Automatic and adaptive preprocessing for the development of predictive models

MANUEL MARTÍN SALVADOR

A thesis submitted in partial fulfilment of the requirements of Bournemouth University for the degree of

Doctor of Philosophy

April, 2017

Copyright statement

Abstract

In recent years, there has been an increasing interest in extracting valuable information from large amounts of data. This information can be useful for making predictions about the future or inferring unknown values. There exists a multitude of predictive models for the most common tasks of classification and regression. However, researchers often assume that data is clean and far too little attention has been paid to data preprocessing. Despite the fact that there are a number of methods for accomplishing individual preprocessing tasks (e.g. outlier detection or feature selection), the effort of performing comprehensive data preparation and cleaning can take between 60% and 80% of the whole data mining process time. One of the goals of this research is to speed up this process and make it more efficient. To this end, an approach for automating the selection and optimisation of multiple preprocessing methods and predictors has been proposed.

The combination of multiple data mining methods forming a workflow is known as Multi-Component Predictive System (MCPS). There are multiple software platforms like Weka and RapidMiner to create and run MCPSs including a large variety of preprocessing methods and predictors. There is, however, no common mathematical representation of MCPSs. An objective of this thesis is to establish a common representation framework of MCPSs. This will allow validating workflows before beginning the implementation phase with any particular platform. The validation of workflows becomes even more relevant when considering the automatic generation of MCPSs.

In order to automate the composition and optimisation of MCPSs, a search space is defined consisting of a number of preprocessing methods, predictive models and their hyperparameters. Then, the space is explored using a Bayesian optimisation strategy within a given time or computational budget. As a result, a parametrised sequence of methods is returned which after training form a complete predictive system. The whole process is data-driven and does not require human intervention once it has been started.

The generated predictive system can then be used to make predictions in an online scenario. However, it is possible that the nature of the input data changes over time. As a result, predictive models may need to be updated to capture the new characteristics of the data in order to reduce the loss of predictive performance. Similarly, preprocessing methods may have to be adapted as well. A novel hybrid strategy combining Bayesian optimisation and common adaptive techniques is proposed to automatically adapt MCPSs. This approach performs a global adaptation of the MCPS. However, in some situations, it could be costly to update the whole predictive system when maybe just a little adjustment is needed. The consequences of adapting a single component can, however, be significant. This thesis also analyses the impact of adapting individual components in an MCPS and proposes an approach to propagate changes through the system.

This thesis was initiated due to a joint research project with a chemical production company, which has provided several datasets with common raw data issues in the process industry. The final part of this thesis evaluates the feasibility of applying such automatic techniques for building and maintaining predictive models for real chemical production processes.

Contents

List of Figures

List of Tables

Notation

	Symbol	Description	Example
Predictive models	x	Scalar value	$x = 10$
	$\boldsymbol{x}$	Vector	$\boldsymbol{x} = \{x_1, ...\}$
	X	Matrix	$X = \{\boldsymbol{x}_1, ...\}$
	$\hat{y}$	Estimated value	$\hat{y} = 0.5$
	f	Function / predictive model	$\hat{y} = f(\boldsymbol{x})$
	p	Probability function	$p(c) = 0.3$
	$\mathcal{X}$	Domain	$\mathcal{X} = \mathbb{R}^2$
	$\mathcal{D}$	Dataset	$\mathcal{D} = \{(\boldsymbol{x}_1, y_1), ...\}$
	λ	Hyperparameter	$\lambda = 5$
	$\boldsymbol{\lambda}$	Set of hyperparameters	$\boldsymbol{\lambda} = \{\lambda_1, ...\}$
	Λ	Hyperparameter space	$\Lambda = \mathbb{R}$
	$\boldsymbol{\Lambda}$	Combined hyperparameter spaces	$\boldsymbol{\Lambda} = \Lambda_1 \times ...$
	A	Learning algorithm	$f = A_{\boldsymbol{\lambda}}(\mathcal{D}_{train})$
	$\mathcal{L}$	Loss function	$\mathcal{L}(f, \mathcal{D})$
	ϵ	10-fold CV error on training data	
	$\mathcal{E}$	Holdout error on testing data	
Petri nets	PN	Petri net	$PN = (P, T, F)$
	p	Single place	$p \in P$
	P	Places of the PN	$P = \{p_1, ...\}$
	t	Single transition	$t \in T$
	T	Transitions of the PN	$T = \{t_1, ...\}$
	$f_{p,t}$	Directed arc from p to t	$f_{p,t} \in F$
	F	Arcs of the PN	$F = \{f_{p_1,t_1}, ...\}$
	$w(p_i, t_i)$	Weight function associated to f_{p_i,t_i}	$w(p_1, t_1) = 1$
	W	Weights of the arcs in the PN	$W = \{w(p_1, t_1), ...\}$
	M_k	State of a PN at moment k	$M_k \in \mathcal{M}$
	$M_k(p_i)$	Number of tokens in the place p_i in the state M_k	$M_k(p_i) = 1$
	$\mathcal{M}$	Set of states of the PN	$\mathcal{M} = \{M_0, ...\}$
	i	Input/source place of a WF-net	$i \in P$
	o	Output/sink place of a WF-net	$o \in P$
	n	Node of a PN	$n \in P \cup T$
	C	Path in a PN	$C = \langle n_1, ..., n_k \rangle$
	$\bullet p$	Inputs nodes of p	$\bullet p = \{t_1, ...\}$
	$p\bullet$	Outputs nodes of p	$p\bullet = \{t_2, ...\}$
	θ	Multicomponent Predictive System	$\theta = (P, T_{\boldsymbol{\lambda}}, F)$
	Θ	Search space of MCPSs	$\Theta = \{\theta_1, ...\}$
	ψ	Surrogate model	
	α	Acquisition function	
	$d(\theta_a, \theta_b)$	Distance between MCPSs θ_a and θ_b configurations	

Acknowledgements

In the first place, I would like to thank my supervisors Prof. Bogdan Gabrys and Dr Marcin Budka for their constant support and advice. They are not only excellent academics but honest men with a great sense of humour that have kept me motivated to finish this thesis. Special thanks also to my former supervisor Dr Indre Žliobaitė for her support during the beginning of my research.

The PhD research is a long journey and I could not have had better travel companions than my colleagues and friends at Bournemouth University. Special thanks to Rashid, Amir, Tauheed, Walter, Bernadette, Ali, Najmeh, Mohsen, Mahmood, Parisha, Diana, Lina, Mandy, Nico, Katherine, Tucker, Nazmul, Pree, Ed, Thanos, Bassma, Abbas, Utku, Bastian, Alex, Emmy, and many more! And of course, I cannot forget my favourite group of Spaniards: Azahara, Alejandro and Oxala. My thanks also go to my friends in Granada and around the globe, who have been always there to cheer me up in this period.

I would also like to thank Bournemouth University staff that have always been nice and helpful to me. Special thanks to Dr Emili Balaguer, Dr Damien Fay, Dr Katarzyna Musial-Gabrys, Naomi Bailey, Patti Davies, Malcolm Green, Natalie Andrade, and Shaun Bendall.

A large part of this research has been reviewed by anonymous academics. Many thanks to them for their valuable time and constructive feedback. Thanks also to the peers for the exciting discussions in various conferences and events.

Last but not least, I would like to thank my family for their love and belief in me. Special thanks to my parents for guiding me through the right direction in both personal and academic sense. Finally, I have no words to express my gratitude to Cristina ♡, who has been sharing with me both joy and pain during the last 5 years.

Author's declaration

The work contained in this thesis is the result of my own investigations and has not been accepted nor concurrently submitted in candidature for any other award.

To my grand and godparents José and Concepción

Chapter 1

Introduction

In the last decades, computing and telecommunications have changed the world (Castells (2009)). The field of artificial intelligence (AI) has promised since the '60s that machines would be able to solve any task a human can. Despite significant advances, AI systems were still far from solving many labour-intensive tasks. This has changed however in the last decades. The deployment of computational systems in various fields has led to an exponential explosion in the amount of data generated by different sources. The combination of this massive amount of data and affordable computational power has led to the development of smarter systems in many areas. For instance, visual recognition (Lee et al. (2009)), spoken language understanding (Deng et al. (2012)), and self-driving vehicles (Bojarski et al. (2016)) are just some of the most recent and exciting applications.

Machine learning (ML) is the field that studies the construction of intelligent systems able to learn from data (Hastie et al. (2009)). For extracting useful patterns from data, a mathematical model can be built. The process of building these models for predicting or forecasting is known as predictive modelling. Dozens of learning algorithms exist to approach this task and they are usually grouped into two main categories according to the nature of the problem at hand: a) classification, when the value to predict is discrete (e.g. to decide if a loan is granted or not, Huang et al. (2004)), or b) regression, if the value to predict is continuous (e.g. predicting the monthly sales of jeans, Sun et al. (2008)).

Data mining (DM) is the process of analysing data for extracting and interpreting knowledge (Han et al. (2011)). There is a large number of fields in which data mining plays a crucial role in the business revenue. Examples are banking (e.g. fraud detection, Wei et al. (2013)), health (e.g. gene selection, Guyon et al. (2002)), energy (e.g. consumption pattern identification, Tso & Yau (2007)) or supermarket chains (e.g. sales prediction, Kuo (2001)) to name just a few. Nevertheless, many works in this field assume that data is clean and ready to be learnt from, but in real problems that is very often not the case (Pearson (2005)). For that reason, data preprocessing is an essential step in data mining, since raw data may come with imperfections such as missing values or outliers that can reduce the performance of the predictive system or even make it impossible to build a predictive model (Pyle (1999)).

A survey[1] of DM practitioners carried out in 2003 indicates that preprocessing tasks can account for as much as 60-80% of the total time spent on developing a predictive model. More recent surveys from 2012 (Munson (2012)) and 2016[2] confirm these numbers and practitioners say that preprocessing is the least enjoyable part of data science. The reason for this large amount of time is due to all the manual work necessary to identify the defects in the raw data and look for the best solutions to approach them. Despite 13 years between the surveys, no significant advances have been made to address this issue. Therefore, it is desirable to automate as many of the tasks of data preprocessing as possible in order to reduce the human involvement and the level of necessary interactions. The consequence of this would be the speeding up of the data mining process and making the procedures more robust.

1.1 Background and motivation

The research described in this thesis was initiated in the scope of INFER[3], a European project which aimed to develop a software platform for predictive modelling applicable in different industries. The organisations involved in the project were Bournemouth University as the research partner, Evonik Industries as the industrial partner and REC (Research and Engineering Centre) as the software developer partner. As Evonik Industries is a major chemical manufacturer, one of the main interests of the project was to develop adaptive predictive systems for online prediction, monitoring and control of chemical production processes.

Building a successful predictive system depends mainly on two aspects: 1) the amount and quality of the available data, and 2) the choice of the learning algorithm and hyperparameters (i.e. the settings that influence the learnt model). Although raw data from process industry can be abundant, it is also often imperfect and requires intensive cleaning and preprocessing to prepare a dataset for modelling. Preprocessing and predictive models can be combined forming a workflow. This workflow receives raw input data and returns prediction values. Composing a successful workflow requires to select the best machine learning algorithms and to additionally optimise its hyperparameters. These tasks require an intensive trial and error, becoming a tedious process for an expert. Therefore, techniques allowing to reduce this effort by automatically building a predictive system are extremely desirable.

Maintainability of the predictive systems over time is also a crucial aspect in the process industry. Once a predictive system is deployed in a production environment, one can observe a gradual deterioration of its performance over time (Kadlec & Gabrys (2011)). There are several causes of this phenomenon including degradation of physical sensors, varied quality of raw materials, changes in the chemical reaction and even environmental changes like summer temperatures (Kadlec et al. (2011)). Adaptive models are often used

[1] `http://www.kdnuggets.com/polls/2003/data_preparation.htm`

[2] 'Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says' by Gil Press. Forbes 2016. `http://bit.ly/forbes-data-preparation`

[3] `http://infer.eu`

to cope with changes in the data distribution and therefore to continue delivering accurate predictions (Kadlec et al. (2011)). Nevertheless, some of the preprocessing steps belonging to the same workflow might need to be adapted as well (Žliobaitė & Gabrys (2014)). Fully adapting these predictive workflows has not been yet considered in practice or even in the scientific literature.

The challenging tasks of automating the construction of predictive systems and adapting them to changes in data with a special focus on data preprocessing are therefore the main subjects of the research pursued in this thesis. The work from other colleagues from the same research group has focused on meta-learning (Lemke et al. (2015)), adaptive systems (Bakirov et al. (2017)) and multi-objective optimisation (Al-Jubouri & Gabrys (2014)), effectively tackling other parts of the broader problem of development adaptive online prediction models (Kadlec & Gabrys (2009)).

1.2 Process industry: a case study

After the industrial revolution, modern societies depend heavily on process manufacturing. Food, beverages, clothes and drugs are some of the products that are made in factories all over the world and delivered to final customers on a daily basis. One of the largest industries worldwide is chemical manufacturing, with 2015 turnover of €3,534 billion[4]. The EU chemical industry is the second major seller after China, employing 1.2 million workers and contributing €551 billion to the EU economy. Being competitive in such global market without lowering wages is a challenge that involves increasing process efficiency using improvement methodologies like Six Sigma (Tennant (2001)) and investing in R&D to innovate.

One of the major innovations in process industry in the last decades was the introduction of soft sensors (Kadlec et al. (2009)), which are predictive models built on top of data recorded from sensors in the processing plants. There is a range of tasks that soft sensors can be used for. The main ones are online prediction, process monitoring and fault detection.

1.2.1 Chemical processes

A chemical process consists of obtaining one or more products through a reaction process involving one or more compounds. The four basic chemical reactions are:

- **Synthesis**, where two or more compounds are combined to obtain a new one (e.g. $C + O_2 \rightarrow CO_2$).
- **Decomposition**, where a compound is split into two or more products (e.g. electrolysis of water $2H_2O \rightarrow 2H_2 + O_2$).

[4]cefic (European Chemical Industry Council). "The European Chemical Industry. Facts and figures 2016." `http://www.cefic.org/Facts-and-Figures/`

- **Single replacement**, where an element displaces a less active element of a compound (e.g. $2AgNO_3 + Zn \rightarrow 2Ag + Zn(NO_3)_2$).
- **Double replacement**, where two elements displace other two elements of compounds (e.g. salt production $HCl + NaOH \rightarrow NaCl + H_2O$).

Manufacturing a product often requires running several reactions. Mass production is performed in chemical plants where vessels are usually connected by pipes that carry the materials. The chemical reaction takes place in a vessel called reactor, where the mix is heated during a period of time. The output of the reactor is then moved to the distillation column where the final products are separated (see Figure 1.1).

Depending on the nature of the process, chemical plants can run continuous or batch production. The first ones can initially operate indefinitely assuming that materials are continuously provided. Nevertheless, they need to be stopped for maintenance and often adjusted to cope with changes in product demand. On the other hand, batch processes run for a limited amount of time. They are started on demand and stopped when the

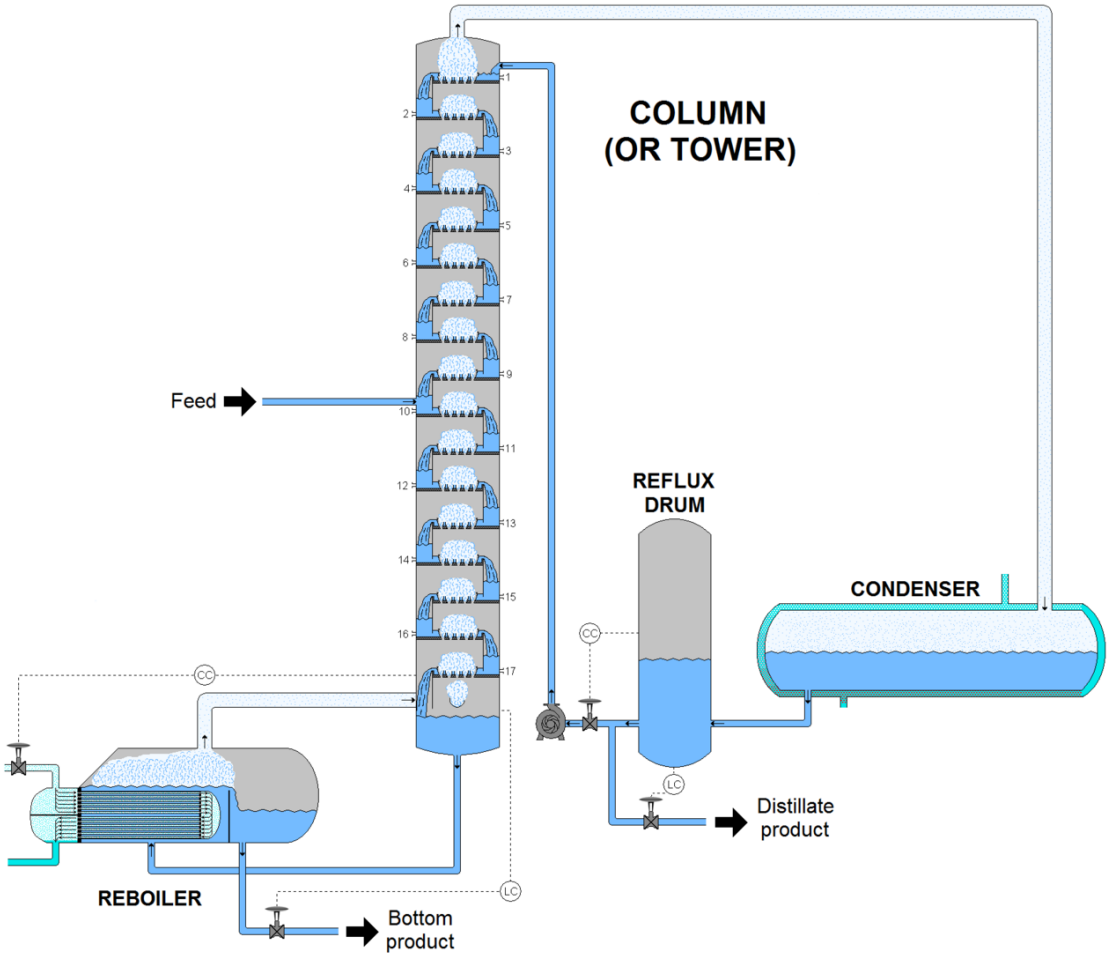


Figure 1.1: *Diagram of a distillation column. The mix from the reactor is fed into the column and then separated into two products. (Creative Commons by Marco Guzman.)*

desired amount of product has been achieved. The datasets used for experimentation in this thesis contain measures from sensors located in continuous processing plants (see Appendix A for descriptions).

1.3 Raw data from chemical processes

Sensors located in different points of the production plants collect physical measures such as temperatures, flows and pressures. These values are stored in large databases called Process Information Management Systems (PIMS). Raw values in PIMS can be either continuous or discrete, and are usually timestamped.

1.3.1 Data availability

The main goal of predictive models is to estimate a target value (i.e. a discrete label in case of classification, or a continuous value for regression). For that, a model is built based on a set of historical data which can often be gathered from PIMS. Despite usual abundance of sensor measurements, ground truth values may not be always available with the same frequency.

Once a soft sensor is deployed, it needs to predict values from real-time data. This input data arrives in a sequential way, but the correct target values are usually delayed. For example, some measurements could be available every two hours because they are manually entered once a sample has been analysed in the laboratory. Once the true values are known, it is possible to evaluate the model performance to assess if any update is needed.

1.3.2 Common issues

Raw data from sensors usually contain anomalies that can affect model building and predictions. Common issues found due to faulty sensors are missing values, noise and outliers to name just a few. In addition, data from process industry is often high dimensional and redundant/highly correlated due to the large number of sensors and their physical arrangement along the production pipeline. For the same reason, the recorded measurements present delays or even different sampling rates between them which would need to be taken into account in the predictive model building process. Also, depending on the sampling rate, one can end up with a high volume of data that can slow down the mining process. To reduce the space requirements, data is often stored using lossy compression which once decompressed can carry artifacts that should be taken into consideration when building and evaluating a predictive model. All these issues are discussed further in this section.

Missing values

Data can contain variables whose values are missing. There are many reasons why the data may be missing (Pani & Mohanta (2011); Han et al. (2011)). For example, in process industry common causes include sensor failures, transmissions errors and human mistakes. There are robust models that handle incomplete data, but others can have problems dealing with them and can negatively affect the quality of the learnt model.

Let D be a data matrix with n instances and p variables, and R a $n \times p$ binary matrix that indicates the presence or absence of each value in D. That is, $r_{ij} = 1$ if value $d_{ij} \in D$ is missing and $r_{ij} = 0$ otherwise. Let's denote D_o as the observed values and D_m as the missing values. According to Rubin (1976), missing data can be divided into three categories:

- **Missing completely at random**, if the probability that a particular attribute is missing is not related to the missing value or to the values of other features. That is, $\Pr(R|D_o, D_m) = \Pr(R)$. For example, this is the case when a physical sensor fails (at random) to deliver a value;
- **Missing at random**, if the probability that a particular attribute is missing is not related to its own value but is related to the values of other attributes. That is, $\Pr(R|D_o, D_m) = \Pr(R|D_o)$. For example, when a temperature sensor is damaged by high pressure in a vessel;
- **Missing not at random**, if the probability that some attribute is missing is a deterministic function of this attribute value. In this case separate models need to be built for both missing and the observed data. For instance, when values from a sensor are always missing when the temperature is too high.

Noise

Noise is a random error or variance in a measured variable (Han et al. (2011)). Sensor readings could be noisy due to the precision of sensing mechanisms and interferences. According to Nettleton et al. (2010), noise can be characterised by:

- **Distribution**: How noise is distributed in the data (e.g. Gaussian distribution);
- **Location**: Noise can be located in the input attributes, in the output class, in the training data, in the test data, or in a combination of all of the above;
- **Magnitude**: Noise values can be relative to each data value of each variable, or relative to the minimum, maximum and standard deviation for each variable.

Predictive accuracy can decrease if noise in measurements is too large. Reviews of how noise affects different machine learning algorithms can be found in the works of Kalapanidas et al. (2003) and Nettleton et al. (2010). Data visualisation tools, such as boxplots, are helpful for identifying noise in data.

Data outliers

Sometimes, some values lie far away from the typical data distribution. These values are called outliers and may happen in one or more attributes at the same time (i.e. multivariate outliers). Similarly to missing values, typical causes of outliers in process industry are hardware failures, process disturbances, instrument degradation, transmission problems or human errors (Pani & Mohanta (2011); Kadlec et al. (2009)). According to Qin (1997), outliers can be categorised into two basic types:

- **Obvious outliers**, whose values violate physical or technological limitations. For example, a temperature value cannot be below absolute zero;
- **Non-obvious outliers**, which have unusual values and do not reflect the correct variable state. For instance, an abrupt change in temperature measurement might be difficult to detect as an outlier. Even if the value lies between the acceptable limits, it is known that thermal change is a gradual process. See for example Figure 1.2, where temperature values at t_1 and t_2 are the same, but only t_2 is an anomaly. This type of outliers are known as contextual anomalies.

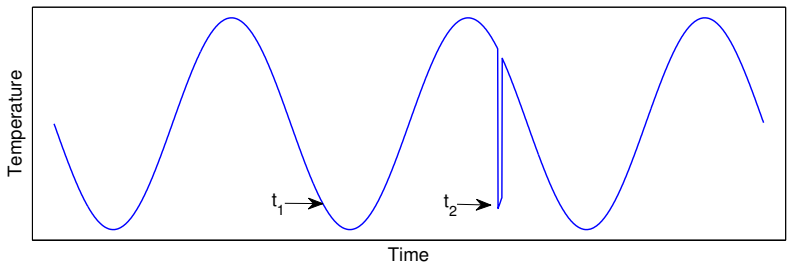


Figure 1.2: *Contextual anomaly t_2 in a temperature time series. Both temperatures at t_1 and t_2 are the same, but t_2 is out of context.*

A model built with data that contain outliers may be less accurate than other models learnt with cleaned data. Though there are some types of models that are robust against outliers, it is often necessary to clean the data for achieving good results. Chandola et al. (2009) present an extensive survey of outlier detection techniques and applications. Another discipline related to outlier detection is *novelty detection* which tries to identify new or unknown data that a machine learning system is not aware of during training (see Markou & Singh (2003a,b) for a review of this topic).

Shutdown periods

Adaptive soft sensors are often updated with new data for capturing well the underlying behaviour of processes that evolve over time (Kadlec et al. (2011)). However, continuous processing plants need to be stopped from time to time for maintenance purposes. If the data from these shutdown periods is not removed, the predictive model can adapt to an undesirable process state in which the predicted values are not being meaningful from the process point of view. Accurately cleaning this data in an online manner is not

straightforward. Appendix B presents a method for dealing with this problem in a real case study.

Redundancy and inconsistency

A dataset can present redundancy at two levels:

- **Attribute redundancy**: An attribute is considered redundant if it can be easily derived from another attribute or set of attributes (Han et al. (2011)). This type of redundancy can be caused by duplicity of sources (e.g. physical sensors measuring the same quantity). The presence of irrelevant and redundant attributes can be misleading for models using all the attributes for predictions such as Naïve Bayes (Gama (2000));
- **Instance redundancy**: Sometimes data can have several instances with the same values (i.e. duplicity), or even worse, with different values for the same measurement (i.e. inconsistency). For example, to avoid the loss of information due to transmission errors, the system can send the data again and occasionally, the information can be duplicated in the database. This duplicity introduces uncertainty into the database which can lead to misleading reports (e.g. when data is aggregated), bias decision-making and ineffective model learning (i.e. due that duplicated data will have more importance than the rest), among others. Sometimes data inconsistency is a symptom of a poor database design, but it also can be caused by the integration of different data sources (Pyle (1999)).

High dimensionality

The dimensionality of a dataset is equal to the number of attributes it contains. Large chemical plants have hundreds of physical sensors that are used to monitor the production process, each of them being a potential input to a soft sensor and hence an attribute. Having high dimensionality presents three main problems (Pyle (1999)):

- **Computational complexity**: The more variables a dataset has, the more expensive it is to process all of them;
- **Increment of state space**: As dimensionality increases, more data points are needed to fill the space to any particular density. This phenomenon is known as the 'curse of dimensionality' (Bellman (1957));
- **Combinatorial explosion**: With the number of possible combinations of values increasing, the creation of a fully representative model becomes impossible.

Furthermore, if the dataset has a large number of attributes it is very possible to find redundant attributes as explained previously. When two or more attributes are highly correlated, it is said that there is **multicolinearity**. For example, the redundancy of sensors in industrial processes results in strongly correlated measurements. This fact might increase model complexity and even negatively affect its performance (Kadlec (2009); Pani & Mohanta (2011)). For this reason, feature selection and data transformation techniques are used for choosing a subset of non-colinear variables as described further in

Section 2.4.1.

Other less known issue due to high dimensionality is **distance concentration** (Aggarwal et al. (2001)). When dimensionality increases, all the pairwise distances between attributes may converge to the same value as shown in Figure 1.3-b (Hinneburg et al. (2000)). The lack of contrast between the nearest and the furthest points affects each area where high-dimensional data processing is required. Aggarwal et al. (2001) show that the L_k norm worsens faster with increasing dimensionality for higher values of k (see Figure 1.3), and recommend to use a fractional distance metric (e.g. $k = 0.5$) in high dimensional problems instead of commonly used Euclidean ($k = 2$) or Manhattan ($k = 1$) distances. Kabán (2011) presents some dimensionality reduction techniques that are aware of this issue.

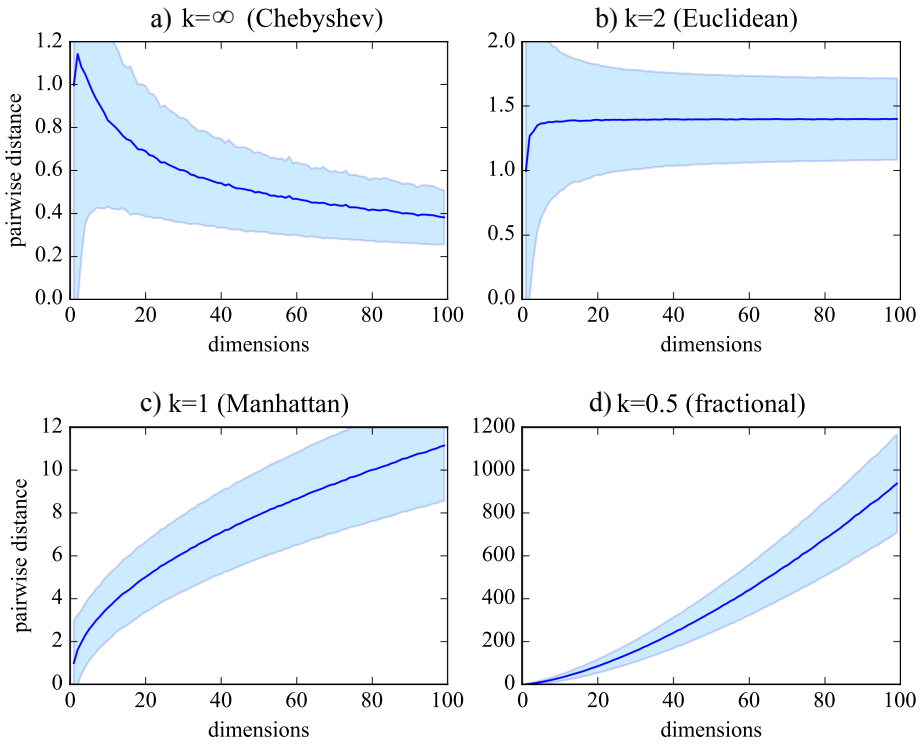


Figure 1.3: *Mean pairwise distance between random vectors drawn from a normal distribution for various L_k measures (shaded region denotes the 95% confidence interval). The L_k norm worsens faster with increasing dimensionality for higher values of k. Thus, it is recommended to use a fractional distance metric (e.g. $k = 0.5$) in high dimensional problems.*

High volumes of data

Datasets with a large number of instances can slow down the mining process or even make it impracticable due to memory limitations. In the last few years, the **Big Data**

term has gained popularity[5]. This term describes not only large datasets, but it also refers to data complexity and integration from different sources. Douglas (2001) defines the big data as a three-dimensional concept: increase of volume (i.e. amount of data), velocity (i.e. speed of data in and out) and variety (i.e. range of data types and sources). More recent definitions of Big Data[6] include additional V's: veracity (i.e. accuracy of data), visualisation, variability (i.e. data is constantly changing) and value.

Compressed data

Product information management systems (PIMS), where historical data is stored, typically use lossy compression. This means that true data points are recorded periodically while the ones in between are interpolated when decompressing the data. In addition, some true values are also stored if they can no longer be approximated to a given tolerance by a line drawn between the last stored point and the current point (Schwan (2011)).

When such decompressed data is used for prediction, the target variable carries compression artifacts that can be more or less serious depending on the tolerance used (see Figure 1.4). Therefore, interpreting predictive performance on such data must be carried out carefully considering that results on decompressed data can be quite different from the ones in raw true data.

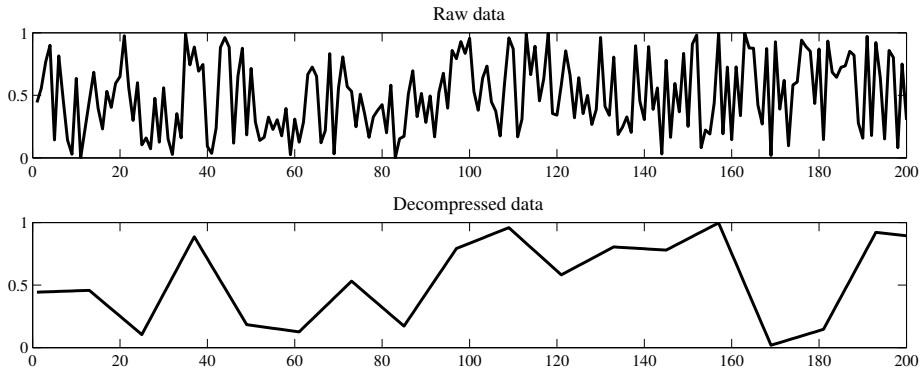


Figure 1.4: *Raw data (top) and decompressed data (bottom) belonging to the same time series. When decompressed data is used for prediction, the target variable carries compression artifacts that can be more or less serious depending on the tolerance used.*

Sampling rates

The sampling rate defines how often data is collected from the original source (e.g. every second). Ideally, this rate should be the same for each variable of the process, but in

[5]Scopus results for query "big data" (from only 31 documents in 2010 to 10341 in 2016): `http://bit.ly/NnUoam`

[6]McNulty E. Understanding Big Data: The seven V's `http://dataconomy.com/2014/05/seven-vs-big-data/`

reality each variable can have a different sampling rate (i.e. multi-rate systems). Even a sampling rate of a single variable can vary over time. Such scenario occurs in a system where some of the variables, usually critical for the process control, are evaluated in laboratories at much lower sampling rate than the rest of the automatically measured data. To work with useful data for process modelling, an expert usually defines how often samples should be taken (Kadlec et al. (2009)).

Measurement delays

In industrial processes data often comes from multiple sources, which may be separated physically (e.g. a long flow pipe) or virtually (e.g. data is stored in different ways), and need to be synchronised. Synchronisation of virtual sources requires consolidating the data into a single database or stream, and is relatively straightforward. Synchronising data from different physical locations is usually more challenging, and can be estimated based on the physical properties of the process (e.g. speed of flow), or approached as a computational feature selection problem, where for each sensor different time lags are tried as candidate features.

When an action is taken over the input of a process, it can take several minutes or even hours for its effect to reach the end of the process. These delays are sometimes difficult to measure and can vary with process load. Expert knowledge is recommended to check the correctness of the delays. A good synchronisation of data will determine the performance of the model learnt (Kadlec et al. (2009)).

Drifting data

Conventional machine learning algorithms often assume that all the samples in a dataset come from the same source and follow the same distribution. However, data from on-line processes can vary their distribution over time. The change of state of a process is referred to as *concept drift* (i.e. change of the relationship between input variables and target – Widmer & Kubat (1996)). The state of a chemical process can change due to external conditions (e.g. degradation of sensors) or internal conditions (e.g. change of the temperature due to reactions of a chemical process). These changes have a direct effect on the data distribution although this could be gradual (i.e. slow) or sudden (i.e. quick). In the last years, there has been an increasing interest in the literature to deal with this problem (see Gama et al. (2014) for a survey).

1.4 Soft sensors

Processing plants have a large number of sensors that measure physical properties in different parts of the process. Values such as temperatures, pressures or humidity are easy to capture. However, acquiring other measurements is more expensive and often require human interaction. For instance, measuring the product concentration may require taking a sample and analysing it in the laboratory.

In order to improve production efficiency, a predictive model could deliver estimates of such hard-to-measure values based on the process state given by the easy-to-measure values from the sensors. This type of predictive models are called *soft sensors* because they can be seen as software or virtual sensors instead of physical.

There are two main trends for building a soft sensor. The classical one in which a first principle model (FPM) is built following the chemical and physical principles of the process (e.g. exothermic equations). The main disadvantage of this method is the requirement of a lot of expert knowledge. In addition, sometimes it is not even possible to model the whole chemical process. Another drawback of FPMs is that they usually focus only on the steady-state of the process and do not model the transient states. In contrast, data-driven models can be built by using historical records only, thus requiring little domain knowledge. This thesis considers only this later type of soft sensors as they have more potential to reduce human interaction during their composition and adaptation.

Although the most common application of soft sensors is online prediction, others include process monitoring, fault detection and sensor backup. In any of them, the main requirements of soft sensors in process industry are:

- reliability – to provide truthful results;
- robustness – to work under any circumstances or inconvenience; and
- transparency – to be comprehensible by human experts.

1.4.1 Building soft sensors

The framework developed as part of this research, and presented in Budka et al. (2014), describes the soft sensor development process in four steps:

1. setting up the performance goals and evaluation criteria;
2. exploratory data analysis;
3. data preparation and preprocessing:
   (a) data acquisition (deciding which data to collect and from which sources);
   (b) data preprocessing (de-noising, handling outliers and missing values);
   (c) data reduction (extracting relevant representation);
4. training and validating the predictive model.

This thesis focuses mainly on the automation and adaptation of steps 3 and 4, which are further described below.

Data preparation and preprocessing

In industrial processes real-time data processing is typically required. For autonomous operation, preprocessing needs to be performed online, and design decisions need to be verifiable. Therefore, the procedure and order of preprocessing actions need to be well defined. The seven-step data preprocessing process proposed in Budka et al. (2014) is as follows (see Figure 1.5):

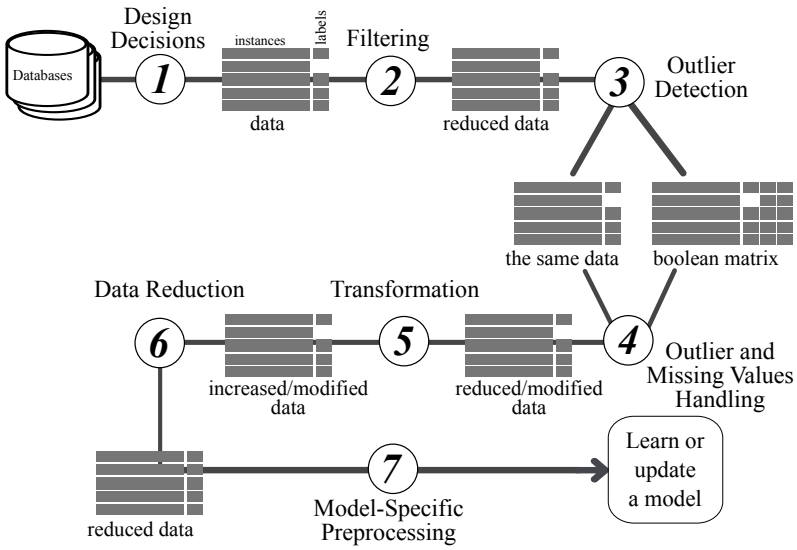


Figure 1.5: *The proposed order of data preprocessing steps.*

1. **Design decisions**: The first step defines the data design decisions, such as how data is queried from databases, how data sources are synchronised, at what sampling rate data arrives, whether any filtering (de-noising) is used. This step produces raw data in a matrix form, where each row corresponds to an observation of sensor readings at one point in time. Typically, the design choices remain fixed.
2. **Filtering**: The second step filters out irrelevant data. For example, data during plant shutdown times and non-steady operation states can be discarded as presented for instance in Appendix B. Rules for detecting such periods can be defined by experts during the design step, or statistical change detection techniques could be used to identify them automatically.
3. **Outlier detection**: The third step detects outliers in three stages. Firstly, recordings are checked against physical constraints (i.e. obvious outliers). Secondly, univariate statistical tests can detect outliers in individual sensors. Thirdly, multivariate statistical tests on all variables together (see e.g. Chandola et al. (2009)) can detect outliers at an observation level.
4. **Outlier and missing values handling**: In the fourth step the identified outliers and missing values are handled. In industrial applications predictions are needed continuously, therefore removing observations with missing values is typically not an option. Standard missing value imputation techniques can be used, ranging from simple methods like replacing by the last observed value or the mean value, to various model based imputation methods (see e.g. J. A. Little & B. Rubin (2002)). The result is a modified data matrix, usually of the same size as the original or, in some other cases, with additional attributes to denote if values were imputed (Budka et al. (2010)).
5. **Transformation**: The fifth step performs data transformations, which can modify the existing features (e.g. discretisation), derive new features (e.g. an indicator if the production is running), scale the data or rotate the data for de-correlation

purposes. The result of this step is a data matrix that can have more or the same number of features than before and the same number of observations.

6. **Data reduction**: The sixth step reduces data by feature selection (or extraction) and observation selection (subsampling). As a result the data matrix will decrease in size.
7. **Model specific preprocessing**: The seventh step performs model specific preprocessing, such as further removing of outliers or undesired observations. This completes data preprocessing and the next step is model training.

While the design decisions (step 1) must be made, other steps (2-7) are optional. It is up to a data scientist to decide which particular techniques to use while taking into account possible model limitations. Though the proposed order of steps is based on previous experiences (Sharmin et al. (2006); Lin et al. (2007); Kadlec et al. (2009)), other authors follow a slightly different order (e.g. Pani & Mohanta (2011) do feature selection at an earlier stage). Nonetheless, it is recommended to keep the order within the same experimental study since it enables reproducibility (Peng (2011)), allows easier documentation, and easier automation when it comes to implementation.

Training and validating the predictive model

Once the data has been preprocessed, the next step is to build a predictive model. To this end, machine learning algorithms are fed with a training set. The resulting model is validated using a test set. This validation is done to assess the generalisation error of the model. That is, its ability to predict the target value of unseen instances. The next chapter is dedicated to predictive modelling where this aspect will be discussed at a greater detail.

A common practice to evaluate models is holdout testing, where a dataset is split into two parts: training and testing. The model is built using the training set and then evaluated over the testing set. However, in some cases the lack of sufficient amount of historical data can lead to a bad estimation of the generalisation error. For that reason, a good practice is to use k-fold cross-validation (see Figure 1.6), where a dataset is split randomly into k folds of approximately the same size. Then $k-1$ folds are used to train a model and the remaining one to test it. This process is repeated k times and then results are aggregated. This technique gives also an indication of model robustness by looking at the standard deviation and range of the k results. This technique is useful to avoid the overfitting problem, where model is very well fitted to the training data, but generalises poorly to new, unseen data (see Figure 1.7). Usually, the dataset is shuffled, so instances are randomly distributed across the folds. However, randomising process industry datasets where instances are timestamped is not recommended since the trends over time would be removed (Gama et al. (2012)).

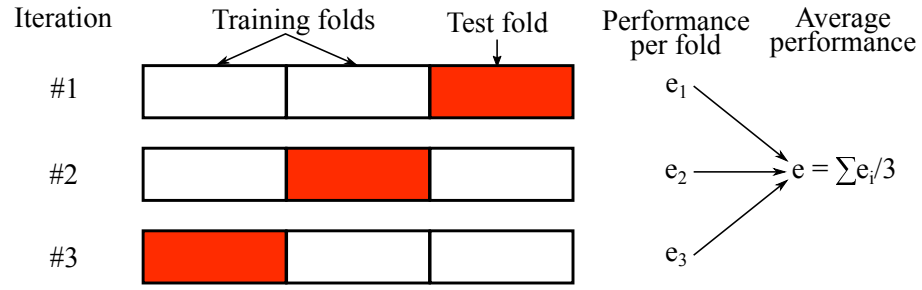


Figure 1.6: *3-fold cross validation. A dataset is split randomly into k folds of approximately the same size. Then $k-1$ folds are used to train a model and the remaining one to test it. This process is repeated k times and then results are aggregated.*

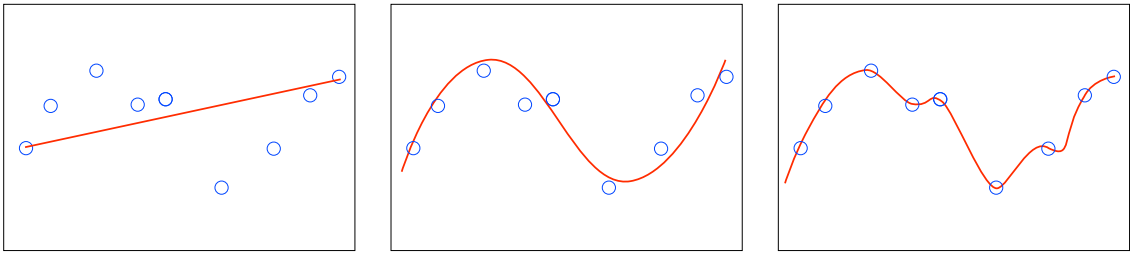


Figure 1.7: *Illustration of three cases of model fitting in a regression problem: Underfitting (left) when model does not capture well data relationship; Overfitting (right) when model is very well fitted to the training data, but generalises poorly to new, unseen data; and ideal fitting (middle) when there is a good balance between fitting and generalisation.*

1.4.2 Maintaining soft sensors

Chemical production plants are rarely stable from the point of view of the process. Internal factors such as chemical reactions affect for example the temperature in a vessel and therefore data representing the process state will drift as explained previously in Section 1.3.2. Potential problems resulting from these factors can be mitigated by plant operators (e.g. cooling down the vessel). On the other hand, external factors such as weather conditions or quality of the raw materials can also produce a drift in the process state that could be more difficult to detect and to act upon (e.g. recalibration of the sensors might be needed).

The drifts are likely to affect the deployed soft sensors since they heavily rely on the input data and known concepts acquired from historical data during training. Thus, adaptation of soft sensors is a key aspect for them to be successful. There is a number of adaptive soft sensors using different techniques (see Kadlec et al. (2011) for a review), but there is no one that fits all purposes. Main techniques include relearning a new model, parametric adaptation of existing model, and adaptive ensembles, among others. This thesis investigates not only adaptation mechanisms for predictive models, but also

for the whole predictive workflow including preprocessing and postprocessing methods. Chapter 5 and Appendix C are dedicated to this problem.

1.5 Thesis goals

This thesis will focus on data preprocessing for predictive modelling. The aim is to study the feasibility of automating the composition and optimisation of workflows to make accurate predictions on unseen data. Though there have been approaches for automatic algorithm selection and hyperparameter optimisation of predictive models (e.g. Bengio (2000); Hutter et al. (2011); Bergstra & Bengio (2012); Thornton et al. (2013)), this thesis is the first one in addressing such problem in complex predictive workflows.

In addition, due to the evolving nature of many data streams, another goal is to plan a maintainability strategy of predictive systems in production environments. There are works in the literature presenting adaptive models (e.g. Joe Qin (1998); Kadlec & Gabrys (2009); Kadlec et al. (2011); Stahl et al. (2012)), however, there is a lack of research when tackling the problem of adapting complete workflows (Žliobaitė et al. (2012); Krempl et al. (2014)).

To achieve these goals three major objectives need to be addressed:

1. Propose, design and evaluate a framework for connecting multiple components to compose valid predictive workflows including preprocessing methods, machine learning models, and postprocessing operations;
2. Develop and evaluate a smart data-driven mechanism to automate the creation of such workflows with minimum human intervention;
3. Develop and evaluate an approach of adapting predictive workflows in changing environments.

The thesis is organised around the objectives, which are evaluated with several benchmark datasets. In order to validate the feasibility of the developed approaches in real problems of process industry, a number of datasets were provided by the industrial partner (Evonik Industries) from real chemical processes.

1.6 Original contributions and list of publications

The original contributions of this work are:

1. Framework for data preprocessing to develop soft sensors in process industry (Chapter 1).
2. Novel formalism of predictive workflows as Multicomponent Predictive Systems (MCPSs) using Petri nets (Chapter 3). This links with objective 1.
3. New extension of Auto-WEKA software to automate the composition and optimisation of MCPSs (Chapter 4). This fulfils objective 2.
4. Experimental comparative study of search strategies for finding the best MCPSs in

different search spaces (Chapter 4).

5. Novel hybrid strategy for adapting MCPSs to cope with concept drift (Chapter 5). This satisfies objective 3.
6. Extension of MCPS definition to deal with change propagation when individual components are adapted (Appendix C).
7. Novel method for online identification of shutdown periods of chemical plants (Appendix B).

The following peer-reviewed conference and journal publications are a result of this work:

1. M. Martin Salvador, M. Budka, B. Gabrys. 'Automatic composition and optimisation of multicomponent predictive systems'. Submitted to *IEEE Transactions on Automation Science and Engineering* (under review).
2. M. Martin Salvador, M. Budka, B. Gabrys. 'Modelling Multi-Component Predictive Systems as Petri Nets'. Accepted in *15th Annual Industrial Simulation Conference (ISC 2017).*
3. M. Martin Salvador, M. Budka, B. Gabrys. 'Effects of change propagation resulting from adaptive preprocessing in multicomponent predictive systems'. In *20th International Conference on Knowledge Based and Intelligent Information and Engineering Systems (KES 2016). Procedia Computer Science. Elsevier.*
4. M. Martin Salvador, M. Budka, B. Gabrys. 'Adapting MultiComponent Predictive Systems of Chemical Production Processes with Bayesian Optimisation'. In *AutoML 2016 at 33rd International Conference on Machine Learning (ICML 2016). Journal of Machine Learning Research.*
5. M. Martin Salvador, M. Budka, B. Gabrys. 'Towards Automatic Composition of Multicomponent Predictive Systems'. In *11th International Conference on Hybrid Artificial Intelligence Systems (HAIS 2016). Lecture Notes in Artificial Intelligence. Springer.*
6. M. Budka, M. Eastwood, B. Gabrys, P. Kadlec, M. Martin Salvador, S. Schwan, A. Tsakonas, I. Žliobaitė. 'From Sensor Readings to Predictions: On the Process of Developing Practical Soft Sensors'. In *Advances in Intelligent Data Analysis XIII (IDA 2014). Lecture Notes in Computer Science. Springer.*
7. M. Martin Salvador, B. Gabrys, I. Žliobaitė. 'Online Detection of Shutdown Periods in Chemical Plants: A Case Study'. In *18th International Conference in Knowledge Based and Intelligent Information and Engineering Systems (KES 2014). Procedia Computer Science. Elsevier.*
8. F. Stahl, M. Medhat Gaber, M. Martin Salvador. 'eRules: A Modular Adaptive Classification Rule Learning Algorithm for Data Streams'. In *32nd SGAI International Conference in Innovative Techniques and Applications of Artificial Intelligence (AI 2012). Research and Development in Intelligent Systems XXIX. Springer.*

1.7 Organisation of the thesis

The structure of the thesis is presented in Figure 1.8. This introduction has presented the most common issues found in the development of predictive models in the process industry. This particular field is a source of a set of case studies for this thesis, although this research is also useful in many other application areas of machine learning.

Chapter 2 introduces the fundamentals of building predictive systems both from a theoretical and practical perspective. Popular machine learning approaches for classification and regression problems, as well as preprocessing methods for cleaning and transforming data, are reviewed.

Chapter 3 starts by describing Petri nets, that serve as the basis to formalise multicomponent predictive systems (MCPSs) as a novel extension of Petri nets. The problem of composing and optimising MCPSs is also considered for the first time in this chapter. An instance of this problem is presented in Appendix B, which includes a study of the manual process needed to build an MCPS for a regression problem requiring online filtering of shutdown periods from a chemical production plant.

In order to speed up these labour-intensive processes, Chapter 4 includes a critical literature review of the state-of-the-art approaches in composing and optimising predictive models. The extension of Auto-WEKA – a tool for finding the best predictive model for a given dataset – to support MCPSs is also presented in this chapter. Experiments on a number of publicly available datasets demonstrate that it is possible to automate the composition of MCPSs using data-driven techniques without human intervention.

Chapter 5 shows the experiments of applying these techniques to process industry datasets (described in Appendix A) as well as considering the maintenance of the generated MCPSs. To this end, several adaptation strategies have been evaluated demonstrating the feasibility of this research for a real production environment. Nonetheless, if instead of adapting a complete MCPS one would like to update only one component there are a number of things that should be considered. A preliminary discussion on such interesting topic is presented in Appendix C.

Finally, Chapter 6 concludes this thesis by discussing the most significant findings and possible directions for further research.

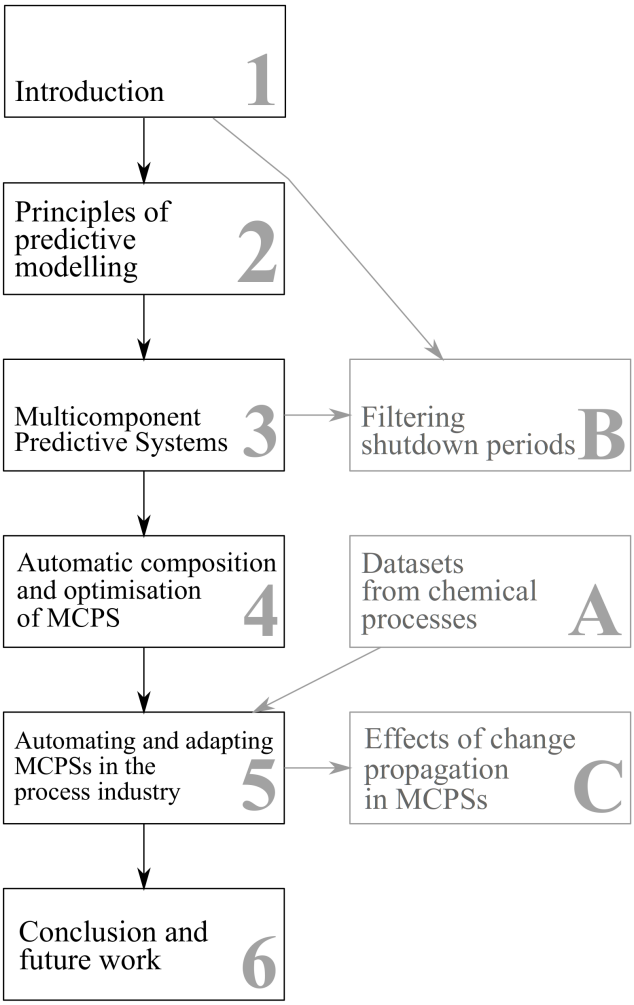


Figure 1.8: *Structure of the thesis and chapter dependencies*

Chapter 2

Principles of predictive modelling

2.1 Introduction

From clairvoyants that claimed to guess a person's love life to ancient Egyptians predicting Nile river floodings, humans have craved the ability to predict the future. However, while some constantly failed on their predictions based on a crystal ball, others were able to give quite accurate dates which helped to increase crop production based on past experiences and positions of the stars (Bell (1970)).

With the development of computer science and artificial intelligence, we are now able to make quite reliable predictions in a large number of fields (Witten & Frank (2005)). In machine learning, three main elements are involved in developing such predictive ability: a) a set of historical data, b) a learning algorithm to create a predictive model, and c) the assumption that future data is similar to data that has been already observed.

The procedure of building predictive systems is known as predictive modelling, and is part of a larger overarching process called data mining. Fayyad & Piatetsky-Shapiro (1996) introduced the knowledge discovery in databases (KDD) process which became the seminal work on structuring the data mining process. There have been many efforts in the data mining community to standardise this process (Marban et al. (2009)). The de-facto standard is the Cross Industry Standard Process for Data Mining (CRISP-DM – Shearer (2000)) which is an iterative process consisting of six steps (see Figure 2.4): business understanding; data understanding; data preparation; modelling; evaluation; and deployment. The outer circle of Figure 2.4 symbolises the cyclical nature of data mining itself. This chapter introduces the theory behind predictive systems in Section 2.2, followed by a description of CRISP-DM steps in Section 2.3, with an emphasis on data preparation (Section 2.4) and modelling (Section 2.5) as they are the main topics addressed in this thesis.

2.2 Predictive systems

Let $f : \mathcal{X} \rightarrow \mathcal{Y}$ be an unknown process that maps data from a d-dimensional input space $\mathcal{X}$ into a c-dimensional output space $\mathcal{Y}$ such as

$$y = f(\boldsymbol{x}) + \epsilon \tag{2.1}$$

where $y \in \mathcal{Y}$ is the output or target value, $\boldsymbol{x} \in \mathcal{X}$ is the input vector, and ϵ is the noise term which is often assumed to follow a normal distribution.

It is often the case that $\boldsymbol{x}$ is available but y is not. For example, $\boldsymbol{x}$ could be the results of a blood test (easy to acquire) and y could be the likelihood of having a particular disease (difficult to know). Therefore it would be very desirable to be able to predict y on the basis of $\boldsymbol{x}$. To this end, a predictive model $\hat{f}$ is built to approximate the unknown process f based on past observations. The estimated output for an input $\boldsymbol{x}$ is then given by

$$\hat{y} = \hat{f}(\boldsymbol{x}) \tag{2.2}$$

The data-driven process of building a predictive model is known as learning. This thesis considers the case of supervised learning, where a dataset made of instances $(\boldsymbol{x}, y)$ is available. The predictive model

$$\hat{f} = A_{\boldsymbol{\lambda}}(\mathcal{D}_{train}) \tag{2.3}$$

is built using a learning algorithm A with a set of hyperparameters $\boldsymbol{\lambda} = \{\lambda_1, ..., \lambda_m\}$ and a training dataset

$$\mathcal{D}_{train} = (X_{train}, \boldsymbol{y}_{train}) = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)\} \tag{2.4}$$

There exist a wide variety of learning algorithms (see e.g. Witten & Frank (2005) for a taxonomy) and most of them have user-tunable hyperparameters that modify their learning behaviour (e.g. number of hidden layers, number of neurons or learning rate of a neural network).

Depending on the type of the target variable one can find two types of predictive problems:

- **Classification** (when y is discrete): A classification problem is concerned with labelling of an entity based on its characteristics. For instance, a mushroom can be classified as poisonous or edible depending on its colour, shape and odour (Schlimmer (1987)). Given a dataset with instances made of mushroom properties and edibility, one can build a model (i.e. a classifier) that divides the input space into different regions for explaining the relationship between them. Therefore, the classifier would be able to predict the edibility of a new unknown mushroom solely on the basis of its characteristics. See Figure 2.1 where a decision tree has been built using the ID3 algorithm (Quinlan (1986)) on the mushroom dataset from UCI ML

repository [1].

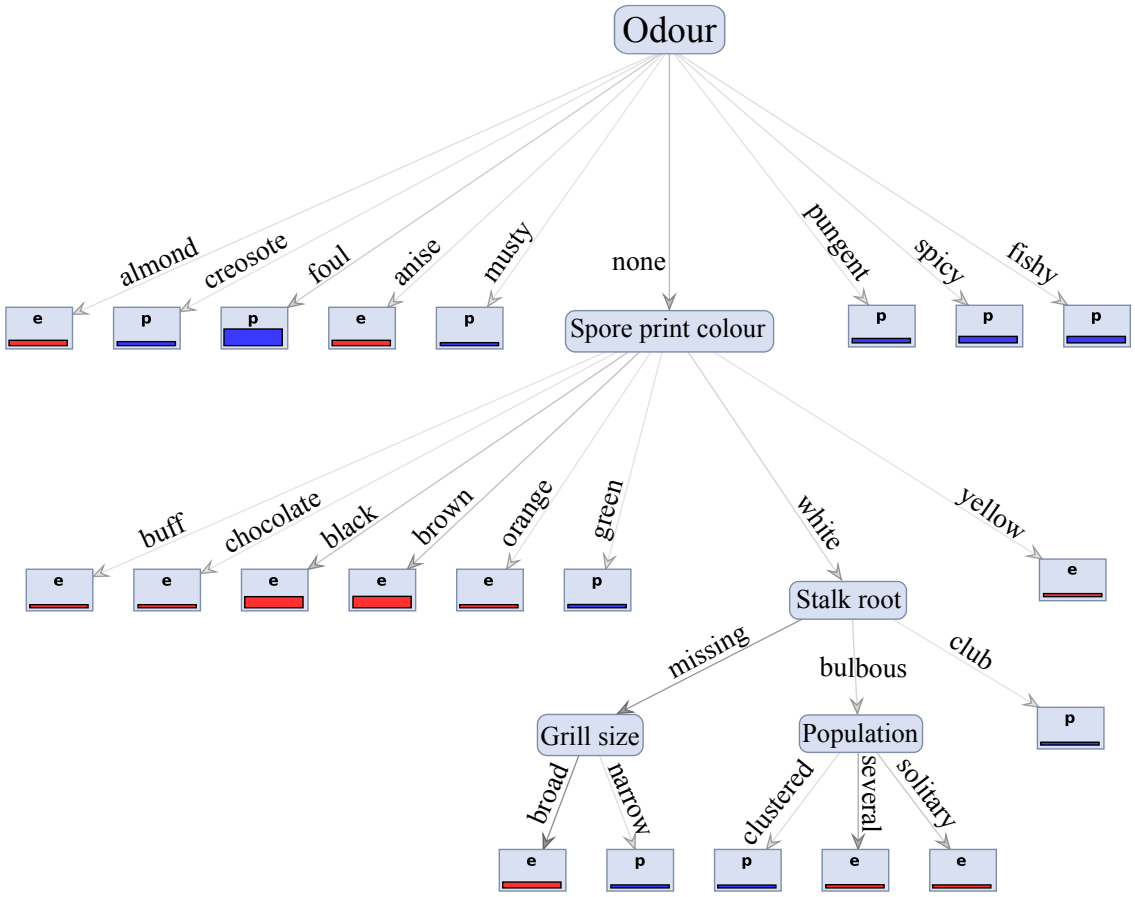


Figure 2.1: *Decision tree of mushroom dataset (e = edible, p = poisonous) generated using ID3 algorithm in RapidMiner. According to the tree, the first feature to inspect in a mushroom should be the odour. For instance, all the mushrooms in this dataset with foul smell are poisonous. If it has no odour, the next feature to look at is the spore print colour. If it is green, the mushroom is more likely to be poisonous as well. Exploring the decision tree in that way, one would be able to classify any mushroom accordingly.*

- **Regression** (when y is continuous): The goal of a regression task is to fit a function between the input $\boldsymbol{x}$ and the continuous target $y \in \mathbb{R}$. For example, a tree age can be estimated based on growth rings by fitting a regression model (Fraver et al. (2011)). The most basic method is a linear regression which assumes a linear relationship between the input features and the target value. See for instance Figure 2.2 which shows a linear model $y = 1.18x - 226.84$ fitted by the least squares method using a dataset[2] of tree girth measurements and age estimations of ancient oaks.

[1] `https://archive.ics.uci.edu/ml/datasets/Mushroom`
[2] `http://wbrc.org.uk/atp/EstimatingAgeofOaks-WoodlandTrust.pdf`

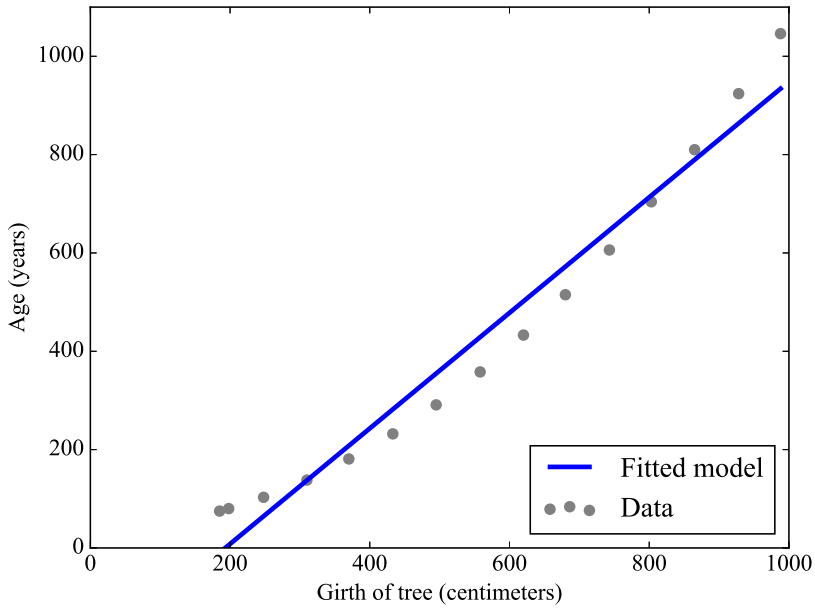


Figure 2.2: *Linear model* $y = 1.18x - 226.84$ *fitted on a 1-dimensional example using least squares method on a dataset of tree girth measurements and age estimations of ancient oaks.*

At a high level of abstraction, both classification and regression problems share the purpose of predicting a value, however they are conceptually very different problems. The regression problem is about finding the optimal relationship between the input and the output, while in a classification problem the aim is to separate the data into classes (see Figure 2.3).

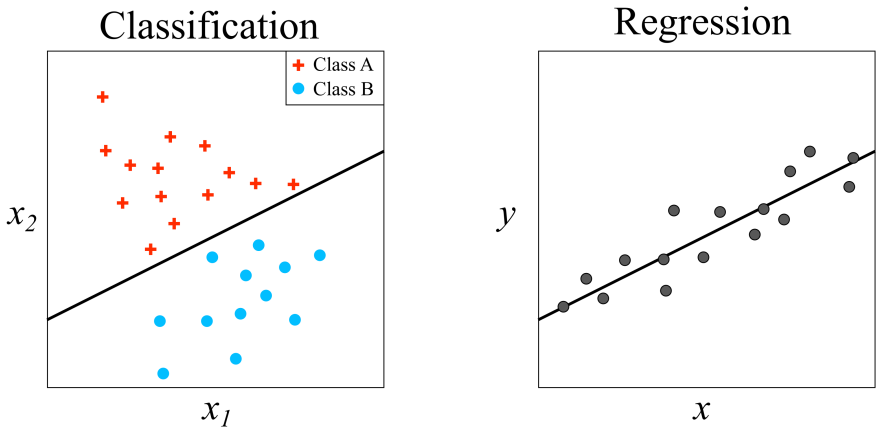


Figure 2.3: *Toy examples of classification (left) and regression (right) problems*

In order to assess the model performance, a loss function representing the cost of

wrong or inaccurate predictions is required. This function $\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$ calculates the loss of the predictions $\hat{\boldsymbol{y}}$ given the correct target values $\boldsymbol{y}$. In the classification problems, a common loss function is the classification error given by

$$\mathcal{L}_{CE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = 1 - \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{\hat{y}_i = y_i\} \quad \in [0, 1] \tag{2.5}$$

where $\mathbb{I}$ is the indicator function which returns 1 if the condition is true or 0 otherwise. For regression problems it is common to calculate the mean absolute error (MAE) given by

$$\mathcal{L}_{MAE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i| \tag{2.6}$$

or the mean squared error (MSE) given by

$$\mathcal{L}_{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{2.7}$$

which penalises large deviations of the predicted values. To facilitate the interpretation of MSE, it is usual to take the root square such that

$$\mathcal{L}_{RMSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sqrt{\mathcal{L}_{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y})} \tag{2.8}$$

since it has the same units as the target variable.

The goodness of fit, also known as training accuracy, measures how well the model has captured the relation between the input and output of the training instances. Therefore, from the predictiveness point of view, the best model would be the one that minimises the loss function for a given dataset

$$f_{min} = \arg\min_{f} \mathcal{L}(\hat{f}(X), \boldsymbol{y}) \tag{2.9}$$

However, a model having a very good training accuracy may face the problem of overfitting (see Figure 1.7). That is, the model's parameters are very well adjusted to the training data but generalise poorly to new data.

Since it is not possible to calculate the performance on future data, the generalisation error has to be estimated from the available training data. There are several ways to estimate this error, with the following being the most common:

- **Hold-out estimation**, where the training dataset is split into a training and a validation part. The model $\hat{f}$ is built using the new training set and then tested on the validation set, which gives an estimation of its generalisation error.

$$err = \mathcal{L}(\hat{f}(X_{test}), \boldsymbol{y}_{test}) \tag{2.10}$$

- k**-fold cross-validation**, where the hold-out method is repeated k times using $k-1$

equally split folds for training and the remaining fold for testing (see Figure 1.6). The generalisation error is then estimated by aggregating the predictions of the resulting k models as

$$err = \frac{1}{k}\sum_{i=1}^{k} \mathcal{L}(\hat{f}(X_{test}^{(i)}), \boldsymbol{y}_{test}^{(i)}) \tag{2.11}$$

- **Bootstrapping**, where the hold-out method is repeated k times, but the training and validation sets are created in a different way. A new training set is populated with instances randomly sampled with replacement from the original training set. The validation set is formed of the remaining instances that have not been sampled. One of the most popular estimators is .632 bootstrap (Efron & Gong (1983)) given by

$$err = \frac{1}{k}\sum_{i=1}^{k} 0.632 \cdot \mathcal{L}(\hat{f}(X_{test}^{(i)}), \boldsymbol{y}_{test}^{(i)}) + 0.368 \cdot \mathcal{L}(\hat{f}(X_{train}^{(i)}), \boldsymbol{y}_{train}^{(i)}) \tag{2.12}$$

Depending on the availability of the target value in the training set, the learning algorithms can be classified as: a) supervised, when all the instances include the target value; b) unsupervised, when no target value is available; and c) semi-supervised, when only some instances have associated target values. Although only supervised learning algorithms are considered in this thesis, this research could naturally be extended to the other paradigms.

2.3 Designing a predictive system

This section describes the six steps of CRISP-DM process (Shearer (2000)) aiming to build a data-driven predictive system which can be used in a production environment to deliver online predictions. See Figure 2.4 where flow and dependencies between steps are illustrated.

2.3.1 Business understanding

The first phase of the CRISP-DM process consists of 4 steps: 1) Identify the business objectives (e.g. keep customers that are likely to churn); 2) Assess the situation, that is, what are the resources, constrains and other factors that should be considered for creating a working plan; 3) Determine the data mining goals (i.e. project objectives from the technical point of view, like predicting customer's churn rate with 80% accuracy); and 4) Produce the project plan with tasks that will guide the rest of the process. The plan should also include the tools and techniques to support each stage of the process. The plan could be reviewed and updated in the future if needed.

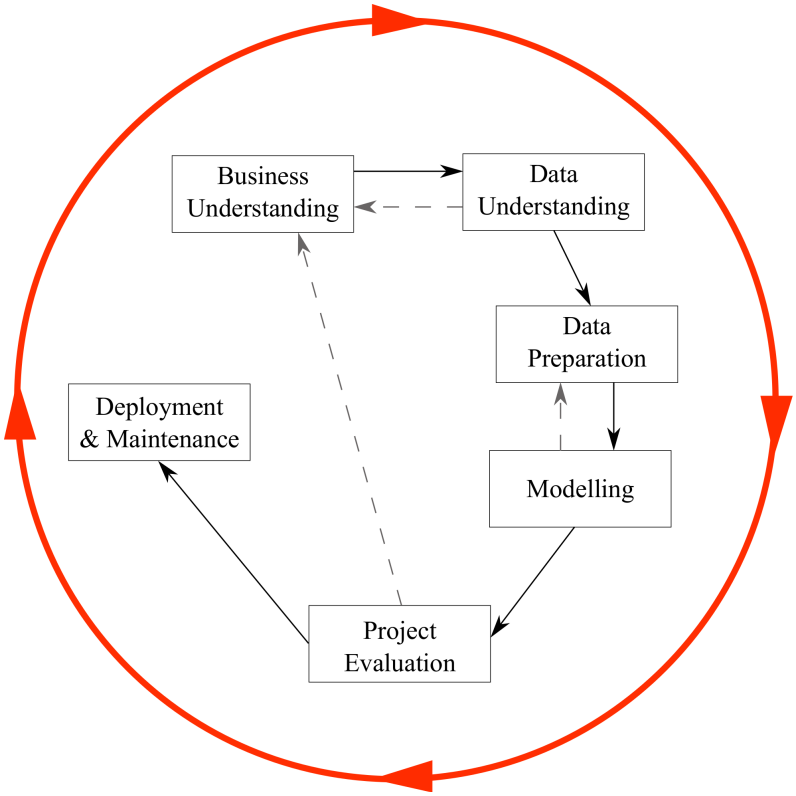


Figure 2.4: *Phases of the CRISP-DM process. The dependencies between the steps are represented by solid directed lines. The dashed directed lines indicate loops that are often required in order to refine the outcomes. The outer circle represents the iterative nature of the process. Figure inspired by Figure 1 from Shearer (2000).*

2.3.2 Data understanding

CRISP-DM process is usually applied to data available in some storage system (e.g. a relational database) from which a set of instances is selected. In the case of structured data, the selected records form a dataset in which usually each column represents an attribute and each row an instance. On the other hand, if data is unstructured, a dataset can be seen as a collection of items (e.g. images).

This phase often requires domain knowledge to understand what is the meaning of the variables and what is the target value to be predicted. One could start by performing a preliminary exploratory analysis including checking the type of attributes (e.g. continuous, categorical), basic statistics (e.g. range, mean, standard deviation) and attribute units (e.g. minutes, meters). Plotting the data can help to find existing relationships between variables. Also data visualisation tools like boxplots and scatter plots can highlight obvious data imperfections that could be addressed in the next phase.

2.3.3 Data preparation

As shown in Section 1.3, poor quality of raw data may lead to building poor models (Pearson (2005)). Often data is also collected from multiple sources, thus the implications of merging such data should be taken into consideration (e.g. redundancy and inconsistencies as seen in Section 1.3.2). During the data preparation phase, multiple methods for cleaning and repairing the data are applied. This includes a range of preprocessing operations such as noise removal, missing values replacement and feature selection. In case of a classification problem, this phase can also include other data transformations to make data linearly separable. Since data preparation is one of the main focuses of this thesis, preprocessing methods for dealing with common issues are described in Section 2.4. The output of this stage is a dataset that will be used for building and evaluating the predictive system.

2.3.4 Modelling

In the modelling phase, data mining experts select several machine learning models for a comparative study. Model selection is usually a manual process based on the available data and expert bias. The data prepared in the previous phase is used to train and evaluate models tyically using one or a combination of main techniques to estimate the generalisation performance described in Section 2.2. The overwhelming number and variety of models available in data mining toolboxes, as well as the multitude of their associated hyperparameters, make this process labour and compute intensive. Automation of this iterative process has been considered by different authors (e.g. Thornton et al. (2013); Feurer et al. (2015); Sparks et al. (2015)).

The original CRISP-DM process considers data preparation and modelling as independent phases. This can lead to a situation in which preprocessing is a one-off step for all the evaluated models. However, different models can benefit from, or even require, different data preprocessing. As a consequence, one can form a workflow connecting multiple preprocessing methods and models. Chapter 3 propose a new formalism to this type of predictive workflows and Chapter 4 presents how to jointly automate preprocessing and modelling. Section 2.5 unfolds the modelling phase and describes the four main steps: test design; algorithm selection; hyperparameter optimisation; and model assessment. The output of this phase is a summary of the generated models ranked by their predictive performance.

2.3.5 Project evaluation

In this phase, the results of the experimentation are evaluated. The generated models are assessed to determine if they meet the business objectives and the data mining goals defined in the previous phases. The most appropriate model fulfilling those objectives is selected for deployment. Otherwise, if the results are not conclusive or do not achieve the objectives, the process is reviewed and a plan is made to refine it (e.g. selecting additional

data, or applying other preprocessing methods).

2.3.6 Deployment and maintenance

The final phase of CRISP-DM consists of deployment of the predictive model in a production environment. A deployment plan has to be elaborated, including also aspects such as monitoring and maintenance of the system. The motivation to adapt a predictive system is clear in evolving environments where input data is continuously changing (Widmer & Kubat (1993)). Nevertheless, maintenance is also needed in, a priori, static systems where data distribution is expected to be stable. The reason behind it is that there are external factors than can affect model performance. For instance, consider a system where data is recorded by physical sensors. The accuracy of these sensors can degrade over time (e.g. due to wear and tear) and therefore the data distribution will change despite of the underlying process remaining the same. Such changes between the input features and the target variable are known as concept drift – as previously introduced in Section 1.3.2 – and can cause a decrease in the model performance (Widmer & Kubat (1996)). There exist various approaches to cope with concept drift, ranging from manual model adjustments to using adaptive learning algorithms (see Gama et al. (2014) for a survey). Common maintenance operations include hyperparameter tuning, model retraining, additional preprocessing or even hardware upgrades to deal with scalability issues. Chapter 5 describes the adaptation strategies that have been used in this thesis to deal with such changes in predictive systems.

2.4 Data preparation

Data preparation can take as much as 60-80% of the total time spent on developing a predictive model (Munson (2012)). A large fraction of this is due to the need of integrate data from a number of disparate systems and the operations for converting raw data into a suitable training dataset. These operations are usually applied in the following order: data filtering $\rightarrow$ data cleaning $\rightarrow$ data enrichment $\rightarrow$ data transformation, as shown in Figure 2.5. This section describes these four phases in the context of process industry since it has been the main driver behind INFER project.

2.4.1 Data filtering

Data from process industry is often abundant. The complexity of plants and redundancy of sensors contribute to a large number of input variables in the data, potentially introducing a number of issues due to high dimensionality as discussed in Section 1.3.2. Reducing the dimensionality not only helps to relieve these problems, but having fewer attributes also improves the interpretability of the learnt model (Han et al. (2011)). Typical techniques for dimensionality reduction include wavelet transforms (e.g. Keogh et al. (2001); Bruce & Koger (2002); Qu et al. (2003)), principal component analysis (e.g.

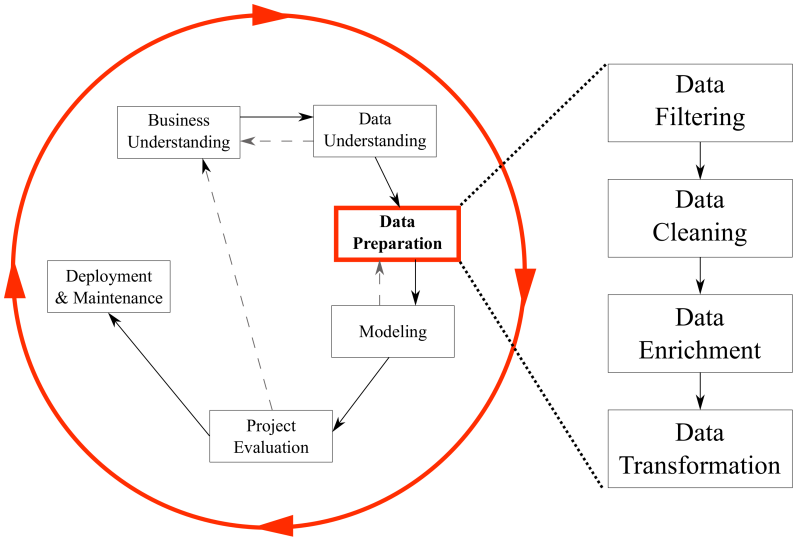


Figure 2.5: *Data preparation phases of the CRISP-DM process*

Pearson (1901); Kambhatla & Leen (1997)) or feature subset selection (e.g. Mladenić (2006); Wei & Billings (2007)), among others. A review of dimensionality reduction techniques can be found in van der Maaten et al. (2009).

Attribute redundancy and multicollinearity can be a problem as shown in Section 1.3.2 and can be mitigated by dimensionality reduction techniques. Some redundancies in attributes can also be detected by correlation analysis. This analysis can measure how strongly one attribute determines the values of another based on the available data, assuming a liner relationship between the two. A typical method for correlation analysis in nominal data is the χ^2 test (Bai & He (2004)), while correlation coefficient and covariance are commonly applied in numerical data (Pearson (2005)).

A common source of data redundancy and inconsistency is the integration of different data sources (e.g. Dawyndt et al. (2005)). To this end, approaches like Hernández & Stolfo (1998) run an iterative approach for cleansing very large databases. On the other hand, other authors propose to address the redundancy before the integration is actually performed (e.g. Calì et al. (2005) find semantic relations between the data sources).

In addition, datasets can contain instances that are not meaningful from the prediction point of view. For example, some sensors still record data even if the plant is not running. These instances can bias the model and lead to worse predictions. Removing these instances can be done by exploring the data manually or by applying change-detection methods as the ones reviewed in Appendix B.

2.4.2 Data cleaning

The purpose of this step is to raise the data quality by polishing imperfections. To this end, several cleaning tasks are applied to address the issues found during the data exploration phase.

Feature alignment/synchronisation

As pointed out in Section 1.3.2, there might be some variables which values are shifted in time for several reasons. A common method in chemical production plants to identify such delays between variables is to provoke a sudden change in the process (Kadlec et al. (2009)). Then, the elapsed time between the changes in the sensors is measured. Nevertheless, this is a quite disruptive procedure that ideally would be avoided. Another method is Dynamic Time Warping (DTW – Berndt & Clifford (1994)) which calculates the similarity between two time series and allows to minimise the misalignment between them.

Handling missing values

As shown in Section 1.3.2, some instances can contain missing values due to different reasons. Classification algorithms such as support vector machines (SVM) and neural networks (NN) are particularly affected by the presence of missing values since most discriminative learning approaches have no natural ability to deal with missing input features (Marlin (2008)). There are three main approaches that one could take when working with missing data:

- **Nothing**, where no action is taken since some models are able to handle datasets with missing values (e.g. Gabrys (2002));
- **Deletion**, where instances with missing values are deleted. It is the simplest approach, but it can lead to loss of information. Also this approach is not feasible in certain systems when missing values occur during the online phase and a prediction must be always delivered (Žliobaitė & Hollmén (2014));
- **Imputation**, where missing values are replaced by an estimation. Approaches under this category include single imputation (i.e. a constant value such as the attribute mean, e.g. Allison (2001)) and model imputation (i.e. a predictive model is built using non-missing data for imputing the missing values, e.g. Dudoit & Fridlyand (2002)).

Noise removal

Datasets can contain noisy values introduced as consequence of human error or due to interferences in the measurement equipment as described in Section 1.3.2. Noise is often associated to continuous values, but also can be present in other type of attributes. A noisy dataset can decrease the quality of the learnt model (Nettleton et al. (2010)) and therefore it is desirable to smooth noisy values.

One common noise removal technique is binning, where data is smoothed by consulting the surrounding values (i.e. local smoothing). Each value is assigned to a bin (i.e. consecutive number of values) and then replaced by an aggregated value of that bin (e.g. mean).

Another technique to reduce noise in continuous data consists on fitting a regression

model (see e.g. Reis & Saraiva (2004)) between different variables and then replace the noisy values in the affected attribute with a prediction given by the model. There is however no guarantee that the new value from a regression model will not be an outlier and therefore add even more noise.

Other approaches use discrete wavelet transform (DWT) for smoothing data by taking advantage of both frequency and temporal location of the values. For example, Slišković et al. (2011) apply a three step procedure for denoising sensor signals of a chemical plant. Firstly, DWT is used for decomposing the signal. Then non-significant values are filtered using a threshold. Finally, the signal is reconstructed by inverse of the DWT.

Principal component analysis (PCA) techniques have been also used for denoising. For example, Romero (2011) apply PCA and Independent Component Analysis (ICA) to reduce the noise in a multi-lead electrocardiogram. However, these methods are very sensitive to outliers. For that reason, robust PCA methods have been developed (e.g. Brubaker (2009)).

Other authors propose to characterise the noise and use such information to modify the existing predictive model instead of removing the noise from data. For instance, Wu & Zhu (2008) add prior knowledge about the noise of each variable by transforming the conditional probabilities in the Naive Bayes classifier.

Handling outliers

The consequences that outliers can have in predictive modelling were presented in Section 1.3.2. An extensive survey of outlier detection techniques and applications was presented by Chandola et al. (2009). Their proposed taxonomy is summarised below:

- **Classification**, where a classifier is built using training data and then used to categorise instances as normal or anomalous (e.g. Chaloner & Brant (1988); Yang (2005));
- **Nearest neighbours** methods which assume that normal data instances occur in dense neighbourhoods, while anomalies lie far from their closest neighbours (e.g. Östermark (2009); Kim et al. (2010)). These techniques require a distance or similarity measure defined between two instances;
- **Clustering** techniques, which can be grouped in three categories corresponding to the following assumptions: 1) normal data instances belong to a cluster in the data, while anomalies do not belong to any cluster (e.g. Ester et al. (1996)); 2) normal data instances lie close to their closest cluster centroid, while anomalies are far away from their closest cluster centroid (e.g. Emamian et al. (2000)); 3) normal data instances belong to large and dense clusters, while anomalies either belong to small or sparse clusters (e.g. He et al. (2003));
- **Statistical** anomaly detection techniques which assume that normal data instances occur in high probability regions of a stochastic model, while anomalies occur in low probability regions (e.g. Shewhart (1931); Parzen (1962));
- **Information theoretic** measures, such as Kolmogorov Complexity (Arning et al.

(1996)) or entropy (He et al. (2005)), are used to detect outliers assuming that anomalies lead to irregularities in the information content of a dataset;

- **Spectral** techniques assume that data can be embedded into a lower dimensional subspace in which normal instances and anomalies appear significantly different (e.g. Parra et al. (1996)).

Typically, the output of outlier detection techniques is a score or label within the instances. The score gives a value of how anomalous an instance is, hence analysts can use a threshold to select the outliers. On the other hand, if outliers are being detected at attribute level, a matrix is generated instead indicating what particular attribute in the instance is anomalous.

The way of dealing with outliers can vary depending on the characteristics of the dataset, as well as the application. Sometimes deleting outlier instances is an option, but in other cases replacement of outliers with other values such as mean, mode, maximum or minimum is more effective. Methods for replacing missing values are often used for handling outliers.

Although outliers are usually considered as something negative that should be removed, other applications try to look for outliers or rare events in order to analyse them. For example, if a system is detecting illegal access to a network, the abnormal situations are less frequent but more important. For dealing with this type of applications, different approaches have been proposed (e.g. Denning (1987); Bloch et al. (1995); Fawcett & Provost (1997)).

2.4.3 Data enrichment

Predictive accuracy highly depends on the selected features for building the model. Although the original set of features may contain sufficient information, this may not always be leveraged by the learning algorithm. A common approach to maximise the use of latent information is to generate new features (Cheng et al. (2011); Islamaj et al. (2006)). For instance, given the length and width of a plot of land as input attributes, one could create a new attribute with the area of the plot. While it does not add any new information to the dataset, it makes the existing information easier to exploit to some models (e.g. linear models are not able to pick up such multiplicative relationships). Similarly, given a date, one could calculate the day of the week (Monday to Sunday), the month and the year (Fawcett & Provost (1997)). Also, one could discretise a time attribute into intervals of the day (i.e. morning, afternoon, evening). In the prediction of multivariate time series it is common to generate new features including moving average of previous values or differences between different periods (De Silva et al. (2013)).

Most of the classification algorithms work well under the assumption that there is a similar number of instances for each class. However, some datasets are highly imbalanced. That is, some classes are very small relative to others, and in many cases it is these small classes that are actually of interest. For example, in applications like credit card fraud detection there is a tiny fraction of fraudulent transactions which are actually the ones that need to be investigated (Wei et al. (2013)). There are several ap-

proaches to deal with this problem, including (1) under-sampling of the majority classes (Liu et al. (2006)), (2) over-sampling of the minority classes (Chawla et al. (2002)), or (3) using cost-sensitive classifiers which weight the classification error depending on the class (Thai-Nghe et al. (2010)).

2.4.4 Data transformation

Even after cleaning the data with different preprocessing methods it is still possible that learning algorithms cannot find a good relationship between the features and the target value. Thus, data can be transformed for better exposing such relationship. Typical transformation methods include principal component analysis (PCA) that uses an orthogonal transformation to maximise the variance between features; partial least squares (PLS) that finds a set of latent variables that maximises the variance between the input features and the target variable; and the 'kernel trick', that applies a kernel function to transform the input features into a higher dimensional space where the data might be linearly separable (see Figure 2.6). Therefore, instead of just adding new features as in Section 2.4.3, all the features are replaced by the new ones.

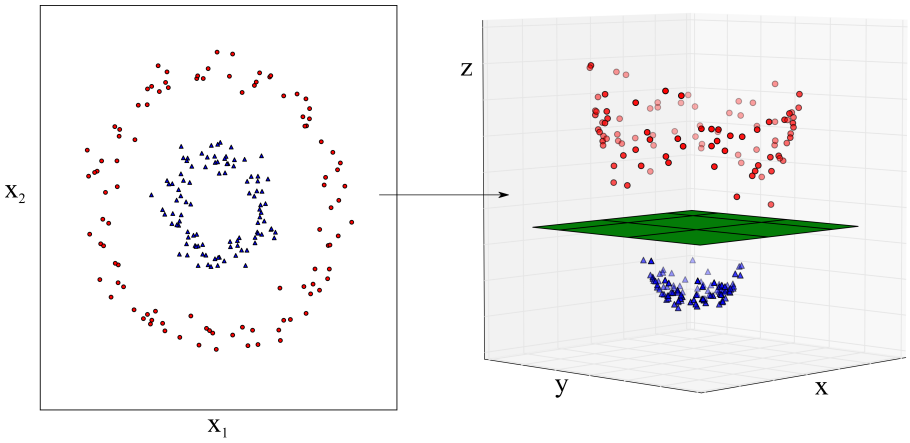


Figure 2.6: *Converting a non-linearly separable dataset (left) into linearly separable one (right) by applying the transformation* $[x_1, x_2] \rightarrow [x_1, x_2, {x_1}^2 + {x_2}^2]$

2.5 Modelling

The aim of the modelling phase is to find the best model for a given problem. To this end, four steps are usually applied in the following order: test design $\rightarrow$ algorithm selection $\rightarrow$ hyperparameter optimisation $\rightarrow$ model assessment, as seen in Figure 2.7. This section describes these steps in more detail.

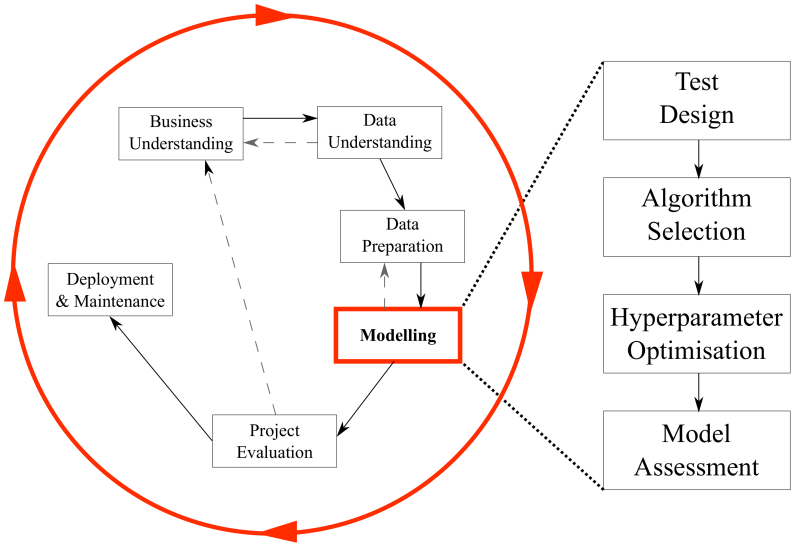


Figure 2.7: *Modelling phases of the CRISP-DM process*

2.5.1 Test design

The generalisation error is an indication of how the model performs on unseen data. There are different measures of predictive performance depending on the task, being classification error, MAE, MSE and RMSE the most frequently used in predictive modelling, which were already introduced in Section 2.2. In addition, the main error estimators are holdout, cross-validation and bootstrapping which were also covered in Section 2.2. The way of splitting the dataset into training and testing sets can vary depending on the data characteristics. Traditionally, instances are randomly sampled. However, in case labels are not evenly distributed, it is common to apply stratified sampling to generate representative sets (Kohavi (1995)). In time series, instances can be selected either in consecutive way or in periodic intervals (Cao & Rhinehart (1995)).

2.5.2 Algorithm selection

There are dozens of learning algorithms and data processing methods in any data mining toolbox (e.g. 420 operators in RapidMiner[3]). Despite the multitude of models exist, according to the 'no free lunch theorem' (Wolpert & Macready (1997)), none of them is universally superior across all possible problems. Nevertheless, there will be specific datasets for which one algorithm performs better than others. The resultant algorithm selection problem (Rice (1976)) hence consists of finding the algorithm

$$A \in \mathcal{A} \mid f_{min} = A(\mathcal{D}_{train}) \tag{2.13}$$

whose model minimises the predictive error for a given dataset (see Equation 2.9).

[3] `https://rapidminer.com`

This process is often a matter of trial and error that can become very tedious if done manually. Thus, there are different smart techniques that help to speed up this process. Serban et al. (2013) presented a review grouping these techniques under the umbrella of intelligent discovery assistants (IDAs). IDAs are systems that help to solve a machine learning problem in an automatic or semi-automatic way.

Consultant-2 (Sleeman et al. (1995)) was the first IDA intended to assist with machine learning problems. It is an interactive system made of about 250 heuristic rules curated by machine learning experts. After a question-answering session with the user, it recommends a learning algorithm by reasoning from the set of rules. This system was limited to small number of algorithms and quickly became obsolete due to its lack of scalability. Another important weakness was the expert bias on the rule generation instead of following a data-driven approach.

The need of overcoming this issue led to the development of meta-learning systems (Lemke et al. (2015)). These systems use a learning algorithm (the so called meta-learner) to generate a model from a repository of previous experiments (e.g. decision tree (Fulkerson et al. (1995)) or k-NN (Giraud-Carrier (2005))). This repository typically contains records made of meta-data such as data characteristics and model performance (e.g. error, speed, complexity). Thus, given a new dataset, the system produces a ranking of learning algorithms according to user preferences. The current state-of-the-art machine learning repository is OpenML[4] with more than 1.7 million experiments and it has been successfully used as a meta-dataset for algorithm selection (e.g. van Rijn et al. (2014)).

A similar approach are case-based reasoning systems. Instead of learning a model, they store all the successful experiments (called cases) in a repository maintained by human experts. Then a case-based reasoner looks for similar cases when a new one is provided by the user. Examples are CITRUS (Wirth et al. (1997)) – which was connected with the KDD suite called Clementine – and the Algorithm Selection Tool (AST – Lindner & Studer (1999)).

All these previous approaches share the same weakness known as the cold start problem. That is, if a new problem is different from the ones seen before, the system will not be able to provide any confident answer. An alternative approach are planning-based data analysis systems which do not need previous experiments or rules but meta-data of the algorithms (i.e. inputs, outputs, preconditions, and effects) instead. This meta-data is generated manually by experts and stored in ontologies. These approaches generate all possible plans for a given dataset by inspecting its characteristics and querying the ontology.

An ontology is a collection of definitions, properties and relations of a domain (Neches et al. (1991)). Over the last decades there have been a number of efforts to create a standard ontology for data mining and KDD process but there is still neither a consensus among researchers nor a definitive solution (Suyama & Yamaguchi (1998); Bernstein & Provost (2001); Euler et al. (2003); Panov et al. (2008); Diamantini et al. (2009); Kietz et al. (2009); Zakova et al. (2011)). One of the main advantages of ontologies is, at the same time, their biggest limitation: they are vendor-independent. That is, due to their

[4] `http://www.openml.org`

aim of being a global standard they tend to be as abstract as possible and therefore it is very difficult to apply them in practice. The most recent work to apply planning-based approaches with ontologies is the e-LICO Intelligent Discovery Assistant (eIDA – Serban et al. (2012)) which uses two ontologies: The Data Mining Workflow Ontology (DMWF – Kietz et al. (2009)) and The Data Mining Optimization Ontology (DMOP – Keet et al. (2015)) to generate a list of plans (i.e. workflows including preprocessing and modelling). The plans are then ranked according to user preferences and using a heuristic that estimates the predictive performance based on previous experiences from similar datasets. The top ranked plans are therefore based on an estimation and hence are not guaranteed to be the best. The generated plans can then be run by the user in the RapidMiner suite (see Figure 2.8).

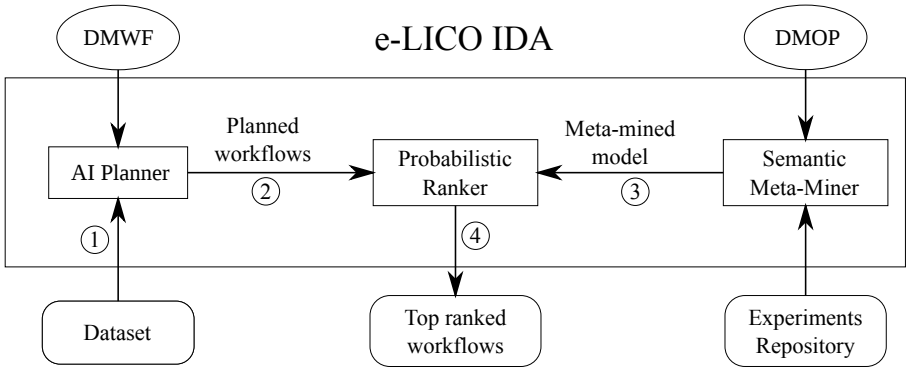


Figure 2.8: *e-LICO Intelligent Discovery Assistant. Firstly, the user provides a dataset and specifies the goal. Then, the AI planner generates a set of valid workflows based on data characteristics and querying the DMWF ontology. The Probabilistic Ranker ranks these workflows based on the meta-mined model which was built by the Semantic Meta-Miner using the DMOP ontology and meta-data from previous experiments. Finally, top ranked workflows are presented to the user. Figure inspired by Figure 7 from Keet et al. (2015).*

Planning-based approaches have three main limitations: (1) they are heuristic approaches that do not guarantee finding the best solution; (2) since ontologies aim to be vendor-independent they are difficult to apply in practice for a range of different problems; and (3) ontologies still require a considerable effort by experts to be maintained and kept up to date. These limitations have motivated approaching the problem as an optimisation task instead of a planning one. The next chapter describes model-based optimisation approaches, which are becoming a de-facto standard in algorithm selection for machine learning due to the successful results and the fact that they are completely data-driven.

2.5.3 Hyperparameter optimisation

Most of the machine learning algorithms have one or more hyperparameters that modify how they operate. For example, for a multilayer perceptron one needs to define the type

of activation function, the number of hidden layers and the learning rate, among other hyperparameters.

A machine learning algorithm $A \in \mathcal{A}$ has associated a set of hyperparameters $\boldsymbol{\lambda} = \{\lambda_1, ..., \lambda_m\} \mid \lambda_i \in \Lambda_i$. The hyperparameter space of $A_{\boldsymbol{\lambda}}$ is defined as $\boldsymbol{\Lambda} = \Lambda_1 \times ... \times \Lambda_m$. The hyperparameter optimisation problem consists of finding the hyperparameters that minimise the loss function of an algorithm for a given dataset

$$\boldsymbol{\lambda}^* = \underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}{\operatorname{argmin}} \mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{test}) \tag{2.14}$$

Hyperparameters are usually continuous or categorical, though they can also be more complex (e.g. a submethod containing additional hyperparameters). In the latter case, the hyperparameters of the submethod are treated as conditional. For instance, Gaussian kernel width parameter in SVM is only present if SVM is using Gaussian kernels.

The value of hyperparameters can drastically affect the method output. Consider a regression problem in which the 'unknown' function is $y = x * sin(x)$ with $x \in [0, 10]$. In order to approximate this function one could use a polynomial regression method, where the hyperparameter is the degree of the polynomial. Figure 2.9 shows the ground truth as well as several curves that have been approximated from a set of training points drawn from the 'unknown' function. By looking at that figure one may assume that the curve will be closer to the true function by increasing the degree of the polynomial. That is however not the case as shown in Figure 2.10. In this toy example, $\Lambda = \{0, 1, ..., 20\}$ and $\lambda^* = 10$ because it is the value that minimises the loss function (MSE on training data).

There are various approaches to selecting optimal values of the hyperparameters. A grid search, that is an exhaustive search over a discretised search space $\boldsymbol{\Lambda}$, represents a simple but often computationally very expensive brute force approach which can be used for hyperparameter optimisation. Other approaches use mathematical optimisation techniques such as dynamic programming (Deisenroth et al. (2009)) or metaheuristics (Lessmann et al. (2005); Bartz-Beielstein et al. (2010)) to explore promising areas of the search space. However, most of these methods do not perform well in problems with a large number of hyperparameters. Instead, a random search has been shown to perform better than grid search in high dimensional problems given a limited amount of time (Bergstra & Bengio (2012)). Bayesian optimisation approaches are another group of optimisation techniques which have recently been shown to be more efficient in finding good solutions in large search spaces (Hutter et al. (2011); Thornton et al. (2013)). These approaches will be described in more detail in Chapter 4.

2.5.4 Model assessment

Different models generated during the modelling phase are ranked according to their predictive performance. Apart from predictive accuracy additional criteria can be taken into account for comparison such as model complexity and/or computational cost. There are some cases in which more than one criterion has to be optimised. Such problems

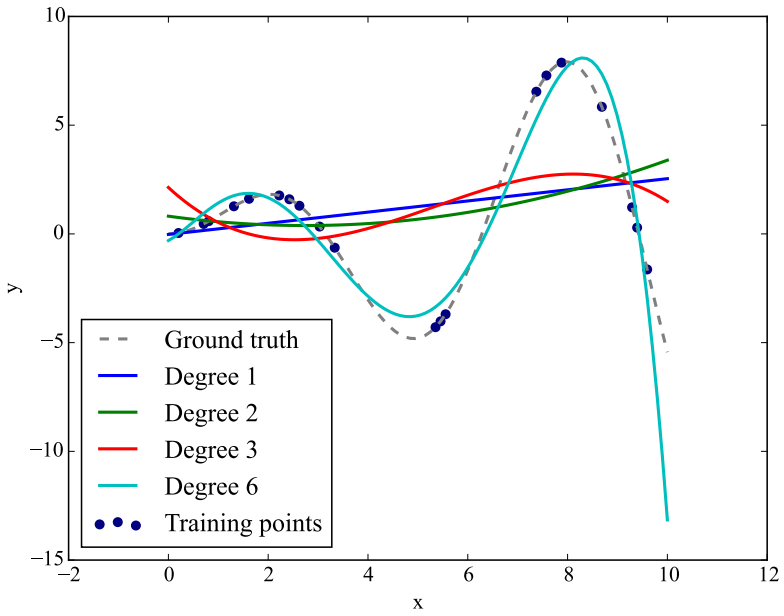


Figure 2.9: *Curve fitting varying the polynomial degree. The ground truth is the function* $y = x \cdot sin(x)$ *with* $x \in [0, 10]$. *Four polynomials with degrees 1, 2, 3 and 6 have been approximated from a set of training points drawn from the 'unknown' function.*

are studied in the multi-objective optimisation field where a Pareto front analysis usually identifies the trade-off between the different optimised objectives and can generate a set of non-dominated models (e.g. Al-Jubouri & Gabrys (2014)).

Performance results of predictive models are typically reported with 95% confidence intervals on the generalisation error. In that way, one can assess if a model is significantly better than other. The final decision on what model is selected for deployment is therefore based on these results.

Nonetheless it may happen that some models perform better than others in different subsets of data. Instead of selecting a single one, it would be possible to combine the predictions of multiple models to achieve a better performance than a single model could obtain (Dietterich (2000)). This is the idea behind ensemble methods that can be built using different training datasets (e.g. Ruta & Gabrys (2010); Budka & Gabrys (2010b)) and/or different learning algorithms (e.g. Littlestone & Warmuth (1994); Tsakonas & Gabrys (2013)). A review of ensemble-based classifiers can be found in Rokach (2010).

2.6 Summary

This chapter has introduced the principles of predictive modelling and has described the steps involved in the data mining process for building predictive systems. Data preparation and modelling are the two key steps to success on classification and regression tasks.

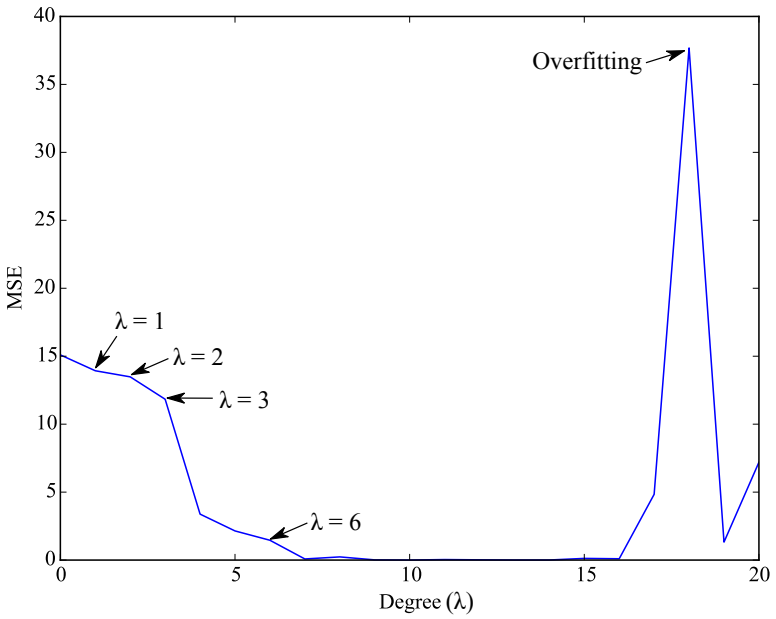


Figure 2.10: *Mean squared error on the test set varying the polynomial degree in range [0,20]. Tagged values belong to the curves represented in Figure 2.9.*

Moreover, applying particular preprocessing methods depending on the model may be beneficial or even required (Žliobaitė & Gabrys (2014)). Therefore, it does make sense to combine data preprocessing and predictive models to form workflows representing multiple data transformation steps. A novel formulation for these types of predictive workflows is introduced in the next chapter. Finding the best workflow that fulfils the project objectives for a particular dataset is a challenging task. This problem is further addressed in Chapter 4, where a fully automatic approach is proposed.

Chapter 3

Multicomponent predictive systems

3.1 Introduction

Previous chapters introduced predictive models and data preprocessing methods as standalone components. In practice however, they are usually combined creating a workflow. Data-driven workflows have been used to guide data processing in a variety of fields. Some examples are astronomy (Berriman et al. (2007)), biology (Shade & Teal (2015)), clinical research (Teichmann et al. (2010)), archive scanning (Messaoud et al. (2011)), telecommunications (Maedche et al. (2000)), banking (Wei et al. (2013)) and process industry (Budka et al. (2014)) to name a few. The common methodology in all these fields consists of following a number of steps to prepare a dataset for data mining. In the field of predictive modelling, the workflow resulting from connecting different methods is known as a Multi-Component Predictive System (MCPS) (Tsakonas & Gabrys (2012)). At the moment, tools like WEKA[1], RapidMiner[2] or Knime[3] allow to create and run MCPSs including a large variety of operators.

Each of these tools has a different representation of workflows. One of the goals of this thesis is to establish a common framework to connect multiple components forming workflows. This will allow to analyse and validate workflows before beginning the implementation phase with any particular tool. The validation of workflows becomes even more relevant when considering the automatic generation of MCPSs.

In order to formalise the notion of MCPSs under a common abstract framework, various approaches can be considered:

a) **Function composition**. Each component is a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that makes an operation over an input tensor[4] $\boldsymbol{x}$ and returns an output tensor $\boldsymbol{y}$. Several components can be connected by composing functions i.e. $f(g(\boldsymbol{x}))$. However, this notation can become a tangle when representing complex workflows involving multiple components with parallel paths of different lengths. Moreover it is not expressive enough to

[1] `http://weka.sourceforge.net`
[2] `https://rapidminer.com`
[3] `https://www.knime.org`
[4] A tensor is a multidimensional array made of continuous or categorical values.

represent different states of a concurrent system.

b) **Directed Acyclic Graphs (DAG)**. A DAG $G = (F, A)$ is a graph made of a set of components (or nodes) F and a set of directed arcs A connecting pairs of components. This approach is very flexible and makes it easy to understand the structure of the MCPS. However, DAGs on their own are not expressive enough to model system execution (e.g. how data is transformed, what meta-data is generated, multiple iterations, and preconditions) or temporal behaviour (e.g. duration and delays).

c) **Petri nets (PN)**. Petri nets are a modelling tool that can be applicable to many types of systems (Petri (1962)). A Petri net $PN = (P, T, F)$ is a directed bipartite graph consisting of a set of places P and transitions T connected by arcs F. Depending on the system, places and transitions can be interpreted in different ways. In this thesis, places are considered to be data buffers and transitions are data processing methods. PNs have been shown to be very useful to model workflows (van der Aalst (1998a)) since they are very flexible, can accommodate complex process logic including concurrency and have a strong mathematical foundation (Murata (1989)). Using workflow algebra (Pankratius & Stucky (2005)) one can modify and create PNs with relational operators like *selection* or *union*. Analysis methods like van der Aalst (2000) inspect PN structure to find potential design errors. An important advantage is that the graphical nature of PNs makes them intuitive and easy to understand for any domain expert. Moreover, PNs are vendor independent and, once composed, can be easily translated to any data mining tool and vice versa. This approach has not been considered before to model MCPSs and it is proposed in this thesis for the first time.

The remainder of this chapter is organised as follows. First, Petri nets and their main properties are described in Section 3.2. Then, the formal definition of MCPS as a type of Petri net is introduced in Section 3.3. After that, Section 3.4 explains the problem of composing MCPS followed by the problem of optimising the hyperparameters of an MCPS in Section 3.5. The combination of both algorithm selection and hyperparameter optimisation of MCPSs into a single problem is formally described in Section 3.6. Finally, the chapter is summarised in Section 3.7.

3.2 Petri nets

While Petri nets were introduced in Petri (1962), the most recent definition of a Petri net, which has been adopted in this thesis, was given in Murata (1989) as the following tuple

$$PN = (P, T, F, W, M_0) \tag{3.1}$$

where $P = \{p_1, ..., p_m\}$ is a finite set of places, $T = \{t_1, ..., t_n\}$ is a finite set of transitions, $F \in (P \times T) \cup (T \times P)$ is a set of arcs, $W : F \to \mathbb{N}^+$ is a weight function, $M_0 : P \to \mathbb{N}$ is the initial marking (i.e. state of the net – number of tokens in each place). Additionally, a Petri net contains one or more tokens that represent units of the system to be processed. The lifetime of a PN is defined by a set of states $\mathcal{M} = \{M_0, ..., M_q\}$. Each state is the distribution of the tokens over P. Figure 3.1 shows an example of the visual representation of a Petri net.

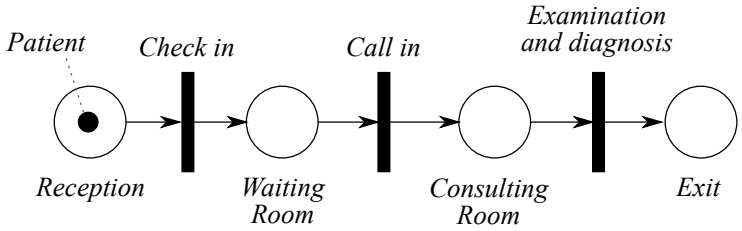


Figure 3.1: *Petri net representing the patient flow in a surgery. Transitions are represented by black vertical bars, places by circles, and arcs by directed arrows. The token is a small black dot which symbolises a patient. In the UK, patients have to check-in in the reception when they arrive to the surgery. Then, they seat in the waiting room until they are called into the consultation room where the doctor proceeds to examine and diagnose the patient. After that, the patient leaves the surgery.*

Nodes are connected by arcs forming paths. Formally,

Definition 1 (Path) *A path C from a node $n_1 \in P \cup T$ to a node $n_k \in P \cup T$ is a sequence of nodes $\langle n_1 n_2 ... n_k \rangle$ such that $f_{i,i+1} \in F \mid 1 \leq i \leq k-1$. A path C is elementary iff, for any two nodes n_i and n_j on C, $i \neq j \Rightarrow n_i \neq n_j$.*

The behaviour of a Petri net is described by firing of transitions. A transition can be fired (i.e. activated) when each of its input places are marked at least with the number of tokens indicated by the value of the function $w(p,t)$ associated with the arc $p \rightarrow t$ (i.e. minimum number of tokens needed in p to fire the transition t). When the active transitions t are fired, the state of the net changes from M_n to M_{n+1} and tokens are transferred from the input to the output places for each transition. Figure 3.2 illustrates such behaviour with an example.

The nodes of a Petri net (i.e. places and transitions) can have multiple output and input arcs. Depending on the behaviour of the node, there are four main constructions:

- **AND-split**, when a token is produced for each of the output arcs;
- **XOR-split**, when a token is produced for only one of the output arcs;
- **AND-join**, when a token is received for each of the input arcs;
- **XOR-join**, when a token is received for only one of the input arcs.

The input nodes of a node $n \in P \cup T$ are denoted as $\bullet n$, while output ones are $n\bullet$.

The flexibility when building Petri nets can lead to anomalous behaviours that can be difficult to trace. Figure 3.3 shows a *deadlock* situation, where the AND-join transition cannot be fired because the XOR-split implies that only one token will be available. Another undesired situation is a *livelock*, shown in Figure 3.4, where the net is always active but blocked by a deadlock. There are a number of analysis techniques that can be used to detect this kind of problems (see e.g. van der Aalst (1997)) and verify certain properties (Murata (1989)). These problematic situations can be completely avoided by applying some restrictions on the construction of Petri nets (van der Aalst et al. (2011)). Next section introduces some properties to ensure the absence of deadlocks and livelocks. These properties are later considered in the formal definition of MCPSs (Definition 6).

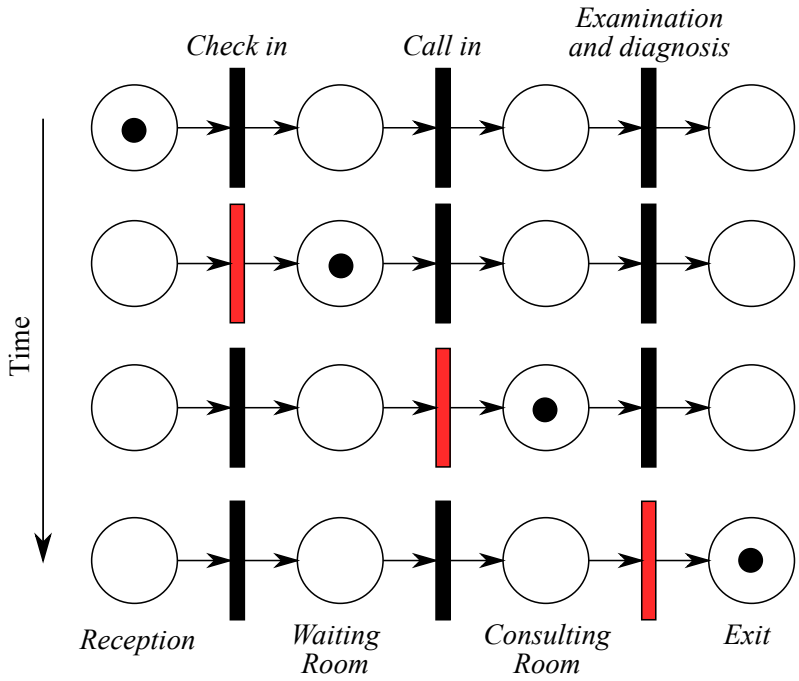


Figure 3.2: *Example of Petri net behaviour over time. The token (i.e. patient) progresses from one place to the next one when the in-between transition is fired (represented in red).*

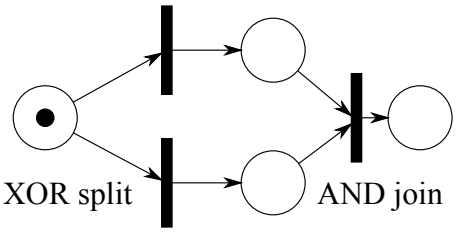


Figure 3.3: *The deadlock situation in this example happens when the AND-join transition cannot be fired because the XOR-split implies that only one token will be available.*

3.2.1 Types of Petri nets

Despite of the fact that the classical Petri nets can cover a large range of applications (e.g. manufacturing (DiCesare et al. (1993)), business process management (van der Aalst et al. (2000)), hardware design (Yakovlev et al. (2000)), molecular biology systems (Hardy & Robillard (2004)), there are sometimes circumstances when new properties have to be defined in order to cover additional types of systems.

For example, van der Aalst (1998a) presented a new type of Petri net to represent business process logic called WorkFlow net (WF-net). WF-nets are sequential workflows with a single starting point and ending point. The simplest WF-net is shown in Figure 3.5 where transitions represent tasks, places could be seen as conditions, and tokens are cases (e.g. a patient, a document or a picture). Formally:

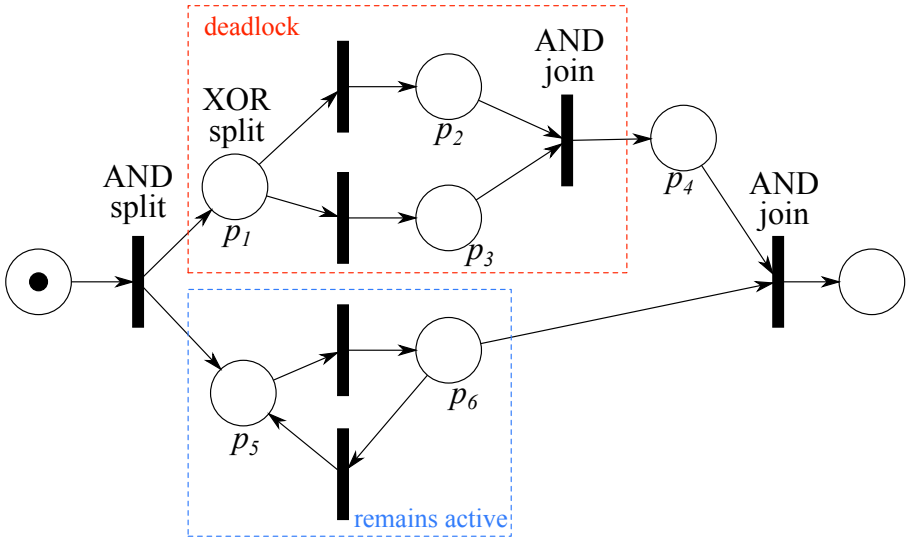


Figure 3.4: *The livelock situation happens when the net is always active but blocked by a deadlock.*

Definition 2 (WF-net) *A Petri net is a WF-net iff:*

a) there is only one source place $i \in P$ *such that* $\bullet i = \emptyset$*;*

b) there is only one sink place $o \in P$ *such that* $o\bullet = \emptyset$*;*

c) every node $n \in P \cup T$ *is on a path from* i *to* o*.*

The third point of this definition entails that if a new transition t connecting o with i is added, then the resulting Petri net is strongly connected.

Definition 3 (Strongly connected) *A Petri net is strongly connected iff for every pair of nodes (i.e. places and transitions)* x *and* y*, there is a path leading from* x *to* y*.*

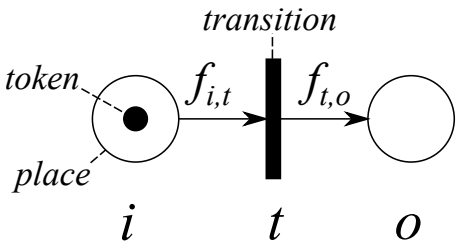


Figure 3.5: *The simplest WorkFlow net with a single transition. Transitions represent tasks, places could be seen as conditions, and tokens are cases (e.g. a patient, a document or an image).*

The **soundness** property for WF-nets introduced by van der Aalst (1997) implies that if a net has k tokens in the input place during the initial marking, it will have k tokens in the output place at the final marking (i.e. $M_0(i) = M_q(o)$). A WF-net with soundness is guaranteed to terminate (i.e. it does not have deadlocks or livelocks).

In order to avoid deadlocks in WF-nets, van der Aalst (2000) introduced the well-

handled property that ensures the lack of bad constructions (e.g. XOR-split followed by AND-join block). Formally:

Definition 4 (Well-handledness) *A Petri net is well-handled iff, for any pair of nodes n_i and n_j such that one is a place and the other a transition, and for any pair of elementary paths C_1 and C_2 leading from n_i to n_j, $C_1 \cap C_2 = \{n_i, n_j\} \Rightarrow C_1 = C_2$.*

Petri nets can become very large when defining complex processes (van der Aalst (1998b)). To facilitate the representation, **hierarchical** Petri nets were introduced as an extension of PNs where a transition can be represented by another PN (called subnet) – see Figure 3.6. The action of replacing a transition by a subnet is called an iteration. Iterations are denoted as regular when the subnet has entrance and exit nodes acting as dummy nodes. This concept leads to a new type of Petri nets known as WRI-WF nets (Well-handled with Regular Iterations WF-net) presented in Ping et al. (2004). Formally:

Definition 5 (WRI-WF net) *A Petri net is a WRI-WF net iff:*

a) the PN is a WF-net (see Definition 2);

b) the PN is well-handled (see Definition 4);

c) the PN is acyclic;

d) the iterations of the PN are regular.

WRI-WF nets are inherently sound (see Ping et al. (2004) for proof).

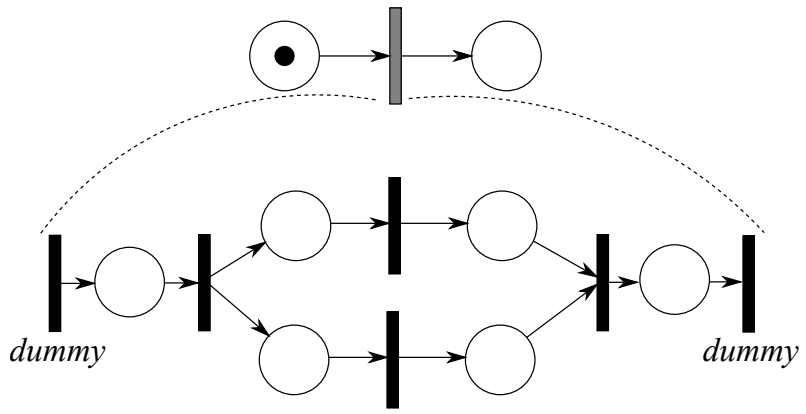


Figure 3.6: *Hierarchical WF-net with parallel paths. Transitions containing subnets are in grey. At the most abstract level, this WF-net has only a single transition, but this is made of a subnet with parallel paths. The subnet has entrance and exit nodes acting as dummy nodes.*

Another common extension of PNs are **Coloured** Petri nets (CPN), where each token has a colour/type associated with it. Places and transitions can therefore handle each token type differently. To see how CPNs are useful in relation to MCPS, please refer to Appendix C where data and meta-data are represented by different type of tokens in the same net.

PNs do not originally support modelling time. For that, **Timed** Petri net (TPN) is an

extension of PNs where time can be associated with tokens, places and/or transitions. This is useful for example to model task duration or delays. Although TPNs are not considered in this thesis, they could be a matter of further study in future work.

3.3 Modelling MCPS as Petri nets

WRI-WF-nets are the base for defining MCPSs. However, to comply with the predictive nature of MCPSs there are some additional restrictions that have to be added. Formally:

Definition 6 (MCPS) *A Petri net is an MCPS iff all the following conditions apply:*

a) the PN is WRI-WF-net (see Definition 5);

b) each place $p \in P \setminus \{i, o\}$ has only a single input and a single output;

c) the PN is 1-bounded, that is, there is a maximum of one token in each $p \in P$ for every reachable state (i.e. $M(p) \leq 1$);

d) the PN is 1-sound (i.e. $M_0(i) = M_q(o) = 1$);

e) the PN is ordinary (i.e. $w = 1 \; \forall w \in W$);

f) all the transitions $t \in T$ with multiple inputs or outputs are AND-join or AND-split, respectively;

g) any token is a tensor (i.e. multidimensional array);

h) the token at i is a set of unlabelled instances and the token at o is a set of predictions for such instances.

In an MCPS, an atomic transition $t \in T$ is an algorithm with a set of hyperparameters λ that affect how the token is processed. An MCPS can be as simple as the one shown in Figure 3.5 with a single transition. For example, the token in i can be a set of unlabelled instances, and t a classifier which consumes such token from the arc $f_{i,t}$ and generates one token in o with the predicted labels through $f_{t,o}$.

An MCPS can however be hierarchically extended since each transition t can be either atomic or another WRI-WF-net (with additional starting and ending dummy transitions) – see Figure 3.6 where atomic transitions are black and special transitions are grey. As a consequence, an MCPS can model very complex systems with multiple data transformations and parallel paths (see e.g. Figure 4.4 for a multi-hierarchy example).

In predictive modelling, the semantics for transitions are: (1) preprocessing methods, (2) predictors, (3) ensembles and (4) postprocessing methods. Transitions representing (1), (2), and (4) can be either atomic or special. However, type (3) transitions are necessarily special since ensembles are made of several predictors and a combination method (e.g. voting).

Depending on the number of inputs and outputs, MCPSs can have any of the following types of transitions (see Figure 3.7):

- $1 \rightarrow 1$ transitions (e.g. a classifier that consumes unlabelled instances and returns predictions)
- $1 \rightarrow n$ transitions (e.g. a random subsampling method that consumes a set of

instances and returns several subsets)

- $n \rightarrow 1$ transitions (e.g. a voting classifier that consumes multiple predictions per instance and returns a single prediction per instance)

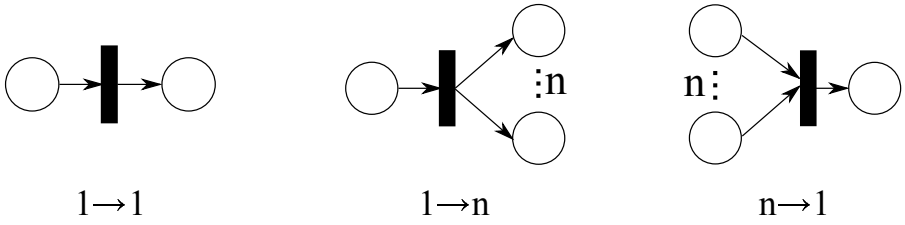


Figure 3.7: *Types of transitions according to the number of inputs and outputs*

Building an MCPS is typically an iterative, labour and knowledge intensive process. Despite a substantial body of research in the area of automated and assisted MCPS creation and optimisation (see Section 2.5.2), a reliable fully automated approach is still far from being available. Figure 3.8 shows the five stages of an MCPS life cycle. Let's consider the classification problem of 'wine' dataset[5] to illustrate these stages. This is a well known dataset in the machine learning community consisting of 178 records with 13 numerical attributes resulting from chemical analysis of 3 types of wines produced in the same region of Italy. The classification task of this dataset is not particularly challenging, but it is suitable for illustration purposes.

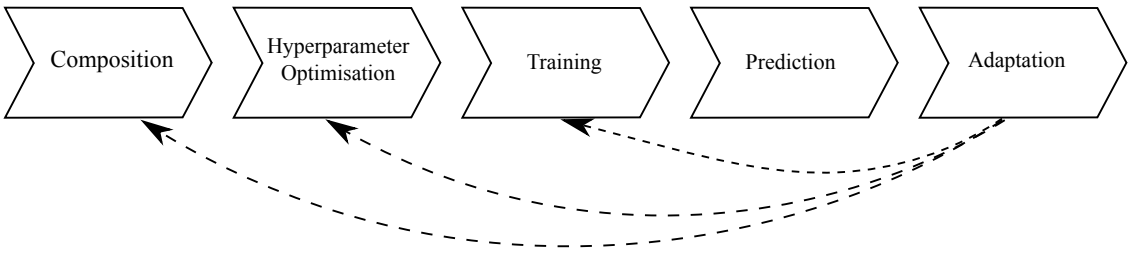


Figure 3.8: *Stages of an MCPS*

The first stage is MCPS composition. Typically, a practitioner would design one or more MCPSs by selecting different methods based on some criteria (e.g. interpretability) and his/her own cognitive bias. For the sake of the argument, let's say that the MCPS composed for 'wine' is the one shown in Figure 3.9 which uses Fisher's linear discriminant analysis (LDA – Fisher (1936)) to reduce the number the attributes and classify the instances. The MCPS composition problem is later formalised in Section 3.4.

[5] `https://archive.ics.uci.edu/ml/datasets/Wine`

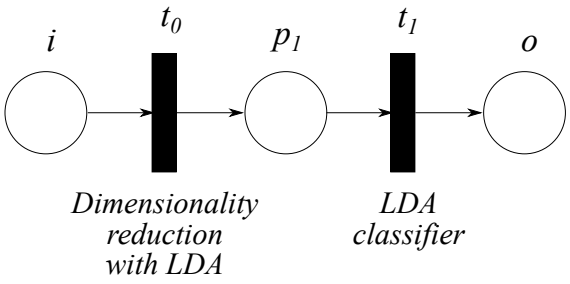


Figure 3.9: *Example of MCPS for 'wine' dataset. The first transition t_0 will reduce the dimensionality of the dataset from place i using a linear projection that maximises class separability. Then, the processed instances in p_1 are classified by LDA classifier in transition t_1.*

Once an MCPS has been composed, hyperparameters of the selected methods need to be optimised. This can be done for example using grid search or any other optimisation method. The hyperparameter optimisation problem is further discussed in Section 3.5, but let's assume that the optimisation strategy has found that the best number of LDA components is 2. This is not surprising since the the maximum number of components in LDA is the number of classes-1, but this example has been chosen for illustrative purposes.

Then, the MCPS components have to be trained with a set of labelled instances. During the training process, each transition $t \in T$ is initialised using the tokens available in $p \in \bullet t$. One of the advantages of Petri nets is the ability to refer to any part of the system at any particular state. The MCPS shown in Figure 3.9 has 3 states during the training stage $\mathcal{M} = \{M_0, M_1, M_2\}$. In the initial state M_0, one token representing raw data is at place i (see Figure 3.10). Then the transition t_0 is triggered consuming the token from i and producing one new token at p_1. Thus, the PN progresses to the state M_1 (see Figure 3.11). At this state, the token at p_1 is the transformed data after LDA has been applied. After that, t_1 is triggered consuming the token at p_1 and producing one new token at o which is a vector of predictions. That is the last state of the training stage (M_2) which will finalise with the trained classifier shown in Figure 3.12.

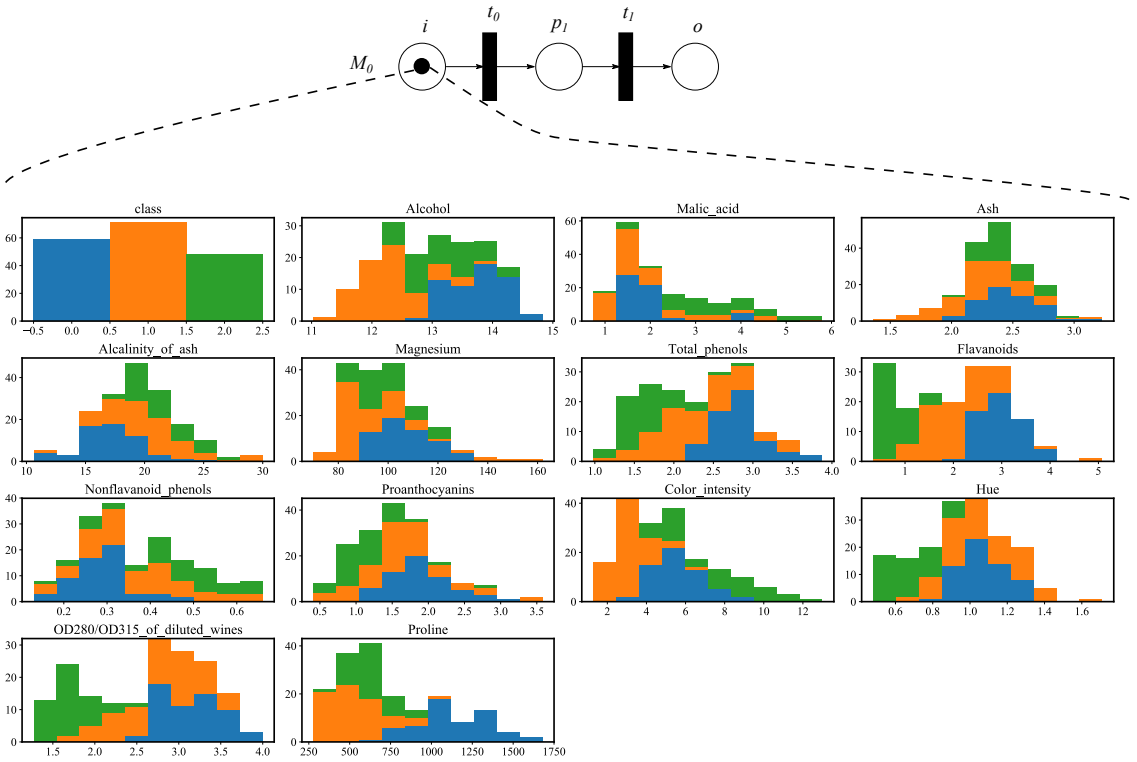


Figure 3.10: *Initial state of the MCPS for 'wine' dataset during the training stage. The token at this state represents the complete dataset. The figure shows the histograms of each attribute values per class.*

After that, the MCPS is ready to make predictions for unlabelled data. Predictions are delivered during the online phase by $\hat{\boldsymbol{y}} = \theta(X)$, where θ is the trained MCPS, X is the raw data made of unlabelled instances (i.e. token in i) and $\hat{\boldsymbol{y}}$ is the token in o containing the predictions of $\boldsymbol{y}$. The number of states of the prediction phase is unbounded. At any new state, one new token made of new raw data might appear in i which will initiate the predictive process.

Finally, the MCPS can be adapted to avoid a potential degradation of predictive performance due to changes in data. The adaptation phase can involve a recomposition (e.g. adding a new component or replacing an existing one), adjusting a few parameters or just some additional training. Appendix C presents coloured MCPSs which help to model local adaptation of components by propagating meta-data through the system components. Furthermore, Chapter 5 explains several adaptation strategies including global re-composition and global parametrisation of MCPSs.

3.4 Composition of MCPS

The next step, after data understanding, for building a predictive system in CRISP-DM is to select the right methods for cleaning and transforming the raw data. For this pur-

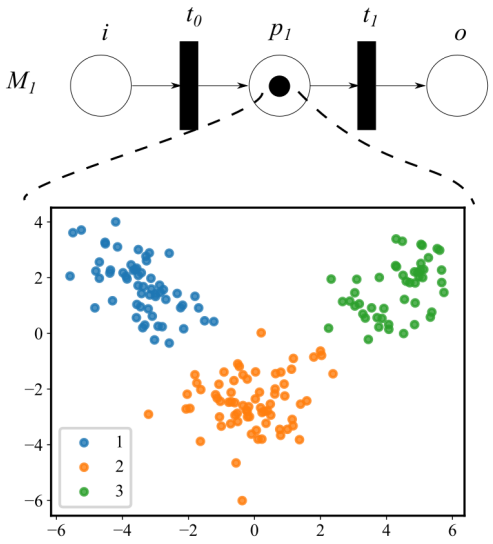


Figure 3.11: *State M_1 of the MCPS for 'wine' dataset during the training stage. The token at this state represents the reduced dataset after LDA has been applied. The plot shows the transformed dataset, where each colour is a different class.*

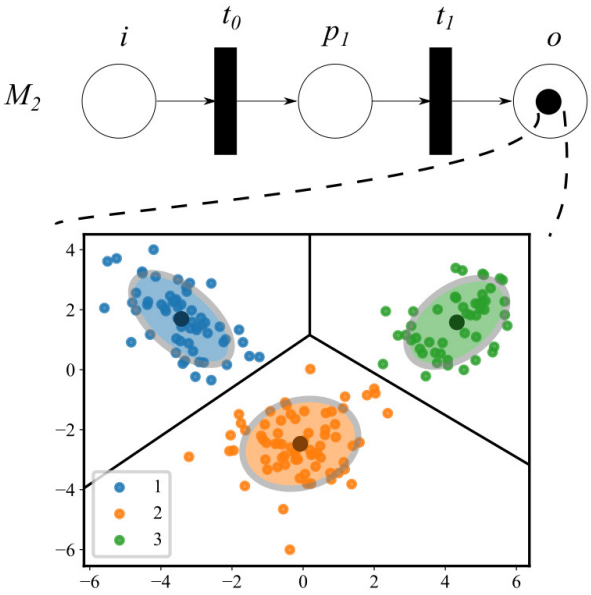


Figure 3.12: *State M_2 of the MCPS for 'wine' dataset during the training stage. The token at this state represents the classified instances. This is the last state for this particular MCPS. Thus, both t_0 and t_1 are trained and the MCPS is ready to make predictions for unlabelled data.*

pose, data mining toolboxes contain a number of methods to choose from. Either using a graphical interface or a text based approach, one can pick the desired methods and connect them forming a workflow (see e.g. Figure 3.13).

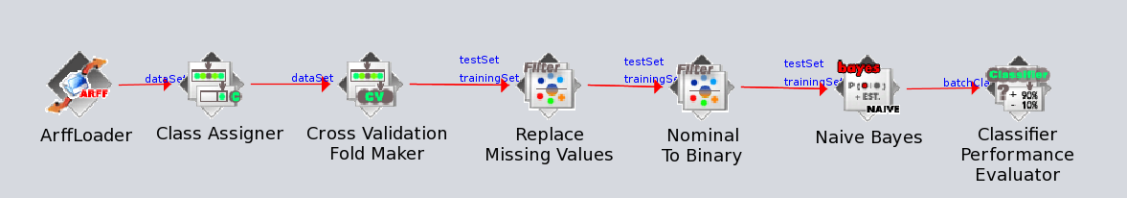


Figure 3.13: *Example of WEKA workflow. Components have been manually selected. (Screenshot from WEKA Knowledge Flow).*

The algorithm selection problem defined by Rice (1976) consists of finding the best algorithm for solving a certain problem. In the context of predictive modelling, the 'best algorithm' is the MCPS that minimises the predictive error for a given dataset.

Let Θ be the set of all possible MCPSs that can be composed using the available methods. For example, if there is only a single method available, the only MCPS that can be composed has an input place, a transition, and an output place as in Figure 3.5. However, if there are no constraints on how the components can be connected, the size of Θ is infinite (i.e. $|\Theta| = \infty$). That could be done for example by infinitely concatenating the same component or having an unbounded hyperparameter range (e.g. $\lambda \in \mathbb{R}$). For that reason, it is common to apply constraints to limit $|\Theta|$ like limiting the maximum number of nodes or restricting the list of methods by using meta-learning (Feurer et al. (2014)), prior knowledge (Swersky et al. (2013)) or surrogates (i.e. cheap-to-evaluate models, Eggensperger et al. (2012)). MCPS composition framework is represented in Figure 3.14. The set Θ defines the search space which is generated from a pool of methods taking into account a number of constraints. Each MCPS from the search space is evaluated on the given dataset. Finally the best MCPS according to the evaluation criteria is returned.

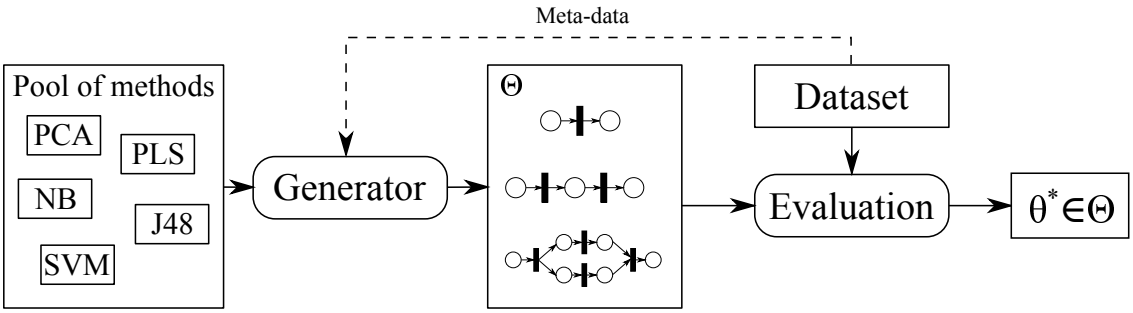


Figure 3.14: *Diagram of MCPS composition framework. The search space Θ is made of MCPSs generated from the pool of available methods which is filtered by the meta-data of the given dataset. Then, MCPSs are evaluated and the best one according to the loss function is returned.*

Formally, the composition problem consists of finding the MCPS

$$(P, T, F)^* = \theta^* = \underset{\theta^{(j)} \in \Theta}{\arg\min}\, \mathcal{L}(\theta^{(j)}, \mathcal{D}_{train}, \mathcal{D}_{test}) \tag{3.2}$$

where $\mathcal{L}$ is the loss function which in this thesis is the predictive error for the testing set $\mathcal{D}_{test}$ achieved by the $\theta^{(j)}$ after being trained with the training set $\mathcal{D}_{train}$.

Nevertheless, most of the methods have hyperparameters that play an important role. Next section introduces the problem of hyperparameter optimisation for MCPSs.

3.5 Hyperparameter optimisation of MCPS

The hyperparameter optimisation problem for a learning algorithm was described in Section 2.5.3. In the context of MCPSs, each transition $t \in T$ has a set of hyperparameters λ, hence Equation 3.2 needs to be modified as follows

$$(P, T_{\boldsymbol{\lambda}^*}, F)^* = \theta^*_{\boldsymbol{\lambda}^*} = \underset{\theta^{(j)} \in \Theta, \boldsymbol{\lambda} \in \Lambda^{(j)}}{\arg\min}\, \mathcal{L}(\theta^{(j)}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{test}) \tag{3.3}$$

where $\Lambda^{(j)}$ is the set of all possible combinations of hyperparameter values for $\theta^{(j)}$. However, finding the best values for a set of hyperparameters could be a challenging task for several reasons:

- if the number of hyperparameters to optimise is large;
- if the domain of the hyperparameters is large and therefore $|\Lambda^{(j)}| \to \infty$;
- if the loss function is not monotonic and has many local minima.

Main strategies for approaching hyperparameter optimisation problem include:

- **Coordinate descent**, where a greedy approach is used in which each hyperparameter is optimised while the rest stay fixed (Friedman et al. (2007)).
- **Grid search**, where the search space is fully explored (Bergstra & Bengio (2012)).
- **Random search**, where the search space is partially explored in a random way during a certain amount of time (Bergstra & Bengio (2012)).
- **Model-based methods**, where the unknown loss function is sought by exploring the search space in a sequential manner (Hutter et al. (2011)).

The state-of-the-art of these methods is discussed further in the next chapter.

Hyperparameter optimisation is usually performed after the model has been selected (see e.g. Bengio (2000); Guo et al. (2008); Bergstra & Bengio (2012)). However, the problem is very similar to model selection since different parametrisations of the same model (e.g. different kernels in a support vector machine (SVM), different structures of a neural network) can in fact be treated as different models. Therefore it makes sense to approach both model and hyperparameter selection problems jointly. The Combined Algorithm Selection and Hyperparameter optimisation (CASH) problem presented in the next section defines a search space in which both these tasks are merged.

3.6 CASH problem for MCPS

The Combined Algorithm Selection and Hyperparameter optimisation (CASH) problem – originally defined by Thornton et al. (2013) – consists of finding the best combination of learning algorithm A^* and hyperparameters $\boldsymbol{\lambda}^*$ that optimise an objective function (e.g. Equation 3.4 minimises the k-fold cross-validation error) for a given dataset $\mathcal{D}$. Formally, CASH problem is given by

$$A^*_{\boldsymbol{\lambda}^*} = \underset{A^{(j)} \in \mathcal{A}, \boldsymbol{\lambda} \in \Lambda^{(j)}}{\arg\min} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A^{(j)}_{\boldsymbol{\lambda}}, \mathcal{D}^{(i)}_{train}, \mathcal{D}^{(i)}_{valid}) \tag{3.4}$$

where $\mathcal{A} = \{A^{(1)}, \ldots, A^{(k)}\}$ is a set of algorithms with associated hyperparameter spaces $\Lambda^{(1)}, \ldots, \Lambda^{(k)}$. The loss function $\mathcal{L}$ takes as arguments an algorithm configuration $A_{\boldsymbol{\lambda}}$ (i.e. an instance of a learning algorithm and hyperparameters), a training set $\mathcal{D}_{train}$ and a validation set $\mathcal{D}_{valid}$.

To accommodate the definition of MCPS into a CASH problem, $\mathcal{A}$ is generalised from Equation 3.4 to be a set of MCPSs Θ rather than individual algorithms. As a consequence, each A is now an MCPS $\theta^{(j)} = (P, T_{\boldsymbol{\lambda}}, F)^{(j)}$ which has a hyperparameter space $\Lambda^{(j)}$, made of the concatenation of the hyperparameter spaces of all its transitions T. The CASH problem is now concerned with finding

$$(P, T_{\boldsymbol{\lambda}^*}, F)^* = \underset{(P,T,F)^{(j)} \in \Theta, \boldsymbol{\lambda} \in \Lambda^{(j)}}{\arg\min} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}((P, T_{\boldsymbol{\lambda}}, F)^{(j)}, \mathcal{D}^{(i)}_{train}, \mathcal{D}^{(i)}_{valid}) \tag{3.5}$$

Complexity of MCPSs can vary significantly depending on the number of components and their hyperparameters. Increasing the number of components in an MCPS also increases the number of hyperparameters that need to be set and therefore makes it a much more difficult optimisation problem. The next chapter focuses on how to tackle the CASH problem and reports on an extensive experimental analysis performed as part of this study.

3.7 Summary

Data-driven predictive modelling requires a number of steps to prepare a dataset for building a model. The workflow connecting different data processing methods is known as Multi-Component Predictive System (MCPS). There is however a gap in the literature to formalise the representation of an MCPS. To this end, this thesis proposes a novel approach based on Petri nets. The advantages of Petri nets over other representations lie on their expressive power to model process execution and their analysis techniques for verifying workflow logic (van der Aalst (2000)). The proposed approach also allows the generalisation of different data preprocessing methods and learning algorithms under the same concept of 'transition'. Moreover, this new definition has made possible the gen-

eralisation of the CASH problem to not only optimise a single algorithm but any-length workflow, which is also a novel contribution.

A further extension of MCPS to support multiple type of tokens is presented in Appendix C, where the concept of coloured tokens is used to represent data and meta-data within the same system. This serves as an example of how the flexibility of Petri nets helps to represent a process flow in complex systems.

The case study presented in Appendix B shows the process of manually composing an MCPS in a real scenario. In such case, the optimisation of a single preprocessing component (i.e. shutdown detection) has been quite time-consuming, including not only tuning of hyperparameters but also help of domain-knowledge experts to select the right features. For that particular problem, MCPS composition was completed in the first place and then hyperparameter optimisation was performed on the selected components. That means that some methods were discarded either based on expert bias or testing only on default hyperparameters. Therefore, some potentially good methods which performance highly depends on hyperparameter values could have been discarded upfront and not further considered. Therefore, it makes sense to join both problems into a single one forming a CASH problem.

It would be then highly desirable to automate the process of composing and optimising an MCPS for a given dataset. This would not only save time but also it could potentially find better solutions than a human could, given the same amount of time. However, finding the best MCPS in large search spaces is a challenging task that requires a high computational cost which is aimed to be reduced by smart strategies. The following chapter presents how to approach this problem with different strategies.

Chapter 4

Automatic composition and optimisation of MCPSs

4.1 Introduction

Performance of data-driven predictive models heavily relies on the quality and quantity of data used to build them. However, in real applications, even if the data is abundant, it is also often imperfect and considerable effort needs to be invested into a labour-intensive task of cleaning and preprocessing such data in preparation for subsequent modelling. Some authors claim that these tasks can account for as much as 60-80% of the total time spent on developing a predictive model (see e.g. Pyle (1999); Linoff & Berry (2011)). Therefore, approaches and practical techniques allowing to reduce this effort by at least partially automating some of the data preparation steps, can potentially transform the way in which predictive models are typically built.

In many scenarios one needs to sequentially apply multiple preprocessing methods to the same data (e.g. outlier detection $\rightarrow$ missing value imputation $\rightarrow$ dimensionality reduction), effectively forming a preprocessing chain. Composition of such a preprocessing chain is a challenging problem as described in the previous chapter. This task, apart from choosing the components to use and arranging them in a particular order, also includes setting their hyperparameters.

After the data has been preprocessed in an appropriate way, the next step in a data mining process is modelling. Similarly to preprocessing, this step can also be very labour-intensive, requiring evaluation of multiple alternative models. Hence automatic model selection has been attempted in different ways, for example using active testing (Leite et al. (2012)), meta-learning (Lemke & Gabrys (2010)) or information theory (McQuarrie & Tsai (1998)). A common theme in the literature is comparison of different models using data always preprocessed in the same way. However, some models may perform better if they are built using data specifically preprocessed with a particular model type in mind. In addition, hyperparameters play an important role in most of the models and setting them manually is time-consuming mainly for two reasons: (1) there are typically multiple hyperparameters which can take many values (with an extreme case being con-

tinuous hyperparameters), and (2) they are validated using cross-validation.

The motivation for automating composition of MCPSs is twofold. In the first instance it will help to reduce the amount of time spent on the most labour-intensive activities related to predictive modelling, and therefore allow to dedicate human expertise to other tasks. The second motivation is to achieve better results than a human expert could, given a limited amount of time. The number of possible methods and hyperparameter combinations increases exponentially with the number of components in an MCPS and, in majority of the cases, it is not computationally feasible to evaluate all of them.

This chapter tackles the problem of automating the CASH problem of MCPSs as an optimisation task to minimise the predictive error for a given dataset. Firstly, Section 4.2 explains how to solve the CASH problem using Bayesian optimisation, which is the base for the most promising approaches in recent years. These approaches are known as Sequential Model-Based Optimisation (SMBO) methods and are presented in Section 4.2.2. After that, Section 4.2.3 explains the software development carried out during this research to build MCPS using SMBO methods. An extensive experimental analysis with multiple datasets to compare different optimisation strategies, results and related discussion are presented in Section 4.3. The chapter concludes in Section 4.4.

4.2 Automating the CASH problem

Automatic construction of Petri nets has been previously addressed in the literature. For instance, Anastasiou (2013) infers a Petri net from traces of indoor location data of individuals using cluster analysis. Another approach is process mining (van der Aalst (2012)), in which a WF-net is composed by extracting knowledge from the event logs of a working process. However, there is a research gap on automating the composition of Petri nets for predictive modelling. In particular, this thesis considers the problem of composition and optimisation of MCPSs for classification and regression tasks.

This problem is formulated in this thesis as a CASH problem (see Equation 3.5). For the sake of clarity, let $\theta = (P, T_\lambda, F) \in \Theta$ be an MCPS configuration within the search space Θ formed of all possible MCPS configurations; and $c_\theta = \mathcal{L}(\theta, \mathcal{D}_{train}, \mathcal{D}_{test})$ the cost of applying MCPS over the test set $\mathcal{D}_{test}$. In this thesis, the cost refers to the predictive error. Nevertheless, it could represent a different value (e.g. evaluation time) or a combination of values (e.g. both error and time).

One way of finding the best MCPS θ^* for a given dataset $\mathcal{D}$ is to perform a **grid search**. This consists of an exhaustive search over Θ. However, such technique could be computationally very expensive in large search spaces or with big datasets. Instead, a simpler mechanism like **random search**, where the space is randomly explored within a given time budget, has been shown to be more effective in high-dimensional settings (Bergstra & Bengio (2012)).

The main reason for MCPS composition being a challenging problem is the computational power and the amount of time needed to explore high dimensional search spaces. To begin with, an undetermined number of nodes can make the workflow very simple

or very complex. Secondly, the order in which the nodes should be connected is unknown a priori. Also, even transitions belonging to the same category (e.g. missing value imputation) can vary widely in terms of the number and type of hyperparameters (e.g. continuous, categorical or conditional), with defining of a viable range for each of the hyperparameters being an additional problem in itself. This complexity makes techniques like grid search not feasible. Even 'intelligent' strategies can struggle with exploration because the high dimensional search space is likely to be plagued with a multitude of local minima.

As previously mentioned in Section 3.4, the size of the search space (i.e. $|\Theta|$) can be reduced by applying a range of constraints (Feurer et al. (2014); Swersky et al. (2013); Eggensperger et al. (2012)). However, this thesis investigates the impact of extending the search space, not by including more predictive models, but considering preprocessing methods instead. Nonetheless, some constrains are applied like limiting the amount and order of components which will be explained in Section 4.3.

4.2.1 Bayesian optimisation

A promising approach to the CASH problem that has gained popularity in the last few years is Bayesian optimisation (Brochu et al. (2010); Shahriari et al. (2016)). This approach – outlined in Algorithm 1 – aims to find

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \, \mathcal{L}(\theta, \mathcal{D}_{train}, \mathcal{D}_{test}) \tag{4.1}$$

that globally minimises the loss function $\mathcal{L}$. It assumes that the posterior distribution $p(\mathcal{L} \mid R_{1:n})$ can be estimated by the likelihood function $p(R_{1:n} \mid \mathcal{L})$ and the prior distribution $p(\mathcal{L})$ using Bayes' theorem

$$p(\mathcal{L} \mid R_{1:n}) \propto p(R_{1:n} \mid \mathcal{L}) p(\mathcal{L}) \tag{4.2}$$

where $R_{1:n} = \{(\theta_1, c_{\theta_1}), ..., (\theta_n, c_{\theta_n})\}$ is the set of run configurations and its associated costs. Since evaluating the loss function is costly, an acquisition function $\alpha_{p(\mathcal{L})} : \Theta \to \mathbb{R}$ quantifying the utility of an evaluation is used instead as a cheaper alternative. This function has an inherent trade-off between exploration (where there is more uncertainty) and exploitation (where the cost is expected to be low). There are different types of acquisition functions based on the likelihood of improvement (Kushner (1964)), the upper confidence bound criterion (Lai & Robbins (1985)), or information gain (Hennig & Schuler (2012)).

The procedure of Algorithm 1 is also visualised with a toy example in Figure 4.1. In this one-dimensional problem, the unknown loss function is represented by a dashed line. The prior probability $p(\mathcal{L})$ is captured by a Gaussian process (GP). At time $t = 2$, two runs have already been completed and the acquisition function is recommending to evaluate the next point that is likely to minimise the cost. When the new candidate point is evaluated, the GP is updated and the procedure continues.

Algorithm 1 Bayesian optimisation

1: **for** $n = 1, 2, \ldots$ **do**
2: $\theta_{n+1} = \underset{\theta \in \Theta}{\operatorname{argmax}}\, \alpha(\theta)$ ▷ select most promising configuration
3: $c_{\theta_{n+1}} = \mathcal{L}(\theta_{n+1}, \mathcal{D}_{train}, \mathcal{D}_{test})$ ▷ compute cost
4: $R_{n+1} = \{R_n, (\theta_{n+1}, c_{\theta_{n+1}})\}$ ▷ update list of run configurations
5: update $p(\mathcal{L} \mid R_{1:n+1})$
6: **end for**

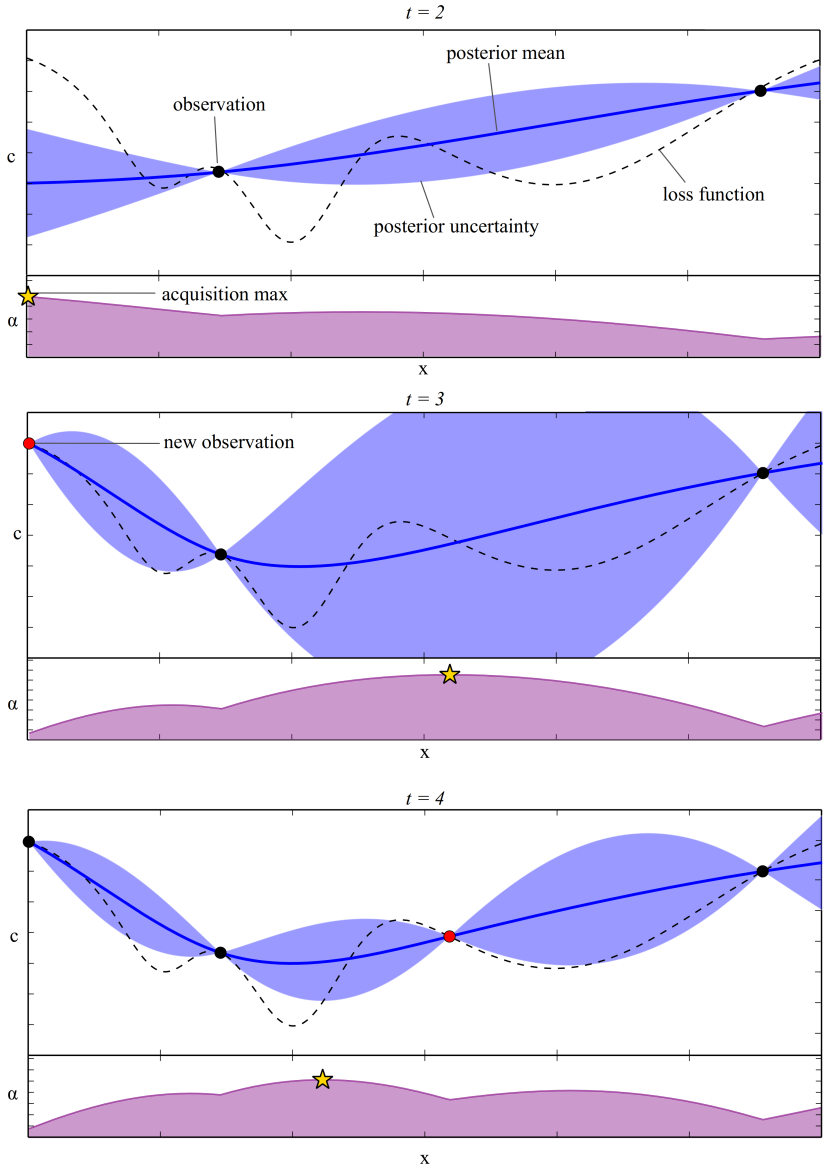


Figure 4.1: *Example of Bayesian optimisation on a one-dimensional problem. At $t = 2$, the GP has been initialised with two observations of the costly-to-evaluate loss function. The cheap-to-evaluate acquisition function finds what is most promising value of x to evaluate next based on the posterior mean and uncertainty. At $t = 3$, the GP has been updated with a new observation. The procedure continues until time budget is exhausted.*

4.2.2 Sequential Model-Based Optimisation

Sequential Model-Based Optimisation (SMBO), introduced by Hutter et al. (2011), is a Bayesian optimisation framework that incrementally builds a regression model ψ – known as surrogate model – using instances from R. Then, such model is used to predict the performance of promising candidate configurations. The selection of promising configurations is guided by an acquisition function $\alpha_\psi : \Theta \to \mathbb{R}$. A function that has been shown to work well in SMBO framework (Hutter et al. (2009)) is the expected improvement (EI – Mockus et al. (1978)) given by

$$\alpha_\psi(\theta \mid R_n) = \mathbb{E}_\psi[I(\theta)] = \mathbb{E}_\psi[\max\{0, c_{min} - c_\theta\}] \tag{4.3}$$

where c_{min} is the cost of the best configuration found so far. The advantage of this function is that it can be evaluated without computing the loss function for each θ (i.e. running the configuration) since c_θ can be estimated using ψ. A common technique to select the next promising configuration consists of evaluating EI for thousands of random samples and then returning the best one (Hutter et al. (2011)). Algorithm 2 shows the SMBO procedure that returns the best configuration found θ_{min} (also known as 'incumbent').

Algorithm 2 Sequential Model-Based Optimisation

1: θ_{min} = initial configuration (usually random sample from Θ)
2: $R = \{[\theta_{min}, \mathcal{L}(\theta_{min}, \mathcal{D})]\}$ ▷ initialise set of run configurations and associated costs
3: **repeat**
4: $\quad \psi = FitModel(R)$
5: $\quad \theta = FindNextConfiguration(\alpha, \psi, \theta_{min}, \Theta)$ ▷ See Figure 4.1
6: $\quad c_\theta = \mathcal{L}(\theta, \mathcal{D})$ ▷ compute cost
7: $\quad R.add([\theta, c_\theta])$ ▷ update list of run configurations
8: $\quad \theta_{min} = \underset{\theta}{\operatorname{argmin}}\, c_\theta \mid [\theta, c_\theta] \in R$ ▷ update best configuration found
9: **until** budget exhausted
10: **return** θ_{min}

Apart from continuous and categorical, it is often the case of having conditional attributes. That is, some attributes may influence the optimisation problem only when some other attributes take certain values. For example Gaussian kernel width parameter in SVM is only present if SVM is using Gaussian kernels. Search spaces containing this type of attributes are known as conditional spaces (Shahriari et al. (2016)).

The ability of SMBO methods to work in conditional spaces is given by the surrogate model they use. Models like Random Forests or the Tree Parzen Estimator (TPE – Bergstra et al. (2011)) support conditional attributes. A successful SMBO approach using random forests is SMAC (Sequential Model-based Algorithm Configuration by Hutter et al. (2011)) where an ensemble of decision trees makes it possible to model conditional variables. Another state-of-the-art approach uses TPE, where a graph-structured model matches the conditional structure of the search space. Other SMBO approaches use Gaussian processes as surrogate models (e.g. Snoek et al. (2012)). However, they

Name	Surrogate model	Language	URL
SMAC	Random forest	Java	http://www.cs.ubc.ca/labs/beta/Projects/SMAC
Hyperopt	Tree Parzen estimator	Python	https://github.com/hyperopt/hyperopt
Spearmint	Gaussian process	Python	https://github.com/HIPS/Spearmint
Bayesopt	Gaussian process	C++	https://bitbucket.org/rmcantin/bayesopt
PyBO	Gaussian process	Python	https://github.com/mwhoffman/pybo
MOE	Gaussian process	Python / C++	https://github.com/Yelp/MOE
Scikit-Optimize	Various	Python	https://scikit-optimize.github.io
Auto-WEKA*	SMAC,TPE	Java	https://github.com/automl/autoweka
Auto-Sklearn*	SMAC	Python	https://github.com/automl/auto-sklearn

* Toolkits for automating algorithm selection in WEKA and Scikit-learn, respectively.

Table 4.1: *Popular open-source tools supporting SMBO methods*

cannot work in conditional spaces because standard kernels are not defined over variable-length spaces (Shahriari et al. (2016)) and therefore are not used in this thesis. A list of popular open-source tools for running SMBO methods is detailed in Table 4.1.

4.2.3 Extension and generalisation of Auto-WEKA

In order to apply this research to process industry datasets, a substantial amount of time has been spent on developing an extension of Auto-WEKA tool to support automatic composition and optimisation of MCPSs. The search space of this tool now consists of all the available WEKA filters, predictors and meta-predictors with the constraint of limiting the number of components to follow the framework described in Section 1.4.

Auto-WEKA is a software developed by Thornton et al. (2013) which allows algorithm selection and hyperparameter optimisation both in regression and classification problems. The current Auto-WEKA version is 2.1 and it provides a blackbox interface for the CASH problem as expressed in Equation 3.4, where the search space is defined by WEKA predictors and meta-predictors.

As part of this thesis, Auto-WEKA 0.5 was extended due to its flexibility for the necessary extensions to model MCPSs, not present in the current blackbox version. Both versions provide a one-click solution for automating algorithm selection and hyperparameter optimisation. However, version 0.5 is much more flexible, offering multiple customisation possibilities like pre-selection of WEKA predictors, choosing the optimisation strategy or setting the optimisation criteria. Auto-WEKA also supports various usage scenarios depending on user knowledge, needs and available computational budget. One can for example, run several optimisations in parallel, ending up with multiple solutions that can then be analysed individually or used to build an ensemble (Feurer et al. (2015)).

The search space is automatically generated by Auto-WEKA by inspecting what WEKA methods are compatible with the given dataset. The hyperparameters of these methods are also included in the search space. However, the original Auto-WEKA implementation had the limitation of not including second-level hyperparameters (e.g. the WEKA implementation of support vector classifiers allows to select between different kernel types which contain their own hyperparameters). As part of this new Auto-WEKA

extension, this limitation has been sorted out (see Martin Salvador et al. (2016c)). Automatic creation of conditional hyperparameters is therefore now possible by recursively 'expanding' complex hyperparameters (i.e. hyperparameters containing further hyperparameters) during the generation of the search space.

For the purpose of this thesis, Auto-WEKA has been extended to support the composition of MCPS (Martin Salvador et al. (2016a)). Any WEKA filter can now be included as part of the composition process. In addition, a new WEKA filter has been developed to create flexible chains of common preprocessing steps including missing value handling, outlier removal, data transformation, dimensionality reduction and sampling.

The following external WEKA packages[1] have been included as part of the developed extended Auto-WEKA tool to increase the number of preprocessing methods: *EMImputation*, *RBFNetwork*, *StudentFilters*, *baggedLocalOutlierFactor*, *localOutlierFactor*, *partialLeastSquares* and *wavelet*. Furthermore, a new filter has been created to combine any outlier detection method and their removal in a single step. A new data sampling filter for WEKA has been implemented in which instances are periodically selected given a fixed interval of time – this is common in process industry to reduce the size of datasets.

Moreover, this Auto-WEKA extension generates an MCPS in a PNML (Petri Net Markup Language) format which can be analysed using any tool supporting this standard language (e.g. WoPeD[2]). Therefore, there are three main outputs once a new, extended Auto-WEKA run has finished: a) the trained MCPS ready to make predictions on unseen data; b) WEKA configuration (i.e. parametrised components); c) the Petri net in a PNML format.

This extended version of Auto-WEKA and all the scripts for the analysis of the results such as the creation of plots and tables have been also released in the BU's Data Science Initiative repository[3].

4.3 Experiments

A major limitation of the study presented in Thornton et al. (2013) was the use of only one preprocessing step (i.e. feature selection) whose optimisation was decoupled from model building. That is, both feature selection and predictor were individually optimised and therefore selected features were not based on their predictive power in conjunction with the learning algorithm used in the subsequent step, but on some other criteria (e.g. correlation with the target variable). The extended CASH problem presented in this thesis includes not only the feature selection but any additional preprocessing components. The proposed approach extends the work of Thornton et al. (2013) to support joint optimisation of predictive models (classifiers and regressors), preprocessing and postprocessing components (in short, MCPS).

This section describes the experiments carried out to compare the results of au-

[1] `http://weka.sourceforge.net/packageMetaData/`
[2] `http://woped.dhbw-karlsruhe.de/woped/`
[3] `https://github.com/dsibournemouth/autoweka`

tomating the composition and optimisation of MCPSs in different search spaces. For a fair comparison with the experiments in Thornton et al. (2013), the same datasets and methodology have been followed.

4.3.1 Methodology

The three main characteristics which define a CASH problem are: a) the search space, b) the objective function and c) the optimisation algorithm.

In this study three search spaces of very different sizes have been considered (see Table 4.2):

- PREV: This is the search space used in Thornton et al. (2013) where predictors and meta-predictors (which take outputs from one or more base predictive models as their input) were considered (756 hyperparameters). Feature selection is also performed as a preprocessing step before the optimisation process (30 hyperparameters). It is used as a baseline.
- NEW: This search space only includes predictors and meta-predictors. In contrast with PREV space, no previous feature selection stage is performed. Please note however that some WEKA classifiers perform internal preprocessing steps (e.g. MultiLayerPerceptron (MLP) removes instances with missing values and scales the attributes to range [-1,1]), as shown in Martin Salvador et al. (2016c). A categorical hyperparameter can be either simple or complex (i.e. when it contains WEKA classes). In the latter case, the search space is increased by adding recursively the hyperparameters of each method belonging to such complex parameter (e.g. the 'DecisionTable' predictor contains a complex hyperparameter whose values are three different types of search methods with further hyperparameters – see Table 4.4 for details). That extension increases the space to 1186 hyperparameters.
- FULL: This search space has been defined to support a flow with up to five preprocessing steps, a predictive model and a meta-predictor (1564 hyperparameters). The transitions are connected in the following order: missing value imputation; outlier detection and removal; data transformation; dimensionality reduction; sampling; predictor; meta-predictor. This flow is based on the framework described in Section 1.4, though these preprocessing steps are also common in other fields. If the meta-predictor transition is either 'Stacking' or 'Vote', its number of inputs can vary from 1 to 5.

	Missing Value	Outlier Handling	Data Transf.	Dimens. Reduction	Sampling	Predictor	Meta Predictor	Complex Hyperp.	Size*
PREV				✓		✓	✓		21,560
NEW						✓	✓	✓	2,369,598
FULL	✓	✓	✓	✓	✓	✓	✓	✓	812 billion

Table 4.2: *Summary of search spaces. * Size considering only methods, categorical and complex hyperaparameters.*

The methods that can be included in each component are listed in Tables 4.3 and 4.4. Note, that FULL search space is more than twice as large as the one presented in Thornton et al. (2013) in terms of the raw number of hyperparameters.

Method	Num.	Categorical	
		Simple	Complex
Missing values (MV)			
No handling	0	0	0
ReplaceMissingValues	0	0	0
CustomReplaceMissingValues	0	M	0
↪ (M) Zero	0	0	0
↪ (M) Mean	0	0	0
↪ (M) Median	0	0	0
↪ (M) Min	0	0	0
↪ (M) Max	0	0	0
↪ (M) LastKnown	0	0	0
EMImputation	3	1	0
Outliers (OU)			
No handling	0	0	0
RemoveOutliers	0	0	O
↪ (O) InterquartileRange (IQR)	2	0	0
↪ (O) BaggedLOF	1	1	0
Transformation (TR)			
No transformation	0	0	0
Center	0	0	0
Standardize	0	0	0
Normalize	2	0	0
Wavelet	0	0	0
IndependentComponents	3	1	0

Method	Num.	Categorical	
		Simple	Complex
Dimensionality reduction (DR)			
No reduction	0	0	0
PrincipalComponents (PCA)	3	1	0
RandomSubset	2	0	0
AttributeSelection	0	0	S,E
↪ (S) BestFirst	1	1	0
↪ (S) GreedyStepwise	2	3	0
↪ (S) Ranker	1	0	0
↪ (E) CfsSubsetEval	0	2	0
↪ (E) CorrelationAttributeEval	0	0	0
↪ (E) GainRatioAttributeEval	0	0	0
↪ (E) InfoGainAttributeEval	0	2	0
↪ (E) OneRAttributeEval	2	1	0
↪ (E) PrincipalComponents	2	3	0
↪ (E) ReliefFAttributeEval	2	1	0
↪ (E) Sym.UncertAttributeEval	0	1	0
↪ (E) WrapperSubsetEval	0	0	0
PLSFilter	1	4	0
Sampling (SA)			
No sampling	0	0	0
Resample	2	0	0
ReservoirSample	2	0	0
Periodic sampling	1	0	0

Table 4.3: *Number of parameters of the available preprocessing methods*

As the datasets used in the experiments are intended for classification, the goal is to minimise the classification error averaged over 10 cross-validation folds within the optimisation process.

Two SMBO strategies (SMAC and TPE) have been compared against two baselines (WEKA-Def and random search). The following experimental scenarios have been devised:

- **WEKA-Def**: All the predictors and meta-predictors listed in Table 4.4 are run using WEKA's default hyperparameter values. Filters are not included in this strategy, although some predictors may perform specific preprocessing steps as part of their default behaviour.
- **Random search**: The whole search space is randomly explored allowing 30 CPU core-hours for the process.
- **SMAC** and **TPE**: An initial configuration is randomly selected and then the optimiser is run for 30 CPU core-hours to explore the search space in an intelligent way, allowing for comparison with the random search.

In order to compare the results with the ones presented in Thornton et al. (2013) the experimental settings have been replicated as closely as possible. Different optimisation strategies have been evaluated over 21 well-known datasets representing classification (see Table 4.5). Each dataset $\mathcal{D} = \{\mathcal{D}_{train}, \mathcal{D}_{test}\}$ has been split into 70% training and 30% testing sets, unless partition was already provided. Please note that $\mathcal{D}_{train}$ is then

Method	Num.	Categorical	
		Simple	Complex
Predictors (P)			
BayesNet	0	1	Q
↪ (Q) local.K2	1	4	0
↪ (Q) local.HillClimber	1	4	0
↪ (Q) local.LAGDHillClimber	3	4	0
↪ (Q) local.SimulatedAnnealing	3	2	0
↪ (Q) local.TabuSearch	3	4	0
↪ (Q) local.TAN	0	2	0
NaiveBayes	0	2	0
NaiveBayesMultinomial	0	0	0
Logistic	1	0	0
MLP	6	5	0
SMO	1	2	K
↪ (K) NormalizedPolyKernel	1	1	0
↪ (K) PolyKernel	1	1	0
↪ (K) Puk	2	0	0
↪ (K) RBFKernel	1	0	0
SimpleLogistic	0	3	0
IBk	1	4	A
↪ (A) BallTree	0	1	2
↪ (A) CoverTree	1	1	1
↪ (A) KDTree	2	1	2
↪ (A) LinearNNSearch	0	1	0
KStar	1	2	0
DecisionTable	0	3	S
↪ (S) BestFirst	1	2	0
↪ (S) GreedyStepwise	2	3	0
↪ (S) Ranker	1	0	0
JRip	2	2	0
OneR	1	0	0
PART	2	2	0
ZeroR	0	0	0
DecisionStump	0	0	0
J48	2	5	0
LMT	1	6	0
REPTree	2	2	0
RandomForest	3	2	0
RandomTree	4	4	0

Method	Num.	Categorical	
		Simple	Complex
Meta-predictors (MP)			
LWL	0	2	A
↪ (A) BallTree	0	1	2
↪ (A) CoverTree	1	1	1
↪ (A) KDTree	2	1	2
↪ (A) LinearNNSearch	0	1	0
AdaBoostM1	2	3	0
AttributeSelectedClassifier	0	0	S,E
↪ (S) BestFirst	1	1	0
↪ (S) GreedyStepwise	2	3	0
↪ (S) Ranker	1	0	0
↪ (E) CfsSubsetEval	0	2	0
↪ (E) GainRatioAttributeEval	0	0	0
↪ (E) InfoGainAttributeEval	0	2	0
↪ (E) OneRAttributeEval	2	1	0
↪ (E) WrapperSubsetEval	0	0	0
Bagging	2	1	0
ClassificationViaRegression	0	0	0
FilteredClassifier	0	0	0
LogitBoost	5	4	0
MultiClassClassifier	1	2	0
RandomCommittee	1	0	0
RandomSubSpace	2	0	0
Stacking	0	1	0
Vote	0	1	0

Table 4.4: *Number of parameters of the available predictors*

split into 10-folds for Equation 3.5 and therefore $\mathcal{D}_{test}$ is not used during the optimisation or training process at all.

For each strategy 25 runs are performed with different random seeds within a 30 CPU core-hours optimisation time limit on Intel Xeon E5-2620 six-core 2.00GHz CPU. In the case a configuration step exceeds 30 minutes or 3GB of RAM to evaluate, its evaluation is aborted and not considered further. Once the optimisation process has finished, the returned MCPS is trained using the whole training set $\mathcal{D}_{train}$ and produce predictions for the testing set $\mathcal{D}_{test}$.

4.3.2 Results

The analysis of results is organised around the following aspects: a) impact of significantly extending the search space in the optimisation process; b) identification of promis-

Dataset	Continuous	Discrete	Classes	Train	Test	%Missing
abalone	7	1	28	2924	1253	0
amazon	10000	0	50	1050	450	0
car	0	6	4	1210	518	0
cifar10	3072	0	10	50000	10000	0
cifar10small	3072	0	10	10000	10000	0
convex	784	0	2	8000	50000	0
dexter	20000	0	2	420	180	0
dorothea	100000	0	2	805	345	0
germancredit	7	13	2	700	300	0
gisette	5000	0	2	4900	2100	0
kddcup09app	192	38	2	35000	15000	69.47
krvskp	0	36	2	2238	958	0
madelon	500	0	2	1820	780	0
mnist	784	0	10	12000	50000	0
mnistrot	784	0	10	12000	50000	0
secom	590	0	2	1097	470	4.58
semeion	256	0	10	1116	477	0
shuttle	9	0	7	43500	14500	0
waveform	40	0	3	3500	1500	0
wineqw	11	0	11	3429	1469	0
yeast	8	0	10	1039	445	0

Table 4.5: *Datasets, number of continuous attributes, number of categorical attributes, number of classes, number of instances, and percentage of missing values.*

ing methods for each dataset; and c) trade-off between exploration and exploitation in the search strategies.

Impact of extending the search space

Classification performance for each dataset can be found in Tables 4.6 and 4.7, which show the 10-fold cross-validation error over $\mathcal{D}_{train}$ (denoted as $\epsilon = \frac{1}{10}\sum_{i=1}^{10}\mathcal{L}(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)})$, where $\mathcal{L}$ is the classification error and $\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}$ the ground truth and the predictions for the fold i, respectively) and the holdout error over $\mathcal{D}_{test}$ (denoted as $\mathcal{E} = \mathcal{L}(\boldsymbol{y}_{test}, \hat{\boldsymbol{y}}_{test})$) achieved by each strategy, respectively (see Figure 4.2 for the overall process). Random search, SMAC and TPE results have been calculated using the mean of 100,000 bootstrap samples (i.e. randomly selecting 4 of the 25 runs and keeping the one with lowest cross-validation error as done in Thornton et al. (2013)), while only the lowest errors are reported for WEKA-Def. PREV columns contain the values reported in Thornton et al. (2013), while NEW and FULL columns contain the results for the search spaces described in Section 4.3.1. An upward arrow indicates an improvement when using extended search spaces (NEW and FULL) in comparison to previous results (PREV) reported in Thornton et al. (2013). Boldface values indicate the lowest classification error for each dataset.

dataset	WEKA-DEF	RANDOM			SMAC			TPE		
		PREV	NEW	FULL	PREV	NEW	FULL	PREV	NEW	FULL
abalone	73.33	72.03	72.53	72.93	**71.71**	72.21	72.20	72.14	72.01 ↑	72.17
amazon	43.94	59.85	45.72 ↑	62.22	47.34	**39.57** ↑	53.51	50.26	40.27 ↑	56.26
car	2.71	0.53	0.47 ↑	2.58	0.61	0.38 ↑	0.46 ↑	0.91	**0.21** ↑	0.29 ↑
cifar10	65.54	69.46	58.89 ↑	66.18 ↑	62.36	56.44 ↑	64.76	67.73	**55.59** ↑	70.40
cifar10small	66.59	67.33	60.45 ↑	70.32	58.84	57.90 ↑	71.05	58.41	**56.56** ↑	68.65
convex	28.68	33.31	25.02 ↑	34.82	25.93	**21.88** ↑	25.52	28.56	23.19 ↑	29.93
dexter	10.20	10.06	7.54 ↑	8.92 ↑	**5.66**	6.42	7.51	9.83	6.19 ↑	7.76 ↑
dorothea	6.03	8.11	6.25 ↑	6.94 ↑	**5.62**	5.95	6.58	6.81	5.92 ↑	6.16 ↑
germancredit	22.45	20.15	21.31	21.80	**17.87**	19.65	20.49	21.56	19.88 ↑	20.07 ↑
gisette	3.62	4.84	2.30 ↑	3.66 ↑	2.43	**2.21** ↑	2.76	3.55	2.35 ↑	3.07 ↑
kddcup09app	1.88	1.75	1.80	**1.70** ↑	**1.70**	1.80	1.80	1.88	1.80 ↑	1.80 ↑
krvskp	0.89	0.63	0.42 ↑	0.56 ↑	0.30	**0.28** ↑	0.35	0.43	0.31 ↑	0.33 ↑
madelon	25.98	27.95	19.20 ↑	27.60 ↑	20.70	**15.61** ↑	17.24 ↑	24.25	16.03 ↑	19.14 ↑
mnist	5.12	5.05	3.78 ↑	7.85	3.75	**3.50** ↑	5.51	10.02	3.60 ↑	7.19 ↑
mnistr	66.15	68.62	58.10 ↑	67.21 ↑	57.86	**55.73** ↑	63.62	73.09	57.17 ↑	65.89 ↑
secom	6.25	5.27	5.85	**4.87** ↑	5.24	6.01	6.10	6.21	5.85 ↑	6.05 ↑
semeion	6.52	6.06	4.82 ↑	7.85	4.78	4.48 ↑	4.92	6.76	**4.28** ↑	5.96 ↑
shuttle	0.0328	0.0345	0.0121 ↑	0.0349	0.0224	0.0112 ↑	0.0112 ↑	0.0251	**0.0104** ↑	0.0112 ↑
waveform	12.73	12.43	12.50	13.09	**11.92**	12.33	12.48	12.55	12.43 ↑	12.42 ↑
wineqw	38.94	35.36	33.08 ↑	35.95	34.65	**32.64** ↑	33.24 ↑	35.98	32.67 ↑	32.97 ↑
yeast	39.43	38.74	37.16 ↑	39.63	35.51	36.50	37.23	**35.01**	36.17	36.43

Table 4.6: *10-fold Cross Validation error ϵ (% missclassification). An upward arrow indicates an improvement with respect to PREV space. Boldfaced values indicate the lowest classification error for each dataset.*

dataset	WEKA-DEF	RANDOM			SMAC			TPE		
		PREV	NEW	FULL	PREV	NEW	FULL	PREV	NEW	FULL
abalone	73.18	74.88	**72.92** ↑	73.76 ↑	73.51	73.41 ↑	73.16 ↑	72.94	73.04	73.26
amazon	**28.44**	41.11	39.22 ↑	58.94	33.99	36.26	49.90	36.59	35.69 ↑	62.52
car	0.7700	0.0100	0.1300	1.8400	0.4000	0.0526 ↑	0.1958 ↑	0.1800	**0.0075** ↑	0.0484 ↑
cifar10	64.27	69.72	58.23 ↑	67.81 ↑	61.15	55.54 ↑	69.99	66.01	**54.88** ↑	70.06
cifar10small	65.91	66.12	59.84 ↑	72.93	56.84	57.87	72.64	57.01	**56.41** ↑	72.58
convex	25.96	31.20	24.75 ↑	36.87	23.17	**21.31** ↑	31.05	25.59	22.62 ↑	34.88
dexter	8.89	9.18	8.29 ↑	12.59	7.49	7.31 ↑	10.14	8.89	**6.90** ↑	7.99 ↑
dorothea	6.96	5.22	5.27	5.37	6.21	**5.12** ↑	5.51 ↑	6.15	5.25 ↑	5.15 ↑
germancredit	27.33	29.03	**25.40** ↑	26.88 ↑	28.24	25.42 ↑	26.68	27.54	25.49 ↑	26.61 ↑
gisette	2.81	4.62	2.28 ↑	3.83 ↑	**2.24**	2.34	2.85	3.94	2.37 ↑	3.08 ↑
kddcup09app	1.7405	1.74	**1.72** ↑	2.05	1.7358	1.74	1.74	1.7381	1.74	1.74
krvskp	0.31	0.58	0.34 ↑	0.39 ↑	0.31	**0.23** ↑	0.39	0.54	0.36 ↑	0.33 ↑
madelon	21.38	24.29	19.10 ↑	24.48	21.56	**16.80** ↑	18.26 ↑	21.12	16.91 ↑	18.35 ↑
mnist	5.19	5.05	4.00 ↑	13.92	**3.64**	4.10	22.70	12.28	3.96 ↑	25.48
mnistr	63.14	66.4	57.16 ↑	69.67	57.04	**54.86** ↑	67.03	70.20	56.31 ↑	70.04 ↑
secom	8.09	8.03	7.88 ↑	8.20	8.01	7.87 ↑	7.87 ↑	8.10	**7.84** ↑	7.87 ↑
semeion	8.18	6.10	**4.78** ↑	8.27	5.08	5.10	5.46	8.26	4.91 ↑	6.31 ↑
shuttle	0.0138	0.0157	0.0071 ↑	0.0219	0.0130	0.0070 ↑	0.0075	0.0145	**0.0069** ↑	0.0077 ↑
waveform	14.40	14.27	14.26 ↑	14.28	14.42	14.17 ↑	**13.99** ↑	14.23	14.34	14.04 ↑
wineqw	37.51	34.41	32.99	36.72	33.95	**32.90** ↑	34.14	33.56	32.93 ↑	34.09
yeast	40.45	43.15	37.68 ↑	40.86 ↑	40.67	**37.60** ↑	39.02 ↑	40.10	37.89 ↑	38.92 ↑

Table 4.7: *Holdout error $\mathcal{E}$ (% missclassification). An upward arrow indicates an improvement with respect to PREV space. Boldfaced values indicate the lowest classification error for each dataset.*

The first thing to note is that, on average, SMBO approaches are considerably better than the best WEKA-Def results. The only exception is the holdout error of 'amazon' dataset. However, the best solution in the FULL search space is still better than WEKA-Def (20.89% vs 28.44%). That result is expected since default values for WEKA methods have been set by human experts and are not optimised for any particular dataset.

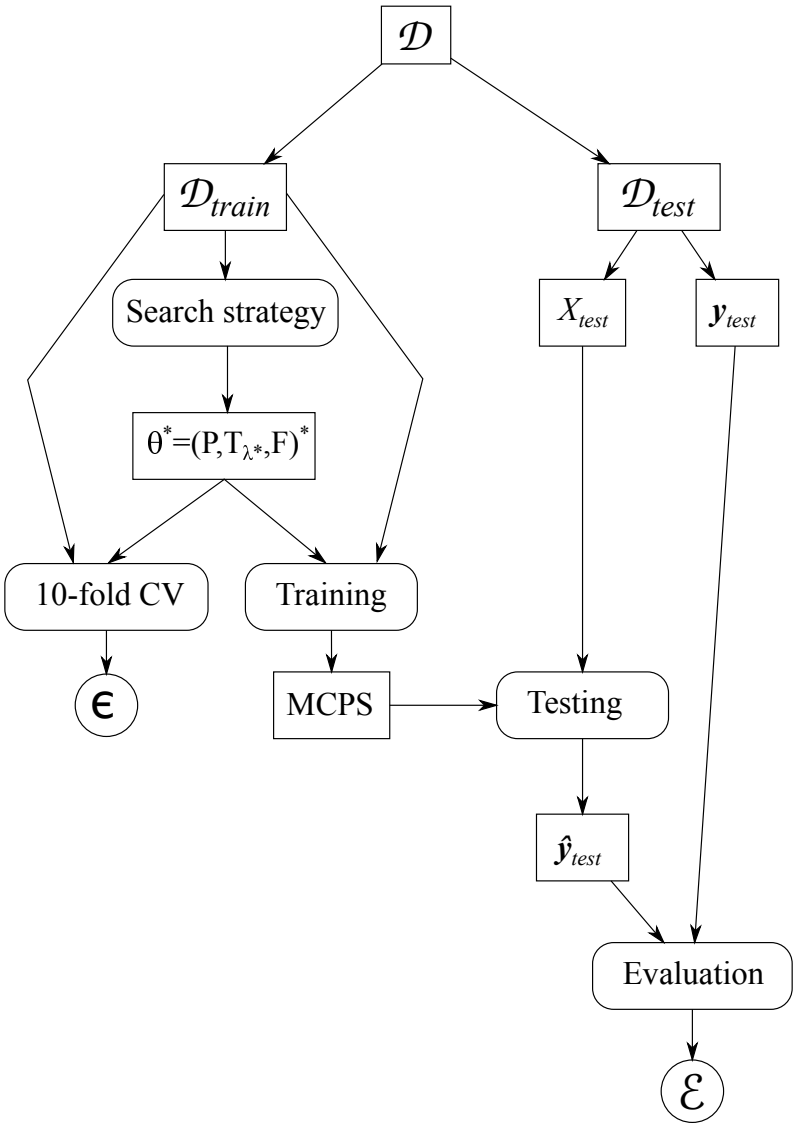


Figure 4.2: *MCPS training and testing process. The dataset is split into $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$ sets. The search space Θ is explored using the 10-fold CV performance on $\mathcal{D}_{train}$ as guidance. Then, the resultant MCPS $\theta^* = (P, T_{\lambda^*}, F)^*$ is trained on $\mathcal{D}_{train}$ and evaluated on D_{test} to calculate the holdout error $\mathcal{E}$.*

As shown in Table 4.6, in the majority of cases (49 of 63), the MCPSs found in the NEW search space achieve better results (i.e. $\epsilon_{NEW} < \epsilon_{PREV}$). In 28 out of 63 cases, the FULL search space also gets better performance (i.e. $\epsilon_{FULL} < \epsilon_{PREV}$). However, finding good MCPS is more challenging due to the large increase in search space size (Hoos et al. (2014)). As an example, consider Figure 4.3 where the evolution of the best solution for 'madelon' dataset and SMAC strategy is represented over time for each of the 25 runs. Comparing Figures 4.3-a) and b) one can see that the rate of convergence is much higher in the smaller space (denoted as NEW). Nevertheless, the overall best-performing model for 'madelon' was found in the FULL space as seen in Table 4.9.

The way in which the search space is extended can have a considerable impact on the accuracy of the MCPS found. Additional hyperparameters allowing for extra tuning flexibility (PREV to NEW) improved the performance in most of the cases. However, adding more transitions to the MCPS (NEW to FULL) does not seem to help on average, given the same CPU time limit. Nevertheless, the MCPSs found in the FULL search space for 8 out of 21 datasets have better or comparable performance to the solutions found in the NEW space as shown in Table 4.9.

Identifying promising configurations

The best MCPSs found for each dataset are reported in Table 4.9. Each row of this table represents a sequence of data transformations and predictive models as explained in Section 4.3.1. See for example Figures 4.4 and 4.5, where the best MCPSs of 'kddcup09app' and 'amazon' datasets in the FULL search space are shown. Transitions are effectively WEKA methods that process the incoming tokens. The solutions found for different datasets are quite diverse, and they often also vary a lot across the 25 random seed runs performed for each dataset. In order to better understand the observed differences in the found MCPSs, the average pairwise similarity of the 25 MCPSs found, and the variance of their performances have been measured for each dataset (see Figure 4.7). To calculate the similarity between configurations a weighted sum of Hamming distances given by

$$d(\theta_a, \theta_b) = 1 - \frac{\sum_{i=1}^{n} (\omega_i \cdot \delta_i)}{\sum_{i=1}^{n} \omega_i} \tag{4.4}$$

is used, where θ_a and θ_b are MCPSs with n transitions, $\omega_i \in \Omega$ is the weight for the ith transition and δ_i is the Hamming distance (a standard measure of string dissimilarity) of components at position i.

Weights have been fixed manually to $\Omega = \{2, 1.5\}$ in the NEW search space and $\Omega = \{1, 1, 1, 1, 1, 2, 1.5\}$ in the FULL search space. One could however set the weights in a different way depending on what components are believed to be more relevant. In this case, preprocessing transitions have the same weight while both predictors and meta-predictors have higher weights because of their importance (Hoos et al. (2014)). An analysis to future work could be to investigate the impact of the weights in the generated dendrograms.

After computing the similarity matrix for the 25 MCPSs found in each dataset/strategy pair, they are grouped using a complete-linkage hierarchical clustering algorithm. The resultant dendrogram for the 'waveform' dataset and SMAC strategy in the NEW search space (i.e. only predictor and meta-predictor as transitions) is shown in Figure 4.6a. As it can be seen, there are three main clusters defined by the base classifier which have been found to perform best: SimpleLogistic, Logistic and LMT (Logistic Model Tree). The dendrogram for the FULL scenario is shown in Figure 4.6b. Most of the MCPSs also use logistic classifiers (as in the smaller search space) but other good solutions include SMO (support vector classifier) and JRip (rule-based classifier). It can also be noted that the clusters themselves are no longer as pure as before in terms of the base classifier chosen.

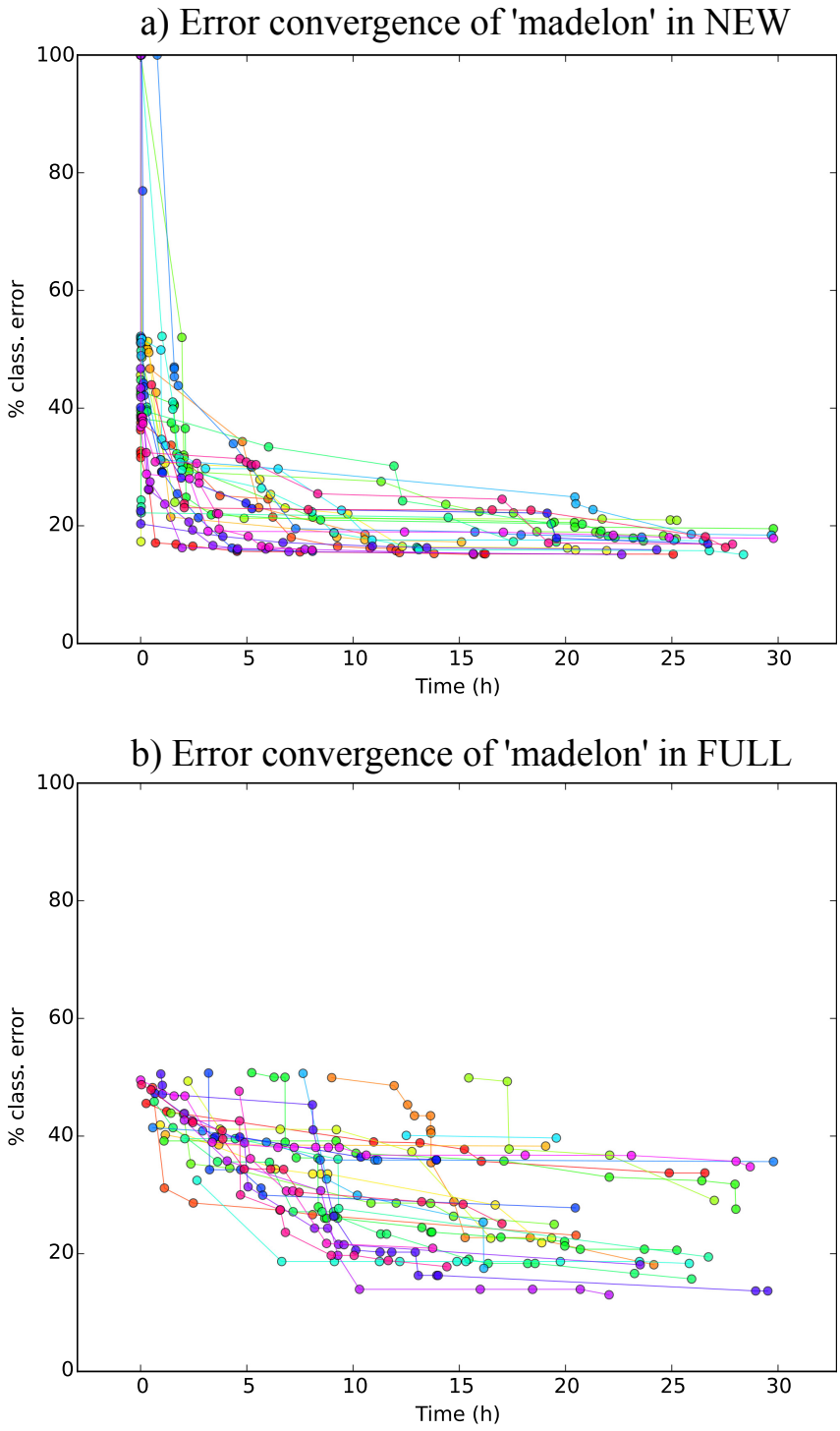


Figure 4.3: *10-fold CV error of best solutions found over time for 'madelon' dataset and SMAC strategy in a) NEW and b) FULL search spaces. Each colour represents a different run. Solutions in b) are converging much slower than in a) due to the increased complexity of exploring a much larger search space.*

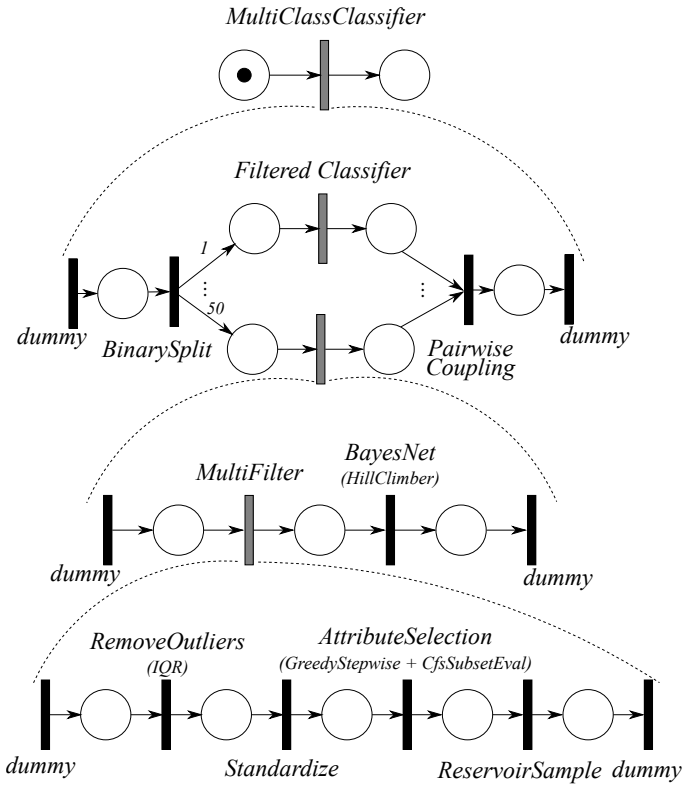


Figure 4.4: *The best MCPS in FULL search space for 'kddcup09app' dataset as shown in Table 4.9. The type and name of the preprocessing methods and predictors used are explained in Tables 4.3 and 4.4 respectively.*

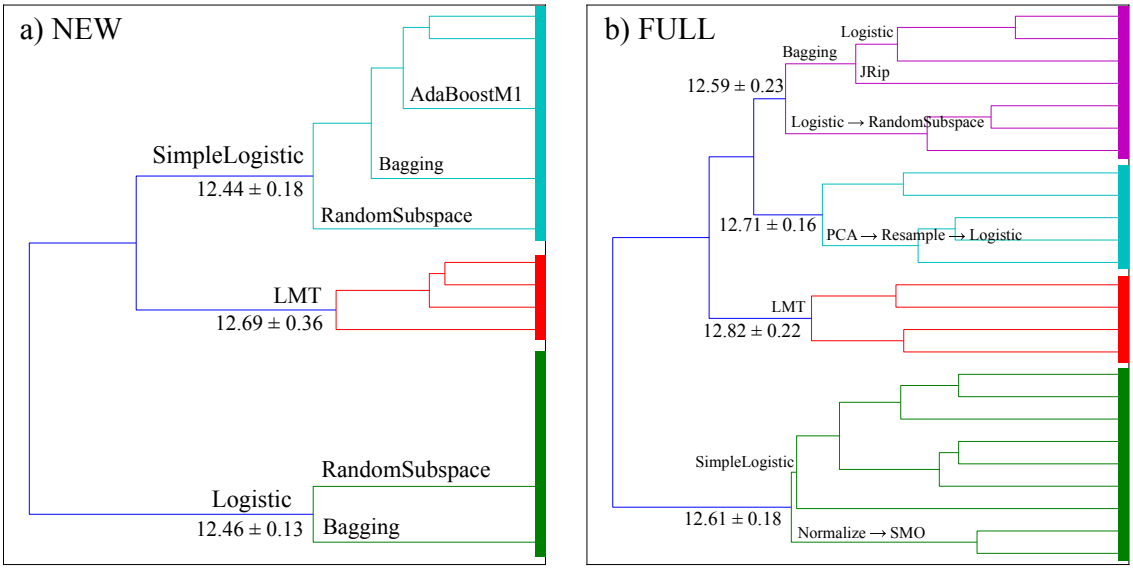


Figure 4.6: *Dendrogram for 'waveform' dataset and SMAC strategy in the (a) NEW and (b) FULL search space. Average CV error and standard deviation are shown for the main clusters. There are more clusters in (b) because of the addition of preprocessing components.*

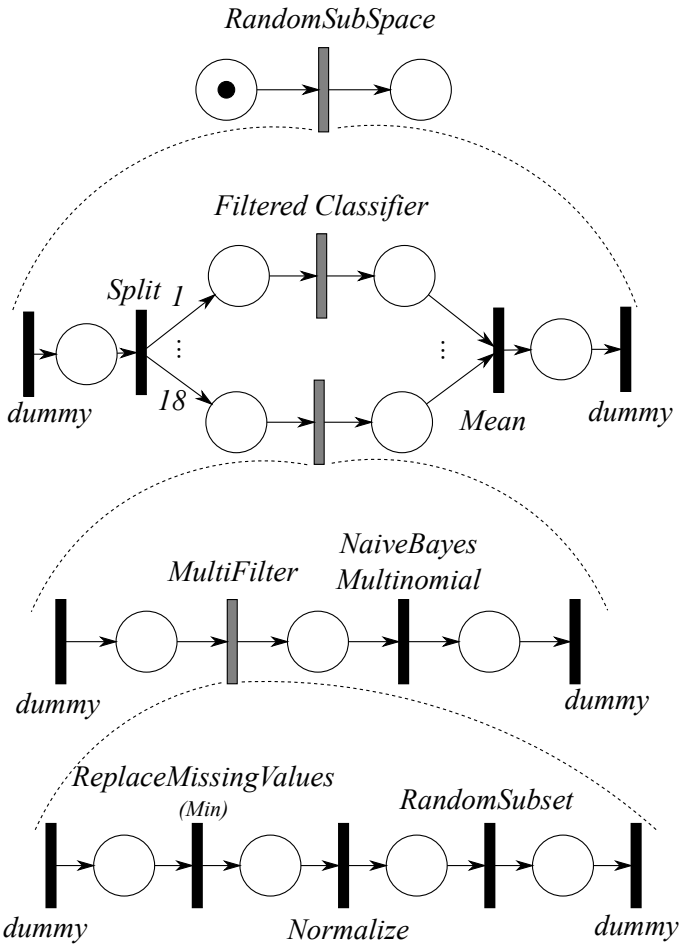


Figure 4.5: *The best MCPS in FULL search space for 'amazon' dataset as shown in Table 4.9. The type and name of the preprocessing methods and predictors used are explained in Tables 4.3 and 4.4 respectively.*

It is worth mentioning that most of the MCPSs found for the 'waveform' dataset include a missing value replacement method even though there are no missing values in this dataset and therefore it does not have any effect on the data or the classification performance. The presence of such an unnecessary component likely stems from the fact that selecting a method for replacing missing values at random has a prior probability of 0.75 (i.e. 3 out of 4 possible actions as seen in Table 4.3) which means that it can be selected when randomly initialising the configurations of MCPSs to start from and using the search method which does not penalise unnecessary elements in the data processing chains. However, it is not the case with other nodes like 'Transformation' in which although the prior probability of selecting one of the available transformation methods is 5/6, selecting an appropriate method has a potential impact on the performance and therefore better transitions tend to be retained in the found solutions.

For illustrative purposes three interesting cases from Figure 4.7 have been selected for a more detailed analysis. These cases are:

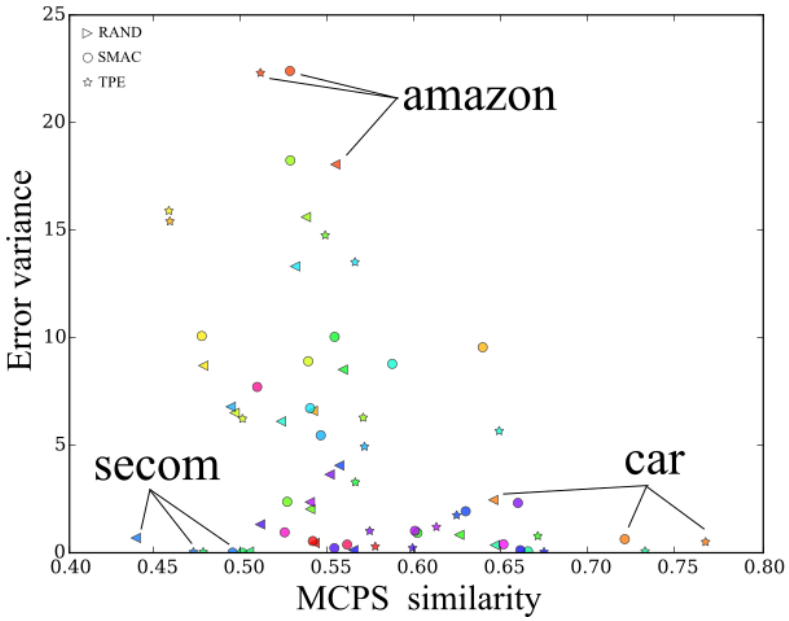


Figure 4.7: *Error variance vs. MCPS similarity in FULL search space. Each colour represents a different dataset.*

- Low error variance and high MCPS similarity. Most of the best solutions found follow a very similar sequence of methods. Therefore similar classification performance is to be expected. For example, a repeated sequence in 'car' dataset with TPE optimisation is MultiLayerPerceptron (13/25) → AdaBoostM1 (22/25).
- Low error variance and low MCPS similarity. Despite having different solutions, classification performance in a group of analysed datasets does not vary much. This can mean that the classification problem is not difficult and a range of different MCPSs can perform quite well on it. This is for instance the case of the solutions found for the 'secom' and 'kddcup09app' datasets.
- High error variance and low MCPS similarity. In such cases, there are many differences between both the best MCPSs found and their classification performances. For instance, it is the case of 'amazon' dataset for which a high error variance was observed in all of the optimisation strategies (see Figure 4.7). Such difference likely results from a combination of difficulty of the classification task (i.e. high input dimensionality, large number of classes and a relatively small number of training samples) and/or insufficient exploration from the random starting configuration in a very large space.

Exploration vs exploitation analysis

An inherent trade-off in a search through space with size exploding exponentially with the number of parameters, is that of exploration vs. exploitation. To study the impact of varying the relative importance of these two factors, two datasets ('amazon' and 'made-

lon') have been selected because their performance convergence changes considerably between different runs, and therefore the starting point in the SMAC search strategy makes a large difference. Both the number of seeds and the maximum optimisation time have been varied, keeping the same fixed CPU-core hour budget as in the previous experiments (i.e. 25 seeds x 30 h/seed = 750 h). Increasing the number of seeds to 50 allows to explore more regions of the search space. On the other hand, increasing the running time to 50 CPU core-hours/seed makes it possible to exploit further the hyperparameter settings of potential good configurations.

The results are presented in Table 4.8 which contains both the best and the mean of 10-fold cross validation error computed similarly as in Table 4.6. The results suggest that increasing the optimisation time (i.e. more exploitation) allows finding better solutions on average. However, in three of the cases the best solutions were found when there is a balance between exploration and exploitation (i.e. 30h and 25 seeds). This is not the case for 'amazon' dataset in the NEW search space, in which either increasing optimisation time or the number of random starting points (i.e. seeds) have both resulted in improved performance.

While this section describes a preliminary investigation into the potential trade-off in exploration versus exploitation aspects of the investigated search strategies, it also highlights that there is a need for significantly improving the fundamental search algorithms in the context of MCPS optimisation.

space	**dataset**	**15hx50seeds**		**30hx25seeds**		**50hx15seeds**	
		Best	Mean	Best	Mean	Best	Mean
NEW	amazon	**24.19**	32.09	30.57	36.45	**24.19**	<u>27.85</u>
	madelon	15.16	16.05	**15.11**	15.74	15.16	<u>15.47</u>
FULL	amazon	40.76	65.38	**26.76**	53.51	32.29	<u>47.04</u>
	madelon	14.45	21.65	**13.02**	17.24	15.22	<u>16.62</u>

Table 4.8: *10-fold CV error ϵ (% missclassification) varying the number of hours and seeds. Bold faced values are the lowest errors for each dataset in each search space. Similarly, underlining indicates the lowest mean error.*

4.4 Summary

This chapter has reviewed main approaches for tackling the CASH problem for building predictive systems. Ontologies are an effective tool to model search spaces. However, their use by AI planners require an extensive repository of experiments to get good results. Combining these approaches with the recent OpenML[4] repository could be an interesting future study. An alternative approach that has been shown to work well in high dimensional search spaces is Bayesian optimisation, and in particular SMBO.

[4] `http://www.openml.org/`

dataset	space	MV	OU	TR	DR	SA	predictor	meta-predictor	$\mathcal{E}$
abalone	NEW	-	-	-	-	-	MLP	RandomCommittee	71.43
	FULL	Median	-	Center	RandomSubset	Resample	Logistic	Bagging	72.39
amazon	NEW	-	-	-	-	-	SimpleLogistic	RandomSubSpace	26.67
	FULL	Min	-	Normalize	RandomSubset	-	NaiveBayesMult.	RandomSubSpace	20.89
car	NEW	-	-	-	-	-	SMO	MultiClassClassifier	0.00
	FULL	-	-	Standardize	-	Resample	SMO	AdaBoostM1	0.00
cifar10	NEW	-	-	-	-	-	RandomForest	MultiClassClassifier	52.28
	FULL	-	-	-	-	Resample	RandomTree	Bagging	59.63
cifar10small	NEW	-	-	-	-	-	RandomTree	MultiClassClassifier	54.48
	FULL	-	-	-	RandomSubset	-	RandomTree	AdaBoostM1	59.97
convex	NEW	-	-	-	-	-	RandomForest	AdaBoostM1	18.47
	FULL	-	-	Center	-	Resample	RandomTree	AdaBoostM1	22.97
dexter	NEW	-	-	-	-	-	DecisionStump	AdaBoostM1	5.00
	FULL	-	-	-	-	Resample	VotedPerceptron	AdaBoostM1	5.00
dorothea	NEW	-	-	-	-	-	OneR	RandomSubSpace	4.64
	FULL	-	-	Standardize	-	-	REPTree	LogitBoost	4.64
germancredit	NEW	-	-	-	-	-	LMT	Bagging	23.33
	FULL	Zero	-	Standardize	RandomSubset	-	LMT	Bagging	24.33
gisette	NEW	-	-	-	-	-	NaiveBayes	LWL	1.95
	FULL	-	-	-	-	-	VotedPerceptron	RandomSubSpace	1.52
kddcup09app	NEW	-	-	-	-	-	ZeroR	LWL	1.67
	FULL	-	IQR	Standardize	Attr. Selection	ReservoirSample	BayesNet	MultiClassClassifier	1.74
krvskp	NEW	-	-	-	-	-	JRip	AdaBoostM1	0.10
	FULL	-	-	Normalize	-	-	JRip	AdaBoostM1	0.21
madelon	NEW	-	-	-	-	-	REPTree	RandomSubSpace	15.64
	FULL	-	-	-	PCA	-	IBk	LogitBoost	12.82
mnist	NEW	-	-	-	-	-	SMO	MultiClassClassifier	2.66
	FULL	Zero	-	Center	-	-	J48	AdaBoostM1	5.15
mnistr	NEW	-	-	-	-	-	RandomForest	RandomCommittee	52.20
	FULL	Zero	-	Normalize	-	-	BayesNet	RandomSubSpace	56.33
secom	NEW	-	-	-	-	-	J48	AdaBoostM1	7.66
	FULL	-	-	Standardize	-	ReservoirSample	ZeroR	FilteredClassifier	7.87
semeion	NEW	-	-	-	-	-	NaiveBayes	LWL	3.98
	FULL	EM	-	-	PCA	-	SMO	FilteredClassifier	4.61
shuttle	NEW	-	-	-	-	-	RandomForest	AdaBoostM1	0.01
	FULL	-	-	Center	-	Resample	REPTree	AdaBoostM1	0.01
waveform	NEW	-	-	-	-	-	SMO	RandomSubSpace	14.00
	FULL	-	IQR	Normalize	-	-	SMO	Attr.SelectedClassifier	13.40
wineqw	NEW	-	-	-	-	-	RandomForest	AdaBoostM1	32.33
	FULL	Mean	-	Wavelet	Attr.Selection	-	IBk	RandomSubSpace	33.42
yeast	NEW	-	-	-	-	-	RandomForest	Bagging	36.40
	FULL	-	-	Normalize	-	-	RandomTree	Bagging	38.20

Table 4.9: *Best MCPS for each dataset in NEW and FULL spaces and its holdout error. MV = missing value replacement, OU = outlier detection and removal, TR = transformation, DR = dimensionality reduction, SA = sampling.*

State-of-the-art SMBO methods have been used in an extensive experimetal analysis comparing different search spaces over a number of datasets. This study demonstrates that it is indeed possible to automate the composition and optimisation of MCPSs. The novel formulation of MCPSs using WRI-WF nets proposed in Section 3.3 has became a theoretical foundation of the study, allowing to cast the problem within a rigorous mathematical framework (van der Aalst (1998a)). At the same time, it opens the door to formally verify that MCPSs are correctly composed which is still an outstanding and non-trivial problem (Sadiq et al. (2004)).

Results have indicated that Sequential Model-Based Optimisation (SMBO) strategies perform better than random search given the same time for optimisation in the majority of analysed datasets. The analysis of the trade-off between exploration and exploitation of the search space suggests however that increasing the optimisation time (i.e. more

exploitation) allows to find better solutions on average. Investing time into fine-tuning of hyperparameters of a solution seems worthwhile only if this can significantly and quickly improve the quality of the solution as otherwise the optimisation strategy can stall in a local minimum that keeps away the exploration from other promising regions of the search space. The above highlights a clear need for significantly improving the fundamental search algorithms in the context of MCPS optimisation.

In addition, it would also be valuable to investigate if using a different data partitioning of the training set like Density-Preserving Sampling (DPS – Budka & Gabrys (2010a)) instead of CV would make any difference in the optimisation process. This could have a considerable impact on SMAC strategy which discards potential poor solutions early in the optimisation process based on the performance on only a few CV folds. In case the folds used are not representative of the overall data distribution, which as shown in Budka & Gabrys (2013) can happen quite often with CV, the effect on the solutions found can be detrimental.

At the moment, available SMBO methods only support single objective optimisation. However, it would be useful to find solutions that optimise more than one objective, including for instance a combination of prediction error, model complexity and running time as discussed in Al-Jubouri & Gabrys (2014).

Application of these techniques to process industry datasets to automatically build soft sensors is one of the topics discussed in the next chapter, where also strategies for adapting MCPSs in evolving environments are presented.

Chapter 5

Automating and adapting MCPSs in the process industry

5.1 Introduction

The research carried out in this thesis was initiated in the scope of INFER project, in which one of the partners was a large chemical manufacturer. Working closely with domain experts was beneficial to understand the current limitations of soft sensor development in the process industry. The framework presented in Section 1.4 describes the steps that practitioners usually follow to build a soft sensor. A common issue in different works (Yan et al. (2004); Sharmin et al. (2006); Lin et al. (2007); Kadlec et al. (2009)) is the amount of human effort needed to complete the tasks of data preprocessing, algorithm selection, and hyperparameter optimisation. Chapters 3 and 4 have covered the proposed approach to automate this labour intensive task. This chapter studies the feasibility of applying such an approach for a number of real chemical plants and processes from the process industry. In order to address these issues, an extensive experimental analysis on automating the composition of MCPSs for classification (for process monitoring, in which classification models are trained to identify different states of the process and recognise possible process faults) and regression (for online prediction, in which regression models are used to predict hard-to-measure values in processes like fermentation, polymerisation and refinery) tasks have been carried out and the results are presented in Section 5.2.

Datasets from chemical production processes contain readings from physical sensors that are located in different parts of the chemical plants. These sensors measure parameters such as temperatures, flows and pressures that are constantly changing during chemical reactions. Some reactions are relatively stable and therefore present a stationary data distribution. Others, however, can vary significantly between production batches or even within a single, long-running production process (e.g. Sharmin et al. (2006)). In addition, the degradation of sensors over time, is likely to produce a change in the input values that can severely affect predictive performance (e.g. Downs & Vogel (1993)). These changes in data distribution are known as concept drifts. According to Kadlec et al. (2011), the

most common causes of concept drift in process industry are:

- changes of process input (raw) materials;
- process fouling (i.e. accumulation of unwanted material);
- abrasion of mechanic components;
- catalyst activity changes;
- production of different product quality grades;
- changes in external environment (e.g. weather, seasons).

There are different adaptation strategies for dealing with such changes (see e.g. Gama et al. (2014) for a general survey and Kadlec et al. (2011) which focused on adaptation mechanisms for soft sensors in process industry). For example, 'active-detection' techniques monitor a certain measure over time and react when its running average changes significantly or goes over a given threshold (e.g. Page (1954); Gama et al. (2004); Baena-García et al. (2006)). These techniques require optimisation of parameters like the threshold value or averaging period, and can also lead to false positives. On the other hand, 'passive' adaptation techniques update the model periodically with new data (e.g. most recent data or most representative samples), even if it is not strictly necessary (e.g. Widmer & Kubat (1993); Gabrys & Bargiela (2000); Klinkenberg (2004)). From a practical perspective, 'passive' adaptation may be a waste of resources and provide little benefits towards the predictive performance in problems where data distribution does not change too often. An intermediate approach presented in Žliobaitė et al. (2015) proposes to estimate the gain and cost of adaptation and therefore assess when it is worth to perform a model adaptation.

Adaptive soft sensors do not often consider the adaptation of preprocessing methods (Kadlec et al. (2011)). This situation can lead to the adaptation of predictors to an undesirable state or even failures (see Appendix C). Moreover, the maintenance of adaptive soft sensors can require expert knowledge to select new critical process variables (Fortuna et al. (2005); Kämpjärvi et al. (2008)). Taking this need of adaptation into account, Section 5.3 studies the feasibility of adapting MCPS simulating a continuous process environment. This aims once again to reduce the need for human involvement and thus make the maintenance process more efficient.

The goal of this chapter is to assess the feasibility of applying search strategies for automatically building and maintaining soft sensors in process industry. The chapter is organised as follows: Section 5.2 discuss the results of automating the composition and optimisation of MCPSs to solve the problems of online prediction and process monitoring in datasets from real chemical processes; Section 5.3 presents a novel hybrid adaptation strategy combining SMBO with common adaptive techniques and the results of applying such strategy on real datasets to fully automate the maintenance of soft sensors; finally, conclusions are in Section 5.4.

5.2 Automatic building of soft sensors

Raw data from chemical plants requires a considerable manual effort in preprocessing and modelling as demonstrated in Section 1.3. Thus, one of the main motivations in this thesis for automating the composition and optimisation of MCPSs has been the need for speeding up the process of developing soft sensors.

Two of the main tasks requiring soft sensors in this industry are: a) online prediction, in which regression models are used to predict hard-to-measure values in processes like fermentation, polymerisation and refinery; and b) process monitoring, in which classification models are trained to identify different states of the process and recognise possible process faults. To evaluate the feasibility of automatically building and optimising MCPSs for addressing both tasks in a real-world setting, a number of experiments using datasets representing real chemical processes have been performed (please refer to the Appendix A for further information and references). These datasets have been made available by Evonik Industries as part of the collaboration within the INFER project, and have been extensively used in previous studies (e.g. Kadlec & Gabrys (2009); Budka et al. (2014); Bakirov et al. (2015)). The datasets are:

- 'absorber' which contains 38 continuous attributes from an absorption process. No additional information has been provided apart from this being a regression task;
- 'drier' which consists of 19 continuous features from physical sensors (i.e. temperature, pressure and humidity) and the target value is the residual humidity of the process product (Kadlec & Gabrys (2009));
- 'oxeno' which contains 71 continuous attributes also from physical sensors and a target variable which is the product concentration measured in the laboratory (Budka et al. (2014)); and
- 'thermalox' which has 38 attributes from physical sensors and the two target values are concentrations of NO_x and SO_x in the exhaust gases (Kadlec & Gabrys (2009)).

Due to confidentiality reasons the datasets listed above cannot be published. However, 3 additional publicly available datasets from the same domain have also been used in the experiments. These are:

- 'catalyst' consisting of 14 attributes, where the task is to predict the activity of a catalyst in a multi-tube reactor (Kadlec & Gabrys (2011));
- 'debutanizer' which has 7 attributes (temperature, pressure and flow measurements of a debutanizer column) and where the target value is the concentration of butane at the output of the column (Fortuna et al. (2005)); and
- the 'sulfur' recovery unit, which is a system for removing environmental pollutants from acid gas streams before they are released into the atmosphere (Fortuna et al. (2003)). The washed out gases are transformed into sulfur. The dataset has five input features (flow measurements) and two target values: concentration of H_2S and SO_2.

The experimentation framework is similar to the one in Section 4.3 and therefore some

aspects are repeated here for consistency. The search space for composing MCPSs has been defined to support a flow with up to five preprocessing steps, a predictive model and a meta-predictor. The nodes are connected in the following order: missing value handling; outlier detection and handling[1]; data transformation; dimensionality reduction; sampling; predictor; meta-predictor. Although some of the predictors include inner preprocessing such as removal of missing values or data normalisation as shown in Martin Salvador et al. (2016c), many datasets still need additional preprocessing steps to build effective predictive models. Thus, the fixed order of the preprocessing nodes in the search space has not been set arbitrarily. Instead, it follows the preprocessing guidelines that are common in process industry when developing predictive models (see e.g. Budka et al. (2014); Kadlec et al. (2009)).

The best SMBO strategy from previous experiments in Chapter 4 (SMAC) is compared against random search as baseline:

- Random search: The whole search space is randomly explored allowing 30 CPU core-hours for the process.
- SMAC: An initial configuration is randomly selected and then the optimiser is run for 30 CPU core-hours to explore the search space.

Each dataset $\mathcal{D} = \{\mathcal{D}_{train}, \mathcal{D}_{test}\}$ has been split into 70% training and 30% testing sets, unless partition was already provided. Please note that $\mathcal{D}_{train}$ is then split into 10-folds for Equation 3.5 and therefore $\mathcal{D}_{test}$ is not used during the optimisation or training process at all. The evaluation process is summarised in Figure 4.2.

For each strategy 25 runs are performed with different random seeds within a 30 CPU core-hours optimisation time limit on Intel Xeon E5-2620 six-core 2.00GHz CPU. In the case a configuration step exceeds 30 minutes or 3GB of RAM to evaluate, its evaluation is aborted and not considered further. Once the optimisation process has finished, the returned MCPS is trained using the whole training set $\mathcal{D}_{train}$ and produce predictions for the testing set $\mathcal{D}_{test}$.

5.2.1 Online prediction: a regression problem

Many critical process values (e.g. the fermentation progress in a biochemical process, or the progress of polymerisation in a batch reactor) are difficult, if not impossible to measure in an automated way at a required sampling rate. Sometimes the first-principle models, that are based on the physical and chemical process knowledge, are available. Although such models are preferred by practitioners (De Assis & Maciel Filho (2000); Prasad et al. (2002)), they are primarily meant for planning and design of the processing plants, and therefore usually focus on the steady states of the process (Chéruy (1997)). Thus, such models can seldom be used in practice in a wide range of operating conditions. Moreover, often the process knowledge for modelling is not available at all. In such cases data-driven models fill the gap and often play an important role for the operation of the processes as they can extract the process knowledge automatically from the

[1] Outliers are handled in a different way than missing values

provided data. A review of data-driven soft sensors in the process industry is presented in Kadlec et al. (2009). The most popular methods are multivariate statistical techniques like Principal Component Analysis (PCA) in a combination with a regression model (PCR – Jolliffe (2002)), and Partial Least Squares (PLS – Wold et al. (2001)). Other common approaches like Multi-Layer Perceptron (MLP – Qin (1997)) and Radial Basis Function (RBF – Wang et al. (2006)) are based on neural networks. Kadlec et al. (2009) show that there are indeed dozens of methods to build soft sensors and each of them with various hyperparameters. There is however no single method that is universally superior across all the problems (Wolpert & Macready (1997)).

As seen in Section 1.3, raw data usually contain imperfections that can decrease model performance. Therefore, the building of soft sensors will require including a number of preprocessing steps as in the framework described in Section 1.4 and effectively forming an MCPS. Building the best possible soft sensor for a particular dataset can therefore be formulated as a CASH problem (see Section 3.6).

Not only it is important to choose a loss function that is possible to optimise but it is even more important that the cost measures the performance aspects that are practically relevant. The Root Mean Squared Error (RMSE) is very popular in research due to convenient analytical properties (see Equation 2.8). It punishes large deviations from the target, which is often very relevant for industrial applications. Therefore RMSE has been the performance measure chosen to be optimised by SMAC.

Average RMSE of the 25 runs for each dataset are presented in Table 5.1. Though SMAC has been the best strategy when optimising the 10-fold CV error of all datasets, it does not generalise as well as random search which performed better on average for the testing sets in 5 out of 7 datasets. This behaviour is due to overfitting on the training process as has been observed before in Thornton et al. (2013).

	10-fold CV error ϵ				**Holdout error $\mathcal{E}$**			
	RAND		**SMAC**		**RAND**		**SMAC**	
	μ	2σ	μ	2σ	μ	2σ	μ	2σ
absorber	0.4513	0.0168	**0.4341**	0.0161	**0.9361**	0.0798	0.9461	0.0762
catalyst	0.0404	0.0261	**0.0270**	0.0131	0.1105	0.0389	**0.1013**	0.0493
debutanizer	0.0588	0.0099	**0.0496**	0.0026	**0.1822**	0.0581	0.1868	0.0464
drier	1.5100	0.0467	**1.5000**	0.0389	**1.4100**	0.0194	1.4200	0.0176
oxeno	0.0050	0.0027	**0.0025**	0.0037	0.0261	0.0259	**0.0251**	0.0400
sulfur	0.0262	0.0099	**0.0247**	0.0016	**0.0434**	0.0756	0.0519	0.1094
thermalox	0.5833	0.0170	**0.5804**	0.0088	**1.1300**	0.6229	1.5600	7.6100

Table 5.1: *Average RMSE (μ) and standard deviation for 95% confidence interval (2σ). Boldfaced values indicate the lowest mean error for each dataset.*

The best MCPS configurations for each dataset (i.e. lowest holdout error) are shown in Table 5.2. Ensemble methods (Bagging and RandomSubSpace), which train multiple predictors with different subsets of data, have been found to provide the best performance for all analysed datasets. Predictions of these MCPSs are shown together with the real

target values in the right hand side of Figures A.1 to A.7 in Section A.9. The prediction plot corresponding to 'absorber' dataset is shown in Figure 5.1. Initial predictions on testing set are quite accurate, but from around $t = 1300$ the predicted values are shifted with respect to true vales. This is due to changes in data during the process. Adaptive strategies are later considered in Section 5.3. Applying an adaptive strategy can provide better predictions as shown in Figure 5.4. Though the final predictions could benefit from additional post-processing, results are very promising taking into account that they have been generated without any human interaction. In fact, the solutions found outperform the four most popular methods for building soft sensors (PCA, PLS, MLP and RBF) in 6 out of 7 datasets (see δ in Table 5.2). None of the most popular techniques have been selected among the best MCPSs, indicating the potential disadvantage of human bias towards well-known methods.

dataset	**MV**	**OU**	**TR**	**DR**	**SA**	**predictor**	**meta-predictor**	$\mathcal{E}$	δ
absorber	-	-	Wavelet	Rand.Subs.	-	KStar	Rand.SubSp.	0.8989	↑ 0.0844
catalyst	Max	-	Normalize	-	-	GP	Bagging	0.0736	↑ 0.1144
debutanizer	EM	-	Wavelet	-	-	IBk	Rand.SubSp.	0.1745	-0.0035
drier	EM	-	-	Rand.Subs.	Res.Samp.	M5P	Bagging	1.3744	↑ 0.0573
oxeno	Zero	-	Normalize	-	-	M5P	Rand.SubSp.	0.0226	↑ 0.0042
sulfur	Zero	-	Standardize	-	-	M5P	Bagging	0.0366	↑ 0.0030
thermalox	Mean	-	Wavelet	-	-	GP	Rand.SubSp.	0.6904	↑ 0.6170

Table 5.2: *Best MCPS for each dataset, holdout error $\mathcal{E}$ and difference with baseline δ (↑ indicates an improvement). MV = missing value replacement, OU = outlier detection and removal, TR = transformation, DR = dimensionality reduction, SA = sampling.*

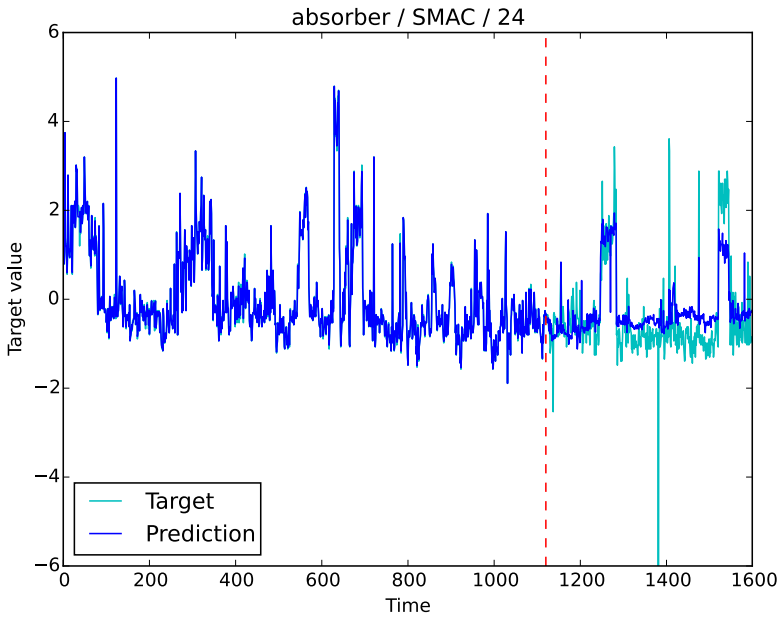


Figure 5.1: *Target value vs. prediction of best MCPS found for 'absorber' dataset. Values to the left of the vertical dashed line correspond to the training set, while the ones to the right belong to the test set.*

5.2.2 Process monitoring: a classification problem

Process monitoring is a task in which one or more variables are monitored to inspect chemical process status and recognise possible process faults. A common tool to help human operators with this task are control charts (also known as Shewhart charts, Shewhart (1931)). These charts contain a measured signal and thresholds that serve as visual aid for operators (see Figure 5.2). Sometimes it is interesting to monitor values such as product concentrations that cannot be measured by physical sensors. Instead of predicting a continuous value one could be interested in a level value such as 'high', 'normal' or 'low', or the direction of concentration change ('up' or 'down'). Thus, a classifier can be built using a labelled set of historical data.

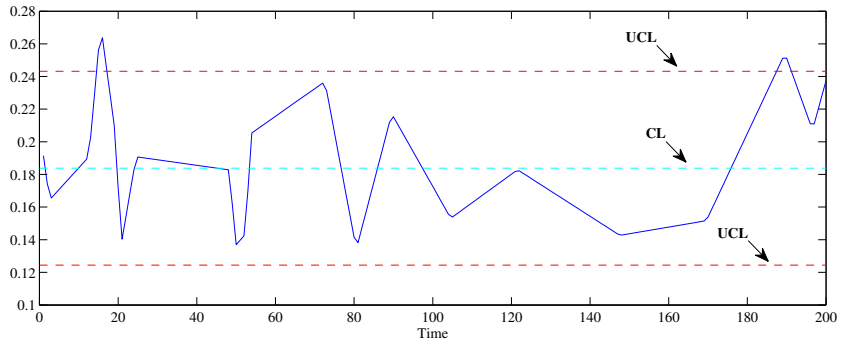


Figure 5.2: *Example of control chart showing a signal with UCL (Upper Control Limit), CL (Central Line), and LCL (Lower Control Limit).*

In order to simulate a process monitoring task, the target value of the same regression datasets previously used has been transformed into classes (each of them representing the state of the output – high, normal, low). In datasets with 2 target values 9 classes are used (combining the 3 states of both targets).

The experimentation framework is identical as the one employed for online prediction in the previous section, varying only the evaluation measure to the classification error instead. Results in Table 5.3 are the average percentage of classification error of 100,000 bootstrap samples (i.e. randomly selecting 4 out of the 25 runs and keeping the one with the lowest cross-validation error as done in Thornton et al. (2013)). Similarly to the results in previous section, SMAC has found, on average, better MCPSs than random search for all the datasets given the 10-fold CV error in $\mathcal{D}_{train}$. SMAC is also getting better results in the majority of datasets for the holdout error in $\mathcal{D}_{test}$. The best MCPS configurations found are shown in Table 5.4. Ensemble methods are present again in 3 out of 7 datasets, reinforcing the fact that combining outputs of multiple models can achieve better performance than a single model could obtain (Dietterich (2000)).

The solutions found outperform the four most popular methods for building soft sensors for process monitoring (PCA, MLP, RBF and SOM) in 4 out of 7 datasets (see δ in Table 5.4). The holdout error rates reported in Table 5.3 are however quite high and with a large deviation which might be due to overfitting, so one could think that the best

MCPSs found might not be suitable for being used in a real environment. Results show also a large difference between the CV error and the holdout test error in some of these datasets (e.g. $\epsilon = 2.60\%$ to $\mathcal{E} = 61.29\%$ in 'catalyst'). This is due to the evolving nature of some chemical processes over time. The test set (last 30% of samples) can be significantly different from the initial 70% of data used for training the MCPS. Next sections show how to adapt these MCPSs when there are changes in data and therefore make them suitable in a production environment.

	10-fold CV error ϵ				**Holdout error $\mathcal{E}$**			
	RAND		**SMAC**		**RAND**		**SMAC**	
	μ	2σ	μ	2σ	μ	2σ	μ	2σ
absorber	21.19	2.77	**19.73**	0.9838	52.48	10.21	**51.00**	9.79
catalyst	3.35	2.83	**2.60**	0.439	63.46	25.57	**61.29**	20.91
debutanizer	17.28	9.19	**10.57**	1.67	56.18	11.7	**55.18**	8.22
drier	39.58	2.11	**37.96**	2.46	45.86	7.65	**45.35**	8.47
oxeno	5.95	1.54	**4.86**	0.7635	42.43	1.63	**42.20**	10.67
sulfur	50.73	10.76	**40.30**	3.87	**75.67**	10.55	79.17	0.34
thermalox	20.52	1.82	**20.12**	1.06	**33.91**	26.95	40.38	30.73

Table 5.3: *Average classification error (%) and standard deviation for 95% confidence interval. Boldfaced values indicate the lowest mean error for each dataset.*

dataset	**MV**	**OU**	**TR**	**DR**	**SA**	**predictor**	**meta-predictor**	$\mathcal{E}$	δ
absorber	-	-	Wavelet	Rand.Subset	-	KStar	Bagging	45.63	-9.38
catalyst	-	-	Wavelet	Rand.Subset	-	JRip	Filt.Classifier	48.64	-16.2
debutanizer	Median	-	Wavelet	-	-	REPTree	Filt.Classifier	50.14	$\uparrow$ 2.09
drier	-	IQR	Normalize	-	-	Logistic	Bagging	40.98	$\uparrow$ 10.11
oxeno	-	-	Standardize	-	-	JRip	Rand.SubSpace	38.24	-4.74
sulfur	-	-	Wavelet	Rand.Subset	-	JRip	Filt.Classifier	70.96	$\uparrow$ 8.7
thermalox	-	-	Wavelet	-	-	MLP	Filt.Classifier	26.71	$\uparrow$ 20.81

Table 5.4: *Best MCPS for each dataset in NEW and FULL spaces, holdout error $\mathcal{E}$ and difference with baseline δ ($\uparrow$ indicates an improvement). MV = missing value replacement, OU = outlier detection and removal, TR = transformation, DR = dimensionality reduction, SA = sampling.*

5.3 Adapting MCPS in continuous processes

To illustrate how MCPSs can be adapted in a realistic production environment, several adaptation strategies have been implemented. Two main approaches utilising the automated MCPSs composition and optimisation approach proposed in this thesis have been compared in this section: global re-composition and global parameterisation, varying also the forgetting mechanism. Results from extensive experiments are presented and discussed below.

The same datasets from previous section have been used for experimentation. In process industry it is common to have continuous processes but which data arrive in batches. To simulate such processes, $\mathcal{D}_{test}$ has been split into 10 batches of approximately equal size. Sizes of initial training set and each batch for each dataset are listed in Table 5.5.

			Instances		
	Attributes	**Classes**	**Total**	**Initial training**	**Batch**
absorber	38	3	1599	1119	48
catalyst	14	3	5867	4109	176
debutanizer	7	3	2394	1676	72
drier	19	3	1219	853	37
oxeno	71	3	17588	12311	528
sulfur	5	9	10081	7057	303
thermalox	38	9	2820	1974	58

Table 5.5: *Datasets properties*

Four different global adaptation strategies have been selected for a comparison within the same evaluation framework. All these approaches assume that an initial MCPS has been composed and optimised using SMAC integrated in the Auto-WEKA extension presented in this thesis during 30 CPU core-hours for the initial training set. This MCPS is then used to predict the target value of a batch of incoming instances from the test set. After that, the true labels are provided and one of the following strategies is executed (for summary of the strategies please refer to Table 5.6):

- **Baseline**, where the initial MCPS is not adapted at all and continues making predictions for the consecutive batches.
- **Batch**, where a new MCPS is trained using only the labelled data from the most recent batch and the configuration (including hyperparameters) from the initial MCPS. This strategy learns patterns from the most recent data and forgets the old ones.
- **Batch + SMAC**, where a new MCPS is composed using the most recent batch as training set and SMAC as optimisation strategy for 5 CPU core-hours[2] (see Figure 5.3 where a sequence diagram of this strategy is presented). Although the old patterns are being forgotten, the historical information of the underlying SMAC model remains. That is, the random forest built by SMAC (learnt during the previous training runs and utilising their classification errors) is preserved, so the exploration will not start from scratch.
- **Cumulative**, where a new MCPS is trained using all the available labelled instances including the most recent batch, but the configuration of the initial MCPS is preserved. Therefore, there is no forgetting of concepts.

[2]The reason behind choosing this value is two fold: i) it has been observed that the majority of the runs converge after that time for these datasets; ii) from the practical point of view it simulates a plant maintenance stop.

- **Cumulative + SMAC** uses the same training strategy as Cumulative, but similarly to Batch+SMAC strategy a new MCPS is composed using SMAC after every batch for 5 CPU core-hours. The historical information of the underlying SMAC model remains.

Each experiment has been repeated 25 times with different random initialisations of SMAC (i.e. seeds), but keeping the same data partitions.

Strategy	Data for Training	Forgetting	Parametric Adaptation	MCPS Optimisation
Baseline	No			
Batch	Batch	✓	✓	
B + SMAC	Batch	✓	✓	✓
Cumulative	Cumulative		✓	
C + SMAC	Cumulative		✓	✓

Table 5.6: *Evaluated strategies*

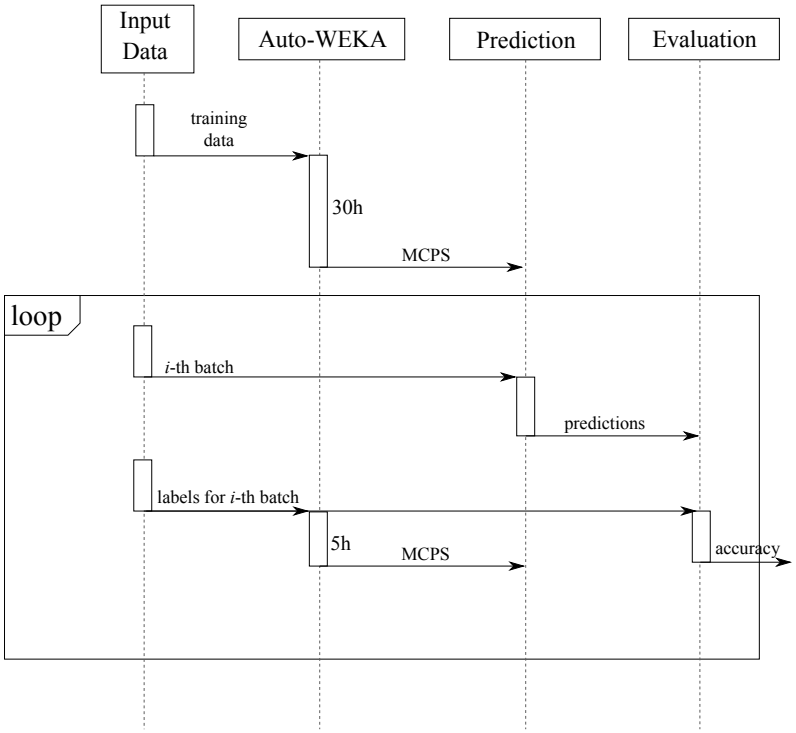


Figure 5.3: *Sequence diagram of Batch+SMAC strategy. Firstly, Auto-WEKA is run during 30 CPU-hours to find the best MCPS for the training data. Then, such MCPS is used to predict the first batch of unlabelled data. When the labels for that batch arrive, the predictive accuracy is measured and the MCPS is adapted running Auto-WEKA during 5 CPU-hours with the labelled batch. For every new batch of data, the process within the loop is repeated.*

	Base.	Batch		B+SMAC		Cumulative		C+SMAC	
	$\mathcal{E}$	$\mathcal{E}$	δ	$\mathcal{E}$	δ	$\mathcal{E}$	δ	$\mathcal{E}$	δ
absorber	0.9461	0.9754	-0.0292	1.4310	-0.4849	0.8962	↑ 0.0499	**0.8946**	↑ 0.0515
catalyst	0.1014	**0.0308**	↑ 0.0706	0.0324	↑ 0.0690	0.0615	↑ 0.0399	0.0785	↑ 0.0230
debutanizer	0.1868	**0.1533**	↑ 0.0334	0.1669	↑ 0.0199	0.1680	↑ 0.0188	0.1832	↑ 0.0036
drier	1.4238	1.4500	-0.0262	1.5812	-0.1574	1.4107	↑ 0.0131	**1.3419**	↑ 0.0819
oxeno	0.0196	0.0215	-0.0019	0.0231	-0.0035	0.0203	-0.0007	**0.0180**	↑ 0.0016
sulfur	0.0519	0.0557	-0.0038	0.0583	-0.0064	**0.0447**	↑ 0.0072	0.0481	↑ 0.0038
thermalox	1.5637	4.1272	-2.5635	27.1927	-25.6290	**1.0500**	↑ 0.5137	3.5019	-1.9382
avg. rank	3.29	3.14		4.14		2.14		2.29	

Table 5.7: *Average RMSE ($\mathcal{E}$) and difference with baseline (δ, ↑ indicates an improvement) for each dataset and adaptation strategy. Best result of each dataset is in bold.*

5.3.1 Regression results

Average RMSE for the regression task are presented in Table 5.7. Predictions of the best MCPS found for each dataset are shown together with real target values in Figures A.8 to A.14. Points before the vertical dashed line belong to the initial training set, while the ones after that belong to the testing set which is split in 10 batches of approximately equal size.

As one would expect, the strategies performing cumulative training are, on average, the ones performing better than baseline in 6 out of 7 datasets ($\delta > 0$) since they have more data to learn from. See for example the predicted values for 'absorber' dataset in Figure 5.4, which are much closer to the target values than the ones found in the static experiments (see Figure 5.1 to contrast). Nonetheless, the Batch strategy performed the best on average for 'catalyst' and 'debutanizer' datasets. These two datasets are the ones which present more variability across batches as can be seen in Figures A.9 and A.10. A comparison with results from baseline (see Figures A.2 and A.3, respectively) shows that the Batch strategy is able to cope better in some batches.

On the other hand, in the case of 'drier' and 'oxeno' datasets the best MCPSs found are from baseline strategy (i.e. no adaptation – see Figures A.11 and A.12, respectively), despite Cumulative+SMAC strategy performed better in average (see Table 5.7). This is an understandable result for 'drier' since data distribution is quite stable. However, it is a surprising result for 'oxeno' dataset since one would expect an adaptive strategy to perform better when data shifts.

The results for 'thermalox' dataset are very interesting. The Cumulative strategy presents a remarkable result improving baseline (see Figure A.14 in contrast with A.7, and average result shown in Table 5.7). However, the remaining strategies have got very poor results. By taking a closer look to those particular results, one can see that there are a couple of predictions in batches 5 and 10 which are large outliers and therefore penalise the RMSE. Adding an additional post-processing step to mitigate such extreme outliers could be something to consider when coming to implementation.

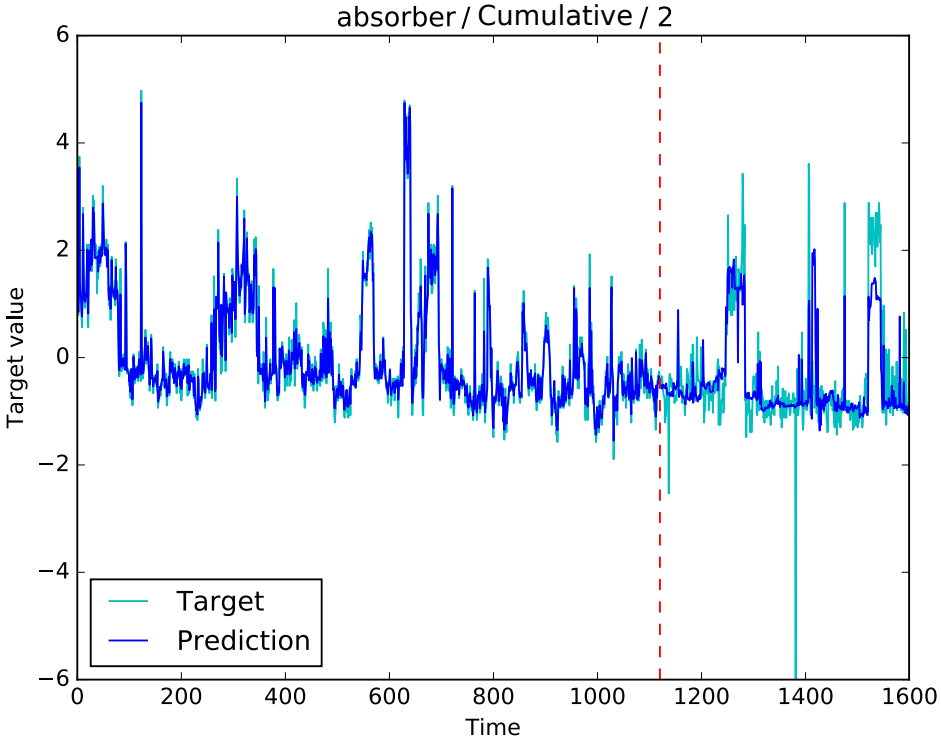


Figure 5.4: *Target value and prediction of best MCPS found for 'absorber'. Values to the left of the vertical dashed line correspond to the training set, while the ones to the right belong to the test set.*

5.3.2 Classification results

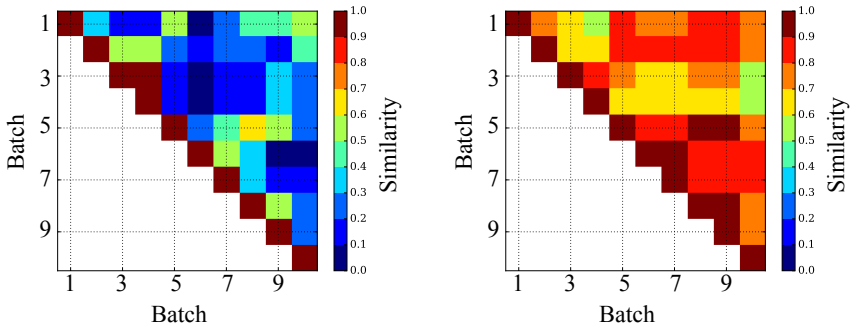
In the case of the classification task, average classification error for each dataset and strategy are shown in Table 5.8. The Batch adaptation strategy performed better than the baseline in 6 out of 7 datasets (i.e. lower mean error). The only dataset in which the predictive accuracy has worsened is 'drier'. This dataset does not change much in terms of any predictable trends and the performance improves with adding and using more data for training the predictor. Therefore one of the main causes of such deterioration is the small batch size used for training (only 37 samples per batch) in comparison to the size of the training data for the baseline method. On average, the use of SMAC with Batch strategy has resulted in better performances in 4 out of 7 datasets compared with only Batch. That means that over-optimising an MCPS may not always be the best approach when there is a risk of over-fitting due to a drastic forgetting mechanism employed. Cumulative strategy has improved the predictions for all the datasets. These results were expected since there is no forgetting of previous samples. In addition, applying SMAC optimisation to Cumulative strategy has helped to refine MCPSs and has improved the average results of standalone Cumulative strategy in 5 out of 7 datasets.

	Base.	**Batch**		**B+SMAC**		**Cumulative**		**C+SMAC**	
	$\mathcal{E}$	$\mathcal{E}$	δ	$\mathcal{E}$	δ	$\mathcal{E}$	δ	$\mathcal{E}$	δ
absorber	54.32	40.13	↑ 14.19	43.43	↑ 10.88	33.37	↑ 20.95	**33.13**	↑ 21.18
catalyst	68.34	25.09	↑ 43.24	**25.06**	↑ 43.27	37.93	↑ 30.41	38.08	↑ 30.25
debutanizer	58.88	**47.54**	↑ 11.34	48.73	↑ 10.15	53.35	↑ 5.53	52.77	↑ 6.11
drier	49.89	55.54	-5.65	54.18	-4.29	**48.12**	↑ 1.77	49.56	↑ 0.33
oxeno	45.92	40.60	↑ 5.33	**38.08**	↑ 7.84	39.44	↑ 6.48	38.70	↑ 7.22
sulfur	80.67	79.91	↑ 0.76	80.19	↑ 0.48	79.70	↑ 0.97	**78.92**	↑ 1.75
thermalox	55.07	39.42	↑ 15.65	35.83	↑ 19.24	39.95	↑ 15.12	**33.25**	↑ 21.81
avg. rank	4.71	3.00		2.29		2.71		2.00	

Table 5.8: *Average % classification error ($\mathcal{E}$) and difference with baseline (δ, ↑ indicates an improvement) for each dataset and adaptation strategy. Best result of each dataset is in bold.*

5.3.3 Evolution of MCPS over batches

Analysing the evolution of MCPSs found after applying SMAC optimisation between batches can help to identify how robust they are to changes in data. To calculate the similarity between MCPSs a weighted sum of Hamming distances described in Section 4.3.2 has been used. As an example, Figure 5.5 shows two triangular matrices representing the MCPS similarity between batches for 'catalyst' dataset in a) Batch+SMAC and b) Cumulative+SMAC strategies for the classification task. One can observe how MCPSs between batches 3 and 4 are very similar but then there are considerable changes between the others. The large differences between the MCPSs in a) are due to the extreme forgetting mechanism that is not present in b), where the MCPSs are more stable due to the accumulation of historical data as no forgetting is used. Table 5.9 shows the evolution of the MCPS configuration between batches for the same case as Figure 5.5-b. The meta-predictor is the same for all batches (AdaBoost), varying only its hyperparameters. The logistic model tree (LMT) classifier has been selected in 8 out of 10 batches, while a multilayer perceptron (MLP) was chosen for batches 3 and 4. The transformation component is the one with more variation. Finally, the method for replacing missing values has slightly varied across the batches.



a) Batch+SMAC *b) Cumulative+SMAC*

Figure 5.5: *MCPS similarity between batches for 'catalyst' dataset (seed=17). Brown and red colours indicate very high similarity – e.g. between batches 6 and 7 in (b). On the other hand, blue and dark blue colours represent very low similarity – e.g. between batches 4 and 5 in (a). Thus, as seen in (b), MCPS configuration between batches is more stable when training data is accumulated instead of being suddenly forgotten.*

#	missing values	transformation	predictor	meta-predictor
1	EMImputation -E 346.813 -N 467 -Q 799.631	Standardize	LMT -R -C -P -M 1 -W 0.029 -A	AdaBoostM1 -P 100 -I 60 -Q -S 1
2	ReplaceMissing -M 2	Normalize -S 1.0 -T 0.0	LMT -M 15 -W 0	AdaBoostM1 -P 100 -I 35 -Q -S 1
3	ReplaceMissing -M 2	Standardize	MLP -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a	AdaBoostM1 -P 100 -I 85 -S 1
4	ReplaceMissing -M 2	Center	MLP -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a	AdaBoostM1 -P 100 -I 81 -S 1
5	ReplaceMissing -M 2	Standardize	LMT -M 15 -W 0	AdaBoostM1 -P 100 -I 28 -S 1
6	ReplaceMissing -M 2	Wavelet	LMT -M 15 -W 0	AdaBoostM1 -P 100 -I 29 -S 1
7	ReplaceMissing -M 2	Wavelet	LMT -M 15 -W 0	AdaBoostM1 -P 100 -I 10 -S 1
8	ReplaceMissing -M 5	Standardize	LMT -P -M 13 -W 0	AdaBoostM1 -P 100 -I 17 -S 1
9	ReplaceMissing -M 1	Standardize	LMT -R -C -M 19 -W 0 -A	AdaBoostM1 -P 95 -I 2 -S 1
10	-	Wavelet	LMT -M 15 -W 0	AdaBoostM1 -P 100 -I 10 -S 1

Table 5.9: *Components found for 'catalyst' dataset for each batch in C+SMAC strategy (seed=17). No outlier handling, dimensionality reduction and sampling components were selected by SMAC for this particular case.*

5.4 Conclusion

This chapter has discussed the feasibility of applying search strategies for automatically building soft sensors with the purposes of online prediction and process monitoring of chemical production processes. The results of an intensive experimentation has shown that it is indeed feasible, and even solutions found can achieve better performance than most popular soft sensor techniques. This is a very significant step towards reducing the cost and time needed for developing soft sensors.

The proposed automatic approach is fully data-driven, not requiring any human intervention. The developed tool allows practitioners to create an MCPS by just providing a dataset. The visual representation of Petri nets can help to understand how data is processed. Nevertheless, there is a risk of ending up with complex models that might not be transparent enough for domain experts. To overcome such situation, one could restrict the search space to only those models that are known to be easier to interpret. A direction of further investigation would be to optimise towards multiple objectives (e.g. high accuracy and low complexity) creating a Pareto of optimal solutions.

Moreover, there is a need of adapting soft sensors to cope with changes in a real environment. There are multiple adaptive models that can be used to that purpose. However, there are situations when the preprocessing methods need also to be adapted. To this end, several hybrid adaptation strategies including global re-composition and global parameterisation, varying also the forgetting mechanism have been proposed and evaluated in a number of datasets. Though the results indicate that adaptation brings benefit to model performance, there is not a single strategy that performs the best in all the scenarios. Instead, it could be interesting to combine several adaptive mechanisms as the approach presented by Bakirov et al. (2017) which has shown to improve the predictive performance.

Another challenging problem is the partial adaptation of an MCPS (e.g. adapting or replacing one or more components). A preliminary study of this subject is presented in Appendix C, where the need for appropriate propagation of changes when performing local adaptation of components is discussed.

The fact that adaptation of the full predictive system can be carried out in an automatic and autonomous way is very beneficial from the practical point of view. The lack of human intervention will reduce operational costs and potentially keep processes running for longer periods while maintaining a good predictive performance.

Chapter 6

Conclusion and future work

6.1 Thesis summary

The main problem addressed in this thesis has been the automatic configuration and optimisation of multicomponent predictive systems. In particular, how this can be applied to speed up the initial building and maintenance of soft sensors in the process industry.

In order to provide a clear motivation for the conducted research the initial chapters of this thesis have reviewed what are the main challenges when applying data mining techniques to raw data from chemical production processes. The development of a predictive system is an iterative process that has been presented in Chapter 2. The output of this process is a workflow connecting a number of preprocessing methods and predictors. Chapter 3 has formalised this workflow using a novel definition based on the Petri net theoretical framework.

Building a predictive system is known to be a tedious process, especially when the data quality is low. Appendix B presents a case study where a predictive system is manually composed including several preprocessing steps. One of these steps is the identification of shutdown periods, that is, when a chemical plant is stopped for maintenance purposes. A novel method for online identification of shutdown periods is proposed. A comparison with state-of-the-art methods shows an improvement on the predictive accuracy. This case study shows as well how tedious and time consuming is the process of composing and optimising an MCPS.

Finding the best combination of data cleaning, data transformation and predictive methods is often impractical due to the large number of methods and associated hyperparameters available in data mining toolboxes. Thus, Chapter 4 has proposed the use of Bayesian optimisation techniques to explore and exploit the search space made of all possible preprocessing methods, predictors and ensemble methods of WEKA. An intensive experimental study has conducted and evaluated different optimisation techniques varying the search space size for 21 datasets of benchmark classification problems. One of the novel contributions of this thesis is the addition of preprocessing steps into the search space. This allows a custom data preparation for each particular model instead

of using the same preprocessing methods for all evaluated models as it is often found in the literature. The results indicate that model-based optimisation techniques perform better than a random search in the majority of the evaluated datasets even in this extended search space. The developed tool has been made freely available so other researchers and data practitioners can benefit from automatic MCPS development.

To assess the feasibility of this automatic approach in process industry, an experimental study has been carried out with datasets from real chemical processes. The best MCPSs found for 6 out of 7 datasets outperform the results of most popular soft sensor methods in online prediction (regression). In addition, best MCPSs found for 4 out of 7 datasets work better than common soft sensor methods for process monitoring (classification). These results indicate that this technique can be used to build soft sensors in an automatic fashion, but one should be aware of the possibility of getting stuck in local minima.

Moreover, a hybrid adaptation strategy has been proposed for dealing with changes in data due to the evolving nature of chemical processes and external causes. A set of experiments have compared different adaptation strategies for regression and classification tasks. It has been found that MCPS adaptation has led to reduce the predictive error for all the datasets. Nonetheless, there is no single adaptation strategy that outperform the rest. Instead one could run them in parallel and pick the one that performs best, similarly to the work done in Bakirov et al. (2017).

While MCPSs can be adapted as a whole as shown in Chapter 5, Appendix C concentrates on investigating and discussing the effects that an adaptation a single component can have on the overall predictive system behaviour, complexity and performance. The concept of change propagation in MCPSs is also introduced together with a novel definition of coloured MCPSs. This definition allows MCPS to include meta-data as an additional token and therefore propagate a local adaptation in a component to the rest of the system.

Automatic and autonomous adaptation of MCPSs is very beneficial in a production environment. Reducing the amount of human intervention implies saving of costs and potentially allows the process to run for longer periods while maintaining a good predictive performance.

6.2 Main findings and conclusions

The aim of this thesis was to study the feasibility of automating the composition and optimisation of workflows to make accurate predictions on unseen data. The proposed approach using SMBO techniques has been shown to be effective in finding MCPSs to address classification and regression tasks. To achieve that aim, the three main objectives pursued in this thesis have been successfully fulfilled:

"Propose, design and evaluate a framework for connecting multiple components to compose valid predictive workflows including preprocessing methods, machine learning models, and postprocessing operations."

Chapters 1 and 2 have presented the need of data preprocessing and modelling to build effective prediction systems. In order to formalise this practical concept, a new type of Petri net has been proposed in Chapter 3 to connect multiple components forming a predictive system. The benefit of this approach is the abstraction of individual methods and considering them instead as data transformations known in Petri nets formalism as transitions with inputs and outputs. The visual representation of MCPSs as Petri nets also helps to understand how data flows across the system. This new formalism has been integrated into the CASH problem – which now states the problem of composing and optimising MCPSs instead of a single algorithm as it was originally defined. An example of composing and optimising an MCPS for a chemical production process needing a considerable amount of data preprocessing has been presented in Appendix B. That particular example focused on the online cleaning of data from shutdown periods of a chemical plant. The amount of time and effort needed to build MCPSs is one of the main motivations leading to automate this process.

"Develop and evaluate a smart data-driven mechanism to automate the creation of such workflows with minimum human intervention."

After conducting a thorough literature review, it has been found that recent works using the Bayesian optimisation framework have been particularly attractive for tackling the main goals of the project related to the automatic predictive algorithm selection problem (Hutter et al. (2011); Bergstra et al. (2011)). In particular, SMBO methods like SMAC and TPE have become the start-of-the-art in this field and therefore they have been chosen as the starting point for the presented research. Both methods were implemented in the Auto-WEKA tool which has been substantially extended in this thesis to support search spaces made of arbitrary number of sequential data transformation steps forming workflows or, in other words, the MCPSs configurations. This has been a critical step on the way towards automatic generation and optimisation of MCPSs for classification and regression problems without any human interaction and fully driven by the available data. An intensive experimentation – presented in Chapters 4 and 5 – on 28 datasets has taken more than 162,750 CPU-core hours[1].

"Develop and evaluate an approach of adapting predictive workflows in changing environments."

A hybrid adaptation approach consisting on combining global parameterisation and global re-composition of MCPSs has been proposed in Chapter 5 and evaluated on a number of challenging, real life datasets from the process industry. Results from further extensive experimental analysis on these datasets have shown that the proposed approach can be used not only for automating a very challenging process of combining data cleansing and preprocessing together with a predictive model building, but also to streamline and automate the labour intensive maintenance of predictive systems in production environments. An initial study on local adaptation of MCPSs has been carried out in Appendix C, demonstrating the challenges of this strategy. A novel extension of MCPSs called coloured MCPSs which propagates a token of meta-data through the system has been shown to be helpful to adapt correctly the system.

[1]Running the experiments has been possible thanks to the computer cluster at Bournemouth University

The speed up of preprocessing and modelling steps is one of the main benefits resulting from the work of this thesis. Thus, the concept of 'Agile Data Science' (Jurney (2013)) is more feasible since it becomes faster to iterate over different ways of solving a changing data mining problem.

6.3 Future work

Despite good results have been achieved, this topic is still a very active field of research and there are many possibilities for new approaches and improvements. The following list provides a brief discussion of further research topics that can be done as continuation of this thesis:

- **Different data partitioning**: Using different data partitioning of the training set like Density-Preserving Sampling (DPS, Budka & Gabrys (2010a)) instead of CV could make a difference in the optimisation process of SMAC. SMAC discards potential poor solutions early in the optimisation process based on performance on only a few CV folds. In case the folds used are not representative of the overall data distribution, which as shown in Budka & Gabrys (2013) can happen quite often with CV, the effect on the solutions found can be detrimental.
- **Multi-objective optimisation**: Current SMBO methods only support single objective optimisation. However, it would be useful to find solutions that optimise more than one objective, including for instance a combination of prediction error, model complexity and running time as discussed in Al-Jubouri & Gabrys (2014).
- **Transparency and interpretability**: The optimisation process and the resultant MCPSs can be difficult to understand for domain experts. An option to penalise complex MCPS configurations could guide the optimisation strategy to find more interpretable solutions.
- **Better search strategies**: As a result of the experiments presented and discussed in this thesis, and as a recent work by Li et al. (2016) suggests, random search is not performing much worse than smarter methods such as SMAC and TPE in large search spaces. Further investigation is needed to find better search strategies that can perform well in such situations.
- **Dynamic search spaces**: In order to reduce the search time, a practical solution is to constrain the size of the search space. This however has the drawback of potentially limiting the finding of optimal solutions. One interesting line of research could be the dynamic modification of the search space by pruning and/or extension. This is especially interesting in adaptive scenarios where new data quality issues may significantly affect the existing MCPS's performance and additional preprocessing would be needed resulting in a dramatically different MCPS configuration.
- **Effective partial adaptation**: Global MCPS adaptation is a costly process (Žliobaitė et al. (2015)) that could be simplified by adapting only some parts of the system. Though theoretical work like the one in Appendix C points out the benefits of partial adaptation, there is still further research and development needed in this

topic.

- **Meta-optimisation**: Optimisation strategies require also setting hyperparameters such as time limit, exploration/exploitation rate, and starting point that influence on the quality of the solutions found. For instance, Andrychowicz et al. (2016) present an approach to optimise gradient descent on neural networks by casting the optimisation problem as a learning problem. A similar approach could be interesting to optimise a Bayesian optimisation problem.

Appendix A

Datasets from chemical processes

This appendix provides more details of the datasets used for experiments in Chapter 5 and Appendix B. Most of them have been kindly provided by Evonik Industries and consist of real measurements from Evonik's chemical processes. Due to confidentiality reasons, the datasets have been anonymised and no further details than those provided in the following description can be given. The remaining datasets are publicly available and also belong to real processes. Table A.1 lists the size of datasets and related publications where they have been used.

Dataset	Instances	Attributes	References
acrylic	1,097,281	82	Martin Salvador et al. (2014)
absorber	1,599	38	Martin Salvador et al. (2016a,b)
catalyst	5,867	14	Strackeljan (2006); Ruta & Gabrys (2010); Kadlec & Gabrys (2011); Souza & Araujo (2014); Martin Salvador et al. (2016a,b); Bakirov et al. (2017)
debutanizer	2,394	7	Fortuna et al. (2005, 2007); Martin Salvador et al. (2016a,b)
drier	1,219	19	Kadlec & Gabrys (2009); Martin Salvador et al. (2016a,b); Bakirov et al. (2017)
oxeno	17,588	71	Budka et al. (2014); Martin Salvador et al. (2016a,b)
sulfur	10,081	5	Fortuna et al. (2003, 2007); Martin Salvador et al. (2016a,b)
thermalox	2,820	38	Kadlec & Gabrys (2009); Martin Salvador et al. (2016a,b); Bakirov et al. (2017)

Table A.1: *Chemical datasets*

A.1 Acrylic Acid Dataset

This dataset was provided by Evonik Industries. The data was extracted from the database of the acrylic acid production plant. This dataset was collected with the purpose of building a soft sensor to predict the concentration of acrylic acid in the final product. It contains 1,097,281 instances with only 3,335 of them having the target value which has been measured in the laboratory. The 82 attributes of this dataset are mostly physical measurements like flows, temperatures or pressures, although other derived variables are also included.

A.2 Absorption Process Dataset (absorber)

This dataset was provided by Evonik Industries. It is formed by 38 numerical attributes, including the target value. It has 1,198 instances for training and 401 instances for validation. No additional information has been provided apart from this being a regression task.

A.3 Catalyst Activation Dataset (catalyst)

This dataset was made available for the NiSIS 2006 competition (Strackeljan (2006)). The task is to predict the activity of a catalyst in a multi-tube reactor. The dataset has 5,867 timestamped instances with 14 sensor measurements including flows, concentrations and temperatures, and one target variable. It covers one year of operation of the process plant. This dataset is known to need strong adaptive mechanisms since the process evolves quite drastically over time. The data also shows high co-linearity between features and high amount of outliers which can be found in almost 80% of the features.

A.4 Debutanizer Column Dataset (debutanizer)

This dataset is publicly available and described in Fortuna et al. (2007). It contains 2,394 instances formed by four temperatures, one pressure and two flows of a debutanizer column. This column is a part of the desulfuring and naphtha splitter plant. The target value is the concentration of butane at the output of the column.

A.5 Drier Process Dataset (drier)

This dataset was provided by Evonik Industries. It has 19 input features, most of them being physical properties from the chemical plant. The target value is the humidity of the process product which is measured in the laboratory. The dataset has 1,219 timestamped instances covering almost 7 months of process operation. It consists of raw unprocessed

data recorded by the process information and measurement system. The main issue of this dataset is the high noise level which affects all of the data features as well as outliers which an also be found in most of the features. Missing values are present in 16% of the features.

A.6 Oxeno Dataset (oxeno)

This dataset was provided by Evonik Industries. It contains 211,051 instances formed by 71 attributes, however for practical purposes it has been subsampled by periodically selecting one sample every hour, resulting in 17,588 instances. The attributes include temperatures, pressures, flows and concentrations from different sensors of the plant. The target variable is the product concentration which is measured in the laboratory. Data samples were collected from a historical database which uses lossy compression.

A.7 Sulfur Recovery Unit Dataset (sulfur)

This dataset is also publicly available and described in Fortuna et al. (2007). The sulfur recovery unit is a system for removing environmental pollutants from acid gas streams before they are released into the atmosphere. The washed out gases are transformed into sulfur. The dataset contains 10,081 instances and has five input features (i.e. gas and air flow measurements) and two target values which are the concentration of H_2S and SO_2. In the regression experiments, only SO_2 is considered, while in the classification experiments both target values are grouped into a single categorical variable.

A.8 Thermal Oxidiser Dataset (thermalox)

This dataset was provided by Evonik Industries. It has 2,820 timestamped instances of raw data with 38 attributes belonging to sensor measurements. The input features are physical values like concentrations, flows, pressures and temperatures measured during the operation of the plant. Many of the variables present common issues of industrial data like measurement noise and data outliers which severely affect approximately half of the features. The task is to predict the concentrations of NO_x and SO_x in the exhaust gases, which are measured in the laboratory. Only SO_x is considered for the regression experiments. Both target values are grouped into a single categorical variable in the classification experiments.

A.9 Results of online prediction

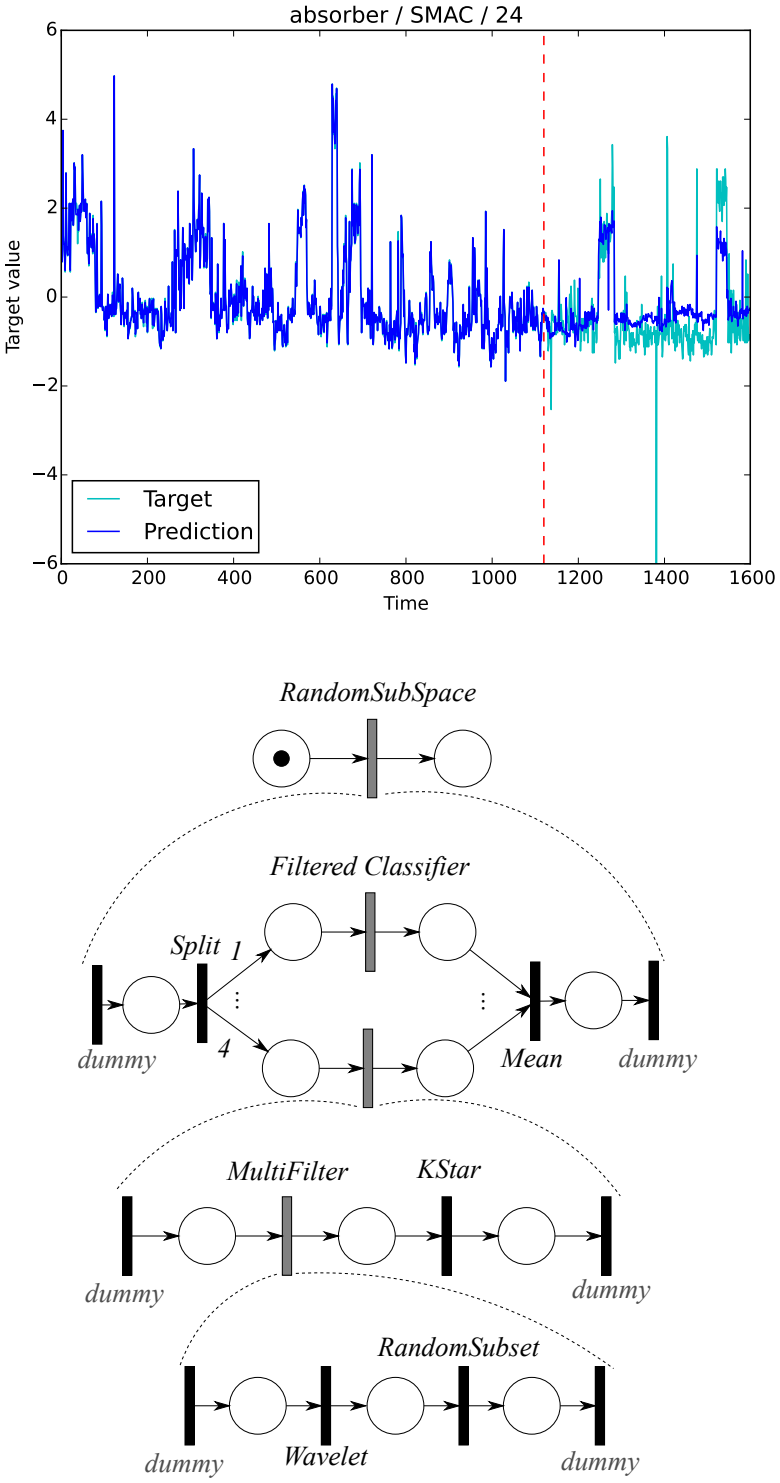


Figure A.1: *Configuration of best MCPS for 'absorber' (bottom) and associated target value vs. prediction (top)*

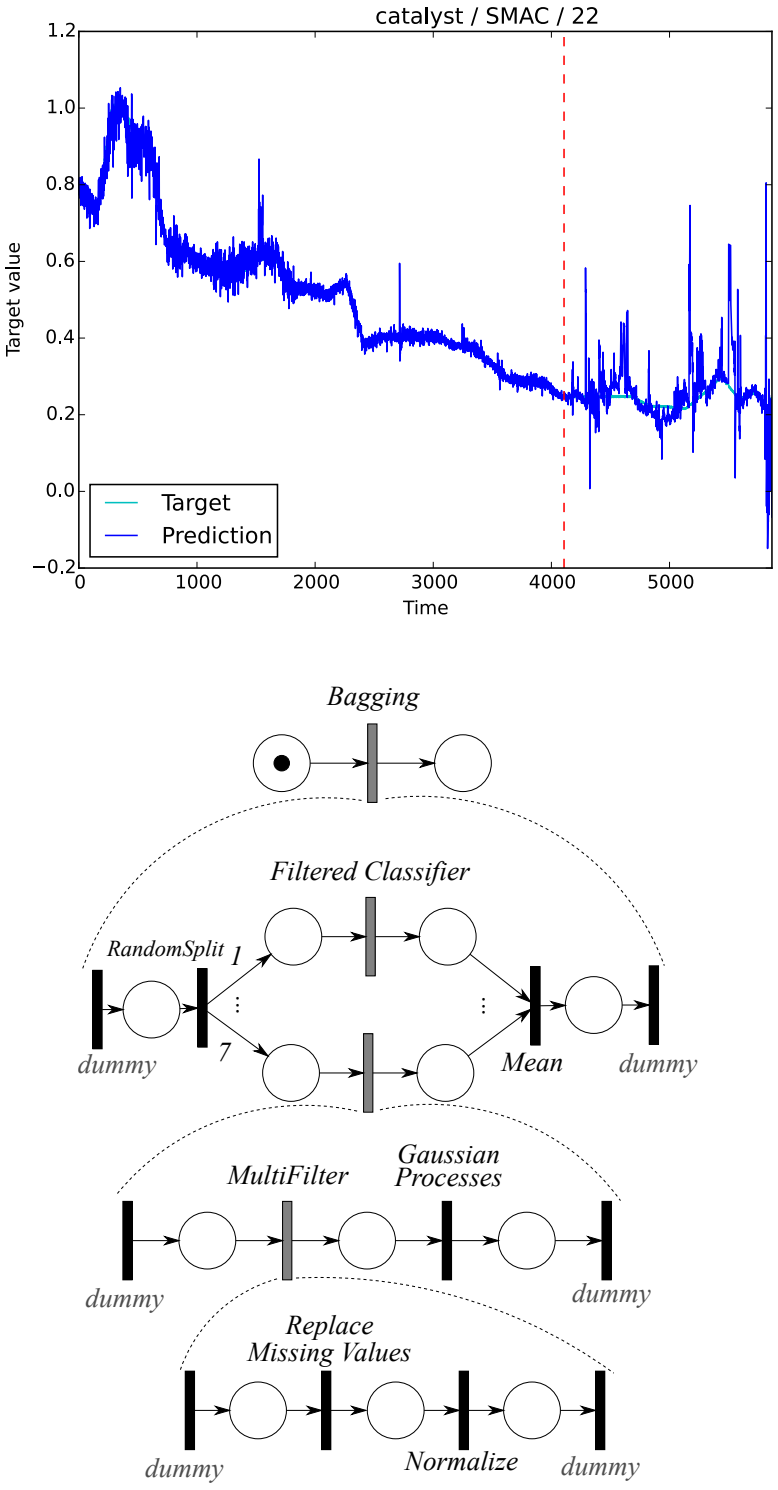


Figure A.2: *Configuration of best MCPS for 'catalyst' (bottom) and associated target value vs. prediction (top)*

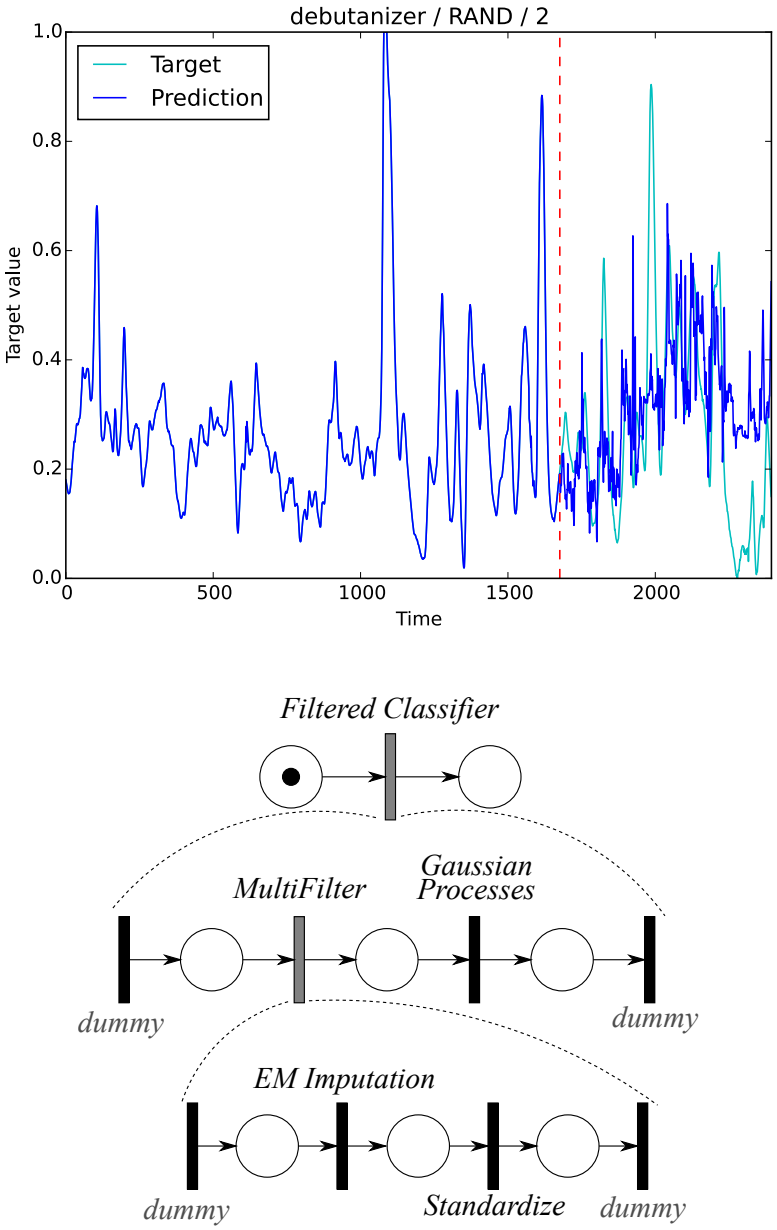
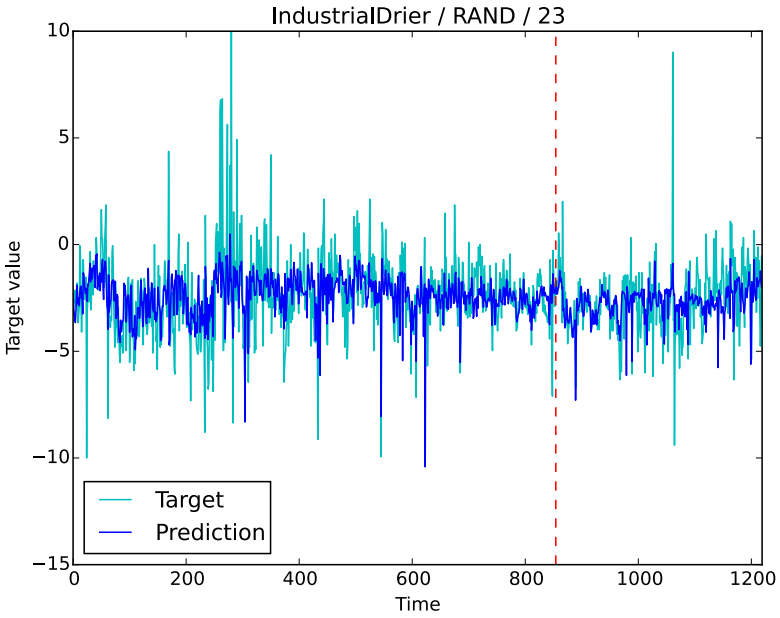


Figure A.3: *Configuration of best MCPS for 'debutanizer' (bottom) and associated target value vs. prediction (top)*



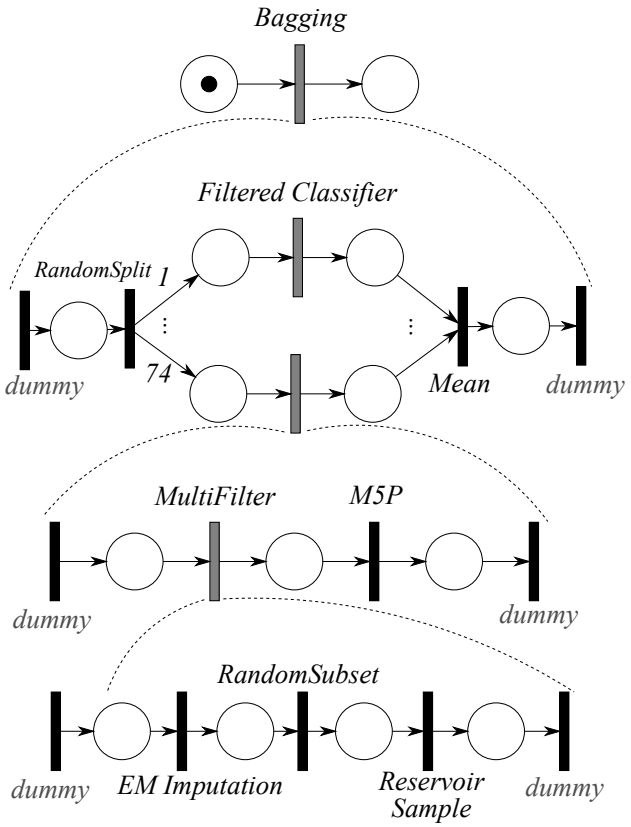


Figure A.4: *Configuration of best MCPS for 'drier' (bottom) and associated target value vs. prediction (top)*

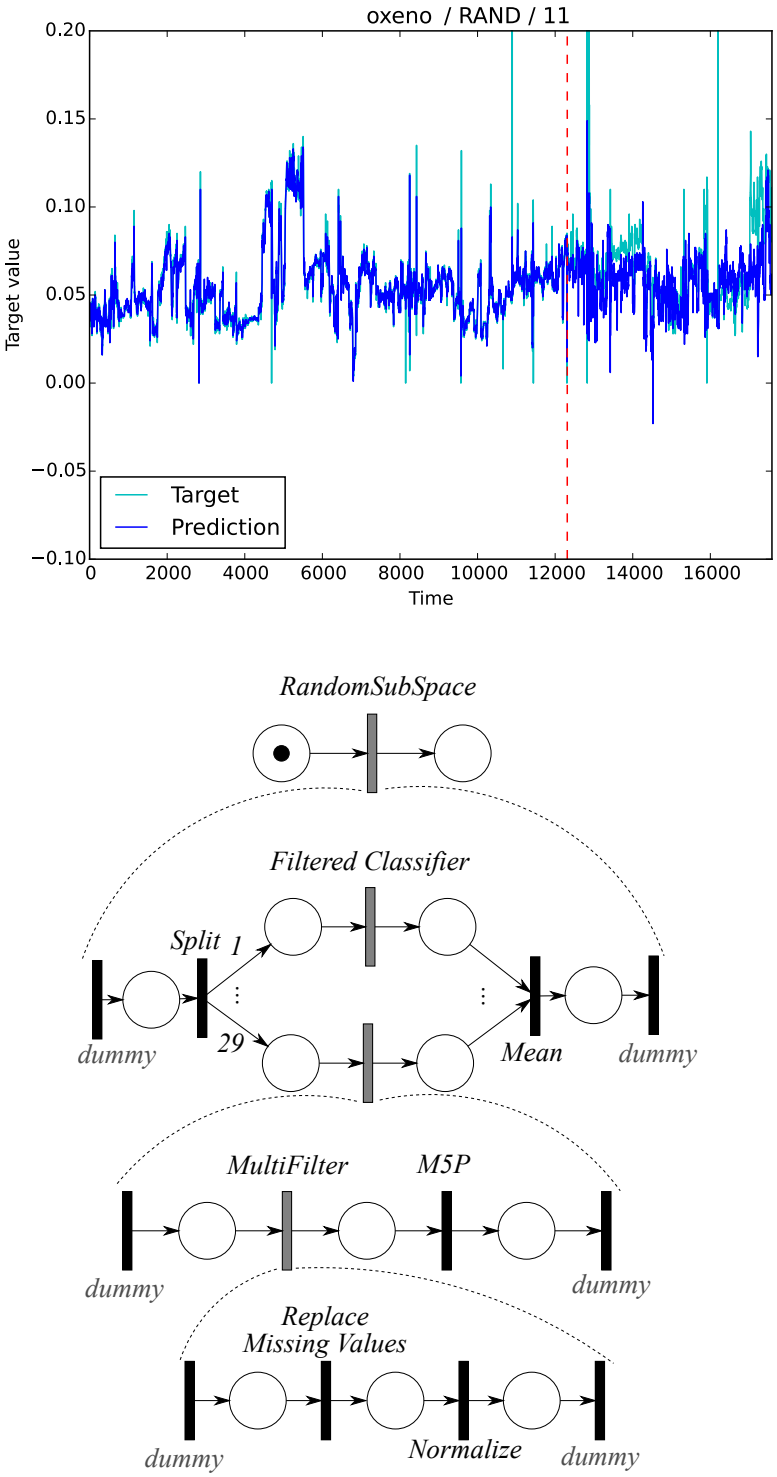


Figure A.5: *Configuration of best MCPS for 'oxeno' (bottom) and associated target value vs. prediction (top)*

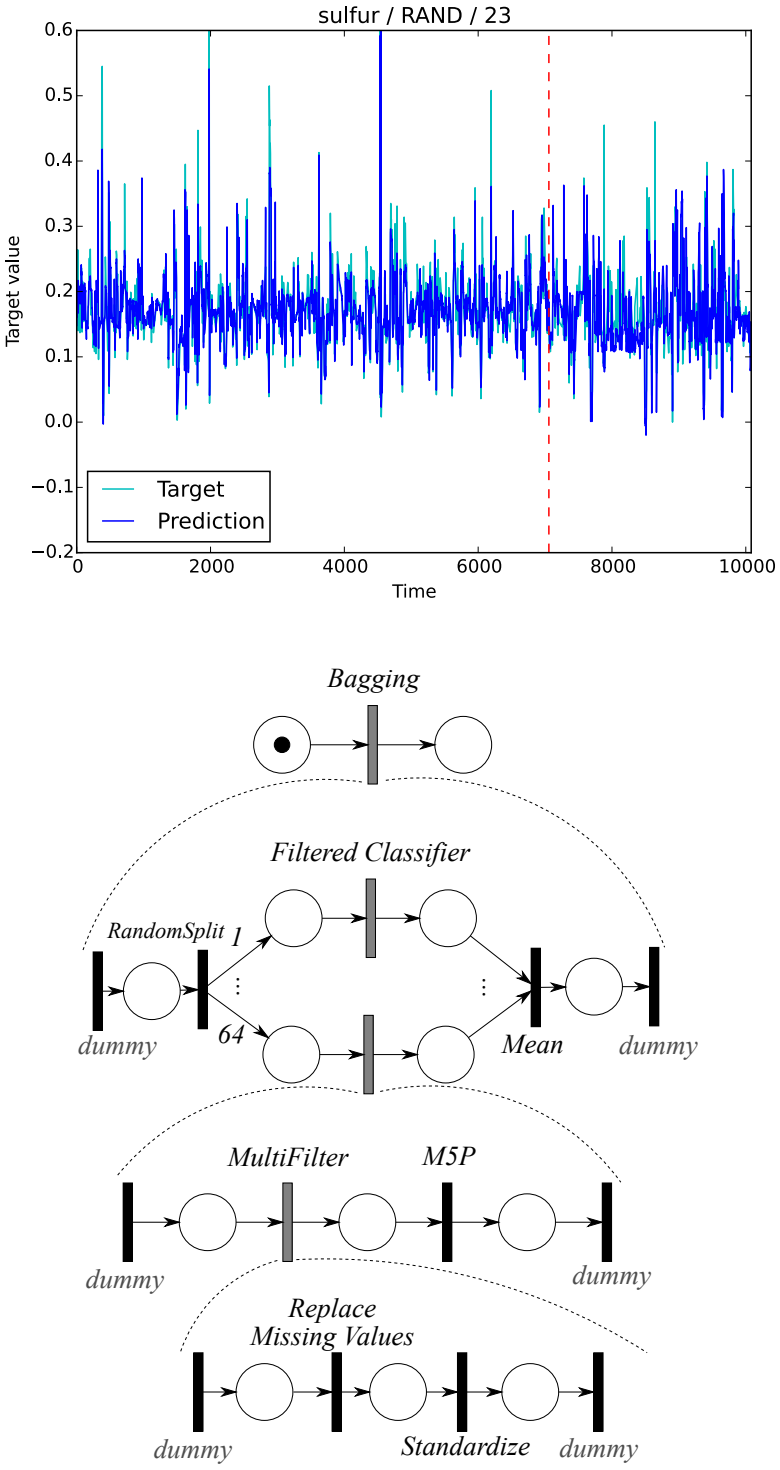
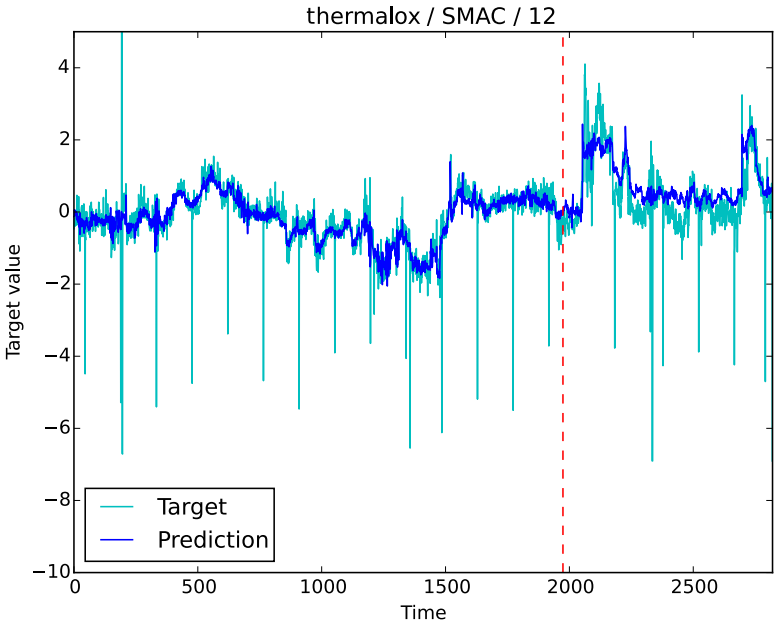


Figure A.6: *Configuration of best MCPS for 'sulfur' (bottom) and associated target value vs. prediction (top)*



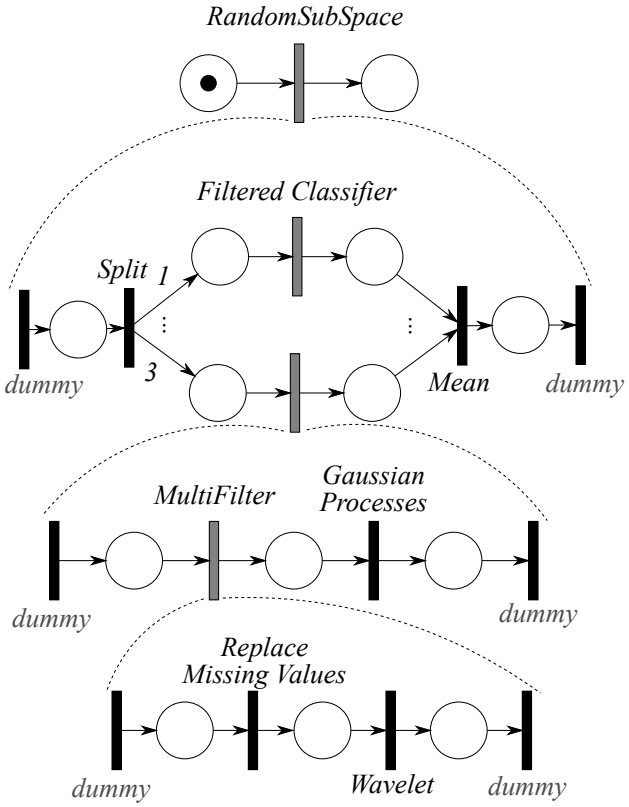


Figure A.7: *Configuration of best MCPS for 'thermalox' (bottom) and associated target value vs. prediction (top)*

A.10 Results of adaptive online prediction

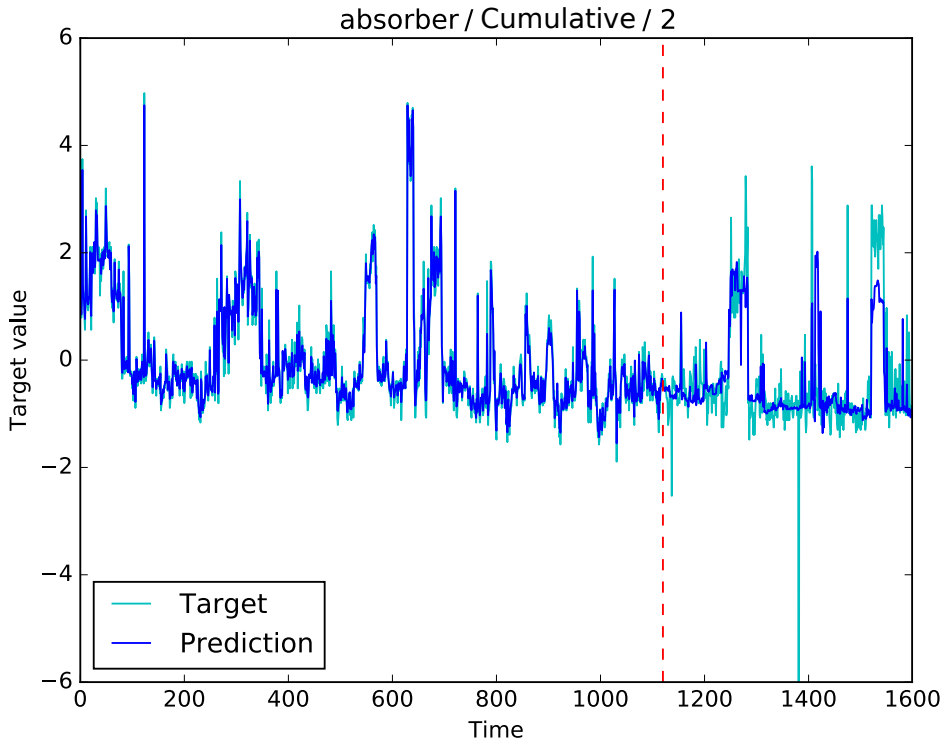


Figure A.8: *Target value and prediction of best MCPS found for 'absorber'*

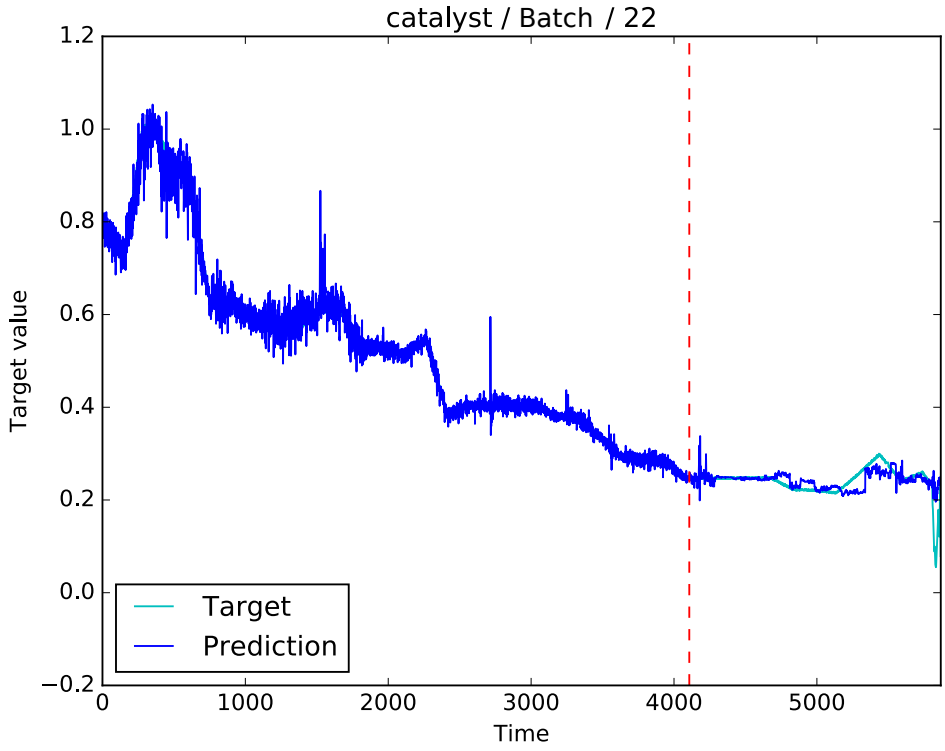


Figure A.9: *Target value and prediction of best MCPS found for 'catalyst'*

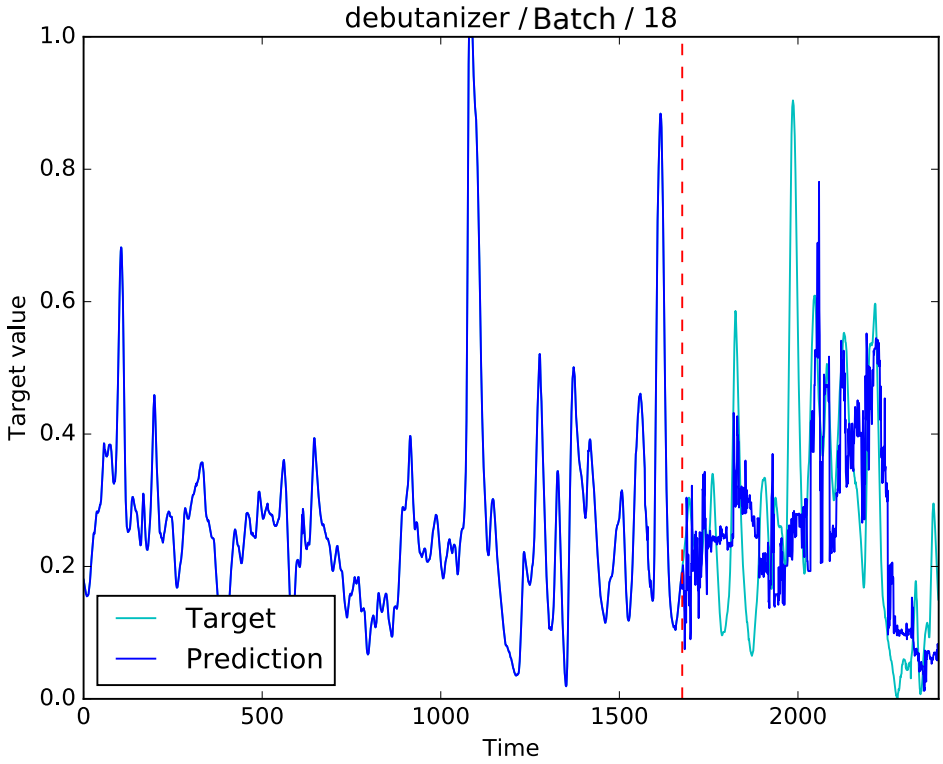


Figure A.10: *Target value and prediction of best MCPS found for 'debutanizer'*

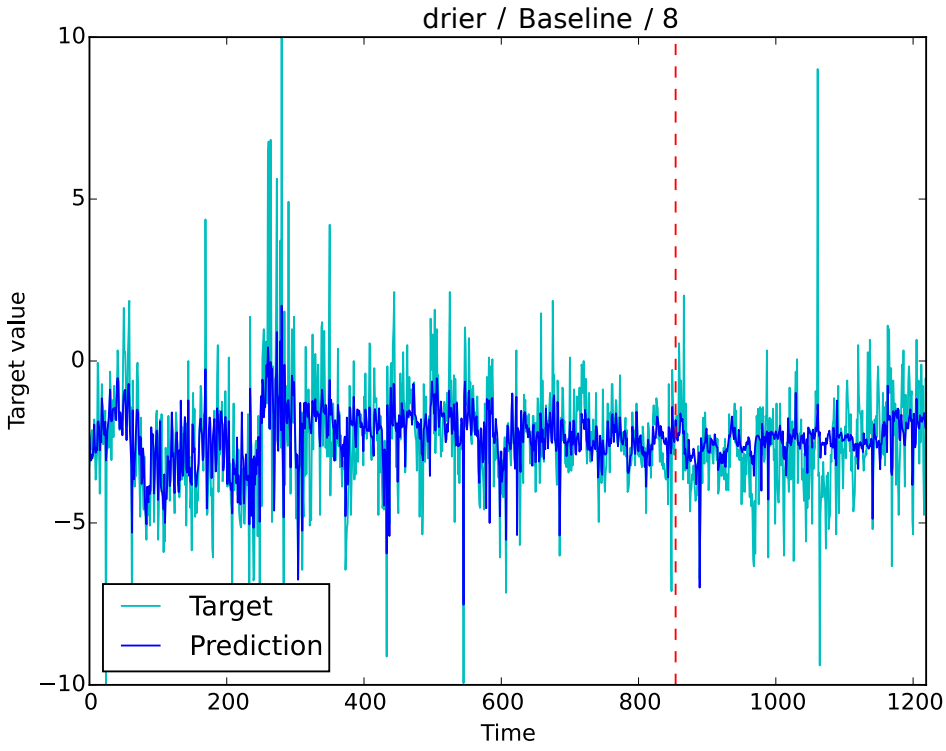


Figure A.11: *Target value and prediction of best MCPS found for 'drier'*

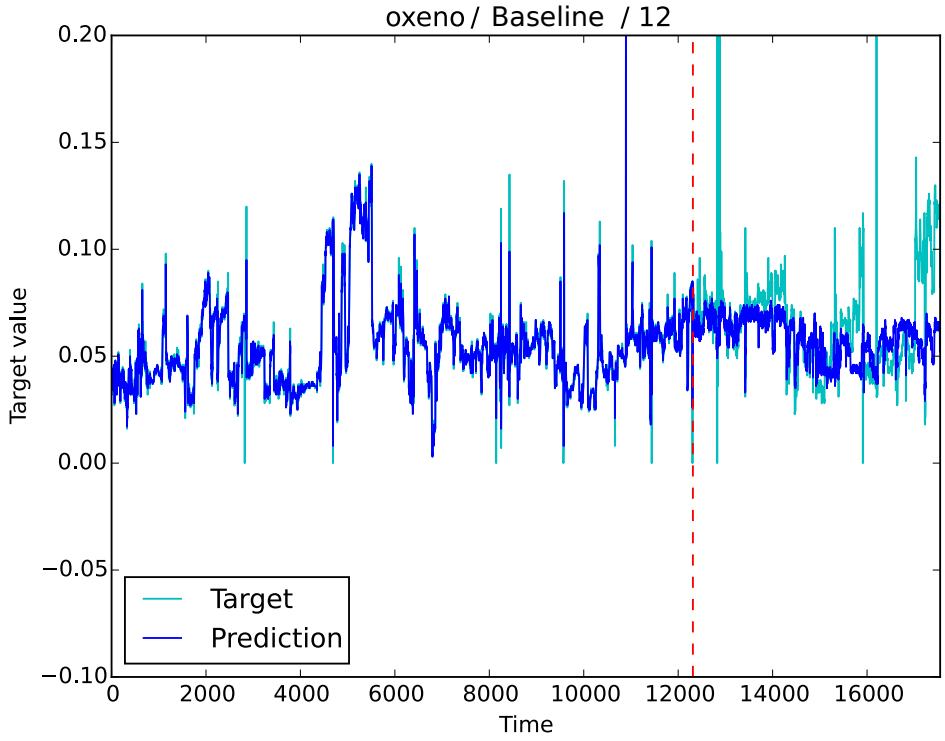


Figure A.12: *Target value and prediction of best MCPS found for 'oxeno'*

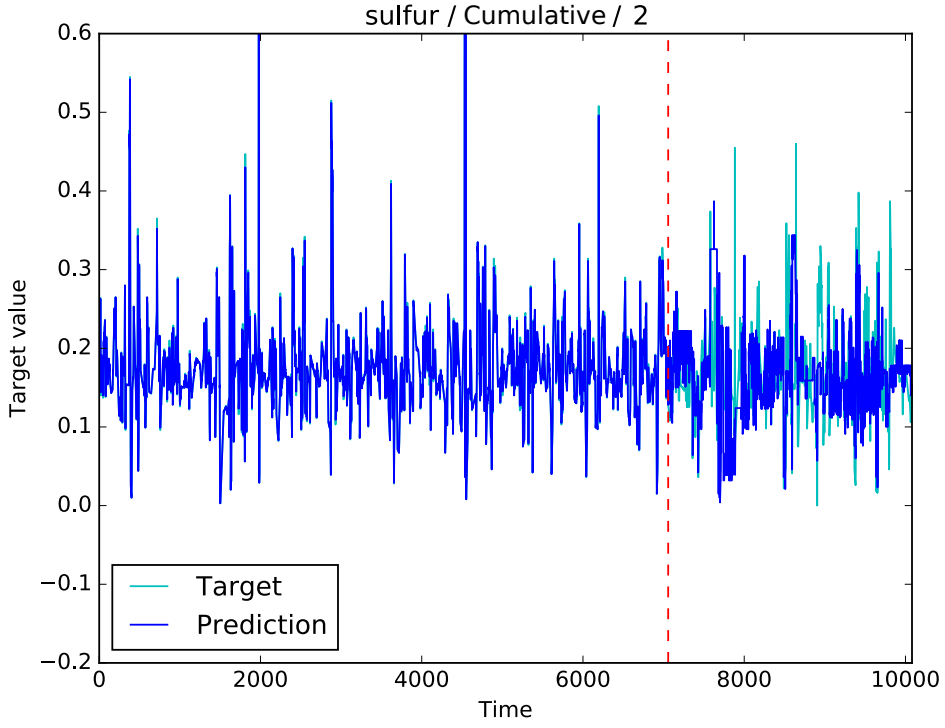


Figure A.13: *Target value and prediction of best MCPS found for 'sulfur'*

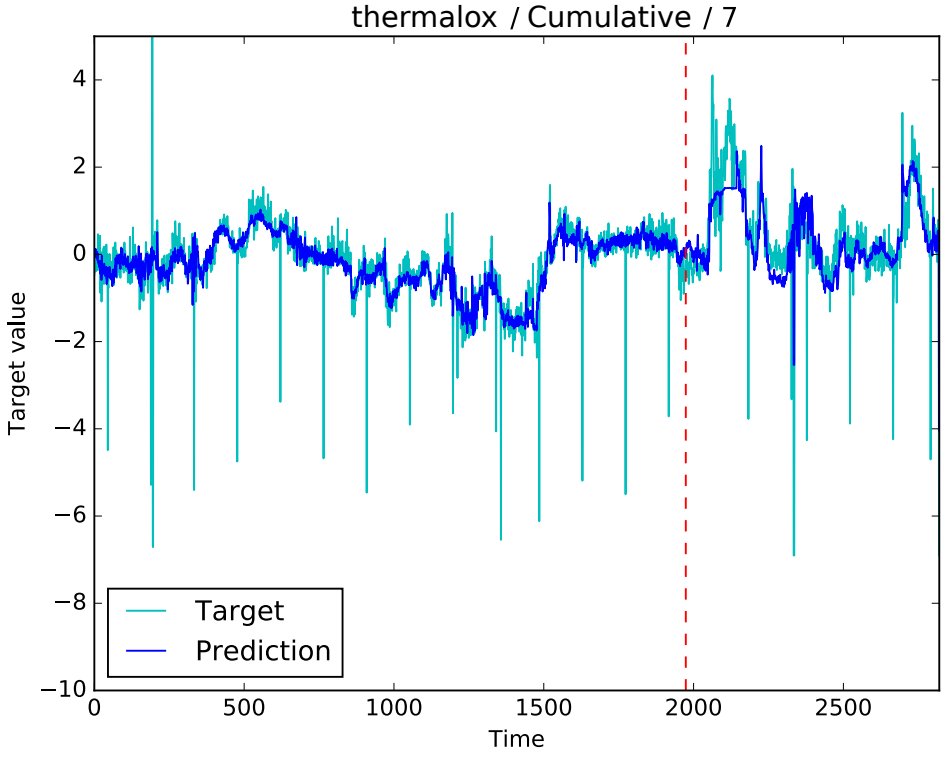


Figure A.14: *Target value and prediction of best MCPS found for 'thermalox'*

Appendix B

Filtering out shutdown periods in chemical plants: a case study

B.1 Introduction

Chemical production processes are controlled both manually and automatically to achieve a desired product quality. Physical sensors around the plants provide data streams such as temperature, pressure, humidity or flow, that are essential to monitor plant operation in real time.

Soft sensors are usually built for predicting hard-to-measure values in real time (Kadlec et al. (2009)). This building process involves data cleaning such as the removal of data from shutdown periods. Adaptive soft sensors are often updated with new data for capturing well the underlying behaviour of the process that evolves over time (Kadlec et al. (2011)). If the data from shutdowns are not removed, the predictive model can adapt to an undesirable process state. Also, during these inactive periods of production the predicted values are not meaningful from the process point of view. Therefore, an online method is necessary to automatically detect shutdowns in order to stop model adaptation.

The decision to stop a plant is usually taken by an human operator. Despite the fact that shutdowns can be scheduled for a year ahead, they may vary depending on the operating conditions (e.g. if heat suddenly increases to a dangerous level the plant has to be stopped for safety reasons).

The problem is also challenging because there is usually no single variable that can accurately and unambiguously identify the operating state of the chemical process. The solution starts from monitoring sensor values to detect changes in the process. However, not all sensors react in the same way to a shutdown. Expert support is usually needed to select the relevant sensors to monitor.

In addition, physical sensors can fail and as a result detection may be interrupted. Therefore, a shutdown detection method that works by monitoring only one sensor is unreliable in an industrial environment. It is essential to build robust methods that are able to monitor and combine several sensors at the same time.

The data of this case study has been provided by Evonik Industries and it has been collected from a chemical plant over a period of 2 years of operation. Data from 81 physical sensors has been aligned by time-stamp in order to form instances. The location of sensors in the plant causes delays between sensor signals during both shutdowns and startups. These delays make the detection more challenging. Furthermore, the annotation of the shutdown periods as ground truth for evaluation purposes has not been trivial.

B.1.1 Problem setting

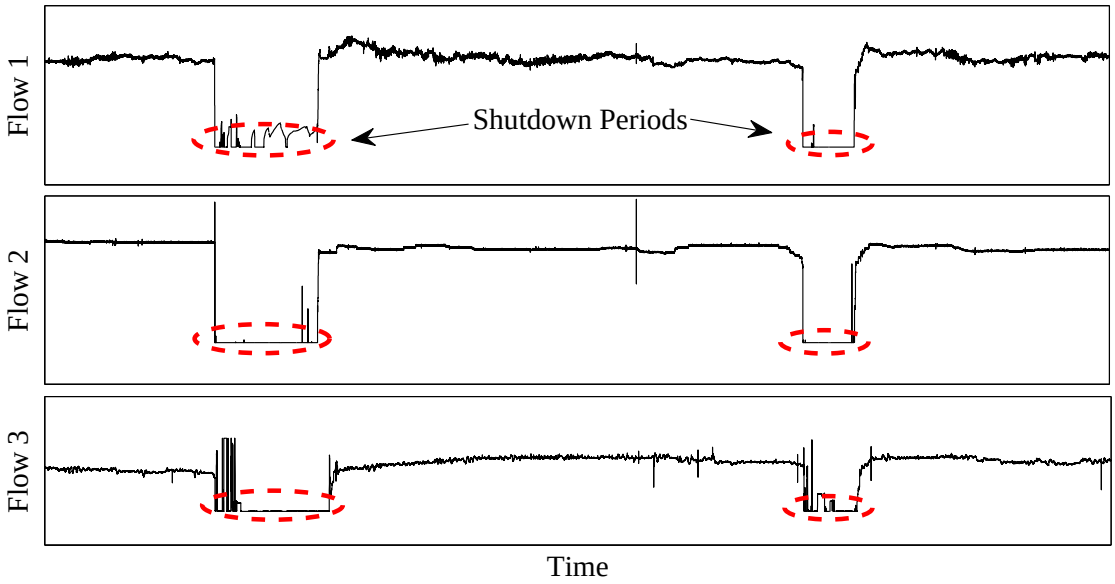


Figure B.1: *This plot shows how sensors values suddenly drop when a shutdown starts. After a period of inactivity, the values suddenly increase again when shutdown ends.*

A shutdown is a period of time $[t_\alpha, t_\omega]$ during which a process is inactive but its duration is not defined a priori. Process operation is monitored using a group of sensors. The visualisation of the values of some sensors over time makes possible a clear distinction of groups of out-of-control values that represent shutdown periods (see Figure B.1). However, other sensors are not showing any change during those periods. The selection of relevant sensors for shutdown detection is not straightforward and usually domain experts select them manually. For this particular dataset, 11 flow sensors have been manually selected after analysing the data with domain experts.

This work focused in an online scenario where data from sensors are continuously arriving to the system at fixed time intervals (e.g. every second). Let $\boldsymbol{x}_t = (x_{1,t}, ..., x_{N,t})$ be the vector of N sensor values at time t. The distribution of a relevant sensor (i.e.

sensitive to shutdowns) is given by a Gaussian mixture model

$$x_{n,t} \sim \begin{cases} \mathcal{N}(\mu_{n,0}, {\sigma_{n,0}}^2), & \text{if } t \notin [t_\alpha, t_\omega] \\ \mathcal{N}(\mu_{n,1}, {\sigma_{n,1}}^2), & \text{if } t \in [t_\alpha, t_\omega] \end{cases} \tag{B.1}$$

The formulation of a change-point in a data stream is usually given by the stopping rule

$$T = \inf\{t : s_t(\boldsymbol{x}_t) \geq \tau\} \tag{B.2}$$

where $s_t(\boldsymbol{x}_t)$ is the statistic computed over the input data and τ is the detection threshold.

Since sensors are physically located in different places of a plant, they will perceive the change of the process state at different moments. As a consequence, when a shutdown takes place there is a time interval $[t_\alpha, t_\beta]$ in which some parts of the plant are still working while others are stopped. The same situation happens during a startup. In this case, the time interval is $[t_\psi, t_\omega]$. Figure B.2 shows these time intervals in three flow sensor signals during both events. Usually, $t_\beta - t_\alpha \ll t_\omega - t_\psi$. That is, shutdowns are characterised by sudden changes while startups present gradual changes in sensor values.

The statistic s_t will increase during the interval $[t_\alpha, t_\beta]$ and it will decrease during $[t_\psi, t_\omega]$. As a consequence, different stopping rules have to be used according to the type of change that one would like to detect. That is, Equation B.2 is suitable for detecting shutdowns, while the following stopping rule is more suitable for detecting startups

$$T = \inf\{t : s_t(\boldsymbol{x}_t) < \tau\} \tag{B.3}$$

The right use of either stopping rules is associated with the process state since they are contradictory.

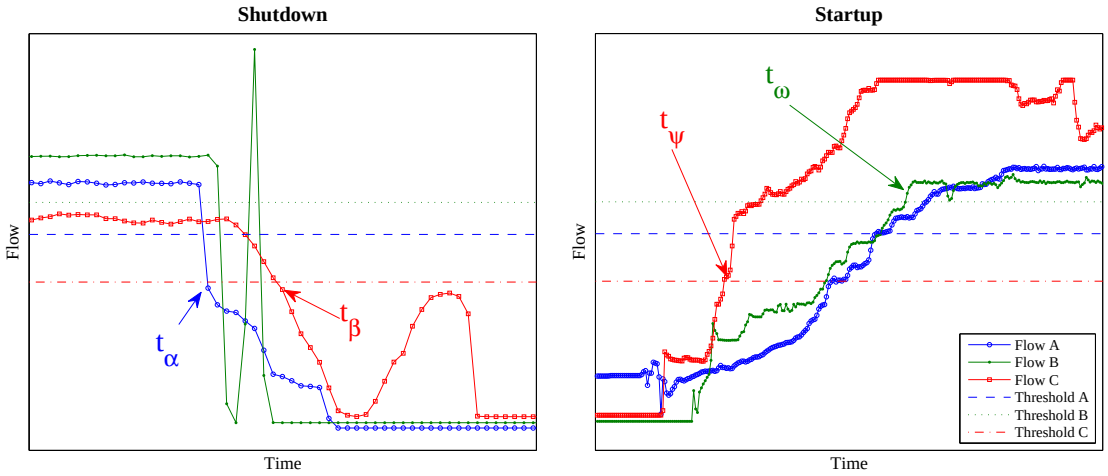


Figure B.2: *Plot on the left shows the points t_α and t_β which are chosen in annotation as the time interval of the beginning of the shutdown period. Plot on the right shows the points t_ψ and t_ω which are annotated as the time interval of the end of the shutdown.*

In the case of a shutdown its beginning should be detected quickly, that is, in a time point $t_a \geq t_\alpha$ as close as possible to t_α. On the other hand, in case of a startup one would like to detect the very end of the shutdown period, that is, a time point $t_z \geq t_\omega$ as close as possible to t_ω.

Deployment of a method for detecting these periods can result in three different outcomes: a) correct detection; b) false detection; and c) no detection. A good method should be able to maximize the number of correct detections and to minimize the number of false detection/no detection cases. This requirement is directly related to the reduction of the detection delay and the rate of false detections, which are the two most common metrics in the change-point detection literature.

B.2 Multi-sensor change-point detection methods

The detection of abrupt changes in single-sensory data (i.e. one-dimensional) has been well studied and solved. For example, the book by Basseville & Nikiforov (1993) is one of the main references for this problem. Lai (1995) surveys the sequential change-point detection methods in quality control and dynamical systems. A more recent state-of-the-art in single-sensor sequential change-point detection is presented by Polunchenko & Tartakovsky (2011) where methods of main formulations are reviewed.

The extension of this problem to multiple-sensory data has been also addressed by several authors. A two-part review of methods using different topologies and approaches can be found in Viswanathan & Varshney (1997); Blum et al. (1997). Classical methods use all the data collected until the current time t. However, those approaches are not feasible for practical purposes where data streams are continuously arriving to be processed. For this case study, a number of window limited versions of the state-of-the-art methods have been selected and implemented in order to carry out a comparative performance study.

The first chosen method was proposed by Tartakovsky & Veeravalli (2008) where a likelihood ratio test is carried out for each sensor and individual results are aggregated. The statistic to define the stopping rules of this method is referred to TV in Table B.1.

Mei (2010) proposes a family of scalable schemes for global online monitoring of data streams based on CUSUM statistics from each individual data stream. The same author extends this work in Mei (2011) where the fact that the change point may be different in each data stream is taking into account. The statistic monitored by Mei's method is referred to MEI in Table B.1.

Xie & Siegmund (2013) propose a mixture procedure based on the aggregation of the local generalized likelihood ratio (GLR) statistic of each sensor. This method assumes that the pre- and post-change distributions are Gaussian with pre-change mean being zero. This mixture includes a fraction of affected sensors by the change that has to be fixed a priori. The statistics of two different stopping rules proposed by the authors can be found in Table B.1 (as XS1 and XS2).

Finally, a new method based on Shewhart's control charts (Shewhart (1931)) has been

developed. These charts are widely used in the industry to distinguish between two states of a process (i.e. in-control and out-of-control). Section B.2.1 provides more detailed explanation of this method. The statistic monitored by this method is referred to SGZ in Table B.1.

In the recent years, multi-sensor change-point detection methods have been applied for example to fault detection (Rajagopal et al. (2008)) and intrusion detection (Tartakovsky et al. (2006)), but no works in shutdown periods detection are available so far to date.

A common assumption in the literature is that a change has to be detected as soon as possible. This is also true in the case of detecting the beginning of a shutdown. On the other hand, the pipeline structure of a big chemical plant means that the re-initialization of the sensors after a startup is delayed according to their spatial location. Thus, the detection of a startup should be deferred until all the parts of the chemical plant are working in a steady state which makes the startup detection a much more challenging problem.

B.2.1 Multi-sensor change-point detection method based on control charts

In order to control a quality measure, an upper and lower thresholds are computed with historical data. It is common to get these thresholds using the 3σ-rule, which state that for a normal distribution the 99.7% of values lies in the interval $(\mu - 3\sigma, \mu + 3\sigma)$ where μ is the mean and σ is the standard deviation of the sample.

To extend this method to this case study, the limits for each sensor are computed as $L_n = \tilde{\mu}_n - 3\hat{\sigma}_n$ and $U_n = \tilde{\mu}_n + 3\hat{\sigma}_n$, where $\tilde{\mu}_n$ is the median and $\hat{\sigma}_n \approx 1.4826\text{MAD}$ is the estimation of the standard deviation using the median absolute deviation (MAD) computed over the historical data.

The presence of outliers in the data might lead to false detections. Therefore, to increase robustness of the method a window of out-of-control values have been introduced for each sensor. Thus,

$$\boldsymbol{C}_{n,t} = (\mathcal{B}(x_{n,t-r}), \ldots, \mathcal{B}(x_{n,t})) \tag{B.4}$$

is the window of r binary values for the sensor n where

$$\mathcal{B}(x_{n,t}) = \begin{cases} 1, & \text{if } x_{n,t} \notin [L_n, U_n] \\ 0, & \text{otherwise} \end{cases} \tag{B.5}$$

A window for each sensor is monitored and weighted according to its reliability. Binary weights are used to discard those sensors which might be failing. A similar approach to identify faulty sensors is presented in Seron et al. (2012). Let

$$\gamma_{n,t} = \sum c_i, \quad \forall c_i \in \boldsymbol{C}_{n,t} \tag{B.6}$$

be the number of out-of-control values in the window $\boldsymbol{C}_{n,t}$. Each weight is then updated

in each time t as

$$w_n = \begin{cases} 1, & \text{if } \gamma_{n,t} \in [\mathcal{L}, \mathcal{U}] \\ 0, & \text{otherwise} \end{cases} \tag{B.7}$$

where $\mathcal{L}$ and $\mathcal{U}$ are the thresholds of $\boldsymbol{C}_{1..N,t}$ computed using the Hampel identifier (Davies & Gather (1993)).

Final decision for detecting a change is taken by aggregating the weighted counters of all sensors. Therefore, using

$$s_t(\boldsymbol{x}_t) = \max_{1 \leq n \leq N} (w_n \gamma_{n,t}) \tag{B.8}$$

as a statistic (SGZ in Table B.1) both quick detection during a shutdown and deferred detection during a startup are ensured. This aggregation can therefore deal with the delays between sensors due to their spatial location. Pseudo-code for this method is presented in Algorithm 3.

Algorithm 3 Function processSample($\boldsymbol{x}_t$)

for $n = 1 \rightarrow N$ **do**
    $\mathcal{B}$ = isOutlier($x_{n,t}$) ▷ Eq. B.5 equivalent
    $\boldsymbol{C}_n$.removeOldest() ▷ $\boldsymbol{C}_n$ is a list of size r initialized as a global variable
    $\boldsymbol{C}_n$.append($\mathcal{B}$) ▷ Eq. B.4 equivalent
end for
$[\mathcal{L}, \mathcal{U}]$ = getReliabilityThresholds($\boldsymbol{C}_{1,\ldots,N}$)
$s_t = 0$
for $n = 1 \rightarrow N$ **do**
    γ_n = sum($\boldsymbol{C}_n$) ▷ Eq. B.6 equivalent
    w_n = inLimits($\gamma_n, \mathcal{L}, \mathcal{U}$) ▷ Eq. B.7 equivalent
    s_t = max($s_t, w_n \cdot \gamma_n$) ▷ Eq. B.8 equivalent
end for
if processActive **and** $s_t \geq \tau$ **then** ▷ Eq. B.2 equivalent
    shutdownDetected()
    processActive = false ▷ processActive is defined as a global variable
else if ¬processActive **and** $s_t < \tau$ **then** ▷ Eq. B.3 equivalent
    startupDetected()
    processActive = true
else
    continueProcessSample($\boldsymbol{x}_t$)
end if

B.3 Experimental evaluation

The goal of this experimental evaluation is to compare the performance and reliability of different multi-sensor change-point detection methods in this case study and to select

the most suitable for a production environment. For that purpose, evaluation measures are defined and then an experimental protocol is established. Finally, the results of the conducted experiments using the Acrylic Acid dataset (described in Appendix A.1) are discussed.

B.4 Evaluation measures

In change-point detection literature there is a trade-off between detection delay and false alarm rate. While the objective is to minimize both measures, a threshold for a quick detection delay can increase the number of false alarms (i.e. incorrect detections). In this case study the aim is to avoid false detection at all cost but at the same time without long delays.

Although shutdowns and startups are both changes from the theoretical point of view, they are being distinguished during the experiments because they are different in practice as explained in Section B.1.1. Therefore, the selected measures are the shutdown's detection delay as $\Delta_\alpha = t_a - t_\alpha$ and the startup's detection delay as $\Delta_\omega = t_z - t_\omega$ where t_a and t_z are the times of the detection for each case.

Three common quantities to measure the predictive performance in regression problems are computed: a) mean absolute error (MAE); b) root mean squared error (RMSE); and c) correlation between target and prediction (ρ).

B.5 Experimental setting

The dataset has been split in two equally-sized parts. The first half is used for training and the calibration of parameters and the second half for evaluating the methods. Each half contains 22 change-points (11 shutdowns and 11 startups) that have been manually annotated.

Table B.1 contains the distinctive formulas for calculating the time changing statistical values used in the stopping rule of each method. In this table, t is the current time, r is the size of a temporal window,

$$\ell_n(t, k, \mu_n) = \sum_{i=k+1}^{t} (\mu_n x_{n,i} - \mu_n^2/2) \tag{B.9}$$

is the log-likelihood of observations accumulated by time t, μ_n is the mean of the data during a shutdown,

$$\hat{\mu}_{n,k,t} = \frac{\sum_{i=k+1}^{t} x_{n,i}}{t-k} \tag{B.10}$$

is the maximum likelihood estimator of the mean, and p_0 is the posterior probability of x_t in the distribution $\mathcal{D}_0$ (i.e. data in steady state).

Method	$s_t(\boldsymbol{x}_t)$
TV	$\max_{t-r\leq k\leq t} \sum_{n=1}^{N} \ell_n(t,k,\mu_n)$
MEI	$\sum_{n=1}^{N} \max_{t-r\leq k\leq t} \ell_n(t,k,\mu_n)$
XS1	$\max_{t-r\leq k\leq t} \sum_{n=1}^{N} \log(1-p_0+p_0\exp[\ell_n^+(t,k,\mu_n)])$
XS2	$\max_{t-r\leq k\leq t} \sum_{n=1}^{N} \log(1-p_0+p_0\exp[(\hat{\mu}_{n,k,t}^+)^2/2])$
SGZ	$\max_{1\leq n\leq N}(w_n\gamma_{n,t})$

Table B.1: *Formulas used for $s_t(\boldsymbol{x}_t)$ in Equations B.2 and B.3*

r	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
TV	16	16	16	16	16	16	16	16	16	16	**10**	10	10	10	10	10	10
MEI	18	17	17	18	19	17	18	18	19	20	19	**11**	11	11	11	11	11
XS1	80	68	**17**	80	23	58	60	25	46	57	57	57	57	57	46	57	35
XS2	16	16	16	16	16	16	16	16	16	16	**10**	10	10	10	10	10	10
SGZ	9	5	**5**	5	5	5	5	5	5	5	5	5	5	5	5	5	5

Table B.2: *Limit values τ for each method and window size r*

The limits of the stopping rules for all the methods have to be chosen to minimize both the detection delay and the rate of false alarms. In the case of XS1, XS2, MEI and TV this limit lies in $0 < \tau \lesssim \sum_{n=1}^{N} r \cdot \mu_{n,1}^2/2$, and for SGZ in $0 < \tau < r$. Limits for each method have been automatically set using the first half of the dataset as input. Thus, the computed limit is the minimum s_t value that avoids the false alarms and detects all the change-points. Table B.2 contains these limits for each window size.

To assess the impact of filtering out shutdown periods in the predictive performance, a PLS regression model with 10 latent variables is trained using the first half of the dataset once it has been preprocessed. Then, in the online phase, the model predicts the target value for each incoming instance. After that, the model is adapted using recursive PLS algorithm (Joe Qin (1998)) when the true target value is provided. The resultant MCPS is shown in Figure B.3.

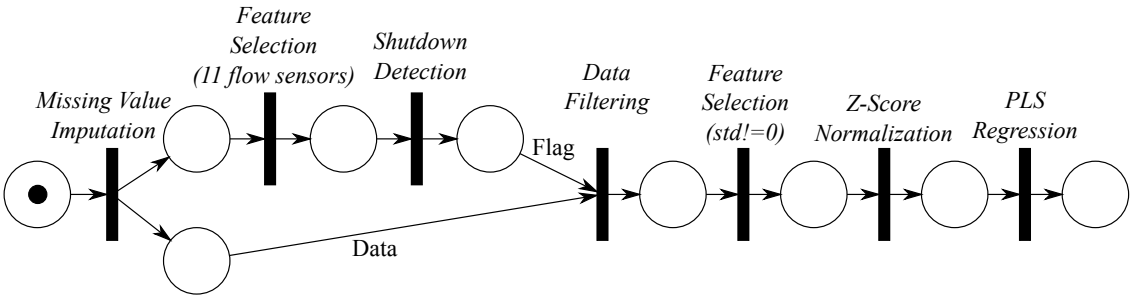


Figure B.3: *MCPS for acrylic acid dataset. The 'Data Filtering' transition receives two tokens: 1) a flag indicating the state of the process (active or not); and 2) the data to be processed. Once fired, it generates a new token with the filtered data that continues to the rest of the MCPS.*

B.6 Experimental results

B.6.1 Detection delay

A sample of the s_t values computed for each method is shown in Figure B.4. The statistics for TV, MEI, XS1 and XS2 are very similar because they are all based on the likelihood. The flat signal of SGZ indicates that the windows of out-of-control values is full ($r = 25$).

Figure B.5 compares the median values of the detection delays during the shutdown phases. The window size almost does not affect to the these types of detections because the s_t values quickly increase during the shutdown phase and overcome the threshold τ. The MEI, TV and XS2 methods reported lower detection delays than XS1 and SGZ.

On the other hand, the window size has a significant effect on the median values of the detection delays during the startup phases as shown in Figure B.6. A negative delay means that the change-point has been detected before t_ω. In this case study, a small positive detection delay is desired. In both XS1 and SGZ a value of $r = 30$ satisfies that requirement. However, a value of $r = 75$ is needed for TV, MEI and XS2.

False alarms have only be reported for MEI with window sizes $r = \{20, 25, 30, 45, 50, 55, 60\}$ and for SGZ with window sizes $r = \{20, 65\}$. The rest of the methods have not raised any false alarms within the experimental setup.

The methods behave similarly although selection of window size makes a difference. The MEI method detects the changes quicker than the other methods but at the same time it raises false alarms in some of the cases. Any of the following configurations would be suitable to be implemented in a production environment: (TV, $r = 70$), (MEI, $r = 75$), (XS1, $r = 30$), (XS2, $r = 70$) and (SGZ, $r = 30$). If memory requirements are costly, the XS1 method would be the best choice because its performance is better than the others with lower window size.

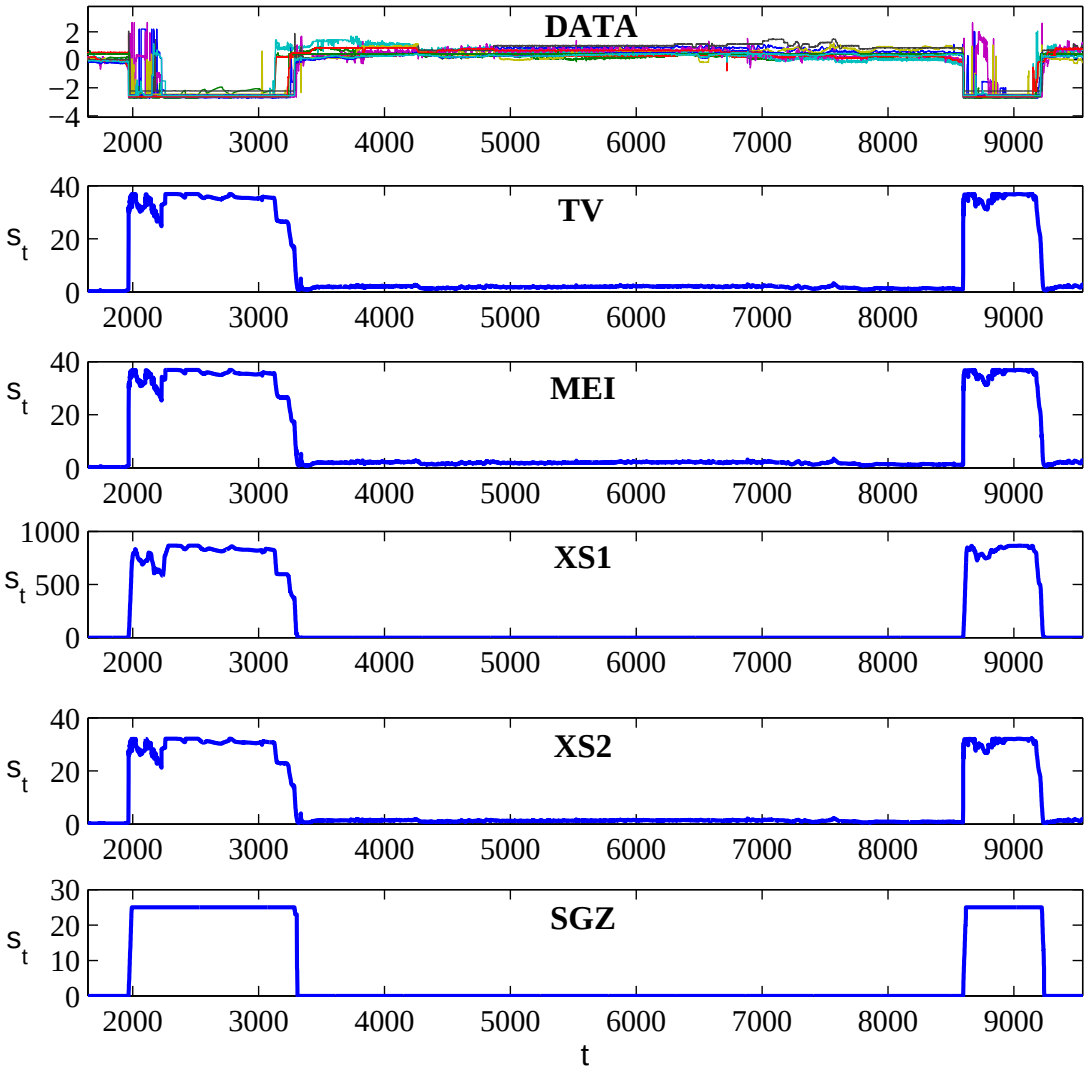


Figure B.4: *Subset of the observed data and s_t values for all the methods for $r = 25$*

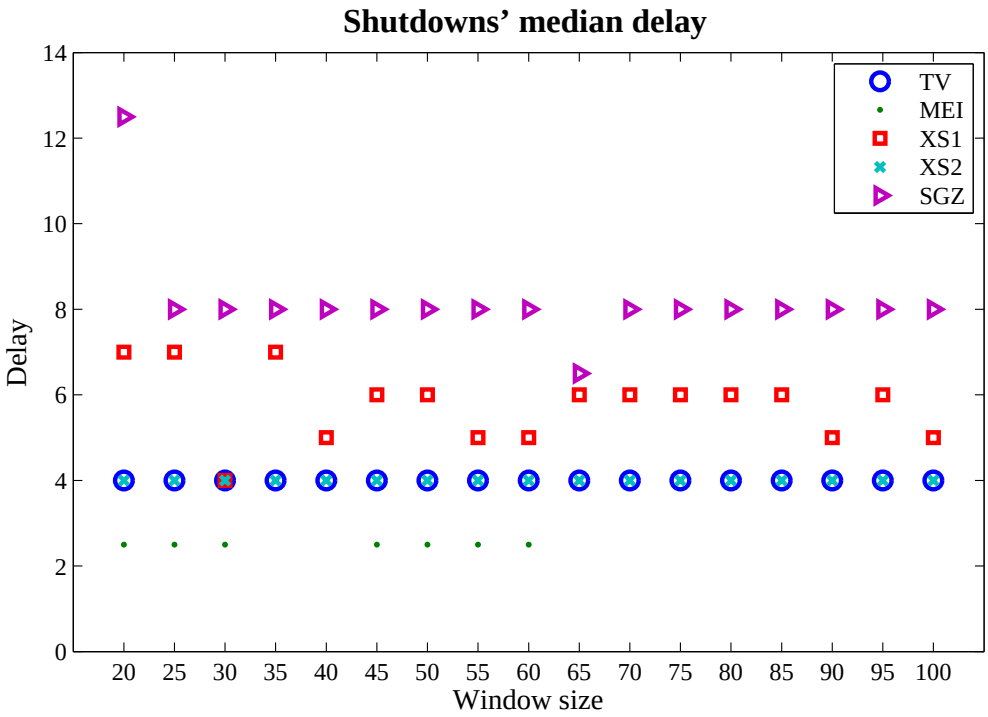


Figure B.5: *Median of the detection delays of the shutdowns*

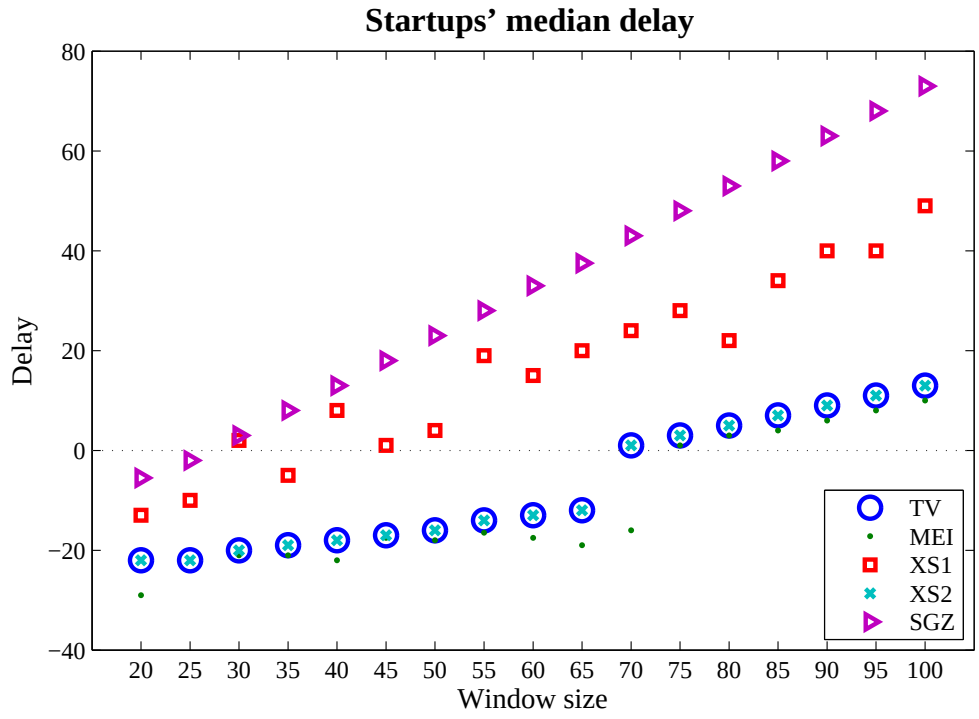


Figure B.6: *Median of the detection delays of the startups*

B.6.2 Predictive performance

Results of predictive performance are shown in Table B.3. 'None' is when no shutdown detection is performed, 'Oracle' means that the detection has been done manually and the rest are the different shutdown detection methods evaluated. The 'Oracle' method has the lowest error and the highest correlation as expected. Also, not using any detection has the poorest performance as expected. See Figure B.7 that shows how the regression model keeps predicting (green line) when it should not because there is a shutdown (red line). There is not much difference between the different multi-sensor change-point detection methods evaluated, though SGZ has the lowest RMSE and the higher correlation among them.

Method	**MAE**	**RMSE**	ρ
None	0.717023	0.333022	0.946592
Oracle	0.708273	0.084207	0.999845
TV	0.708572	0.271488	0.963712
MEI	0.708586	0.271493	0.963711
XS1	0.708663	0.274622	0.962837
XS2	0.708572	0.271488	0.963712
SGZ	0.708743	0.269687	0.964231

Table B.3: *Predictive performance of RPLS using different shutdown detection methods*

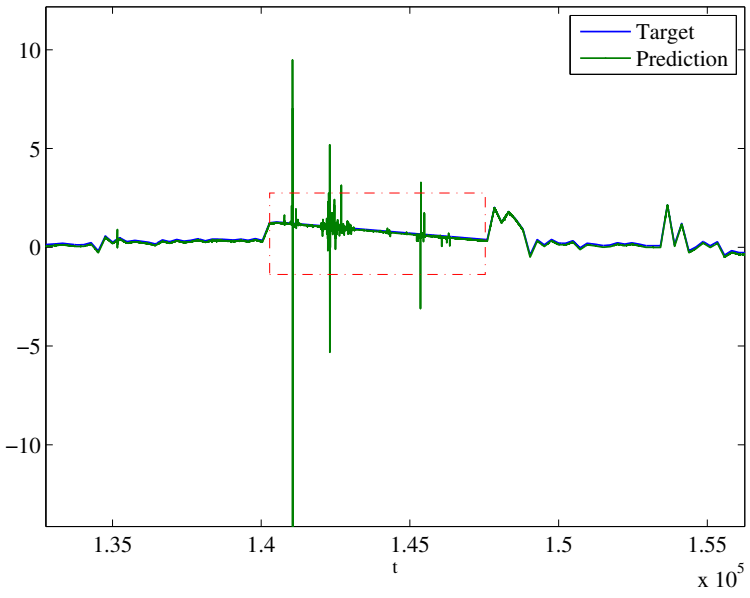


Figure B.7: *Target value and prediction during a shutdown period. The regression model keeps predicting (green line) when it should not because there is a shutdown (red box) and thus is returning predictions which are not meaningful.*

B.7 Conclusions

The task of this real case study has been to identify the shutdown periods of a chemical plant using raw data provided by the physical sensors. For that, the problem setting has been initially formulated and then a number of state-of-the-art methods in multi-sensor change-point detection have been selected for comparison. In addition, a new robust method based on control charts (which are very popular in the industry) has been developed.

A series of experiments have been conducted using a common framework in order to evaluate the detection performance of all these methods for this case study. The impact of these detection methods on the predictive performance of a soft sensor based on PLS has been evaluated. The results show that the automatic detection helps to increase the predictive performance. In particular, the new SGZ method is the one with lowest RMSE and highest correlation.

Appendix C

Effects of change propagation resulting from adaptive preprocessing in multicomponent predictive systems

C.1 Introduction

In online supervised learning, data-driven predictive models are built incrementally using labelled instances. Preprocessing of those instances is often needed before the predictive model learning phase (Han et al. (2011)). Common preprocessing methods include imputation of missing values, data normalisation, noise reduction and feature selection, to name a few. A combination of various preprocessing steps with a predictive model and possible postprocessing steps results in a so called multicomponent predictive system (MCPS) which was presented in Chapter 3.

The simplest MCPS includes a predictive model as a single transition (e.g. Naïve Bayes classifier). A more complex MCPS contains several preprocessing methods (e.g. moving average filter and PCA), an ensemble of predictive models (e.g. regression tree and neural network) and an ensemble aggregation method (e.g. majority voting). In this investigation, an intermediate example consisting of three preprocessing methods (z-score normalisation, PCA, and min-max normalisation) and one classifier (GFMM neural network) have been selected (see Fig. C.1) and it is explained further in Section C.2. Please note that this example only serves as a convenient case of discussing the change propagation issues and other preprocessing steps and predictive models could have also been used.

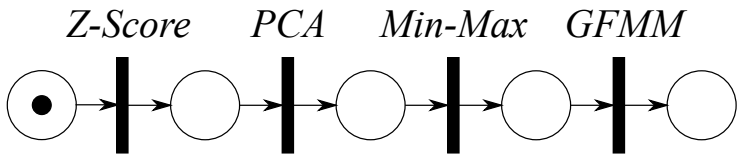


Figure C.1: *MCPS used as example*

Preprocessing methods are usually trained offline and therefore their parameters remain fixed during the whole model lifetime. However, in non-stationary environments, where data statistics can vary over time, parameters of preprocessing methods should be updated to avoid performance loss. In addition, the predictive model may need to be adapted to new data as well.

There are two main types of strategies to adapt predictive models: (a) incremental update (Bouchachia et al. (2007)) or (b) replacement with a new model (Gama et al. (2014)). The GFMM model used here as an example is updated using the former strategy, although other learning/adaptation strategies are also possible (Gabrys (2004)).

Adaptation of individual preprocessing methods has been considered by different authors. Adaptive feature selection (Anagnostopoulos et al. (2008)), recursive PCA (Li et al. (2000)) and adaptive normalisation (Ogasawara et al. (2010)) are some examples of adaptive preprocessing methods. Žliobaitė & Gabrys (2014) was the first attempt to address decoupling of adaptive preprocessing from the remaining components in a learning process with evolving data streams. One of the authors' conclusions was that such decoupling can be highly beneficial though can cause various problems if the change resulting from adapting a preprocessing step is not correctly propagated to subsequent components of an MCPS.

While in Chapter 5 the proposed autonomous adaptation strategies were for the whole MCPS, this appendix concentrates on investigating and discussing the effects that an adaptation of a single preprocessing step can have on the overall predictive system behaviour, complexity and performance. The concept of change propagation in MCPSs is also introduced here highlighting its importance when adapting individual components. Furthermore it is shown how the lack of a change propagation mechanism can lead to inconsistencies in the model representation.

The remainder of this appendix has been organised as follows. Section C.2 begins by introducing the adaptation of MCPSs, and presents the components of the case study. Section C.3 describes four adaptation scenarios used in the experiments, followed by an experimental study with different datasets in Section C.4. Conclusions are presented in Section C.5.

C.2 Reactive adaptation of MCPSs

In non-stationary data-streams, the relation between the input and the target can drift over time and predictive models have to be updated to avoid performance loss. Currently, the most common way to detect the need for adaptation is to monitor model performance (Gama et al. (2004)). However, such an approach means that a number of instances have to be wrongly predicted before the problem can be actually detected. In addition, true labels (correct values which are predicted) are needed to estimate the loss and they may not always be available. In an MCPS one could add detection mechanisms in the early stages to monitor and adapt to the changes of the input data distribution (e.g. measuring abrupt changes in the mean of an input attribute) therefore not completely relying on the

ability to calculate prediction error. This approach is referred as reactive adaptation to the input data distribution changes.

A possible control mechanism is to establish preconditions and postconditions in all the system components. Therefore, if any of the conditions are not fulfilled, action must be taken. For example, in the case of min-max normalisation method if an input value is not within the $[min, max]$ range, the preprocessing component should be adapted to the new range. Other methods such as recursive PCA may include a post-condition that detects if previous and current principal components differ significantly.

In order to illustrate the point discussed above, three preprocessing methods (z-score normalisation, PCA for dimensionality reduction, and min-max normalisation) and one incremental predictive model (GFMM neural network) have been selected. In the following subsections all four components are briefly described.

C.2.1 Dimensionality reduction

Many datasets include variables which might not be relevant for the prediction task, unnecessarily increasing model complexity. For that reason, reducing the number of variables is often desirable. A common technique is performing Principal Component Analysis (PCA) and then selecting the first k components that explain a given percentage of dataset variance. Let X be the $n \times p$ input dataset where n is the number of samples and p the number of variables. Then, the reduced dataset is given by $X' = XW_k$, where W_k is the $p \times k$ loading matrix.

C.2.2 Z-Score normalisation

Data normalisation is recommended before performing PCA, particularly if the variables have different units. Z-score normalisation, which scales the data to a standard normal distribution with zero mean and unit variance, is a common choice (see Eq. C.1).

$$f(x) = \frac{x - \mu}{\sigma} \tag{C.1}$$

C.2.3 Min-max normalisation

The min-max method normalises the values of each attribute of a dataset to a range $[low, high]$ using the following formula, assuming $max > min$:

$$f(x) = (high - low) \cdot \frac{x - min}{max - min} + low \tag{C.2}$$

If the max and min values of an attribute are known a priori and do not change over time (e.g. they are given by the physical limitations of a measuring equipment), it can be assumed that all the values are going to be in the range $[low, high]$. On the other hand, if those limits are unknown, they have to be estimated from the available data.

Although the consequences of not dealing with extreme outliers (Pyle (1999)) are taken into consideration, this study assumes that they are being handled in a preceding preprocessing step for the sake of simplicity. For instance, Ogasawara et al. (2010) smooth the data and remove outliers during the training phase before the normalisation. Nevertheless, detection and handling of outliers are another preprocessing components that can potentially be adapted.

C.2.4 GFMM classifier

The General Fuzzy Min-Max (GFMM) neural network is a method that can be used for both classification and clustering problems. GFMM models the density of the input space using fuzzy hyperboxes (i.e. multidimensional generalisation of a rectangle with an associated membership function) to cover all training instances, with the j-th hyperbox being characterised by its min ($\boldsymbol{V}_j$) and max ($\boldsymbol{W}_j$) points (for an example refer to Fig. C.4). The reason of selecting GFMM as model in this study is two-fold: 1) it is easy to interpret and visualise with two-dimensional data; and 2) it requires data within the $[0, 1]$ range so a min-max normalisation step is needed. For full description of the incremental learning algorithm and further details of the GFMM family, please refer to the relevant literature (Gabrys & Bargiela (2000); Gabrys (2004)).

C.2.5 Change propagation

Change propagation is a recurrent topic in complex systems where changing a component may require to adapt further parts of the system (Giffin et al. (2009)). Understanding how components are connected is crucial for performing a correct adaptation. Some works have addressed change propagation in different type of structures such as dynamic trees (Acar et al. (2005)) or directed acyclic graphs (Acar et al. (2002)). However no works focusing on MCPSs have been found.

Any parametric change in one component of an MCPS can have a knock-on effect on the data consumed by the remaining components. In this context, change propagation can be defined as the mechanism of transferring relevant information across the system to keep it consistent. For instance, if the input space is transformed, the model should be mapped to this new space. That is, any change in the preprocessing parameters transforming the input space should be reflected in the model. In the MCPS used as example, the three preprocessing methods perform consecutive linear transformations, so inverting such transformations should be done in the right order. In other cases however, such propagation may not be necessary.

Figure C.2 illustrates how a change in the min-max component is propagated to the GFMM model via a coloured token with meta-data, using the Petri net formalism. The token triggers adaptation of GFMM using supplied information. Since the original formal definition of MCPS described in Chapter 3 does not support this kind of token, it can be expanded to define a coloured MCPS (or cMCPS) as an MCPS with two additional conditions: a) the places $P \setminus \{i, o\}$ can contain up to two tokens assuming that each of

them is of a different colour/type (i.e. data and meta-data) and, as a consequence, b) the Petri net is not safe (i.e. each place $p \in P$ can contain more than one token at any given time).

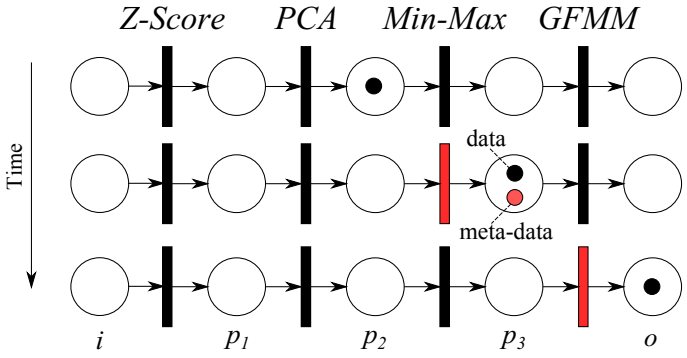


Figure C.2: *Change propagation in an MCPS. When [min,max] values of data at p_2 are out of the range of [min,max] parameters of Min-Max transition, the latter needs to be adapted. In that case, a meta-data token (in red) is generated containing the new [min,max] values which will be used to adapt the GFMM model.*

Although a sequence of linear transformations can be unified into a single linear transformation, the cost of system adaptation (Žliobaitė et al. (2015)) can be reduced by inverting only the transformations of affected components. In this scenario, if max or min limits of Eq. C.2 are updated, the hyperboxes have to be renormalised. To this end, the hyperboxes are first denormalised from the previous $[min', max']$ range using Eq. C.3, and then normalised again with the new max and min values using Eq. C.2.

$$f'(x) = \frac{x - low}{high - low} \cdot (max' - min') + min' \tag{C.3}$$

Similarly, if the loading vectors of PCA are modified, hyperboxes have to be denormalised using Eq. C.3 before reversing the PCA transformation. The reverse PCA transformation is given by $X = X'W_k^T$. Then, the hyperboxes are transformed using the new loading vectors and normalised again to the $[0, 1]$ range.

The same situation occurs when any parameters of z-score are changing (e.g. μ is now μ' and σ is σ'). All the transformations have to be reversed ($x' = \sigma x + \mu$) and then reapplied with the new parameters ($f(x') = \frac{x'-\mu'}{\sigma'}$). Otherwise, model representation would be inconsistent with the new input space.

While these reverse transformations may seem tedious and not always possible, the following section introduces four scenarios in order to investigate the severity of possible impact which may result from not propagating the changes correctly.

C.3 Scenarios

This section introduces four scenarios where data is preprocessed and then classified as shown in Fig. C.1. All the scenarios share the same training phase, where a batch of data is preprocessed and then used to build the initial classifier. The main difference between the scenarios is the degree of adaptation of the predictive system, as summarised in Table C.1:

#1 - No adaptation All the components remain fixed at all times. The parameters extracted during the training phase are then used during online preprocessing of the data stream but no adaptation is performed at any level.

#2 - No adaptive preprocessing The preprocessing methods are not adapted. However, this and the following scenarios assume that true labels of the instances are provided just after they are classified. Hence, the labelled instance is fed to the incremental learning algorithm that updates the classification model.

#3 - Adaptive preprocessing without propagation A buffer with recent instances is used for computing the PCA loadings. Then, the incoming instances are transformed into the new space formed by the principal components. In addition, scaling parameters (max and min) are constantly checked and updated if any value is out of those limits. These local changes are however not propagated further through the model.

#4 - Adaptive preprocessing with propagation Local changes are propagated through the system. The classification model is therefore mapped to the new input space: the hyperboxes of GFMM network are transformed to the original space by reversing the preprocessing steps, and then the updated preprocessing methods are applied to the hyperboxes.

Scenario	Adaptive Preprocessing	Change Propagation	Adaptive Model
#1			
#2			✓
#3	✓		✓
#4	✓	✓	✓

Table C.1: *Summary table of scenarios*

C.4 Experimental study

This experimental study considers the scenarios presented in Section C.3 in order to illustrate the implications of adapting some components of the preprocessing and the classifier.

To compare the system performance between the scenarios, the following values have been measured over four different datasets: a) classification error; b) maximum hyperbox membership degree (as a measure of classifier confidence); c) number of samples out of $[0, 1]$ range; and d) number of hyperboxes.

The first 200 samples of each dataset have been selected for training and the rest for sequential testing/adaptation.

C.4.1 Synthetic data stream

A synthetic dataset (SYN) has been generated for illustration purposes, consisting of 2 classes that drift over time. The mean of attribute 1 is increasing over time for class 1, while the mean of attribute 2 remains constant. The opposite situation happens for class 2. Fig. C.3 shows the state of the dataset at different timestamps.

The dataset contains 600 samples, 2 numerical attributes and 2 classes. Samples are randomly generated from a normal distribution with different means for each attribute but unit standard deviation. Only min-max normalisation is performed as part of preprocessing for the synthetic dataset.

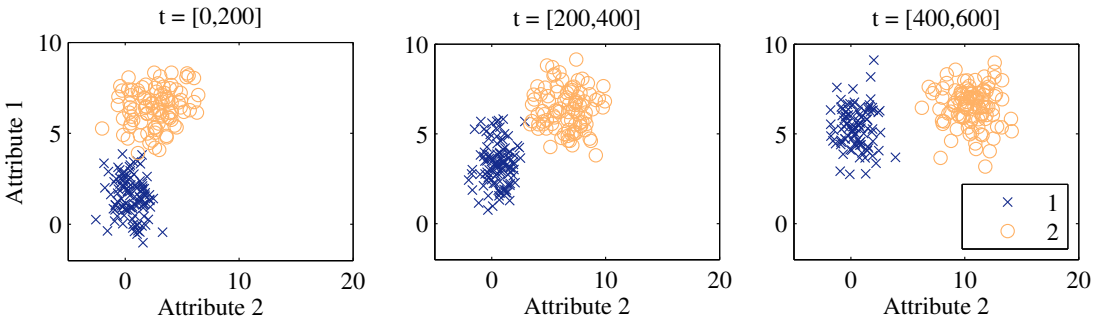


Figure C.3: *SYN dataset at different timestamps*

C.4.2 Real data streams

Three publicly available datasets have been selected. The first real dataset (ELEC, Harries (1999)) was collected from the Australian New South Wales Electricity Market and contains 45,312 instances, 7 attributes and 2 classes which are affected by demand and supply of the market.

The second dataset (COVERTYPE, Blackard & Dean (1999)) contains the forest cover type for 30x30 meter cells obtained from US Forest Service. The task is to predict forest cover type from cartographic variables. It consists of 581,012 instances, 54 attributes and 7 classes.

The third dataset (GAS, Vergara et al. (2012)) contains 13,910 measurements from chemical sensors used to distinguish between different gases. It consists of 128 attributes and 6 classes.

For each dataset, the first 600 samples have been selected. First 200 samples have been used as training set for building the model and the rest for online testing. The pre-processing phase is composed of z-score normalisation, dimensionality reduction using PCA and finally, min-max normalisation.

C.4.3 Results for synthetic data stream

The results obtained from SYN dataset are shown in Figs. C.4 and C.5. Fig. C.4 shows the final snapshot of data points and hyperboxes for each scenario.

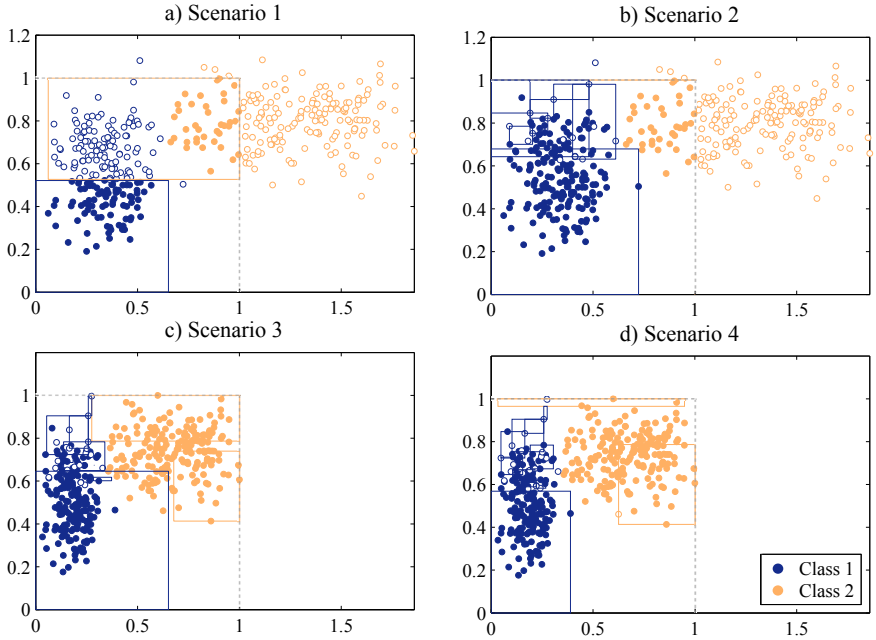


Figure C.4: *Plots of the SYN testing data together with GFMM hyperboxes for each scenario*

Points from class 1 (blue) are drifting to the space covered by a hyperbox of class 2 (gold) as can be seen in Fig. C.4-a. This fact explains why the classification error is highest in Scenario 1, where the model is not updated. In Scenarios 2, 3 and 4 the model is evolving over time and hyperboxes are adjusted as shown in Figs. C.4-(b,c,d), to capture new data points.

A direct consequence of scenarios without adaptive preprocessing (1 and 2) can be seen in Figs. C.4-(a,b) where new points lie outside the $[0, 1]$ range and therefore cannot be classified by the predictive model since it only operates within the $[0, 1]$ range. Fig. C.5-c shows that the number of off-limit samples is increasing linearly over time because the data keeps drifting and the input space is not rescaled.

The adaptive normalisation carried out in Scenarios 3 and 4 guarantees that all the points are within the $[0, 1]$ range (see Figs. C.4-c,d). Although the model of Scenario 3 has classified correctly most of the samples, it is actually inconsistent. That is, the rescal-

ing factor is not propagated to the model and therefore its representation is incorrect. Such inconsistency is however not reflected in the classification results (see Fig. C.5-a) and hence might be impossible to detect. In contrast, the model representation in Scenario 4 is correct because the hyperboxes have been rescaled to the new input space (see Fig. C.4-d).

The classification errors in Fig. C.5-a show that adapting the predictive system leads to considerably better results than in the static approach. Also the maximum membership degree is decreasing over time in Scenarios 1 and 2 (see Fig. C.5-b) which indicates a lower classification confidence. The lack of forgetting mechanism for GFMM method results in increase of model complexity over time, manifesting itself in growing number of hyperboxes (see Fig. C.5-d) and thus contributing to the model becoming more difficult to understand and interpret.

C.4.4 Results for real data streams

In general, Scenario 4 has a lower classification error in the three datasets (see Figs. C.6, C.7, and C.8). The classification error of Scenario 4 is considerably lower in the periods where there are new samples out of the min-max limits than in scenarios without adaptive preprocessing (see Figs. C.6, C.7, and C.8).

As observed in the synthetic dataset, there seems to be a negative correlation between the classification error and the maximum membership degree. Figs. C.6 C.7, and C.8 show that propagating the changes from the preprocessing methods to the model for adapting the hyperboxes increases the maximum membership degree and therefore the classification confidence. On the other hand, the fact of learning new concepts that cannot be classified by existing hyperboxes causes the creation of new hyperboxes that increases model complexity in Scenario 4 as shown in Figs. C.6, C.7, and C.8.

It is worth mentioning that the execution of Scenario 3 has failed in all three real datasets. That is, the lack of change propagation causes an inconsistency between the input data and the model when the number of principal components varies over time (see Fig. C.9), leading to a runtime execution error. In such case it is easy to detect that something is going wrong. However, if the number of principal components is constant but the loading vectors actually change, there is still an inconsistency between the input data and the model in Scenario 3 that will be more difficult to detect because it will not produce any software exception.

Dataset	Scenario #1	Scenario #2	Scenario #3	Scenario #4
SYN	69.25 (40.25)	43.75 (40.25)	4.75 (0)	5.00 (0)
ELEC	49.50 (33.50)	43.50 (33.50)	-	16.75 (0)
COVERTYPE	47.50 (15.50)	49.00 (15.50)	-	18.50 (0)
GAS	61.75 (51.50)	56.50 (51.50)	-	7.50 (0)

Table C.2: *Accumulated classification error (%) for all datasets. Values in brackets denote contribution of offlimits samples to the error.*

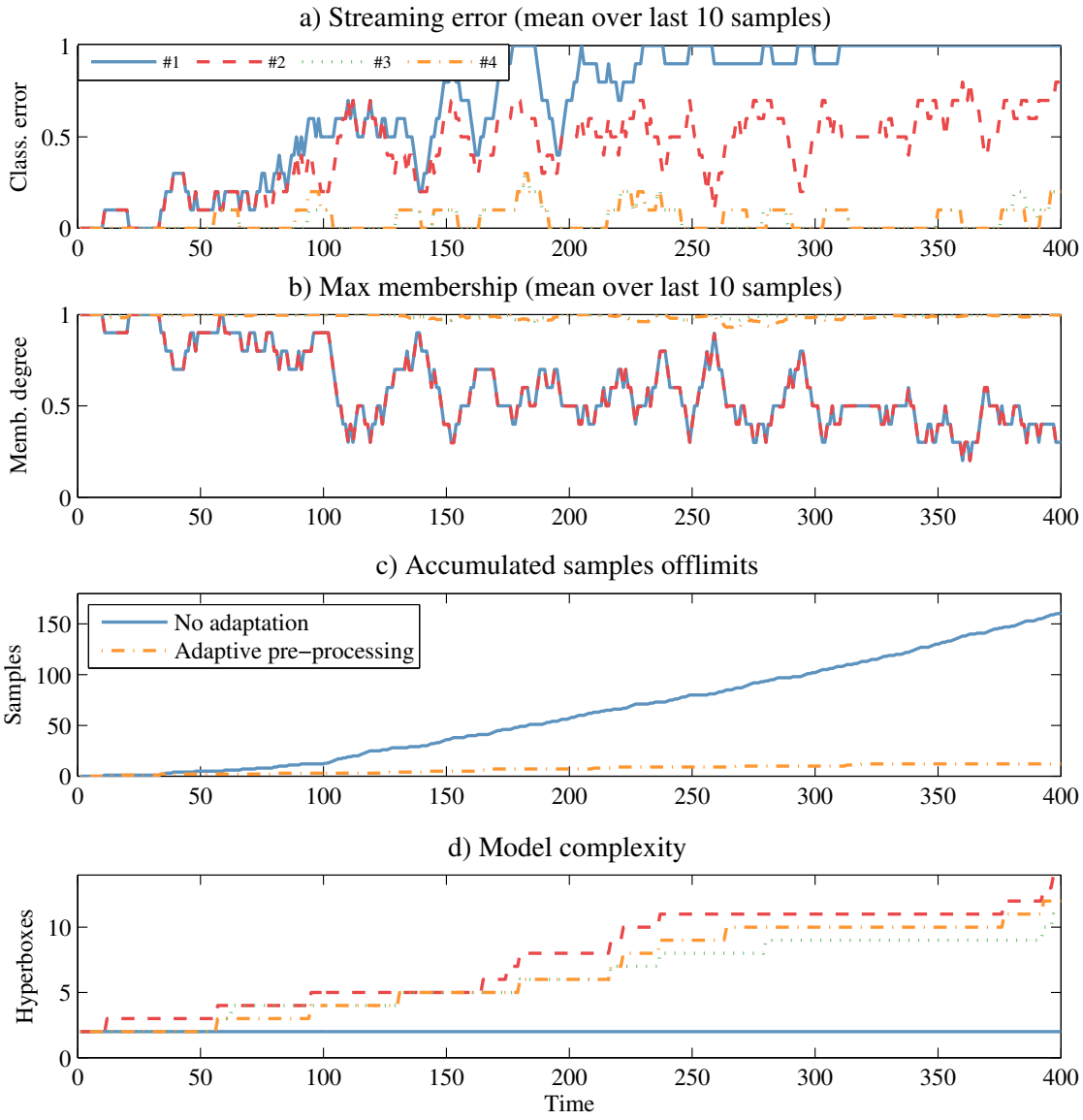


Figure C.5: *Results for SYN dataset*

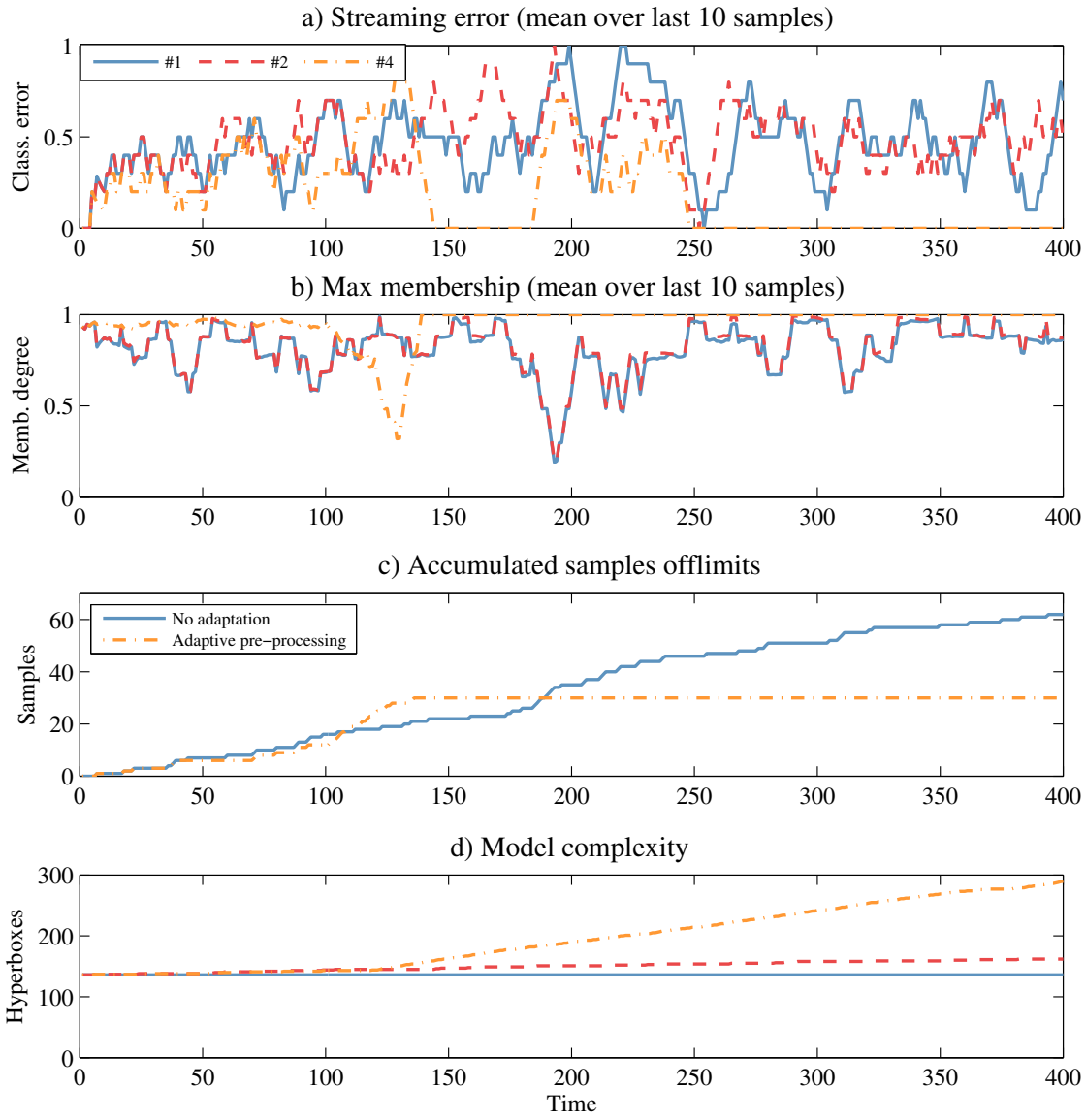


Figure C.6: *Results for ELEC dataset*

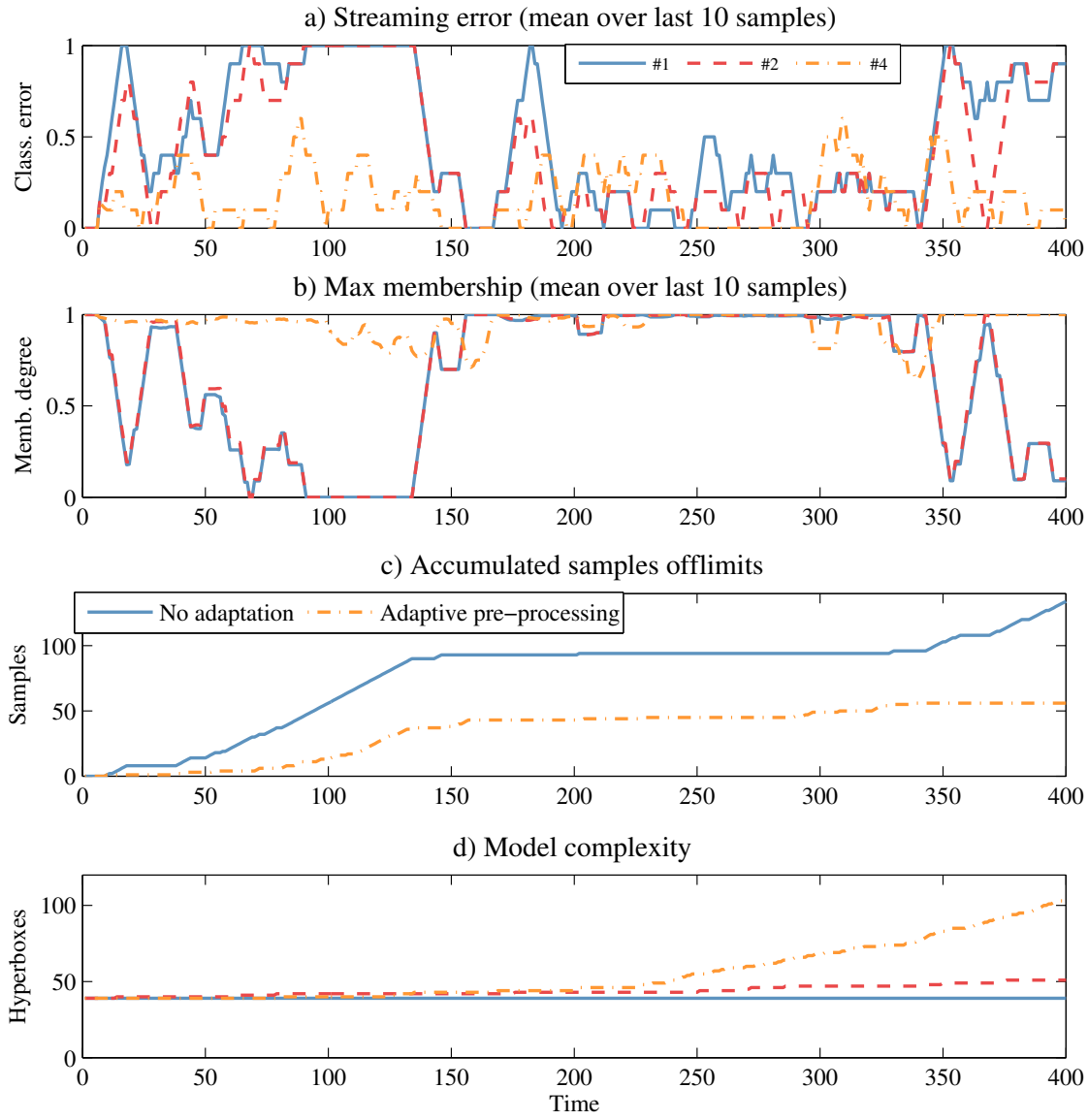


Figure C.7: *Results for COVERTYPE dataset*

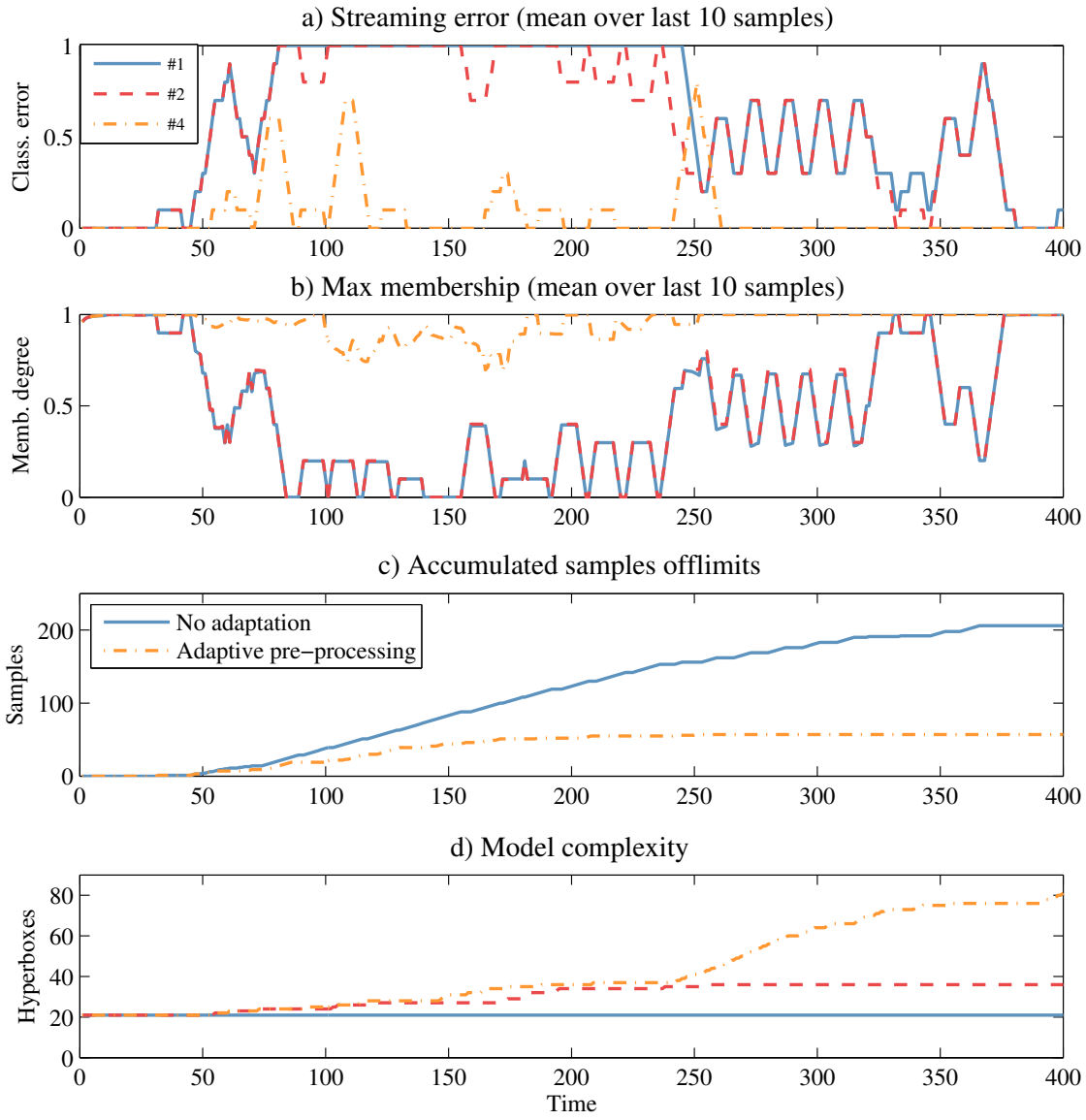


Figure C.8: *Results for GAS dataset*

C.5 Conclusions

This study demonstrated that adapting individual components of an MCPS must be carried out very carefully taking into account the impact of a change on the rest of the system. In order to formalise the process of adapting components in the correct order, the concept of change propagation has been introduced. An illustrative example in which an MCPS consisting of three preprocessing methods and one classifier has been evaluated in four scenarios with different datasets.

The need of performing adaptive preprocessing is clear in all the tested datasets. The

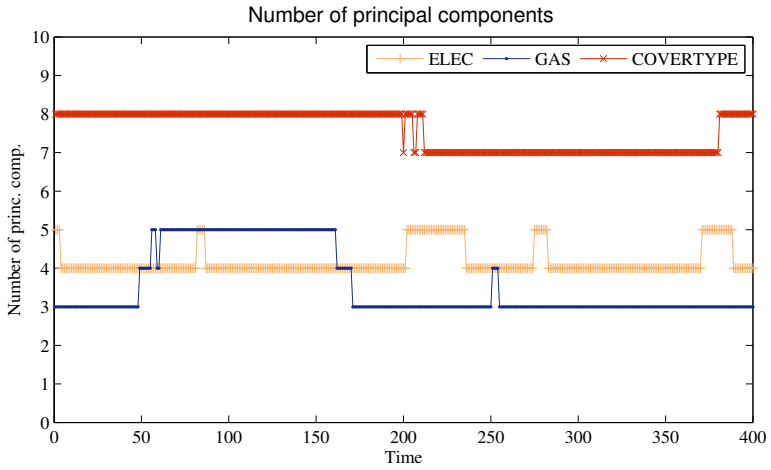


Figure C.9: *Number of principal components of ELEC, COVERTYPE and GAS testing data.*

percentage of unclassified samples in scenarios without adaptive preprocessing varies between 15.5% and 51.5% (see Table C.2). Consequently, the deterioration of system performance is considerable. A simple adaptive min-max normalisation method was shown to reduce the classification error when coupled with a classifier that requires such adaptation.

Modifying the input space requires however updating of model parameters. Otherwise, model representation can be inconsistent with the input data. Such inconsistency can even produce a system failure. To illustrate such adaptation, the GFMM method has been extended to react to such changes caused by the adaptive preprocessing methods. Other models will require similar type of transformations (e.g. changing node values in a decision tree).

While this study represents an early stage of research and has been primarily intended to highlight a range of very important issues rather than provide comprehensive solutions, it has raised a number of questions in need of further investigation. Does a change in a component always need to be propagated? How to handle propagation in systems with nonlinear transformations? How to order the preprocessing methods to reduce the required propagation? How to reduce the cost of adaptation while maximising the benefits?

The above questions and the issues raised in this study are particularly relevant and timely in the context of how easy it is to compose an MCPS in popular data analysis tools like KNIME, RapidMiner or SAS. All of these tools facilitate graphical composition of data transformation flows by linking data transformation blocks. While such approach provides a tremendous opportunity for non-expert users, as illustrated in this study, it poses potential issues when MCPSs composed in such a way were to be deployed in non-stationary environments requiring adaptation and even more so if the preprocessing steps could be easily adapted but without awareness of a need for appropriate change propagation mechanisms currently not implemented or even available.

References

Acar, U. A., Blelloch, G. E., & Harper, R. (2002). 'Adaptive functional programming'. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 2002)*, vol. 37, pp. 247–259.

Acar, U. A., Blelloch, G. E., & Vittes, J. L. (2005). 'An experimental analysis of change propagation in dynamic trees'. In *Joint Proc. 7th Worksh. Algorithm Engineering and Experiments (ALENEX 2005) and 2nd Worksh. Analytic Algorithmics and Combinatorics (ANALCO 2005)*, pp. 41–54.

Aggarwal, C. C., Hinneburg, A., & Keim, D. A. (2001). 'On the Surprising Behavior of Distance Metrics in High Dimensional Space'. In *Database Theory (ICDT 2001)*, pp. 420–434.

Al-Jubouri, B. & Gabrys, B. (2014). 'Multicriteria Approaches for Predictive Model Generation: A Comparative Experimental Study'. In *IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making*, pp. 64–71.

Allison, P. D. (2001). *Missing Data*, vol. 136. University of Pennsylvania.

Anagnostopoulos, C., Tasoulis, D., Hand, D. J., & Adams, N. M. (2008). 'Online optimization for variable selection in data streams'. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, vol. 1, pp. 132–136.

Anastasiou, N. (2013). *Automated Construction of Petri Net Performance Models from High-Precision Location Tracking Data*. Ph.D. thesis, Imperial College London.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., & de Freitas, N. (2016). 'Learning to learn by gradient descent by gradient descent'. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, pp. 3981–3989.

Arning, A., Agrawal, R., & Raghavan, P. (1996). 'A Linear Method for Deviation Detection in Large Databases'. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD96)*, pp. 164–169.

Baena-García, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., & Morales-Bueno, R. (2006). 'Early drift detection method'. In *Proceedings of the 4th International Workshop on Knowledge Discovery from Data Streams*, pp. 77–86.

Bai, Z. D. & He, X. (2004). 'A chi-square test for dimensionality with non-Gaussian data'. *Journal of Multivariate Analysis* **88**(1):109–117.

Bakirov, R., Gabrys, B., & Fay, D. (2015). 'On sequences of different adaptive mechanisms in non-stationary regression problems'. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2015)*, vol. July, pp. 1–8.

Bakirov, R., Gabrys, B., & Fay, D. (2017). 'Multiple adaptive mechanisms for data-driven soft sensors'. *Computers & Chemical Engineering* **96**:42–54.

Bartz-Beielstein, T., Chiarandini, M., Paquete, L., & Preuss, M. (2010). *Experimental Methods for the Analysis of Optimization Algorithms*. Springer Berlin Heidelberg.

Basseville, M. & Nikiforov, I. (1993). *Detection of abrupt changes: theory and application*. Prentice-Hall, Inc.

Bell, B. (1970). 'The Oldest Records of the Nile Floods'. *The Geographical Journal* **136**(4):569.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Bengio, Y. (2000). 'Gradient-based optimization of hyperparameters'. *Neural computation* **12**(8):1889–1900.

Bergstra, J., Bardenet, R., Bengio, Y., & Kegl, B. (2011). 'Algorithms for Hyper-Parameter Optimization'. In *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, pp. 1–9.

Bergstra, J. & Bengio, Y. (2012). 'Random Search for Hyper-Parameter Optimization'. *Journal of Machine Learning Research* **13**:281–305.

Berndt, D. & Clifford, J. (1994). 'Using dynamic time warping to find patterns in time series'. *Workshop on Knowledge Discovery in Databases* **398**:359–370.

Bernstein, A. & Provost, F. (2001). 'An intelligent assistant for the knowledge discovery process'. Tech. rep., New York University. Leonard Stern School of Bussiness.

Berriman, G. B., Deelman, E., Good, J., Jacob, J. C., Katz, D. S., Laity, A. C., Prince, T. A., Singh, G., & Su, M.-H. (2007). *Workflows for e-Science: Scientific Workflows for Grids*, chap. Generating complex astronomy workflows, pp. 19–38. Springer London.

Blackard, J. A. & Dean, D. J. (1999). 'Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables'. *Computers and Electronics in Agriculture* **24**(3):131–151.

Bloch, G., Ouladsine, M., & Thomas, P. (1995). 'On-line fault diagnosis of dynamic systems via robust parameter estimation'. *Control Engineering Practice* **3**(12):1709–1717.

Blum, R., Kassam, S., & Poor, H. (1997). 'Distributed detection with multiple sensors I. Advanced topics'. *Proceedings of the IEEE* **85**(1):64–79.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). 'End to End Learning for Self-Driving Cars'. *arXiv:1604* pp. 1–9.

Bouchachia, A., Gabrys, B., & Sahel, Z. (2007). 'Overview of Some Incremental Learning Algorithms'. In *2007 IEEE International Fuzzy Systems Conference*, pp. 1–6.

Brochu, E., Cora, V. M., & de Freitas, N. (2010). 'A Tutorial on Bayesian Optimization of Expensive Cost Functions with Application to Active User Modeling and Hierarchical Reinforcement Learning'. Tech. rep., University of British Columbia, Department of Computer Science.

Brubaker, S. (2009). 'Robust PCA and clustering in noisy mixtures'. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009)*, pp. 1078–1087. Society for Industrial and Applied Mathematics.

Bruce, L. & Koger, C. (2002). 'Dimensionality reduction of hyperspectral data using discrete wavelet transform feature extraction'. *IEEE Transactions on Geoscience and Remote Sensing* **40**(10):2331–2338.

Budka, M., Eastwood, M., Gabrys, B., Kadlec, P., Martin Salvador, M., Schwan, S., Tsakonas, A., & Žliobaitė, I. (2014). 'From Sensor Readings to Predictions: On the Process of Developing Practical Soft Sensors'. In *Advances in Intelligent Data Analysis XIII*, vol. 8819, pp. 49–60. Springer International Publishing.

Budka, M. & Gabrys, B. (2010a). 'Correntropy-based density-preserving data sampling as an alternative to standard cross-validation'. *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2010)* pp. 1–8.

Budka, M. & Gabrys, B. (2010b). 'Ridge regression ensemble for toxicity prediction'. *Procedia Computer Science* **1**(1):193–201.

Budka, M. & Gabrys, B. (2013). 'Density-Preserving Sampling: Robust and Efficient Alternative to Cross-Validation for Error Estimation'. *IEEE Transactions on Neural Networks and Learning Systems* **24**(1):22–34.

Budka, M., Gabrys, B., & Ravagnan, E. (2010). 'Robust predictive modelling of water pollution using biomarker data'. *Water Research* **44**(10):3294–3308.

Calì, A., Lembo, D., & Rosati, R. (2005). 'A comprehensive semantic framework for data integration systems'. *Journal of Applied Logic* **3**(2):308–328.

Cao, S. & Rhinehart, R. (1995). 'An efficient method for on-line identification of steady state'. *Journal of Process Control* **5**(6):363–374.

Castells, M. (2009). *The Information Age. Economy, Society and Culture. Volume I.* Wiley Blackwell, 2nd edn.

Chaloner, K. & Brant, R. (1988). 'A Bayesian approach to outlier detection and residual analysis'. *Biometrika* **75**(4):651–659.

Chandola, V., Banerjee, A., & Kumar, V. (2009). 'Anomaly detection: A survey'. *ACM Computing Surveys* **41**(3):1–58.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). 'SMOTE: Synthetic minority over-sampling technique'. *Journal of Artificial Intelligence Research* **16**:321–357.

Cheng, W., Kasneci, G., Graepel, T., Stern, D., & Herbrich, R. (2011). 'Automated feature generation from structured knowledge'. In *Proceedings of the 20th ACM international conference on information and knowledge management (CIKM 2011)*, p. 1395. ACM Press.

Chéruy, A. (1997). 'Software sensors in bioprocess engineering'. *Journal of Biotechnology* **52**(3):193–199.

Davies, L. & Gather, U. (1993). 'The Identification of Multiple Outliers'. *Journal of the American Statistical Association* **88**(423):782.

Dawyndt, P., Vancanneyt, M., Meyer, H. D., & Swings, J. (2005). 'Knowledge Accumulation and Resolution of Data Inconsistencies during the Integration of Microbial Information Sources'. *Knowledge Creation Diffusion Utilization* **17**(8):1111–1126.

De Assis, A. J. & Maciel Filho, R. (2000). 'Soft sensors development for on-line bioreactor state estimation'. *Computers and Chemical Engineering* **24**(2-7):1099–1103.

De Silva, A. M., Noorian, F., Davis, R. I. A., & Leong, P. H. W. (2013). 'A hybrid feature selection and generation algorithm for electricity load prediction using Grammatical evolution'. In *12th International Conference on Machine Learning and Applications (ICMLA 2013)*, vol. 2, pp. 211–217.

Deisenroth, M. P., Rasmussen, C. E., & Peters, J. (2009). 'Gaussian process dynamic programming'. *Neurocomputing* **72**(7-9):1508–1524.

Deng, L., Tur, G., He, X., & Hakkani-Tur, D. (2012). 'Use of kernel deep convex networks and end-to-end learning for spoken language understanding'. *2012 IEEE Workshop on Spoken Language Technology (SLT 2012)* pp. 210–215.

Denning, D. (1987). 'An Intrusion-Detection Model'. *IEEE Transactions on Software Engineering* **SE-13**(2):222–232.

Diamantini, C., Potena, D., & Storti, E. (2009). 'KDDONTO: An ontology for discovery and composition of kdd algorithms'. In *Third Generation Data Mining: Towards Service-Oriented Knowledge Discovery (SoKD 2009)*, pp. 13–24.

DiCesare, F., Harhalakis, G., Proth, J. M., Silva, M., & Vernadat, F. B. (1993). *Practice of Petri Nets in Manufacturing*. Springer Netherlands.

Dietterich, T. G. (2000). 'Ensemble Methods in Machine Learning'. *Multiple Classifier Systems* **1857**:1–15.

Douglas, L. (2001). '3D Data Management: Controlling Data Volume, Velocity and Variety'. Tech. rep., Gartner.

Downs, J. & Vogel, E. (1993). 'A plant-wide industrial process control problem'. *Computers & Chemical Engineering* **17**(3):245–255.

Dudoit, S. & Fridlyand, J. (2002). 'A prediction-based resampling method for estimating the number of clusters in a dataset.'. *Genome biology* **3**(7):36.1–36.21.

Efron, B. & Gong, G. (1983). 'A leisurely look at the bootstrap, the jackknife, and cross-validation'. *American Statistician* **37**(1):36–48.

Eggensperger, K., Hutter, F., Hoos, H. H., & Leyton-brown, K. (2012). 'Efficient Benchmarking of Hyperparameter Optimizers via Surrogates Background : Hyperparameter Optimization'. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pp. 1114–1120.

Emamian, V., Kaveh, M., & Tewfik, A. (2000). 'Robust clustering of acoustic emission signals using the Kohonen network'. In *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, pp. 3891–3894. IEEE.

Ester, M., Kriegel, H., S, J., & Xu, X. (1996). 'A density-based algorithm for discovering clusters in large spatial databases with noise'. In *The 2nd International Conference on Knowledge Discovery and Data Mining (KDD96)*, pp. 226–231.

Euler, T., Morik, K., & Scholz, M. (2003). 'MiningMart: Sharing Successful KDD Processes'. In *Tagungsband der GI-Workshop-Woche Lehren - Lernen - Wissen - Adaptivitat (LLWA 2003)*, pp. 121–122.

Fawcett, T. & Provost, F. (1997). 'Adaptive fraud detection'. *Data mining and knowledge discovery* **1**(3):291–316.

Fayyad, U. & Piatetsky-Shapiro, G. (1996). 'From data mining to knowledge discovery in databases'. *AI Magazine* pp. 37–54.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). 'Efficient and Robust Automated Machine Learning'. *Advances in Neural Information Processing Systems 28 (NIPS 2015)* pp. 2944–2952.

Feurer, M., Springenberg, J. T., & Hutter, F. (2014). 'Using Meta-Learning to Initialize Bayesian Optimization of Hyperparameters'. In *Proceedings of the 2014 International Conference on Meta-learning and Algorithm Selection (MLAS 2014)*, pp. 3–10.

Fisher, R. A. (1936). 'The use of multiple measurements in taxonomic problems'. *Annals of Eugenics* **7**(2):179–188.

Fortuna, L., Graziani, S., Rizzo, A., & Xibilia, M. G. (2007). *Soft Sensors for Monitoring and Control of Industrial Processes*, vol. 78 of *Advances in Industrial Control*. Springer.

Fortuna, L., Graziani, S., & Xibilia, M. G. (2005). 'Soft sensors for product quality monitoring in debutanizer distillation columns'. *Control Engineering Practice* **13**:499–508.

Fortuna, L., Rizzo, A., Sinatra, M., & Xibilia, M. G. (2003). 'Soft analyzers for a sulfur recovery unit'. *Control Engineering Practice* **11**:1491–1500.

Fraver, S., Bradford, J. B., & Palik, B. J. (2011). 'Improving tree age estimates derived from increment cores: A case study of red pine'. *Forest Science* **57**(2):164–170.

Friedman, J., Hastie, T., Höfling, H., & Tibshirani, R. (2007). 'Pathwise coordinate optimization'. *The Annals of Applied Statistics* **1**(2):302–332.

Fulkerson, B., Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (1995). 'Machine Learning, Neural and Statistical Classification'. *Technometrics* **37**(4):459.

Gabrys, B. (2002). 'Neuro-fuzzy approach to processing inputs with missing values in pattern recognition problems'. *International Journal of Approximate Reasoning* **30**(3):149–179.

Gabrys, B. (2004). 'Learning hybrid neuro-fuzzy classifier models from data: to combine or not to combine?'. *Fuzzy Sets and Systems* **147**(1):39–56.

Gabrys, B. & Bargiela, A. (2000). 'General fuzzy min-max neural network for clustering and classification.'. *IEEE Transactions on Neural Networks* **11**(3):769–83.

Gama, J. (2000). 'Iterative Bayes'. *Intelligent Data Analysis* **4**:475–488.

Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). 'Learning with Drift Detection'. In *Advances in Artificial Intelligence (SBIA 2004)*, pp. 286–295.

Gama, J., Sebastiao, R., & Rodrigues, P. P. (2012). 'On evaluating stream learning algorithms'. *Machine Learning* **90**(3):317–346.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). 'A survey on concept drift adaptation'. *ACM Computing Surveys* **46**(4):1–37.

Giffin, M., de Weck, O., Bounova, G., Keller, R., Eckert, C., & Clarkson, P. J. (2009). 'Change Propagation Analysis in Complex Technical Systems'. *Journal of Mechanical Design* **131**:1–14.

Giraud-Carrier, C. (2005). 'The Data Mining Advisor: Meta-learning at the Service of Practitioners'. In *4th International Conference on Machine Learning and Applications (ICMLA 2005)*, pp. 113–119. IEEE.

Guo, X. C., Yang, J. H., Wu, C. G., Wang, C. Y., & Liang, Y. C. (2008). 'A novel LS-SVMs hyper-parameter selection based on particle swarm optimization'. In *Neurocomputing*, vol. 71, pp. 3211–3215.

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). 'Gene Selection for Cancer Classification using Support Vector Machines'. *Machine Learning* **46**(4):389–422.

Han, J., Kamber, M., & Pei, J. (2011). *Data mining: concepts and techniques*. Elsevier, 3 edn.

Hardy, S. & Robillard, P. N. (2004). 'Modeling and simulation of molecular biology systems using petri nets: modeling goals of various approaches'. *Journal of Bioinformatics and Computational Biology* **02**(04):619–637.

Harries, M. (1999). 'Splice-2 comparative evaluation: Electricity pricing'. Tech. rep., The University of South Wales.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*, vol. 18. Springer-Verlag New York, 2 edn.

He, Z., Deng, S., & Xu, X. (2005). 'An optimization model for outlier detection in categorical data'. In *Proceedings of the 2005 international conference on Advances in Intelligent Computing - Volume Part I*, pp. 400–409. Springer-Verlag.

He, Z., Xu, X., & Deng, S. (2003). 'Discovering cluster-based local outliers'. *Pattern Recognition Letters* **24**(9-10):1641–1650.

Hennig, P. & Schuler, C. J. (2012). 'Entropy Search for Information-Efficient Global Optimization'. *Machine Learning Research* **13**(1999):1809–1837.

Hernández, M. A. & Stolfo, S. J. (1998). 'Real-world data is dirty: Data cleansing and the merge/purge problem'. *Data Mining and Knowledge Discovery* **2**(1):9–37.

Hinneburg, A., Aggarwal, C. C., & Keim, D. A. (2000). 'What is the Nearest Neighbor in High Dimensional Spaces?'. In *Proceedings of the 26th VLDB Conference*, pp. 506–515.

Hoos, H., Ca, U. B. C., Leyton-Brown, K., & Hutter, F. (2014). 'An Efficient Approach for Assessing Hyperparameter Importance'. *The 31st International Conference on Machine Learning (ICML 2014)* **32**(1):754–762.

Huang, Z. Z., Chen, H. H., Hsu, C.-J. C.-J., Chen, W.-H. W.-H., & Wu, S. S. (2004). 'Credit rating analysis with support vector machines and neural networks: A market comparative study'. *Decision Support Systems* **37**(4):543–558.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). 'Sequential Model-Based Optimization for General Algorithm Configuration'. *Learning and Intelligent Optimization* **6683 LNCS**:507–523.

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Murphy, K. P. (2009). 'An Experimental Investigation of Model-Based Parameter Optimisation: SPO and Beyond'. *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* pp. 271–278.

Islamaj, R., Getoor, L., & Wilbur, W. J. (2006). 'A Feature Generation Algorithm for Sequences with Application to Splice-Site Prediction'. In *Knowledge Discovery in Databases: PKDD 2006*, pp. 553–560.

J. A. Little, R. & B. Rubin, D. (2002). *Statistical Analysis with Missing Data, 2nd Edition*. Wiley.

Joe Qin, S. (1998). 'Recursive PLS algorithms for adaptive data modeling'. *Computers & Chemical Engineering* **22**(4-5):503–514.

Jolliffe, I. T. (2002). *Principal Component Analysis, Second Edition*. Springer.

Jurney, R. (2013). *Agile Data Science*. O'Reilly Media.

Kabán, A. (2011). 'On the distance concentration awareness of certain data reduction techniques'. *Pattern Recognition* **44**(2):265–277.

Kadlec, P. (2009). *On robust and adaptive soft sensors*. Ph.D. thesis, Bournemouth University.

Kadlec, P. & Gabrys, B. (2009). 'Architecture for development of adaptive on-line prediction models'. *Memetic Computing* **1**(4):241–269.

Kadlec, P. & Gabrys, B. (2011). 'Local learning-based adaptive soft sensor for catalyst activation prediction'. *AIChE Journal* **57**(5):1288–1301.

Kadlec, P., Gabrys, B., & Strandt, S. (2009). 'Data-driven Soft Sensors in the process industry'. *Computers & Chemical Engineering* **33**(4):795–814.

Kadlec, P., Grbić, R., & Gabrys, B. (2011). 'Review of adaptation mechanisms for data-driven soft sensors'. *Computers & Chemical Engineering* **35**(1):1–24.

Kalapanidas, E., Avouris, N., Craciun, M., & Neagu, D. (2003). 'Machine Learning Algorithms: A study on noise sensitivity'. *1st Balcan Conference in Informatics* pp. 356–365.

Kambhatla, N. & Leen, T. K. (1997). 'Dimension Reduction by Local Principal Component Analysis'. *Neural Computation* **9**(7):1493–1516.

Kämpjärvi, P., Sourander, M., Komulainen, T., Vatanski, N., Nikus, M., & Jämsä-Jounela, S.-L. (2008). 'Fault detection and isolation of an on-line analyzer for an ethylene cracking process'. *Control Engineering Practice* **16**(1):1–13.

Keet, C. M., Ławrynowicz, A., D'Amato, C., Kalousis, A., Nguyen, P., Palma, R., Stevens, R., & Hilario, M. (2015). 'The Data Mining OPtimization Ontology'. *Web Semantics: Science, Services and Agents on the World Wide Web* **32**:43–53.

Keogh, E., Chakrabarti, K., Pazzani, M., & Mehrotra, S. (2001). 'Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases'. *Knowledge and Information Systems* **3**(3):263–286.

Kietz, J.-U., Serban, F., Bernstein, A., & Fischer, S. (2009). 'Towards cooperative planning of data mining workflows'. In *Third Generation Data Mining: Towards Service-Oriented Knowledge Discovery (SoKD 2009)*.

Kim, S., Cho, N. W., Lee, Y. J., Kang, S., Kim, T., Hwang, H., & Mun, D. (2010). 'Application of density-based outlier detection to database activity monitoring'. *Information Systems Frontiers* .

Klinkenberg, R. (2004). 'Learning drifting concepts: Example selection vs. example weighting'. *Intelligent Data Analysis* **8**:281–300.

Kohavi, R. (1995). 'A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection'. *International Joint Conference on Artificial Intelligence* **14**(12):1137–1143.

Krempl, G., Žliobaitė, I., Brzezinski, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M., & Stefanowski, J. (2014). 'Open challenges for data stream mining research'. *SIGKDD Explorations* **16**(1):1–10.

Kuo, R. (2001). 'A sales forecasting system based on fuzzy neural network with initial weights generated by genetic algorithm'. *European Journal of Operational Research* **129**(3):496–517.

Kushner, H. J. (1964). 'A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise'. *Journal of Basic Engineering* **86**(1):97.

Lai, T. (1995). 'Sequential changepoint detection in quality control and dynamical systems'. *Journal of the Royal Statistical Society. Series B (Methodological)* **57**(4):613–658.

Lai, T. L. & Robbins, H. (1985). 'Asymptotically efficient adaptive allocation rules'. *Advances in Applied Mathematics* **6**(1):4–22.

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). 'Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations'. *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)* **2008**:1–8.

Leite, R., Brazdil, P., & Vanschoren, J. (2012). 'Selecting classification algorithms with active testing'. In *8th International Conference (MLDM 2012)*, vol. 7376, pp. 117–131.

Lemke, C., Budka, M., & Gabrys, B. (2015). 'Metalearning: a survey of trends and technologies'. *Artificial Intelligence Review* **44**(1):117–130.

Lemke, C. & Gabrys, B. (2010). 'Meta-learning for time series forecasting and forecast combination'. *Neurocomputing* **73**(10-12):2006–2016.

Lessmann, S., Stahlbock, R., & Crone, S. (2005). 'Optimizing hyperparameters of support vector machines by genetic algorithms'. *Proceedings of the 2005 International Conference on Artificial Intelligence (ICAI 2005)* **1**:74–80.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2016). 'Efficient Hyperparameter Optimization and Infinitely Many Armed Bandits'. *arXiv preprint* .

Li, W., Yue, H., Valle-Cervantes, S., & Qin, S. (2000). 'Recursive PCA for adaptive process monitoring'. *Journal of Process Control* **10**(5):471–486.

Lin, B., Recke, B., Knudsen, J. K., & Jørgensen, S. B. (2007). 'A systematic approach for soft sensor development'. *Computers & Chemical Engineering* **31**(5-6):419–425.

Lindner, G. & Studer, R. (1999). 'AST: Support for Algorithm Selection with a CBR Approach'. In *Principles of Data Mining and Knowledge Discovery (PKDD'99)*, pp. 418–423. Springer Berlin Heidelberg.

Linoff, G. S. & Berry, M. J. A. (2011). *Data mining techniques: for marketing, sales, and customer relationship management*. Wiley.

Littlestone, N. & Warmuth, M. (1994). 'The Weighted Majority Algorithm'. *Information and Computation* **108**(2):212–261.

Liu, X. Y., Wu, J., & Zhou, Z. H. (2006). 'Exploratory under-sampling for class-imbalance learning'. In *IEEE International Conference on Data Mining (ICDM 2006)*, pp. 965–969.

Maedche, A., Hotho, A., & Wiese, M. (2000). 'Enhancing Preprocessing in Data-Intensive Domains using Online-Analytical Processing'. In *2nd International Conference on Data Warehousing and Knowledge (DaWaK 2000)*, pp. 258–264. Springer Berlin Heidelberg.

Marban, O., Mariscal, G., & Segovi, J. (2009). 'A Data Mining & Knowledge Discovery Process Model'. In *Data Mining and Knowledge Discovery in Real Life Applications*, no. February, pp. 1–17. I-Tech Education and Publishing.

Markou, M. & Singh, S. (2003a). 'Novelty detection: a reviewpart 1: statistical approaches'. *Signal Processing* **83**(12):2481–2497.

Markou, M. & Singh, S. (2003b). 'Novelty detection: a reviewpart 2: neural network based approaches'. *Signal Processing* **83**(12):2499–2521.

Marlin, B. M. (2008). *Missing Data Problems in Machine Learning*. Ph.D. thesis, University of Toronto.

Martin Salvador, M., Budka, M., & Gabrys, B. (2016a). 'Automatic composition and optimisation of multicomponent predictive systems'. *ArXiv e-prints* .

Martin Salvador, M., Budka, M., & Gabrys, B. (2016b). 'Effects of change propagation resulting from adaptive preprocessing in multicomponent predictive systems'. In *20th International Conference on Knowledge Based and Intelligent Information and Engineering Systems. KES-2016*, vol. 96, pp. 713–722.

Martin Salvador, M., Budka, M., & Gabrys, B. (2016c). 'Towards Automatic Composition of Multicomponent Predictive Systems'. In *11th International Conference on Hybrid Artificial Intelligence Systems*, pp. 27–39. Springer International Publishing.

Martin Salvador, M., Gabrys, B., & Žliobaitė, I. (2014). 'Online Detection of Shutdown Periods in Chemical Plants: A Case Study'. In *18th International Conference in Knowledge Based and Intelligent Information and Engineering Systems*, vol. 35, pp. 580–588. Elsevier.

McQuarrie, A. D. R. & Tsai, C.-L. (1998). *Regression and Time Series Model Selection*. World Scientific.

Mei, Y. (2010). 'Efficient scalable schemes for monitoring a large number of data streams'. *Biometrika* **97**(2):419–433.

Mei, Y. (2011). 'Final Report for the Project Robust Rapid Change-Point Detection in Multi-Sensor Data Fusion and Behavior Research'. Tech. rep., Georgia Institute of Technology.

Messaoud, I., El Abed, H., Märgner, V., & Amiri, H. (2011). 'A design of a preprocessing framework for large database of historical documents'. In *Proceedings of the 2011 Workshop on Historical Document Imaging and Processing*, pp. 177–183. ACM Press.

Mladenić, D. (2006). 'Feature selection for dimensionality reduction'. In *Lecture Notes in Computer Science*, pp. 84–102.

Mockus, J., Tiesis, V., & Zilinskas, A. (1978). 'The application of Bayesian methods for seeking the extremum'. *Towards Global Optimization* **2**:117–129.

Munson, M. A. (2012). 'A study on the importance of and time spent on different modeling steps'. *ACM SIGKDD Explorations Newsletter* **13**(2):65–71.

Murata, T. (1989). 'Petri nets: properties, analysis and applications'. *Proceedings of the IEEE* **77**(4):541–580.

Neches, R., Fikes, R. E., Finin, T., Gruber, T., Patil, R., Senator, T., & Swartout, W. R. (1991). 'Enabling technology for knowledge sharing'. *AI magazine* **12**(3):36.

Nettleton, D. F., Orriols-Puig, A., & Fornells, A. (2010). 'A study of the effect of different types of noise on the precision of supervised learning techniques'. *Artificial Intelligence Review* **33**(4):275–306.

Ogasawara, E., Martinez, L. C., de Oliveira, D., Zimbrao, G., Pap, G. L., & Mattoso, M. (2010). 'Adaptive Normalization: A novel data normalization approach for non-stationary time series'. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE.

Östermark, R. (2009). 'A fuzzy vector valued KNN-algorithm for automatic outlier detection'. *Applied Soft Computing* **9**(4):1263–1272.

Page, E. S. (1954). 'Continuous Inspection Schemes'. *Biometrika* **41**(1/2):100–115.

Pani, A. K. & Mohanta, H. K. (2011). 'A Survey of Data Treatment Techniques for Soft Sensor Design'. *Chemical Product and Process Modeling* **6**(1):1–21.

Pankratius, V. & Stucky, W. (2005). 'A Formal Foundation for Workflow Composition, Workflow View Definition, and Workflow Normalization based on Petri Nets'. In *2nd Asia-Pacific Conference on Conceptual Modelling (APCCM 2005)*, pp. 79–88.

Panov, P., Deroski, S., & Soldatova, L. (2008). 'OntoDM: An Ontology of Data Mining'. In *2008 IEEE International Conference on Data Mining Workshops*, pp. 752–760.

Parra, L., Deco, G., & Miesbach, S. (1996). 'Statistical Independence and Novelty Detection with Information Preserving Nonlinear Maps'. *Neural Computation* **8**(2):260–269.

Parzen, E. (1962). 'On Estimation of a Probability Density Function and Mode'. *The Annals of Mathematical Statistics* **33**(3):1065–1076.

Pearson, K. (1901). 'Online and planes of closest fit to systems of points in space'. *Philosophical Magazine* **2**(6):559–572.

Pearson, R. K. (2005). *Mining imperfect data*. Society for Industrial and Applied Mechanics.

Peng, R. D. (2011). 'Reproducible Research in Computational Science'. *Science* **334**(6060):1226–1227.

Petri, C. A. (1962). *Kommunikation mit Automaten*. Ph.D. thesis, Universität Hamburg.

Ping, L., Hao, H., & Jian, L. (2004). 'On 1-soundness and Soundness of Workflow Nets'. In *3rd Workshop on Modelling of Objects, Components, and Agents (MOCA 2004)*, pp. 21–36.

Polunchenko, A. & Tartakovsky, A. (2011). 'State-of-the-Art in Sequential Change-Point Detection'. *Methodology and Computing in Applied Probability* **14**(3):649–684.

Prasad, V., Schley, M., Russo, L. P., & Wayne Bequette, B. (2002). 'Product property and production rate control of styrene polymerization'. *Journal of Process Control* **12**(3):353–372.

Pyle, D. (1999). *Data preparation for data mining*. Morgan Kaufmann.

Qin, S. (1997). 'Chapter 8 - Neural Networks for Intelligent Sensors and ControlPractical Issues and Some Solutions'. In Omidvar, O. & Elliott, D. (eds.), *Neural Syst. Control*, pp. 213–234. Academic Press.

Qu, Y., Adam, B., Thornquist, M., Potter, J. D., Thompson, M. L., Yasui, Y., Davis, J., Schellhammer, P., Cazares, L., Clements, M., Wright, G., & Feng, Z. (2003). 'Data Reduction Using a Discrete Wavelet Transform in Discriminant Analysis of Very High Dimensionality Data'. *Biometrics* **59**(1):143–151.

Quinlan, J. (1986). 'Induction of decision trees'. *Machine learning* pp. 81–106.

Rajagopal, R., Nguyen, X., Ergen, S., & Varaiya, P. (2008). 'Distributed Online Si-

multaneous Fault Detection for Multiple Sensors'. In *International Conference on Information Processing in Sensor Networks (IPSN 2008)*, pp. 133–144. IEEE.

Reis, M. S. & Saraiva, P. M. (2004). 'A comparative study of linear regression methods in noisy environments'. *Journal of Chemometrics* **18**(12):526–536.

Rice, J. R. (1976). 'The Algorithm Selection Problem'. *Advances in Computers* **15**(C):65–118.

Rokach, L. (2010). 'Ensemble-based classifiers'. *Artificial Intelligence Review* **33**(1-2):1–39.

Romero, I. (2011). 'PCA and ICA applied to noise reduction in multi-lead ECG'. In *2011 Computing in Cardiology*, pp. 613–616.

Rubin, D. B. (1976). 'Inference and Missing Data'. *Biometrika* **63**(3):581.

Ruta, D. & Gabrys, B. (2010). 'Neural Network Ensembles for Time Series Prediction'. *2007 International Joint Conference on Neural Networks* pp. 1204–1209.

Sadiq, S., Orlowska, M., Sadiq, W., & Foulger, C. (2004). 'Data Flow and Validation in Workflow Modelling'. In *Proceedings of the 15th Australasian database conference (ADC 2004)*, vol. 27, pp. 207–214.

Schlimmer, J. C. (1987). *Concept Acquisition Through Representational Adjustment*. Ph.D. thesis, University of California, Irvine.

Schwan, S. (2011). 'CCP Case Study'. Tech. rep., INFER.eu.

Serban, F., Bernstein, A., & Fischer, S. (2012). 'Designing KDD-Workflows via HTN-Planning for Intelligent Discovery Assistance'. In *Proceedings of the International Workshop on Planning to Learn 2012*, pp. 10–17.

Serban, F., Vanschoren, J., Kietz, J.-U., & Bernstein, A. (2013). 'A survey of intelligent assistants for data analysis'. *ACM Computing Surveys* **45**(3):1–35.

Seron, M., De Dona, J., & Olaru, S. (2012). 'Fault Tolerant Control Allowing Sensor Healthy-to-Faulty and Faulty-to-Healthy Transitions'. *IEEE Transactions on Automatic Control* **57**(7):1657–1669.

Shade, A. & Teal, T. K. (2015). 'Computing Workflows for Biologists: A Roadmap'. *PLoS Biology* **13**(11):1–10.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & de Freitas, N. (2016). 'Taking the Human Out of the Loop: A Review of Bayesian Optimization'. *Proceedings of the IEEE* **104**(1):148–175.

Sharmin, R., Sundararaj, U., Shah, S., Vande Griend, L., & Sun, Y. (2006). 'Inferential sensors for estimation of polymer quality parameters: Industrial application of a PLS-based soft sensor for a LDPE plant'. *Chemical Engineering Science* **61**(19):6372–6384.

Shearer, C. (2000). 'The CRISP-DM model: the new blueprint for data mining'. *Journal of Data Warehousing* **5**(4):13–22.

Shewhart, W. (1931). *Economic Control of Quality of Manufactures Product*. Asq Press.

Sleeman, D., Rissakis, M., Craw, S., Graner, N., & Sharma, S. (1995). 'Consultant-2: pre- and post-processing of Machine Learning applications'. *International Journal of Human-Computer Studies* **43**(1):43–63.

Slišković, D., Grbić, R., & Hocenski, Ž. (2011). 'Online Data Preprocessing in the Adaptive Process Model Building based on Plant Data'. *Tehnicki Vjesnik* **18**(1):41–50.

Snoek, J., Larochelle, H., & Adams, R. (2012). 'Practical Bayesian Optimization of Machine Learning Algorithms.'. *Advances in Neural Information Processing Systems 25 (NIPS 2012)* pp. 2960–2968.

Souza, F. & Araujo, R. (2014). 'Online Mixture of Univariate Linear Regression Models for Adaptive Soft Sensors'. *IEEE Transactions on Industrial Informatics* **10**(2):937–945.

Sparks, E. R., Talwalkar, A., Haas, D., Franklin, M. J., Jordan, M. I., & Kraska, T. (2015). 'Automating model search for large scale machine learning'. In *Proceedings of the 6th ACM Symposium on Cloud Computing (SoCC 2015)*, pp. 368–380. ACM Press.

Stahl, F., Medhat Gaber, M., & Martin Salvador, M. (2012). 'eRules: A Modular Adaptive Classification Rule Learning Algorithm for Data Streams'. In *Research and Development in Intelligent Systems XXIX*, pp. 65–78. Springer.

Strackeljan, J. (2006). 'NiSIS Competition 2006 - Soft Sensor for the adaptive Catalyst Monitoring of a MultiTube Reactor'. Tech. rep., Universität Magdeburg.

Sun, Z.-L., Choi, T.-M., Au, K.-F., & Yu, Y. (2008). 'Sales forecasting using extreme learning machine with applications in fashion retailing'. *Decision Support Systems* **46**(1):411–419.

Suyama, A. & Yamaguchi, T. (1998). 'Specifying and learning inductive learning systems using ontologies'. In *AAAI Workshop on the Methodology of Applying Machine Learning: Problem Definition, Task Decomposition, and Technique Selection*, pp. 29–36.

Swersky, K., Snoek, J., & Adams, R. P. (2013). 'Multi-Task Bayesian Optimization'. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pp. 2004–2012.

Tartakovsky, A., Rozovskii, B., Blažek, R., & Kim, H. (2006). 'Detection of intrusions in information systems by sequential change-point methods'. *Statistical Methodology* **3**(3):252–293.

Tartakovsky, A. & Veeravalli, V. (2008). 'Asymptotically Optimal Quickest Change Detection in Distributed Sensor Systems'. *Sequential Analysis* **27**(4):441–475.

Teichmann, E., Demir, E., & Chaussalet, T. (2010). 'Data preparation for clinical data mining to identify patients at risk of readmission'. In *2010 IEEE 23rd International Symposium on Computer-Based Medical Systems (CBMS)*, pp. 184–189. IEEE.

Tennant, G. (2001). *SIX SIGMA: SPC and TQM in Manufacturing and Services*. Gower Publishing.

Thai-Nghe, N., Gantner, Z., & Schmidt-Thieme, L. (2010). 'Cost-sensitive learning methods for imbalanced data'. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, vol. 3025, pp. 1–8. IEEE.

Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). 'Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms'. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2013)*, pp. 847–855. ACM Press.

Tsakonas, A. & Gabrys, B. (2012). 'GRADIENT: Grammar-driven genetic programming framework for building multi-component, hierarchical predictive systems'. *Expert Systems with Applications* **39**(18):13253–13266.

Tsakonas, A. & Gabrys, B. (2013). 'A fuzzy evolutionary framework for combining ensembles'. *Applied Soft Computing Journal* **13**(4):1800–1812.

Tso, G. K. F. & Yau, K. K. W. (2007). 'Predicting electricity energy consumption: A comparison of regression analysis, decision tree and neural networks'. *Energy* **32**(9):1761–1768.

van der Aalst, W. M. P. (1997). 'Verification of workflow nets'. In *Application and Theory of Petri Nets*, pp. 407–426.

van der Aalst, W. M. P. (1998a). 'The Application of Petri Nets To Workflow Management'. *Journal of Circuits, Systems and Computers* **08**(01):21–66.

van der Aalst, W. M. P. (1998b). 'Three Good Reasons for Using a Petri-Net-Based Workflow Management System'. In *Information and Process Integration in Enterprises*, pp. 161–182. Springer US.

van der Aalst, W. M. P. (2000). 'Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques'. *Business Process Management* **1806**:19–128.

van der Aalst, W. M. P. (2012). 'Process Mining: Making Knowledge Discovery Process Centric'. *ACM SIGKDD Explorations Newsletter* **13**(2):45–49.

van der Aalst, W. M. P., Desel, J., & Oberweis, A. (eds.) (2000). *Business Process Management*, vol. 1806 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

van der Aalst, W. M. P., Van Hee, K. M., Ter Hofstede, A. H. M., Sidorova, N., Verbeek, H. M. W., Voorhoeve, M., & Wynn, M. T. (2011). 'Soundness of workflow nets: Classification, decidability, and analysis'. *Formal Aspects of Computing* **23**(3):333–363.

van der Maaten, L., Postma, E., & van den Herik, H. (2009). 'Dimensionality Reduction: A Comparative Review'. Tech. rep., Tilburg University.

van Rijn, J. N., Holmes, G., Pfahringer, B., & Vanschoren, J. (2014). 'Algorithm Selection on Data Streams'. In *17th International Conference (DS 2014)*, vol. 8777, pp. 325–336.

Vergara, A., Vembu, S., Ayhan, T., Ryan, M. A., Homer, M. L., & Huerta, R. (2012). 'Chemical gas sensor drift compensation using classifier ensembles'. *Sensors and Actuators B: Chemical* **166-167**:320–329.

Viswanathan, R. & Varshney, P. (1997). 'Distributed detection with multiple sensors I. Fundamentals'. *Proceedings of the IEEE* **85**(1):54–63.

Wang, L., Shao, C., Wang, H., & Wu, H. (2006). 'Radial Basis Function Neural Networks-Based Modeling of the Membrane Separation Process: Hydrogen Recovery from Refinery Gases'. *Journal of Natural Gas Chemistry* **15**(3):230–234.

Wei, H. & Billings, S. (2007). 'Feature Subset Selection and Ranking for Data Dimensionality Reduction'. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **29**(1):162–166.

Wei, W., Li, J., Cao, L., Ou, Y., & Chen, J. (2013). 'Effective detection of sophisticated online banking fraud on extremely imbalanced data'. *World Wide Web* **16**(4):449–475.

Widmer, G. & Kubat, M. (1993). 'Effective learning in dynamic environments by explicit context tracking'. In *European Conference on Machine Learning (ECML 1993)*, vol. 667, pp. 227–243.

Widmer, G. & Kubat, M. (1996). 'Learning in the Presence of Concept Drift and Hidden Contexts'. *Machine Learning* **01**:69–101.

Wirth, R., Shearer, C., Grimmer, U., Reinartz, T., Schlösser, J., Breitner, C., Engels, R., & Lindner, G. (1997). 'Towards process-oriented tool support for knowledge discovery in databases'. In *1st European Symposium (PKDD 1997)*, pp. 243–253.

Witten, I. & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc.

Wold, S., Sjöström, M., & Eriksson, L. (2001). 'PLS-regression: A basic tool of chemometrics'. *Chemometrics and Intelligent Laboratory Systems* **58**(2):109–130.

Wolpert, D. H. & Macready, W. G. (1997). 'No free lunch theorems for optimization'. *IEEE Transactions on Evolutionary Computation* **1**(1):67–82.

Wu, X. & Zhu, X. (2008). 'Mining With Noise Knowledge: Error-Aware Data Mining'. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* **38**(4):917–932.

Xie, Y. & Siegmund, D. (2013). 'Sequential multi-sensor change-point detection'. *The Annals of Statistics* **41**(2):670–692.

Yakovlev, A., Gomes, L., & Lavagno, L. (eds.) (2000). *Hardware Design and Petri Nets*. Springer US.

Yan, W., Shao, H., & Wang, X. (2004). 'Soft sensing modeling based on support vector machine and Bayesian model selection'. *Computers & Chemical Engineering* **28**(8):1489–1498.

Yang, T. (2005). 'Neural Networks for Solving On-Line Outlier Detection Problems'. In *Advances in Neural Networks (ISNN 2005)*, pp. 451–456.

Zakova, M., Kremen, P., Zelezny, F., & Lavrac, N. (2011). 'Automating Knowledge Discovery Workflow Composition Through Ontology-Based Planning'. *IEEE Transactions on Automation Science and Engineering* **8**(2):253–264.

Žliobaitė, I., Bifet, A., Gaber, M., Gabrys, B., Gama, J., Minku, L., & Musial, K. (2012). 'Next challenges for adaptive learning systems'. *ACM SIGKDD Explorations Newsletter* **14**(1):48–55.

Žliobaitė, I., Budka, M., & Stahl, F. (2015). 'Towards cost-sensitive adaptation: When is it worth updating your predictive model?'. *Neurocomputing* **150, Part A**:240–249.

Žliobaitė, I. & Gabrys, B. (2014). 'Adaptive Preprocessing for Streaming Data'. *IEEE Transactions on Knowledge and Data Engineering* **26**(2):309–321.

Žliobaitė, I. & Hollmén, J. (2014). 'Optimizing regression models for data streams with missing values'. *Machine Learning* **99**(1):47–73.