Extreme Algorithm Selection with Dyadic Feature Representation

Alexander Tornede(✉), Marcel Wever, and Eyke Hüllermeier

Heinz Nixdorf Institute and Department of Computer Science, Paderborn University, Warburger Str. 100, 33100 Paderborn, Germany
{alexander.tornede,marcel.wever,eyke}@uni-paderborn.de

Abstract. Algorithm selection (AS) deals with the automatic selection of an algorithm from a fixed set of candidate algorithms most suitable for a specific instance of an algorithmic problem class, e.g., choosing solvers for SAT problems. Benchmark suites for AS usually comprise candidate sets consisting of at most tens of algorithms, whereas in algorithm configuration (AC) and combined algorithm selection and hyperparameter optimization (CASH) problems the number of candidates becomes intractable, impeding to learn effective meta-models and thus requiring costly online performance evaluations. In this paper, we propose the setting of extreme algorithm selection (XAS), which, despite assuming limited time resources and hence excluding online evaluations at prediction time, allows for considering thousands of candidate algorithms and thereby facilitates meta learning. We assess the applicability of state-of-the-art AS techniques to the XAS setting and propose approaches leveraging a dyadic representation, in which both problem instances and algorithms are described in terms of feature vectors. We find this approach to significantly improve over the current state of the art in various metrics.

Keywords: Extreme algorithm selection · Dyadic ranking · Surrogate model

1 Introduction

Algorithm selection (AS) refers to a specific recommendation task, in which the choice alternatives are algorithms: Given a set of candidate algorithms to choose from, and a specific instance of a problem class, such as SAT or integer optimization, the task is to select or recommend an algorithm that appears to be most suitable for that instance, in the sense of performing best in terms of criteria such as runtime, solution quality, etc. Hitherto practical applications of AS, as selecting a SAT solver for a logical formula, typically comprise candidate sets consisting of at most tens of algorithms, and this is also the order of magnitude that is found in standard AS benchmark suites such as ASlib [2].

This is in contrast with the problem of combined algorithm selection and hyperparameter optimization (CASH) [24] as considered in automated machine

A. Appice et al. (Eds.): DS 2020, LNAI 12323, pp. 309–324, 2020.
https://doi.org/10.1007/978-3-030-61527-7_21

learning (AutoML), where the number of potential candidates is very large and potentially infinite [6,16,24]. Corresponding methods heavily rely on computationally extensive search procedures combined with costly online evaluations of the performance measure to optimize for, since learning effective meta models for an instantaneous recommendation becomes infeasible.

In this paper, we propose *extreme algorithm selection* (XAS) as a novel setting in-between traditional AS and AC/CASH, which is motivated by application scenarios characterized by

– the demand for prompt recommendations in quasi real time,
– an extremely large (though still finite) set of candidate algorithms.

An example is the scenario of "On-the-fly computing" [10], including "On-the-fly machine learning" [17] as one of its instantiations, where users can request online (machine learning) software services customized towards their needs. Here, users are unwilling to wait for several hours until their service is ready, but rather claim a result quickly. Hence, for providing a first version of an appropriate service, costly search and online evaluations are not affordable. As will be seen, XAS offers a good compromise solution: Although it allows for the consideration of extremely many candidate solutions, and even offers the ability to recommend configurations that have never been encountered so far, it is still amenable to AS techniques and avoids costly online evaluations.

In a sense, XAS relates to standard AS as the emerging topic of extreme classification (XC) [1] relates to standard multi-class classification. Similar to XC, the problem of learning from sparse data is a major challenge for XAS: For a single algorithm, there are typically only observations for a few instances. In this paper, we propose a benchmark dataset for XAS and investigate the ability of state-of-the-art AS approaches to deal with this sparsity and to scale with the size of candidate sets. Furthermore, to support more effective learning from sparse data, we propose methods based on "dyadic" feature representations, in which both problem instances and algorithms are represented in terms of feature vectors. In an extensive experimental study, we find these methods to yield significant improvements.

2 From Standard to Extreme Algorithm Selection

In the standard (per-instance) algorithm selection setting, first introduced in [20], we are interested in finding a mapping $s : \mathcal{I} \longrightarrow \mathcal{A}$, called algorithm selector. Given an instance i from the instance space $\mathcal{I}$, the latter selects the algorithm a^* from a set of candidate algorithms $\mathcal{A}$, optimizing a performance measure $m : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R}$. Furthermore, m is usually costly to evaluate. The optimal selector is called *oracle* and is defined as

$$s^*(i) := \arg\max_{a \in \mathcal{A}} \mathbb{E}\left[\, m(i,a) \,\right] \qquad (1)$$

for all $i \in \mathcal{I}$. The expectation operator $\mathbb{E}$ accounts for any randomness in the application of the algorithm—in the non-deterministic case, the result of applying a to i, and hence the values of the performance measure, are random variables.

Most AS approaches leverage machine learning techniques, in one way or another learning a surrogate (regression) model $\widehat{m} : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R}$, which is fast to evaluate and thus allows one to compute a selector $\widehat{s} : \mathcal{I} \longrightarrow \mathcal{A}$ by

$$\widehat{s}(i) := \arg\max_{a \in \mathcal{A}} \widehat{m}(i, a) \,. \tag{2}$$

In order to infer such a model, we usually assume the existence of a set of training instances $\mathcal{I}_D \subset \mathcal{I}$ for which we have instantaneous access to the associated performances of some or often all algorithms in $\mathcal{A}$ according to m.

The XAS setting distinguishes itself from the standard AS setting by two important properties. Firstly, we assume that the set of candidate algorithms $\mathcal{A}$ is *extremely* large. Thus, approaches need to be able to scale well with the size of $\mathcal{A}$. Secondly, due to the size of $\mathcal{A}$, we can no longer reasonably assume to have evaluations for each algorithm on each training instance. Instead, we assume that the training matrix spanned by the training instances and algorithms is only sparsely filled. In fact, we might even have algorithms without any evaluations at all. Hence, suitable approaches need to be able to learn from very few data and to tackle the problem of "zero-shot learning" [29].

Similarly, the XAS setting differs from the AC and CASH settings in two main points. Firstly, dealing with real-valued hyperparameters, the set of (configured) algorithms $\mathcal{A}$ is generally assumed to be *infinite* in both AC and CASH, whereas $\mathcal{A}$ is still finite (even if extremely large) in XAS. More importantly, in both AC and CASH, one usually assumes having time to perform online evaluations of solution candidates at recommendation time. However, as previously mentioned, this is not the case in XAS, where instantaneous recommendations are required. Hence, the XAS setting significantly differs from the AS, AC, and CASH settings. A summary of the main characteristics of these settings is provided in Table 1.

Table 1. Overview of the characteristics of the problem settings we distinguish.

Characteristics/Setting	AS	XAS	AC	CASH
Size of $\mathcal{A}$	at most tens	extremely many	potentially infinite	potentially infinite
Training data	complete	sparse	mostly not present	mostly not present
Online evaluations	no	no	yes	yes

3 Exploiting Instance Features

Instance-specific AS is based on the assumption that instances can be represented in terms of feature information. For this purpose, $f_I : \mathcal{I} \longrightarrow \mathbb{R}^k$ denotes a function representing instances as k-dimensional, real-valued feature vectors, which can be used to learn a surrogate model (2). This can be done based on different types of data and using different loss functions.

3.1 Regression

The most common approach is to tackle AS as a regression problem, i.e., to construct a regression dataset for each algorithm, where entries consist of an instance representation and the associated performance of the algorithm at question. Accordingly, the dataset associated with algorithm $a \in \mathcal{A}$ consists of tuples of the form $\big(f_I(i), m(i,a)\big)$, created for those instances $i \in \mathcal{I}_D$ to which a has been applied, so that a performance evaluation $m(i,a) \in \mathbb{R}$ is available. Using this dataset, a standard regression model $\widehat{m}_a$ can be learned per algorithm a, and then used as a surrogate. The model can be realized as a neural network or a random forest, and trained on loss functions such as root mean squared or absolute error. For an overview of methods of this kind, we refer to Sect. 6.

This approach has two main disadvantages. Firstly, it is not well suited for the XAS setting, as it requires learning a huge number of surrogate models, one per algorithm. Although these models can usually be trained very quickly, the assumption of sparse training data in the XAS setting requires them to be learned from only a handful of training examples—it is not even uncommon to have algorithms without any performance value at all. Accordingly, the sparser the data, the more drastically this approach drops in performance, as will be seen in the evaluation in Sect. 5. Secondly, it requires precise real-valued evaluations of the measure m as training information, which might be costly to obtain. In this regard, one may also wonder, whether regression is not solving an unnecessarily difficult problem: Eventually, AS is only interested in finding the best algorithm for a given problem instance, or, more generally, in ranking the candidate algorithms in decreasing order of their expected performance. An accurate prediction of absolute performances is a *sufficient* but not a *necessary* condition for doing so.

3.2 Ranking

As an alternative to regression, one may therefore think of tackling AS as a *ranking* problem. More specifically, the counterpart of the regression approach outlined above is called *label ranking* (LR) in the literature [28]. Label ranking deals with learning to rank choice alternatives (referred to as "labels") based on given contexts represented by feature information. In the setting of AS, contexts and labels correspond to instances and algorithms, respectively. The type of training data assumed in LR consists of rankings π_i associated with training instances $i \in \mathcal{I}_D$, that is, order relations of the form $(f_I(i), a_{i,1}) \succ \ldots \succ (f_I(i), a_{i,l_i})$, in which $\succ$ denotes an underlying preference relation; thus, $(f_I(i), a) \succ (f_I(i), a')$ means that, for instance i represented by features $f_I(i)$, algorithm a is preferred to (better than) algorithm a'. If i is clear from the context, we also represent the ranking by $a_1 \succ \ldots \succ a_{l_i}$. Compared to the case of regression, a ranking dataset of this form can be constructed more easily, as it only requires qualitative comparisons between algorithms instead of real-valued performance estimates.

A common approach to label ranking is based on the so-called Plackett-Luce (PL) model [4], which specifies a parameterized probability distribution on rankings over labels (i.e., algorithms in our case). The underlying idea is to associate each algorithm a with a latent utility function $\widehat{m}_a : \mathcal{I} \longrightarrow \mathbb{R}_+$ of a context (i.e., an instance), which estimates how well an algorithm is suited for a given instance. The functions $\widehat{m}_a$ are usually modeled as log-linear functions

$$\widehat{m}_a(i) = \exp\left(\boldsymbol{\theta}_a^\top \, f_I(i)\right) , \tag{3}$$

where $\boldsymbol{\theta}_a \in \mathbb{R}^k$ is a real-valued, k-dimensional vector, which has to be fit for each algorithm a. The PL model specifies a probability distribution on rankings: given an instance $i \in \mathcal{I}$, the probability of a ranking $a_1 \succ \ldots \succ a_z$ over any subset $\{a_1, \ldots, a_z\} \subseteq \mathcal{A}$ is

$$\mathbb{P}(a_1 \succ \ldots \succ a_z \,|\, \boldsymbol{\Theta}) = \prod_{n=1}^{z} \frac{\widehat{m}_{a_n}(i)}{\widehat{m}_{a_n}(i) + \ldots + \widehat{m}_{a_z}(i)} \, . \tag{4}$$

A probabilistic model of that kind suggests learning the parameter matrix $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_a \,|\, a \in \mathcal{A}\}$ via maximum likelihood estimation, i.e., by maximizing

$$L(\boldsymbol{\Theta}) = \prod_{i \in \mathcal{I}_D} \mathbb{P}(\pi_i \,|\, \boldsymbol{\Theta})$$

associated with (4); this approach is explained in detail in [4]. Hence, the associated loss function under which we learn is now of a probabilistic nature (the logarithm of the PL-probability). It no longer focuses on the difference between the approximated performance $\widehat{m}_a(i)$ and the true performance $m(i, a)$, but on the ranking of the algorithms with respect to m—putting it in the jargon of preference learning, the former is a “pointwise” while the latter is a “listwise” method for learning to rank [3].

This approach potentially overcomes the second problem explained for the case of regression, but not the first one: It still fits a single model per algorithm a (the parameter vector $\boldsymbol{\theta}_a$), which essentially disqualifies it for the XAS setting.

3.3 Collaborative Filtering

This may suggest yet another approach, namely the use of collaborative filtering (CF) [8], in the setting of AS originally proposed by [23]. In CF for AS, we assume a (usually sparse) performance matrix $R^{|\mathcal{I}_D| \times |\mathcal{A}|}$, where an entry $R_{i,a} = m(i, a)$ corresponds to the performance of algorithm a on instance i according to m if known, and $R_{i,a} = ?$ otherwise. CF methods were originally designed for large-scale settings, where products (e.g. movies) are recommended to users, and data to learn from is sparse. Hence, they appear to fit well for our XAS setting.

Similar to regression and ranking, model-based CF methods also learn a latent utility function. They do so by applying matrix factorization techniques to the performance matrix R, trying to decompose it into matrices $U \in \mathbb{R}^{|\mathcal{I}_D| \times t}$ and $V \in \mathbb{R}^{t \times |\mathcal{A}|}$ w.r.t. some loss function $L(R, U, V)$, such that

$$R \approx \widehat{R} = UV^\top , \tag{5}$$

where U (V) can be interpreted as latent features of the instances (algorithms), and t is the number of latent features. Accordingly, the latent utility of a known algorithm a for a known instance i can be computed as

$$\widehat{m}_a(i) = U_{i,\bullet} V_{\bullet,a}^{\top}\,, \tag{6}$$

even if the associated value $R_{i,a}$ is unknown in the performance matrix used for training. The loss function $L(R, U, V)$ depends on the exact approach used—examples include the root mean squared error and the absolute error restricted by some regularization term to avoid overfitting. In [15], the authors suggest a CF approach called Alors, which we will use in our experiments later on. It can deal with unknown instances by learning a feature map from the original instance to the latent instance feature space. Alors leverages the CF approach CoFi$^{\text{RANK}}$ [31] using the normalized discounted cumulative gain (NDCG) [30] as loss function $L(R, U, V)$. Since the NDCG is a ranking loss, it focuses on decomposing the matrix R so as to produce an accurate ranking of the algorithms. More precisely, it uses an exponentially decaying weight function for ranks, such that more emphasis is put on the top and less on the bottom ranks. Hence, it seems particularly well suited for our use case.

4 Dyadic Feature Representation

As discussed earlier, by leveraging instance features, or learning such a representation as in the case of Alors, the approaches presented in the previous section can generalize over instances. Yet, none of them scales well to the XAS setting, as they do not generalize over algorithms; instead, the models are algorithm-specific and trained independently of each other. For the approaches presented earlier (except for Alors), this does not only result in a large number of models but also requires these models to be trained on very few data. Furthermore, it is not uncommon to have algorithms without any observation. A natural idea, therefore, is to leverage feature information on algorithms as well.

More specifically, we use a feature function $f_A : \mathcal{A} \longrightarrow \mathbb{R}^d$ representing algorithms as d-dimensional, real-valued feature vectors. Then, instead of learning one latent utility model per algorithm, the joint feature representation of a "dyad" consisting of an instance and an algorithm, allows us to learn a single joint model

$$\widehat{m} : f_I(\mathcal{I}) \times f_A(\mathcal{A}) \longrightarrow \mathbb{R}\,, \tag{7}$$

and hence to estimate the performance of a given algorithm a on a given instance i in terms of $\widehat{m}(f_I(i), f_A(a))$.

4.1 Regression

With the additional feature information at hand, instead of constructing one dataset per algorithm, we resolve to a single joint dataset comprised of examples

$\Big(\psi\big(f_I(i), f_A(a)\big), m(i,a)\Big)$ with dyadic feature information for all instances $i \in \mathcal{I}_D$ and algorithms $a \in \mathcal{A}$ for which a performance value $m(i,a)$ is known. Here,

$$\psi : \mathbb{R}^k \times \mathbb{R}^d \longrightarrow \mathbb{R}^q \tag{8}$$

is a joint feature map that defines how the instance and algorithm features are combined into a single feature representation of a dyad. What is sought, then, is a (parametrized) latent utility function $\widehat{m}_{\boldsymbol{\theta}} : \mathbb{R}^q \longrightarrow \mathbb{R}$, such that

$$\widehat{m}_{\boldsymbol{\theta}}\Big(\psi\big(f_I(i), f_A(a)\big)\Big) \tag{9}$$

is an estimation of the performance of algorithm a on instance i. Obviously, the choice of ψ will have an important influence on the difficulty of the regression problem and the quality of the model (9) induced from the data $\mathcal{D}^{REG}$. The regression task itself comes down to learning the parameter vector $\boldsymbol{\theta}$. In principle, this can be done exactly as in Sect. 3.1, also using the same loss function. Note that this is a generalization of the approach used by SMAC [11] for predicting performances across instances in algorithm configuration. We allow for a generic joint feature map ψ and an arbitrary model for $\widehat{m}_{\boldsymbol{\theta}}$, whereas SMAC limits itself to a concatenation of features and trains a random forest for modeling $\widehat{m}_{\boldsymbol{\theta}}$. Once again, it is noteworthy that SMAC by itself is not applicable in the XAS setting, as it relies on costly online evaluations.

4.2 Ranking

A similar adaptation can be made for the (label) ranking approach presented in Sect. 3.2 [25]. Formally, this corresponds to a transition from the setting of label ranking to the setting of *dyad ranking* (DR) as recently proposed in [21]. The first major change in comparison to the setting of label ranking concerns the training data, where the rankings π_i over subsets of algorithms $\{a_{i,1}, \ldots, a_{i,l_i}\} \subseteq \mathcal{A}$ for instance i are now of the form

$$\psi\big(f_I(i), f_A(a_{i,1})\big) \succ \ldots \succ \psi\big(f_I(i), f_A(a_{i,l_i})\big)\,. \tag{10}$$

Thus, we no longer represent an algorithm a simply by its label (a) but by features $f_A(a)$. Furthermore, like in the case of regression, we no longer learn one latent utility function per algorithm, but a single model of the form (9) based on a dyadic feature representation. In particular, we model $\widehat{m}_{\boldsymbol{\theta}}$ as a feed-forward neural network, where $\boldsymbol{\theta}$ represents its weights, which, as shown in [21], can be learned via maximum likelihood estimation on the likelihood function implied by the underlying PL model. Note that the use of a neural network is of particular interest here, since it allows one to learn the underlying joint feature map ϕ implicitly. Although both instance and algorithm features are simply fed as a concatenated vector into the network, it can recombine these features due to its structure and thus implicitly learn such a joint feature representation.

In contrast to the methods presented in the previous section, the methods based on dyadic feature information are capable of assigning a utility to unknown algorithms. Thus, they are well suited for the XAS setting and in principle even applicable when $\mathcal{A}$ is infinite, as long as a suitable feature representation f_A is available. Furthermore, as demonstrated empirically in Sect. 5, the dyadic feature approaches are very well suited for dealing with sparse performance matrices that are typical of the XAS setting.

5 Experimental Evaluation

In our experiments, we evaluate well established state-of-the-art approaches to algorithm selection as well as the proposed dyadic approaches in the XAS setting. More specifically, we consider the problem of selecting a machine learning classifier (algorithm) for a new classification dataset (instance) as a case study related to the "on-the-fly machine learning" scenario [17]. Please note that this is just one amongst many conceivable instantiations of the XAS setting, which is supposed to demonstrate the performance of the presented methods. To this end, we first generate a benchmark and then use this benchmark for comparison. The generated benchmark dataset as well as the implementation of the approaches including detailed documentation is provided on GitHub[1].

5.1 Benchmark Dataset

In order to benchmark the generalization performance of the approaches presented above in the XAS setting, we consider the domain of machine learning. More precisely, the task is to select a classification algorithm for an (unseen) dataset. Therefore, a finite set of algorithms $\mathcal{A}$ for classification and a set of instances $\mathcal{I}$ corresponding to classification datasets need to be specified. Furthermore, a performance measure is needed to score the algorithms' performance.

The set of candidate algorithms $\mathcal{A}$ is defined by sampling up to 100 different parameterizations of 18 classification algorithms from the machine learning library WEKA [7], ensuring these parameterizations not being too similar. An overview of the algorithms, their parameters and the number of instantiations contained in $\mathcal{A}$ is given in Table 2. This yields $|\mathcal{A}| = 1,270$ algorithms in total. The last row of the table sums up the items of the respective column, providing insights into the dimensionality of the space of potential candidate algorithms.

The set of instances $\mathcal{I}$ is taken from the OpenML CC-18 benchmarking suite[2] [27], which is a curated collection of various classification datasets that are considered interesting from a model selection resp. hyperparameter optimization point of view. This property makes the datasets particularly appealing for the XAS benchmark dataset, as it ensures more diversity across the algorithms.

Accordingly, the total performance matrix spanned by the algorithms and classification datasets in principle features $1,270 \cdot 71 = 88,900$ entries for which the benchmark contains $55,919$ actual values and the rest are unknown.

[1] https://github.com/alexandertornede/extreme_algorithm_selection.

[2] https://docs.openml.org/benchmark/#openml-cc18 (Excluding datasets 554, 40923, 40927, 40996 due to technical issues.).

In the domain of machine learning, one is usually more interested in the generalization performance of an algorithm than in the runtime. Therefore, m is chosen to assess the solution performance of an algorithm. To this end, we carry out a 5-fold cross validation and measure the mean accuracy across the folds[3]. As the measure of interest, accuracy is a reasonable though to some extent arbitrary choice. Note that in principle any other measure could have been used for generating the benchmark as well.

Table 2. The table shows the types of classifiers used to derive the set $\mathcal{A}$. Additionally, the number of numerical parameters (#num.P), categorical parameters (#cat.P), and instantiations (n) is shown.

Learner	0R	1R	BN	DS	DT	IBk	J48	JR	KS	L	LMT	MP	NB	PART	REPT	RF	RT	SMO
#num.P	0	1	0	0	1	1	2	2	1	1	2	2	0	2	3	3	4	1
#cat.P	0	0	2	0	3	3	6	2	2	0	5	6	2	2	2	2	4	2
n	1	30	12	1	45	89	100	100	99	100	100	100	3	91	100	99	100	100

Training data for CF and regression-based approaches can then be obtained by using the performance values as labels. In contrast, for training ranking approaches, the data is labeled with rankings derived by ordering the algorithms in a descending order w.r.t. their performance values. Note that information about the exact performance value itself is lost in ranking approaches.

We would like to note that the problem underlying this benchmark dataset could of course be cast as an AC or CASH problem. However, here we make the assumption that there is no time for costly online evaluations due to the on-the-fly setting and hence standard AC and CASH methods are not applicable.

Instance Features. For the setting of machine learning, the instances are classification datasets and associated feature representations are called meta-features [18]. To derive a feature description of the datasets, we make use of a specific subclass of meta-features called *landmarkers*, which are performance scores of cheap-to-validate algorithms on the respective dataset. More specifically, we use all 45 landmarkers as provided by OpenML [27], for which different configurations of the following learning algorithms are evaluated based on the error rate, area under the (ROC) curve, and Kappa coefficient: Naive Bayes, One-Nearest Neighbour, Decision Stump, Random Tree, REPTree and J48. Hence, in total we use 45 features to represent a classification dataset.

Algorithm Features. The presumably most straight-forward way of representing an algorithm in terms of a feature vector is to use the values of its hyperparameters. Thus, we can describe each individual algorithm by a vector of their hyperparameter-values. Based on this, the general feature description

[3] The standard deviation of the performance values per dataset is on average 0.101, minimum 0.0064 and maximum 0.33.

is obtained by concatenation of the vectors. As already mentioned, the neural network-based dyad ranking approach implicitly learns a more sophisticated joint feature map. Due to the way in which we generated the set of candidate algorithms $\mathcal{A}$, we can compress the vector sharing features for algorithms of the same type. Additionally, we augment the vector by a single categorical feature denoting the type of algorithm. Given any candidate algorithm, its feature representation is obtained by setting the type of algorithm indicator feature to its type, each element of the vector corresponding to one of its hyperparameters to the specific value, and other entries to 0. Categorical parameters, i.e. features, are one-hot encoded yielding a total of 153 features to represent an algorithm.

5.2 Baselines

To better relate the performance of the different approaches to each other and to the problem itself, we employ various baselines. While `RandomRank` assigns ranks to algorithms simply at random, `AvgPerformance` first averages the observed performance values for each candidate algorithm and predicts the ranking according to these average performances. k-NN LR retrieves the k nearest neighbors from the training data, averages the performances and predicts the ranking which is induced by the average performances. Since `AvgRank` is commonly used as another baseline in the standard AS setting, we note that we omit this baseline on purpose. This is because meaningful average ranks of algorithms are difficult to compute in the XAS setting, where the number of algorithms evaluated, and hence the length of the rankings of algorithms, vary from dataset to dataset.

5.3 Experimental Setup

In the following experiments, we investigate the performance of the different approaches and baselines in the setting of XAS for the example of the proposed benchmark dataset as described in Sect. 5.1.

We conduct a 10-fold cross validation to divide the dataset into 9 folds of known and 1 fold of unknown instances. From the resulting set of known performance values, we then draw a sample of 25, 50, or 125 pairs of algorithms for every instance under the constraint that the performances of the two algorithms is not identical. Thus, a maximum fill degree of 4%, 8% respectively 20% of the performance matrix is used for training, as algorithms may occur more than once in the sampled pairs. The sparse number of training examples is motivated by the large number of algorithms in the XAS setting. The assumption that performance values are only available for a small subset of the algorithms is clearly plausible here. Throughout the experiments, we ensure that all approaches are provided the same instances for training and testing, and that the label information is at least based on the same performance values.

In the experiments, we compare various models with each other. This includes two versions of `Alors`, namely `Alors (REGR)` and `Alors (NDCG)` optimizing for a regression respectively ranking loss. Furthermore, we consider a state-of-the-art regression approach learning a RandomForest regression model per algorithm (`PAReg`). Note that for those algorithms with no training data at all, we make `PAReg` predict a performance of 0, as recommending such an algorithm does not seem reasonable. Lastly, we consider two approaches leveraging a dyadic feature representation, internally fitting either a RandomForest for regression (`DFReg`) or a feed-forward neural network for ranking (`DR`). For both dyadic approaches, the simple concatenation of instance and algorithm features is used as a feature map. In contrast to the other methods, the ranking model is only provided the information which algorithm of a sampled pair performs better, as opposed to the exact performance value that is given to other methods. A summary of the type of features and label information used by the different approaches/baselines is given on the left side of Table 3.

Table 3. Overview of the data provided to the approaches and their applicability to the considered scenarios.

	Approach	f_I	f_A	Label		Approach	f_I	f_A	Label
approaches	`Alors (REGR)`	✓	✗	m	baselines	`RandomRank`	✗	✗	
	`Alors (NDCG)`	✓	✗	m		`AvgPrfm`	✗	✗	m
	`PAReg`	✓	✗	m		`AvgRank`	✗	✗	π
	`DFReg`	✓	✓	m		k-`NN LR`	✓	✗	m
	`DR`	✓	✓	π					

The test performance of the approaches is evaluated by sampling 10 algorithms for every (unknown) instance to test for. The comparison is done with respect to different metrics detailed further below, and the outlined sampling evaluation routine is repeated 100 times.

Statistical significance w.r.t performance differences between the best method and any other method is determined by a Wilcoxon rank sum test with a threshold of 0.05 for the p-value. Significant improvements of the best method over another one is indicated by •.

Experiments were run on nodes with two Intel Xeon Gold "Skylake" 6148 with 20 cores each and 192 GB RAM.

5.4 Performance Metrics

On the test data, we compute the following performance metrics measuring desirable properties of XAS approaches.

regret@k is the difference between the performance value of the best algorithm within the predicted top-k of algorithms and the actual best algorithm. The domain of regret@k is $[0, 1]$, where 0 is the optimum meaning no regret.

NDCG@k is a position-dependent ranking measure (*n*ormalized *d*iscounted *c*umulative *g*ain) to measure how well the ranking of the top-k algorithms can be predicted. It is defined as

$$\text{NDCG@}k(\pi, \pi^*) = \frac{\text{DCG@}k(\pi)}{\text{DCG@}k(\pi^*)} = \frac{\left(\sum_{n=1}^{k} \frac{2^{m(i,\pi_n)-1}}{\log(n+2)}\right)}{\left(\sum_{n=1}^{k} \frac{2^{m(i,\pi^*_n)-1}}{\log(n+2)}\right)},$$

where i is a (fixed) instance, π is a ranking and π^* the optimal ranking, and π_n gives the algorithm on rank n in ranking π. The NDCG emphasizes correctly assigned ranks at higher positions with an exponentially decaying importance. NDCG ranges in $[0, 1]$, where 1 is the optimal value.

Kendall's τ is a rank correlation measure. Given two rankings (over the same set of elements) π and π', it is defined as

$$\tau(\pi, \pi') = \frac{C - D}{\sqrt{(C + D + T_\pi) \cdot (C + D + T_{\pi'})}} \tag{11}$$

where C/D is the number of so-called *concordant*/*discordant* pairs in the two rankings, and $T_\pi/T_{\pi'}$ is the number of ties in π/π'. Two elements are called a concordant/discordant pair if their order within the two rankings is identical/different, and tied if they are on the same rank. Intuitively, this measure determines on how many pairs the two rankings coincide. It takes values in $[-1, 1]$, where 0 means uncorrelated, -1 inversely, and 1 perfectly correlated.

Table 4. Results for the performance metrics Kendall' tau (τ), NDCG@k (N@3, N@5), and regret@k (R@1, R@3) for varying number of performance value pairs used for training. The best performing approach is highlighted in bold, the second best is underlined, and significant improvements of the best approach over others is denoted by ●.

Approach	4% fill rate / 25 performance value pairs					8% fill rate / 50 performance value pairs				
	τ	N@3	N@5	R@1	R@3	τ	N@3	N@5	R@1	R@3
PAReg	0.1712 ●	0.9352 ●	0.9433 ●	0.0601 ●	0.0185 ●	0.2537 ●	0.9453	0.9594	0.0493	0.0136
Alors (NDCG)	0.0504 ●	0.9205 ●	0.9223 ●	0.0686 ●	0.0225	0.0472 ●	0.9155 ●	0.9164 ●	0.0614 ●	0.0208
Alors (REGR)	0.0303 ●	0.9117 ●	0.9191 ●	0.0794 ●	0.0190 ●	0.0807 ●	0.9172 ●	0.9304 ●	0.0754 ●	0.0285 ●
DR	0.3445	0.9523	0.9604	0.0381	0.0089	**0.3950**	**0.9584**	**0.9685**	0.0322	**0.0087**
DFReg	**0.3819**	**0.9564**	**0.9652**	**0.0302**	**0.0079**	0.3692	0.9573	0.9661	**0.0300**	0.0123
RandomRank	-0.0038 ●	0.8933 ●	0.9105 ●	0.0878 ●	0.0272 ●	-0.0038 ●	0.8933 ●	0.9105 ●	0.0878 ●	0.0272 ●
AvgPerformance	0.1384 ●	0.9388 ●	0.9433 ●	0.0337	0.0090	0.2083 ●	0.9355 ●	0.9508 ●	0.0493 ●	0.0199 ●
1-NN LR	0.1227 ●	0.9290 ●	0.9310 ●	0.0733 ●	0.0230 ●	0.1059 ●	0.9246 ●	0.9296 ●	0.0564 ●	0.0209
2-NN LR	0.1303 ●	0.9278 ●	0.9310 ●	0.0642 ●	0.0193 ●	0.0874 ●	0.9269 ●	0.9343 ●	0.0541 ●	0.0206

Approach	20% fill rate / 125 performance value pairs				
	τ	N@3	N@5	R@1	R@3
PAReg	0.3003 ●	0.9525	0.9632	0.0395	0.0107
Alors (NDCG)	0.0540 ●	0.9220 ●	0.9242 ●	0.0542 ●	0.0228 ●
Alors (REGR)	0.1039 ●	0.9160 ●	0.9329 ●	0.0604 ●	0.0222 ●
DR	**0.4507**	**0.9696**	0.9715	**0.0241**	**0.0055**
DFReg	0.4264	0.9629	**0.9720**	0.0292	0.0071
RandomRank	-0.0038 ●	0.8933 ●	0.9105 ●	0.0878 ●	0.0272 ●
AvgPerformance	0.2541 ●	0.9437 ●	0.9536 ●	0.0523 ●	0.0084
1-NN LR	0.1152 ●	0.9245 ●	0.9318 ●	0.0594 ●	0.0249 ●
2-NN LR	0.1142 ●	0.9292 ●	0.9350 ●	0.0412	0.0176 ●

5.5 Results

The results of the experiments are shown in Table 4. It is clear from the table that the methods for standard algorithm selection tend to fail especially in the scenarios with only few algorithm performance values per instance. This includes the approach of building a distinct regression model for each algorithm (`PAReg`) as well as for the collaborative filtering approach `Alors`, independently of the loss optimized for, even though the NDCG variant has a slight edge over the regression one. Moreover, `Alors` even fails to improve over simple baselines, such as `AvgPerformance` and k-`NN LR`. With an increasing number of training examples, `PAReg` improves over the baselines and also performs better than `Alors`, but never yields the best performance for any of the considered settings or metrics.

In contrast to this, the proposed dyadic feature approaches clearly improve over both the methods for the standard AS setting and the considered baselines for all the metrics. Interestingly, `DFReg` performs best for the setting with only 25 performance value pairs, while `DR` has an edge over `DFReg` for the other two settings. Still, the differences between the dyadic feature approaches are never significant, whereas significant improvements can be achieved in comparison to the baselines and the other AS approaches.

Moreover, our study demonstrates the heterogeneity of the benchmark dataset. As described in [22], a relevant measure for heterogeneity is the per-instance potential for improvement over a solution that is static across instances, i.e., what is often called the single best algorithm or solver (SBS). In this case study, the SBS is represented by the `AvgPerformance` baseline, which is always worse than the oracle with respect to all measures and in particular the regret@k measures. Hence, as the superior performances of our approach compared to the `AvgPerformance` demonstrate, the benchmark dataset offers a potential for per-instance algorithm selection.

The results of our study show that models with strong generalization performance can be obtained despite the small number of training examples. Moreover, the results suggest that there is a need for the development of specific methods addressing the characteristics of the XAS setting. This concerns the large number of different candidate algorithms as well as the sparsity of the training data.

6 Related Work

As most closely related work, we subsequently highlight several AS approaches to learning latent utility functions. For an up-to-date survey, we refer to [12].

A prominent example of a method learning a regression-based latent utility function is [32], which features an empirical hardness model per algorithm for estimating the runtime of an algorithm, i.e., its performance, for a given instance based on a ridge regression approach in the setting of SAT solver selection. Similarly, [13] learn per-algorithm hardness models using statistical (non-)linear regression models for algorithms solving the winner determination problem. Depending on whether a given SAT instance is presumably satisfiable or not, conditional runtime prediction models are learned in [9] using ridge linear regression.

In [5], a label-ranking-based AS approach for selecting collaborative filtering algorithms in the context of recommender systems is presented leveraging nearest neighbor and random forest label rankers.

Similar to our work, [19] leverages algorithm features in the form of a binary vector indicating which algorithm is considered to learn a probabilistic ranking model considering up to tens of algorithms. AS was first modeled as a CF problem in [23], using a probabilistic matrix factorization technique to select algorithms for the constraint solving problem. Assuming a complete performance matrix for training, low-rank latent factors are learned in [14] using singular value decomposition to obtain a selector en par with the oracle. Lastly, in [26] a decision-theoretic approach is proposed leveraging survival analysis to explicitly acknowledge timeouts of algorithms in the learning process.

7 Conclusion

In this paper, we introduced the extreme algorithm selection (XAS) setting and investigated the scalability of various algorithm selection approaches in this setting. To this end, we defined a benchmark based on the OpenML CC-18 benchmark suite for classification and a set of more than 1,200 candidate algorithms. Furthermore, we proposed the use of dyadic approaches, specifically dyad ranking, taking into account feature representations of both problem instances (datasets) and algorithms, which allows them to work on very few training data. In an extensive evaluation, we found that the approaches exploiting dyadic feature representations perform particularly well according to various metrics on the proposed benchmark and outperform other state-of-the-art AS approaches developed for the standard AS setting.

The currently employed algorithm features allow for solving the cold start problem only to a limited extent, i.e., only algorithms featuring known hyperparameters can be considered as new candidate algorithms. Investigating features to describe completely new algorithms is a key requirement for the approaches considered in this paper, and therefore an important direction for future work.

Acknowledgements. This work was supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901/3 project no. 160364472) and by the Paderborn Center for Parallel Computing (PC^2) through computing time.

References

1. Bengio, S., Dembczynski, K., Joachims, T., Kloft, M., Varma, M.: Extreme classification (dagstuhl seminar 18291). Dagstuhl Reports **8**(7), 62–80 (2018)
2. Bischl, B., et al.: Aslib: a benchmark library for algorithm selection. Artif. Intell. **237**, 41–58 (2016)
3. Cao, Z., Qin, T., Liu, T., Tsai, M., Li, H.: Learning to rank: from pairwise approach to listwise approach. In: ICML (2007)

4. Cheng, W., Hüllermeier, E., Dembczynski, K.J.: Label ranking methods based on the plackett-luce model. In: ICML (2010)
5. Cunha, T., Soares, C., de Carvalho, A.C.: A label ranking approach for selecting rankings of collaborative filtering algorithms. In: SAC (2018)
6. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: NIPS (2015)
7. Frank, E., Hall, M., Witten, I.: The WEKA Workbench. Data Mining (2016)
8. Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using collaborative filtering to weave an information tapestry. Commun. ACM **35**(12), 61–70 (1992)
9. Haim, S., Walsh, T.: Restart strategy selection using machine learning techniques. In: SAT (2009)
10. Happe, M., Meyer auf der Heide, F., Kling, P., Platzner, M., Plessl, C.: On-the-fly computing: a novel paradigm for individualized it services. In: ISORC (2013)
11. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: LION (2011)
12. Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated algorithmselection: Survey and perspectives. ECJ **27**(1), 3–45 (2019)
13. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Learning the empirical hardness of optimization problems: the case of combinatorial auctions. In: CP (2002)
14. Malitsky, Y., O'Sullivan, B.: Latent features for algorithm selection. In: SOCS (2014)
15. Mısır, M., Sebag, M.: Alors: an algorithm recommender system. Artif. Intell. **244**, 219–344 (2017)
16. Mohr, F., Wever, M., Hüllermeier, E.: ML-Plan: automated machine learning via hierarchical planning. Mach. Learn. **107**(8), 1495–1515 (2018). https://doi.org/10.1007/s10994-018-5735-z
17. Mohr, F., Wever, M., Tornede, A., Hüllermeier, E.: From automated to on-the-fly machine learning. In: INFORMATIK (2019)
18. Nguyen, P., Hilario, M., Kalousis, A.: Using meta-mining to support data mining workflow planning and optimization. JAIR **51**, 605–644 (2014)
19. Oentaryo, R.J., Handoko, S.D., Lau, H.C.: Algorithm selection via ranking. In: AAAI (2015)
20. Rice, J.R.: The algorithm selection problem. In: Advances in computers, vol. 15. Elsevier (1976)
21. Schäfer, D., Hüllermeier, E.: Dyad ranking using plackett-lucemodels based on joint feature representations. Mach. Learn. **107**(5), 903–941 (2018)
22. Schneider, M., Hoos, H.H.: Quantifying homogeneity of instance sets for algorithm configuration. In: LION (2012)
23. Stern, D.H., Samulowitz, H., Herbrich, R., Graepel, T., Pulina, L., Tacchella, A.: Collaborative expert portfolio management. In: AAAI (2010)
24. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In: SIGKDD (2013)
25. Tornede, A., Wever, M., Hüllermeier, E.: Algorithm selection as recommendation: from collaborative filtering to dyad ranking. In: CI Workshop, Dortmund (2019)
26. Tornede, A., Wever, M., Werner, S., Mohr, F., Hüllermeier, E.: Run2survive: a decision-theoretic approach to algorithm selection based on survival analysis. CoRR abs/2007.02816 (2020)
27. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: Openml: networked science in machine learning. SIGKDD Explorations **15**(2), 49–60 (2013)

28. Vembu, S., Gärtner, T.: Label ranking algorithms: a survey. In: Preference Learning. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14125-6_3
29. Wang, W., Zheng, V.W., Yu, H., Miao, C.: A survey of zero-shot learning: settings, methods, and applications. ACM TIST (2019)
30. Wang, Y., Wang, L., Li, Y., He, D., Chen, W., Liu, T.: A theoretical analysis of NDCG ranking measures. In: COLT (2013)
31. Weimer, M., Karatzoglou, A., Le, Q.V., Smola, A.J.: Cofi rank-maximum margin matrix factorization for collaborative ranking. In: NIPS (2008)
32. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: portfolio-based algorithm selection for sat. JAIR **32**, 565–606 (2008)