Towards Automatic Composition of Multicomponent Predictive Systems

Manuel Martin Salvador(✉), Marcin Budka, and Bogdan Gabrys

Data Science Institute, Bournemouth University, Poole, UK
{msalvador,mbudka,bgabrys}@bournemouth.ac.uk

Abstract. Automatic composition and parametrisation of multicomponent predictive systems (MCPSs) consisting of chains of data transformation steps is a challenging task. In this paper we propose and describe an extension to the Auto-WEKA software which now allows to compose and optimise such flexible MCPSs by using a sequence of WEKA methods. In the experimental analysis we focus on examining the impact of significantly extending the search space by incorporating additional hyperparameters of the models, on the quality of the found solutions. In a range of extensive experiments three different optimisation strategies are used to automatically compose MCPSs on 21 publicly available datasets. A comparison with previous work indicates that extending the search space improves the classification accuracy in the majority of the cases. The diversity of the found MCPSs are also an indication that fully and automatically exploiting different combinations of data cleaning and preprocessing techniques is possible and highly beneficial for different predictive models. This can have a big impact on high quality predictive models development, maintenance and scalability aspects needed in modern application and deployment scenarios.

Keywords: KDD process · CASH problem · Bayesian optimisation · Data mining and decision support systems · Data preprocessing

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 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 total time spent on development of a predictive model [1,2]. Therefore, approaches and practical techniques that allow 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 → missing value imputation →

F. Martínez-Álvarez et al. (Eds.): HAIS 2016, LNAI 9648, pp. 27–39, 2016.
DOI: 10.1007/978-3-319-32034-2_3

dimensionality reduction), effectively forming a preprocessing chain. Composition of a preprocessing chain is a challenging problem that has been addressed in different fields (e.g. clinical data [3,4], historical documents [5] and process industry [6]). Despite these works attempting to be as abstract as possible, the frameworks they propose are optimised for a particular case study. Hence their underlying approaches are lacking in the context of cross-domain generalisation, and potentially require additional manual work to improve the results.

After the data has been pre-processed in an appropriate way, the next step in a data mining process is modelling. Similarly to preprocessing, this step can also be very time-consuming, requiring evaluation of multiple alternative models. Hence automatic model selection has been attempted in different ways, e.g. active testing [7], meta-learning [8] and information theory [9]. A common theme in the literature is comparison of different models using data always pre-processed in the same way. However, some models may perform better if they are built using data specifically pre-processed with a particular model type in mind.

Hyper-parameter optimisation is an additional step which is usually performed after the model has been selected (see e.g. [10–12]). However, the problem is in fact very similar to model selection since different parametrisations of the same model (e.g. different kernels in an SVM, different structures of a neural network) can be treated as different models. Therefore it makes sense to approach both model and hyper-parameter selection problems jointly. The Combined Algorithm Selection and Hyper-parameter optimisation (CASH) problem was presented in [13]. A limitation of that study was the use of only two native types of attributes (i.e. numerical and categorical). This paper extends the approach presented in [13] to support complex categorical attributes, which consists of a WEKA class and therefore can contain additional hyper-parameters.

We refer to a sequence of preprocessing steps followed by a machine learning model as a multicomponent predictive system (MCPS). The motivation for automating composition of MCPS is twofold. In the first instance it will help to reduce the amount of time spent on such activities and therefore allow to dedicate the 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.

This paper is organised as follows. The next section reviews previous work in automating the CASH problem and available software. Section 3 describes multicomponent predictive systems and the challenges related to automation of their composition. In Sect. 4, our contributions to Auto-WEKA software, now allowing to optimise additional hyperparameters, are presented. The methodology used to automate MCPS composition is discussed in Sect. 5 followed by the results in Sect. 6, and conclusions in Sect. 7.

2 Related Work

The Combined Algorithm Selection and Hyper-parameter configuration (CASH) problem [13] consists of finding the best combination of learning algorithm A^* and hyperparameters λ^* that optimise an objective function $\mathcal{L}$ (e.g. Eq. 1 minimises the k-fold cross-validation error) for a given dataset $\mathcal{D}$. Formally, CASH problem is given by:

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

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

The CASH problem can be approached in different ways. One example is grid search – 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. Instead, 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 [12].

A promising approach that is gaining popularity in the last years is Bayesian optimization [14]. In particular, Sequential Model-Based Optimization (SMBO) [15] is a framework that incrementally builds a regression model using instances formed of an algorithm configuration λ and its predictive performance c. Such a model is then used to explore promising candidate configurations.

Examples of SMBO methods are SMAC (Sequential Model-based Algorithm Configuration [15]) and TPE (Tree-structure Parzen Estimation [16]). SMAC models $p(c|\lambda)$ using a random forest, while TPE maintains separate models for $p(c)$ and $p(\lambda|c)$. Both methods support continuous, categorical and conditional attributes (i.e. attributes whose presence in the optimisation problem depend on the values of some other attributes – e.g. Gaussian kernel width parameter in SVM is only present if SVM is using Gaussian kernels).

Two other methods falling into the same category are ROAR (Random Online Aggressive Racing [15]) which randomly selects the set of candidates instead of using a regression model, and Spearmint [17] which uses Gaussian processes to model $p(c|\lambda)$ similarly to SMAC. However, due to their limitations (it has been shown that SMAC outperforms ROAR [15], while Spearmint does not support conditional attributes) we are not using these methods in this study.

Currently available software tools supporting SMBO methods are:

- SMAC[1] [15] tool, which includes both SMAC and ROAR methods.
- Hyperopt[2] [16] is a Python library including random search and TPE.

[1] http://www.cs.ubc.ca/labs/beta/Projects/SMAC.
[2] https://github.com/hyperopt/hyperopt.

- HPOLib[3] [18] is a wrapper for SMAC, TPE and Spearmint methods.
- Auto-Sklearn[4] [19] uses HPOLib and meta-learning to automate the selection of scikit-learn methods.
- Auto-WEKA[5] [13] allows to use either SMAC or TPE to automatically select and optimise 39 classifiers and 24 possible feature selection methods included in WEKA[6].

3 MCPS Description

A multicomponent predictive system can be represented as a directed acyclic graph, where the vertices correspond to data transformations and the edges represent the flow of data between the components. In this study we focus on MCPSs of the $F = \langle f_1, ..., f_N \rangle$ form, which as shown in Fig. 1, are directed linear graphs of length N. The components f_1 to f_{N-1} are preprocessing methods while component f_N is either a predictive model or an ensemble of models. Although post-processing methods can also be included as additional components, without the loss of generality, we are not investigating such MCPSs here.

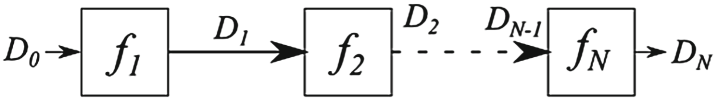

Fig. 1. N-component predictive system

The i^{th} component is a function $D_i = f_i(D_{i-1})$ that takes a data frame D_{i-1} as input and returns a data frame D_i. Such data frames can have any size and contain any type of data (e.g. continuous or categorical). However, values of a single column are assumed to be of identical type.

The connection between components must be consistent, i.e. output of component f_n must be compatible with the input of f_{n+1}. For example, if component f_{n+1} cannot handle missing values, output from f_n should not contain any.

Typically, the first component receives the raw data as input, and the last component returns the predictions (i.e. a label in case of classification problems or a continuous value in regression problems). The number of components in an MCPS can vary from 1 (i.e. when no preprocessing is used) to $N \in \mathbb{N}_{>1}$. Some flows in OpenML experiments repository[7] contain up to 7 components, while data mining tools like RapidMiner or Knime allow building even longer flows.

[3] https://github.com/automl/hpolib.
[4] https://github.com/mfeurer/auto-sklearn.
[5] http://www.cs.ubc.ca/labs/beta/Projects/autoweka.
[6] An open-source data mining package developed at the University of Waikato.
[7] http://openml.org.

An MCPS can be seen as a type of workflow that can be found in platforms like Taverna[8]. However, we prefer to avoid such generic term in this paper in order to highlight the predictive nature of this type of systems.

Building an MCPS is typically a manual process. Even though some efforts are being made to automate such process (see e.g. [20] for a survey), a reliable fully automated approach is still far from being available.

The main reason that makes MCPS composition a challenging problem is the potentially huge size of the search space. To begin with, the undetermined number of components can make the flow very simple or very complex. Secondly, we do not know a priori the order in which the components have to be connected. Also, the type of each hyper-parameter can be either continuous, categorical or conditional. Therefore, defining a range for each of the hyperparameters is an additional problem in itself.

The search space size can be reduced by applying constraints like limiting the number of components, restricting the list of methods using meta-learning [21], prior knowledge [22] or surrogates [23].

In this paper we use the predictive performance as a single optimisation objective. We note however, that some problems may require to optimise several objectives at the same time (e.g. error rate, model complexity and runtime [24]).

4 Contribution to Auto-WEKA

Auto-WEKA is a software developed by Thornton et al. [13] which allows algorithm selection and hyper-parameter optimisation both in regression and classification problems. The current Auto-WEKA version is 0.5 and it provides an interface for a CASH problem, where the search space is defined by WEKA predictors and feature selection methods. Available optimisation strategies are SMAC and TPE.

We have developed a new feature consisting of automatic expansion of WEKA classes during the creation of the search space. That is, if a hyper-parameter is a category formed of several WEKA classes, Auto-WEKA will now add the hyperparameters of such class recursively. For example, the kernel of an SVM classifier can be either NormalizedPolyKernel, PolyKernel, Puk, or RBFKernel; and each of them contains different hyperparameters.

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

5 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 we have considered two search spaces:

[8] http://www.taverna.org.uk.

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

- PREV: This is the search space used in [13] 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). We use it as a baseline.
- NEW: This search space only includes predictors and meta-predictors. In contrast with PREV space, no preprocessing steps are involved. Instead, 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). That extension increases the search space to 1186 hyperparameters.

Table 1. Datasets, number of continuous attributes, number of categorical attributes, number of classes, number of instances, and percentage of missing values.

Dataset	Cont	Disc	Class	Train	Test	%Miss
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

The objective function guides the optimisation process. Since the datasets we use in our experiments are intended for classification, we have chosen to minimise the classification error averaged over 10 cross-validation folds.

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 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-hours for the process.
- SMAC and TPE: An initial configuration is randomly selected and then the optimiser is run for 30 CPU-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 [13] 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 1). For each strategy we performed 25 runs with different random seeds and a 30 CPU-hours optimisation time limit. In case a configuration exceeds 30 min or 3 GB of RAM to evaluate, it is stopped and not considered further. Once the optimisation process has finished, the returned configuration is used to build a model using the whole training set and produce predictions over the testing set.

6 Results and Discussion

Classification performance for each dataset are presented in Tables 2 and 3, which show the 10-fold cross-validation error and the test error achieved by each strategy, respectively. Random search, SMAC and TPE results have been calculated using the mean 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 in order to simulate a 4-core CPU), while only the lowest errors are reported for WEKA-Def. PREV columns contain the values reported in [13], while NEW columns contain the results of the strategies using extended search space investigated in this paper. An upward arrow indicates an improvement when using extended search space (NEW) in comparison to previous results (PREV) reported in [13]. Boldfaced values indicate the lowest classification error for each dataset.

Table 2. 10-fold Cross Validation error (% 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	PREV	NEW	PREV	NEW	PREV	NEW
abalone	73.33	72.03	72.53	**71.71**	72.21	72.14	72.01 ↑
amazon	43.94	59.85	45.72 ↑	47.34	**39.57** ↑	50.26	40.27 ↑
car	2.71	0.53	0.47 ↑	0.61	0.38 ↑	0.91	**0.21** ↑
cifar10	65.54	69.46	58.89 ↑	62.36	56.44 ↑	67.73	**55.59** ↑
cifar10small	66.59	67.33	60.45 ↑	58.84	57.90 ↑	58.41	**56.56** ↑
convex	28.68	33.31	25.02 ↑	25.93	**21.88** ↑	28.56	23.19 ↑
dexter	10.20	10.06	7.54 ↑	**5.66**	6.42	9.83	6.19 ↑
dorothea	6.03	8.11	6.25 ↑	**5.62**	5.95	6.81	5.92 ↑
germancredit	22.45	20.15	21.31	**17.87**	19.65	21.56	19.88 ↑
gisette	3.62	4.84	2.30 ↑	2.43	**2.21** ↑	3.55	2.35 ↑
kddcup09app	1.88	1.75	1.80	**1.70**	1.80	1.88	1.80 ↑
krvskp	0.89	0.63	0.42 ↑	0.30	**0.28** ↑	0.43	0.31 ↑
madelon	25.98	27.95	19.20 ↑	20.70	**15.61** ↑	24.25	16.03 ↑
mnist	5.12	5.05	3.78 ↑	3.75	**3.50** ↑	10.02	3.60 ↑
mnistr	66.15	68.62	58.10 ↑	57.86	**55.73** ↑	73.09	57.17 ↑
secom	6.25	5.27	5.85	**5.24**	6.01	6.21	5.85 ↑
semeion	6.52	6.06	4.82 ↑	4.78	4.48 ↑	6.76	**4.28** ↑
shuttle	0.0328	0.0345	0.0121 ↑	0.0224	0.0112 ↑	0.0251	**0.0104** ↑
waveform	12.73	12.43	12.50	**11.92**	12.33	12.55	12.43 ↑
wineqw	38.94	35.36	33.08 ↑	34.65	**32.64** ↑	35.98	32.67 ↑
yeast	39.43	38.74	37.16 ↑	35.51	36.50	**35.01**	36.17

As shown in the tables, in the majority of the cases, expanding the search space has been beneficial for finding better solutions (i.e. NEW < PREV). The potential negative impact on predictive accuracy by optimising many more parameters has not been observed. Although, we still found some cases with higher classification error (i.e. NEW > PREV).

Figure 2 compares CV error and test error for all the runs. Although both errors are in general consistent, we found some cases in which the model seems to overfit. That is the case for example in 'germancredit' and 'amazon' datasets. A possible explanation is given below, at the end of this section.

The best MCPSs found for each dataset are reported in Table 4. Each row of this table represents a sequence of data transformations and predictive models. It is not feasible to include all the hyperparameters in the table due to limited space, but they do play a crucial role in the final performance. We can see that all the MCPSs include at least one or more preprocessing steps.

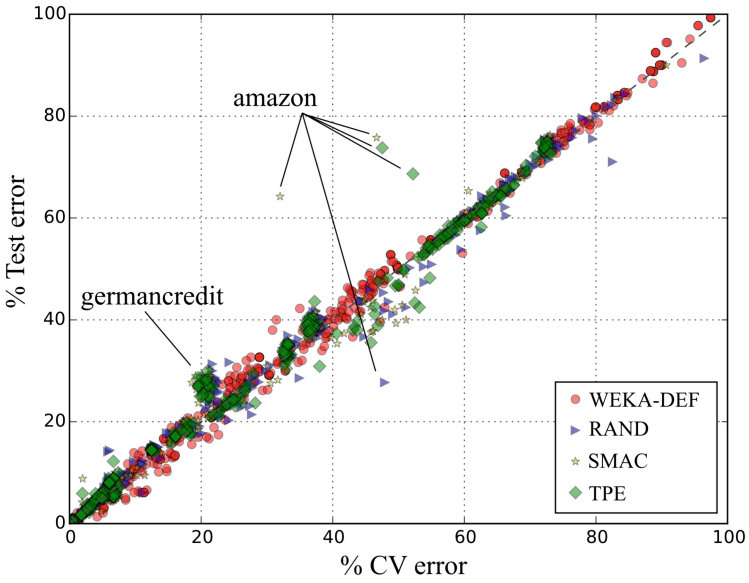

Fig. 2. CV vs Test error of all runs

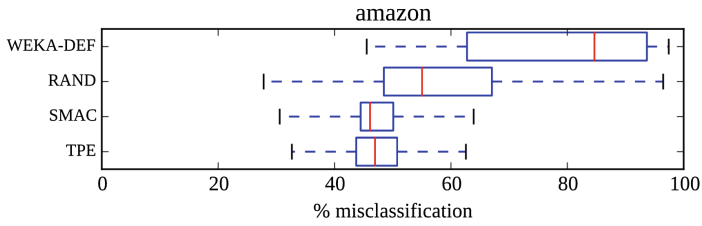

Fig. 3. Range of CV errors for 'amazon'

As shown in Table 4 the solutions found are quite diverse for different datasets but they often also vary a lot across the 25 random seed runs performed for each dataset. In order to better understand all of the observed differences in the MCPSs found we have therefore also measured the average pairwise similarity of the 25 MCPSs found for each dataset and the variance between their performances. While in this paper there is no space for full analysis or explanation of how the pairwise similarity was measured, below we have selected three interesting cases for a brief discussion:

– 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 'wineqw' dataset with SMAC optimisation is RandomForest (22/25) → MultiClassClassifier (8/25).

Table 3. Test error (% missclassification). An upward arrow indicates an improvement with respect to PREV space. Boldfaced values indicate the lowest classification error for each dataset.

Dataset	DEF	RANDOM		SMAC		TPE	
	PREV = NEW	PREV	NEW	PREV	NEW	PREV	NEW
abalone	73.18	74.88	**72.92** ↑	73.51	73.41 ↑	72.94	73.04
amazon	**28.44**	41.11	39.22 ↑	33.99	36.26	36.59	35.69 ↑
car	0.7700	0.0100	0.1300	0.4000	0.0526 ↑	0.1800	**0.0075** ↑
cifar10	64.27	69.72	58.23 ↑	61.15	55.54 ↑	66.01	**54.88** ↑
cifar10small	65.91	66.12	59.84 ↑	56.84	57.87	57.01	**56.41** ↑
convex	25.96	31.20	24.75 ↑	23.17	**21.31** ↑	25.59	22.62 ↑
dexter	8.89	9.18	8.29 ↑	7.49	7.31 ↑	8.89	**6.90** ↑
dorothea	6.96	5.22	5.27	6.21	**5.12** ↑	6.15	5.25 ↑
germancredit	27.33	29.03	**25.40** ↑	28.24	25.42 ↑	27.54	25.49 ↑
gisette	2.81	4.62	2.28 ↑	**2.24**	2.34	3.94	2.37 ↑
kddcup09app	1.7405	1.74	**1.72** ↑	1.7358	1.74	1.7381	1.74
krvskp	0.31	0.58	0.34 ↑	0.31	**0.23** ↑	0.54	0.36 ↑
madelon	21.38	24.29	19.10 ↑	21.56	**16.80** ↑	21.12	16.91 ↑
mnist	5.19	5.05	4.00 ↑	**3.64**	4.10	12.28	3.96 ↑
mnistr	63.14	66.4	57.16 ↑	57.04	**54.86** ↑	70.20	56.31 ↑
secom	8.09	8.03	7.88 ↑	8.01	7.87 ↑	8.10	**7.84** ↑
semeion	8.18	6.10	**4.78** ↑	5.08	5.10	8.26	4.91 ↑
shuttle	0.0138	0.0157	0.0071 ↑	0.0130	0.0070 ↑	0.0145	**0.0069** ↑
waveform	14.40	14.27	14.26 ↑	14.42	**14.17** ↑	14.23	14.34
wineqw	37.51	34.41	32.99 ↑	33.95	**32.90** ↑	33.56	32.93 ↑
yeast	40.45	43.15	37.68 ↑	40.67	**37.60** ↑	40.10	37.89 ↑

– 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 models can perform well on it. This is for instance the case of the solutions found by using random search for the 'secom' dataset.
– 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 (with 50 different classes) for which a high error variance was observed in all of the optimisation strategies (see Fig. 3 for CV error and Fig. 2 for test error). We believe such difference likely results from a combination of difficulty of the classification task (i.e. high input dimensionality, large number of small classes) and/or an insufficient exploration from the random starting configuration in a very large search space.

Table 4. Best MCPS for each dataset and its test error. MC = Remove instances with missing class. MV = Remove/Replace instances with missing values. N2B = NominalToBinary. NOR = Normalization. DIS = Discretize. FS = Random Feature Selection. BAG = Bagging. BOOST = Boosting.

Dataset	MC	MV	N2B	NOR	DIS	FS	BAG	BOOST	Predictor	Meta-predictor	Error
abalone	•	•	•	•					MLP	RandomCommittee	71.43
amazon	•	•	•			•			SimpleLogistic	RandomSubSpace	26.67
car	•	•	•	•				•	SMO	AdaBoostM1	0.00
cifar10	•					•	•		RandomForest	MultiClassClassifier	52.28
cifar10small	•					•			RandomTree	MultiClassClassifier	54.48
convex	•					•	•	•	RandomForest	AdaBoostM1	18.47
dexter	•							•	DecisionStump	AdaBoostM1	5.00
dorothea	•				•	•			OneR	RandomSubSpace	4.64
germancredit	•	•	•				•		LogisticModelTree	Bagging	23.33
gisette	•								NaiveBayes	LWL	1.95
kddcup09app	•								ZeroR	LWL	1.67
krvskp	•							•	JRip	AdaBoostM1	0.10
madelon	•					•			REPTree	RandomSubSpace	15.64
mnist	•	•	•	•					SMO	LWL	3.28
mnistr	•					•	•		RandomForest	RandomCommittee	52.20
secom	•							•	J48	AdaBoostM1	7.66
semeion	•								NaiveBayes	LWL	3.98
shuttle	•					•	•	•	RandomForest	AdaBoostM1	0.0069
waveform	•	•	•	•		•			SMO	RandomSubSpace	14.00
wineqw	•					•	•	•	RandomForest	AdaBoostM1	32.33
yeast	•					•	•		RandomForest	Bagging	36.40

7 Conclusion and Future Work

We have shown that it is possible to automate the composition of a multicomponent predictive system. Our contribution to Auto-WEKA demonstrates that a lot of time and effort can be saved in the process of building predictive systems.

Extending the search space has led optimisation strategies to find solutions that improve the predictive performance in the majority of cases. Results have indicated that SMBO strategies perform better than random search giving the same time for optimisation in most of the cases. While we have performed optimisation for each of the datasets starting with 25 random seeds, in the future it would be interesting to investigate further how the best performance changes over time while varying the number of starting seeds.

The datasets used in the study have been chosen primarily for the purpose of comparison with previous work [13]. As it turned out, the datasets have been relatively 'clean', hence not requiring much preprocessing. In future work we will focus on datasets that require extensive preprocessing. Furthermore, the use of custom preprocessing chains will allow optimising the data that is being used to build predictive models.

In addition, it would also be valuable to investigate if using a different data partitioning like DPS [25] would make any difference in the optimisation process. We believe that it could have a considerable impact in SMAC strategy since it discards potential poor solutions early in the optimisation process based on the performance on only a few folds. In case the folds used are not representative of the overall data distribution, which as shown in [25] can happen quite often in the case of cross-validation, this can have a detrimental effect on the final solutions found.

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 [24].

References

1. Pyle, D.: Data Preparation for Data Mining. Morgan Kaufmann, San Francisco (1999)
2. Linoff, G.S., Berry, M.J.A.: Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management. Wiley (2011). ISBN: 978-0-470-65093-6
3. Teichmann, E., Demir, E., Chaussalet, T.: Data preparation for clinical data mining to identify patients at risk of readmission. In: IEEE 23rd International Symposium on Computer-Based Medical Systems, pp. 184–189 (2010)
4. Zhao, J., Wang, T.: A general framework for medical data mining. In: Future Information Technology and Management Engineering, pp. 163–165 (2010)
5. Messaoud, I., El Abed, H., Märgner, V., Amiri, H.: 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 (2011)
6. Budka, M., Eastwood, M., Gabrys, B., Kadlec, P., Martin Salvador, M., Schwan, S., Tsakonas, A., Žliobaitė, I.: From sensor readings to predictions: on the process of developing practical soft sensors. In: Blockeel, H., van Leeuwen, M., Vinciotti, V. (eds.) IDA 2014. LNCS, vol. 8819, pp. 49–60. Springer, Heidelberg (2014)
7. Leite, R., Brazdil, P., Vanschoren, J.: Selecting classification algorithms with active testing. In: Perner, P. (ed.) MLDM 2012. LNCS, vol. 7376, pp. 117–131. Springer, Heidelberg (2012)
8. Lemke, C., Gabrys, B.: Meta-learning for time series forecasting and forecast combination. Neurocomputing **73**(10–12), 2006–2016 (2010)
9. MacQuarrie, A., Tsai, C.L.: Regression and Time Series Model Selection. World Scientific (1998). ISBN: 978-981-02-3242-9
10. Bengio, Y.: Gradient-based optimization of hyperparameters. Neural Comput. **12**(8), 1889–1900 (2000)
11. Guo, X.C., Yang, J.H., Wu, C.G., Wang, C.Y., Liang, Y.C.: A novel LS-SVMs hyper-parameter selection based on particle swarm optimization. Neurocomputing **71**, 3211–3215 (2008)
12. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
13. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD, pp. 847–855 (2013)

14. Brochu, E., Cora, V.M., de Freitas, N.: A Tutorial on Bayesian Optimization of Expensive Cost Functions with Application to Active User Modeling and Hierarchical Reinforcement Learning. Technical report, University of British Columbia, Department of Computer Science (2010)
15. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011)
16. Bergstra, J., Bardenet, R., Bengio, Y., Kegl, B.: Algorithms for hyper-parameter optimization. In: Advances in NIPS, vol. 24, pp. 1–9 (2011)
17. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Advances in NIPS, vol. 25, pp. 2960–2968 (2012)
18. Eggensperger, K., Feurer, M., Hutter, F.: Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In: NIPS Workshop on Bayesian Optimization in Theory and Practice, pp. 1–5 (2013)
19. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F.: Methods for improving bayesian optimization for AutoML. In: ICML (2015)
20. Serban, F., Vanschoren, J., Kietz, J.U., Bernstein, A.: A survey of intelligent assistants for data analysis. ACM Comput. Surv. **45**(3), 1–35 (2013)
21. Feurer, M., Springenberg, J.T., Hutter, F.: Using meta-learning to initialize bayesian optimization of hyperparameters. In: Proceedings of the Meta-Learning and Algorithm Selection Workshop at ECAI, pp. 3–10 (2014)
22. Swersky, K., Snoek, J., Adams, R.P.: Multi-task bayesian optimization. In: Advances in NIPS, vol. 26, pp. 2004–2012 (2013)
23. Eggensperger, K., Hutter, F., Hoos, H.H., Leyton-brown, K.: Efficient benchmarking of hyperparameter optimizers via surrogates background: hyperparameter optimization. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence, pp. 1114–1120 (2012)
24. Al-Jubouri, B., Gabrys, B.: Multicriteria approaches for predictive model generation: a comparative experimental study. In: IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making, pp. 64–71 (2014)
25. Budka, M., Gabrys, B.: Density-preserving sampling: robust and efficient alternative to cross-validation for error estimation. IEEE Trans. Neural Netw. Learn. Syst. **24**(1), 22–34 (2013)