Memetic Comp. (2009) 1:241–269
DOI 10.1007/s12293-009-0017-8

SPECIAL ISSUE - REGULAR RESEARCH PAPER

Architecture for development of adaptive on-line prediction models

Petr Kadlec · Bogdan Gabrys

Received: 15 April 2009 / Accepted: 9 September 2009 / Published online: 29 September 2009

Abstract This work presents an architecture for the development of on-line prediction models. The architecture defines unified modular environment based on three concepts from machine learning, these are: (i) ensemble methods, (ii) local learning, and (iii) meta learning. The three concepts are organised in a three layer hierarchy within the architecture. For the actual prediction making any data-driven predictive method such as artificial neural network, support vector machines, etc. can be implemented and plugged in. In addition to the predictive methods, data pre-processing methods can also be implemented as plug-ins. Models developed according to the architecture can be trained and operated in different modes. With regard to the training, the architecture supports the building of initial models based on a batch of training data, but if this data is not available the models can also be trained in incremental mode. In a scenario where correct target values are (occasionally) available during the run-time, the architecture supports life-long learning by providing several adaptation mechanisms across the three hierarchical levels. In order to demonstrate its practicality, we show how the issues of current soft sensor development and maintenance can be effectively dealt with by using the architecture as a construction plan for the development of adaptive soft sensing algorithms.

Keywords Adaptive systems · Local learning · Meta learning · Ensemble methods · Industrial applications · Soft sensors · Life-long learning

P. Kadlec (✉) · B. Gabrys
Smart Technology Research Centre,
Bournemouth University, BH12 5BB, Fern Barrow, Poole, UK
e-mail: pkadlec@bournemouth.ac.uk

1 Introduction

The traditional predictive modelling scenario dealing with industrial data deploys a static model after the training phase [1]. However, this approach often fails to provide a satisfactory solution in an environment with non-stationary data which can often be found in industrial modelling tasks. As a result, the predictive model building often requires a lot of effort that has to be dedicated to manual data pre-processing and treatment prior to the training of the model, model selection and evaluation as well as periodical tuning or re-training of the model during its run-time. Furthermore, these steps have to be performed by experts having sound knowledge of the data and the underlying process (a priori knowledge) which makes the development of such models expensive. The minimisation of the effort needed for the development and run-time treatment of the models are thus two of the main motivations for the conceptual architecture proposed in this work.

This goal is achieved by resorting to available techniques from computational intelligence and by organising these in a three level hierarchical structure. At each level of the hierarchy, there are different mechanisms for the control, diversity management and adaptation implemented. The architecture not only defines these three levels but also the links and interaction channels between them. Furthermore, in order to provide an interaction capability between the model developer and the model, the architecture supports channels for user intervention at all three levels of the hierarchy.

The three key concepts reflected within the architecture are (i) ensemble methods and diversity management, (ii) local learning, and (iii) meta learning. These techniques will be briefly reviewed in the following paragraphs.

Investigations into *ensemble methods* have formed a lively research area since the first works in this field (see e.g. [2,3])

were published. For valuable reviews of ensemble methods refer to [4,5]. The success of the ensemble methods originates in the various theoretical proofs of their effectiveness [3,6–8] as well as empirical evidence of their superior performance when compared to single predictive methods [9–14]. In terms of the Bias-Variance Decomposition (BVD) [15], ensemble methods lead to the reduction of the variance term due to the averaging effect of the combination [16]. Some authors have demonstrated that ensemble methods such as Boosting or Bagging can also achieve reduction of the bias term of the decomposition [10]. In relation to the presented architecture, Jacobs analysed the behaviour of his Mixture Of Experts (ME) architecture in terms of the BVD in [17]. In his work Jacobs has shown that the variance term of the BVD can be further split into the actual variance and covariance term which has a reducing effect on the squared error of the architecture. The same effect has been shown in terms of the *ambiguity decomposition* [6] which is also based on the BVD. The decomposition shows that a crucial aspect for the benefit of ensemble methods is a certain degree of diversity among the combined models. This aspect has been brought further by presenting different types and levels of diversity in [18]. Inspired by the presented evidence, the ensemble methods and diversity management play an important role within the architecture presented in this work.

A particular way of achieving a diverse population of models is by applying the *local learning* concept, which evolved from techniques like Radial Basis Function Networks [19] and Resource Allocating Networks [20], and was theoretically considered in [21]. In the case of local learning, the particular models are trained on partitions of the data space. This partitions of the data space are often called *receptive fields* (RF) in the literature (see e.g. [22]) which is also adopted in this work. An approach related to this work is the Locally Weighted Learning (LWL) technique [23] which was proposed as an effective way of dealing with the bias/variance dilemma [15] and negative interference [24]. Based on LWL, the Locally Weighted Projection Regression (LWPR) where additionally a local dimensionality reduction has been applied in order to deal with high dimensional data with locally low dimensional manifold has been proposed in [25]. Since the LWL and LWPR were developed in the framework of non-parametric statistics, the (local) models are based on (non-parametric) linear regression predictors.

The next concept represented within the architecture is *meta learning* (see [26] for a review of meta learning). Meta-learning can be effectively applied when targeting the goals of the architecture. So far, there is no unified definition of the term "meta learning" but in many publications it is described as a flexible way to algorithm selection and finding of mappings between task characteristics and algorithmic performance (e.g. [27–30]). The task characteristics are based on *meta attributes* which ideally should be a description of the task which is relevant for the algorithm selection and thus links the task with the performance of the considered algorithms. In this work, we adopt a slightly different definition of meta learning. In the context of the architecture we describe any method or mechanism which is applied at a level, which is above the level of the ensemble methods, as meta learning. The meta learner can also be represented as building the high-level knowledge of the model which should incrementally grow by applying the model to different tasks [31].

In order to demonstrate the usefulness of the theoretical concept, we show some practical implementations of the architecture. Based on the measurements within the processing plants, the task of the developed models is to predict some critical process variables, like for example the quality of the process product. This kind of applied models are often referred to as *on-line prediction soft sensors* [32]. In practical scenarios the soft sensor's performance can quickly deteriorate as a result of operating state changes, process input material changes, etc. Such a dynamically changing environment calls for adaptive capabilities of the soft sensors in order to avoid their frequent manual tuning and re-training [33].

The rest of this paper is organized as follows. Section 2 puts the proposed architecture in the context of so far published related work. The main contribution of this work is covered in Sect. 3 which first describes the concepts and main goals of the proposed architecture and then provides a detailed description of its modules. A case study showing a practical instance of the architecture applied to deal with on-line predictive modelling of several process industry data sets is shown in Sect. 4 and Sect. 5. Finally, the paper is concluded in Sect. 6.

2 Related work

To our best knowledge, there is no approach dealing with the outlined tasks in a way similar to the proposed architecture. Nevertheless, there are several works which are related and partially served as inspiration of the architecture proposed in this work.

A framework which is related to the proposed architecture is the Evolving Connectionist System (ECOS) discussed in [34]. The goal of ECOS is to deal with dynamic real-world problems. The ECOS system deals with the changing environment by deploying a multi-level modular structure. The modules in ECOS are feature selection part, neural network modules, higher level decision part, action modules, adaptation module and knowledge-based part. The ECOS framework was implemented in terms of a neuro-fuzzy system (EFuNN) and shown to be able to deal with changing environment, especially with changing dimensionality of the input space [35].

ECOS and another evolving model, namely Evolving Fuzzy Systems (EFS) [36], were presented in the light of a more general paradigm called Evolving Computational Intelligence Systems (ECIS) [37]. ECIS focuses on the introduction of a unified framework for knowledge-based evolving system. The primary focus of ECIS is on neuro-fuzzy models as these fulfil the ECIS requirements [37]. In contrast to this, in this work we propose a framework which is less constrained in terms of the applied predictive methods. Due to the heterogeneous environment of the different model types, there is a need for automated approaches for model selection, parametrisation and validation within the proposed framework. In contrast to the ECOS and the other EFS approaches, providing such mechanisms are one of the critical aspects of the proposed framework. Another distinguishing feature is the way of knowledge representation. While in ECOS the learnt knowledge is concentrated within the knowledge-based part, in our architecture it is distributed across the architecture in a hierarchical way ranging from the lowest level storing local information to the meta level where the global, high-level, accumulated knowledge is represented.

From the point of view of hierarchical representation, the DIVACE-II framework [18] is related to the proposed architecture. The DIVACE-II model focuses on building and evolving hybrid ensembles in order to improve generalisation capabilities of the final model. It is also based on a three-level hierarchical structure. The three levels present in DIVACE-II deal with three different types of diversity required for successful hybrid ensemble building. The framework enforces both the competition of the models at the same level and their co-operative behaviour by grouping them into ensembles at the higher levels at the same time. The DIVACE-II model could be represented as a specific instance of the proposed architecture as some of its functionality is related to the functionality of some modules of the proposed architecture (e.g. Instance Selection Management (ISM) implementing the mechanism to generate and evolve the *Training set structure* diversity).

3 The architecture

3.1 General overview

The main idea of the proposed architecture revolves around a certain degree of diversity represented by multiple competitive paths and their flexible combinations.

There are three hierarchical levels of information processing within the architecture. These are: (i) Computational path (local) level; (ii) Path combinations (PC) (intermediate) level and (iii) Meta (global) level. Figure 1 shows the hierarchy upon which the presented architecture is built. In the following the three hierarchical levels are going to be briefly outlined.

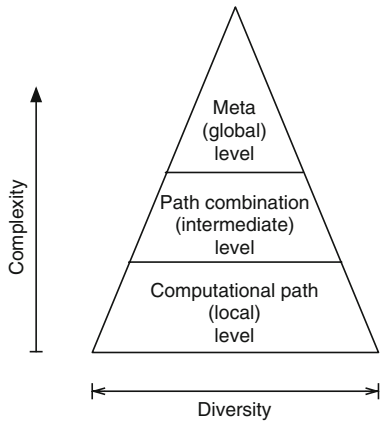

Fig. 1 Three levels of information processing within the architecture

Computational path level As shown in Fig. 1 this is the level with the lowest complexity but the largest diversity. This is achieved by maintaining a pool of diverse *computational paths* (CPs). In the terminology of this work, CPs are basic units of information processing. Each computational path consists of none, one, or more pre-processing methods (PPM) and one predictive technique which maps the (pre-processed) input data onto the output space (see Fig. 4). For further details regarding the path structure and functionality see Sect. 3.3.3. A special attention is also paid to the diversity managed among the paths as this plays an important role for the performance of the whole model. In [38], there are described four types of diversity generation mechanisms, these are:

- the initial conditions
- the modelling method
- the training set structure
- the training algorithm.

The proposed architecture allows the application of all of these diversity mechanisms at the level of the CPs as it is going to be shown later in this paper in Sect. 3.3.3. The training set structure diversity is achieved by applying local learning. Involving this kind of models also increases the flexibility of the architecture and allows the implementation of CPs which focus on partitions of the input space. This kind of localised models are further on in this work referred to as *local experts* (LEs) which is a term commonly used in the context of local learning (see e.g. [39]). Using local learning is beneficial especially in the case of industrial data where the information content of the available data is often limited and the underlying data structure is complex at the same time. Another positive effect of this approach is that data pre-processing like for example feature selection can be tuned locally which is often more effective than applying

Table 1 Main concepts within the architecture

Concept	Purpose of introduction
Adaptation	Dealing with changing environment, keeping validity of the model
Ensemble building and diversity management	Improving prediction performance, providing different prediction and adaptation mechanisms
Local learning	Dealing with sparse data, limiting the complexity of the applied models
Meta-learning	Implementing global strategy, building abstraction layer between user and model

global pre-processing. More details on local learning implementation within the architecture can be found in Sect. 3.3.5.

Path combination level This level shows an increased level of complexity and decreased diversity. This is achieved by combining individual CPs from the preceding level. At this level the paths, which are competing against each other at the path level, are forced to a collaborative behaviour. By doing this, teams of individuals which complement each other can be formed. Referring to Fig. 1, one can see that there is still a certain amount of diversity at this level as not only a single combination but a set of them is maintained at this level of the hierarchy. Managing a set of combinations dramatically increases the flexibility of the architecture. The ensemble building concept is in detail described in Sect. 3.3.4.

Meta level At the top of the complexity pyramid is the meta level. From this level the model is controlled to optimise the predictions in terms of the global performance function which is the actual function which has to be optimised by the developed model. It can be approached by (i) controlling the populations at the lower levels, e.g. by launching paths to cover unexplored parts of the input space; (ii) looking for relations between parameters of the paths and the achieved performance; (iii) adapting the combinations, i.e. teams of experts, in order to reflect the current state of the data. Another task performed at this level, which corresponds to the traditional understanding of meta learning, is the extraction and storage of knowledge from past runs of the architecture. In this way, the high-level knowledge of the architecture is built. In this type of memory the experience of past applications to different tasks is accumulated and generalised in order to make it efficiently applicable to future tasks. In accordance with the traditional meta learning approaches, this can be done by extracting meta features and linking them to the observed performance of the models. In summary, using meta learning capabilities facilitates the usage of high-level mechanisms for learning and building the high-level knowledge of the architecture. Furthermore, meta learning can take control of the parameters of the methods at the lower levels of complexity and adjust these according to the global strategy defined by the user. In this way an abstraction layer between the model operator and the lower layers of the architecture is built. The meta learning mechanisms within the architecture are in further detail explained in Sect. 3.3.6.

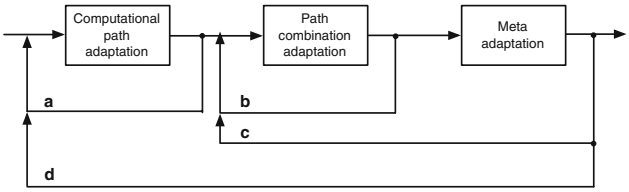

Fig. 2 Adaptation loops within the architecture arranged consistently with the three layer structure

In summary, Table 1 presents the main concepts represented within the architecture and of their main goals.

3.2 Adaptation capability of the architecture

The adaptation capability follows the hierarchy of the architecture and is present across the three different levels of model complexity. The adaptation loops and the interaction between the three levels of complexity is schematically shown in Fig. 2. The figure shows the self-adaptation capability of the local and intermediate levels (see loops a, b in Fig. 2). In contrast to this, from the meta-level there is a connection to the lower levels (loops c, d). The adaptation functionality of the architecture is in further detail addressed in Sect. 3.4.

3.3 Elements of the architecture

This section describes in detail the elements represented within the particular modules of the architecture. Figure 3 presents a detailed view of the architecture consisting of the following modules:

- Two pools of methods (PPMP, CLMP)
- Data Source module
- Paths module
- Path Combinations module

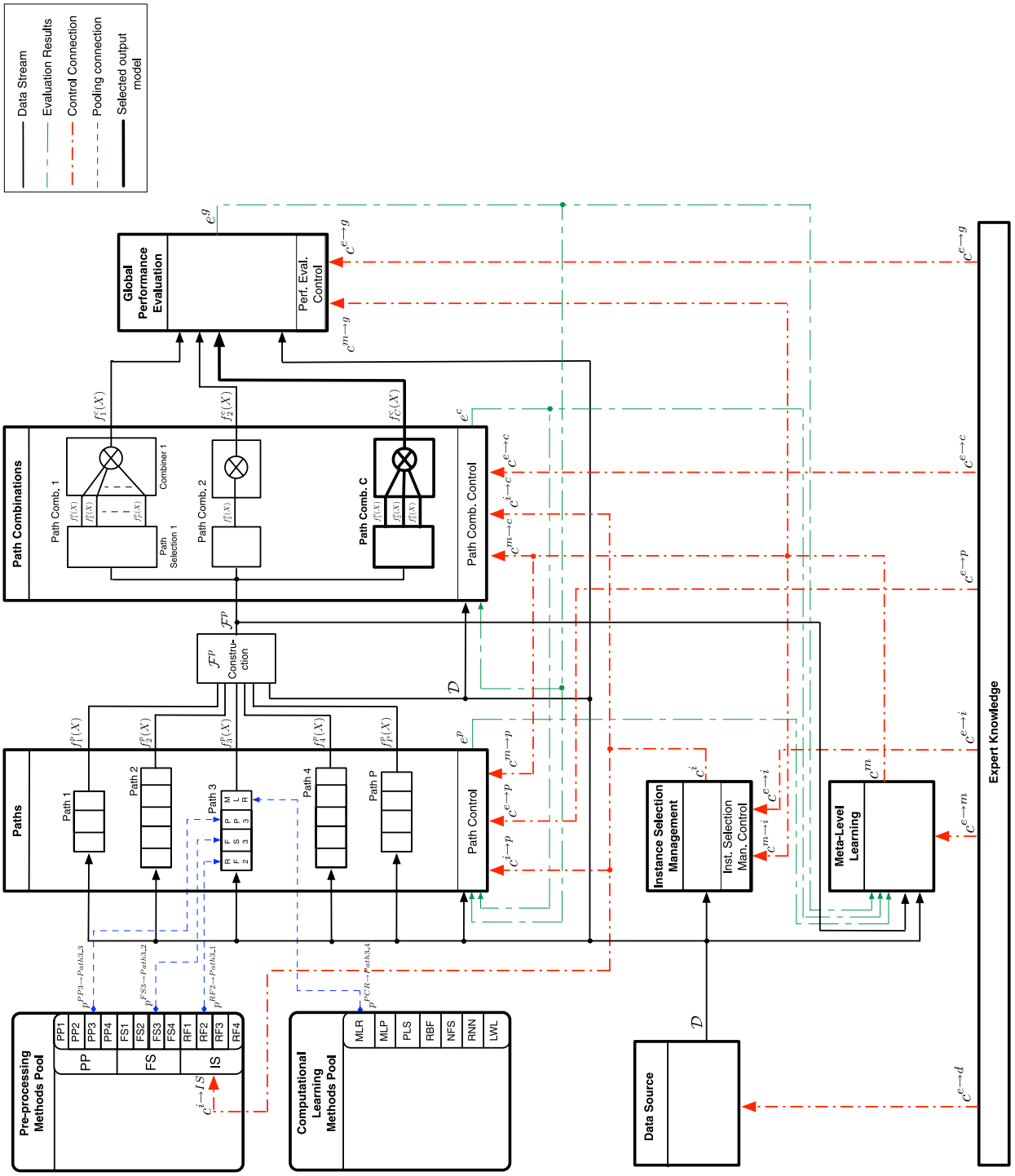

Fig. 3 The general structure of the proposed architecture

- ISM module
- Meta-Level Learning (MLL) module
- Global Performance Evaluation (GPE) module.

Furthermore, there are four different connection types:

- $\mathcal{D}$: Data Stream connection which distributes the data to the remaining modules of the architecture.
- e: Evaluation Results which is another data object carrying the results of evaluation of the predictions.
- c: Control Connections transporting control information originating from the Instance Selection (IS), the MLL or from the Expert Knowledge (EK) module.
- p: Pooling Connections indicating which of the methods from the pools are used within the Paths or PC module.

There are tree different types of the Evaluation Connections distinguishable in dependency on its origin::

- c^p: results of the CPs in the Paths module
- c^c: results of the PC
- c^g: results of the model at the Meta level.

As for the Control Connections, these can be distinguished on one hand on the module of their origin and on the other hand on the destination module. The details of the Control Connections are provided in the following descriptions of particular relevant modules.

As one can see, the architecture is rather general and provides a lot of degrees of freedom for its implementation which in turn provides the possibility to focus on a particular functionality required by the task to be solved.

3.3.1 Data source

This module acts as an interface between the physical database or any other type of data storage and the actual architecture. The data are encapsulated into special data objects $\mathcal{D}$ which are distributed across the architecture. These objects consist not only of the input data X and target data Y but also of basic statistical information S (e.g. variable distributions) which can be useful for the other parts of the architecture. The data object has the following form: $\mathcal{D} := \{(X, Y), S\}$.

Additionally, there are different regimes of the data provision possible. For the traditional learning scenario the data is distributed in batches of training, validation and test data. In a scenario dealing with an industrial modelling task, there can be a set of historical data distributed to the other modules during the training phase in batch mode followed by a stream of single data instances during the on-line phase.

The way how the data is distributed to the model is controlled using the connection $c^{e\rightarrow d}$. In the experiments described later the control signal is for example used to switch between the batch and streamed distribution of the data.

3.3.2 Method pools

The next part of the architecture are two pools of methods, namely the Pre-Processing Methods Pool (PPMP) and the Computational Learning Methods Pool (CLMP). These pools are repositories of objects which represent the data preprocessing and computational learning techniques instantiated and used within the other parts of the architecture. The role of the pools is limited to providing the incorporated objects (e.g. data normalisation in PPMP or Multi-Layer Perceptron (MLP) in CLMP) to the other parts of the architecture.

The communication with the other parts of the architecture is carried out using *pooling connections* p. These connections indicate that a certain computational path is using an instance of the object (e.g. $p^{PP3\rightarrow Path3\_3}$ which indicates that $Path3$ uses the pre-processing method $PP3$ from PPMP and places it at the third position within the path). Figure 3 shows an example where Path3 uses objects RF2, FS3 and PP3 from PPMP.

The PPMP is further split into three different sub-pools: (i) Pre-Processing (PP) methods (outlier detection, normalisation, etc.); (ii) Feature Selection (FS) methods (correlation-based feature selection, etc.); and (iii) Instance Selection (IS) methods containing instance filters in the form of RF used for the local learning functionality.

3.3.3 Computational Path

The concept of CPs is one of the essences of the proposed architecture. This module implements the bottom level of the hierarchy presented in Sect. 3.1. The task of this module is to maintain a competitive environment where the CPs are managed. The management involves the building, adapting and removing of the particular paths which is all controlled from the Path Control of the module.

The input of this module is formed by the following signals:

- $\mathcal{D}$: Data object from the Data Source
- e^c: Performance (e.g. Mean Squared Error) of the path combination objects managed in the PC module (see Sect. 3.3.4)
- e^g: Information about the performance of the whole model at the global level coming from the GPE module (see Sect. 3.3.7)
- $c^{i\rightarrow p}$: Control information from the ISM, for example notifications about the appearance of a new receptive field which triggers the deployment of a new computational path dealing with the new receptive field

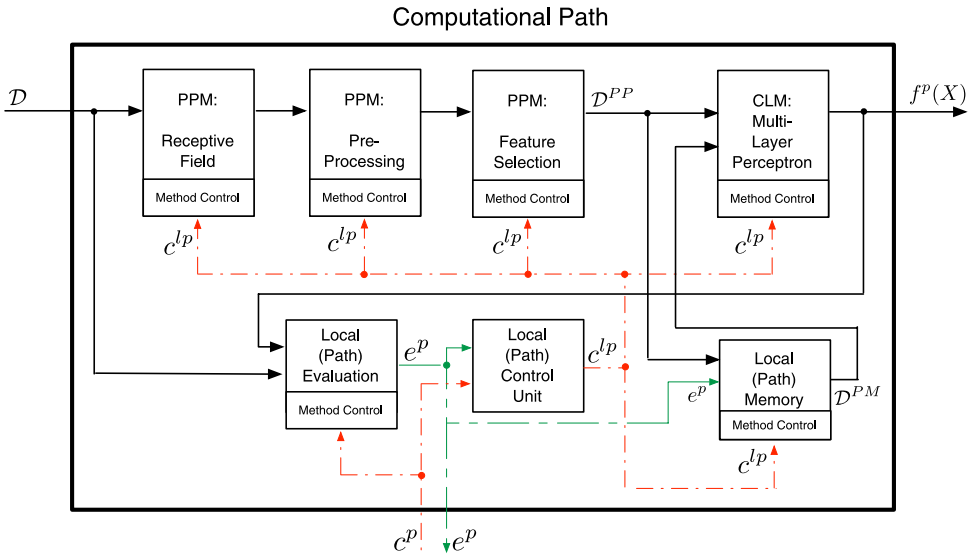

Fig. 4 The inner structure of the Computational Path object

– $c^{m\to p}$: Meta-level control information from the MLL module (see Sect. 3.3.6). An example of this kind of information is the adaptation rate for a particular computational path.
– $c^{e\to p}$: This control signal provides a possibility to involve EK into the decisions made in the Path Control (see Sect. 3.3.8). This can be for example explicit use of a pre-processing method for all paths.

Given all this information the Path Control can implement various strategies for launching, adaptation and removal of the CPs.

The actual CPs consist of several elements as shown in Fig. 4. Among these, there can be one or more pre-processing steps from the PPMP and one computational learning method from the CLMP. Furthermore, to be able to assess the local (path-level) performance there is a need for local evaluation provided by the Local (Path) Evaluation unit which calculates the performance of the path e^p according to the locally[1] defined performance measure. This is set and controlled using the c^p control signal from the Path Control. The performance data is then provided to the Local Control Unit (LCU) which sends the control information c^{lp} to the path elements and to the Local (Path) Memory. This control information can be for example a trigger for the adaptation of the path elements. The LCU receives not only the path performance data but also information from the higher levels (i.e. from MLL through Path Control) of the architecture which can additionally influence the adaptation steps. The Local (Path) Memory is responsible for storing past data instances and/or information extracted from them (e.g. variable distribution and statistics). The stored information plays an important role for the adaptation of the computational path (for details see Sect. 3.4).

The embedding of the computational path presented in Fig. 4 into the Paths module is shown in Fig. 7.

The computational path can be operated in three different modes with flexible switching between them performed from the LCU. These are:

Training mode In this mode the path is trained. This applies to the Computational Learning Method as well as to the PPM within the path (e.g. training of PCA as the dimensionality reduction method). If local learning is applied, it is important that the Receptive Field is set-up and activated for the training phase. This guaranties that the data instances are filtered and the path elements are trained using only the relevant data instances from the current receptive field.

Prediction mode For the prediction mode, where the path is required to provide predictions of the target values given the input data instances, the Receptive Field element of the path can be deactivated since the path is requested to provide predictions to all data samples independent of their receptive field origin. Depending on the scenario and the adaptation mechanism, the data instances during this phase may also be stored in the Local (Path) Memory.

Incremental mode In this mode the data arrives sequentially either in the form of single instances or groups of them. The task of the CP is to implement the adaptation strategy at the lowest level (see Fig. 2 - loop a). Once a new data instance arrives, the path makes a prediction of the target value. When the correct target value becomes available the performance of the predictions is assessed in the Local

[1] In this context local refers to the particular computational path rather than local in terms of the data space as used in local learning.

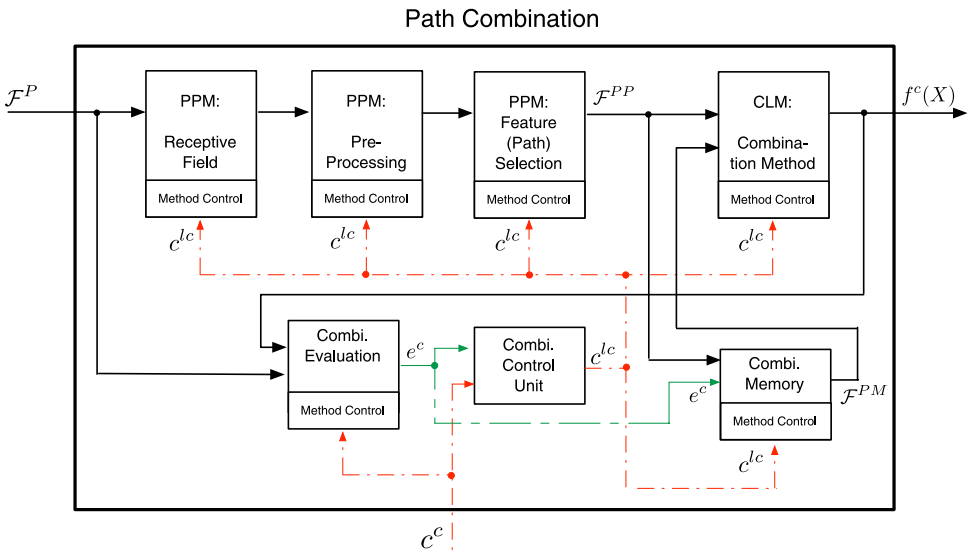

Fig. 5 The inner structure of the path combination object

Evaluation and passed to the LCU which makes a decision whether the path shall be adapted or not. In the case when a decision in favour of the adaptation is made the particular elements of the path are adapted using the implemented adaptation strategy controlled from the LCU. For example, in the case of *moving window* adaptation the samples stored in the Local Memory are retrieved and passed to the learning method which uses these data instances for retraining of the path elements (both pre-processing as well as the prediction methods).

3.3.4 Path Combinations

The concept of ensemble building is well established and accepted as a way of improving the generalisation performance of prediction models. For this reason, there is a special module, namely the Path Combination module, devoted to ensemble building techniques. This module operates at the intermediate level of complexity pyramid shown in Fig. 1. At this level the individual paths which are competing against each other at the path level are merged to groups where they cooperatively perform the target value prediction.

In the traditional combination scenario there is a set of models whose predictions are combined by a combiner [4]. In order to increase the flexibility of the architecture we go further, and provide a possibility to manage a set of independent model combinations or even their hierarchies (as shown for example in [40,41]).

The particular elements within the PC module are managed from the Path Combination Control (PCC) of the module.

Figure 3 shows that the predictions of particular CPs $f^p(X)$ are, together with the target values y from the data object $\mathcal{D}$, aggregated to a new data object $\mathcal{F}^p := \left\{(f_i^p(\mathbf{x}_j), y_j)\right\}_{i,j}$. The aim of the PC is to make a prediction of the target value in this space.

The advantage of such a representation, which is very similar to the one at the lower (i.e. path) level, is that combinations can be represented in the same way as the CPs only operating in the space formed by the data object $\mathcal{F}^p$ (compare Fig. 4 with Fig. 5). In particular, it means that for example feature selection methods can be applied to select a subset of paths (i.e. s subspace of $\mathcal{F}^p$) which have to be combined.

Furthermore, the PC can be driven in the same three modes, i.e. training, prediction and incremental learning, as it was the case for the computational path and the same approaches can be used for the implementation of three modes (e.g. moving window adaptation for the incremental mode).

The embedding of the combinations into the PC module of the architecture is also shown in Fig. 7.

3.3.5 Instance Selection Management

The ISM module is responsible for the partitioning of the input data space into locally coherent sub-spaces called RF. Having this functionality the architecture can incorporate local learning models (see e.g. [23]).

The construction of the RF can for example be done using clustering algorithms (e.g. k-means [42]) or any other data partitioning method such as the algorithm presented in Sect. 4.3.

The ISM module is managed from the ISM Control which has the following two signals as input:

– $c^{m\to i}$: Control connection from the MLL module which can provide parameters like the number of receptive fields to be managed or threshold parameters for the deployment of new RF.

– $c^{e\to i}$: EK, which can overwrite the $c^{m\to i}$ signal or manually define RF which are of advantage for the global model from the expert's point of view.

Providing the full local learning functionality involves several parts of the architecture. The ISM handles the deploying of new RF, their adaptation and removing. Furthermore, the ISM provides the information about the RF to the PPMP using the control connection c^i.

According to the three globally defined modes of operation the ISM module can be run in the following modes:

Training mode In this mode the ISM receives a batch of historical data and its task is to split the data into local partitions. The number of the partitions may be provided by the EK (i.e. by the user) if available. If not, techniques for the estimation of the optimal number of RF like cross-validation of the parameter in MLL or techniques which derive the number of local partitions intrinsically like the algorithm presented later on in Sect. 4.3 have to be applied.

Prediction mode In this mode, there is no task for the ISM and it can be deactivated.

Incremental mode As the goal of the incremental mode is to make prediction of the incoming data instances and to adapt the model at the same time, the task of the ISM is to update the RF depending on the on-line data. This task includes deploying of new RF, adaptation and the removing (pruning) of the existing ones. Examples of such approaches can be found in [43–49] or in Sect. 4.3 in this work.

3.3.6 Meta-Level Learning

The Meta-Level Learning (MLL) module of the architecture is responsible for the high-level learning, control and decision making. On the basis of the collected information from the other parts of the architecture, a picture of the global behaviour of the architecture is constructed in this module.

The input to the MLL module is formed by:

– $\mathcal{D}$: The data object
– $\mathcal{F}^p$: The paths predictions object
– The evaluation results signals from all levels across the architecture, for details see Sect. 3.3.
– $c^{e\to i}$: The control connection coming from the EK module. This signal is especially useful for setting the parameters of the applied meta-level technique.

A particular role of this module is to learn the dependency between the methods from the pools and the performance at the different levels of complexity. which is collected across different RF, datasets, initial parameter sets, etc. There are several approaches dealing with this task which can be utilised at this level. Examples of these are:

– Statistical/Information theoretical meta-attributes [29]
– Model based meta-attributes [50]
– Landmarking [28].

3.3.7 Global Performance Evaluation

Referring to the common modelling scenario, the prediction performance of the developed model is usually based on performance measures like the Mean Squared Error (MSE) which is universally applied when checking the prediction performance of a model.

The GPE module goes further and provides a flexible way to implement various performance measures. This does not necessarily need to refer to the predictive performance only. Several performance measures, which may be considered in the architecture, are discussed in [51]. Examples of such methods are: compactness, computational complexity and noise handling.

This module can also incorporate multi-objective evaluation techniques. Extending the previous example the performance measure could be a combination of the models' performance prediction and its diversity.

Since the output of the GPE module is used in the decision making at all three levels of information processing (see signal e^g being input to the Paths, PC and MLL modules), there is a default performance measure implemented (e.g. the MSE). However, for cases where another performance measure is required, there is mechanism for changing the default measure using the EK module and the control signal $c^{e\to g}$ available. Using the connection $c^{m\to g}$ the meta-level methods of the architecture can be used to switch between the available ensemble prediction and thus to select the final output of the model.

The GPE module also acts as the output module which presents the predictions results and other information about the model to the user.

3.3.8 Expert Knowledge

This part of the architecture provides an interface for the interaction between the model operator and the model. Using this interface the operator gets access to the particular modules and can manually influence the operation of the parts of the architecture. For the high-level parts of the architecture (MLL and GPE) this module is used to define the techniques operating at this level and to set their parameters. In the case of the GPE the expert has the possibility to set the performance evaluation functions and criteria if the default measure is not desired.

The particular control connections from the EK module together with possible examples of information which can be distributed using these connections are:

- $c^{e\to p}$: In this case the expert can manually control either the CPs themselves, their parameters or the Path Control where decisions about the adaptation, launching and removing of the paths are made. In a particular example the expert can, based on the available knowledge about the data, define necessary pre-processing steps and their parameters (e.g. PCA with ten principal components).
- $c^{e\to c}$: Similarly to the previous case, the expert can influence either the decision making within the PCC or suggest some PC (e.g. combine paths 1, 3 and 5 using the mean ensemble building method) which have to be included in the combinations population.
- $c^{e\to g}$: The task of this signal is the definition of the GPE function, for example balancing the MSE and correlation coefficient values of the predictions as a multi-objective performance measure.
- $c^{e\to i}$: This signal provides the expert a possibility to influence the building of new RF and the adaptation and removal of available ones. In a particular case the signal could carry the numbers of instances which according to the expert's knowledge represent a particular state of the data and have to be presented as a separate receptive field.
- $c^{e\to m}$: This connection carries the information about the method which has to be applied at the meta level and its parameters. Relating to the examples mentioned in Sect. 3.3.6, it could for example be the type of meta-features and meta-learner which have to be applied in the MLL module.

Figure 6 shows the hierarchical structure of the architecture and the interaction possibilities of the EK module at the different levels of the pyramid.

Another task controlled from the EK module can be the switching between the three operating modes of the model. From here, the control signals c^e are used for switching of the Paths, PC, ISM and MLL modules between the training, prediction and incremental modes.

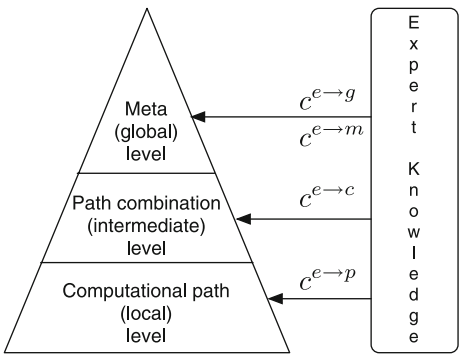

Fig. 6 Relation between the Expert Knowledge module and the three levels of information processing

3.4 Adaptation mechanisms

One of the key features of the proposed architecture is its ability to deal with dynamic environment represented by changing data. In order to use the adaptation capabilities of the architecture the particular modules have to be switched to the incremental mode. Figure 7 shows details of the adaptation mechanisms and their interaction possibilities and the next three paragraphs provide a detailed discussion of the adaptation mechanisms at the three complexity levels.

Path level adaptation The adaptation at the path level refers to the adaptation ability of the particular CPs. At this level, the paths can be adapted in two different ways. On one hand using the knowledge about their own performance only, which is also referred to as *self-adaptation*, (see loop *a* in Fig. 2 and the evaluation connection e_1^p in Fig. 7) or on the other hand by using the global control coming from the higher level of the architecture (represented by loop *d* in Fig. 2 and connection c_1^p in Fig. 7).

The self-adaptation loop consists of the feedback of the prediction which is compared to the correct target values in the Local (Path) Evaluation unit (see also Fig. 4). Given the implemented local error measure, the error e^p between the prediction $f^p(X)$ and the correct values y is calculated in the Local Evaluation unit and passed to the LCU. Another input to the LCU is the information from the Path Control (i.e. c^p) which is further linked to the two higher levels of the architecture. This kind of input is useful for the stimulation of the adaptation in a way, which is beneficial from the point of view of the global behaviour of the architecture (see loop *d* in Fig. 2). An example of such a control is the management of highly adaptive paths which follow the frequent changes of the data and at the same time to have another set of paths which are more stable and focus on the long term dynamics such slow data drifts.

An element which is important for the CPs' adaptation is the Local Memory. Dependent on the implemented adaptation strategy it might be necessary to retrieve some old data samples for the adaptation, as it is for example the case for the *moving window* technique. In a more advanced scenario, for example the one presented in Sect. 4 or the fully incremental algorithm presented in [52], it might be sufficient to store some statistics of the data instead of the data samples themselves.

The actual adaptation techniques are implemented within the Method Control of the Computational Learning Methods (CLM) and of the PPM because they may differ from method to method.

The adaptation at this level is activated from the higher level through the Path Control of the Paths module using the c^p signals which switches the path to the incremental mode.

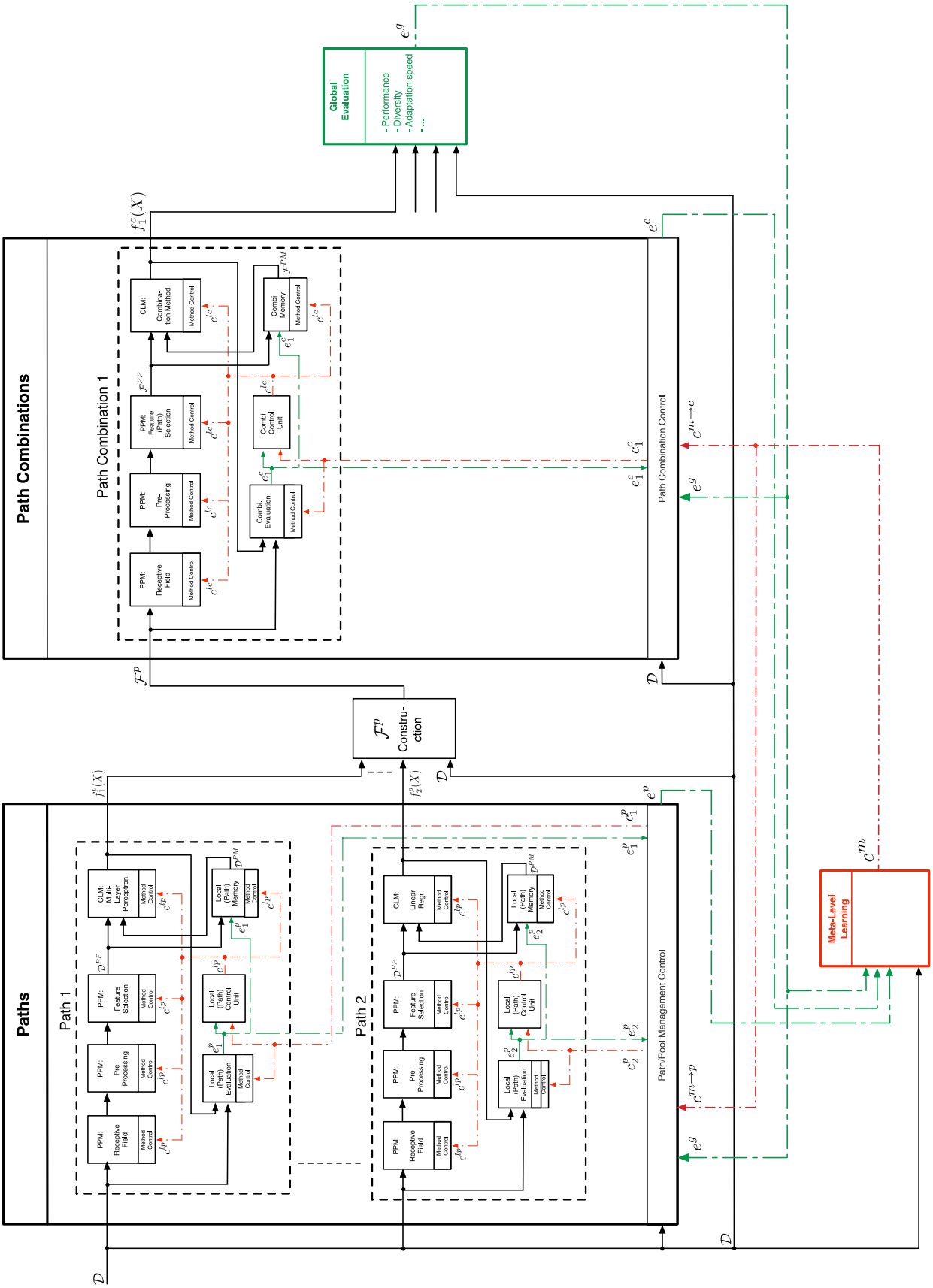

Fig. 7 Adaptation loops of the architecture

The choice of adaptation techniques which can be implemented at this place is large. In fact, one can implement any concept drift handling techniques. Examples of such techniques are:

- Moving window: [53]
- Partial memory learning: [54]
- Instance weighting and selection: [55]
- Gradual forgetting: [56].

Path combination level adaptation At this level, the PC are adapted. As mentioned in Sect. 3.3.4, the combinations can be represented in the same way as the computational path, a benefit of this representation is that similar adaptation mechanisms as in the case of the path level adaptation can be applied at this level. In the case of weighted combinations, a common adaptation technique is the modification of the combination weights. By changing the weights, one can dynamically change the contribution of the particular CPs to the final prediction. This adaptation technique has the advantage that there is no need to adapt the CPs themselves. However, the limitation of this method is that it assumes that there is at least one computational path which is able to deliver correct prediction, and as such it is not able to adapt to new states if applied on its own.

Meta level adaptation At the meta level, the adaptation has influence on the dynamic behaviour of the whole architecture. As shown in Fig. 3, the MLL module collects the information about the performance of the elements from all levels of the architecture, i.e. from the CPs e^p, PC e^c as well as the global performance e^g. This information allows to analyse not only the performance achieved across the different architecture levels but also to estimate the influence of the changes at different positions in the model.

A simple example of a strategy, which uses the meta-level adaptation capabilities, and can be implemented within the architecture is AdaBoost [8]. In this case the MLL monitors the performance of the previously deployed CPs and looks for area of the data space $\mathcal{D}$ which need improved prediction performance. Once identified, this area is deployed as a new receptive field within the IS module and a new computational path is trained. After deploying the new path, the involved combination in the Path Combination module has also to be adapted to incorporate the new paths.

4 Case study: a robust and adaptive soft sensing algorithm

In this section the theoretical concept of the architecture is applied to the development of an on-line prediction soft sensing algorithm which is able to deal with data impurities and changing environment. The presented case study also shows how combining several, rather simple, approaches for data pre-processing, predictive modelling, adaptation and meta learning inside the architecture can lead to a robust model.

The core of the model is the algorithm published by the authors in [33] which is a local learning approach. As such it not only implicitly provides several possibilities for model adaptation but also has the ability to be extended into a complex model which can be represented as an instance of the proposed architecture. The key aspects where the previous method is extended are:

- Multiple local experts per receptive field
- Different pre-processing and predictive methods for the LEs
- Local expert management using competitive, diversity and cooperative selection
- More complex data management
- Adaptation mechanisms at the combination level.

Another effect which can be observed throughout the presented instance is that the methods applied at the higher levels of the complexity pyramid are mainly non-parametric methods or methods with only few parameters while at the lower levels there can be also methods with larger number of parameters applied. The reason for this is that the management of the parameters of the methods at the lower levels is done by the MLL, whereas the parameters of the techniques at the highest level need to be controlled by the user and it is therefore of benefit to use non-parametric methods such as the k-nearest neighbour technique (see Sect. 4.6).

4.1 Data Source

This module emulates a typical scenario for the development of a model based on industrial data. It is assumed that there is a batch of historical data $\mathcal{D}^{\text{hist}}$ describing the behaviour of the process in the past available. This data is fully labelled, i.e. the target value is available for each of the data samples:

$$\mathcal{D}^{\text{hist}} = \{\mathbf{x}_i, y_i\}_{i=1}^{I}, \quad \text{with } \mathbf{x}_i = \left[x_{i,1}, \ldots, x_{i,J}\right], \tag{1}$$

where I is the number of historical samples and J the number of variables (or measurements) for each sample. The historical data is applied for the initial training of the model. Although there are no formal requirements for the size of this data, the model will in general benefit from larger historical data sets and so all available historical data should be applied at this stage.

After the training phase, the Data Source streams the on-line data $\mathcal{D}^{\text{online}}$ on sample-by-sample basis as it becomes available. This data can be either unlabelled:

$$\mathcal{D}^{\text{online}} = \{\mathbf{x}\}, \quad \text{with } \mathbf{x} = \left[x_{\cdot,1}, \ldots, x_{\cdot,J}\right], \tag{2}$$

where $x_{\cdot,1}$ refers to the first variable of the data. In such a case the model is expected to make a prediction y^p for the given sample. On the other hand, in industrial scenarios it is often the case that the correct target value is (occasionally) available during the on-line phase. In such a case the on-line data have the following format:

$$\mathcal{D} = \{\mathbf{x}, y\}, \quad \text{with } \mathbf{x} = \left[x_{\cdot,1}, \ldots, x_{\cdot,J}\right] \tag{3}$$

and the target values can be exploited for the adaptation of the model.

4.2 Method pools

The two method pools provide several methods for data pre-processing and predictive modelling. The provided techniques are methods commonly used for soft sensor development [32].

The PPMP provides the following objects for the implementation:

- Standardisation (STD): mapping the particular variable to the range (0, 1)
- Smoothing filter (SF): Smoothing the particular variables using an averaging sliding window
- Robust Principal Component Analysis (RPCA): Variable transformation [57]

The following methods are implemented in the CLMP:

- Multiple Linear Regression (MLR): linear regression [58]
- Multi-Layer Perceptron (MLP): feed-forward ANN [16]
- Least Squares Support Vector Machines (LS-SVM): kernel-based technique [59]
- Radial Basis Function Network (RBF): another variant of the feed-forward ANN [16]
- LWL: lazy-learning algorithm [60]

The aim of providing such a set of methods is achieve to certain level of diversity at the level of the architecture of the learning machine which was described in [18] as an effective way of creating diverse ensembles which in turn led to improved generalisation performance.

4.3 Instance Selection Management

This module performs the partitioning of the data into RF. In this work the same algorithm which we have developed and published earlier in [33] is used. This algorithm splits the data using a concept detection approach. The algorithm can be applied during the training phase for the splitting of the historical data as well as during the on-line phase for on-line partitioning of the data.

4.3.1 Training phase

The notion of data concept is linked to the area where a model, called *landmarker*, provides constant performance. The decrease of performance of the landmarker is interpreted as a new concept which leads to building of a new receptive field. Provided the historical data set $\mathcal{D}^{\text{hist}}$, the first step of the algorithm is training the landmarker using samples from an initial window $\mathcal{D}^{\text{init}}$ which is a subset of the historical data:

$$\mathcal{D}_i^{\text{init}} = \{X, \mathbf{y}\} = \{(\mathbf{x}_i, y_i)\}_{i=k}^{k+N^{\text{init}}}, \tag{4}$$

where k is the index of the first sample in the current receptive field and N^{init} is the length of the initial window.

Provided the initial set, the landmarker f^{lm} can be trained and the residual vector $\mathbf{r}^{\text{init}}$ of the landmarker's prediction on the training data can be calculated:

$$\mathbf{r}_i^{\text{init}} = \mathbf{y}^{\text{init}} - f^{\text{lm}}(X^{\text{init}}), \tag{5}$$

where $f^{\text{lm}}(X^{\text{init}})$ is the prediction of the landmarker.

The next step is shifting the window one step forward ($s = 1$), while keeping its size constant:

$$\mathcal{D}^{\text{shifted}} = (X^{\text{shifted}}, \mathbf{y}^{\text{shifted}}) := \{(\mathbf{x}_i, y_i)\}_{i=k+s}^{k+s+l} \tag{6}$$

$$\text{with} \quad s = 1, \tag{7}$$

and calculating the new residual values $\mathbf{r}^{\text{shifted}}(s)$ of landmarker's prediction using the shifted data window:

$$\mathbf{r}^{\text{shifted}}(s) = \mathbf{y}^{\text{shifted}} - f^{\text{lm}}(X^{\text{shifted}}). \tag{8}$$

Following this, the two residual vectors ($\mathbf{r}_i^{\text{init}}$ and $\mathbf{r}_i^{\text{shifted}}(s)$) are tested for a statistically significant difference using the t test [61]. This test was chosen because the residuals can be, ideally, assumed as normally distributed. The t test is looking for a significant difference in the mean values of the two residual vectors and so it is able to identify a significant change in the performance of the landmarker as an effect of the concept drift of the data. As long as the null hypothesis remains valid, it can be assumed that the performance of the landmarker on the data within the shifted window is comparable to the performance on the training data and thus that the data samples, within the shifted window $\mathcal{D}^{\text{shifted}}$, belong to the same concept as the samples from the initial window $\mathcal{D}^{\text{init}}$. This procedure is repeated, i.e the window is shifted, as long as the null hypothesis of the significance test remains valid:

$$s_i^{\text{final}} = \underset{s \in [1,\ldots,N-k]}{\text{argmin}} \; (t\text{test}(\mathbf{r}^{\text{init}}, \mathbf{r}^{\text{shifted}}(s)) == 1), \tag{9}$$

where N is the number of samples in the historical data set and s^{final} corresponds to the first sample for which the t test rejects the null hypothesis and thus there is a significant difference in the residuals.

Finally, the receptive field is constructed in the following way:

$$\mathcal{D}^{\mathrm{RF}} = \{(\mathbf{x}_i, y_i)\}_{i=k}^{k+l+s_i^{\mathrm{final}}-1}, \tag{10}$$

and the algorithm can move to the next receptive field by constructing new initial window. This is constructed by taking the last N^{init} samples of the previous receptive field (i.e. using the last shifted window of the previous receptive field):

$$\mathcal{D}^{\mathrm{init}} := \mathcal{D}^{\mathrm{shifted}} = \{(\mathbf{x}_i, y_i)\}_{i=k+s^{\mathrm{final}}}^{k+s^{\mathrm{final}}+N^{\mathrm{init}}}. \tag{11}$$

The procedure of RF building is graphically illustrated in Fig. 8.

After the RF are built in ISM, they are deployed as IS filters in the PPMP.

4.3.2 On-line phase

The same algorithm can be applied to build new RF during the on-line phase. The building of a new RF is triggered from MLL (see Sect. 4.6). However, during the on-line phase, since we are interested in launching a new receptive field for the latest data, the algorithm runs backwards from the latest data sample until the landmarker's performance drops.

4.4 Paths

As discussed in Sect. 3.3.3, the Paths module is responsible for the management of the actual models. The actual management of the paths is performed from the Path Control of the Paths module (see Fig. 3) and therefore all the functionality described bellow is implemented in this part of the architecture. As such, the role of the module is to deploy, manage, monitor and remove the paths based on information coming from the other parts of the architecture.

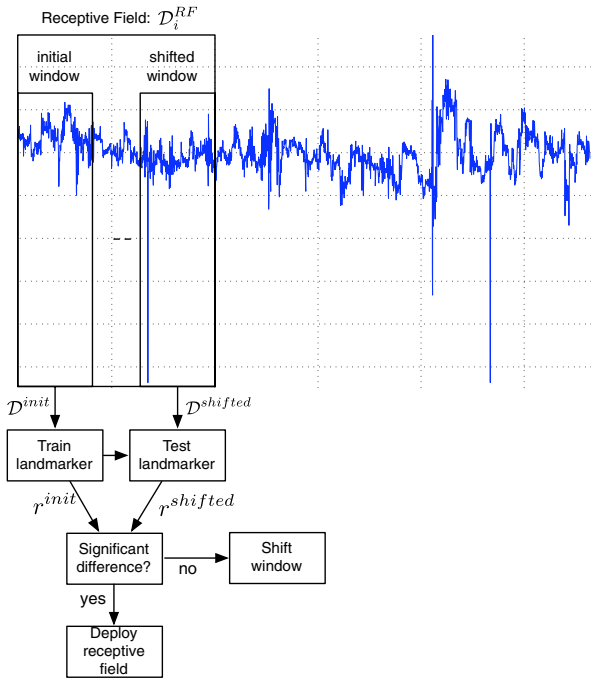

Fig. 8 Process of the receptive fields construction

4.4.1 Training phase

In the presented case, there is a set of LEs launched for each built receptive field. The initial set of models is referred to as *Local Expert Candidates* (LEC):

$$\mathcal{F}^{\mathrm{LEC}} := \left\{ f_{(m)}^{\mathrm{LEC}} \right\}_{m=1}^{N^{\mathrm{LEC}}}, \tag{12}$$

with N^{LEC} being the number of LEC.

The PPM, predictive algorithm and the parameters of these for each of the LEC are sampled from the performance distributions P which are built and stored in MLL (for details see Sect. 4.6 and Fig. 13). In order to further increase the diversity of the trained models each of the LEC is trained on random sub-sample of the receptive field data. Further more, in order to effectively exploit the limited available data, each LEC is trained using the cross-validation technique. Furthermore each LEC is trained on a random subsample of the data. This mechanism supports on one hand the diversity of the LECs and the dealing with data outliers on the other hand. Figure 9 shows the model space after the initial phase. Each of the LECs is defined by a point in three-dimensional space with the following dimensions:

- Receptive field number
- Local expert candidate number
- Cross-validation fold.

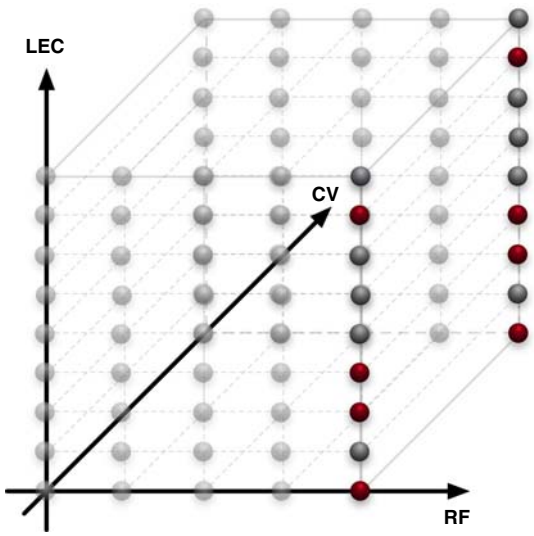

Fig. 9 The model space after the competitive selection, red models showing LECs which passed the competitive selection

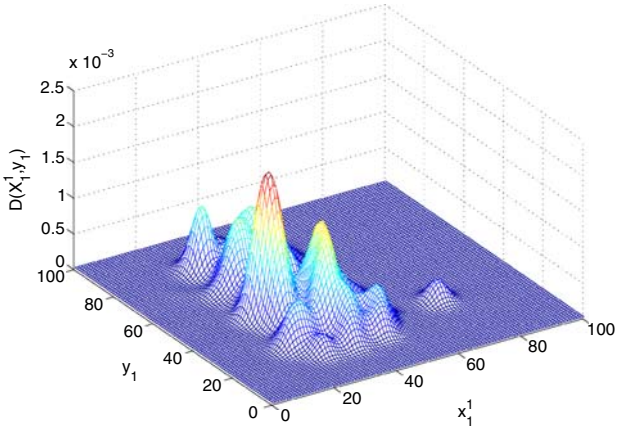

Fig. 10 Example of the two-dimensional LEC descriptor L

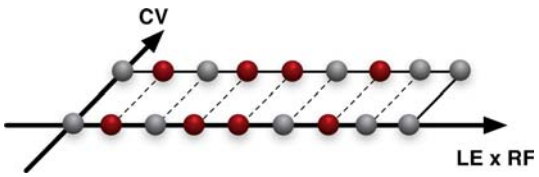

Fig. 11 The model space after the diversity selection

In order to reduce the number of paths there are two selection steps: (i) the competitive selection; and (ii) the diversity selection implemented at this stage. The aim of competitive selection is to remove LECs with low performance and is carried out inside the receptive field (see Fig. 9). The performance is measured in terms of the Local (Path) Evaluation function (see Fig. 4) and is based on the cross-validation performance estimation. The LECs, in order to pass this selection, have to fulfil the following condition:

$$\begin{aligned} &\forall_{m=1:N^{\mathrm{LEC}}} : e^{\mathrm{rel}}_{(m)} \leq e^{\mathrm{rel}}_{(min)} * t^{\mathrm{comp}}, \\ &\quad \text{with} \quad e^{(min)} = \underset{m=1:N^{\mathrm{LEC}}}{\operatorname{argmin}} \left(e_{(m)}\right) \\ &\quad \text{and} \quad e^{\mathrm{rel}}_{(m)} = \frac{e_{(m)}}{e_{(min)}} \end{aligned} \tag{13}$$

where $e^{\mathrm{rel}}_{(m)}$ is the relative error of the mth LEC, $e_{(m)}$ the absolute performance of the mth path in terms of its Local (Path) Evaluation and t^{comp} a threshold value which can be used to manipulate the size of the local expert population after the competitive selection.

After the competitive selection, the barriers between the RF are removed and the LECs are treated independently of their origin as shown in Fig. 11. For each of the LEC, which have passed the competitive selection, descriptors $L(X, \mathbf{y})$ which describe the relative performance of the LECs in the input-output space are built in the same way as shown in [33] (see Fig. 10 for an example of a descriptor). In order to be able to compare the performance of LECs from all RF there is a common validation data set $\mathcal{D}^{\mathrm{val}}$ required. In this work this data set is built by randomly sampling the historical data. The descriptors are constructed using the two-dimensional Parzen window method [62]:

$$L_{(m),j} = \frac{1}{\left\|\mathcal{D}^{\mathrm{val}}\right\|} \sum_{i \in \mathcal{D}^{\mathrm{val}}} w_{(m),i} \Phi(\mu, \Sigma) \tag{14}$$

where m refers to a local expert candidate, j to an input variable of the data, $\left\|\mathcal{D}^{\mathrm{val}}\right\|$ is the number of data points in $\mathcal{D}^{\mathrm{eval}}$, $w_{(m),i}$ is the weight of the contribution of the point (for more details see bellow), $\Phi(\mu, \Sigma)$ is a two-dimensional Gaussian kernel function with mean value at the position defined by $\mu := \left[x_{i,j}, y_i\right]$ and variance matrix Σ (a diagonal 2×2 matrix with the kernel width σ at the diagonal positions). The kernel width σ is a parameter of the algorithm which defines the size of the neighbourhood influenced by each data point. For simplicity we keep the variance in both dimensions equal but the approach can be easily extended to a more general case with different kernel widths along different dimensions.

The weights $w_{(m),i}$ for the construction of the descriptors (see Eq. 14) are proportional to the prediction error of the LECs:

$$w_{(m),i} = \exp(-e_{(m),i}) = \exp(-(f^{\mathrm{LEC}}_{(m)} - y_i)^2) \tag{15}$$

Weighting the contribution of each sample by the prediction performance of the corresponding LEs assures that the descriptors model the LEs' area of expertise in the input-output space and as such can be later sampled to estimate the local expert's performance given the input data and its prediction.

It should be also noted that the descriptors L are built in the data space which results from the data pre-processing of each computational path (i.e. in Fig. 4 just before processing the data in CLM). In combination with the PCA or any other feature selection method this leads to significantly reduced computational requirements. The descriptors are stored in the Local (Path) Memory of the computational path (see Fig. 4). Since there is one descriptor built for each variable of the data space the final descriptor $\mathcal{L}_{(m)}$ for the mth local expert candidate has the following form:

$$\mathcal{L}_{(m)} = \left\{L_{(m),j}\right\}_{j=1}^{J} \tag{16}$$

where J is the number of variables.

Coming back to the diversity selection, the LEC descriptors can be used to obtain a weight vector $\omega_{(m)}$ for the particular LEC by sampling them at positions defined on one hand by the input data of the evaluation data $\mathbf{x} \in \mathcal{D}^{\mathrm{eval}}$ and the predictions of LECs ($f^{\mathrm{LEC}}(\mathbf{x})$) on the other hand (for details of the calculation of the weights vector see Eq. 23 and 24). Having the weight vectors $\omega_{(m)}$ for all LECs, their correlation is calculated and those having the strongest correlation patterns are removed. In our implementation, we iteratively remove

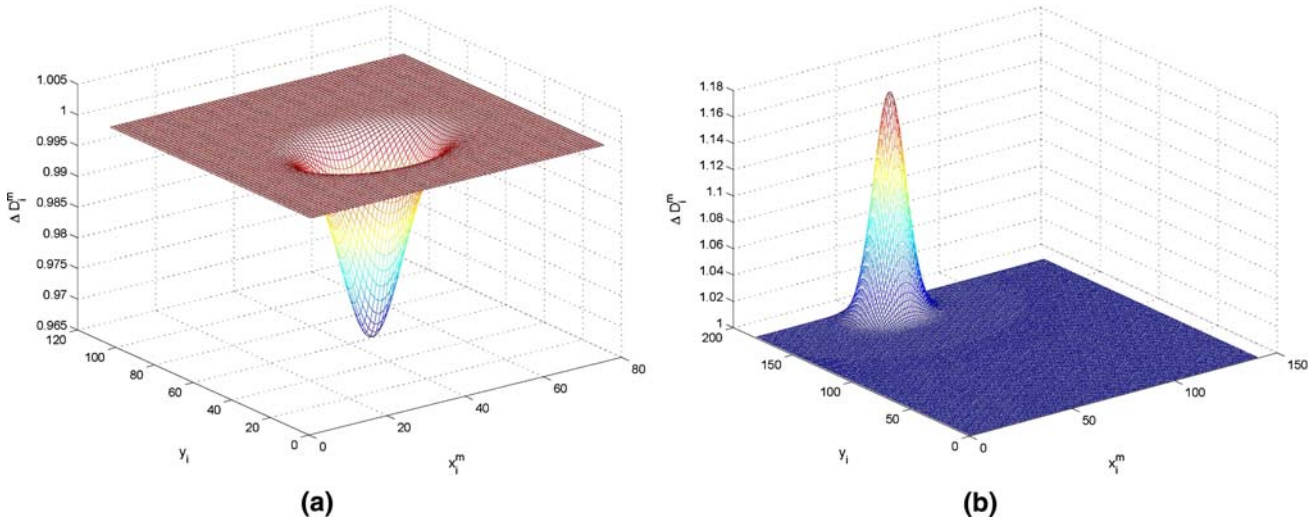

Fig. 12 Adaptation mask for the modification of the neighbourhood of the current sample point

LECs having the highest correlation coefficient with all other LECs which is repeated as long as there is no LEC with correlation coefficient higher than an user-defined threshold t^{corr}. This threshold can be manipulated in order to control the local expert population size.

After the diversity selection the remaining LEC are called LEs $\mathcal{F}^{\text{LE}}$ as they will be kept (unless they are pruned later) in the Paths module and used as part of the final model.

4.4.2 On-line phase

During the on-line phase the Paths module is passing the incoming data through the LEs, obtaining their predictions which are provided to the other modules of the architecture. In an adaptive model scenario with available target values during the on-line phase, there are also several possibilities for model adaptation provided.

The simplest way of adaptation, to which we refer as *Type 1 adaptation*, is the adaptation of the local expert descriptors $\mathcal{L}_{(m)}$ stored in the Local (Path) Memory of each computational path. Provided a labelled on-line data sample:

$$\mathcal{D}^{\text{online}} := \{\mathbf{x}, y\}, \tag{17}$$

the prediction error $e_{(m)}$ of all of the LE can be calculated using the Local (Path) Evaluation (e.g. using the squared error bellow):

$$e_{(m)}(\mathbf{x}) = (f_i^{\text{LE}}(\mathbf{x}) - y)^2. \tag{18}$$

The error is further on mapped onto a performance index $u_{(m)}$:

$$u_{(m)} = \exp\left(-\frac{e_{(m)} - \text{med}(\mathbf{e})}{\text{med}(\mathbf{e})\log(2)}\right), \tag{19}$$
$$\text{with } \mathbf{e} = \left[e_{(1)}, \ldots, e_{(N^{\text{LE}})}\right]$$

where med($\mathbf{e}$) is the median squared error across the LEs and N^{LE} the number of LEs. The above mapping transforms the prediction error in such a way that the best performing local expert receives a weight equal to 1 and the weights of the remaining LEs decay exponentially with the increasing error, whereas the median error is mapped to the value 0.5. This mapping function leads to a decrease in the neighbourhood of the current sample within the local expert descriptors (see Fig. 12a for the adaptation mask in such a case) of LEs providing weak performance for the current on-line data sample. Contrary to this, descriptors of LEs whose performance is better than the average are increased in the neighbourhood of the current sample (see Fig. 12b for an example of such an adaptation mask).

The Paths module is also part of another more complex adaptation scenario, which involves the deployment of new RF during the on-line phase (see Type 2 adaptation in Sect. 4.6). The tasks performed during the Type 2 adaptation inside the Paths module are equivalent to those performed at the training phase of the modelling. The Path Control is provided with past samples and performs the following actions: (i) building and training of the LECs; (ii) competitive selection; and (iii) diversity selection. The only difference to the training phase is that now the newly deployed LEC are competing with the LEs, which were built earlier, during the diversity selection. In other words, for the diversity selection it does not matter if the path was deployed as part of the latest receptive field or if it was already present in the

system. Additionally to improving the diversity of the paths, this mechanism is also an effective pruning approach as one can maintain a stable number of the LEs inside the Paths module.

4.5 Path Combinations

The PC module provides the functionality for ensemble methods. More than that, it also has the ability to manage a set of combinations and to apply similar mechanisms as for the CPs.

4.5.1 Training phase

The task of the PCC during the training phase is to perform the third selection step, namely the *co-operative selection*. The goal of this selection is to build teams of LEs which co-operatively achieve high performance. In contrast to the previous two selection steps, the LEs which are not selected for any ensemble are kept in the Paths module since they can be useful at a later stage. The ensemble building is implemented in the PCC and proceeds as follows: Based on the evaluation data set $\mathcal{D}^{\text{eval}}$ used earlier, the best performing local expert ($f^{\text{LE}}_{(min)}$) in terms of its predictive performance is selected:

$$f^{\text{LE}}_{(min)} = \underset{m=1:N^{\text{LE}}}{\text{argmin}} \left(\sum_i (f^{\text{LE}}_{(m)}(\mathbf{x}_i) - y_i)^2 \right)$$
$$\text{with } \mathcal{D}^{\text{eval}} = \{\mathbf{x}_i, y_i\}_{i=1}^{N^{\text{eval}}}. \tag{20}$$

After finding the best performing LE, it is added to the first ensemble $\mathcal{F}^{\text{ens}}_1$:

$$\mathcal{F}^{\text{ens}}_1 = \left\{ f^{\text{LE}}_{\text{min}} \right\}. \tag{21}$$

As next, the LEs, which most improve the ensembles prediction, are iteratively added to the ensemble:

$$f^{\text{LE}}_{(min)} = \underset{\substack{m=1:N^{\text{LE}} \\ f^{\text{LE}}_{(m)} \notin \mathcal{F}^{\text{ens}}_1}}{\text{argmin}} \left(\sum_i \left(c\left(\mathcal{F}^{\text{ens}}_{(\text{temp})}(\mathbf{x}_i) \right) - y_i \right)^2 \right)$$
$$\text{with } \mathcal{F}^{\text{ens}}_{\text{temp}} = \left\{ \mathcal{F}^{\text{ens}}_1, f^{\text{LE}}_{(i)} \right\} \tag{22}$$
$$\mathcal{F}^{\text{ens}}_1 = \left\{ \mathcal{F}^{\text{ens}}_1, f^{\text{LE}}_{(\text{min})} \right\},$$

where $c(\mathcal{F}^{\text{ens}}_{\text{temp}})$ is a combination function which forms the combined prediction of the ensemble (for detail on the combination function see bellow). The procedure described in Eq. 22 is repeated as long as adding a new member to the ensemble improves the performance.

In a non-adaptive scenario, there is only one combination built and managed in the PC module. However, in a scenario with available feedback information during the on-line phase, it is useful to create several PC and switch between them based on their performance. In this particular implementation, the different combinations are varied by changing the initial Local Expert, which leads to different PC. The number of managed PC N^{PC} is a parameter which can be defined externally by the user.

Within the architecture the discussed method is implemented within the PCC which launches the PC. The Local Expert selection for the combinations is achieved by filtering the LEs in the Feature (Path) Selection part of the path combination (see Fig. 5).

4.5.2 On-line phase

In order to obtain the prediction $\mathcal{F}^{\text{ens}}(\mathbf{x})$ from the ensembles given the on-line input data $\mathbf{x}$, the predictions of the particular LEs need to be combined. The combination is based on the approach described in [33]. The combination approach is a weighted sum of the predictions of LEs which are part of the ensemble:

$$\mathcal{F}^{\text{ens}} = \sum_{m=1}^{N^{\text{ens}}} v_{(m)}(\mathbf{x}, f^{\text{LE}}_{(m)}) f^{\text{LE}}_{(m)}(\mathbf{x}), \tag{23}$$

where N^{ens} is the number of LEs in the ensemble, $f^{\text{LE}}_{(m)}$ are the particular ensemble members, $f^{\text{LE}}_{(m)}(\mathbf{x})$ their predictions given the on-line data sample and $v_{(m)}$ the combination weight of the mth local expert's prediction. In the case of this instance, the weights are read from the local expert descriptors $\mathcal{L}_{(m)}$. Since the descriptors store maps of the LEs performance in the input-output space, they can be sampled to get the predicted performance of the LEs for the given on-line data sample. This can be expressed as the posterior probability of the mth local expert given the test sample $\mathbf{x}$ and the Local Expert's prediction $f^{\text{LE}}_{(m)}(\mathbf{x})$:

$$v_{(m)}(\mathbf{x}, f^{\text{LE}}_{(m)}) = p(m|\mathbf{x}, f^{\text{LE}}_{(m)}) = \frac{p(\mathbf{x}, f^{\text{LE}}_{(m)}|m)\, p(m)}{\sum_m p(\mathbf{x}, f^{\text{LE}}_{(m)}|m)}, \tag{24}$$

where $p(m)$ is the a priori probability of the mth Local Expert (in our implementation equal for all LEs but in general it can be used to prioritise between them), $\sum_m p(\mathbf{x}, f^{\text{LE}}_{(m)}|m)$ is a normalisation factor and $p(\mathbf{x}, f^{\text{LE}}_{(m)}|m)$ the likelihood of $\mathbf{x}$ and the local expert which can be calculated by reading the descriptor of the mth Local expert $\mathcal{L}_{(m)}$:

$$p(\mathbf{x}, f^{\text{LE}}_{(m)}|m) = \prod_{j=1}^{J} p(x_{\cdot,j}, f^{\text{LE}}_{(m)}|m)$$
$$= \prod_{j=1}^{J} L_{(m),j}(x_{\cdot,j}, f^{\text{LE}}_{(m)}(\mathbf{x})). \tag{25}$$

Equation 25 shows that the descriptors $\mathcal{L}_{(m)}$ are sampled at positions which are given by the scalar value $x_{\cdot,j}$ of the jth variable of the sample point $\mathbf{x}$ and at the position of the

predicted output $f_{(m)}^{\mathrm{LE}}(\mathbf{x})$ of the mth local expert. Sampling the descriptors at the positions of the predicted outputs may be potentially ineffective because the predicted value does not necessarily need to be similar to the correct target value. However, as the correct target values are not available during the on-line phase at the time of the prediction this is the only way how to read the values from the descriptors. Furthermore, the rationale for this approach is that the local expert is likely to make correct prediction if it generates a prediction which conforms with an area which was occupied by a large number of accurate predictions during the training phase.

There are two adaptation mechanisms related to the PC called *Type 3 adaptation* and *Type 4 adaptation*, respectively. In case of Type 3 adaptation, the MLL module switches between the available combinations and thus selects current output model. In this instance, the MLL checks every ten samples which of the combinations performed best over the last ten samples and selects the output model accordingly. As for the Type 4 adaptation, in this case new PC are built. This adaptation is also controlled from the MLL module and is triggered if the performance of the model drops bellow a certain threshold value. When triggered, the adaptation mechanisms perform the same action (with different evaluation data though) as it was done during the training phase (see Sect. 4.5.1) and builds new combination schemes. The details of this adaptation type are provided in Sect. 4.6.2.

4.6 Meta-Level Learning

The major task of this module is to manage the underlying modules by monitoring the performance of the full model and taking appropriate actions if necessary. This is on one hand achieved by controlling some critical parameters, such as the correlation coefficient threshold t^{corr} discussed in Sect. 4.4, and on the other hand by controlling the adaptation mechanisms.

4.6.1 Training phase

During the training phase, an important role of the MLL is to build and store performance descriptors P of the RF (see Fig. 13 for an example of such a map). The goal of these descriptors is to build a map of the performances of different pre-processing and predictive techniques. The descriptors are built by randomly sampling from the space of all possible parameters of the pre-processing and predictive techniques which are available in the two method pools of the architecture (i.e PPMP and CLMP, respectively). The parameter ranges of the techniques (e.g. the number of hidden units range of an MLP) to be tested have to be defined by the user through the EK module. Once built, the performance descriptors can be sampled in order to obtain the configurations of the LECs as described in Sect. 4.4.

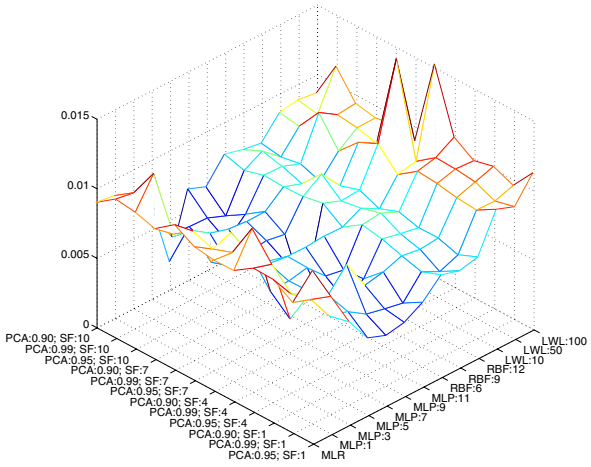

Fig. 13 Performance distribution of various methods and their parameters

Additionally to the receptive field performance descriptors, meta-descriptors M describing the RF are also calculated in the MLL. The meta-descriptors have the following format:

$$M_r = \left[\left\| \mathcal{D}_r^{\mathrm{RF}} \right\|, J_r, \mathrm{perf}(f_r^{\mathrm{LM}}) \right], \tag{26}$$

where $\left\| \mathcal{D}_r^{\mathrm{RF}} \right\|$ is the number of data samples in the rth receptive field, J_r is the number of variables of the receptive field data after the local pre-processing using the PCA and $\mathrm{perf}(f^{\mathrm{LM}})$ is the quadratic error of the prediction of the landmarker associated with the receptive field. The purpose of the meta-descriptor is to be able to find similar RF during the on-line phase and thus to inherit their performance descriptors which will be useful to avoid the need to calculate these as it is related to a high computational demand (for more details see the Type 4 adaptation in Sect. 4.6.2).

4.6.2 On-line phase

The most critical role of the MLL module is to control the adaptation of the model. The four available adaptation possibilities were already discussed through this work. In what follows is a discussion of the role of the MLL with regard to the adaptation techniques.

For the Type 1 adaptation the MLL does not play any role, as this adaptation is controlled and performed by CPs themselves. Referring to Fig. 2, this adaptation mechanism corresponds to the loop b as adapting the LE descriptors changes the combination weights of the PC.

The Type 2 adaptation is more complex and requires an attention of the MLL. This adaptation is triggered if the current performance of the full model drops bellow a certain threshold t^{ada2} (defined by the user through the EK module) and its goal is to build new RF during the on-line phase. In order to achieve this, firstly the receptive field detection

in ISM needs to be performed (see Sect. 4.3.2). Provided the receptive field, the performance descriptor P of the new receptive field needs to be calculated. Due to its high computational costs the calculation of the performance descriptor P of the new receptive field is avoided by building the meta-descriptor of the new receptive field, finding the most similar stored receptive field (i.e. using the k-nearest neighbour technique in the meta-descriptor space) and inheriting the performance descriptor of the closest receptive field. Once having the performance descriptor, the LECs can be built and reduced to LEs using the competitive and diversity selection as discussed in Sect. 4.4.1. The Type 2 adaptation is completed by building new PC in the PC module according to Sect. 4.5.1. The above discussion shows that performing this type of adaptation is quite a complex process involving several parts of the architecture. The Type 2 adaptation is part of the loop d in Fig. 2.

In contrast to the Type 2 adaptation, the Type 3 adaptation is rather simple. The goal of this adaptation method is to select the currently best performing path combination managed in the PC module. As such, it is started periodically. In the case of this instance it is performed every ten data data points. It involves checking the performance of the PC over the latest ten samples in the MLL module and selecting the combination with best performance as the current output model in the GPE module.

The Type 4 adaptation involves building new combinations in the PC module. This adaptation is triggered in a similar way as the Type 2 adaptation, namely by monitoring the performance of the full model. The threshold for the adaptation t^{ada4} is in general lower than the one for the Type 2 adaptation as it is less complex and can be used as a first attempt to improve the decreasing performance of the full model. The procedure for this adaptation involves building a data set $\mathcal{D}^{\text{eval}}$ which consists of the data samples between the last Type 2 or Type 4 adaptation, dependent on which occurred later, and the current on-line data samples. This data is provided to the PCC which performs the building of the new ensembles according to Sect. 4.5.1. The Type 3 and 4 adaptation mechanisms are represented as loop c in Fig. 2.

Apart from the adaptation control, the MLL is also responsible for the control of the population size of the LEs in the Paths module. The number of CPs is controlled by changing the t^{corr} parameter of the diversity selection. In the presented instance the target number of LEs is 50 and depending on the actual number of the LEs the parameter is either increased or decreased.

4.7 Performance Evaluation

Within this module, there are the MSE and correlation coefficient measures implemented. These two measures are used to assess the performance of the output model which is on one hand distributed throughout the architecture and provided to the model operator on the other hand.

4.8 Expert Knowledge

As discussed throughout this paper, the Expert Knowledge (EK) module's main purpose is to provide an interaction channel between the model developer, in this case soft sensor developer, and the actual model. In the presented instance we restrict the expert interaction to the control of the critical parameters of the model.

The EK module also provides default values for all the parameters (see Table 2). This should make the model development even more straightforward and effective while still providing the flexibility to change some of the parameters or their ranges if required. This fact is demonstrated in Sect. 5.2, where three different soft sensors are developed using the same input parameters for all three models.

5 Experiments

The set-up and parametrisation of the architecture instances applied to the data sets is performed according to Sect. 4. The model applied to all three data sets is the same without any data set specific manual parameter optimisation. In the experiments, there are two different scenarios, namely (i) a non-adaptive case and (ii) an adaptive case. In the first case it is assumed that no target data is available during the on-line phase and thus the model can not adapt and after the training phase remains unchanged. In the latter case we assume that the target values for all on-line data samples becomes available after making the prediction for the current on-line data sample. Such a scenario allows to assess the adaptation ability of the models.

5.1 Data sets

The three data sets used for the experiments were kindly provided by Evonik Degussa GmbH. The following paragraphs briefly outline the data sets.

Thermal oxidiser This on-line prediction data set deals with the prediction of exhaust gas concentration of an industrial process. The task is to predict the concentrations of NO_x in the exhaust gases. The data set consists of 36 input features (i.e. hard sensor measurements). The input features are physical values like concentrations, flows, pressures and temperatures measured during the operation of the plant. The data set consists of 2,053 samples of raw unprocessed data as it was recorded by the process information and measurement system. As a result of this, many of the input variables

Table 2 Summary of the parameters of this instantiation of the architecture (for abbreviation see Sect. 4.4)

Module	Parameter	Comments
Data source	–	–
Instance Selection Management	N^{init}	The size of the initial window
		Default: set to 3*J (number of variables of the data)
	f^{LM}	The landmarker computational path
		Default: PCA (covering 95% of the variance + MLR)
	N^{CV}	Number of cross-validation folds
		Default: 2
Pre-Processing Methods Pool	SF: N^{smooth}	Length of the window of a smoothing filter
		Default: [1, 4, 7, 10]
	RPCA: cv	Covered variance of the RPCA [57]
		Default: [0.92, 0.97, 0.99]
Computational Learning Methods Pool	SVM: k	Kernel size of the Gaussian kernel of the SVM
		Default: [0.1, 1, 10, 100, 1,000]
	MLP: N^{hidden}	Number of hidden units of an MLP
		Default: [1, 3, 5, 7, 9, 11]
	RBF: N^{hidden}	Number of hidden units of an RBF
		Default: [6, 9, 12]
Paths	LWL: $N^{\text{neighbour}}$	Neighbourhood size of the LWL
		Default: [10, 50, 100]
	N^{LEC}	Number of Local Expert Candidates to be launched
		Default: 100
	t^{comp}	Threshold for the competitive selection
		Default: 2.0
	t^{div}	Threshold for the diversity selection, controlled from MLL
		Default: 0.8
	σ	Kernel size of the Gaussian function for LE descriptors
		Default: 0.001
Path Combinations	N^{PC}	Number of managed path combinations
		Default: 10
Meta-Level Learning	t^{ada2}	Threshold for triggering the Type 2 adaptation (absolute error)
		Default: 0.1
	t^{ada4}	Threshold for triggering the Type 4 adaptation (absolute error)
		Default: 0.08
	N^{ada3}	Number of samples between the Type 3 adaptation
		Default: 10

present common issues of industrial data like measurement noise (high level of noise can be found in 20% of the data features) and data outliers which severelly affect approximatelly half of the features.

Industrial drier The target values of this data set are laboratory measurements of the residual humidity of the process product. The data set has 19 input features, most of them being temperatures, pressures and humidities measured within the processing plant. The data set consists of 1219 data samples covering almost seven months of the operation of the process. It consists of raw unprocessed data as it was recorded by the process information and measurement system. The main issue of this data set is the high noise level which affects all of the data features as well as outliers which can also be found in most of the features. Missing values are present in 16% of the features.

Catalyst activation This data set was used for the NiSIS 2006 competition.[2] The task was to predict the activity of a

[2] http://www.nisis.risk-technologies.com/filedown.aspx?file=125.

catalyst in a multi-tube reactor. The input data are 14 sensor measurements like flows, concentrations and temperatures, from a real process together with one feature describing the timestamps of the measurements. The target variable is modelled activity of the catalyst in the reactor. The data set covers one year of operation of the process plant. In particular this data set requires strong adaptive mechanisms since the test data are unrelated to the training data. Apart from this problem the data shows also very high co-linearity of the features and high amount of outliers which can be found in as many as 80% of the features.

5.2 Experimental Results

In this section the model which is developed according to the presented architecture is called Robust On-line Soft Sensor (ROSS). In the non-adaptive scenario this model is benchmarked with a model based on MLP and PCA data pre-processing, which is a common way of developing a soft sensor as well as with a non-incremental version of the LWPR method. In the adaptive case we compare ROSS using different adaptation methods with an incremental version of the LWPR. The parameters of the ROSS model were set according to Table 2 and were kept constant for all three soft sensors (with an exception of the t^{ada2} and t^{ada4} parameters in the case of the industrial drier soft sensor, for more details see Sect. 5.2.3). In order to analyse the impact of the four adaptation types, there are three different adaptation experiments, each using a different set of adaptation types, performed and analysed. The different experiments are summarised in Table 3.

The parameters for the remaining two methods (MLP and LWPR) were optimised using an exhaustive search in the space of their parameters. This approach was chosen in order to guarantee an optimal performance of the benchmark methods. The selected parameters are listed in the corresponding experiments.

5.2.1 Thermal oxidiser

Non-adaptive soft sensors The first model analysed in the non-adaptive scenario is the MLP with PCA pre-processing. For the pre-processing 12 principal components were selected as optimal parameter of the PCA. Although for the MLP there are not many parameters to be optimised, the model selection for this soft sensor is rather difficult as demonstrated in Fig. 14. This figure shows the MSE and correlation coefficient of the predictions of the MLP models for the on-line data. Each of the box-plots represents an MLP model with a given number of hidden units and is based on ten random initialisations of the MLP. As one can see the distribution of the performance is wide and it is virtually impossible to select an optimal number of hidden units.

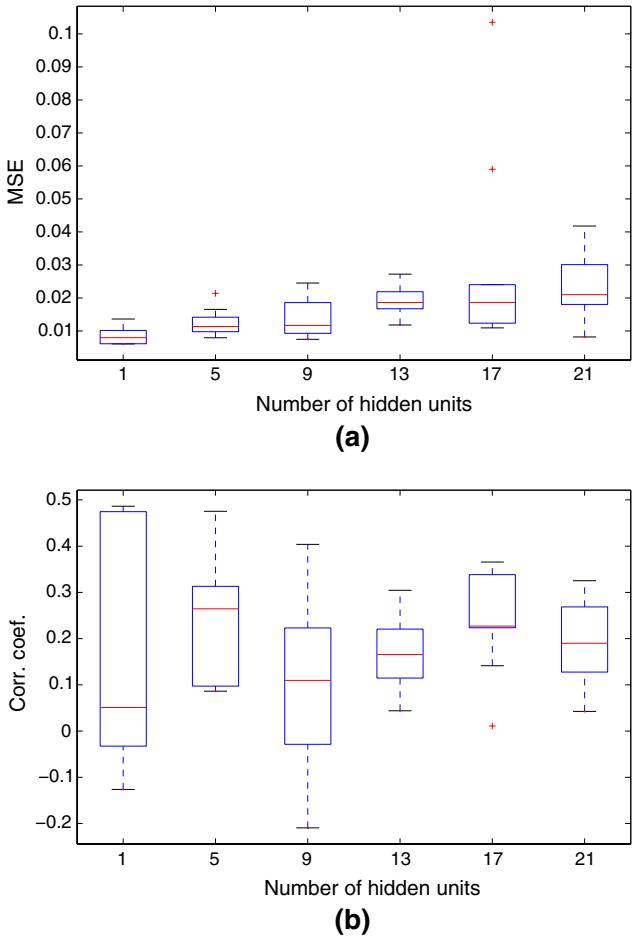

Fig. 14 Performance of the MLP soft sensors for the thermal oxidiser data. **a** MSE, **b** correlation coefficient

Nevertheless for the following comparison an MLP with one hidden neuron has been selected as the optimal model.

For the second benchmark model, namely the LWPR, the optimal parameters were identified using greedy search based on parameter cross-validation in the space of reasonable parameter values (as proposed in [63]). The input parameters found as optimal are:

- 'init-D' = 4
- 'init-alpha' = 100
- 'diag-only' = 0
- 'w-gen' = 0.9
- 'w-prune' = 0.9
- 'penalty' = 1e−7
- 'meta' = 0
- 'update-D' = 0
- 'kernel' = 'Gaussian'

The parameters used for the ROSS are listed in Table 2. During the training phase there were only two RF built which

Table 3 Summary of the three adaptation cases

	Type 1 adaptation	Type 2 adaptation	Type 3 adaptation	Type 4 adaptation
Adaptation case 1	1	0	0	0
Adaptation case 2	1	1	0	0
Adaptation case 3	1	1	1	1

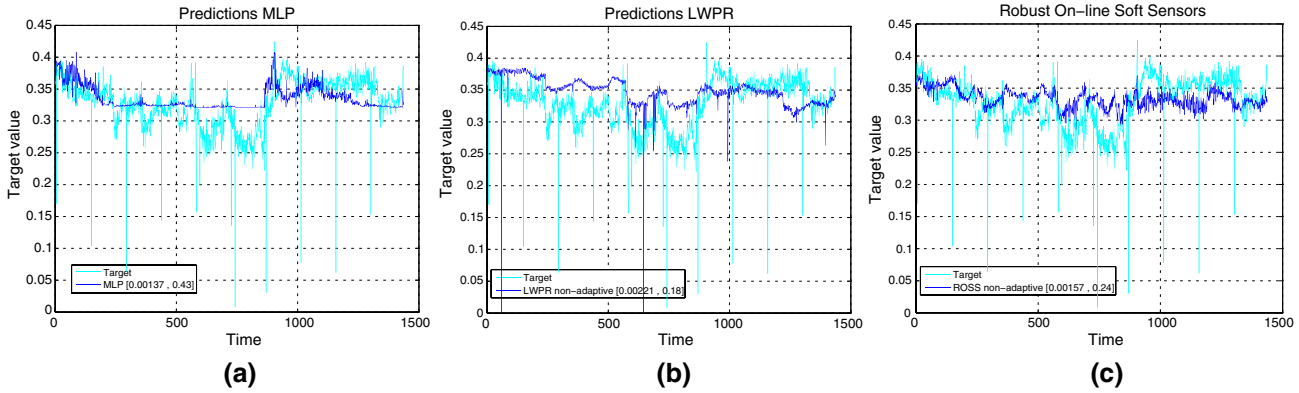

Fig. 15 Predictions of the three non-adaptive models for the thermal oxidiser data. **a** MLP, **b** LWPR, **c** ROSS

results into the training of 200 LEC. From these, there were 22 LEs (i.e. $N^{\mathrm{LE}} = 22$) left after the competitive and diversity selection. Among these the distribution of the predictive methods was the following:

- MLR: 2
- LS-SVM: 10
- MLP: 1
- RBF: 5
- LWL: 5

However, for the final model, there were only two LWL models selected during the co-operative selection which means that the results shown in Fig. 15c are based on two LWL models only. Taking into account that the non-adaptive ROSS model outperformed the non-adaptive LWPR model it is an encouraging result.

The results for the three non-adaptive models are presented in Fig. 15. First inspection of the data shows that none of the non-adaptive models is able to deal with the data and that the data set calls for an adaptive approach. Although the best performance in terms of MSE and correlation coefficient is achieved by the MLP model, after a visual inspection of the results in Fig. 15 and due to the problems with the model selection of the MLP discussed above the non-adaptive ROSS seems to be a better option for this task. Table 4 gives ones again a summary of the performances of the non-adaptive models.

Adaptive soft sensors As next the adaptive case is analysed. The benchmark model in this case is an incremental version

Table 4 Performance of the non-adaptive models for the thermal oxidiser data

	MLP	LWPR	ROSS
MSE	**1.37e−3**	2.21e−3	1.57e−3
CC	**0.43**	0.18	0.24

of the LWPR algorithm. Using the same parameters as in the non-adaptive case the method is allowed to adapt after making prediction for each of the on-line data samples. The model dramatically improves its performance as demonstrated in Fig. 16.

The results of the different adaptation cases for the adaptive ROSS model are outlined in Table 5.

The table shows that in the case of this data it is necessary to use the Type 2 adaptation, which involves deploying new RF during the on-line operation. This fact is confirmed by the results of the LWPR techniques as well as of the ROSS model. Especially the ROSS model using all available adaptation mechanisms (i.e. adaptation case 3) achieves outstanding performance and is able to follow the data very well as demonstrated in Fig. 16.

The composition of the LEs pool in the Paths module is analysed next. Figure 17 shows that during the on-line phase the LS-SVM method is dominant. One can also see the almost negligible role of the linear regression (MLR) method. Another interesting aspect is the growing importance of the MLP method in the second half of the data. The figure also shows the total number of the LEs in the Paths module, one can see that the number of LEs N^{LE} remains

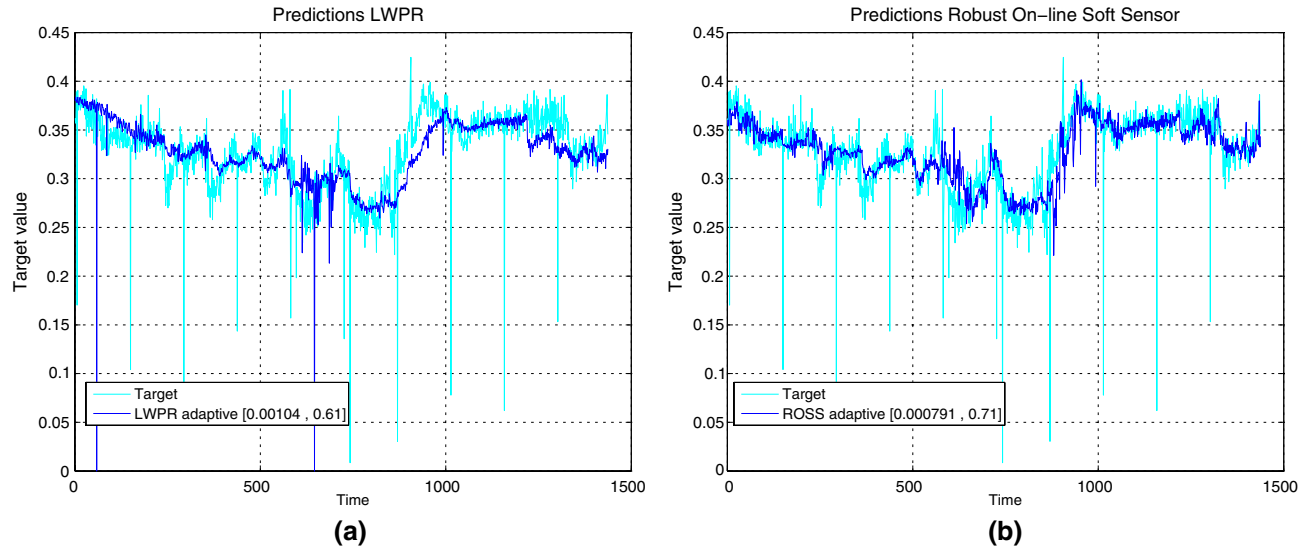

Fig. 16 Predictions of the adaptive models for the thermal oxidiser data. **a** LWPR, **b** ROSS using adaptation case 3

Table 5 Performance of the adaptive models for the thermal oxidiser data

	LWPR	Ad. case 1	Ad. case 2	Ad. case 3
MSE	1.04e−3	1.52e−3	8.48e−4	**7.91e−4**
CC	0.61	0.52	0.69	**0.71**

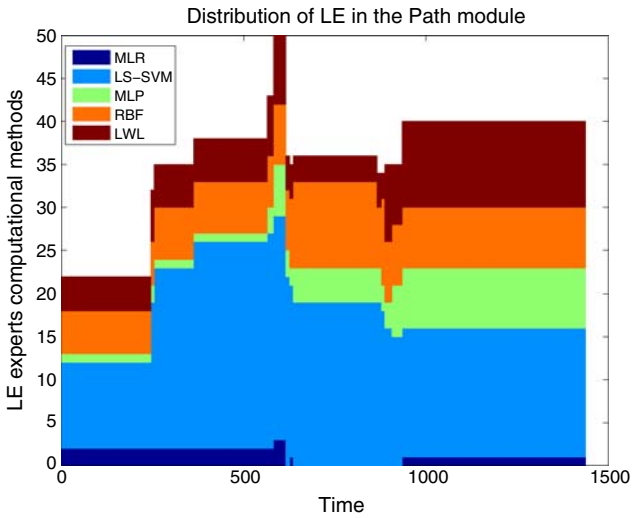

Fig. 17 Composition of the local experts computational methods in the Paths module for the thermal oxidiser data

stable despite the fact that several new RF are deployed during the on-line phase. This fact demonstrates the efficiency of the diversity selection step. The actual deployment of new RF can be observed as the step changes in Fig. 17. Reading the plot one can see that there are 11 RF deployed during the on-line phase.

5.2.2 Catalyst activation

Non-adaptive soft sensors For this data set, the parameter selection for the MLP model was also difficult. Referring to Fig. 18 one can see that both the MSE and correlation coefficient does not provide any strong information for the selection of a particular number of hidden units for the MLP. The selected number of hidden units for the benchmarking was nine and the number of principal components for the PCA pre-processing is five.

As for the LWPR, the following parameters were found as optimal:

- 'init-D' = 10
- 'init-alpha' = 1000
- 'diag-only' = 0
- 'w-gen' = 1.51
- 'w-prune' = 1.0
- 'penalty' = 1e−5
- 'meta' = 1
- 'update-D' = 1
- 'kernel' = 'Gaussian'

The parameters for the ROSS model are not different from the parameters in the previous experiment. The resulting composition of the LEs after the trainings phase was:

- MLR: 0
- LS-SVM: 20
- MLP: 4
- RBF: 7
- LWL: 0.

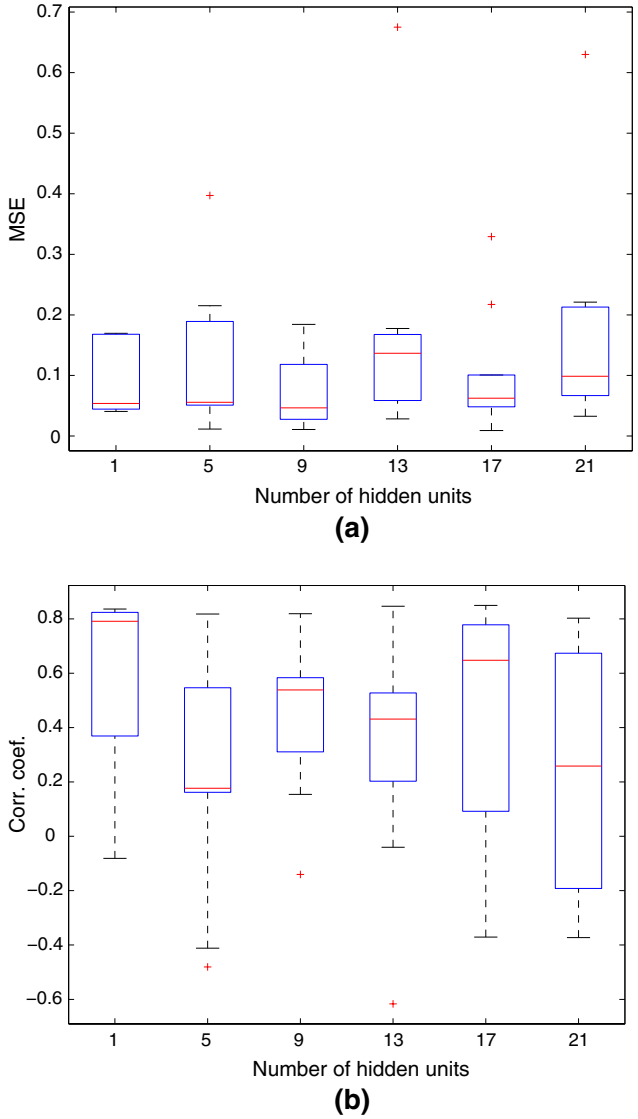

Fig. 18 Performance of the MLP soft sensors for the catalyst activation data. **a** MSE, **b** correlation coefficient

Table 6 Performance of the non-adaptive models for the catalyst activation data

	MLP	LWPR	ROSS
MSE	**4.46e−2**	5.76e−2	1.05e−1
CC	**0.82**	0.32	0.55

One can again observe the dominance of the support vector machines among the 31 LEs. However, in contrast to the previous experiment there are no MLRs or LWLs in the pool in this case. The final model (after the co-operative selection) is built on the basis of four SVM LEs.

Interpreting the results of the non-adaptive soft sensors presented in Fig. 19 and Table 6, it can be clearly seen that none of the models is able to follow the target variable and thus there is a strong need for an adaptive model.

Adaptive soft sensors In the adaptive scenario one can again observe an increase in the performance as demonstrated in Fig. 20. One can also see that the ROSS model outperforms the LWPR model even despite the fact that the LWPR model was manually optimised for this data set which is not the case for the ROSS model. In Table 7 one can see the importance of the Type 2 adaptation for this data set as using this adaptation technique leads to a significant increase in the performance.

Figure 21 shows the stable population of the LEs in the Paths module. It is again clearly demonstrated that the population size remains stable despite the numerous deployments of new RF. The composition of the computational learning patterns shows a different pattern compared to the previous experiment. In this case the distribution of the MLP, RBF and LS-SVM methods is approximately equal and the LWL and MLR methods play an insignificant role.

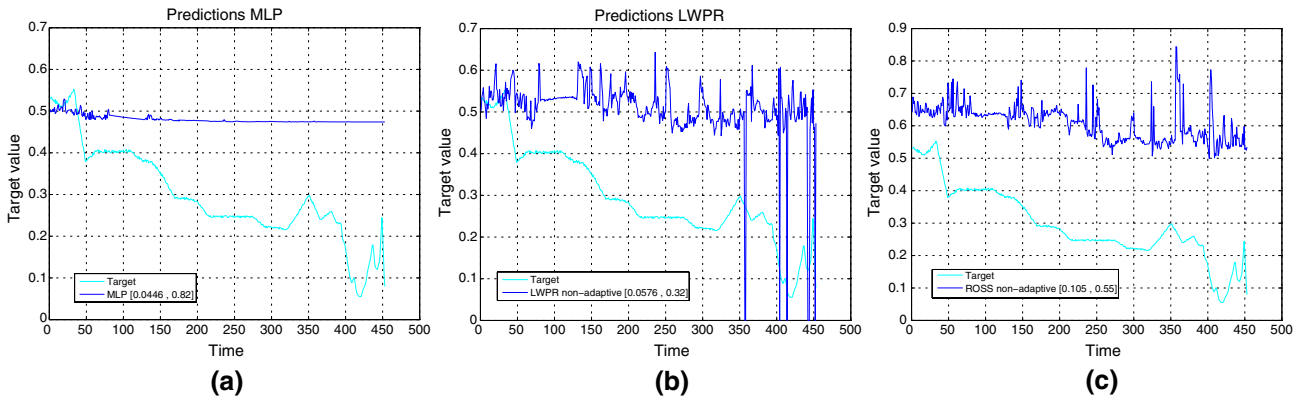

Fig. 19 Predictions of the three non-adaptive models for the catalyst activation data. **a** MLP, **b** LWPR, **c** ROSS

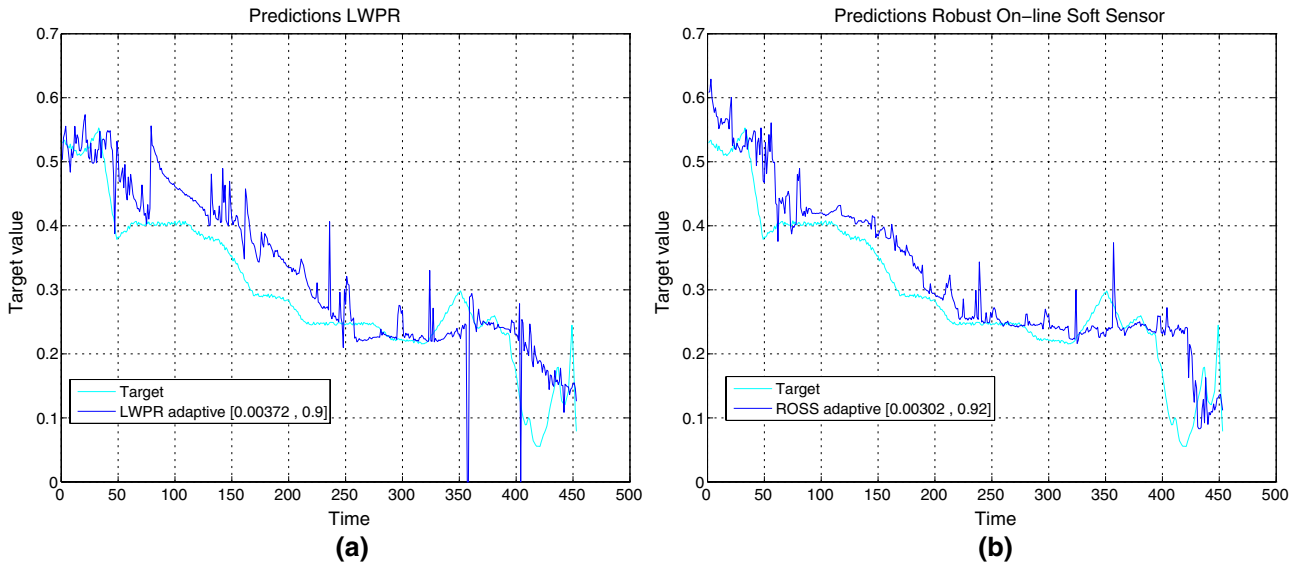

Fig. 20 Predictions of the adaptive models for catalyst activation soft sensor. **a** LWPR, **b** ROSS using adaptation case 3

Table 7 Performance of the adaptive models for the catalyst activation data

	LWPR	Ad. case 1	Ad. case 2	Ad. case 3
MSE	3.72e−3	8.61e−2	3.10e−3	**3.02e−3**
CC	0.90	0.19	0.92	**0.92**

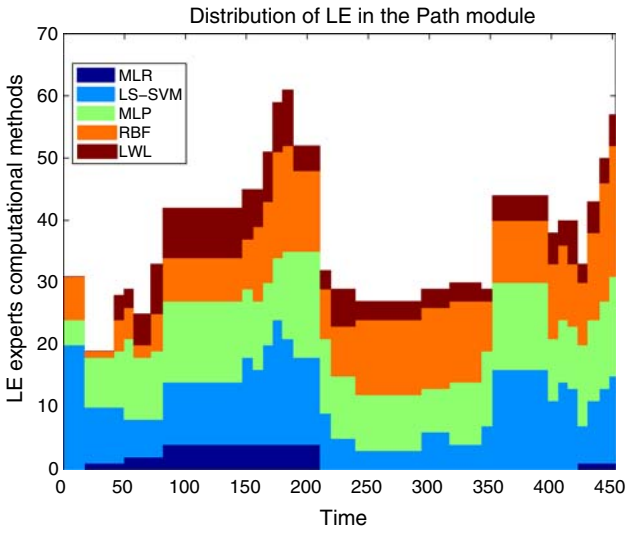

Fig. 21 Composition of the local experts computational methods in the Paths module for the catalyst activation data

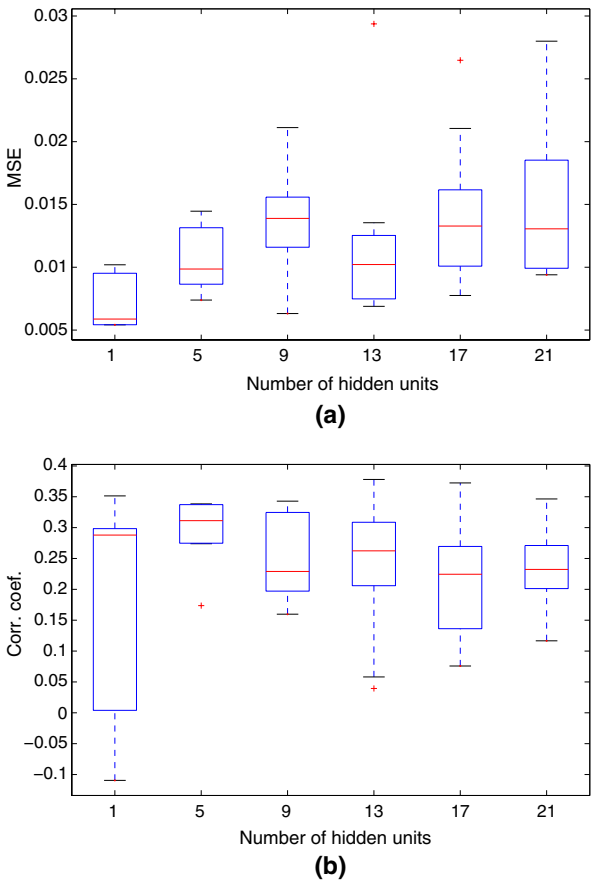

Fig. 22 Performance of the MLP soft sensors for the industrial drier data. **a** MSE, **b** correlation coefficient

5.2.3 Industrial drier

Non-adaptive soft sensors For the pre-processing of this data set the PCA using 8 principal components has been found as optimal. Performing model selection using the MLP leads to similar problems as with the other data sets. Figure 14 shows the box-plots of the MSE and correlation performance of the various MLP models. An MLP with five hidden units has been selected as optimal (Fig. 22).

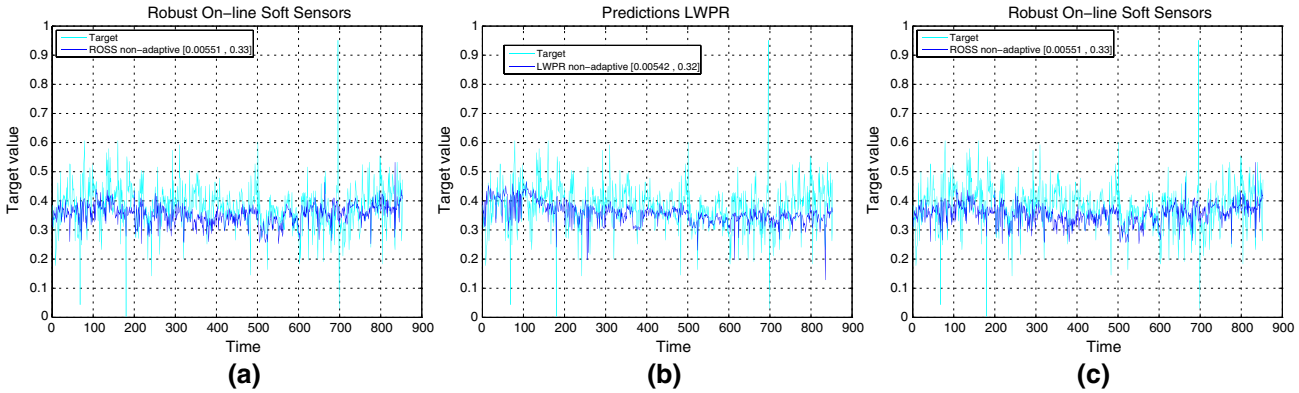

Fig. 23 Predictions of the three non-adaptive models for the industrial drier data. **a** MLP, **b** LWPR, **c** ROSS

The optimal parameter for the LWPR are:

- 'init-D' = 5
- 'init-alpha' = 0.1
- 'diag-only' = 0
- 'w-gen' = 1.0
- 'w-prune' = 0.9
- 'penalty' = 1e−5
- 'meta' = 0
- 'update-D' = 1
- 'kernel' = 'Gaussian'

The parameters used for the ROSS are again the same as for the other data sets (see Table 2) with an exception of the thresholds for the triggering of the Type 2 and Type 4 adaptation (t^{ada2} and t^{ada4}, respectively). Due to the high noise in the target variable (see e.g. Fig. 23) the thresholds for this data sets were increased to $t^{ada2} = 0.20$ and $t^{ada4} = 0.15$ in order to prevent the model from excessive and unnecessary deployment of RF.

During the training phase there were four RF built which resulted in training of 400 LEC. From these, there were 43 LEs left after the competitive and diversity selection. Among these the distribution of the predictive methods was the following:

- MLR: 2
- LS-SVM: 27
- MLP: 1
- RBF: 4
- LWL: 9

After the co-operative selection the final non-adaptive soft sensor consists of four models, namely one MLP and three LWLs.

The results for the three non-adaptive models are presented in Fig. 23. In contrast to the previous two data sets, in this case it is possible to build a non-adaptive model which will deliver good performance as demonstrated by the LWPR and ROSS models which deliver similar performance (Table 8).

Table 8 Performance of the non-adaptive models for the industrial drier data

	MLP	LWPR	ROSS
MSE	9.81e−3	**5.42e−3**	5.51e−3
CC	0.27	0.32	**0.33**

Adaptive soft sensors Although one can again observe a certain increase in the performance when applying the adaptation techniques in this case the increase is not that significant as it was with the other data sets. On the contrary one can also observe a decrease in performance when using only the Type 1 adaptation technique for the ROSS model. The performance drop can lead to the small size of the path combination (only one MLP and three LWL are used) which builds the final model (Fig. 24; Table 9).

Figure 25 is concluding the discussion of this data set. The figure shows once more the stable size of the models in the Paths module as well as a dominant role of the LS-SVM method.

6 Conclusions

We proposed a complex and generic architecture for the development of on-line predictive models. For such models it not only defines an environment in which the models can use several concept from machine learning such as local learning, ensemble methods and meta learning but also defines the interaction channels between these techniques. Through its generic structure the architecture can be applied to a variety

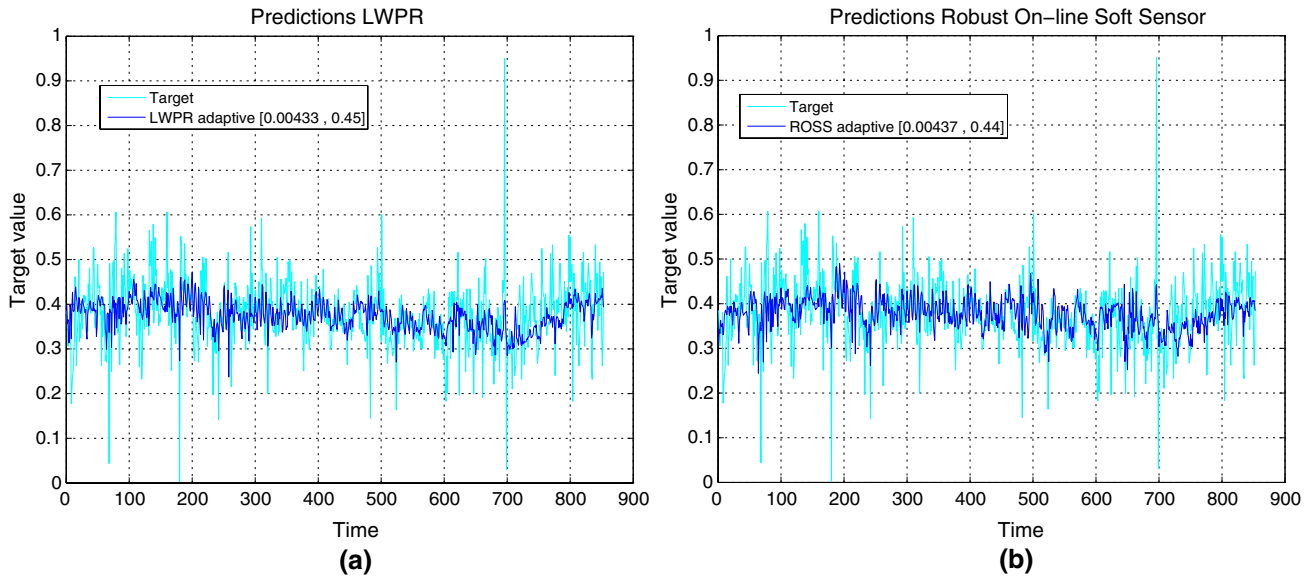

Fig. 24 Predictions of the adaptive models for industrial drier soft sensor. **a** LWPR, **b** ROSS using the adaptation case 3

of different modelling problems. However, the focus of the architecture is put on dealing with difficult predictive problems often found in industrial or commercial applications. In this work we have focused on the application of the architecture to on-line prediction soft sensors which is a challenging task due to the impurities and changing quality of the data upon which the models have to be built. In the presented case study we have shown an instance of the architecture which is based on some commonly available and rather simple techniques from machine learning. As such the contribution of this work is not the invention of yet another predictive modelling technique but rather the demonstration of how the various available techniques can be put into an unified scheme which manages the techniques on different levels in order to deliver strong performance. In the experiments dealing with three real data sets from the process industry we have demonstrated several benefits of the model developed according to the architecture. The model not only outperformed the state-of-the-art non-adaptive and adaptive techniques but also by providing automated data pre-processing and model selection takes a lot of the model development burden away from the model developer. Another strength of the model developed according to the architecture is a variety of adaptation mechanisms which can be applied at different positions in the architecture. As demonstrated in the experiments the adaptation techniques can have a significant effect on the performance of the models. The last but probably most significant contribution of the proposed architecture is that it opens space for further research which can focus for example on the interaction between the techniques in the architecture, on the dynamic behaviour of the architecture, on the implementation of novel adaptation techniques, on the application of more sophisticated approaches for the meta-level methods, etc.

Table 9 Performance of the adaptive models for the catalyst activation data

	LWPR	Ad. case 1	Ad. case 2	Ad. case 3
MSE	**4.33e−3**	7.26e−3	4.67e−3	4.37e−3
CC	**0.45**	0.32	0.39	0.44

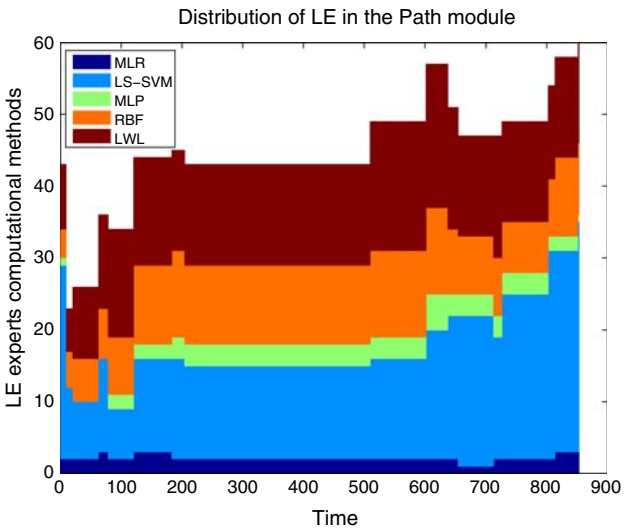

Fig. 25 Composition of the local experts computational methods in the Paths module for the industrial drier data

Acknowledgments We would like to thank Evonik Degussa GmBH for their substantial financial support and for providing the data sets. For numerous discussions we would also like to express our gratitude to Monika Berendsen, Reinhard Dudda and Sibylle Strandt.

References

1. Dote Y, Ovaska SJ (2001) Industrial applications of soft computing: a review. Proc IEEE 89(9):1243–1265
2. Wolpert DH (1992) Stacked generalization. Neural Netw 5(2): 241–259
3. Perrone MP, Cooper LN (1993) When networks disagree: ensemble methods for hybrid neural networks. Neural Netw Speech Image Proc 126–142
4. Kuncheva LI (2004) Combining pattern classifiers: methods and algorithms. Wiley, New Jersey
5. Valentini G, Masulli F (2002) Ensembles of learning machines. In: 13th Italian workshop on neural nets, vol 2486, Lecture Notes in Computer Sciences. Springer, Berlin, pp 3–22
6. Krogh A, Vedelsby J (1995) Neural network ensembles, cross validation and active learning. Adv Neural Inf Proc Syst (7):231–238
7. Kittler J, Hatef M, Duin RPW, Matas J (1998) On combining classifiers. IEEE Trans Pattern Anal Mach Intell 20(3):226–239
8. Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119–139
9. Opitz D, Maclin R (1999) Popular ensemble methods: an empirical study. J Artif Intell Res 11:169–198
10. Bauer E, Kohavi RON (1999) An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Mach Learn 36:105–139
11. Ruta D, Gabrys B (2000) An overview of classifier fusion methods. Comput Inf Syst 7(1):1–10
12. Ruta D, Gabrys B (2005) Classifier selection for majority voting. Inf Fusion 6(1):63–81
13. Gabrys B (2004) Learning hybrid neuro-fuzzy classiffer models from data: to combine or not to combine. Fuzzy Sets Syst 147: 39–56
14. Gabrys B, Ruta D (2006) Genetic algorithms in classifier fusion. Appl Soft Comput 6(4):337–347
15. Geman S, Bienenstock E, Doursat R (1992) Neural networks and the bias/variance dilemma. Neural Comput 4(1):1–58
16. Bishop CM (1995) Neural networks for pattern recognition. Oxford University Press, USA
17. Jacobs R (1997) Bias/variance analyses of mixtures-of-experts architectures. Neural Comput 9(2):369–383
18. Chandra A, Yao X (2006) Evolving hybrid ensembles of learning machines for better generalisation. Neurocomputing 69(7–9): 686–700
19. Poggio T, Girosi F (1990) Regularization algorithms for learning that are equivalent to multilayer networks. Science 247(4945): 978–982
20. Platt J (1991) A resource-allocating network for function interpolation. Neural Comput 3(2):213–225
21. Bottou L, Vapnik V (1992) Local learning algorithms. Neural Comput 4(6):888–900
22. Schaal S, Atkeson CG (1998) Constructive incremental learning from only local information. Neural Comput 10(8):2047–2084
23. Atkeson CG, Moore AW, Schaal S (1997) Locally weighted learning. Artif Intell Rev 11(1):11–73
24. French R (1999) Catastrophic forgetting in connectionist networks: Causes, consequences and solutions. Trends Cogn Sci 3(4): 128–135
25. Vijayakumar S, D'Souza A, Schaal S (2005) Incremental online learning in high dimensions. Neural Comput 17(12):2602–2634
26. Vilalta R, Drissi Y (2002) A perspective view and survey of meta-learning. Artif Intell Rev 18(2):77–95
27. Aha DW (1992) Generalizing from case studies: a case study. In: Proceedings of the ninth international conference on machine learning, pp 1–10
28. Pfahringer B, Bensusan H, Giraud-Carrier C (2000) Meta-learning by landmarking various learning algorithms. In: Proceedings of the Seventeenth international conference on machine learning, vol 951. Morgan Kaufmann, Menlo Park, pp 743–750
29. Kalousis A, Hilario M (2001) Model selection via meta-learning: A comparative study. Int J Artif Intell Tools 10(4):525–554
30. Peng Y, Flach PA, Soares C, Brazdil P (2002) Improved dataset characterisation for meta-learning. Lect Notes Comp Sci 2534:141–152
31. Wong RO (1995) Use, disuse, and growth of the brain. In: Proceedings of the National Academy of Sciences of the United States of America. vol 92, National Academy of Sciences, USA, pp 1797–1799
32. Kadlec P, Gabrys B, Strandt S (2009) Data-driven soft sensor in the process industry. Comput Chem Eng 33(4):795–814
33. Kadlec P, Gabrys B (2008) Soft sensor based on adaptive local learning. In: Coghill MK, Kasabov N, George (eds) Proceedings of the international conference on neural information processing, vol 5506, Lecture Notes in Computer Science. Auckland, New Zealand, Springer, Berlin, pp 1172–1179
34. Kasabov N (2001) Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning. IEEE Trans Syst Man Cybern B 31(6):902–918
35. Kasabov N, Song Q (2002) Denfis: dynamic evolving neural-fuzzy inference system and its application for time-series prediction. IEEE Trans Fuzzy Syst 10(2):144–154
36. Angelov P, Filev DP (2004) Flexible models with evolving structure. Int J Intell Syst 19(4):327–340
37. Angelov P, Kasabov N (2005) Evolving computational intelligence systems. In: IEEE workshop on genetic and fuzzy systems, Grenada, Spain
38. Brown G, Wyatt J, Harris R, Yao X (2005) Diversity creation methods: a survey and categorisation. Inf Fusion 6(1):5–20
39. Jacobs R (1991) Adaptive mixtures of local experts. Neural Comput 3(1):79–87
40. Ruta D, Gabrys B (2007) Neural network ensembles for time series prediction. In: International joint conference on neural networks 2007. IEEE Computer Society, Orlando, pp 1204–1209
41. Riedel S, Gabrys B (2007) Dynamic pooling for the combination of forecasts generated using multi level learning. In: International joint conference on neural networks 2007, IEEE Computer Society, pp 454–459
42. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability 1(14): 281–297
43. Angelov P, Filev D (2005) Simplets: a simplified method for learning evolving takagi-sugeno fuzzy models. In: The 14th IEEE international conference on fuzzy systems, IEEE, pp 1068–1073
44. Angelov P, Zhou X (2006) Evolving fuzzy systems from data streams in real-time. In: International symposium on evolving fuzzy systems 2006, pp 29–35
45. Chung PJ, Bohme JF (2003) Recursive em algorithm with adaptive step size. In: Seventh international symposium on signal processing and its applications, vol 2, IEEE, pp 519–522
46. Gabrys B, Bargiela A (2000) General fuzzy min-max neural network for clustering and classification. IEEE Trans Neural Netw 11(3):769–783

47. Gabrys B, Petrakieva L (2004) Combining labelled and unlabelled data in the design of pattern classification systems. Int J Approx Reason 35(3):251–273
48. Neal RM, Hinton GE (1999) A view of the em algorithm that justifies incremental, sparse, and other variants. In: Learning in graphical models, vol 89. MIT Press, Cambridge, pp 355–368
49. Zivkovic Z, Heijden Fvan der (2004) Recursive unsupervised learning of finite mixture models. IEEE Trans Pattern Anal Mach Intell 26(5):651–656
50. Bensusan H, Giraud-Carrier C, Kennedy C (2000) A higher-order approach to meta-learning. In: Proceedings of the ECML^{TM} 2000 workshop on Meta-Learning, pp 109–117
51. Giraud-Carrier C (1998) Beyond predictive accuracy: what? In: Proceedings of the ECML-98 workshop on upgrading learning to meta-level, pp 78–85
52. Bouchachia A (2006) Incremental learning by decomposition. In: ICMLA '06: Proceedings of the 5th international conference on machine learning and applications, IEEE Computer Society, pp 63–68
53. Gama J, Medas P, Castillo G, Rodrigues P (2004) Learning with drift detection. In: Advances in artificial intelligence SBIA 2004: 17th Brazilian, vol 3171, pp 286–295
54. Maloof MA, Michalski RS (2000) Selecting examples for partial memory learning. Mach Learn 41(1):27–52
55. Klinkenberg R (2004) Learning drifting concepts: Example selection vs. example weighting. Intell Data Anal 8(3):281–300
56. Koychev I (2000) Gradual forgetting for adaptation to concept drift. In: Proceedings of ECAI 2000 workshop current issues in spatio-temporal reasoning, pp 101–106
57. Croux C, Ruiz-Gazen A (2005) High breakdown estimators for principal components: the projection-pursuit approach revisited. J Multivar Anal 95(1):206–226
58. Dobson AJ (2002) An introduction to generalized linear models. Chapman and Hall, London
59. Vapnik VN (1998) Statistical learning theory. Wiley, New York
60. Frank E, Hall M, Pfahringer B (2003) Locally weighted naive bayes. In: Proceedings of the conference on uncertainty in artificial intelligence, pp 249–256
61. Gosset WS (1908) The probable error of a mean. Biometrika 6(1):1–25
62. Parzen E (1962) On estimation of a probability density function and mode. Ann Math Stat 33(3):1065–1076
63. Klanke S, Vijayakumar S, Schaal S (2008) A library for locally weighted projection regression. J Mach Learn Res 9:623–626