Algorithm Selection as Recommendation: From Collaborative Filtering to Dyad Ranking

Alexander Tornede, Marcel Wever, Eyke Hüllermeier

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

1 Introduction

Problem classes such as integer optimization, SAT, and classification can be tackled by a large variety of algorithms, the performance of which may differ depending on the concrete problem instance at hand. In fact, theoretical arguments even exclude the existence of a single algorithm that is superior to all other algorithms on all instances of a problem class [17]. Hence, compared to using the algorithm that is best on average across an entire class of problem instances, called the single best solver (SBS), selecting a suitable algorithm for each instance separately should result in an increased overall performance.

This expectation has been confirmed in recent algorithm selection (AS) competitions [1]. Algorithm selection seeks to support and automate the selection of an algorithm that is most suitable for a given problem instance. Meanwhile, quite a number of methods for AS has been proposed in the literature [8]. One interesting idea is to treat AS as a recommendation problem, and to apply techniques such as collaborative filtering [6]. Going beyond standard collaborative filtering, we propose to tackle AS as a problem of so-called dyad ranking [13]. This approach is motivated by at least two potential advantages:

- First, treating problem/algorithm pairs as dyads allows the learner to exploit properties (features) of both the problem instances and the candidate algorithms.
- Second, providing recommendations in the form of rankings of a set of candidate algorithms is presumably easier than evaluating each of them in terms of a precise numerical score.

These advantages are substantiated by first experimental studies in the field of automated machine learning, i.e., the recommendation of machine learning algorithms for model induction on a given dataset.

2 Algorithm Selection

In the setting of (per-instance) algorithm selection, we are given a set of problem instances $\mathcal{I}$, a set of algorithms $\mathcal{A}$, and a performance measure $m : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$. The goal is to find an algorithm selector $s : \mathcal{I} \to \mathcal{A}$ such that, for a given instance $i \in \mathcal{I}$, the selector s chooses the algorithm with best performance according to measure m on instance i. Accordingly, the optimal selector, called oracle, is defined by

$$s^*(i) = \arg\max_{a \in \mathcal{A}} m(i, a) \tag{1}$$

for all $i \in \mathcal{I}$. For simplicity, we subsequently ignore any form of randomness imposed by an algorithm.

In practice, the performance measure m is usually costly to compute. Therefore, the obvious brute-force strategy of evaluating all algorithms for a given instance and returning the one performing best according to m is infeasible. Fortunately, we are usually provided with a subset $\mathcal{I}_D \subset \mathcal{I}$ of the instance space for which several of the algorithms have already been evaluated according to m. Invoking machine learning methods, this information can be used as training data to infer an algorithm selector $s : \mathcal{I} \to \mathcal{A}$ approximating the oracle (1).

Most state-of-the-art approaches to AS are complex systems that involve several steps, such as pre-solvers, portfolios, and other techniques, in addition to their core machine learning component [18, 5]. Here, we only focus on the latter, which is typically realized in the form of a regression model $h : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$. This model is supposed to track the performance of an algorithm for a given instance, and hence can be seen as a form of surrogate for m. Thus, an algorithm selector s can be constructed by returning the algorithm a that performs best on problem instance i according to h:

$$s(i) = \arg\max_{a \in \mathcal{A}} h(i, a) \tag{2}$$

Note that, in contrast to the performance measure m, the function h is usually cheap to evaluate. In contrast to the oracle (1), the computation

of (2) is hence feasible. Nevertheless, one can also cast the learning problem in another form, for example as a recommendation problem in the context of collaborative filtering.

3 Algorithm Selection through Collaborative Filtering

Preference learning methods [3] for predicting rankings of algorithms for a given instance have recently gained attention. This is motivated by the observation that predictions of exact performances are sufficient but actually not necessary for choosing the best algorithm from a set of candidates. Hence, learning a regression model $h : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$ appears to be an unnecessarily difficult problem.

In particular, methods related to collaborative filtering (CF) [6] have recently been studied [9, 10, 19, 2, 15, 4], although this idea was already introduced in [14] about a decade ago. In the standard CF setting, one is confronted with a set of products $\mathcal{P}$ and a set of users $\mathcal{U}$, and given a sparse $|\mathcal{U}| \times |\mathcal{P}|$ rating matrix R. The value contained in $R(u, p)$ is the rating of product p by user u. Common tasks associated with CF include *matrix completion*, which has the goal to infer the missing entries of the matrix R, and the *cold-start problem*, where an entire new row in the rating matrix R has to be predicted for a new user.

By treating problem instances as users, i.e. $\mathcal{U} = \mathcal{I}$, and products as algorithms, i.e. $\mathcal{P} = \mathcal{A}$, we can construct a rating matrix for algorithm selection in a very similar way, namely by filling the matrix with the evaluations of m available in the training data. An example of a corresponding rating matrix is depicted in Fig. 1.

This setting has two main disadvantages. Firstly, instead of incorporating expert knowledge about the algorithms in an explicit way, only latent characteristics (if at all) are induced (as done in [9]). Secondly, precise numerical information about the performance of algorithms is required. In practice, such information is often difficult to obtain, whereas weaker information in the form of qualitative comparisons between algorithms is more readily available. Imagine, for example, a scenario in which several algorithms are run in parallel until the first one found a solution. Then, if runtime is the performance measure to be optimized, precise numerical information is only generated for the first algorithm, while the knowledge that all other algorithms are worse is not directly used.

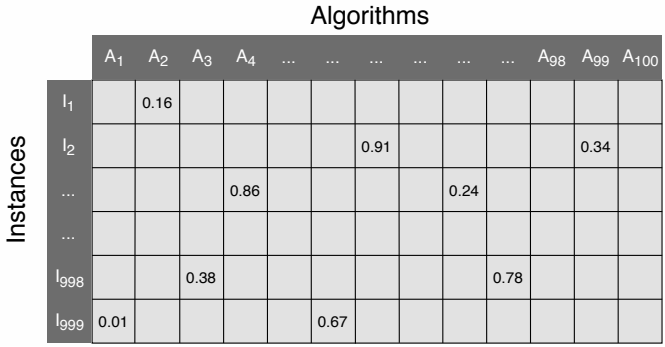

Figure 1: Depiction of a rating matrix filled with performance values from an algorithm selection dataset. Entries $R(i, a)$ contain the known performance of algorithm a on instance i and empty cells indicate unknown performances.

Moving from CF to dyad ranking [13] alleviates both of these disadvantages. Firstly, dyad ranking allows algorithm characteristics to be explicitly incorporated into the learning process. Secondly, dyad ranking merely requires qualitative training information in the form of rankings rather than precise numerical performances.

4 Algorithm Selection through Dyad Ranking

In addition to a feature representation for problem instances, we now also assume a feature representation for algorithms. By exploiting this information, there is hope to either speed up the model inference process or derive a more accuarate model. Moreover, instead of a real-valued rating matrix R, we assume a set of rankings over algorithms for the instances in the training set to be given. More precisely, we assume rankings over so-called *dyads*.

In (contextual) dyad ranking, a dyad $(\boldsymbol{x}, \boldsymbol{y})$ consists of a context $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^k$ from a context space $\mathcal{X}$ and an alternative $\boldsymbol{y} \in \mathcal{Y} \subseteq \mathbb{R}^r$. The training data we assume to be given is of the form

$$\mathcal{D} = \left\{ (\boldsymbol{x}_i, \boldsymbol{y}_{i,1}) \succ \ldots \succ (\boldsymbol{x}_i, \boldsymbol{y}_{i,l_i}) \right\}_{i=1}^{N} \subset \mathcal{R}(\mathcal{X} \times \mathcal{Y}) \tag{3}$$

and contains rankings with an underlying hidden preference relation $\succ$ over the space of dyads $\mathcal{X} \times \mathcal{Y}$, where l_i is the length of the ith ranking in $\mathcal{D}$, and $\mathcal{R}(\mathcal{X} \times \mathcal{Y})$ is the space of rankings over $\mathcal{X} \times \mathcal{Y}$. The goal is to learn a "dyad ranker"

$$h : \mathcal{P}(\mathcal{X} \times \mathcal{Y}) \to \mathcal{R}(\mathcal{X} \times \mathcal{Y}) \tag{4}$$

which, given an arbitrary set of dyads ($\mathcal{P}$ is the power set), ranks these dyads according to the hidden preference relation $\succ$.

To tackle the dyad ranking problem, we make use of the PLNet algorithm, a neural-network-based algorithm for learning a parametrized probability distribution over rankings, called the Plackett-Luce (PL) model [13].

A corresponding training dataset (3) is constructed by computing the feature representation of each algorithm and sorting the algorithms according to their performance in each row of the rating matrix $R(i, \cdot)$, pertaining to problem instance i. Additionally using the feature representation for instances, one can then extend the ranking to a ranking over dyads.

Problems to be considered during the construction include the sparseness of R as well as ties among algorithms for an instance in the rating matrix. The former can be solved by omitting algorithms with an unknown performance from the associated ranking. The easiest solution to the latter problem is to treat ties of algorithms by not comparing them directly. This can be achieved by creating a ranking ignoring n tied algorithms, copying it n times and adding each ignored algorithm in one of these copies at the respective position.

As already mentioned, a feature representation is required for both problem instances and algorithms. In the literature, various ways of representing instances via features have been proposed, depending on the problem domain. In the AutoML setting considered in this work, the instances are machine learning datasets and associated feature representations are called meta-features [11]. An example of such meta-features are *landmarkers*, which are performance values of cheap-to-train algorithms on the respective dataset or a subset thereof. As shown in [12], landmarkers can be used successfully in the context of algorithm selection and can yield better results than statistical measures, such as the number of classes in a dataset. Accordingly, for the experiments in this work, we make use of landmarking features for representing datasets. More specifically, we use 45 OpenML landmarkers [16], which are computed based

on learning algorithms such as Naive Bayes, One-Nearest Neighbour, Decision Stump, Random Tree, REPTree and J48.

Finding a feature representation for algorithms is more difficult, and the related literature is very sparse. We decided to represent algorithms via their parameters. Given a set of algorithms, we compute the union of their parameters and create a vector that has as many entries as the set of parameters. Then, when given a parametrized algorithm, we set the elements of the vector corresponding to its parameters to the respective values. Furthermore, for each component which can be contained in an algorithm, the vector contains a binary feature indicating whether the component is present or not. While this representation is simple, it has the disadvantage of not generalizing well across different algorithms that do not share any parameters, as they are essentially represented by disjoint subvectors of the original vector.

5 Experimental Results

We evaluated our approach in the AutoML setting, more specifically in the multi-class classification AutoML setting. Accordingly, instances correspond to multi-class classifcation datasets. Furthermore, the set of algorithms $\mathcal{A}$ we consider is a set of machine learning (classification) pipelines. By pipeline we mean the sequential combination of a data preprocessing step (such as a PCA) and a classification algorithm (such as an SVM). We considered 10 preprocessing steps and 7 classification algorithms resulting in a total of 70 classification pipelines. In addition, we considered up to 100 parametrizations for each of these pipelines and in total achieve an algorithm set with 5927 elements. We evaluated each of these parametrized pipelines on 29 classification datasets from OpenML[1]. Due to evaluation timeouts, only 89% of the theoretical amount of performance values is used.

Based on these performance values, we randomly sampled 10 train/test (70%/30%) splits on the datasets (i.e., each split features 20 training datasets and 9 test datasets). For each of these splits, we created dyad ranking training datasets by randomly sampling rankings of pipelines of length two, i.e., pairwise comparisons under the condition that the two pipelines do not have the same performance on the respective dataset. In

[1] `https://www.openml.org/`

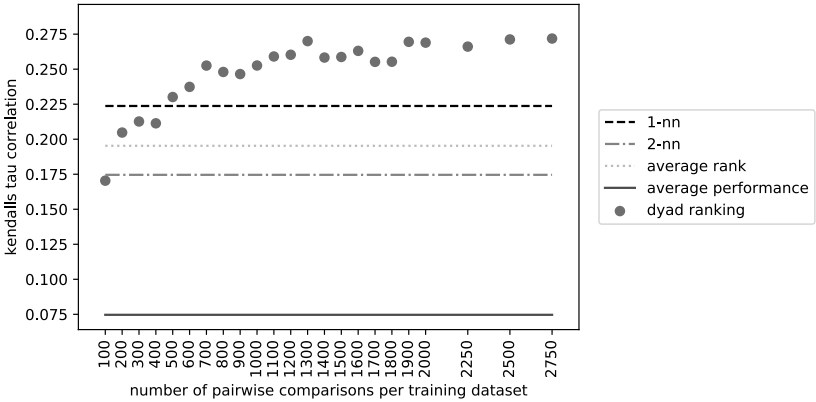

Figure 2: Kendall's τ rank correlation results for our dyad ranking approach based on different training dataset sizes and several baselines. All results are averaged across the test datsets and the train/test splits. On the x-axis, the number of pairwise rankings per training dataset used for training the associated dyad ranker is displayed whereas the y-axis shows the correlation measure value.

order to estimate how much information the learning algorithm (PLNet) requires to perform well, we evaluated different amounts of pairwise comparisons per dataset.

After training, we evaluated the approach by comparing the predicted ranking over all pipelines (for which we have a performance value) for each test dataset to the ground truth ranking obtained from the true performances using the Kendall's τ rank correlation measure [7], which takes values in $[-1, +1]$. We compared our approach against two instantiations of a nearest neighbor baseline, which, given a new dataset, computes the n closest training datasets according to the Euclidean distance, computes the average performance of all pipelines across these datasets and returns a ranking based on these averages. Furthermore, we compare against an average performance baseline, which simply returns a ranking based on the average performance of each pipeline across all training datasets, and an average rank baseline which does the same but averages the ranks instead of the performances. We averaged all results across the test datsets and the train/test splits we sampled.

Fig. 2 shows the value of the correlation measure as a function of the amount of training information (number of pairwise rankings per training

Table 1: This table gives the difference between the best pipeline across all pipelines (in terms of accuracy) and the best one of the top-k pipelines returned by the different approaches.

Approach	k	Perf. Diff.	k	Perf. Diff.
DR	3	**0.032**	5	**0.028**
1-nn	3	0.045	5	0.045
2-nn	3	0.053	5	0.052
avg. rank	3	0.045	5	0.044
avg. perf.	3	0.046	5	0.046

dataset used for training the associated dyad ranker). As the baselines always consider all data available in the training datasets, their performance does not change with different amounts of rankings.

As expected, the performance of the dyad ranking approach increases with the amount of training data — quite strongly up to around 1300 rankings per training dataset and more slowly thereafter. More importantly, the approach surpasses all baselines with only 500 pairwise rankings per training dataset, which is a remarkable result as the training information used by the dyad ranker is only a tiny fraction of the information made available to the baselines.

Furthermore, since the version of AS we consider in this work is mainly concerned with returning the best pipeline for a new dataset, we compared our approach to the baselines by computing the difference between the best pipeline (in terms of accuracy) and the best one of the top-k pipelines returned by the different approaches. This evaluation gives an idea of how much worse it is to run the top-k pipelines returned by the ranking approach compared to running the best pipeline (according to the oracle) only. The results of the experiment are depicted in Table 1.

The dyad ranking approach (trained with 1400 pairwise comparisons per training dataset for this experiment) outperforms all other baselines by at least 1.3% percent points for $k = 3$ and 1.6% points for $k = 5$. Admittedly, even the baselines achieve a reasonable result in this experiment, as even a performance difference of about 5% to the oracle is still very good. Nevertheless, only the dyad ranking approach is able to achieve a considerably better result when increasing k, which makes us believe

that it approximates the ground truth ranking better in the sense that it puts good pipelines in close proximity to their correct rank.

For full details regarding the experiments, we refer the interested reader to the github repository[2] containing all details and code required to reproduce the results presented here.

6 Conclusion and Future Work

We proposed to tackle the algorithm selection problem as a dyad ranking problem and addressed key questions regarding the creation of training datasets and feature representation for both algorithms and datasets.

Our first experimental studies show that dyad ranking outperforms the baselines we used for comparison. In future work, we plan to corroborate these preliminary results by more thorough evaluations of the approach in different scenarios as well as a comparison to state-of-the-art collaborative filtering methods.

Acknowledgement

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).

The authors gratefully acknowledge the funding of this project by computing time provided by the Paderborn Center for Parallel Computing (PC^2).

References

[1] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, et al. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.

[2] `https://github.com/alexandertornede/ci_2019_as_via_dyad_ranking`

[2] Tiago Cunha, Carlos Soares, and André C. P. L. F. de Carvalho. CF4CF: recommending collaborative filtering algorithms using collaborative filtering. In *Proceedings of the 12th ACM Conference on Recommender Systems, RecSys 2018, Vancouver, BC, Canada, October 2-7, 2018*, pages 357–361, 2018.

[3] Johannes Fürnkranz and Eyke Hüllermeier. *Preference learning*. Springer, 2010.

[4] Nicolo Fusi, Rishit Sheth, and Melih Elibol. Probabilistic matrix factorization for automated machine learning. In *Advances in Neural Information Processing Systems*, pages 3348–3357, 2018.

[5] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Marius Thomas Schneider, and Stefan Ziller. A portfolio solver for answer set programming: Preliminary report. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 352–357. Springer, 2011.

[6] David Goldberg, David A. Nichols, Brian M. Oki, and Douglas B. Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, 1992.

[7] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[8] Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, 27(1):3–45, 2019.

[9] Yuri Malitsky and Barry O'Sullivan. Latent features for algorithm selection. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014.*, 2014.

[10] Mustafa Misir and Michèle Sebag. Alors: An algorithm recommender system. *Artif. Intell.*, 244:291–314, 2017.

[11] Phong Nguyen, Melanie Hilario, and Alexandros Kalousis. Using meta-mining to support data mining workflow planning and optimization. *Journal of Artificial Intelligence Research*, 51:605–644, 2014.

[12] Bernhard Pfahringer, Hilan Bensusan, and Christophe G Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *ICML*, pages 743–750, 2000.

[13] Dirk Schäfer and Eyke Hüllermeier. Dyad ranking using plackett-luce models based on joint feature representations. *Machine Learning*, 107(5):903–941, 2018.

[14] David H. Stern, Horst Samulowitz, Ralf Herbrich, Thore Graepel, Luca Pulina, and Armando Tacchella. Collaborative expert portfolio management. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, 2010.

[15] Lisheng Sun-Hosoya, Isabelle Guyon, and Michèle Sebag. Activmetal: Algorithm recommendation with active meta learning. In *Proceedings of the Workshop on Interactive Adaptive Learning@ECML-PKDD 2018 Dublin, Ireland, September 10th, 2018.*, pages 48–59, 2018.

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

[17] David H Wolpert, William G Macready, et al. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[18] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.

[19] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. OBOE: collaborative filtering for automl initialization. *CoRR*, abs/1808.03233, 2018.