24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Personalized Automated Machine Learning

Cedric Kulbach[1] and **Patrick Philipp**[2] and **Steffen Thoma** [3]

Abstract. Automated Machine Learning (AutoML) is the challenge of finding machine learning models with high predictive performance without the need for specialized data scientists. Existing approaches optimize a pipeline of pre-processing, feature engineering, model selection and hyperparameter optimization, and assume that the user is fully aware of the choice of the underlying metric (such as precision, recall or F1-measure). However, end-users are often unaware of the actual implications of choosing a metric, as the resulting models often significantly vary in their predictions.

In this work, we propose a framework to personalise AutoML for individual end-users by learning a designated ranking model from pairwise user preferences and using the latter as the metric function for state-of-the-art AutoML systems. Given a set of possible metrics, we generate candidate models by repeatedly running AutoML with combinations of the former and have the user choose between pairs of resulting models. We use RankNet to learn a personalized ranking function for the end-user, which is used as loss function for final run of a standard AutoML system.

To evaluate our proposed framework we define three preferences a user could pursue and show that a ranking model is able to learn these preferences from pairwise comparisons. Furthermore, by changing the metric function of AutoML we show that a personalized preference is able to improve machine learning pipelines. We evaluated the ability of learning a personalized preference and the entire framework on several OpenML multi-class classification datasets.

1 Introduction

Machine learning (ML) gains increasing importance in a broad range of applications and has led to an ever-growing demand for machine learning systems [6]. To meet the growing demand for machine learning applications without the need of highly specialized data scientists and domain experts automated machine learning (AutoML) simplifies the selection of suitable features, classifiers and their hyperparameters. Current AutoML frameworks, such as *Auto-Weka 2.0* [16], *autosklearn* [6, 7] or *TPOT* [19] take a dataset $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and a metric function $\mathcal{L}$ as input parameter to train a suitable ML pipeline.

Previous research has either focused on (i) optimizing different steps of the data mining pipeline, e.g. preprocessing [24], feature engineering [15], feature selection [7, 12] and hyperparameter optimization or on (ii) integrating multiple metrics (also referred to as *objectives losses* or *criteria*) in the optimization in order to find the Pareto front [21]. However, it is always assumed that the user is fully aware about the available metrics functions and her/his optimal preference (e.g. combination of metrics).

In addition, approaches for integrating multiple metrics do only cover few, significantly diverse metric functions (e.g. accuracy and execution time), but do not explore sets of metrics which inherently follow the same goal (e.g. optimizing predictive performance with precision, recall, accuracy and F1-measure). It turns out that such metric functions yield significantly diverging predictive performances when evaluated for standard ML benchmarks (e.g. OpenML datasets [25] used in [6]), also leading to different optimal classifiers and hyperparameters.

In this work, we aim to close this gap by learning the user-specific preference from a set of potential candidate metric functions and use the resulting model for optimization in a standard AutoML process. Given a set of metric functions, we first generate candidate weightings of the available metric functions and the resulting sets of learned models from AutoML processes in order to retrieve unlabeled datapoints for interacting with the user. We then initiate a preference learning session with the end-user, where he/she is presented with pairwise choices of possible models. Based on the resulting relative labeling of the dataset, we can learn a ranking function with established approaches such as RankNet [2], which we use for a final AutoML process.

In order to evaluate our approach, we show in a first experiment, that the ranking approach is able to learn a new metric (e.g. a linear combination) from a set of predefined metric functions . The second experiment evaluates the integration of the learned metric function into AutoML and shows that a personalized metric function, such as the metric learned within the ranking model is able to outperform AutoML instances fitted on a metric from the predefined set.

In summary, we provide the following contributions:

1 An approach for personalizing AutoML from a set of metric functions.
2 A formalization, implementation, and evaluation for learning a personalized metric functions from pairwise comparisons.
3 The integration of a ranking model into the environment of AutoML and its evaluation.

After dwelling on related work to our problem setting in Sec 2, we describe the foundations of our approach in Sec 3. Sec 4 then presents our approach to *Personalized AutoML* with components to generate the dataset for user interaction, choosing appropriate pairwise comparisons and learning the novel metric function. We evaluate our approach in terms of preference ranking and eventual AutoML performance in Sec 5. After discussion shortcomings of our approach (Sec 6), we conclude the paper in Sec 7 and point out relevant future directions in Sec 8.

[1] FZI Research Center for Information Technology, Germany, email: kulbach@fzi.de
[2] FZI Research Center for Information Technology, Germany, email: philipp@fzi.de
[3] FZI Research Center for Information Technology, Germany, email: thoma@fzi.de

2 Related work

The field of *human guided machine learning* (HGML) [10] comprises heterogeneous goals, such as enabling end-users to request novel quantities to be computed by the system. Goals of HGML are e.g. to give the user the opportunity to directly interact with a ML system by modifying its features, the instances used to train a model and comparing the performance of different models.

Our work deals with a small fraction of HGML, namely to enable intuitive user interactions for learning a single machine learning model. Here, a ground truth for the dataset is available, but the user preference has to be learned on the basis of multiple available metrics which strongly influence the actual predictions of the resulting models.

AutoML targets to solve the CASH problem which was initially introduced in [23] within the *Auto-WEKA* framework. The most established frameworks for AutoML are *Auto-Weka 2.0* [16], *autosklearn* [6, 13, 8], *TPOT* [19] and *h2o*[4]. While *Auto-Weka 2.0* and *autosklearn* use a random-forests and Gaussian processes [12, *SMAC*], *TPOT* employs an evolutionary algorithm and *h2o* a grid search approach, resulting in different features such as ensembles of different pipelines, as solution or parallel evaluation. However, for solving the CASH problem, the actual metric function $\mathcal{L}$ has to be known in advance [26].

A line of research strongly related to personalising AutoML is enabling to take into account multiple objectives within the search process [21]. The authors find the Pareto front for all objectives and enable end-users to interact with gradual version of the front in order to guide the search. While the goal of weighting the importance of individual objectives is equal to ours, the approach assumes that the end-user is fully aware of all objectives and is able to interpret the Pareto front. Our approach, on the other hand, aims to enable the integration of appropriate user interfaces which provide labeled data for preference learning.

Learning models from (human-) preferences has already been approached for reinforcement learning (RL) [4], where sparsely defined reward functions are replaced by a learned ranking model. The approach also relies on pairwise comparisons of examples (here segments of the process to be learned) by end-users and the eventual reward function is used for standard RL algorithms. Due to the relatively long execution time of an AutoML component the initialization process to generate segments within the reinforcement learning model would lead to excessively long response times for the user to compare two segments. Our approach generates a set of segments within an *Evaluation Initiator* component.

We now introduce the required background concepts for our personalized AutoML approach.

3 Background

Our approach mainly consists of two components, that are seperatly well-researched topics. We first introduce the AutoML problem (see Sec 3.1) and the formalized CASH-problem. Then we describe the preference learning setting (see Sec 3.2), that is needed to personalize AutoML and thus to integrate human preferences into the decision for machine learning pipelines.

[4] http://docs.h2o.ai/h2o, last accessed: 14.06.2019

3.1 Automated Machine Learning

Automated Machine Learning AutoML aims to automate a ML pipeline containing the steps (i) data cleaning, (ii) feature engineering and (iii) modeling algorithm (see. [6]). Following the definition from [26] a machine learning pipeline structure $g \in G$ can be modeled as an arbitrary directed acyclic graph (DAG), where each node represents a algorithm type $A_{cleaning}, A_{feature}$ and A_{model} from each step. The problem to find a suitable combination of the presented algorithm steps can be modelled as Combined Algorithm Selection and Hyperparameter optimization (*CASH*) problem [23].

With the definition of the CASH problem it emerges that the metric is predetermined.

Definition 1 *CASH problem, adopted from [6].*
Let $\mathcal{A} = \{A^{(1)}, \ldots, A^{(R)}\}$ be a set of step independent algorithms, and let the hyperparameters of each algorithm $A^{(j)}$ have a domain $\Lambda^{(j)}$. Further, let D_{train} be a training and D_{valid} be a validation set, which is split into K cross-validation folds $\{D^{(1)}_{train}, \ldots, D^{(K)}_{train}\}$ and $\{D^{(1)}_{valid}, \ldots, D^{(K)}_{valid}\}$. Let $\mathcal{L}(\mathcal{P}_{g,\overrightarrow{A},\overrightarrow{\lambda}}(D^{(i)}_{train}), D^{(i)}_{valid})$ denote the metric that algorithm combination $P^{(j)}$ achieves on $D^{(i)}_{valid}$ when trained on $D^{(i)}_{train}$ with hyperparameters $\overrightarrow{\lambda}$. Then the CASH problem is to find the joint algorithm combination and hyperparameter setting that minimizes the metric:

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

3.2 Preference Learning

Learning to rank is a well-researched topic which can be categorised into pointwise [5, 17], pairwise [11, 2, 18, 22] and listwise [3, 14] approaches.

The pointwise approach assumes that each training document or observation with features X is associated with a rating and thus the problem can be reduced to a regression problem. Prominent examples are PRank [5] which performs the ranking by an ordinal regression and McRank [17] which employs multiclass classification and Gradient boosting techniques. Pointwise approaches are, however, not advantageous for our problem setting, as it is difficult for end-users to assign an absolute target value for a machine learning model.

The listwise approach uses a list of ranked documents or observations for training and learns to predict the order of a list. Exemplary listwise approaches are Combined Regression and Ranking [22], PolyRank [18] and ES Rank [14]. While listwise approaches enable end-users to provide relative feedback for lists of machine learning models, it is too difficult to rank more than two models when the underlying personalisation objective is too complex.

We thus base our work on pairwise ranking approaches which take ranked document pairs as input and learn to classify each document or object pair into correctly ranked or incorrectly ranked classes. Ranking SVM [11] uses a support vector machine to maximize the span between classes correctly and incorrectly ranked to solve the pair wise ranking problem. RankNet [2] uses neural networks to on the one hand learn the rating for both documents or observations in a pair within a siamese architecture and on the other hand to maximize the cross entropy between the predicted ratings (see Sec 5.1). LambdaRank [1] extends RankNet by not using the actual costs (number of

inversions in ranking) but their gradients for training. Lastly, LambdaMART combines LambdaRank and multiple additive regression trees [9].

4 Personalized AutoML

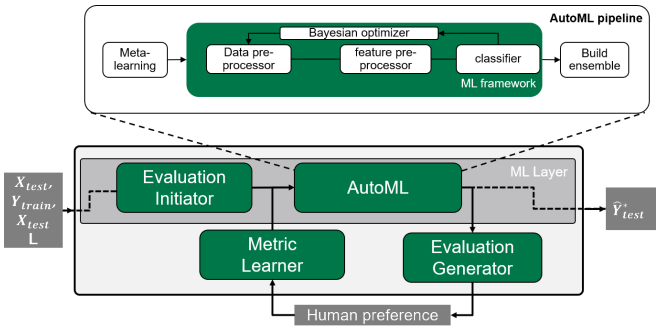

Figure 1. Personalized AutoML framework

In this section we first formalize our novel learning problem and then present our approach to a *Personalized Automated Machine Learning* framework.

Our baseline setting is similar to the prior defined CASH problem (see Def 1), where we assume the availability of algorithms with hyperparameters as well as training and test data sets for calculating a metric signal. However, instead of assuming a single metric function $\mathcal{L}$, we now assume a set of available metric functions $\mathbf{L}$, from which we need to learn an approximation for the optimal combined metric function $\mathcal{L}^*$ for a specific user. We now formally define the personalized CASH problem for AutoML in Def 2.

Definition 2 *Personalized CASH problem.*
We extend the CASH problem where algorithms $\mathcal{A}$ and metric function $\mathcal{L}$ are assumed to be given (see Def 1) with a set of available loss functions $\mathbf{L} = \{\mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(l)}\}$. The goal becomes to learn a novel metric function $\mathcal{L}^*$ which combines the individual metric functions:

$$\mathcal{L}^* : \mathcal{L}^{(1)} \times \cdots \times \mathcal{L}^{(l)} \to \mathbb{R}$$

By assuming that one can model a function $\phi : \mathbf{L} \to \mathbb{R}^l$ which generates a feature vector for the available set of metrics, we can reduce learning $\mathcal{L}$ to a regression problem, where we attempt to learn

$$f_{\mathcal{L}^*} : \phi(\mathbf{L}) \to \mathbb{R}$$

The resulting personalized CASH problem can then be defined as follows.

$$g^*, \overrightarrow{A}^*, \overrightarrow{\lambda}^*, \mathcal{L}^*, \in \underset{P^{(j)} \in \mathcal{P}, \lambda \in \Lambda^{(j)}}{\arg\min} \frac{1}{K} \sum_{i=1}^{K} \hat{f}_{\mathcal{L}^*}(\mathcal{P}_{g,\overrightarrow{A},\overrightarrow{\lambda}}(D^{(i)}_{train}), D^{(i)}_{valid}) \tag{2}$$

where $\hat{f}_{\mathcal{L}^*}$ is the approximated regressor for the novel metric function $\mathcal{L}^*$ for which we we have to learn weights $\theta \in \mathbb{R}^l$, e.g. $\hat{f}_{\mathcal{L}^*} = \phi(\mathbf{L})^T\theta$ for the linear case.

Our proposed framework (see Fig 1) consists of an *Evaluation Initiator*, an *Evaluation Generator* and a *Metric Learner* component which can be paired with any standard AutoML algorithm.

A *Evaluation Initiator* generates a set of pipeline configurations $\mathcal{P}_{g,\overrightarrow{A},\overrightarrow{\lambda}}$ from various AutoML instances trained based on a set of metric functions $\mathbf{L}$.

An *Evaluation Generator* component generates segment pairs with different pipeline configurations $\mathcal{P}_{g,\overrightarrow{A},\overrightarrow{\lambda}}$ which can be evaluated from humans. A *Metric Learner* component learns a new metric function $\mathcal{L}^*$ with given preferences, which is used for a new AutoML component. In a last step the new AutoML component is trained based on the generated metric function $\mathcal{L}^*$ and used to predict on X_{test}.

Besides the overall architecture shown in Fig 2, we describe the overall workflow of our system in Algorithm 1.

Algorithm 1: Personalized AutoML

Input: Dataset $D_{\text{train}} = \{X_{\text{train}}, y_{\text{train}}\}$,
Dataset $D_{\text{valid}} = \{X_{\text{valid}}, y_{\text{valid}}\}$,
Features X_{test},
Set of Metrics L,
Number of pairwise comparisons ω
\\ω is initialized with default value 1000
Output: Prediction $\hat{y}$

\\Generate pipeline configurations P:
Initialize set of pipeline configurations $P \leftarrow \emptyset$
Initialize set of segments $S \leftarrow \emptyset$
for $\mathcal{L} \in \boldsymbol{L}$ **do**
  Fit AutoML($\mathcal{L}, X_{\text{train}}, y_{\text{train}}$)
  $P_{\mathcal{L}} \leftarrow$ Evaluated AutoML pipelines
  $P \leftarrow P \cup P_{\mathcal{L}}$
\\ Generate segment pairs U^{pair} :
$U^{\text{pair}} \leftarrow$ Segment Generation($P, D_{\text{valid}}, \omega$)
  \\ see. Algorithm 2
\\ Get human preference U^{judged} :
$U^{\text{judged}} \leftarrow$ Human Preference(U^{pair}) \\ see Sec 4.3
\\ Train learning to rank see Sec 4.4 :
$X^{(1)}, X^{(-1)} \leftarrow$ Generate RankNet dataset from U^{judged}
$\mathcal{L}^* \leftarrow$ Fit $\text{NN}_{\text{RankNet}}(X^{(1)}, X^{(-1)})$
Fit AutoML($\mathcal{L}^*, X_{\text{train}}, y_{\text{train}}$)
$\hat{y}_{\text{test}} \leftarrow$ Predict AutoML(X_{test}) **Return** $\hat{y}_{\text{test}}$

We first describe the evaluation initiator, which generates a set of candidate models from different AutoML processes.

4.1 Evaluation Initiator

In the initialization phase, we generate different pipeline configurations $\mathcal{P}_{g,\overrightarrow{A},\overrightarrow{\lambda}}$ based on all metrics $\mathcal{L} \in \mathbf{L}$, so that for each $\mathcal{L}^i \in \mathbf{L}$ an AutoML instance is trained on D_{train} and a number of pipeline configurations $\mathcal{P}^{\mathcal{L}^i}_{g,\overrightarrow{A},\overrightarrow{\lambda}}$ exist. All generated configurations are denoted as $\mathcal{P}_{g,\overrightarrow{A},\overrightarrow{\lambda}} = \bigcup_{\mathcal{L}^i \in \mathbf{L}} \mathcal{P}^{\mathcal{L}^i}_{g,\overrightarrow{A},\overrightarrow{\lambda}}$. For easier use the set of pipeline configurations $\mathcal{P}_{g,\overrightarrow{A},\overrightarrow{\lambda}}$ are further denoted as $P = \{P^{(1)}, \ldots, P^{(j)}\} \in \mathcal{P}$. Since not only the best pipeline configurations from the trained AutoML instances, but all pipeline configurations evaluated during AutoML training are used for $\mathcal{P}_{g,\overrightarrow{A},\overrightarrow{\lambda}}$, the number of pipeline configurations is higher than the number of metric functions.

We now describe the evaluation generator, which is called after the initialization phase has yielded a set of candidate models based on different AutoML processes.

4.2 Evaluation Generator

The *Evaluation Generator* component receives a set of pipeline configurations as input from the first AutoML instances a max number of segments pairs (γ) to be generated and a validation dataset D_{valid}. Furthermore, we denote a segment as $s = (X, y, \hat{y}^{(i)})$ as a tuple of features X, its ground truth labels y and a prediction $\hat{y}$ on X. With X_{valid} and each $P^{(i)} \in P$ we predict $\hat{y}^{(i)}_{\text{valid}}$. Based on the predictions we generate for each $P^{(i)} \in P$ a segment $s^{(i)} = (X_{\text{valid}}, y_{\text{valid}}, \hat{y}^{(i)}_{\text{valid}})$ containing the features X_{valid}, the ground truth labels $y^{(i)}_{\text{valid}}$ and the predicted labels $\hat{y}^{(i)}_{\text{valid}}$. Within the *Evaluation Generator* component the question which Segments to compare for Ranking also takes part. Since we aim to implement a pairwise learning to rank approach the selection which segment pairs U^{pair} to evaluate becomes apparent. Algorithm 2 shows a random generation of segment pairs which are parsed to the *human preferences* component. In Algorithm 1 the *Evaluation Generator* is executed in line 9. In algorithm 2 the maximal number of segment pairs is given by all possible combinations of $P^{(i)} \in P$, without taking into account the order of segments within the segment pairs U^{pair}.

Algorithm 2: Segment Generation

Input: Set of pipeline configurations P,
Number of segment pairs to be generated γ
Validation dataset $D_{\text{valid}} = \{X_{\text{valid}}, y_{\text{valid}}\}$
Output: Set of segment pairs U

```
Initialize U^pair = ∅
Initialize ŷ = ∅
for p_i ∈ P do
    ŷ ← p_i(X_valid)
while |U^pair| ≤ γ do
    i, j ← Select random i, j ∈ P
    s_i ← (X_valid, y_valid, ŷ_i)
    s_j ← (X_valid, y_valid, ŷ_j)
    if (s_i, s_j) ∉ U^pair then
        U^pair = U^pair ∪ {(s_i, s_j)}
Return U^pair
```

4.3 Human Preference Interface

Based on the generated segment pairs U^{pair}, within the *Segment Generation* component a *Human Preference* component visualizes the segment pairs $(s_i, s_j) \in U^{\text{pair}}$ so that the user is able to judge which segment is preferred to the other one. Based on the tuples $(X, y, \hat{y})$ the component can build visualizations such as a confusion matrix, the predictions or metric functions which allow the user to express her/his preferences. Due to the need to have different visualizations to evaluate segments for different users, the visualization of the segments represents an own field of research we do not discuss. Instead we provide in our approach an interface to (i) retrieve segment pairs and (ii) to send judgements from the user to the *Metric Learner* component. In Fig 2 we provide an exemplary visualization of segment pairs, where the user could decide whether to choose the left or right segment. All judgments from the user are stored in U^{judged} as triples (s_i, s_j, c), where $c \in \{-1, 1\}$ denotes the judgment whether the left segment (-1) or the right segment (1) was chosen and parsed to the *Metric Learner* component. In Algorithm 1 the *Human Preference* component is executed in line 11.

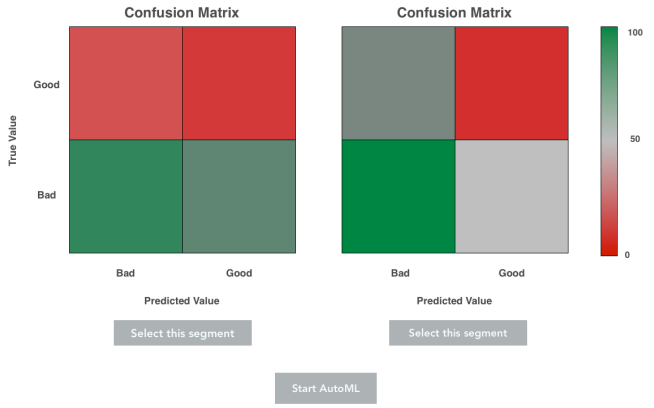

Figure 2. Interface for selecting preferences

4.4 Metric Learner

The *Metric Learner* component is the core component to personalize AutoML. It gets a set of judgments U^{judged} as input to generate a new, personalized metric function $\mathcal{L}^*$. While each segment $s^i \in U^{\text{judged}}$ is related to a pipeline configuration $P^{(i)} \in P$, we can compute for all $P^{(i)} \in P$ all metrics $\mathcal{L} \in \mathbf{L}$ based on the dataset $D = \{X_{\text{valid}}, y_{\text{valid}}\}$ which are already used to visualize the pipeline configuration to a user within the *Human Preference* component. The calculated metrics and the judgment from the user for each segment can be used for a pairwise learning to rank approach. The *Metrics Learner* component generates from the metrics a dataset which can be used to learn a new metric function. We implemented the RankNet[2] algorithm to learn the underlying ranking function. In Algorithm 1 the *Metric Learner* component is executed from line 12 to 14.

The RankNet approach takes two training datasets $X^{(1)}_{\text{train}}$ for the selected segments and $X^{(-1)}_{\text{train}}$ for the non-selected segments from all judgments U^{judged}. Both datasets have the same shape. The base network from the RankNet approach is a deep convolutional neural network (CNN) where the softmax function at the end is removed and replaced with a single neuron (see. [2]). This output is used as a scoring function to rank documents. The meta network has a siamese architecture and predicts within two base networks the scores on $X^{(1)}_{\text{train}}$ and $X^{(-1)}_{\text{train}}$. The difference between both scores is parsed to a sigmoid function which is used to distinguish if the predicted scores provide a correct ranking within $X^{(1)}_{\text{train}}$ and $X^{(-1)}_{\text{train}}$.

The *Metric Learner* component generates for each $P^{(i)} \in P$ all metrics $\mathcal{L} \in \mathbf{L}$ and uses them as features ($X^{(1)}_{\text{train}}$ or $X^{(-1)}_{\text{train}}$) to generate a training dataset for the RankNet approach. Considering e.g. a judgement $(s_1, s_2, -1)$ we calculate for s_1 $L_1 = \{\mathcal{L}^{(1)}_1, \ldots, \mathcal{L}^{(l)}_1\}$ and for s_2 all metrics $L_2 = \{\mathcal{L}^{(1)}_2, \ldots, \mathcal{L}^{(l)}_2\}$. From the human preference we know, that s_1 is ranked lower than s_2, so that the generated features L_2 are added to $X^{(1)}_{\text{train}}$ and the generated features L_1 are added to $X^{(-1)}_{\text{train}}$. With the resulting dataset $X^{(1)}_{\text{train}}$ and $X^{(-1)}_{\text{train}}$ from all

judgements U^{judged} we are able to train a RankNet instance, were the base network is used as a new metric function $\mathcal{L}^*$ for a new AutoML instance that takes the human preference into account. In algorithm 3, we show how the learning to rank based metric function calculates the metric with a similar input to common metric functions e.g. $\mathcal{L}_{\text{accuracy}}(y, \hat{y}_{\text{valid}})$ or Def 1. With this approach we are now able to generate a metric function $\mathcal{L}^*$ as prediction from a trained RankNet model (see. algorithm 3). Based on the new personalized metric function $\mathcal{L}^*$ a new AutoML instance can be fitted on D_{train} and evaluated on D_{test}.

Algorithm 3: Rank based Metric function $\mathcal{L}^*$

Input: Ground truth y, and prediction $\hat{y}$
Trained RankNet model $\text{NN}_{\text{RankNet}}$
Set of metric functions $L = \{\mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(l)}\}$
Output: Metric$\mathcal{L}^*(x)$

Initialize $X \leftarrow \emptyset$
Initialize $\mathcal{L}^* \leftarrow \text{NN}_{\text{RankNet}}$ **for** $\mathcal{L}^{(i)} \in \boldsymbol{L}$ **do**
  $x \leftarrow x \cup \mathcal{L}^{(i)}(\hat{y}, y)$
Return $\mathcal{L}^*(x)$

5 Empirical Evaluation

In this section we evaluate our proposed approach. The first part of the evaluation refers to the *Metric Learner* component and its RankNet approach described in Sec 4.4. The second part refers to the evaluation of the whole framework. We show that our approach of a personalized AutoML is able to take user preferences into account and suggests AutoML classifiers that can outperform previous AutoML instances based on the preferences a user wants to pursue.

To evaluate our approach, we used the metric functions *explained variance score*, *f1 score*, *hamming loss*, *jaccard score*, *log loss*, *max error*, *mean squared error*, *precision score*, and *recall score* from the *scikit-learn* library [20] and trained for each metric and dataset an AutoML instance with TPOT [19]. We evaluated our approach on various established classification datasets e.g. already used by Feurer et. al. [6]. As AutoML framework we used *TPOT* [19] as component to create the pipeline configurations P. For each metric function $\mathcal{L} \in \mathbf{L}$ we fitted an AutoML instance for one hour and extracted all evaluated pipeline configurations. The execution of the first AutoML instances only serves to generate segments, whereby the evaluation time for each pipeline configuration depends on the size of the training dataset X_{train}. Within one hour of training the AutoML instances we generated on small dataset e.g. OpenML 38 with 30 features and 3772 instances 19452 segments in total. On a larger dataset e.g. OpenML 179 with 15 features and 48842 instances we generated 2349 segments in total. Based on the generated segments and predefined preferences we evaluate in Sec 5.1 the chosen learning to rank approach and in Sec 5.2 its integration into a new AutoML instance.

5.1 Metric Learner Evaluation

We introduced the *Metric Generator* component in Sec 4.4 and proposed in Algorithm 3 an approach to use different metrics as features to build a metric function a user wants to pursue. When not only evaluating how capable the RankNet approach is of learning user preferences, but also how many comparisons are needed to learn an adequate model, the *Evaluation Generator* component would also need to be integrated into the evaluation of the RankNet approach. To evaluate the RankNet approach we generated with algorithm 2 and a predefined metric (see. table 1 1250 judgements (see. algorithm 2). We generated a 80/20 (1000/250) train-test split and trained different RankNet models. On the one hand we evaluated how many pairwise comparisons are necessary to learn a suitable metric by varying the training size and using 10, 100, 250, 500 and 1000 judgements from the training dataset for training. Table 1 shows the percentage of correct predicted rankings on the test dataset.

Human Preference	OpenML ID	Training Size				
		10	100	250	500	1000
Accuracy	38	0.913	0.963	0.972	0.975	0.977
	179	0.913	0.945	0.960	0.966	0.967
	772	0.963	0.956	0.966	0.975	0.977
	917	0.943	0.974	0.981	0.984	0.986
	1049	0.890	0.966	0.977	0.978	0.979
	1120	0.917	0.958	0.967	0.971	0.973
Linear Combination	38	0.896	0.935	0.935	0.935	0.935
	179	0.791	0.825	0.810	0.810	0.805
	772	0.593	0.687	0.888	0.933	0.937
	917	0.949	0.965	0.968	0.968	0.969
	1049	0.861	0.943	0.957	0.967	0.972
	1120	0.913	0.961	0.968	0.974	0.979
F1	38	0.937	0.972	0.971	0.977	0.978
	179	0.889	0.931	0.961	0.974	0.977
	772	0.876	0.966	0.972	0.972	0.974
	917	0.964	0.982	0.982	0.983	0.985
	1049	0.883	0.932	0.970	0.980	0.981
	1120	0.940	0.951	0.964	0.977	0.990

Table 1. RankNet - Percentage of correct predictions on test judgements

5.1.1 Accuracy

With the first experiment we show, that RankNet is able to learn a new metric where the metric to pursue is in the set of metric functions $\mathcal{L} \in \mathbf{L}$. We used the set of metric function introduced in Sec 5 and used the accuracy score from the *sklearn* library as preference to learn. It shows, that a RankNet is able to learn a metric function that is already in the set of features within 100 pairwise judgements.

5.1.2 Linear Combination

The second experiment shows that RankNet is able to learn a linear combination of metric functions. As in the first experiment (see Sec 5.1.1) we used the metrics introduced in Sec 5. We implemented a preference that pursues a linear combination, where each metric is weighted and the sum of all weights is equal to 1. Taking a closer look into the chosen metrics functions we also used metric functions that exceeds a definition range between 0 and 1 (e.g. mean squared error), so that the resulting metric from the preference could also exceed the range of 0 to 1. Comparing the results from the linear combination with the results from the first experiment, the results are slightly worse than when only the accuracy needs to be determined and in general there are more judgements needed to learn a new metric function (see OpenML dataset 179). This task is much more complex since the chosen metric functions are not normalized and few metric functions are interdependent (see Sec 5.1.3).

5.1.3 F1

With the third experiment we show, that the RankNet approach is able to learn even more complex metrics. As in the previous experiments we used the metrics from the *scikit-learn* library (see Sec 5), but without the f1 score. We implemented a preference that pursues the f1 score, which can be expressed as function of the precision and recall metric:

$$f(\text{precision}, \text{recall}) = 2\frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} \quad (3)$$

By using the metrics from Sec 5 (without the f1 score) we show similar to the first experiment, that the RankNet approach is able to select the important metrics functions (precision and recall) as features. Furthermore, we show that RankNet is able to learn a complex (non-linear) metric. The RankNet approach achieves within 500 judgements results, that surpass 0.90.

When evaluating the *Metric Learner* component and describing the possibilities of a pairwise learning to rank approach it is also important to define the limitations and which functions cannot be learned by the *Metric Learner* component. The limitations can be depicted with the features chosen for the RankNet approach. For example, learning a metric function that a user wants to pursue requires that it the desired metric functions is describable by a combination of the chosen features. In our evaluation for example it should be expressible by the chosen features *explained variance score*, *f1 score*, *hamming loss*, *jaccard score*, *log loss*, *max error*, *mean squared error*, *precision score* and *recall score* from the *scikit-learn* library.

5.2 Personalized AutoML Evaluation

After evaluating the *Metric Learner* component, we show in this section that the learned RankNet metric function is capable to work as metric for AutoML. First we trained different RankNet Models (see Sec 5.1) with 1000 judgements. The judgmenets were generated from the *Evaluation Generator* component and three different human preferences (accuracy - Sec 5.1.1, linear combination of metrics - Sec 5.1.2 and f1-score - Sec 5.1.3). We used the segments generated in the initially executed AutoML instances.

The generated RankNet models (see Algorithm 3) for the accuracy score, the linear combination and the f1-score are used to fit new AutoML instances on D_{train}. To evaluate the RankNet model we trained for each dataset another AutoML instance on D_{train} and with the metric a user could pursue (accuracy score, linear combination of metrics and f1-score). This leads to an AutoML instance, where the user already exactly knows how to measure the performance of different pipeline configurations. In figures 3 and 4 we plotted the last 50 segments (pipeline configurations evaluated on D_{valid}) for the AutoML instance fitted on D_{train} and the metrics precision, recall and the RankNet metric fitted with the f1-score preference. Additionally we plotted various f1-scores as function of precision and recall, where a higher color intensity means a better f1-score. In Fig 3 we see that the AutoML instance fitted on the recall metric leads to higher recall scores, and the AutoML instance fitted on the precision metric leads to higher precision scores. Would the users preference be the f1-score, but fit the AutoML instance on the precision or recall metric, the AutoML instance would use the pipeline configuration which performs best on the y-axis in case of the recall metric and on the x-axis in case of the precision metric. Using the RankNet model to train a new AutoML instance leads in Fig 3 to slightly better results than only fitting AutoML on precision or recall. In Fig 4 the results for dataset 1049 are clearer. It shows that when an AutoML instance is fitted on a metric a user only wants to pursue partially (see. Equation 3) the AutoML instance fitted on a metric the user defined by pairwise comparisons outperforms the other AutoML instances.

In a next experiment we integrated the *Metric Learner* component into our approach and trained based on different preferences different metric functions $\mathcal{L}^*$. We compared in Table 2 the performance of an AutoML instance fitted on the RankNet metric and of on an AutoML instance fitted on the metric the judgments of the RankNet model were built. In Table 2 we evaluated our approach to a personalized AutoML. We fitted AutoML instances based on the datasets already used in Sec 5.1 and on different metrics. For each dataset and each preference (accuracy, linear combination of metrics and f1-score) we fitted one AutoML instance directly with the preference and one AutoML instance with the learned RankNet metric from the preference. We evaluated the fitted instances on the chosen preference and on D_{test}. The aim of this experiment is to get as close as possible to the preference of the user.

Preference	**OpenML ID**	**AutoML Metric**	
		Preference	RankNet
Accuracy	38	1.00	0.874
	179	0.710	0.611
	772	0.714	0.686
	917	1.00	0.984
	1049	0.774	0.618
	1120	0.922	0.899
Linear Combination	38	0.790	0.951
	179	2.207	1.100
	772	0.481	0.474
	917	4.861	3.395
	1049	6.261	2.023
	1120	1.692	1.067
F1	38	0.971	0.966
	179	0.706	0.700
	772	0.793	0.754
	917	1.000	0.978
	1049	0.978	0.733
	1120	0.984	0.963

Table 2. Personalized AutoML Evaluation

5.2.1 Accuracy

Table 2 shows that our proposed approach is able to almost manage to get close to the accuracy performance an AutoML instance achieves that is directly fitted on the preference. But nevertheless, it turns out that using the RankNet approach to predict the accuracy metric results in some cases (e.g. datasets 38 and 1049) in significantly worse accuracy scores. The AutoML instance fitted on the accuracy preference performs on the preference metric on average 7.4% better than the AutoML instance fitted on the RankNet metric.

5.2.2 Linear Combination

It becomes more difficult to evaluate the performance of a linear combination of metrics. We chose metric functions that are not suitable for classification tasks and exceed a definition range between 0 and 1 (e.g. mean squared error). When choosing a linear combination

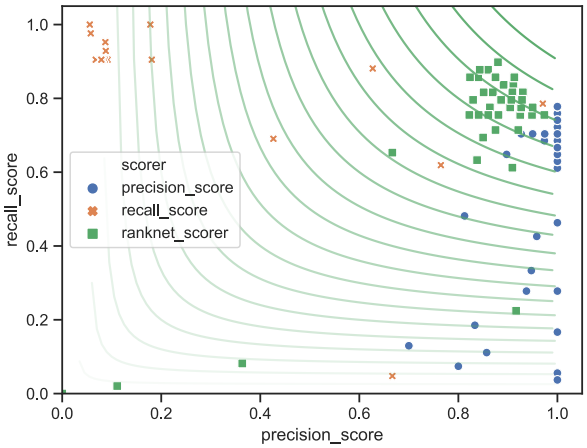

Figure 3. AutoML instances on OpenML 38 - Precision, Recall curve

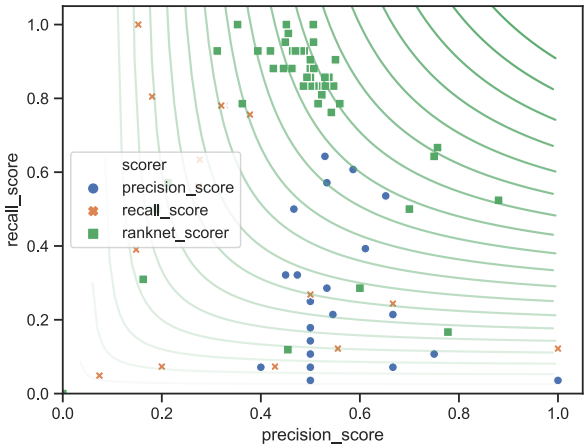

Figure 4. AutoML instances on OpenML 1049 - Precision, Recall curve

of metrics that exceed a range between 0 and 1, the resulting metric also exceeds that range and makes it difficult to compare the results in terms of the results are close to each other or not. Nevertheless, this experiment also shows the difficulty of evaluating personalized metrics with each other and for the sake of completeness, we evaluated this case anyway.

5.2.3 F1

In a last experiment we evaluated the f1-score as metric a user wants to pursue. Already in the evaluation of the *Metric Learner* component we showed, that the percentage of correct predictions for the RankNet metric reached in few judgements results over 0.90%. This result is reflected in the performance of the AutoML instances. In this experiment the AutoML instance fitted on the f1 preference performs on the preference metric on average 5.6% better than the AutoML instance fitted on the RankNet metric.

6 Discussion

We evaluated our personalized AutoML framework based on synthetic pairwise comparisons, i.e. without actual human feedback. A central motivation for synthetic experiments was to empirically prove that learned metrics can be utilized for improving the AutoML process with respect to pre-defined targets. As the utility of the framework has to be measured in terms of actual added value for end-users, a human evaluation is an important next step. Since a human evaluation is dependent on an adequate interface (as sketched in Fig 2), it is essential to first develop and evaluate such an interface on its own.

A further limitation of our approach is the dependency of the learned metric function on $\mathbf{L}$, as it constraints the personalization to weighted combinations of individual metrics. However, the set of available metrics is easily extensible, where custom metrics (e.g. from previous personalization processes) can be directly reused.

7 Conclusion

An approach for a personalized automated machine learning approach was developed. We extended the CASH problem to the possibility of integrating human preferences into the search for machine learning pipelines. To enable the integration of human preferences we showed that a pairwise learning to rank approach is able to learn a new metric $\mathcal{L}^*$ by using a set of metrics L as features. However, the approach to learn a personalized metric function $\mathcal{L}^*$ is limited by the number of features (L). We evaluated within the *Metric Learner* component the pairwise learning to rank approach on different metrics and on several OpenML datasets. Within the *Human Preference* component we proposed a first visualization draft. Furthermore, we integrated the pairwise learning to rank approach into the AutoML pipeline building process (CASH problem) and evaluated the framework against 3 generated preferences on several OpenML datasets. We showed that our approach for a personalized AutoML is able to outperform an AutoML instance fitted on different metric functions. Furthermore, we published our approach on *github*[5].

8 Outlook

A fruitful direction to extend our personalized AutoML system is improving the *Evaluation Initiator*, which generates the model samples which have to be ranked with the help of the user. Here, one could minimize the required resources for initialization by running AutoML processes with weighted metric combinations, but it remains challenging to guarantee that generated models remain sufficiently different for the ranking problem. A possible direction would be to warm-start the initiator based on learned metric functions for previous end-users, which is challenging when user-specific information is unavailable. Such a warm-start would also be directly applicable to the *Metric learner*, which could then converge with fewer samples. Besides initialization, the *Evaluation generator* could be extended in the form of active learning in order to gradually choose comparison tasks with higher utility. By choosing smarter comparisons, one

[5] https://github.com/kulbachcedric/personalizing_automl

could reduce the number of required human interaction which would increase usability of the system.

REFERENCES

[1] Christopher J. C. Burges, Robert Ragno, and Quoc Viet Le, 'Learning to rank with nonsmooth cost functions', in *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, eds., Bernhard Schölkopf, John C. Platt, and Thomas Hofmann, pp. 193–200. MIT Press, (2006).

[2] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender, 'Learning to rank using gradient descent', in *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, eds., Luc De Raedt and Stefan Wrobel, volume 119 of *ACM International Conference Proceeding Series*, pp. 89–96. ACM, (2005).

[3] Fatih Çakir, Kun He, Xide Xia, Brian Kulis, and Stan Sclaroff, 'Deep metric learning to rank', in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 1861–1870. Computer Vision Foundation / IEEE, (2019).

[4] Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei, 'Deep reinforcement learning from human preferences', in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, eds., Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, pp. 4299–4307, (2017).

[5] Koby Crammer and Yoram Singer, 'Pranking with ranking', in *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, eds., Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, pp. 641–647. MIT Press, (2001).

[6] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter, 'Efficient and robust automated machine learning', in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, eds., Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, pp. 2962–2970, (2015).

[7] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter, 'Auto-sklearn: Efficient and robust automated machine learning', In Hutter et al. [13], 113–134.

[8] Matthias Feurer, Benjamin Letham, and Eytan Bakshy, 'Scalable meta-learning for bayesian optimization', *CoRR*, **abs/1802.02219**, (2018).

[9] Jerome H. Friedman and Jacqueline J. Meulman, 'Multiple additive regression trees with application in epidemiology', *Statistics in Medicine*, **22**(9), 1365–1381, (2003).

[10] Yolanda Gil, James Honaker, Shikhar Gupta, Yibo Ma, Vito D'Orazio, Daniel Garijo, Shruti Gadewar, Qifan Yang, and Neda Jahanshad, 'Towards human-guided machine learning', in *Proceedings of the 24th International Conference on Intelligent User Interfaces, IUI 2019, Marina del Ray, CA, USA, March 17-20, 2019*, eds., Wai-Tat Fu, Shimei Pan, Oliver Brdiczka, Polo Chau, and Gaelle Calvary, pp. 614–624. ACM, (2019).

[11] Thore Graepel, Ralf Herbrich, and Klaus Obermayer, 'Bayesian transduction', in *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, eds., Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, pp. 456–462. The MIT Press, (1999).

[12] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown, 'Sequential model-based optimization for general algorithm configuration', in *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, ed., Carlos A. Coello Coello, volume 6683 of *Lecture Notes in Computer Science*, pp. 507–523. Springer, (2011).

[13] *Automated Machine Learning - Methods, Systems, Challenges*, eds., Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, The Springer Series on Challenges in Machine Learning, Springer, 2019.

[14] Osman Ali Sadek Ibrahim and Dario Landa-Silva, 'Es-rank: evolution strategy learning to rank approach', in *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017*, eds., Ahmed Seffah, Birgit Penzenstadler, Carina Alves, and Xin Peng, pp. 944–950. ACM, (2017).

[15] James Max Kanter and Kalyan Veeramachaneni, 'Deep feature synthesis: Towards automating data science endeavors', in *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Campus des Cordeliers, Paris, France, October 19-21, 2015*, pp. 1–10. IEEE, (2015).

[16] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown, 'Auto-weka 2.0: Automatic model selection and hyperparameter optimization in WEKA', *J. Mach. Learn. Res.*, **18**, 25:1–25:5, (2017).

[17] Ping Li, Christopher J. C. Burges, and Qiang Wu, 'Mcrank: Learning to rank using multiple classification and gradient boosting', in *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, eds., John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, pp. 897–904. Curran Associates, Inc., (2007).

[18] Ivo F. D. Oliveira, Nir Ailon, and Ori Davidov, 'A new and flexible approach to the analysis of paired comparison data', *J. Mach. Learn. Res.*, **19**, 60:1–60:29, (2018).

[19] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore, 'Evaluation of a tree-based pipeline optimization tool for automating data science', in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pp. 485–492, New York, NY, USA, (2016). ACM.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research*, **12**, 2825–2830, (2011).

[21] Florian Pfisterer, Stefan Coors, Janek Thomas, and Bernd Bischl, 'Multi-objective automatic machine learning with autoxgboostmc', *CoRR*, **abs/1908.10796**, (2019).

[22] D. Sculley, 'Combined regression and ranking', in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, eds., Bharat Rao, Balaji Krishnapuram, Andrew Tomkins, and Qiang Yang, pp. 979–988. ACM, (2010).

[23] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown, 'Auto-weka: combined selection and hyperparameter optimization of classification algorithms', in *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, eds., Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy, pp. 847–855. ACM, (2013).

[24] Isabel Valera and Zoubin Ghahramani, 'Automatic discovery of the statistical types of variables in a dataset', in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, eds., Doina Precup and Yee Whye Teh, volume 70 of *Proceedings of Machine Learning Research*, pp. 3521–3529. PMLR, (2017).

[25] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo, 'Openml: Networked science in machine learning', *SIGKDD Explorations*, **15**(2), 49–60, (2013).

[26] Marc-André Zöller and Marco F. Huber, 'Survey on automated machine learning', *CoRR*, **abs/1904.12054**, (2019).