Missing Data and their effect on Algorithm Selection for the Bin Packing Problem

José Carlos Ortiz-Bayliss[1 [0000−0003−3408−2166]],
Anna Karen Gárate-Escamilla[1 [0000−0001−8841−0562]], and
Hugo Terashima-Marín[1 [0000−0002−5320−0773]]

Tecnologico de Monterrey, School of Engineering and Sciences,
Monterrey 64849, Mexico
{jcobayliss, karen.garate, terashima}@tec.mx

Abstract. When multiple algorithms are available to solve a particular problem, deciding which to use may be challenging. Depending on the problem at hand and the available algorithms, choosing the wrong solver might represent significant performance losses. Machine learning has emerged as a paramount tool for implementing algorithm selectors. Regardless of their success in various applications, how incomplete information on the individual solvers' performance may affect the overall quality of the decisions remains unexplored. This work uses Neural Networks (multi-layer perceptron) to implement algorithm selectors for the one-dimensional bin packing problem. We explore situations involving missing values and their treatment to observe their impact on the algorithm selectors generated. Our results suggest that the algorithm selectors may be more sensitive to missing values than expected.

Keywords: Algorithm Selection Problem · Bin Packing Problem · Missing Values · Neural Networks.

1 Introduction

Recent years have grown the interest in methods combining the strengths of 'simple' algorithms to solve challenging problems. Notably, we have observed advances related to strategies such as algorithm portfolios [2, 3, 10, 13] and hyper-heuristics [8, 9], which can broadly be considered algorithm selectors [17]. Algorithm selectors are collections of algorithms that run interchangeably, avoiding the single selection of a poor individual algorithm [10].

We have observed a trend for powering algorithm selectors with machine learning models in the past few years [11, 15, 19]. In this context, it is common to treat the problem as a supervised learning problem, in which a set of labeled examples is provided to train the models, and once the models are trained, they are tested on a set of unseen labeled examples. This scheme reduces the algorithm selection problem to a classification one that requires learning the problem features (classifier's input) that map to a suitable algorithm (classifier's output).

Missing values are around us, and they will stay there for a long time [16]. Then, it seems reasonable to think that missing values may also be present when generating algorithm selectors. However, the effect of missing values when using machine learning to produce algorithm selectors remains unexplored in the literature. Let us clarify this knowledge gap in the following lines. The rationale behind algorithm selection is to create a mapping between the problem features and one suitable algorithm from a set of available ones. Since this approach relies on supervised learning (the training requires a set of labeled examples), such training examples are usually represented as a vector containing $k + 1$ elements (the values of the k problem features and the identifier of the recommended algorithm). Producing such a vector requires at least two algorithms to solve the instance represented by such features. Only then can the best algorithm be identified. Thus, generating meaningful labels requires solving the instances with at least two algorithms. However, this may only be possible in some cases. For example, if we have four available algorithms but only the records of two of them are available for some instances in the training set. In light of this scenario, it is essential to consider the following questions: Would the missing values for the other two methods on the remaining instances affect the overall performance of the algorithm selector once it is trained? Can the missing values in the algorithms' results bias the selector towards selecting one particularly 'good' algorithm? How does it affect its capability to generalize? Would it be possible for the algorithm selectors to somehow 'resist' the effect of missing values in the data that produces the training examples? The literature has never deepened into this situation. However, it may represent an essential issue in the future since the algorithms the selectors choose from are getting better, sometimes requiring more computational resources in exchange. This additional consumption of resources may eventually become prohibitive and restrict us from getting the results of all the solvers (heuristics) to train an algorithm selector.

In this work, we explore the impact of missing values when using machine learning models as algorithm selectors. Particularly, we use multi-layer perceptrons (MLPs) as algorithm selectors, and we test our ideas on a widespread problem domain: the one-dimensional Bin Packing Problem (BPP). The BPP is a combinatorial problem in which a finite set of items with a specific length must be packed into bins, respecting a capacity constraint.

The remainder of this document is as follows. Section 2 contextualizes our work with respect to related studies. In Sect. 3, we describe our solution model. Section 4 describes the experiments conducted and analyses their main results. Finally, Sect. 5 presents the conclusion and provides potential paths for future work.

2 Background and Related Work

Machine learning has been used in the past to power algorithm selectors. In this regard, some representative works include that of Guo and Hsu [4], who explored how machine learning could be used for algorithm selection concerning

NP-hard optimization problems and tested their approach on the Most Probable Explanation Problem in probabilistic inference; Ortiz et al. [11], who proposed a framework for generating and testing algorithm selectors for Constraint Satisfaction Problems through various machine learning models; and Tornede et al. [20], who explored meta-learning to support algorithm selection for the Boolean Satisfiability Problem.

Neural networks have been used recurrently for addressing the algorithm selection problem. The earliest works on this topic include the one by Li et al. [7], who combined neural networks and logistic regression to produce problem-independent algorithms selectors. Later, Ortiz et al. [12] explored using Learning Vector Quantization (LVQ) Neural Networks to choose which heuristic to use when solving constrain satisfaction problems. Among the most recent works, we can cite the work of Mohamad et al. [1], who explored using recurrent neural networks to capture the sequential nature existing in combinatorial problems such as bin packing; and the work of Diaz et al. [6], who applied attention-based neural networks as meta-learners to enhance the performance of the mapping mechanism within the algorithm selection problem.

3 Solution Approach

Our solution approach generates algorithm selectors using neural networks — specifically, MLPs. Once trained, such perceptrons receive the problem state of an instance and recommend which algorithm to use. Because we test our approach on the BPP, the algorithms to choose from are simply heuristics.

The following lines provide insights into the main elements of our solution approach.

3.1 The Instances

We initially produced two sets of BPP instances: a training and a test set. We used the evolutionary approach proposed by Plata et al. [14] to generate instances challenging particular heuristics. Following this idea, the two sets contain various instances that allow the heuristics to exhibit opposite behaviors throughout the sets (sometimes they perform well, others do not). The training and test sets contain 100 and 400 instances, each with 20 items, a bin capacity of 64 units, and a maximum item length of 32. Although we admit that these instances may be considered small, they represent a good starting point to analyze how missing values affect the generation process of algorithm selectors through neural networks.

Starting from the training set, we gradually introduced missing values (completely at random) into the results obtained by the heuristics. In this way, we produced six versions of the training set based on the proportion of missing values introduced: NA00 (0%), NA05 (5%), NA10 (10%), NA15 (15%), NA20 (20%), and NA25 (25%). It is relevant to remind the reader that introducing missing values affects only the values of the heuristics. So, the values of the features are

always complete. Once we produced the six training sets containing the different proportions of missing values (NA00 to NA25), we generated two new 'treated' versions of sets NA05 to NA25: one where only complete cases remain (any row with one or more missing values was removed) and another where the median bin usage per heuristic is used to complete missing values. These two techniques are among the most basic when dealing with missing values. Finally, for each of the sets produced, we labeled the best heuristic. At this point, the data is ready to train the networks. In total, we use 16 training sets and a test set in this work. We graphically describe such sets in Figure 1.

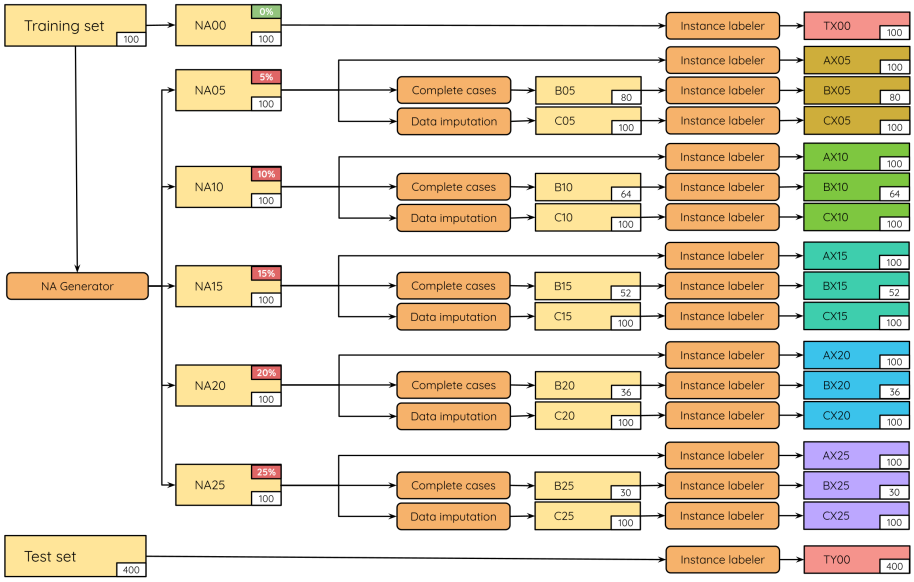

Fig. 1. Graphical depiction of the process to generate the training and test sets used in this investigation. The percentages in the top-right corners indicate the proportion of missing values contained in the corresponding set. The bottom-right numbers indicate the instances contained in the corresponding set.

3.2 Problem Characterization

This investigation deals with the one-dimensional version of the BPP. So, we can only use the length of the items to characterize the instances. For this purpose, we rely on three straightforward features related to the lengths of the items within the instance:

– **LENGTH** indicates the average length of the items within the instance. This value is normalized by dividing the actual average by 32 (the maximum length of items considered for this work).

- **SMALL** represents the proportion of 'small' items in the instance (items with a length smaller than 0.5).
- **LARGE** represents the proportion of 'large' items in the instance (items with a length larger or equal to 0.5).

We chose 0.5 as the threshold value for classifying an item as long or small based on preliminary observations. Besides, all these features lie in the range $[0, 1]$.

3.3 The Heuristics

For this work, we have considered four commonly used heuristics for the BPP [18].

- **First Fit (FF)** packs the item in the first bin with enough capacity for packing the item.
- **Worst Fit (WF)** packs the item in the bin that maximizes the waste.
- **Almost Worst Fit (AWF)** packs the item in the bin with the second most waste produced.
- **Best Fit (BF)** packs the item in the bin that minimizes the waste.

If there is no bin where we can pack the current item, the system opens a new bin for it. In the event of ties, the first bin that meets the heuristic's criterion is preferred.

3.4 Multi-layer Perceptron as an Algorithm Selector

Using neural networks as algorithm selectors is straightforward. As recommended when working with supervised learning tasks, we trained both networks exclusively on the training sets (they are different according to the experiment at hand). The MLPs developed for this work were implemented in Python, using the library `sklearn.neural_network.MLPClassifier`. Regarding the topology of the networks, they contain three neurons in the input layer (one per problem feature) and four neurons in the output layer (one per heuristic, and the neuron with the highest output value in the output layer is the one that determines the solver to use). Besides, the perceptrons have three hidden layers of 12, 10, and 6 neurons, respectively, and their corresponding biases. Although we consider that using some kind of hyper-parameter tuning strategy might improve the performance of the algorithm selectors these networks represent, we have used a straightforward parameter setting based on our previous experience. For adjusting the weights of the networks, we used the LBFGS optimizer. We used the default values in `sklearn.neural_network.MLPClassifier` for the rest of the configuration.

4 Experiments and Results

All the experiments conducted in this work focus on the impact of missing values under the most straightforward case regarding missing values, the Missing Completely at Random (MCAR) scenario. In MCAR situations, all the records of the

heuristics hold the same probability of disappearing [5]. In all the experiments presented in this section, we analyze the performance of the heuristics and the algorithm selectors through boxplots.

4.1 No Treatment on Missing Values

For this first experiment, we generated the algorithm selectors without treating the missing values in the training sets. This way, we chose the heuristic with the largest bin usage for each instance. Please note that the missing values affect different heuristics for each instance. Then, if one heuristic presents a missing value for one of the instances, it cannot be considered for selecting the best heuristic for such an instance.

We generated three algorithm selectors using set TX00 (complete training set with no missing values). Besides, we produced three algorithm selectors per training set AX05 to AX25. This resulted in 18 algorithm selectors with different proportions of missing values: 0%, 5%, 10%, 15%, 20%, and 25%; three selectors for each case. The training sets in this experiment received no treatment to deal with the missing values.

To our surprise, the results of this experiment (Figure 2) suggest that only a small proportion of missing values in the training set are sufficient to harm the resulting algorithm selector. The difference between the algorithm selectors trained with no missing values (EX1-00-01 to EX1-00-03) and the rest of the selectors is evident. We can easily observe how the IQR in the rest of the algorithm selectors increases, indicative of a larger standard deviation; hence, the results' dispersion. In summary, a small proportion of missing values, such as 5%, seems enough to prevent the neural networks from learning the mapping between features and heuristics, at least in this particular setting.

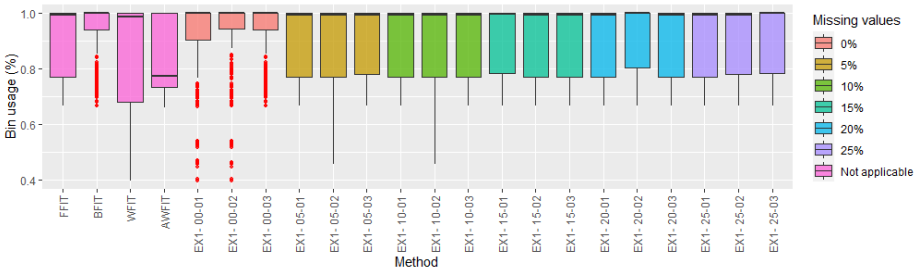

Fig. 2. Boxplot of the bin usage of the four heuristics and the 18 algorithm selectors trained on sets TX00 and AX05 to AX25 (instances with missing values in different proportions). We report the results obtained exclusively for set TY00.

Regarding algorithm selectors' performance, regardless of the different decisions taken in some instances, EX1-00-02 and EX-00-03 are statistically equivalent to BFIT, the best heuristic for the test set. This result was validated for each

algorithm selector using a two-tailed Wilcoxon test on the medians of the selector and BFIT, with a 5% of significance. The null hypothesis, which states that both medians are equal, was accepted with p-values of 0.2209 and 1, respectively.

Although it is always gratifying when the selectors outperform the best individual algorithm, it is not always possible. In this study, we are aware of the limitations of the model proposed and the opportunities to improve its results. For example, choosing a different set of problem features or fixing the neural networks' parameters via a systematic procedure. Indeed, those upgrades should improve the performance of the algorithm selectors produced by the model. However, behaving as well as the best individual heuristic is not a bad result. On the contrary, it confirms that the neural network is learning to discriminate and choose a suitable heuristic for each case. In this case, BFIT is the heuristic that provides the best overall performance, and the selectors have learned to perform as such a heuristic.

4.2 Working only with Complete Cases

For this second experiment, we applied a common technique for dealing with missing values: the complete cases analysis. Thus, we removed from the training sets any instance where at least one of the results from the heuristics was missing. As expected for this procedure, the number of training examples decreased with the number of missing values in the training sets. For example, for training set B25, we lost 70% of the training data when keeping only the complete cases. We acknowledge that such a data loss is out of proportion and unacceptable in any practical application. However, in this experiment, we aim to observe the various effects of missing values and the consequences of some of the frequently used techniques to deal with them.

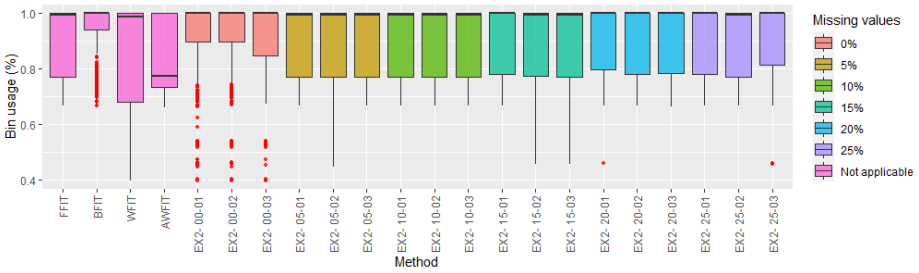

Fig. 3. Boxplot of the bin usage of the four heuristics and the 18 algorithm selectors trained on sets TX00 and BX05 to BX25 (instances that originally contained different proportions of missing values and now only complete cases remain). We report the results obtained exclusively for set TY00.

As in the previous experiment, we produced three algorithm selectors using set TX00. Additionally, we generated three algorithm selectors per training set

(BX05 to BX25). As previously described, the training sets in this experiment contain only complete cases. For this reason, the sets contain only a proportion of their original counterparts NA05 to NA25, reducing the number of examples for training (see Figure 1 for more details). Then, we tested the algorithm selectors on set TY00.

The results shown in Figure 3 look similar to the ones from the previous experiment. The algorithm selectors trained with no missing values tend to replicate the behavior of BFIT in the test set. Based on these results, it seems complicated to appreciate a difference in the performance of the remaining algorithm selectors and their counterparts concerning the proportion of missing values from the previous experiment.

4.3 Using Data Imputation

We conducted a third and last experiment involving data imputation. Instead of keeping only the complete cases, we synthetically produced some values to fill the gaps. The process required calculating the median bin usage of each heuristic (ignoring missing values) in the corresponding training set and then using that median to fill any missing value for the heuristics. Although this approach allowed us to keep all the records, there is a price to pay: some of the values in the data are synthetically produced, and then the labels of the best heuristic per instance may change based on the number of missing values replaced.

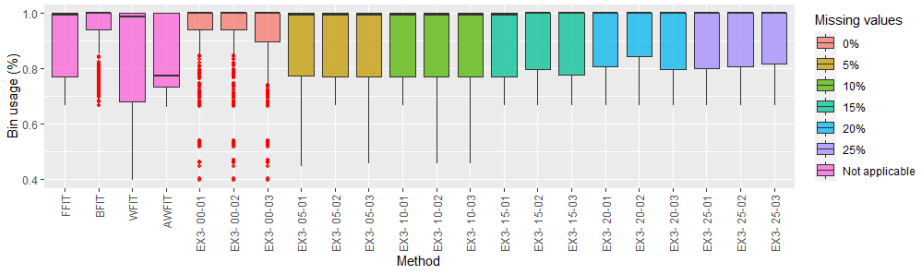

Fig. 4. Boxplot of the bin usage of the four heuristics and the 18 algorithm selectors trained on sets T00 and CX05 to CX25 (instances that originally contained different proportions of missing values but were replaced with the median of each heuristic). We report the results obtained exclusively for set TY00.

The results obtained from this experiment are similar to the previous ones. The selectors trained with TXOO (complete training set) are clearly better than the other algorithm selectors. Statistically, EX3-00-01, EX3-00-02, and EX3-00-03 are equivalent in performance to BFIT, with p-values of 0.2331, 0.1670, and 0.0555. As in previous cases, we cannot observe evident differences between the performance of the algorithm selectors produced with imputed data.

5 Conclusion

We have analyzed a few particular scenarios regarding missing values and their effect on the generation of algorithm selectors implemented as multi-layer perceptrons. Our results suggest that missing values affect the training process and the resulting selector regardless of the treatment used. An explanation for this is that a missing value (and its corresponding imputation) may alter the label of the best heuristic for an instance. On a larger picture, this can derive in contradictory cases that difficult the training. Of course, this preliminary work requires more experiments to confirm our initial observations.

We have considered some paths for future work derived from this investigation. First, we limited the scope of this work to MCAR scenarios within missing data. However, exploring other more complex scenarios, such as Missing at Random (MAR) and Missing not at Random (MNAR), is relevant [5]. Besides, other machine learning models could be used as algorithm selectors. Exploring other popular options like Logistic Regression, Support Vector Machines (SVM), and Decision Trees would be interesting and worth studying.

References

1. Alissa, M., Sim, K., Hart, E.: Automated algorithm selection: From feature-based to feature-free approaches. Journal of Heuristics **29**(1), 1—-38 (jan 2023). https://doi.org/10.1007/s10732-022-09505-4
2. Chalé, M., Bastian, N.D., Weir, J.: Algorithm selection framework for cyber attack detection. In: Proceedings of the 2nd ACM Workshop on Wireless Security and Machine Learning. pp. 37–42. WiseML '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3395352.3402623
3. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence **126**(1), 43–62 (2001). https://doi.org/https://doi.org/10.1016/S0004-3702(00)00081-3
4. Guo, H., Hsu, W.H.: A machine learning approach to algorithm selection for NP\mathcal{NP} -hard optimization problems: a case study on the MPE problem. Ann. Oper. Res. **156**(1), 61–82 (2007). https://doi.org/10.1007/s10479-007-0229-6, https://doi.org/10.1007/s10479-007-0229-6
5. Heymans, M.W., Twisk, J.W.: Handling missing data in clinical research. Journal of Clinical Epidemiology **151**, 185–188 (2022). https://doi.org/https://doi.org/10.1016/j.jclinepi.2022.08.016
6. Díaz de León-Hicks, E., Conant-Pablos, S.E., Ortiz-Bayliss, J.C., Terashima-Marín, H.: Addressing the algorithm selection problem through an attention-based meta-learner approach. Applied Sciences **13**(7) (2023). https://doi.org/10.3390/app13074601
7. Li, J., Burke, E.K., Qu, R.: Integrating neural networks and logistic regression to underpin hyper-heuristic search. Knowledge-Based Systems **24**(2), 322–330 (2011). https://doi.org/https://doi.org/10.1016/j.knosys.2010.10.004
8. Li, W., Özcan, E., Drake, J.H., Maashi, M.: A generality analysis of multiobjective hyper-heuristics. Information Sciences **627**, 34–51 (2023). https://doi.org/https://doi.org/10.1016/j.ins.2023.01.047

9. Marcel Panzer, B.B., Gronau, N.: A deep reinforcement learning based hyper-heuristic for modular production control. International Journal of Production Research **0**(0), 1–22 (2023). https://doi.org/10.1080/00207543.2023.2233641
10. Muñoz, M.A., Soleimani, H., Kandanaarachchi, S.: Benchmarking algorithm portfolio construction methods. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 499—-502. GECCO '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3520304.3528880
11. Ortiz-Bayliss, J.C., Amaya, I., Cruz-Duarte, J.M., Gutierrez-Rodriguez, A.E., Conant-Pablos, S.E., Terashima-Marín, H.: A general framework based on machine learning for algorithm selection in constraint satisfaction problems. Applied Sciences **11**(6) (2021). https://doi.org/10.3390/app11062749
12. Ortiz-Bayliss, J.C., Terashima-Marín, H., Conant-Pablos, S.E.: Learning vector quantization for variable ordering in constraint satisfaction problems. Pattern Recognition Letters **34**(4), 423–432 (2013). https://doi.org/https://doi.org/10.1016/j.patrec.2012.09.009
13. Piechowiak, K., Drozdowski, M., Éric Sanlaville: Framework of algorithm portfolios for strip packing problem. Computers & Industrial Engineering **172**, 108538 (2022). https://doi.org/https://doi.org/10.1016/j.cie.2022.108538
14. Plata-González, L.F., Amaya, I., Ortiz-Bayliss, J.C., Conant-Pablos, S.E., Terashima-Marín, H., Coello, C.A.C.: Evolutionary-based tailoring of synthetic instances for the knapsack problem. Soft Comput. **23**(23), 12711–12728 (2019). https://doi.org/10.1007/s00500-019-03822-w
15. Pylyavskyy, Y., Kheiri, A., Ahmed, L.: A reinforcement learning hyper-heuristic for the optimisation of flight connections. In: 2020 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8 (2020). https://doi.org/10.1109/CEC48606.2020.9185803
16. Ren, L., Wang, T., Sekhari Seklouli, A., Zhang, H., Bouras, A.: A review on missing values for main challenges and methods. Information Systems **119**, 102268 (2023). https://doi.org/https://doi.org/10.1016/j.is.2023.102268
17. Rice, J.R.: The algorithm selection problem. Advances in Computers, vol. 15, pp. 65–118. Elsevier (1976). https://doi.org/https://doi.org/10.1016/S0065-2458(08)60520-3
18. Silva-Gálvez, A., Orozco-Sanchez, J., Lara-Cárdenas, E., Ortiz-Bayliss, J.C., Amaya, I., Cruz-Duarte, J.M., Terashima-Marín, H.: Discovering action regions for solving the bin packing problem through hyper-heuristics. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 822–828 (2020). https://doi.org/10.1109/SSCI47803.2020.9308538
19. Slavchev, B., Masliankova, E., Kelk, S.: A machine learning approach to algorithm selection for exact computation of treewidth. Algorithms **12**(10) (2019). https://doi.org/10.3390/a12100200
20. Tornede, A., Gehring, L., Tornede, T., Wever, M., Hüllermeier, E.: Algorithm selection on a meta level. Mach. Learn. **112**(4), 1253–1286 (apr 2022). https://doi.org/10.1007/s10994-022-06161-4, https://doi.org/10.1007/s10994-022-06161-4