Automatic composition and optimisation of multicomponent predictive systems with an extended Auto-WEKA

Manuel Martin Salvador, Marcin Budka, and Bogdan Gabrys

***Abstract*—Composition and parametrisation of multicomponent predictive systems (MCPSs) consisting of chains of data transformation steps is a challenging task. Auto-WEKA is a tool to automate the Combined Algorithm Selection and Hyperparameter (CASH) optimisation problem. In this paper we extend the CASH problem and Auto-WEKA to support MCPS including preprocessing steps for both classification and regression tasks. We define the optimisation problem in which the search space consists of suitably parametrised Petri nets forming the sought MCPS solutions. In the experimental analysis we focus on examining the impact of considerably extending the search space (from approximately 22,000 to 812 billion possible combinations of methods and categorical hyperparameters). In a range of extensive experiments three different optimisation strategies are used to automatically compose MCPSs for 21 publicly available datasets. The diversity of the composed MCPSs found is an indication that fully and automatically exploiting different combinations of data cleaning and preprocessing techniques is possible and highly beneficial for different predictive models. We also present the results on 7 datasets from real chemical production processes. Our findings can have a major impact on development of high quality predictive models as well as their maintenance and scalability aspects needed in modern applications and deployment scenarios.**

***Note to Practitioners*—The extension of Auto-WEKA to compose and optimise MCPSs developed as part of this paper is freely available on GitHub under GPL licence and we encourage practitioners to use it on a broad variety of classification and regression problems. The software can either be used as a blackbox – where search space is made of all possible WEKA filters, predictors and meta-predictors (e.g. ensembles) – or as an optimisation tool on a subset of pre-selected machine learning methods. The application has a graphical user interface, but also can run from command line and can be embedded in any project as a Java library. There are three main outputs once an Auto-WEKA run has finished: a) the trained MCPS ready to make predictions on unseen data; b) the WEKA configuration (i.e. parametrised components); c) the Petri net in a PNML (Petri Net Markup Language) format which can be analysed using any tool supporting this standard language. There are however some practical considerations affecting the quality of the results that must be taken into consideration such as the CPU time budget or the search starting point. These are extensively discussed in the paper.**

***Index Terms*—Automatic predictive model building and parametrisation; Multicomponent predictive systems; KDD process; CASH problem; Bayesian optimisation; Data preprocessing; Predictive modelling; Petri nets**

M. Martin Salvador is with Blink Technologies, Palo Alto, CA, USA. (email: manuel@blinkeye.ai)

M. Budka is with Bournemouth University, Poole, United Kingdom. (email: mbudka@bournemouth.ac.uk)

B. Gabrys is with Advanced Analytics Institute, University of Technology Sydney, Australia. (email: Bogdan.Gabrys@uts.edu.au)

I. Introduction

PERFORMANCE of data-driven predictive models heavily relies on the quality and quantity of data used to build them. In real applications, even if 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. A survey[1] of data mining 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 [1] and 2016[2] confirm these numbers. The practitioners also note that preprocessing is the least enjoyable part of data science. The reason for this being such a lengthy process is 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 that passed 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 speeding up of the data mining process and making the procedures more robust.

After the data has been preprocessed in an appropriate way, the next step in a data mining process is modelling (i.e. finding an appropriate classifier or regressor). 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 [2], meta-learning [3], information theory [4] or following a multi-criteria decision making process [5]. More recently, Google has launched a new service called Cloud AutoML for automatically building deep neural networks for image classification problems, using transfer learning [6]. There is then an increasing attention from academy and industry in this topic. We have observed that 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

[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

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

In many scenarios one needs to sequentially apply multiple preprocessing methods to the data (e.g. outlier detection ! missing value imputation ! dimensionality reduction), effectively forming a preprocessing chain. Data-driven workflows have been used to guide data processing in a variety of fields. Some examples are astronomy [7], biology [8], clinical research [9], archive scanning [10], telecommunications [11], banking [12] and process industry [13], 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) [14], [15]. At the moment, tools like WEKA[3], RapidMiner[4] or Knime[5] allow to create and run MCPSs including a large variety of operators.

The motivation for automating composition of MCPS 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 cases it is not computationally feasible to evaluate all of them. Therefore it makes sense to approach preprocessing, model selection and hyperparameter optimisation problems jointly.

The Combined Algorithm Selection and Hyperparameter optimisation (CASH) problem presented in [16] consists of finding the best combination of learning algorithm A and hyperparameters that optimise an objective function (e.g. Eq. 1 minimises the k-fold cross-validation error) for a given dataset D. Formally, CASH problem is given by

$$A = \underset{A^{(j)} 2 A;\ 2\ ^{(j)}}{\arg\min} \frac{1}{k} \sum_{i=1}^{k} L(A^{(j)}; D^{(i)}_{train}; D^{(i)}_{valid}) \quad (1)$$

where $A = fA^{(1)}; : : : ; A^{(k)}g$ is a set of algorithms with associated hyperparameter spaces $^{(1)}; : : : ;\ ^{(k)}$. The loss function L takes as arguments an algorithm configuration A (i.e. an instance of a learning algorithm and hyperparameters), a training set D_{train} and a validation set D_{valid}.

In this paper we extend the CASH problem to support MCPSs, i.e. joint optimisation of predictive models (classifiers and regressors) and preprocessing chains. We embrace the representation of MCPS as Petri nets which we proposed in [15] and thus

$$= (P; T\ ; F) \quad (2)$$

[3]http://weka.sourceforge.net
[4]https://rapidminer.com
[5]https://www.knime.org

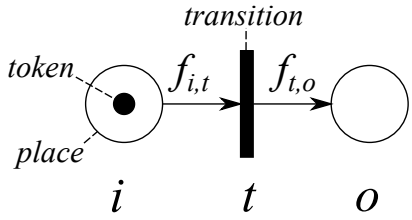

Fig. 1. Multicomponent predictive system with a single transition

In our previous work [17] we presented the first approach for hyperparameter optimisation of WEKA classifiers that modify their inner data preprocessing behaviour by recursively expanding the search space constructed by Auto-WEKA (a tool for solving the CASH problem defined by WEKA algorithms and hyperparameters). In this paper we present a further development of Auto-WEKA to support any combination of preprocessing methods (known as WEKA filters). This leads to significantly enlarging the search space of the CASH problem and automating the composition and optimisation of such complex MCPSs.

The paper is organised as follows. The next section reviews previous work in automating the CASH problem and highlights the available software. Section III extends CASH problem to MCPSs and describes the challenges related to automation of their composition. In Section III-A, our contributions to Auto-WEKA software, now allowing to create and optimise arbitrary chains of preprocessing steps followed by a predictive model, are presented. The methodology used to automate MCPS composition is discussed in Section IV followed by the results of extensive experimental analysis in Section V. In Section VI, we present the results of applying this approach to real datasets from the process industry. Finally, the paper concludes in Section VII.

II. Bayesian optimisation strategies

The CASH problem as shown in Eq. 1 can be approached in different ways. One example is a grid search, i.e. an exhaustive search over all the possible combinations of discretized parameters. Such technique can however be computationally prohibitive in large search spaces or with big datasets. A simpler mechanism like random search, where the search space is randomly explored in a limited amount of time, has been shown to be more effective in high-dimensional spaces [18].

A promising approach gaining popularity in the recent years is based on a Bayesian optimization framework [19], [20]. This approach – outlined in Algorithm 1 – aims to find

$$= \underset{2}{\arg\min}\, L(\ ; D_{train}; D_{test}) \quad (3)$$

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

$$p(L\ j\ R_{1:n})\ /\ p(R_{1:n}\ j\ L)p(L) \quad (4)$$

where $R_{1:n} = \{(\lambda_1, c_1), \ldots, (\lambda_n, c_n)\}$ is the set of run configurations and its associated costs. Since evaluating the loss function is costly, an acquisition function $p(L) : \Lambda \to \mathbb{R}$ 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 [21], the upper confidence bound criterion [22], or information gain [23].

Algorithm 1 Bayesian optimisation

1: **for** $n = 1, 2, \ldots$ **do**
2: $\lambda_{n+1} = \operatorname{argmax}_{\lambda \in \Lambda} (\lambda)$. select most promising configuration
3: $c_{\lambda_{n+1}} = L(\lambda_{n+1}, D_{train}, D_{test})$. compute cost
4: $R_{n+1} = \{R_n, (\lambda_{n+1}, c_{\lambda_{n+1}})\}$. update list of run configurations
5: update $p(L \mid R_{1:n+1})$
6: **end for**

In particular, Sequential Model-Based Optimization (SMBO) [24] is a Bayesian optimisation framework that incrementally builds a regression model – 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 $\lambda : \Lambda \to \mathbb{R}$. A function that has been shown to work well in SMBO framework [25] is the expected improvement (EI) [26] given by

$$(\lambda \mid R_n) = \mathbb{E}_{\lambda}[I(\lambda)] = \mathbb{E}_{\lambda}[\max\{0, c_{min} - c\}] \quad (5)$$

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 λ (i.e. running the configuration) since c can be estimated using λ. A common technique to select the next promising configuration consists of evaluating EI for thousands of random samples and then returning the best one [24]. Algorithm 2 shows the SMBO procedure that returns the best configuration found λ_{min} (also known as 'incumbent').

Algorithm 2 Sequential Model-Based Optimisation

λ_{min} = initial configuration (usually random sample from Λ)
$R = \{[\lambda_{min}, L(\lambda_{min}, D)]\}$. initialise set of run configurations and associated costs
repeat
  $M = FitModel(R)$
  $\lambda = FindNextConfiguration(M, \lambda, \lambda_{min}, \Lambda)$
  $c = L(\lambda, D)$. compute cost
  $R.add([\lambda, c])$. update list of run configurations
  $\lambda_{min} = \operatorname{argmin}_{c} \{[\lambda, c] \in R\}$. update best configuration found
until budget exhausted
return λ_{min}

Some hyperparameters influence the optimisation problem conditionally, i.e. only when some other hyperparameters take certain values. For example Gaussian kernel width in Support Vector Machine (SVM) is only relevant if SVM is using Gaussian kernels in the first place. Search spaces containing this type of hyperparameters are known as conditional spaces [20].

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) support conditional attributes. A successful SMBO approach using random forests is SMAC (Sequential Model-based Algorithm Configuration by [24]) where an ensemble of decision trees makes it possible to model conditional variables. Another state-of-the-art approach uses TPE [27], where a graph-structured model matches the conditional structure of the search space. Other SMBO approaches use Gaussian processes as surrogate models (e.g. [28]). However, they cannot work in conditional spaces because standard kernels are not defined over variable-length spaces [20] and therefore are not used in this paper.

Currently available software tools supporting SMBO methods are listed in Table I. It should be noted however, that to the best of our knowledge, there are no comprehensive studies or tools[6] which would tackle the problem of flexible composition of many data preprocessing steps (e.g. data cleaning, feature selection, data transformation) and their simultaneous parametric optimisation, which is one of the key issues addressed in this paper.

III. Automating MCPS composition

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 e.g. [30] for a survey), a reliable fully automated approach still does not exist.

To accommodate the definition of MCPS into a CASH problem we generalise A from Eq. 1 to be a set of MCPSs $= \{^{(1)}, ^{(2)}, \ldots\}$ rather than individual algorithms. Hence each MCPS $^{(j)} = (P, T, F)^{(j)}$ has now a hyperparameter space $^{(j)}$, which is a concatenation of the hyperparameter spaces of all its transitions T. The CASH problem is now concerned with finding (P, T, F) such as:

$$\operatorname*{arg\,min}_{(P,T,F)^{(j)} \in \Lambda^{(1)}, \Lambda^{(2)}, \ldots, \Lambda^{(j)}} \frac{1}{k} \sum_{i=1}^{k} L((P, T, F)^{(j)}, D^{(i)}_{train}, D^{(i)}_{valid}) \quad (6)$$

The main reason for MCPS composition being a challenging problem is the computational power and time needed to explore high dimensional search spaces. To begin with, an undetermined number of components can make the workflow very simple (see Figure 2) or very complex (see Figure 3). 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

[6] At the moment of writing this paper there were none, but some new tools have appeared in the meantime (see e.g. TPOT [29] https://github.com/EpistasisLab/tpot and DataRobot https://www.datarobot.com).

TABLE I
POPULAR OPEN-SOURCE TOOLS SUPPORTING SMBO METHODS

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.

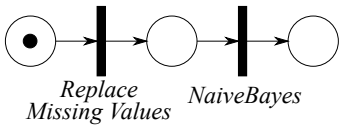

Fig. 2. Example of a simple MCPS

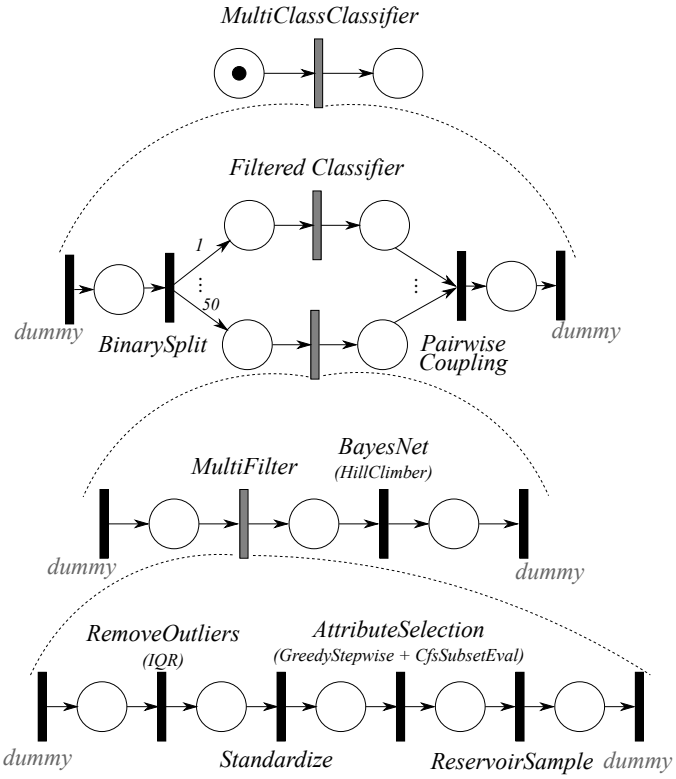

Fig. 3. Example of a complex MCPS

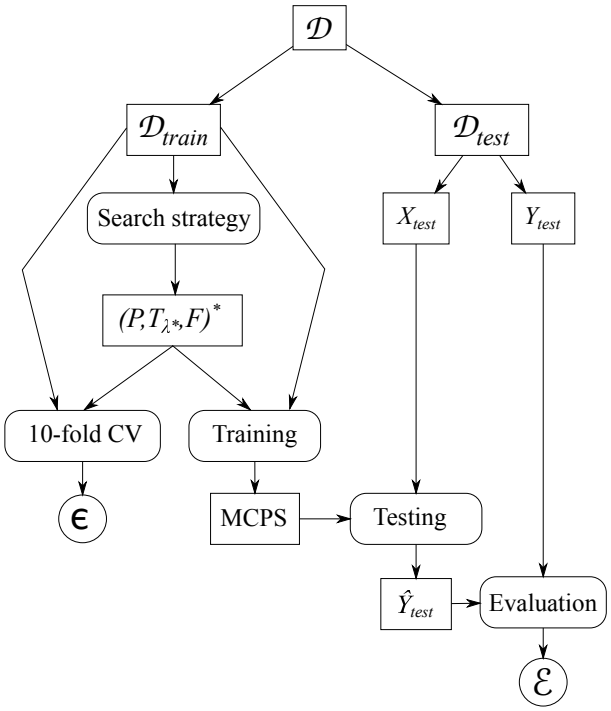

Fig. 4. MCPS training and testing process

(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 bad local minima.

The size of the search space (i.e. j j) can be reduced by applying a range of constraints like limiting the number of components, restricting the list of methods using meta-learning [31], prior knowledge [32] or surrogates (i.e. cheap-to-evaluate models [33]). However, this study 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 number and order of components which will be explained in Section IV.

We use the predictive performance as a sole optimisation objective as shown in Eq. 6, noting however, that some problems may require to optimise several objectives at the same time (e.g. error rate, model complexity and runtime [34]). In this paper we use our extended Auto-WEKA version – described in the next section – which supports automatic composition and optimisation of MCPSs with WEKA filters and predictive models as components.

Once the MCPS is composed and its hyperparameters optimised, it is trained with a set of labelled instances. Then the MCPS is ready to make predictions as shown in Figure 4.

A. Extension and generalisation of Auto-WEKA

Auto-WEKA is a software developed by Thornton et al. [16] which allows algorithm selection and hyperparameter optimisation both in regression and classification problems.

In this work we have extended Auto-WEKA to support MCPSs. This leads to an increasing of the search space from 21,560 up to 812 billion possible solutions as seen in Table II and further discussed in Section IV. This work has been developed taking as base Auto-WEKA version 0.5. Independently and in parallel to our work, there was further development of Auto-WEKA which current version is now 2.6 [35]. Nonetheless, this new version does not affect the work carried out in this paper as its main novelty is the integration with WEKA user interface.

Both versions provide a one-click solution for automating algorithm selection and hyperparameter optimisation. However, version 0.5 is much more flexible, offering as well multiple customisations possibilities like preselection of WEKA predictors, choosing the optimisation strategy or setting the optimisation criteria. Auto-WEKA 0.5 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 [36].

Our new extensions now allow any WEKA filter to be included as part of the composition process. In addition, we have developed a new WEKA filter that creates a flexible chain of common preprocessing steps including missing values handling, outlier detection and removal, data transformation, dimensionality reduction and sampling.

The following external WEKA packages[7] have been included as part of the developed Auto-WEKA extensions to increase the number of preprocessing methods: *EMImputation*, *RBFNetwork*, *StudentFilters*, *baggedLocalOutlierFactor*, *localOutlierFactor*, *partialLeastSquares* and *wavelet*. Furthermore, we have developed two new WEKA filters: (i) an outlier detection and removal filter in a single step; and (ii) a sampling filter in which instances are periodically selected given a fixed interval of time. These new filters are common operations in the preprocessing of datasets from the process industry.

Moreover, our 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[8]).

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.

While the space restriction does not allow us to include more implementation details, the source code and all the scripts for the analysis of the extended Auto-WEKA results such as the creation of plots and tables have been released in our repository[9].

[7]http://weka.sourceforge.net/packageMetaData/

[8]http://woped.dhbw-karlsruhe.de/woped/

[9]https://github.com/dsibournemouth/autoweka

IV. Methodology

The purpose of this experimental study is to analyse the feasibility of SMBO strategies for solving the MCPS related CASH problem as defined in Eq. 6 and the quality of the solutions found using the proposed extended Auto-WEKA with all its features supporting the MCPSs composition and parametric/hyperparametric optimisation of the workflows.

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 we have considered three search spaces of very different sizes (see Table II):

PREV: This is the search space used in [16] where predictors and meta-predictors (which take outputs from one or more base predictive models as their input) were considered (756 hyperparameters). It can also include the best feature selection method found after running 'AttributeSelection' WEKA filter (30 hyperparameters) for 15 minutes before the optimisation process begins. We use it 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. We would like to note however that some WEKA classifiers perform internal preprocessing steps as we showed in our previous work [17] ((e.g. MultiLayerPerceptron (MLP) removes instances with missing values and scales the attributes to a range [-1,1])). We take into account that a categorical hyperparameter can be either simple or complex (i.e. when it contains WEKA classes). In the latter case, we increase the search space 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 IV for details). That extension increases the search 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 nodes are connected in the following order: missing value handling *!* outlier detection and handling[10] *!* data transformation *!* dimensionality reduction *!* sampling *!* predictor *!* meta-predictor. This flow is based on our experience with process industry [13], but these preprocessing steps are also common in other fields. If the meta-predictor is either 'Stacking' or 'Vote', its number of inputs can vary from 1 to 5.

The methods that can be included in each component are listed in Tables III and IV. Note that the FULL search space is more than twice as large as the one presented in [16] in terms of the raw number of hyperparameters.

As the datasets we use in our experiments are intended for classification, we have chosen to minimise the classification error averaged over 10 CV folds within the optimisation process (i.e. $k = 10$ in Eq. 6).

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

TABLE II
SUMMARY OF SEARCH SPACES. MV = MISSING VALUE REPLACEMENT, OU = OUTLIER DETECTION AND REMOVAL, TR = TRANSFORMATION, DR = DIMENSIONALITY REDUCTION, SA = SAMPLING, P = PREDICTOR, MP = META-PREDICTOR, CH = EXPANDING COMPLEX HYPERPARAMETERS. * SIZE CONSIDERING ONLY METHODS, CATEGORICAL AND COMPLEX HYPERPARAMETERS.

	MV	OU	TR	DR	SA	P	MP	CH	Size*
PREV				✓		✓	✓		21,560
NEW						✓	✓	✓	2,369,598
FULL	✓	✓	✓	✓	✓	✓	✓	✓	812 billion

TABLE III
NUMBER OF PARAMETERS OF THE AVAILABLE PREPROCESSING METHODS.

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
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 IV
NUMBER OF PARAMETERS OF THE AVAILABLE PREDICTORS.

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
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

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

WEKA-Def: All the predictors and meta-predictors listed in Table IV 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 our results with the ones presented in [16] we have replicated the experimental settings as closely as possible. We have evaluated different optimisation strategies over 21 well-known datasets representing classification tasks (see Table V). Each dataset $D = \{D_{train}; D_{test}\}$ has been split into 70% training and 30% testing sets, unless partition was already provided. Please note that D_{train} is then split into 10-folds for Eq. 6 and therefore D_{test} is not used during the optimisation or training process at all (see Figure 4).

For each strategy we performed 25 runs 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 D_{train} and produce predictions for the testing set D_{test}. Please note that this budget limit may imply not finding the optimal solution in the optimisation problem.

Holdout error over D_{test} is denoted as $E = L(Y_{test}; \hat{Y}_{test})$. Random search, SMAC and TPE results have been calculated using the mean of 100,000 bootstrap samples (randomly selecting 4 of the 25 runs and keeping the one with lowest CV error as in original Auto-WEKA paper [16]), while only the lowest errors are reported for WEKA-Def.

V. Results

We organised our analysis around the following aspects: a) usefulness of automatic composition and parametrisation of MCPS; b) how efficient are Bayesian optimisation approaches (SMAC and TPE) in comparison to random search; c) impact of significantly extending the search space in the optimisation process; and d) identification of promising methods for each dataset.

A. Usefulness of automatic composition and parametrisation

An interesting aspect to analyse is if 30 CPU-core hours of automatic optimisation can beat a quick running of all WEKA classifiers with default hyperparameters. Table VI shows the results of Auto-WEKA experiments in the NEW search space using RANDOM, SMAC and TPE strategies compared to the WEKA-Def strategy. Not surprisingly, Auto-WEKA has been able to find better results for all datasets (> 0). In fact, Auto-WEKA presents a significant improvement in 19 out of 21 datasets.

TABLE V
Datasets, continuous and categorical attribute count, number of classes, and number of instances.

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

B. Effectiveness of Bayesian optimisation over random search

Another interesting aspect is to analyse how Bayesian optimisation approaches perform in comparison with just a random search. Table VII presents the results of Auto-WEKA runs for RANDOM, SMAC and TPE strategies in the NEW search space. SMAC has been able to find significantly better results in all datasets but one. Similarly, TPE outperforms random search in 18 out of 21 datasets.

C. Impact of extending the search space

As shown in Table II, the size between search spaces vary a lot: PREV with over 21 thousands; NEW with over 2 million; and FULL with over 812 billion possible solutions. We are interested on analysing the impact of extending the search space in the classification performance for each strategy.

Table VIII presents the results of RANDOM, SMAC and TPE over the three different search spaces. In the majority of cases (52 of 63), the MCPSs found in the NEW search space achieve significantly better results than in the smaller search space PREV. In 32 out of 63 cases, the FULL search space also gets significantly better performance than in PREV. However, finding good MCPS within the same time budget (30 CPU-core hours) is more challenging due to a large increase in the search space size [37]. As an example, consider Figure 5 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 5-a) and b) we 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 IX.

The way in which the search space is extended can have a considerable impact on the accuracy of the MCPS found.

TABLE VI
HOLDOUT ERROR E (% MISSCLASSIFICATION). LOWEST ERRORS REPORTED FOR WEKA-DEF AND AUTO-WEKA, BEING THE DIFFERENCE BETWEEN THE TWO. AGGREGATED MEAN HOLDOUT ERROR AND STANDARD DEVIATION ( ) FOR ALL AUTO-WEKA STRATEGIES AND SEARCH SPACES IS REPORTED. EACH AUTO-WEKA RUN HAD A 30 CPU CORE-HOURS BUDGET. AN UPWARD ARROW INDICATES A STATISTICALLY SIGNIFICANT IMPROVEMENT ($p < 0{:}05$) WITH RESPECT TO BEST WEKA-DEF USING ONE-SAMPLE WILCOXON SIGNED-RANK TEST.

Dataset	Best WEKA-Def	Best Auto-WEKA		Mean Auto-WEKA		
abalone	73.18	71.43	1.75	73.12	0.62	"
amazon	28.44	26.67	1.77	37.07	6.61	
car	0.77	0.00	0.77	0.06	0.12	"
cifar10	64.27	52.28	11.99	56.21	3.47	"
cifar10small	65.91	54.48	11.43	58.04	3.39	"
convex	25.96	18.47	7.49	22.89	2.07	"
dexter	8.89	5.00	3.89	7.50	1.23	"
dorothea	6.96	4.64	2.32	5.22	0.21	"
germancredit	27.33	23.33	4.00	25.44	1.38	"
gisette	2.81	1.95	0.86	2.33	0.24	"
kddcup09app	1.7405	1.6700	0.0705	1.7339	0.0191	"
krvskp	0.31	0.10	0.21	0.31	0.12	
madelon	21.38	15.64	5.74	17.61	1.44	"
mnist	5.19	2.66	2.53	4.01	0.61	"
mnistr	63.14	52.20	10.94	56.10	2.43	"
secom	8.09	7.66	0.43	7.86	0.07	"
semion	8.18	3.98	4.20	4.93	0.48	"
shuttle	0.0138	0.0100	0.0038	0.0100	0.0004	"
waveform	14.40	14.00	0.40	14.26	0.15	"
wineqw	37.51	32.33	5.18	32.94	0.43	"
yeast	40.45	36.40	4.05	37.73	0.74	"

TABLE VII
MEAN HOLDOUT ERROR E (% MISSCLASSIFICATION) AND STANDARD DEVIATION ( ) FOR RANDOM, SMAC AND TPE STRATEGIES IN THE NEW SEARCH SPACE. EACH AUTO-WEKA RUN HAD A 30 CPU CORE-HOURS BUDGET. AN UPWARD ARROW INDICATES A STATISTICALLY SIGNIFICANT IMPROVEMENT ($p < 0{:}05$) WITH RESPECT TO RANDOM USING WILCOXON SIGNED-RANK TEST.

Dataset	RANDOM		SMAC			TPE		
abalone	72.52	0.22	72.21	0.13	"	**72.01**	0.21	"
amazon	45.75	8.69	**39.54**	5.96	"	40.24	5.42	"
car	0.47	0.16	0.38	0.12	"	**0.21**	0.10	"
cifar10	58.91	3.67	56.44	2.90	"	**55.60**	2.77	"
cifar10small	60.44	4.08	57.91	2.54	"	**56.57**	1.79	"
convex	25.03	1.83	**21.88**	1.37	"	23.19	1.12	"
dexter	7.54	1.25	6.42	0.42	"	**6.19**	0.49	"
dorothea	6.25	0.56	5.95	0.49	"	**5.92**	0.53	"
germancredit	21.31	0.34	**19.66**	0.75	"	19.89	0.40	"
gisette	2.30	0.28	**2.21**	0.31	"	2.35	0.23	
kddcup09app	1.8000	0.0000	**1.7985**	0.0036	"	1.8000	0.0000	
krvskp	0.42	0.03	**0.28**	0.02	"	0.31	0.03	"
madelon	19.20	1.91	**15.61**	0.70	"	16.03	0.41	"
mnist	3.78	0.57	**3.49**	0.63	"	3.60	0.59	"
mnistr	58.09	1.54	**55.75**	2.44	"	57.17	2.13	"
secom	5.85	0.35	6.00	0.09		5.85	0.41	
semeion	4.82	0.35	4.48	0.47	"	**4.28**	0.37	"
shuttle	0.0109	0.0028	**0.0103**	0.0016	"	0.0107	0.0026	"
waveform	12.50	0.09	**12.33**	0.08	"	12.43	0.04	"
wineqw	33.08	0.24	**32.64**	0.27	"	32.67	0.18	"
yeast	37.16	0.34	36.50	0.35	"	**36.17**	0.23	"

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 best MCPSs found in the FULL search space for 13 out of 28 datasets have better or comparable performance to the best solutions found in the NEW space as shown in Table IX.

D. Identifying promising configurations

The best MCPSs found for each dataset are reported in Table IX, where each row represents a sequence of data transformations and predictive models as explained in Section IV. The solutions found for different datasets are quite diverse, and they often also vary a lot across the 25 random runs performed for each dataset. In order to better understand the observed differences in the MCPSs found we have also measured the average pairwise similarity of the 25 MCPSs found for each dataset and the variance of their performances (see Figure 6). To calculate the similarity between configurations a weighted sum of Hamming distances given by

$$d(\ _a;\ _b) = 1 \quad \frac{\sum_{i=1}^{n} (!_i \quad _i)}{\sum_{i=1}^{n} !_i} \tag{7}$$

is used, where $_a$ and $_b$ are MCPSs with N transitions, $!_i \ 2$ is the weight for the ith transition and $_i$ is the Hamming distance (a standard measure of string dissimilarity) of components at position i.

Weights have been fixed manually to $= f2; 1{:}5g$ in the NEW search space and $= f1; 1; 1; 1; 1; 2; 1{:}5g$ 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 [37].

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 doesn't 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 IV) 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 components 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 transformation methods tend to be retained in the found solutions.

For illustrative purposes we have selected three interesting cases from Figure 6 for a more detailed analysis:

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

TABLE VIII

MEAN HOLDOUT ERROR E (% MISSCLASSIFICATION) AND STANDARD DEVIATION ( ) FOR RANDOM, SMAC AND TPE WITH A 30 CPU CORE-HOURS BUDGET PER RUN. PREV COLUMNS CONTAIN THE VALUES REPORTED IN [16], WHILE NEW AND FULL COLUMNS CONTAIN THE RESULTS FOR THE SEARCH SPACES DESCRIBED IN SECTION IV. BOLDFACED VALUES INDICATE THE LOWEST MEAN CLASSIFICATION ERROR FOR EACH DATASET. AN UPWARD ARROW INDICATES A STATISTICALLY SIGNIFICANT IMPROVEMENT ($p < 0.05$) WITH RESPECT TO PREV SEARCH SPACE USING ONE-SAMPLE WILCOXON SIGNED-RANK TEST.

dataset	RANDOM						SMAC						TPE					
	PREV	NEW			FULL		PREV	NEW			FULL		PREV	NEW			FULL	
abalone	74.88	**72.92**	0.78	″	73.76	0.48	″	73.51	73.40	0.50	″	73.16	0.66	″	72.94	73.03	0.40	73.26	0.54	
amazon	41.11	39.18	9.17	″	56.65	15.60		**33.99**	36.28	4.41		49.11	16.00		36.59	35.71	4.60	″	54.06	11.79
car	0.01	0.13	0.15		1.84	1.04		0.40	0.05	0.11	″	0.20	0.23	″	0.18	0.01	0.04	″	0.05	0.12	″
cifar10	69.72	58.21	3.65	″	66.59	3.35	″	61.15	55.52	2.90	″	68.55	3.71		66.01	**54.88**	2.82	″	65.30	4.29	″
cifar10small	66.12	59.85	4.32	″	71.61	4.55		56.84	57.85	2.62		71.82	8.07		57.01	**56.43**	1.74	″	68.40	7.00	
convex	31.20	24.76	1.90	″	33.02	5.32		23.17	**21.31**	1.48	″	24.52	2.17		25.59	22.62	1.03	″	30.57	3.74	
dexter	9.18	8.27	1.19	″	11.27	2.70		7.49	7.31	0.85	″	8.13	4.02		8.89	**6.90**	1.18	″	8.00	1.82	″
dorothea	5.22	5.27	0.15		5.37	0.40		6.21	5.12	0.27	″	5.49	0.35	″	6.15	5.25	0.16	″	5.12	0.31	″
germancredit	29.03	**25.40**	1.56	″	26.87	1.42	″	28.24	25.43	1.55	″	26.67	1.20	″	27.54	25.49	0.95	″	26.63	1.24	″
gisette	4.62	2.28	0.24	″	3.36	1.39	″	**2.24**	2.35	0.26		2.74	1.57		3.94	2.37	0.19	″	2.86	0.46	″
kddcup09app	1.7400	**1.7214**	0.0296	″	1.7403	0.1103		1.7358	1.7400	0.0000		1.7400	0.0000		1.7381	1.7400	0.0000		1.7400	0.0000	
krvskp	0.58	0.34	0.10	″	0.39	0.15	″	0.31	**0.23**	0.11	″	0.39	0.07		0.54	0.36	0.09	″	0.33	0.08	″
madelon	24.29	19.11	1.30	″	23.80	4.53	″	21.56	**16.80**	0.95	″	17.16	3.68	″	21.12	16.91	0.52	″	17.59	2.59	″
mnist	5.05	4.00	0.68	″	10.93	5.57		**3.64**	4.08	0.38		10.57	6.83		12.28	3.96	0.73	″	12.32	9.13	
mnistr	66.40	57.15	1.80	″	65.89	4.83	″	57.04	**54.84**	2.51	″	65.48	5.62		70.20	56.30	2.33	″	63.90	5.36	″
secom	8.03	7.88	0.05	″	7.87	0.00	″	8.01	7.87	0.01	″	7.87	0.00	″	8.10	**7.84**	0.08	″	7.87	0.00	″
semeion	6.10	**4.78**	0.59	″	8.20	1.40		5.08	5.09	0.49		5.46	0.77		8.26	4.91	0.21	″	6.31	1.12	″
shuttle	0.0157	0.0100	0.0005	″	0.0217	0.0225		0.0130	0.0100	0.0005	″	0.0075	0.0054	″	0.0145	0.0100	0.0003	″	**0.0077**	0.0029	″
waveform	14.27	14.26	0.16	″	14.28	0.46		14.42	14.17	0.09	″	**13.99**	0.30	″	14.23	14.34	0.14		14.05	0.25	″
wineqw	34.41	32.99	0.58	″	36.64	1.48		33.95	32.89	0.31	″	34.14	0.76		33.56	**32.93**	0.33	″	34.09	0.44	
yeast	43.15	37.68	0.75	″	40.86	1.05	″	40.67	**37.60**	0.64	″	39.01	0.78	″	40.10	37.89	0.78	″	38.91	0.66	″

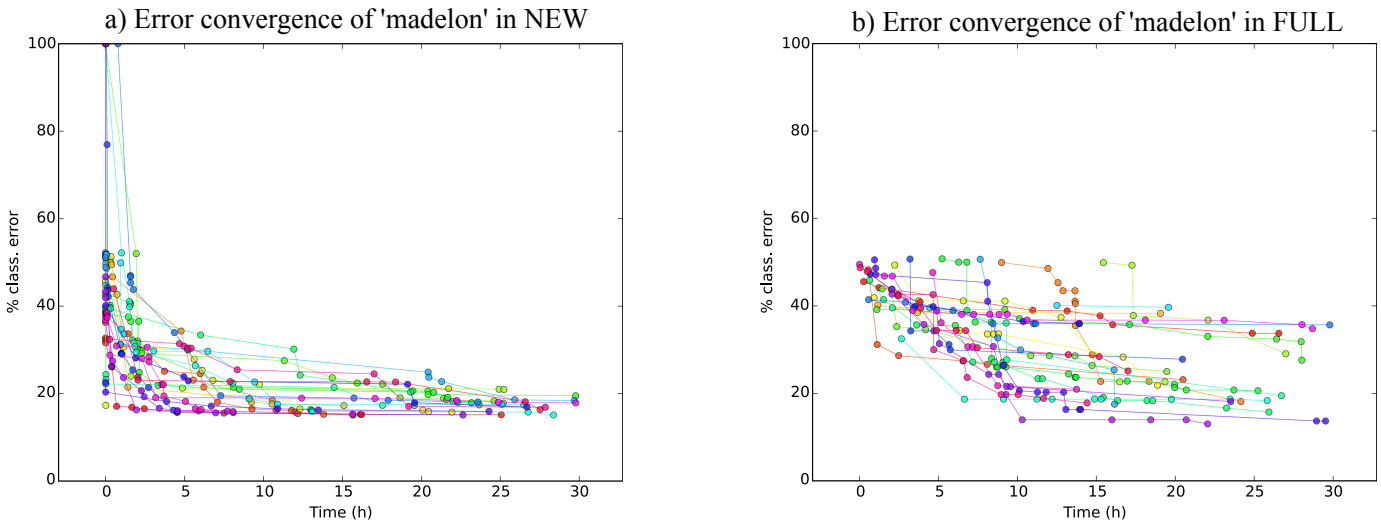

Fig. 5. 10-fold CV error of best solutions found over time for 'madelon' dataset and SMAC strategy in a) NEW and b) FULL search spaces.

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 6). We believe 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.

VI. APPLICATION TO PROCESS INDUSTRY

One motivation for automating the composition and optimisation of MCPSs was the need for speeding up the process of developing soft-sensors [38], which are predictive models based on easy-to-measure quantities used in the process industry. Main applications of soft-sensors are online prediction [39], process monitoring [40] and fault detection [41]. The most popular methods for process monitoring include Principal Component Analysis (PCA [42]) in a combination with a predictor, Multi-Layer Perceptron (MLP [43]), Radial Basis

TABLE IX
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.

dataset	space	MV	OU	TR	DR	SA	predictor	meta-predictor	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	Reservoir	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	-	Reservoir	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

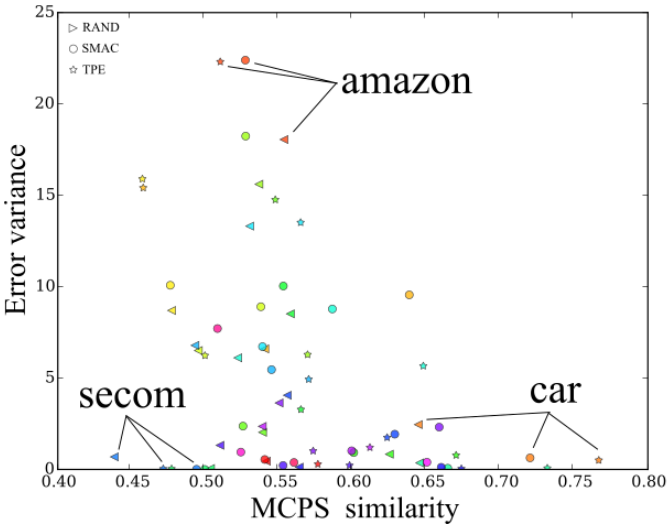

Fig. 6. Error variance vs. MCPS similarity in FULL search space

Function (RBF [44]) and Self Organizing Map (SOM [45]). [38] show that there are indeed dozens of methods to build soft sensors and each of them with various hyperparameters. Our experience in this field comes from past involvement in multiple projects with chemical engineering companies [46]–[48].

Raw data from chemical plants usually requires a considerable preprocessing and modelling effort [13], [49]. Although some WEKA predictors include inner preprocessing such as removal of missing values or normalisation, as shown previously in [17], many datasets need additional preprocessing to build effective predictive models. The fixed order of preprocessing nodes in the FULL search space has not been set arbitrarily – it follows the preprocessing guidelines that are common in process industry when developing predictive models (see e.g. [13], [38]).

We have carried out an experimental analysis on 7 datasets representing process monitoring tasks of real chemical processes (i.e. classification of 3 process states – low, normal and high). Four of these datasets have been made available

TABLE X
BEST MCPS FOR PROCESS INDUSTRY DATASETS IN NEW AND FULL SPACES, HOLDOUT ERROR E AND DIFFERENCE WITH BASELINE (″ INDICATES AN IMPROVEMENT). MV = MISSING VALUE REPLACEMENT, OU = OUTLIER REMOVAL, TR = TRANSFORMATION, DR = DIM. REDUCTION, SA = SAMPLING.

dataset	space	MV	OU	TR	DR	SA	predictor	meta-predictor	E	
absorber	NEW	-	-	-	-	-	REPTree	AdaBoostM1	51.04	-14.79
	FULL	-	-	Wavelet	RandomSubset	-	KStar	Bagging	45.63	-9.38
catalyst	NEW	-	-	-	-	-	LMT	AdaBoostM1	55.34	-22.90
	FULL	-	-	Wavelet	RandomSubset	-	JRip	FilteredClassifier	48.64	-16.20
debutanizer	NEW	-	-	-	-	-	LMT	AdaBoostM1	49.03	″ 3.20
	FULL	Median	-	Wavelet	-	-	REPTree	FilteredClassifier	50.14	″ 2.09
drier	NEW	-	-	-	-	-	JRip	Bagging	43.72	″ 7.37
	FULL	-	IQR	Normalize	-	-	Logistic	Bagging	40.98	″ 10.11
oxeno	NEW	-	-	-	-	-	JRip	AdaBoostM1	35.74	-2.22
	FULL	-	-	Standardize	-	-	JRip	RandomSubSpace	38.24	-4.74
sulfur	NEW	-	-	-	-	-	PART	Bagging	77.91	″ 1.75
	FULL	-	-	Wavelet	RandomSubset	-	JRip	FilteredClassifier	70.96	″ 8.70
thermalox	NEW	-	-	-	-	-	Logistic	MultiClassClassifier	28.01	″ 19.51
	FULL	-	-	Wavelet	-	-	MLP	FilteredClassifier	26.71	″ 20.81

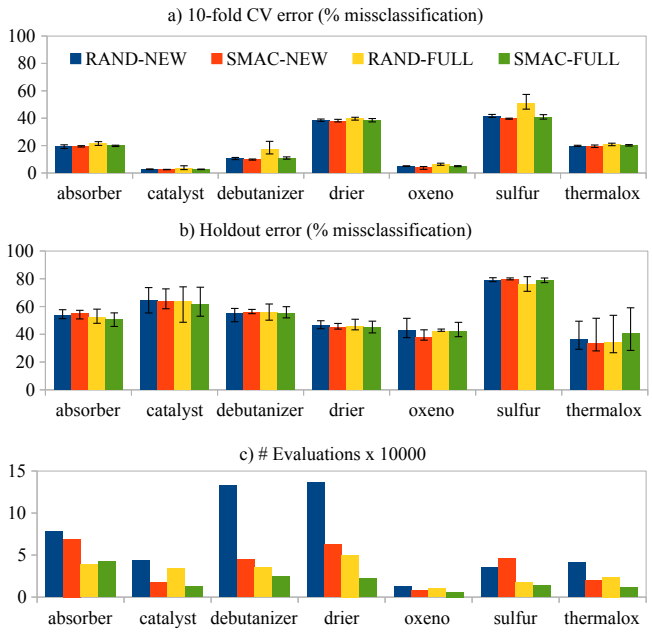

Fig. 7. a) Mean 10-fold CV error and b) holdout error E with 95% bootstrap confidence intervals for process industry datasets. c) Total number of evaluations per dataset and strategy.

by Evonik Industries as part of the collaboration within the INFER project [50], and have been extensively used in previous studies [13], [51], [52]:

'absorber' which contains 38 continuous attributes from an absorption process. No additional information has been provided apart from this being a regression task;
'drier' with 19 continuous features from physical sensors (i.e. temperature, pressure and humidity) and the target value is the residual humidity of the product [51];
'oxeno' which contains 71 continuous attributes also from physical sensors and a target variable which is the product concentration measured in the laboratory [13]; 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 [51].

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:

'catalyst' with 14 attributes, where the task is to predict the activity of a catalyst in a multi-tube reactor [46];
'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 [53]; and
the 'sulfur' recovery unit, which is a system for removing environmental pollutants from acid gas streams before they are released into the atmosphere [54]. 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 results presented in Figure 7 show that including such preprocessing steps has allowed, on average, to find better MCPSs than in the NEW search space for selected datasets. Although the difference is small, that implies that considerably extending the search space does not only have major negative effects but also can be positive for the predictive performance on these datasets. It is interesting to highlight that random search was able to find the MCPSs with lowest holdout error in half of the cases, but on the other hand also presents a higher error variance. We believe that the reason behind this is that random search has evaluated more models than SMAC for almost all the runs (see Figure 7-c)). This suggests that random search – which evaluates more potential solutions and explores more regions of the search space – can, and in these few cases has found better solutions than SMAC.

The best MCPSs found for the chemical datasets are shown at the end of Table X. These solutions 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 in Table X). Also, MCPSs including an attribute selection step have a considerable improvement of performance with respect to NEW (e.g. 'absorber', 'catalyst' and 'sulfur').

We also can see in Figure 7 that there is a large difference between the CV error and the holdout test error in some of these datasets (e.g. = 2:60% to E = 61:27% 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. We have shown in [48] how it is possible to adapt these MCPSs, using the proposed automatic composition and optimisation approach repeatedly, without any human intervention, when there are changes in data.

VII. Conclusions and future work

In this work automatic composition and optimisation of MCPSs has been addressed as a search problem. In particular, we proposed extending the CASH problem to MCPS represented as Petri nets, where transitions are data processing methods with hyperparameters. To apply this approach to real problems, we have extended Auto-WEKA to support any type of preprocessing methods available in WEKA and thus form MCPSs. In an extensive experimental analysis using 21 publicly available datasets and 7 challenging datasets from the process industry, we have demonstrated that it is indeed possible to find feasible MCPSs to solve predictive problems. 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 the analysed datasets. As a consequence, we can considerably reduce the human effort and speed up the data mining process.

Based on the error variance among different runs of the extended Auto-WEKA on the same dataset and the similarity of the MCPSs found, we have identified three interesting cases which may happen when finding the best solution using multiple starting points in the search space: (1) convergence towards very similar solutions; (2) small error variation between different solutions; and (3) very different solutions. From the practical point of view, a single composition and optimisation run should be sufficient when dealing with datasets falling into the first two categories. On the other hand, it is clear that multiple composition and optimisation runs with different starting points are needed for datasets in case (3). The challenge is to identify a priori to which category any of the considered datasets belong. This is a promising future work direction, with a potential of further time and computational cost savings. A similar challenge is to define a priori or in a flexible, dynamic manner a time budget for any particular dataset.

In contrast to the collection of datasets presented in [16] (and also evaluated in this paper), data distribution of the evaluated process industry datasets is changing over time. In these cases there is a need to adapt the optimised MCPSs following the changing environment. The first approach to deal with concept drift while optimising MCPSs in such datasets has been presented in [48], though the adaptation mechanisms for SMBO methods require further systematic research and form one of our future work directions.

In addition, it would also be valuable to investigate if using different data partitioning like Density-Preserving Sampling (DPS [55]) would make any difference in the optimisation process. We believe that DPS could have a considerable impact when using certain datasets in SMAC strategy since 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 [56] can happen quite often with CV, the effect on the solutions found can be detrimental.

Finally, 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 [34].

References

[1] M. A. Munson, "A study on the importance of and time spent on different modeling steps," *ACM SIGKDD Explorations Newsletter*, vol. 13, no. 2, pp. 65–71, 2012.

[2] R. Leite, P. Brazdil, and J. Vanschoren, "Selecting classification algorithms with active testing," in *LN in Computer Science*, vol. 7376 LNAI, 2012, pp. 117–131.

[3] C. Lemke and B. Gabrys, "Meta-learning for time series forecasting and forecast combination," *Neurocomputing*, vol. 73, no. 10-12, pp. 2006–2016, 2010.

[4] A. MacQuarrie and C.-L. Tsai, *Regression and Time Series Model Selection*. World Scientific, 1998.

[5] R. Ali, S. Lee, and T. C. Chung, "Accurate multi-criteria decision making methodology for recommending machine learning algorithm," *Expert Systems with Applications*, vol. 71, pp. 257–278, 2017.

[6] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," *arXiv cs.CV*, 2017.

[7] G. B. Berriman, E. Deelman, J. Good, J. C. Jacob, D. S. Katz, A. C. Laity, T. A. Prince, G. Singh, and M. H. Su, *Generating complex astronomy workflows*. Springer London, 2007, pp. 19–38.

[8] A. Shade and T. K. Teal, "Computing Workflows for Biologists: A Roadmap," *PLoS Biol.*, vol. 13, no. 11, pp. 1–10, 2015.

[9] E. Teichmann, E. Demir, and T. Chaussalet, "Data preparation for clinical data mining to identify patients at risk of readmission," in *IEEE 23rd International Symposium on Computer-Based Medical Systems*, 2010, pp. 184–189.

[10] I. Messaoud, H. El Abed, V. Märgner, and H. Amiri, "A design of a preprocessing framework for large database of historical documents," in *Proc. of the 2011 Workshop on Historical Document Imaging and Processing*, 2011, pp. 177–183.

[11] A. Maedche, A. Hotho, and M. Wiese, "Enhancing Preprocessing in Data-Intensive Domains using Online-Analytical Processing," in *Second Int. Conf. DaWaK 2000*, 2000, pp. 258–264.

[12] W. Wei, J. Li, L. Cao, Y. Ou, and J. Chen, "Effective detection of sophisticated online banking fraud on extremely imbalanced data," *World Wide Web*, vol. 16, no. 4, pp. 449–475, 2013.

[13] M. Budka, M. Eastwood, B. Gabrys, P. Kadlec, M. Martin Salvador, S. Schwan, A. Tsakonas, and I. Žliobaitė, "From Sensor Readings to Predictions: On the Process of Developing Practical Soft Sensors," in *Advances in IDA XIII*, vol. 8819, 2014, pp. 49–60.

[14] A. Tsakonas and B. Gabrys, "GRADIENT: Grammar-driven genetic programming framework for building multi-component, hierarchical predictive systems," *Expert Systems with Applications*, vol. 39, no. 18, pp. 13 253–13 266, 2012.

[15] M. Martin Salvador, M. Budka, and B. Gabrys, "Modelling Multi-Component Predictive Systems as Petri Nets," in *15th Annual Industrial Simulation Conference*, 2017, pp. 17–23.

[16] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms," in *Proc. of the 19th ACM SIGKDD*, 2013, pp. 847–855.

[17] M. Martin Salvador, M. Budka, and B. Gabrys, "Towards Automatic Composition of Multicomponent Predictive Systems," in *11th International Conference on Hybrid Artificial Intelligence Systems*, 2016, pp. 27–39.

[18] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[19] E. Brochu, V. M. Cora, and N. de Freitas, "A Tutorial on Bayesian Optimization of Expensive Cost Functions with Application to Active User Modeling and Hierarchical Reinforcement Learning," University of British Columbia, Department of Computer Science, Tech. Rep., 2010.

[20] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the Human Out of the Loop: A Review of Bayesian Optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.