Data Pipeline Selection and Optimization

Alexandre Quemy
IBM, Cracow, Poland
Faculty of Computing, Poznan University of Technology, Poznan, Poland
aquemy@pl.ibm.com

ABSTRACT

Data pipelines are known to influence machine learning performances. In this paper, we formulate the data pipeline hyperparameter optimization problem as a standard optimization problem that can be solved by (meta)optimizer. We apply Sequential Model-Based Optimization techniques to demonstrate how it can automatically select and tune preprocessing operators to improve baseline score with a restricted budget. For NLP preprocessing operators, we found that some optimal configurations are optimal for several different algorithms. It suggests that there exist algorithm-independent optimal parameter configurations for some datasets.

1 INTRODUCTION

It is now well accepted that in machine learning, data are as important as algorithms. Algorithms received a lot of interest in hyperparameter tuning methods, that is to say, the art of adjusting parameters that are not dependent on the instance data. Contrarily, dataset generation and preprocessing received little if any interest in hyperparameter tuning. For instance, [6] notices that algorithm hyperparameter tuning is performed in 16 out of 19 selected publications while only 2 publications study the impact of data preprocessing. This can probably be explained by the fact that the research community mainly uses ready-to-consume datasets, hence occulting de facto this problematic. However, in practice, raw data are rarely ready to be consumed and must be transformed by a succession of operations usually referred as *data pipeline*.

There are plenty of reasons for which a data source cannot be used directly. For instance, if there are too many descriptive variables, some feature selection or dimensionality reduction algorithms must be applied. If data are too large, subsample techniques can be used. For imbalanced datasets, oversampling or undersampling may help. One of the most common reasons to modify raw data is missing or incorrect values. The most common approaches to cope with this problem is discarding rows with missing or incorrect data, or imputation, i.e. replacing missing values with estimated values based on the available data. Curating datasets from outliers using statistical techniques s.a. winsorization is very common. Finally, it is worth mentioning that some learning models have intrinsic domain restrictions (e.g. Random Forest cannot directly work on categorical variables). This is handled by encoding variables into suitable variables (e.g. numerical variables for Random Forest).

All those operations introduce bias and their presence or not in a data pipeline may be subject to discussion. The data pipeline depends both on the data source and the algorithm such that there is no universal pipeline that can work for every data source and every algorithm. The data pipeline is usually defined by trial and error using the experience of data scientists and the expert knowledge about the data.

In this paper, we propose to apply state-of-the-art hyperoptimization techniques to select and configure data pipelines. The main contributions can be summarized as follows:

- Showing the impact of data pipeline configuration on the classification accuracy[1].
- Defining the Data Pipeline Selection and Optimization (DPSO) problem.
- Showing that addressing the DPSO using Sequential Model-Based Optimization (SMBO) leads to a significant increase of classification performances, even with a restricted CPU or time budget.
- Defining a measure to quantify how an optimal configuration is specific or independent from the algorithm and showing it returns expected results.

In Section 2, we present the related work on data pipeline optimization and hyperparameter tuning. After introducing the problem in Section 3, we perform two set of experiments: Section 4 demonstrates SMBO capacity to solve the problem while Section 5 focuses on the link between optimal configurations and algorithms. We conclude in Section 6 by discussing the limitations of this preliminary work and outlining future work.

2 RELATED WORK

2.1 Data processing impact

The data preprocessing impact has been evaluated for multiple algorithms and operators. In [6], the authors showed that the accuracy obtained by Neural Network, SVM and Decision Trees are significantly impacted by data scaling, sampling and continuous and categorical coding. A correlation link between under and oversampling is also demonstrated.

In [13], three specific data processing operators has been tested for neural networks. Despite the authors do not provide the results without any data processing, the results show an important accuracy variability between the alternatives, thus implying a data processing impact.

For a more comprehensive view on data processing impact, we refer the reader to [7].

2.2 Optimizing data pipeline

AmazonML uses a sort of collaborative filtering to recommend a data pipeline based on data (meta)attributes and a meta-database about efficient pipelines. eIDA [11] solves a planning problem on top of an exhaustive grid which is unsuitable for practicable problems with a large configuration domain.

In [14], guidelines are used to verify the quality of preprocessed data in *continuous machine learning*, i.e. machine learning models in production and receiving continuously new training data. The control is usually semi-automatic and proposed by tools s.a. SeeDB [16] to automatically generate useful visualization of

[1] The approach remains valid for any problem as long as it consists in maximizing a score. In fact, it is enough to have a quality measure on the processed data.

data relations, or QUDE [18] to control false discoveries. The drawback of those methods is the lack of automation.

Recently, a method using meta-features to estimate the impact of preprocessing operators on model accuracy has been proposed [3]. Meta-features can be general (e.g. number of classes or attributes) or statistical (e.g. entropy, noise to signal ratio). This approach constructs a latent space in which any dataset can be represented. A meta-learner is trained over several different datasets obtained from different raw data and data pipeline. The meta-model is thus able to predict the influence of data pipeline operators on new datasets without training the model and evaluating it using e.g. cross-validation.

In [12], the authors use a genetic algorithm to select a representative sample from the data. The objective is to find representative elements to decrease the learning time and increase the model accuracy. The fitness function to evaluate a sample is the model accuracy and thus, this approach is iterative. This work can be seen as a special case of what is being done in this paper: the sample selector operator being one particular operator to be optimized in the data pipeline.

2.3 Hyperparameter tuning and AutoML

The most basic technique for hyperparameter tuning is a grid search or factorial design which consists in exhaustively testing parameter configuration on a grid. Randomized search might help in increasing the probability of finding a good configuration but in most cases the grid approach is computationally intractable.

Modern parameter tuning techniques are divided into two categories. The first one is model-free techniques such as racing algorithm s.a. F-RACE [5] or iterated local search algorithm s.a. ParamILS [10]. The second one can be grouped under a general framework called Sequential Model-Based Optimization (SMBO) that iterates over fitting models to determine promising but unseen regions of the configuration space [2, 9]. Given a new configuration p_{n+1}, the model aims at predicting the performances of the target algorithm o_{n+1} knowing the history $\{(p_1, o_1), ..., (p_n, o_n)\}$. Among this group of techniques, bayesian techniques s.a. gaussian process models and estimates $\mathbb{P}(o|p)$. Another popular approach is the Tree-structured Parzen Estimator (TPE) that models not only $\mathbb{P}(o|p)$ but also $\mathbb{P}(p)$ to provide better recommendations.

AutoML aims at automating the whole design of machine learning experiment. Current AutoML approaches focus on solving the *combined algorithm selection and hyperparameter optimization* (CASH) problem introduced by Auto-WEKA [15]. This problem is rather high-level as it considers the data pipeline selection and its configuration as part of the algorithm selection phase and the general hyperparameter configuration. For instance, Auto-Sklearn defines pipelines as one feature preprocessing operator and up to three data preprocessing methods [8].

The most popular AutoML frameworks such as Auto-WEKA [15], Auto-sklearn [8] or H2O [2] uses Bayesian optimization to solve CASH. They usually add additional components that we do not consider in this study. For instance, Auto-Sklearn reuses the predictions made at every generation in an ensemble way to improve results and prevent overfitting. It also uses meta-learning [1, 17] to solve coldstart problem: a model has been pre-trained offline over 140 datasets to be able to recommend good initial solutions to CASH on new datasets.

[2] https://www.h2o.ai/

In this paper, we propose to deal specifically with selecting and optimizing the data pipeline to demonstrate the influence of data pipeline on the final results, without configuring the algorithm. We hope this to open the road to more efficient techniques to solve CASH, notably by allowing transfer learning at the pipeline configuration step, in addition to meta-learning across datasets.

3 DPSO PROBLEM

We formulate the Data Pipeline Selection and Optimization (DPSO) problem. Let D be a dataset split into D_{train} and D_{test}. A data pipeline is a sequence of operators with their own configuration, transforming a data source into consumable data for a given algorithm A. Let assume a data pipeline configuration space $\mathcal{P}$. Denote by $\mathcal{L}(P, A, D_{\text{test}})$ the loss algorithm A achieved by a cross-validation on D_{test} transformed by P. The DPSO can be formally written:

Definition 3.1 (Data Pipeline Selection and Optimization (DPSO)).

$$P^* \in \underset{P \in \mathcal{P}}{\operatorname{argmin}}\ \mathcal{L}(P, A, D_{\text{test}}) \qquad \text{(DPSO)}$$

In practice, the training set D_{train} is used to find P, and the test set D_{test} to evaluate the overall performances. DPSO can be seen as a subpart of CASH. CASH agglomerates the pipeline and its configuration into the algorithm selection and hyperparameter optimization. To obtain a solution to CASH, a second optimization step can be performed to find the best hyperparameters to the algorithm A.

4 EXPERIMENTS WITH SMBO

In this section, we apply SMBO to solve the Data Pipeline Selection and Optimization problem.

4.1 Protocol

We created a pipeline *prototype* made of 3 steps: "rebalance" (handling imbalanced dataset), "normalizer" (scaling features), "features" (feature selection or dimension reduction). For each step, we selected few possible concrete operators with a specific configuration space. For instance, for "features", there is the choice between a PCA with keeping 1 to 4 axes, selecting the $k \in \{1, ..., 4\}$ best features according to an ANOVA, or a combination of both. Rebalance step consists in downsampling with Near Miss or Condensed Nearest Neighbour method or oversampling with SMOTE. The normalization gives the choice between a standard scaler, a scaler excluding some points based on a quantile interval, a min-max scaler and a power transformation. Each step can also be skipped, and we call *baseline* pipeline, the pipeline skipping all operations. There is a total of 4750 possible pipeline configurations. For an exhaustive description of the configuration space, we refer the reader to the Supplementary Material[3].

We performed the experiment on 3 datasets: Wine, Iris and Breast[4]. We used 4 classification algorithms: SVM, Random Forest, Neural Network and Decision Tree. A 10-fold cross-validation is used to assess the pipeline performances.

We want to quantify the achievable improvement compared to the baseline, measure how likely it is to improve the baseline w.r.t. the configuration space, determine if SMBO is capable to

[3] https://aquemy.github.io/DOLAP_2019_supplementary_material/

[4] The choice of small datasets is justified by the need to know the optimal score in the search space to effectively evaluate SMBO results. Those results justify the SMBO approach as in practice only a fraction of the search space needs to be explored to drastically improve the score.

improve the baseline score, measure how much and fast SMBO is likely to improve the baseline score with a restricted budget.

We performed an exhaustive search and a search using SMBO with a budget of 100 configurations to explore (about 2% of the configuration space).

4.2 Results

Figure 1 provides the result obtained with Random Forest on Breast. A summary of the results is provided by Table 1. All results being qualitatively similar, the plots are provided in the Supplementary Material. Figure 1, top part, shows that the base-

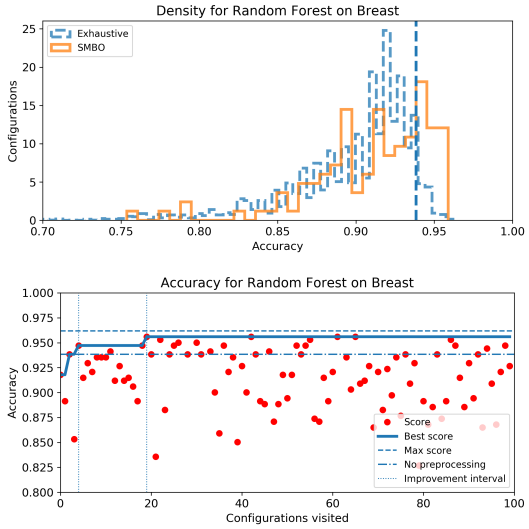

Figure 1: Density of configurations. The vertical line is the baseline score (top). Accuracy with SMBO for 100 configurations explored (bottom).

line score is 0.9384 and the best score 0.9619 i.e. an error reduction of 38% is achievable in the search space. Most configurations deteriorate the baseline score. However, SMBO is skewed towards better configuration compared to the exhaustive search. It indicates SMBO has a better probability to find a good configuration than random search. The bottom part shows that SMBO starts to improve the baseline score after only 4 iterations and reached its best configuration after 19 iterations. There is only one optimal configuration in the search space which is not found. If we normalize the accuracy using the min. and max. on the configuration space, SMBO found a configuration that represents a score of 97.80% with exploring only 0.4% of the configuration space.

Table 1 shows that similar results are obtained for all methods on all datasets. SMBO always found a better configuration than the baseline, in at most 17 iterations. In average, the best score is achieved around 20 iterations (excluding Decision Tree on Iris and Breast). Decision Tree was able to reach the optimal configuration on Iris (resp. Wine) after 1 (resp. 5) iterations. In general, the score in the normalized score space belongs to [0.9780, 1.000]. To summarize, in average, with 20 iterations (0.42% of the search space) SMBO is able to decrease the error by 58.16% compared to the baseline score and found configurations that score 98.92% in the normalized score space.

Figure 2 shows the optimal pipeline in the search space and the four pipelines giving the best score for SMBO. All four pipelines have the correct operator for rebalance and features steps. One

Table 1: Pipeline optimization results.

	Baseline	Exhaustive	SMBO	SMBO (norm.)	Imp. Inter.
			Iris		
SVM	0.9667	0.9889	0.9778	0.9831	[11, 11]
Random Forest	0.9222	0.9778	0.9667	0.9828	[8, 27]
Neural Net	0.9667	0.9889	0.9778	0.9831	[17, 17]
Decision Tree	0.9222	0.9889	0.9889	1.0000	[1, 83]
			Breast		
SVM	0.9501	0.9765	0.9765	1.0000	[12, 20]
Random Forest	0.9384	0.9619	0.9560	0.9780	[4, 19]
Neural Net	0.9326	0.9765	0.9707	0.9903	[1, 7]
Decision Tree	0.9296	0.9619	0.9589	0.9900	[0, 67]
			Wine		
SVM	0.9151	1.0000	0.9906	0.9811	[3, 13]
Random Forest	0.9623	0.9906	0.9811	0.9818	[5, 20]
Neural Net	0.9057	0.9906	0.9906	1.0000	[1, 25]
Decision Tree	0.9057	0.9811	0.9811	1.0000	[5, 35]

The column SMBO (norm.) is the SMBO score normalized within the search space. The last column is the interval where the left bound is the number of configurations required for SMBO to improve the baseline score, and the right, the number of configurations before reaching the best score.

uses the RobustScaler but with an incorrect interval and without centering the data. It is hard to tell which configuration is the closest to the optimal one because there is no obvious metric on the configuration space. However, qualitatively, it seems that the best configurations are relatively similar to the optimal one. As similar results are observed for all methods and datasets, we provided them in the Supplementary Material.

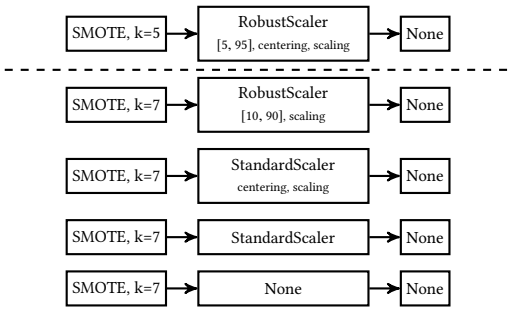

Figure 2: Optimal pipeline (top) and the best pipelines found by SBMO, using Random Forest on Breast.

5 ALGORITHM-SPECIFIC CONFIGURATION

We would like to quantify how much an optimal configuration is specific to an algorithm or is *universal*, i.e. works well regardless of the algorithm. For this, the optimization process might be performed on a collection of methods $\mathcal{A} = \{A_i\}_{i=1}^{N}$. The result is a sample of optimal configurations $\mathbf{p}^* = \{p_i^*\}_{i=1}^{M}$ where $M \geq N$ since an algorithm might have several distinct optimal configurations. After normalizing the configuration space to bring each axis to [0, 1], the link between the processed data and the methods can be studied through the Normalized Mean Absolute Deviation (NMAD). The idea behind this metric is to measure how much the optimal points are distant from a reference optimal point. If the optimal configuration does not depend on the algorithm, the expected distance between the optimal configurations is 0.

Conversely, if a point is specific to an algorithm, the other points will be in average far from it.

Working in the normalized configuration space has two advantages. First, it forces all parameters to have the same impact. Secondly, it allows the comparison from one dataset to another since the NMAD belongs to $[0, 1]$ for any number of algorithms or dimensions of the configuration space.

The Normalized Mean Absolute Deviation is the norm 1 of the Mean Absolute Deviation[5], divided by the number of dimensions K of the configuration space.

Definition 5.1 (Normalized Mean Absolute Deviation (NMAD)).

$$\text{NMAD}(\mathbf{p}^*, r) = \frac{1}{K}\frac{1}{N}||(\sum_{i=1}^{N}|p_i^* - r|)||_1$$

To measure how much each optimal point p_i^* is specific to an algorithm A_j, we use it as a reference point and calculate the NMAD using a sample composed of all the optimal points. However, an algorithm might have several optimal points and to be fair, we use as a representant of each algorithm, the closest point to the reference point.

5.1 Protocol

As the configuration space described in Section 4.1 is not a metric space, we cannot directly use the NMAD. To avoid introducing bias with an *ad-hoc* distance, we perform another experiment with a configuration space that is embedded in $\mathbb{N}$.

We collected 1000 judgements documents provided by the European Court of Human Rights (ECHR) about the Article 6. The database HUDOC[6] provides the ground truth corresponding to a violation or no violation. The cases have been collected such that the dataset is balanced. The conclusion part is removed. To confirm the results, we used a second dataset composed of 855 documents from the categories atheism and religion of 20newsgroups.

Each document is preprocessed using a data pipeline consisting in tokenization, stopwords removal, followed by a n-gram generation. The processed documents are combined and the k top tokens across the corpus are kept, forming the dictionary. Each case is turned into a Bag-of-Words using the dictionary.

There are two hyperparameters in the preprocessing phase: n the size of the n-grams, and k the number of tokens in the dictionary. We defined the parameter configuration domain as follow:

- $n \in \{1, 2, 3, 4, 5\}$,
- $k \in \{10, 100, 1000, 5000, 10000, 50000, 100000\}$.

We used the same four algorithms as in Section 4. As we are interested in the optimal configurations, we performed an exhaustive search.

5.2 Results

For both datasets, Figure 3 shows that the classifier returns poor results for a configuration with a dictionary of only 10 or 100 tokens. Both parameters influence the results, and too high values deteriorate the results.

Table 2 summarizes the best configurations per method. For the first dataset, there are 3 points that gives the optimal value for Random Forest and Linear SVM, however, in practice lowest

[5]As we work on a discrete space, we used the norm 1, but the euclidean norm is probably a better choice in continuous space.
[6]https://hudoc.echr.coe.int/

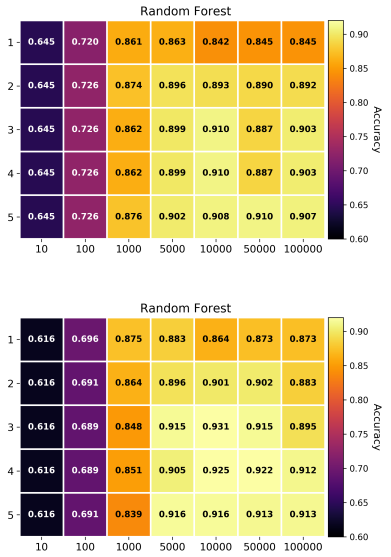

Figure 3: Heatmap depicting the accuracy depending on the pipeline parameter configuration. Top for ECHR, bottom for Newsgroup.

Table 2: Best configurations depending on the method

Method	(n, k)	accuracy
ECHR		
Decision Tree	(5, 50000)	0.900
Neural Network	(5, 50000)	0.960
Random Forest	(3, 10000), (4, 10000), (5, 50000)	0.910
Linear SVM	(3, 50000), (4, 50000), (5, 50000)	0.921
Newsgroup		
Decision Tree	(4, 5000), (4, 100000)	0.889
Neural Network	(5, 50000)	0.953
Random Forest	(3, 10000)	0.931
Linear SVM	(2, 100000)	0.946

parameters values are better because they imply a lower preprocessing and training time. It is interesting to notice that (5, 50000) returns the best accuracy for every model, as this point would be a sort of *universal* configuration for the dataset, taking the best out of the data source, rather than being well suited for a specific algorithm. On the contrary, on Newsgroup, all optimal points are different. Our hypothesis is that the more structured a corpus is, the less algorithm-specific are the optimal configurations, because the preprocessing steps become more important to extract markers used by the algorithms to reach good performances. As ECHR dataset describes standardized justice documents, it is far more structured than Newsgroup. This would also explain why generating n-grams for $n = 5$ still improves the results on ECHR while degrading them on Newsgroup.

This hypothesis is partially confirmed by Table 3, where it is clear that the n-gram operator has a strong impact on the accuracy variation on ECHR dataset (up to 9.8% accuracy improvement) while almost none on Newsgroup dataset (at the exception of Random Forest).
Table 4 contains the NMAD value for each distinct optimal configuration reported in Table 2. The Supplementary Material provides the calculation step by step. As it can be expected, the

Table 3: Impact of parameter n on the accuracy, measured as the relative difference between the best results obtained only using $(1, k)$ and the best results obtained for any configuration (n, k).

Method	$p = (1, k)$	$p = (n, k)$	Δ acc
ECHR			
Decision Tree	0.850	0.900	5.9%
Neural Network	0.874	0.960	9.8%
Random Forest	0.863	0.910	5.4%
Linear SVM	0.892	0.921	6.6%
Newsgroup			
Decision Tree	0.885	0.889	0.5%
Neural Network	0.949	0.953	0.4%
Random Forest	0.883	0.931	5.4%
Linear SVM	0.945	0.946	0.1%

Table 4: Normalized Mean Average Deviation for each optimal configuration found.

ECHR		Newsgroup	
Point	NMAD	Point	NMAD
(5, 50000)	0	(4, 5000)	0.306
(3, 10000)	0.275	(4, 100000)	0.300
(4, 10000)	0.213	(5, 50000)	0.356
(3, 50000)	0.175	(3, 10000)	0.294
(4, 50000)	0.094	(2, 100000)	0.362

point (5, 50000) has a NMAD of 0 since the point is present for every algorithm: (5, 50000) is a *universal* pipeline configuration for this data pipeline and dataset. The point (4, 50000) appears only once but it is really close to (5, 50000) (itself in the 3 other algorithms results) s.t. its NMAD is low. It can be interpreted as belonging to the same area of optimal values. On the opposite, (3, 10000) and (4, 10000) have high NMAD w.r.t. the other points, indicating they are isolated points and may be algorithm specific. Their NMAD values are rather low because despite the points are isolated, they differ significantly from the others points only on the second component. In comparison, if (1, 10) would be an optimal point for Random Forest, its NMAD would be 0.5. On the contrary, for Newsgroup, the NMAD value is rather high and similar for all points, indicating that they are at a similar distance from each other and really algorithm specific.

To summarize, the NMAD metric is coherent with the conclusion drawn from the heatmaps and Table 2, and suggests that there exist two types of optimal configurations: *universal* pipeline configurations that work well on a large range of algorithms for a given dataset, and algorithm-specific configurations. Thus, we are confident the NMAD can be used in larger configuration spaces where heatmaps and exhaustive results are not available for graphical interpretation, and help to reuse configurations.

6 CONCLUSION

In this paper, we successfully applied Sequential Model-Based Optimization techniques to data pipeline selection and configuration. In addition, we provided a metric to study if an optimal configuration is algorithm specific or rather universal.

The main practical drawback of the iterative approach presented in this paper is the cost involved in processing the data and training the model for each selected configuration. To mitigate this problem, we see few possibilities to explore:

- decreasing the amount of data to preprocess using a sample technique as described in [12],
- using in priority data pipelines suggested by a meta-learning algorithm s.a. the one described in [3, 4],
- caching the intermediate results of the data pipeline to reuse, when possible, the outcome of some transformations (e.g. there is no need to regenerate the 2-grams for $n \geq 3$ if a previous configuration with $n = 2$ has been explored.).

Another aspect to be addressed is the compromise between time and performances. Indeed, some parameters increases the processing time but not the model training (e.g. n-grams computation) while others may not affect the processing time but significantly increase the model training (e.g. number of tokens k). A fine grain time analysis would be required, and an intelligent pruning system could be a solution to avoid costly iterations.

Future work should focus on an online version s.t. the pipeline is tuned in a streaming way. Also, the NMAP indicator works only in euclidian spaces which is not the case for the first experiment. Therefore, further work should focus on extending the NMAP to non-vector space.

REFERENCES

[1] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. 2013. Collaborative hyperparameter tuning. In *Int. Conf. Mach. Learn.* 199–207.

[2] J Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. 2011. Algorithms for Hyper-parameter Optimization. In *Proc. Int. Conf. Neural Inf. Process. Syst.* 2546–2554.

[3] B. Bilalli, A. Abelló, and T. Aluja-Banet. 2017. On the Predictive Power of Meta-features in OpenML. *Int. J. Appl. Math. Comput. Sci.* 27, 4 (2017), 697–712.

[4] B. Bilalli, A. Abelló, T. Aluja-Banet, and R. Wrembel. 2018. Intelligent assistance for data pre-processing. *Computer Standards & Interfaces* 57 (2018), 101 – 109.

[5] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. 2010. *F-Race and Iterated F-Race: An Overview.* Springer Berlin Heidelberg, Berlin, Heidelberg, 311–336.

[6] S. F. Crone, S. Lessmann, and R. Stahlbock. 2006. The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. *Eur. J. Oper. Res.* 173, 3 (2006), 781 – 800.

[7] T. Dasu and T. Johnson. 2003. *Exploratory data mining and data cleaning.* Vol. 479. John Wiley & Sons.

[8] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Adv. Neural Inf. Process. Syst.*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). 2962–2970.

[9] F. Hutter, H. H. Hoos, and K. Leyton-Brown. 2011. Sequential Model-based Optimization for General Algorithm Configuration. In *Proc. Int. Conf. Learn. Intel. Optim.* Springer-Verlag, Berlin, Heidelberg, 507–523.

[10] T. Hutter, F. and Hoos, H. H. and Leyton-Brown, K. and Stützle. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *J. Artif. Intel. Res.* 36 (2009), 267–306.

[11] J. Kietz, F. Serban, S. Fischer, and A. Bernstein. 2014. "Semantics Inside!" But Let's Not Tell the Data Miners: Intelligent Support for Data Mining. In *The Semantic Web: Trends and Challenges.* Springer International Publishing, 706–720.

[12] J. Nalepa, M. Myller, S. Piechaczek, K. Hrynczenko, and M. Kawulok. 2018. Genetic Selection of Training Sets for (Not Only) Artificial Neural Networks. In *Proc. Int. Conf. Beyond Databases, Architectures Struct.* 194–206.

[13] N. M. Nawi, W. H. Atomi, and M. Z. Rehman. 2013. The Effect of Data Preprocessing on Optimized Training of Artificial Neural Networks. *Procedia Technology* 11 (2013), 32 – 39. Int. Conf. Elect. Eng. Info.

[14] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. 2017. Data Management Challenges in Production Machine Learning. In *Proc. ACM Int. Conf. Manage. Data.* ACM, 1723–1726.

[15] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Int. Conf. Knowl. Disc. Data Min.* ACM, 847–855.

[16] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. 2015. S ee DB: efficient data-driven visualization recommendations to support visual analytics. *Proc. VLDB Endowment* 8, 13 (2015), 2182–2193.

[17] Dani Yogatama and Gideon Mann. 2014. Efficient transfer learning method for automatic hyperparameter tuning. In *Int. Conf. Artif. Intel. Stat.* 1077–1085.

[18] Z. Zhao, L. De Stefani, E. Zgraggen, C. Binnig, E. Upfal, and T. Kraska. 2017. Controlling False Discoveries During Interactive Data Exploration. In *Proc. ACM Int. Conf. Manag. Data.* ACM, 527–540.