal*** AND ***Crossvalidation***. THE CORRESPONDING AMS ARE HIGHLIGHTED WITH BOLD.

	AM sequence	Same AMs	MAE
Catalyst			
Crossvalidation	214144442444144144242444442212144444222244212214444303412	31/57 (54%)	0.026
Optimal	214214442344344444234444442112344444121141114034222442344		0.024
Drier			
Crossvalidation	13400400000	8/11 (73%)	1.07E-05
Optimal	12101400000		3.42E-06
Oxidizer			
Crossvalidation	2102201101224403400233444144433040043443103400343 34444	18/55 (33%)	0.477
Optimal	4434442104034430414430041433044043041044344334143033444		0.448

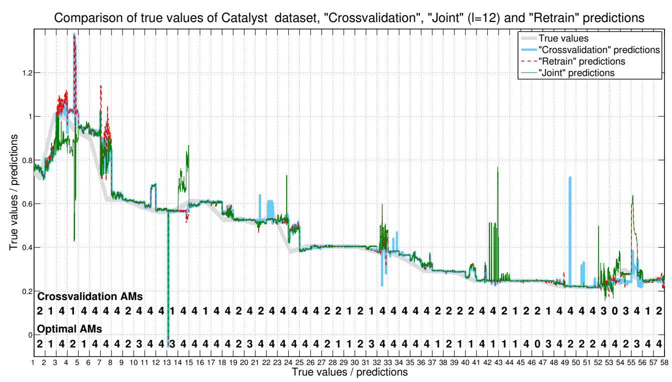

Fig. 2. Results on the catalyst dataset including the ***Crossvalidation*** and ***Optimal*** AM sequences.

Figure 2 shows the comparison of predictions of ***Crossvalidation***, ***Retrain*** and ***Joint*** on catalyst dataset. One can see that comparing to ***Retrain***, the ***Crossvalidation*** performs noticeably better in several segments of data, like batches #5, 8, 33 56, however performs worse in batch #23 and has few outlier predictions in batch #50. Both configurations have outlier predictions in batch #14. ***Joint*** additionally shows unstable behaviour on batches #5, #15, #43 and #53.

In 54% of the batches (see Table III) ***Crossvalidation*** uses the same AMs as ***Optimal***. It must be noted that when the same AM is deployed on a certain batch while executing two different AM sequences, this may have different effects on respective models, depending on the current state of each model.

B. Industrial Drier Dataset

The target value of this dataset describes the laboratory measurements of the residual humidity of the process product. The dataset has 19 input features, most of them being temperatures, pressures and humidities measured in the processing plant. The original dataset consists of 1,219 data samples covering almost seven months of the operation of the process. It consists of raw unprocessed data as recorded by the process information and measurement system. Many of the input variables show problems common in industrial data like measurement noise, missing values or data outliers. We have removed 3 input features which mostly consisted of missing data. The data is presented in batches of 100 data samples. Number of latent variables for PLS was experimentally set to 16.

As seen from the Table II, ***Optimal*** configuration is again the most accurate. Interestingly, the second best is ***Sequence1*** followed by ***Crossvalidation***. This fact shows that the drier dataset does not necessarily require any adaptive method, but rather continuous learning, which indicates the lack of significant changes in the data. The stability of the data is also confirmed by many AM0s in the ***Optimal*** sequence (see Table III). Since the data is quite stable, learning with forgetting actually causes a deterioration in performance when compared to learning without forgetting, which can be seen when comparing the accuracies of ***Sequence1*** and ***Sequence2/Retrain***. Other interesting observations are that there are only two AM1 in ***Optimal*** sequence, despite ***Sequence1*** (consisting *only* of AM1s) being the second best. Despite ***Optimal*** and ***Crossvalidation*** deploy the same AMs on 8 of the 11 batches, the results are significantly different. The AM order generated by ***Crossvalidation*** performs significantly better than the rest

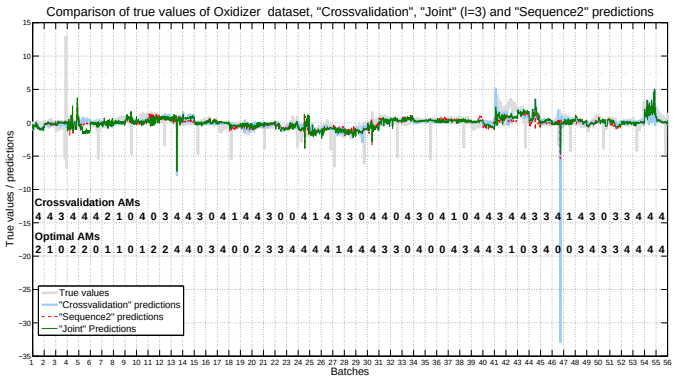

Fig. 3. Results on the oxidizer dataset including the ***Crossvalidation*** and ***Optimal*** AM sequences.

of configurations except ***Sequence1*** ($\alpha = 0.05$).

C. Thermal Oxidizer Dataset

This dataset deals with the prediction of the concentration of exhaust gas during an industrial process where task is to predict the concentrations of NO_x in the exhaust gases. The data set consists of 36 input features which are hard sensor measurements. They are physical values like concentrations, flows, pressures and temperatures measured during the operation of the plant. The data set consists of 2,820 samples. This dataset is also affected by issues like data outliers or missing values. We split it into batches of 50 samples, 55 batches in total. The number of latent vectors for PLS was experimentally set to 3.

From the Table II it can be seen that the best performing configuration for this dataset in terms of MAE is ***Optimal***. On the other hand, ***Sequence2*** performs the best when considering RMSE. This can be explained by the fact that ***Optimal*** optimizes the MAE. ***Retrain*** achieves slightly lower accuracy than ***Sequence2*** but higher accuracy than ***Sequence1*** which shows that the required adaptation level for this dataset is in between ***Retraining*** ($\lambda = 0$) and ***Sequence1*** ($\lambda = 1$). In only 33% of batches ***Crossvalidation*** and ***Optimal*** deploy the same AMs, despite the close results (see Table III). Predictions of ***Crossvalidation*** are significantly more accurate than the rest of realistic configurations with $\alpha = 0.1$.

Figure 3 shows the predictions of ***Crossvalidation***, ***Sequence2*** and ***Joint*** for the oxidizer dataset. The results are visually similar, except perhaps batches #5, #25 where it is possible to see the bigger error of ***Sequence2*** and ***Joint***. On the other hand ***Crossvalidation*** has several large outlier predictions on batch #47. This may explain why ***Crossvalidation*** has lower MAE but higher RMSE than ***Sequence2***. It should be remembered that the measure on which the selection of the AM using cross-validation was based had been MAE.

IX. Discussions and Conclusions

The core aim of this paper was to investigate whether deployment of different sequences of AMs makes a significant difference in the prediction accuracy of the model and how to select an appropriate AM for each new batch of data. We have conducted experiments on 3 industrial datasets describing different processes. During experiments, we have established that the datasets behave quite differently - the catalyst dataset changing faster than the others and requiring strong adaptation, the drier data being quite stable, and the oxidizer data falling between those two. This could be seen by comparing results of ***Retraining***, ***Sequence1*** and ***Sequence2*** for each of the datasets and recognizing that for the catalyst data configuration which performs best among those is ***Retraining***, for drier data ***Sequence1***, and for oxidizer data ***Sequence2***. This conclusion is in line with the analysis in [24].

Using the AM which minimizes the MAE on the current batch (***Optimal***) always led to the lowest MAE and almost always to the lowest RMSE values. Thus, for our setup, it made sense using ***Optimal*** as a benchmark. It must be noted that it is possible that there exits an AM sequence which minimizes the error even further. However it must be sought in the set of all possible AM sequences, which is computationally prohibitive.

Using cross-validation on previous batch is was the method used to select the AM for the next batch. It was found empirically to be the most effective when the process was stable for several batches. In this scenario the algorithm selects the AM based on the first batch after the change, which is the optimal adaptation for the stable period. If needed, the model will be adapted using the next batches as well. Experiments show that this technique (***Crossvalidation***) almost always provides the best results among the realistic configurations. An exception was the relatively stable drier dataset.

Even though AM4 is selected the most often in ***Optimal*** configurations for catalyst (29 out of 57 batches) and oxidizer (26 out of 55) datasets, applying *only* AM4 - configuration ***Sequence4*** does not produce the best results. This fact shows that applying only one, even the best AM may be inferior to using multiple AMs. Similarly applying *all* the AMs at every step (***Joint*** configuration) resulted in inferior performance. This may be explained by the fact that some batches can be inappropriate to deploy certain AMs on them - for example creating a single expert on noisy data or data with high variability would negatively affect the performance of the model. On the other hand deploying all of the AMs together in an uncontrolled manner could adapt the model too strongly. Thus, the need for an intelligent AM selection mechanism is further emphasised.

To sum up, we have established that different adaptation sequences can have significantly different results. There was no single best adaptive mechanism identified among the ones in our experiment, and deploying all of them at every batch was also not the best solution. Thus the benefits of a modular adaptive structure and flexible AM selection, for example using cross-validation, are apparent. The conclusions above may be relevant not only to our experiment, but also to the body of work dealing with adaptation which uses similar adaptive mechanisms - adding and removing local experts, weights' change and retraining. Our study also adds to this body of

work, presenting a viable adaptive regression algorithm, which can be used either with RPLS or other regression techniques as the base method.

Even though cross-validation provides a relatively accurate way of AM selection, more advanced ways of AM sequence generation may improve both the computational time and the accuracy of predictions. One of these ways can be using the severity of changes may be used to identify appropriate AMs. For example, from the Figure 2 we can see that relatively stable parts of the data (e.g. batches #28-#32, #43-#48) do not require AM4. It might be also tried to model the sequence generation.

In this work we have considered a batch learning scenario. As noted in [20] batch learning methods benefit from the availability of larger amounts of data, which allows one to make more informed decisions. Batch size plays a crucial role in the performance of these methods, including SABLE. Smaller batch sizes improve the reaction time to changes, but may result in being overly sensitive to outliers and creation of the less accurate experts. Larger batch sizes provide more stable performance, but react to changes slower. It is therefore planned to extend our research to incremental learning.

The hyperparameters (such as the number of latent vectors, forgetting factor, kernel size etc.) were not adjusted from their initial estimates in this work. To do so would add an extra layer in the models hierarchy, and may improve the results, which will be examined in future work.

Another aspect of the adaptation which was not considered in this paper were various parameters of adaptation and the algorithm, such as the number of latent vectors, forgetting factor, kernel size and so on. These were selected in the beginning of the training and kept the same throughout. Adapting the parameters of algorithm may also improve the results, and can be a topic of future research.

Finally, more complex version of SABLE would be able to partition the batches to possibly create more than one expert from each batch. This feature is useful when there are changes in the process which are happening within the batch.

Acknowledgements

Research leading to these results has received funding from the EC within the Marie Curie Industry and Academia Partnerships and Pathways (IAPP) programme under grant agreement no. 251617. The authors would like to thank Evonik Industries AG for the provided datasets. MATLAB code by P. Kadlec and R. Grbić was used for the experiments.

References

[1] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, no. 1, pp. 79–87, Mar. 1991.

[2] P. Kadlec and B. Gabrys, "Local learning-based adaptive soft sensor for catalyst activation prediction," *AIChE Journal*, vol. 57, no. 5, pp. 1288–1301, May 2011.

[3] R. Grbić, D. Slišković, and P. Kadlec, "Adaptive soft sensor for online prediction and process monitoring based on a mixture of Gaussian process models," *Computers & Chemical Engineering*, vol. 58, pp. 84–97, Nov. 2013.

[4] H. Kaneko and K. Funatsu, "Adaptive soft sensor based on online support vector regression and Bayesian ensemble learning for various states in chemical plants," *Chemometrics and Intelligent Laboratory Systems*, vol. 137, pp. 57–66, Oct. 2014.

[5] S. Gomes Soares and R. Araújo, "An on-line weighted ensemble of regressor models to handle concept drifts," *Engineering Applications of Artificial Intelligence*, vol. 37, pp. 392–406, Jan. 2015.

[6] S. Gomes Soares and R. Araújo, "A dynamic and on-line ensemble regression for changing environments," *Expert Systems with Applications*, vol. 42, no. 6, pp. 2935–2948, Apr. 2015.

[7] F. Souza and R. Araújo, "Online Mixture of Univariate Linear Regression Models for Adaptive Soft Sensors," in *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, May 2014, pp. 937–945.

[8] H. Jin, X. Chen, J. Yang, and L. Wu, "Adaptive soft sensor modeling framework based on just-in-time learning and kernel partial least squares regression for nonlinear multiphase batch processes," *Computers & Chemical Engineering*, vol. 71, pp. 77–93, Dec. 2014.

[9] W. Shao, X. Tian, and P. Wang, "Local Partial Least Squares Based Online Soft Sensing Method for Multi-output Processes with Adaptive Process States Division," *Chinese Journal of Chemical Engineering*, vol. 22, no. 7, pp. 828–836, Jul. 2014.

[10] W. Ni, S. D. Brown, and R. Man, "A localized adaptive soft sensor for dynamic system modeling," *Chemical Engineering Science*, vol. 111, pp. 350–363, May 2014.

[11] J. Bates and C. Granger, "The combination of forecasts," *OR*, vol. 20, no. 4, pp. 451–468, 1969.

[12] P. Kadlec and B. Gabrys, "Soft sensor based on adaptive local learning," *Advances in Neuro-Information Processing*, no. i, 2009.

[13] P. Kadlec and B. Gabrys, "Adaptive on-line prediction soft sensing without historical data," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Jul. 2010, pp. 1–8.

[14] H. Wold, *Estimation of Principal Components and Related Models by Iterative Least squares*. New York: Academic Press, 1966, pp. 391–420.

[15] S. Joe Qin, "Recursive PLS algorithms for adaptive data modeling," *Computers & Chemical Engineering*, vol. 22, no. 4-5, pp. 503–514, Jan. 1998.

[16] H. Kaneko, T. Okada, and K. Funatsu, "Selective Use of Adaptive Soft Sensors Based on Process State," *Industrial & Engineering Chemistry Research*, vol. 53, no. 41, pp. 15 962–15 968, Oct. 2014.

[17] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*. New York, New York, USA: ACM Press, 2003, pp. 226–235.

[18] M. Scholz and R. Klinkenberg, "Boosting Classifiers for Drifting Concepts," *Intelligent Data Analysis*, vol. 11, no. 1, pp. 1–40, 2007.

[19] M. J. Procopio, J. Mulligan, and G. Grudic, "Learning Terrain Segmentation with Classifier Ensembles for Autonomous Robot Navigation in Unstructured Environments," vol. 26, no. 2, pp. 145–175, 2009.

[20] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments." *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 22, no. 10, pp. 1517–31, Oct. 2011.

[21] P. Kadlec and B. Gabrys, "Architecture for development of adaptive on-line prediction models," *Memetic Computing*, vol. 1, no. 4, pp. 241–269, Sep. 2009.

[22] E. Parzen, "On Estimation of a Probability Density Function and Mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. pp. 1065–1076, 1962.

[23] P. Geladi and B. R. Kowalski, "Partial least-squares regression: a tutorial," *Analytica Chimica Acta*, vol. 185, pp. 1–17, Jan. 1986.

[24] P. Kadlec, "On robust and adaptive soft sensors," Ph.D. dissertation, Bournemouth University, 2009.

[25] B. Mizrach, "Forecast comparison in L2," Working Papers, No. 1995-24, Department of Economics, Rutgers, The State University of New Jersey, Tech. Rep. 908, 1996.