Algorithm Selection as Superset Learning: Constructing Algorithm Selectors from Imprecise Performance Data

Jonas Hanselle[0000−0002−1231−4985], Alexander Tornede[0000−0002−2415−2186], Marcel Wever[0000−0001−9782−6818], and Eyke Hüllermeier[0000−0002−9944−4108]

Heinz Nixdorf Institut and Department of Computer Science,
Paderborn University, Paderborn, Germany
{jonas.hanselle, alexander.tornede, marcel.wever, eyke}@upb.de

Abstract. Algorithm selection refers to the task of automatically selecting the most suitable algorithm for solving an instance of a computational problem from a set of candidate algorithms. Here, suitability is typically measured in terms of the algorithms' runtimes. To allow the selection of algorithms on new problem instances, machine learning models are trained on previously observed performance data and then used to predict the algorithms' performances. Due to the computational effort, the execution of such algorithms is often prematurely terminated, which leads to *right-censored* observations representing a lower bound on the actual runtime. While simply neglecting these censored samples leads to overly optimistic models, imputing them with precise though hypothetical values, such as the commonly used penalized average runtime, is a rather arbitrary and biased approach. In this paper, we propose a simple regression method based on so-called superset learning, in which right-censored runtime data are explicitly incorporated in terms of interval-valued observations, offering an intuitive and efficient approach to handling censored data. Benchmarking on publicly available algorithm performance data, we demonstrate that it outperforms the aforementioned naïve ways of dealing with censored samples and is competitive to established methods for censored regression in the field of algorithm selection.

Keywords: algorithm selection · superset learning · censored data.

1 Introduction

Per-instance Algorithm Selection (AS) denotes the problem of recommending an algorithm that appears to be most suitable for a given instance of a problem class. The suitability is assessed with respect to a specific performance criterion, such as solution quality or runtime. The latter is of special interest when dealing with computationally hard problems, such as the Boolean satisfiability problem (SAT) [26,25], the traveling salesperson problem (TSP) [19], or constraint satisfaction problems (CSP) [18], just to name a few. For this kind of problems, different

solvers or optimizers have been developed, which build on different heuristics to exploit certain structures inherent to the problem instances. As the heuristics are complementary in some sense, choosing the solver or optimizer on a per-instance basis can drastically improve the overall performance [13].

A common approach to AS is the use of machine learning methods to predict the runtime of algorithms on unseen problem instances. One major challenge for such approaches concerns the training data, parts of which are usually censored. The censoring has its root in the way the training data is generated. To assess the true runtime of an algorithm for a problem instance, it is simply run on that particular instance. However, for specifically hard instances, the algorithms may take days, weeks, or even years before returning a solution. To keep the computational effort reasonable, a cutoff time is set at which the execution of still running algorithms is aborted. In the commonly used AS benchmarking suite ASlib [2], up to 70% of the instances in the training data are censored (cf. Table 1).

To deal with censored data, several approaches consider them as missing values and apply imputation techniques to replace these values [21,26]. Alternatively, techniques from the field of survival analysis have been proposed [6,5,24]. While the latter approach appropriately captures the information provided by a censored observation, namely that the runtime exceeds the cutoff time C, the former turns it into unduly precise information and comes with the risk of incorporating a bias in the learning process.

In this paper, we propose to consider AS as a so-called superset learning problem [9], where the learner induces a (precise) predictive model from possibly imprecise training data. More concretely, we learn a regression model for runtime prediction from training instances that are either precisely labeled or labeled with an interval of the form (C, ∞). This is not only a just representation of the (weak albeit non-void) information that the true runtime exceeds C, but, as will be seen, also offers an efficient way for handling censored data. In our experimental evaluation, we show that methods based on superset learning can induce (linear) models that outperform naïve strategies for dealing with censored data in AS.

2 The Per-Instance Algorithm Selection Problem

The per-instance algorithm selection problem, first introduced in [20], is essentially a recommendation problem where the sought recommendation is an algorithm to be used on a specific instance of an algorithmic problem class (such as SAT). More formally, given a problem class (instance space) $\mathcal{I}$ and a set of algorithms $\mathcal{A}$ able to solve instances from this class, the goal is to learn a mapping $s : \mathcal{I} \longrightarrow \mathcal{A}$, called *algorithm selector*. Provided a problem instance $i \in \mathcal{I}$ and a performance measure $m : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R}$, such a selector s should ideally return the algorithm performing best in terms of this measure. This ideal mapping s^* is called the *virtual best solver* (VBS), aka. *oracle*, and formally defined as

$$s^*(i) \coloneqq \arg\min_{a \in \mathcal{A}} \mathbb{E}\left[m(i, a)\right] \tag{1}$$

for all instances $i \in \mathcal{I}$, where the expectation accounts for the potential randomness in the application of the algorithm. While conceptually simple to define, the computation cannot be performed through an exhaustive enumeration, as m is usually costly to evaluate and often even requires running the algorithm on the particular instance at hand, like in the case of runtime as a measure. For the remainder of the paper, we will assume that runtime is used as the performance measure. In contrast to the oracle, the *single best solver* (SBS) is the strategy that always selects the algorithm which is best on average across all instances, and accordingly can be seen as a natural baseline.

Closely related to AS is the problem of *algorithm scheduling* [17], where the recommendation target is a schedule of algorithms (instead of a single algorithm) assumed to be executed sequentially for a specific amount of time, until a solution is either found or the schedule is over. Also closely related is the *extreme algorithm selection* problem [22], where the recommendation target is still a single algorithm, but the set of algorithms $\mathcal{A}$ to choose from is assumed to be extremely large, i.e., in the hundreds to thousands compared to tens in the standard AS problem. Finally, *meta-algorithm selection* [23] refers to the problem of choosing among the many existing algorithm selection approaches and can be formalized as an algorithm selection problem itself.

2.1 Common Algorithm Selection Solutions

To circumvent the problem of costly evaluations of the performance measure m, the majority of existing AS approaches in one way or another learns a surrogate $\widehat{m} : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R}$ of the original performance measure, with the property of being cheap to evaluate. This allows for the exhaustive enumeration in (1), and hence to determine an algorithm selector $s : \mathcal{I} \longrightarrow \mathcal{A}$ with

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

For the purpose of representing instances, we assume the existence of a feature function $f : \mathcal{I} \longrightarrow \mathbb{R}^d$ mapping instances to d-dimensional feature vectors. Such features should comprise properties of the instance that are potentially relevant to determine the best algorithm. For SAT instances, examples of such features include the number of variables or the number of clauses in a formula. The computation of such features takes time, which is important to consider when the measure m to optimize is (related to) runtime. Moreover, learning such surrogate models requires training instances $\mathcal{I}_D \subset \mathcal{I}$ for which the performance value $m(i, a) \in \mathbb{R}$ is available for some algorithms $a \in \mathcal{A}$.

One of the simplest instantiations of the surrogate framework described above is to learn a performance predictor $\widehat{m}_a : \mathcal{I} \longrightarrow \mathbb{R}$ separately for each algorithm $a \in \mathcal{A}$ as done in Satzilla'07 [26]. In a later version of Satzilla, these models were replaced by cost-sensitive decision forests $\widehat{m}_{a,a'} : \mathcal{I} \longrightarrow \{0, 1\}$ for each pair of algorithms $a \neq a' \in \mathcal{A}$. Here, predictions are pairwise comparisons suggesting which of two candidate algorithms will most likely perform better on a given instance, and the final selection is determined by majority voting. In fact,

Satzilla'11 can be seen as a cost-sensitive version of the all-pairs decomposition scheme for solving multi-class classification problems, leading to the more general application of multi-class classification approaches $\widehat{m} : \mathcal{I} \longrightarrow \mathcal{A}$, where each algorithm $a \in \mathcal{A}$ is considered a class [24].

Moreover, to estimate the unknown performance of an instance/algorithm pair, instance-based approaches such as SUNNY [1] or ISAC [12] rely on similar instances for which evaluations are available, and make use of k-nearest neighbor or clustering techniques. In a more recent work [8], hybrid ranking and regression models are successfully applied to the problem of algorithm selection, with the aim to benefit from both perspectives (ranking and regression) on the problem.

2.2 The Influence of Censored Data

Recall that we assume a set of training instances $\mathcal{I}_D \subset \mathcal{I}$ with potentially missing values for the performance measure. That is, the performance values $m(i, a) \in \mathbb{R}$ are available for *some* but rarely for all algorithms $a \in \mathcal{A}$. This is because the training data generation for algorithm selection is usually performed under the constraint of a timeout. When an algorithm $a \in \mathcal{A}$ is run on an instance $i \in \mathcal{I}_D$ and does not terminate before a given *cutoff* time C, its execution is simply stopped at this point, as some algorithms can take extremely long to solve specific instances of combinatorial optimization problems [7]. In such a case, no precise performance value, i.e. runtime $m(i, a)$, can be recorded. Instead, the true runtime is only known to exceed the cutoff C, i.e. $m(i, a) > C$. In other words, the observation is *right-censored* [15]. The well-known ASlib [2] benchmark contains scenarios where over 70% of the data is censored. Obviously, since these data points make up a significant amount of available data, they need to be treated with care.

The simplest approach for dealing with censored data points is to completely ignore them when training surrogate models $\widehat{m}$. This strategy does not only waste important information, but also bears the danger of learning overoptimistic models, as potentially very long runtimes are not considered during the training. A slightly more advanced strategy is to impute the censored points either by the cutoff time C or a multiple thereof, motivated by the penalized average runtime (PARN) score. However, once again, this strategy can lead to strongly biased models, as in cases with many censored data points the majority of the training data is deliberately distorted. A more advanced imputation strategy has been proposed in [21] in the context of algorithm configuration (AC) [10,4]. It iteratively estimates truncated normal distributions over the censored data based on generalizations made by models over the uncensored data, and finally replaces each censored sample by the mean of the corresponding distribution. While offering a theoretical foundation, this approach makes a strong assumption on the distribution of the runtimes of the algorithms. Moreover, it is computationally expensive, as the process involves fitting the requested model repeatedly before returning a final model.

An ideal algorithm selection technique would directly incorporate censored samples in the training process, instead of artificially removing any imprecision in

a preprocessing step. This is accomplished by AS techniques leveraging concepts from *survival analysis*, which were first introduced in [6,5] for the setting of algorithm scheduling and recently improved by Run2Survive [24]. The latter is an algorithm for learning runtime distributions conditioned on an instance, which are in turn used in a decision-theoretic framework to tailor the selection of an algorithm towards a specific performance measure. Such methods do nevertheless rely on distributional assumptions.

Therefore, in this paper, we adopt another approach based on so-called *superset learning*, a problem setting that has recently attracted increasing attention in the machine learning literature [9]. Superset learning is an extension of standard supervised learning, which deals with the problem of learning predictive models from possibly imprecise training data characterized in the form of *sets* of candidate values. In principle, every method for supervised learning, whether parametric or non-parametric, can be extended to the setting of superset learning. Moreover, imprecision can be modeled in a rather flexible way and is not restricted to censoring.

3 Superset Learning with Right-Censored Data

In supervised learning, training data is supposed to be given in the form of tuples $(\boldsymbol{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the instance and $\mathcal{Y}$ the output space, and the task is to learn the dependency between instances and outcomes. More specifically, given a hypothesis space $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$ to choose from, the learner seeks to find an accurate predictive model in the form of a risk-minimizing hypothesis

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \int\limits_{(\boldsymbol{x},y) \in \mathcal{X} \times \mathcal{Y}} L(h(\boldsymbol{x}), y) \, \mathrm{d}\, \mathbb{P}(\boldsymbol{x}, y) \,, \tag{3}$$

where $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a loss function and $\mathbb{P}$ an unknown joint probability distribution characterizing the data-generating process. In superset learning, the values of the target variables, i.e., the outcomes y_n (and possibly also the instances $\boldsymbol{x}_n$, though this is of less interest for our purpose), are not necessarily observed precisely. Instead, they are characterized in terms of sets $Y_n \ni y_n$ known to cover the true observation y_n. For example, in the case of classification, Y_n is a (finite) set of candidate classes. In regression, where $\mathcal{Y} = \mathbb{R}$ is infinite, imprecise observations are typically modeled in the form of intervals.

Several methods for learning from imprecise observations of that kind have been proposed in the literature. Here, we adopt the approach of generalized loss minimization based on the so-called *optimistic superset loss* [11], which has also been proposed under the name *infimum loss* [3]. The optimistic superset loss (OSL) is a generalization of the original loss L in (3) that compares $h(\boldsymbol{x}) = \hat{y}$ with the set-valued target in an "optimistic" way:

$$L^*(\hat{y}, Y) = \inf_{y \in Y} L(\hat{y}, y) \tag{4}$$

Learning is then accomplished by using L^* in place of L, for example by minimizing the generalized empirical risk (or a regularized version thereof) on the training data $\mathcal{D} = \{(\boldsymbol{x}_n, Y_n)\}_{n=1}^N$:

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \frac{1}{N} \sum_{n=1}^{N} L^*(h(\boldsymbol{x}_n), Y_n)$$

This way of learning from imprecise data is essentially motivated by the idea of data disambiguation, i.e., of figuring out the true observations y_n among the candidates suggested by the set-valued observations Y_n. We refer to [9] for a more detailed explanation and to [3] for a theoretical foundation of this approach.

In our setting, hypotheses h correspond to algorithm-specific surrogate models $\widehat{m}_a : \mathcal{I} \to \mathbb{R}$ that estimate the runtime of algorithm $a \in \mathcal{A}$ on problem instance $i \in \mathcal{I}$. More specifically, we seek to learn a function $\mathbb{R}^d \to \mathbb{R}$ that maps instance feature vectors to algorithm runtime estimates. Here, precise values of the target variable correspond to true runtimes, i.e., the exact time it took an algorithm to solve a problem instance. However, these values are normally not measured precisely, and there are various reasons for why runtimes might better be characterized in the form of intervals $[l, u] \subset \mathbb{R}$. For example, the actual runtime might have been slowed down due to another process running in parallel, so that only a rough estimate could be derived.

Here, we are specifically interested in imprecision due to censoring, namely due to terminating an algorithm upon exceeding the cutoff time C. As we only know a lower bound on the ground truth runtime, in this case, the observations are right-censored. This information can be modeled in terms of the imprecise observation $Y = (C, \infty)$. As a consequence, the OSL in (4) simplifies as follows[1]:

$$L^*(\hat{y}, Y) = \begin{cases} L(\hat{y}, y) & \text{if } Y = \{y\} \\ L(\hat{y}, C) & \text{if } Y = (C, \infty) \text{ and } \hat{y} < C \\ 0 & \text{if } Y = (C, \infty) \text{ and } \hat{y} \geq C \end{cases}, \tag{5}$$

where the first case corresponds to precise observations and the second case to underestimated right-censored data points. In the third case, the runtime is correctly predicted to exceed C, whence the loss is 0. In principle, the OSL L^* can be instantiated with any suitable regression loss $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. For this paper, we choose the commonly used L_2-loss, also known as the *least squares error* $L(\hat{y}, y) = L_2(\hat{y}, y) := (y - \hat{y})^2$. An illustration of the OSL instantiated with this loss function is given in Figure 1.

In spite of its simplicity, the superset learning approach has several properties that make it appealing for dealing with right-censored data in the algorithm selection setting. Whereas the established iterative imputation scheme by Schmee and Hahn (Schmee&Hahn) [21] assumes that the censored samples are drawn

[1] Here, we make the assumptions that $L(y, y) = 0$ for all $y \in \mathcal{Y}$, $L(y, \cdot)$ is monotone decreasing on $(-\infty, y)$ and monotone increasing on (y, ∞), which hold for all reasonable loss functions.

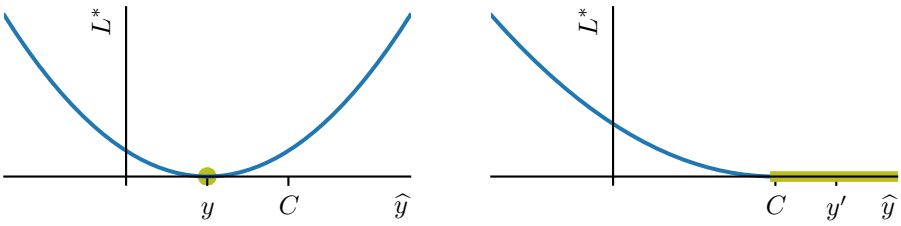

Fig. 1. Illustration of the OSL instantiated with the L_2-loss. The left example shows a precise observation $y \leq C$ for which we simply penalize the prediction with the squared distance to the actual observation. The right example shows a censored observation of a timed out algorithm run. As $y' > C$, the actual runtime y' is unknown and we only know the lower bound C. Thus, we have an imprecise observation $Y' = (C, \infty)$, for which we do not impose a penalty if the prediction $\hat{y}$ lies in the interval, i.e. if $\hat{y} \in Y'$. If $\hat{y} \notin Y'$, we penalize with the squared distance to the interval's lower bound C.

from a truncated normal distribution, the superset learning approach does not make any assumptions about the distribution of censored data. Another significant disadvantage of the iterative method is its computational cost. While Schmee&Hahn needs to iteratively refit a regression model multiple times until convergence (or a maximum number of iterations is reached), the superset learning method only needs a single pass for integrating censored samples into the regression model. Also intuitively, the approach is well-suited for the task of algorithm selection: To select a well-performing algorithm for a new problem instance, there is no need for precisely estimating the runtime of poorly performing algorithms. Still, to avoid selecting these algorithms, they need to be identified, which is reflected by enforcing predictions above the cutoff time C.

A toy example of precise and censored data points as well as the resulting linear regression models when using the L_2-loss and its OSL extension is given in Figure 2. The example illustrates why imputing censored samples with a constant leads to strongly biased models. The superset learning approach circumvents this problem by treating the censored samples as imprecise (interval-valued) observations, for which the loss is 0 if the prediction lies in the interval.

In the scope of this paper, we consider the class of linear models for the algorithm-specific runtime surrogates $\widehat{m}_a$. We minimize the proposed optimistic superset loss as depicted in (5) using standard gradient-based optimization.

4 Experimental Evaluation

For conducting the experimental evaluation of the proposed superset learning approach to algorithm selection, we use the *ASlib* benchmark [2]. This benchmark library consists of AS *scenarios*, which are collections of problem instances from various algorithmic problem domains such as SAT or TSP, characterized in terms of feature vectors as well as the performances achieved by all candidate algorithms on the respective problem instance. These scenarios entail significant

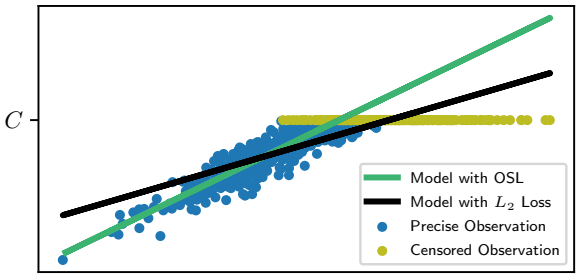

Fig. 2. Toy example of data with a linear dependency between feature and target and Gaussian centered noise. Observations above the cutoff C have been censored. The two lines indicate linear regression models, the green one has been trained using the OSL, the black one using the standard L_2-loss for which the censored observations were imputed with C.

amounts of censored algorithm runs; an overview of all scenarios is given in Table 1.

Table 1. Overview of the *ASlib* scenarios with the corresponding number of instances (#I), unsolved instances (#U), algorithms (#A), instance features (#F), the cutoff (C) as well as the fraction of censored algorithm runs (%C).

	ASP-POTASSCO	BNSL-2016	CPMP-2015	CSP-2010	CSP-MZN-2013	CSP-Minizinc-Time-2016	MAXSAT-PMS-2016	MAXSAT-WPMS-2016	MAXSAT12-PMS	MAXSAT15-PMS-INDU	MIP-2016	PROTEUS-2014	QBF-2011	QBF-2014	QBF-2016	SAT03-16_INDU	SAT11-HAND	SAT11-INDU	SAT11-RAND	SAT12-ALL	SAT12-HAND	SAT12-INDU	SAT12-RAND	SAT15-INDU	TSP-LION2015
#I	1294	1179	527	2024	4642	100	601	630	876	601	218	4021	1368	1254	825	2000	296	300	600	1614	767	1167	1362	300	3106
#U	82	0	0	253	944	17	45	89	129	44	0	456	314	241	55	269	77	47	108	20	229	209	322	17	0
#A	11	8	4	2	11	20	19	18	6	29	5	22	5	14	24	10	15	18	9	31	31	31	31	28	4
#F	138	86	22	86	155	95	37	37	37	37	143	198	46	46	46	483	115	115	115	115	115	115	115	54	122
C	600	7200	3600	5000	1800	1200	1800	1800	2100	1800	7200	3600	3600	900	1800	5000	5000	5000	5000	1200	1200	1200	1200	3600	3600
%C	20.1	28.1	27.6	19.6	70.2	50	39.4	57.9	41.2	48.9	20	60.2	54.7	55.6	36.2	24.7	60.7	33.3	47.5	53.9	66.7	50.4	73.5	23.5	9.6

4.1 Experimental Setup

In this work, we consider linear models for modeling the performance surrogate $\widehat{m}$. These models are trained using the *Adam* optimizer [14], minimizing the OSL instantiated with the L_2-loss. We chose a learning rate of 0.05 and a batch size of 32. For regularization, validation-based early stopping was employed: Before the training procedure starts, a fraction of 0.3 of the training data is put aside as validation data. Then, during training, this validation data is used to compute the validation loss periodically. If this loss increases for 16 consecutive checks,

training is terminated and the model parameters are set to the best observed parameters (with respect to the validation loss) during training. After a maximum number of $5,000$ epochs is reached, the training procedure is terminated. The implementation (together with detailed documentation) used for the conducted experiments is made publicly available on GitHub[2].

The performance of the considered models is assessed in terms of a 10-fold cross-validation. In each fold, a fraction of 0.9 of the problem instances and corresponding algorithm performances is used for the training procedure as described above and the remaining fraction of 0.1 is used for testing the model. The performance of the approaches is quantified in terms of the commonly used *penalized average runtime* with a penalty factor of 10 (PAR10). This measure simply averages the runtimes achieved by the selected algorithms over the problem instances in a scenario, while penalizing algorithm runs that timed out with a constant $10 \cdot C$.

4.2 Baseline Approaches

We compare the proposed algorithm selection technique based on superset learning with the commonly used strategies for dealing with censored samples. Using the same model configuration as described in the previous section, we consider the IGNORE strategy, in which we neglect training instances of censored runs as well as the naïve imputation schemes CLIP and PAR10, in which we impute the censored data points with the cutoff C or $10 \cdot C$, respectively, for training regression models using the L_2-loss. As a more elaborate solution, we compare our method against the iterative imputation scheme SCHMEE&HAHN [21].

4.3 Results

In the following, we discuss the results of the conducted experiments, which are summarized in Table 2. A $\bullet$ next to a baseline result indicates that the SUPERSET approach was significantly better, while a $\circ$ indicates that it was significantly worse on the corresponding scenario according to a Wilcoxon signed-rank test with $p = 0.05$. Per scenario, the best achieved result is highlighted in bold and the rank of each result is given in parentheses. We observe that the IGNORE strategy is the worst strategy among all considered approaches. As the *ASlib* scenarios include up to 73.5% of censored algorithm runs, this comes at no surprise, as large parts of the training data are simply discarded. The approaches CLIP and PAR10 that impute censored observations with constants yield better results than ignoring censored runs. Here, the approach CLIP that imputes censored runs with the exact algorithm cutoff yields better results than imputing with $10 \cdot C$ as done for PAR10.

Due to the computational expense of iteratively refitting the regression models, the majority of the experimental runs for the SCHMEE&HAHN baseline method did not manage to finish within a given time frame of 96 CPU days.

[2] `https://github.com/JonasHanselle/Superset_Learning_AS`

Table 2. PAR10 scores of the proposed approach as well as all considered baselines for all considered scenarios.

Scenario	CLIP	IGNORE	PAR10	SCHMEE&HAHN	SUPERSET
ASP-POTASSCO	**267.16** (1)	785.49 (5) •	285.82 (3)	312.05 (4) •	267.75 (2)
BNSL-2016	**2333.59** (1)	10221.65 (5) •	5829.51 (4) •	2350.71 (2)	2392.99 (3)
CPMP-2015	6223.74 (3)	11490.41 (5) •	6683.52 (4)	**5754.33** (1)	6020.67 (2)
CSP-2010	**723.25** (1)	989.05 (4)	1081.57 (5) •	814.91 (3)	726.05 (2)
CSP-MZN-2013	1384.45 (2)	12466.45 (5) •	1460.69 (3) •	1582.99 (4) •	**1337.03** (1)
CSP-Minizinc-Time-2016	2417.4 (2)	7416.88 (5) •	3314.75 (3)	3394.89 (4)	**2417.29** (1)
MAXSAT-PMS-2016	605.77 (3)	8793.23 (5) •	2474.75 (4) •	563.13 (2)	**542.93** (1)
MAXSAT-WPMS-2016	2391.88 (3) •	14633.54 (5) •	4173.15 (4) •	2124.11 (2)	**2100.98** (1)
MAXSAT12-PMS	961.97 (3) •	2141.06 (5) •	1313.61 (4)	866.79 (2)	**814.62** (1)
MAXSAT15-PMS-INDU	**905.35** (1)	16085.16 (5) •	2658.04 (4) •	1116.16 (3) •	930.83 (2)
MIP-2016	**9403.18** (2) ○	20182.02 (5) •	10097.0 (4)	9784.38 (3)	**9403.18** (1)
PROTEUS-2014	5198.77 (2)	20111.7 (5) •	5623.13 (4)	5508.62 (3)	**5095.73** (1)
QBF-2011	**3601.4** (1)	20402.16 (5) •	6343.31 (4) •	3886.31 (3)	3771.58 (2)
QBF-2014	1706.53 (4) •	4345.37 (5) •	1666.69 (3) •	1469.3 (2)	**1455.48** (1)
QBF-2016	2552.29 (2)	6431.72 (5) •	2997.55 (4)	2766.28 (3)	**2531.79** (1)
SAT03-16_INDU	**3528.8** (1) ○	7254.0 (5) •	3999.94 (4)	3836.84 (3)	3581.81 (2)
SAT11-HAND	11831.89 (3) ○	24563.68 (5) •	11960.84 (4)	**11399.28** (1)	11831.89 (2)
SAT11-INDU	**7107.54** (2) ○	12992.03 (5) •	8419.32 (4)	7326.67 (3)	**7107.54** (1)
SAT11-RAND	**3376.17** (1)	16395.14 (5) •	4193.79 (4)	3929.21 (3)	3380.65 (2)
SAT12-ALL	**1635.69** (1)	8439.57 (5) •	1643.47 (3)	2073.78 (4) •	1639.68 (2)
SAT12-HAND	2612.79 (4)	8046.59 (5) •	2587.83 (3)	2554.14 (2)	**2551.2** (1)
SAT12-INDU	1857.48 (2)	11204.39 (5) •	3411.23 (4) •	2123.33 (3) •	**1840.33** (1)
SAT12-RAND	**568.54** (1)	10576.49 (5) •	1075.56 (4) •	665.15 (3)	659.1 (2)
SAT15-INDU	6133.29 (2)	11637.43 (5) •	8575.57 (4)	7548.6 (3) •	**6126.67** (1)
TSP-LION2015	**485.22** (1)	9585.78 (5) •	743.14 (4) •	650.07 (3)	496.57 (2)
avg. rank	1.96	4.96	3.80	2.76	**1.52**
avg. PAR10	3192.57	11087.64	4104.55	3376.08	**3160.97**

While it is evident that the superset learning approach has a clear advantage with respect to runtime over iterative imputation methods such as SCHMEE&HAHN, it also exhibits the strongest predictive accuracy for the task of algorithm selection in the conducted experiments[3].

5 Conclusion

In this paper, we proposed to consider algorithm selection within the framework of superset learning to handle censored data in a proper way. Instead of replacing the censored labels with more or less arbitrary precise numbers, or leaving out these data points entirely, the labels are considered as imprecise observations in the form of intervals (ranging from the cutoff time to infinity). Invoking a principle of generalized loss minimization, a regression model is then built to predict (precise) runtimes of algorithms for unseen problem instances.

Our experimental evaluation reveals that working with these imprecisely annotated data, the generalization performance of the fitted models is significantly

[3] Note that for the TSP-LION2015 scenario, only 9 out of 10 folds were evaluated using the SCHMEE&HAHN imputation due to technical issues.

better compared to naïve strategies for dealing with censored data. Moreover, we showed superset learning to be competitive or superior to the imputation technique proposed in [21]. Instead of repeatedly rebuilding the model until convergence of the predicted values, each model needs to be fit only once. As the number of such models equals the number of algorithms to choose from, superset learning has a clear advantage over SCHMEE&HAHN in terms of runtime complexity.

Interesting directions for future work include integrating superset learning with a hybrid regression & ranking approach [8], or to consider the decision which algorithm to choose according to the predicted runtimes in terms of a stacked classifier instead of simply taking the algorithm with minimum predicted runtime [16]. Besides, as already said, superset learning is a quite versatile approach that allows for modeling imprecision in a flexible way. This might also be beneficial in the context of algorithm selection, where censoring is certainly not the only source of imprecision.

Acknowledgments. This work was supported by the German Federal Ministry of Economic Affairs and Energy (BMWi) within the "Innovationswettbewerb Künstliche Intelligenz" and the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901/3 project no. 160364472). The authors also gratefully acknowledge support of this project through computing time provided by the Paderborn Center for Parallel Computing (PC^2).

References

1. Amadini, R., Gabbrielli, M., Mauro, J.: SUNNY: a lazy portfolio approach for constraint solving. Theory Pract. Log. Program. **14**(4-5) (2014)
2. Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H.H., Hutter, F., Leyton-Brown, K., Tierney, K., Vanschoren, J.: Aslib: A benchmark library for algorithm selection. Artif. Intell. **237**, 41–58 (2016)
3. Cabannes, V., Rudi, A., Bach, F.: Structured prediction with partial labelling through the infimum loss. In: Proc. ICML, Int. Conf. on Machine Learning (2020)
4. Eggensperger, K., Lindauer, M., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Efficient benchmarking of algorithm configurators via model-based surrogates. Mach. Learn. **107**(1) (2018)
5. Gagliolo, M., Legrand, C.: Algorithm survival analysis. In: Experimental methods for the analysis of optimization algorithms, pp. 161–184. Springer (2010)
6. Gagliolo, M., Schmidhuber, J.: Leaning dynamic algorithm portfolios. Annals of Mathematics and Artificial Intelligence **47**, 295–328 (2006)
7. Gomes, C.P., Selman, B., Crato, N.: Heavy-tailed distributions in combinatorial search. In: CP (1997)
8. Hanselle, J., Tornede, A., Wever, M., Hüllermeier, E.: Hybrid ranking and regression for algorithm selection. In: German Conference on Artificial Intelligence (Künstliche Intelligenz). Springer (2020)
9. Hüllermeier, E., Cheng, W.: Superset learning based on generalized loss minimization. In: ECML PKDD (2015)

10. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Bayesian optimization with censored response data. In: NIPS workshop on Bayesian Optimization, Sequential Experimental Design, and Bandits (2011)
11. Hüllermeier, E.: Learning from imprecise and fuzzy observations: Data disambiguation through generalized loss minimization. International Journal of Approximate Reasoning **55**(7), 1519 – 1534 (2014), special issue: Harnessing the information contained in low-quality data sources
12. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC - instance-specific algorithm configuration. In: ECAI (2010)
13. Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated algorithm selection: Survey and perspectives. Evol. Comput. **27**(1) (2019)
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
15. Kleinbaum, D.G., Klein, M.: Survival analysis, vol. 3. Springer (2010)
16. Kotthoff, L.: Hybrid regression-classification models for algorithm selection. In: ECAI. pp. 480–485 (2012)
17. Lindauer, M., Bergdoll, R.D., Hutter, F.: An empirical study of per-instance algorithm scheduling. In: International Conference on Learning and Intelligent Optimization. pp. 253–259. Springer (2016)
18. Lobjois, L., Lemaître, M., et al.: Branch and bound algorithm selection by performance prediction. In: AAAI/IAAI. pp. 353–358 (1998)
19. Pihera, J., Musliu, N.: Application of machine learning to algorithm selection for TSP. In: 26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014. pp. 47–54. IEEE Computer Society (2014)
20. Rice, J.R.: The algorithm selection problem. Advances in Computers **15** (1976)
21. Schmee, J., Hahn, G.J.: A simple method for regression analysis with censored data. Technometrics **21**(4) (1979)
22. Tornede, A., Wever, M., Hüllermeier, E.: Extreme algorithm selection with dyadic feature representation. In: Discovery Science (2020)
23. Tornede, A., Wever, M., Hüllermeier, E.: Towards meta-algorithm selection. In: Workshop on Meta-Learning (MetaLearn 2020) @ NeurIPS 2020 (2020)
24. Tornede, A., Wever, M., Werner, S., Mohr, F., Hüllermeier, E.: Run2survive: a decision-theoretic approach to algorithm selection based on survival analysis. In: Asian Conference on Machine Learning. pp. 737–752. PMLR (2020)
25. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: Hydra-mip: Automated algorithm configuration and selection for mixed integer programming. In: RCRA workshop @ IJCAI (2011)
26. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla-07: the design and analysis of an algorithm portfolio for sat. In: CP. Springer (2007)