Chapter 2

Meta-Learning

Joaquin Vanschoren

Abstract

Meta-learning, or *learning to learn*, is the science of systematically observing how different machine learning approaches perform on a wide range of learning tasks, and then learning from this experience, or *meta-data*, to learn new tasks much faster than otherwise possible. Not only does this dramatically speed up and improve the design of machine learning pipelines or neural architectures, it also allows us to replace hand-engineered algorithms with novel approaches learned in a data-driven way. In this chapter, we provide an overview of the state of the art in this fascinating and continuously evolving field.

2.1 Introduction

When we learn new skills, we rarely - if ever - start from scratch. We start from skills learned earlier in related tasks, reuse approaches that worked well before, and focus on what is likely worth trying based on experience [82]. With every skill learned, learning new skills becomes easier, requiring fewer examples and less trial-and-error. In short, we *learn how to learn* across tasks. Likewise, when building machine learning models for a specific task, we often build on experience with related tasks, or use our (often implicit) understanding of the behavior of machine learning techniques to help make the right choices.

The challenge in meta-learning is to learn from prior experience in a systematic, data-driven way. First, we need to collect *meta-data* that describe prior learning tasks and previously learned models. They comprise the exact *algorithm configurations* used to train the models, including hyperparameter settings, pipeline compositions and/or network architectures, the resulting *model evaluations*, such as accuracy and training time, the learned model parameters, such as the trained weights of a neural net, as well as measurable properties of the task itself, also known as *meta-features*. Second, we need to *learn* from this prior meta-data, to extract and transfer knowledge that guides the search for optimal models for new tasks. This chapter presents a concise overview of different meta-learning approaches to do this effectively.

The term *meta-learning* covers any type of learning based on prior experience with other tasks. The more *similar* those previous tasks are, the more types of meta-data we can leverage, and defining task similarity will be a key overarching challenge. Perhaps needless to say, there is no free lunch [186, 57]. When a new task represents completely unrelated phenomena, or random noise, leveraging prior experience will not be effective. Luckily, in real-world tasks, there are plenty of opportunities to learn from prior experience.

In the remainder of this chapter, we categorize meta-learning techniques based on the type of meta-data they leverage, from the most general to the most task-specific. First, in Section 2.2, we discuss how to *learn purely from model evaluations*. These techniques can be used to recommend generally useful configurations and configuration search spaces, as well as transfer knowledge from *empirically similar* tasks. In Section 2.3, we discuss how we can *characterize* tasks to more explicitly express task similarity and build meta-models that learn the relationships between data characteristics and learning performance. Finally, Section 2.4 covers how we can *transfer trained model parameters* between tasks that are inherently similar, e.g. sharing the same input features, which enables transfer learning [111] and few-shot learning [125].

Note that while *multi-task learning* [25] (learning multiple related tasks simultaneously) and *ensemble learning* [35] (building multiple models on the same task), can often be meaningfully combined with meta-learning systems, they do not in themselves involve learning from prior experience on other tasks.

2.2 Learning from Model Evaluations

Consider that we have access to prior tasks $t_j \in T$, the set of all known tasks, as well as a set of learning algorithms, fully defined by their *configurations* $\theta_i \in \Theta$; here Θ represents a discrete, continuous, or mixed configuration space which can cover hyperparameter settings, pipeline components and/or network architecture components. $\mathbf{P}$ is the set of all prior scalar evaluations $P_{i,j} = P(\theta_i, t_j)$ of configuration θ_i on task t_j, according to a predefined evaluation measure, e.g. accuracy, and model evaluation technique, e.g. cross-validation. $\mathbf{P}_{new}$ is the set of known evaluations $P_{i,new}$ on a new task t_{new}. We now want to train a *meta-learner* L that predicts recommended configurations Θ^*_{new} for a new task t_{new}. The meta-learner is trained on meta-data $\mathbf{P} \cup \mathbf{P}_{new}$. $\mathbf{P}$ is usually gathered beforehand, or extracted from meta-data repositories [173, 175]. $\mathbf{P}_{new}$ is learned by the meta-learning technique itself in an iterative fashion, sometimes *warm-started* with an initial $\mathbf{P}'_{new}$ generated by another method.

2.2.1 Task-Independent Recommendations

First, imagine not having access to any evaluations on t_{new}, hence $\mathbf{P}_{new} = \varnothing$. We can then still learn a function $f : \Theta \times T \rightarrow \{\theta^*_k\}$, $k = 1..K$, yielding a set of recommended configurations *independent* of t_{new}. These θ^*_k can then be evaluated on t_{new} to select the best one, or to warm-start further optimization approaches, such as those discussed in Section 2.2.3.

Such approaches often produce a ranking, i.e. an *ordered* set θ^*_k. This is typically done by discretizing Θ into a set of candidate configurations θ_i, also called a *portfolio*, evaluated on a large number of tasks t_j. We can then build a ranking per task, for instance using *success rates*, *AUC*, or *significant wins* [21, 34, 85]. However, it is often desirable that equally good but faster algorithms are ranked higher, and multiple methods have been proposed to trade off accuracy and training time [21, 133]. Next, we can aggregate these single-task rankings into a *global ranking*, for instance by computing the average rank [91, 1] across all tasks. When there is insufficient data to build a global ranking, one can recommend *subsets of configurations* based on the best known configurations for each prior task [172, 70], or return *quasi-linear rankings* [30].

To find the best θ^* for a task t_{new}, never before seen, a simple anytime method is to select the top-K configurations [21], going down the list and evaluating each configuration on t_{new} in turn. This evaluation can be halted after a predefined value for K, a time budget, or when a sufficiently accurate model is found. In time-constrained settings, it has been shown that multi-objective rankings (including training time) converge to near-optimal models much faster [1, 133], and provide a strong baseline for algorithm comparisons [1, 85].

A very different approach to the one above is to first fit a differentiable function $f_j(\theta_i) = P_{i,j}$ on all prior evaluations of a specific task t_j, and then use gradient descent to find an optimized configuration θ^*_j per prior task [184]. Assuming that some of the tasks t_j will be similar to t_{new}, those θ^*_j will be useful for warm-starting Bayesian optimization approaches.

2.2.2 Configuration Space Design

Prior evaluations can also be used to learn a better *configuration space* Θ^*. While again independent from t_{new}, this can radically speed up the search for optimal models, since only the more relevant regions of the configuration space are explored. This is critical when computational resources are limited, and proves to be an important factor in practical comparisons of AutoML systems [33].

First, in the functional ANOVA [67] approach, hyperparameters are deemed important if they explain most of the variance in algorithm performance on a given task. In [135], this was explored using 250,000 OpenML experiments with 3 algorithms across 100 datasets.

An alternative approach is to first *learn* an optimal hyperparameter default setting, and then define hyperparameter importance as the *performance gain* that can be achieved by tuning the hyperparameter instead of leaving it at that default value. Indeed, even though a hyperparameter may cause a lot of variance, it may also have one specific setting that always results in good performance. In [120], this was done using about 500,000 OpenML experiments on 6 algorithms and 38 datasets. Default values are learned *jointly* for all hyperparameters of an algorithm by first training surrogate models for that algorithm for a large number of tasks. Next, many configurations are sampled, and the configuration that minimizes the average risk across all tasks is the recommended default configuration. Finally, the importance (or *tunability*) of each hyperparameter is estimated by observing how much improvement can still be gained by tuning it.

In [181], defaults are learned *independently* from other hyperparameters, and defined as the configurations that occur most frequently in the top-K configurations for every task. In the case that the optimal default value depends on meta-features (e.g. the number of training instances or features), simple functions are learned that include these meta-features. Next, a statistical test defines whether a hyperparameter can be safely left at this default, based on the *performance loss* observed when *not* tuning a hyperparameter (or a set of hyperparameters), while all other parameters are tuned. This was evaluated using 118,000 OpenML experiments with 2 algorithms (SVMs and Random Forests) across 59 datasets.

2.2.3 Configuration Transfer

If we want to provide recommendations for a specific task t_{new}, we need additional information on how similar t_{new} is to prior tasks t_j. One way to do this is to evaluate a number of recommended (or potentially random) configurations on t_{new}, yielding new evidence $\mathbf{P}_{new}$. If we then observe that the evaluations $P_{i,new}$ are similar to $P_{i,j}$, then t_j and t_{new} can be considered intrinsically similar, based on empirical evidence. We can include this knowledge to train a meta-learner that predicts a recommended set of configurations Θ^*_{new} for t_{new}. Moreover, every selected θ^*_{new} can be evaluated and included in $\mathbf{P}_{new}$, repeating the cycle and collecting more empirical evidence to learn which tasks are similar to each other.

Relative Landmarks

A first measure for task similarity considers the relative (pairwise) performance differences, also called *relative landmarks*, $RL_{a,b,j} = P_{a,j} - P_{b,j}$ between two configurations θ_a and θ_b on a particular task t_j [53]. *Active testing* [85] leverages these as follows: it warm-starts with the globally best configuration (see Section 2.2.1), calls it θ_{best}, and proceeds in a tournament-style fashion. In each round, it selects the 'competitor' θ_c that most convincingly outperforms θ_{best} on similar tasks. It deems tasks to be similar if the relative landmarks of all evaluated configurations are similar, i.e., if the configurations perform similarly on both t_j and t_{new} then the tasks are deemed similar. Next, it evaluates the competitor θ_c, yielding $P_{c,new}$, updates the task similarities, and repeats. A limitation of this method is that it can only consider configurations θ_i that were evaluated on many prior tasks.

Surrogate Models

A more flexible way to transfer information is to build *surrogate models* $s_j(\theta_i) = P_{i,j}$ for all prior tasks t_j, trained using all available $\mathbf{P}$. One can then define task similarity in terms of the error between $s_j(\theta_i)$ and $P_{i,new}$: if the surrogate model for t_j can generate accurate predictions for t_{new}, then those tasks are intrinsically similar. This is usually done in combination with Bayesian optimization (see Chapter 1) to determine the next θ_i.

Wistuba et al. [185] train surrogate models based on Gaussian Processes (GPs) for every prior task, plus one for t_{new}, and combine them into a weighted, normalized sum, with the (new) mean μ defined as the weighted sum of the individual μ_j's (obtained from prior tasks t_j). The weights of the μ_j's are computed using the Nadaraya-Watson kernel-weighted average, where each task is represented as a vector of relative landmarks, and the Epanechnikov quadratic kernel [104] is used to measure the similarity between the relative landmark vectors of t_j and t_{new}. The more similar t_j is to t_{new}, the larger the weight s_j, increasing the influence of the surrogate model for t_j.

Feurer et al. [45] propose to combine the predictive distributions of the individual Gaussian processes, which makes the combined model a Gaussian process again. The weights are computed following the agnostic Bayesian ensemble of Lacoste et al. [81], which weights predictors according to an estimate of their generalization performance.

Meta-data can also be transferred in the acquisition function rather than the surrogate model [185]. The surrogate model is only trained on $P_{i,new}$, but the next θ_i to evaluate is provided by an acquisition function which is the weighted average of the expected improvement [69] on $P_{i,new}$ and the predicted improvements on all prior $P_{i,j}$. The weights of the prior tasks can again be defined via the accuracy of the surrogate model or via relative landmarks. The weight of the expected improvement component is gradually increased with every iteration as more evidence $P_{i,new}$ is collected.

Warm-Started Multi-task Learning

Another approach to relate prior tasks t_j is to learn a joint task representation using $\mathbf{P}$. In [114], task-specific Bayesian linear regression [20] surrogate models $s_j(\theta_i)$ are trained and combined in a feedforward Neural Network $NN(\theta_i)$ which learns a joint task representation that can accurately predict $P_{i,new}$. The surrogate models are pre-trained on OpenML meta-data to provide a warm-start for optimizing $NN(\theta_i)$ in a multi-task learning setting. Earlier work on multi-task learning [165] assumed that we already have a set of 'similar' source tasks t_j. It transfers information between these t_j and t_{new} by building a joint GP model for Bayesian optimization that learns and exploits the exact relationship between the tasks. Learning a joint GP tends to be less scalable than building one GP per task, though. Springenberg et al. [160] also assumes that the tasks are related and similar, but learns the relationship between tasks during the optimization process using Bayesian Neural Networks. As such, their method is somewhat of a hybrid of the previous two approaches. Golovin et al. [58] assume a sequence order (e.g., time) across tasks. It builds a stack of GP regressors, one per task, training each GP on the residuals relative to the regressor below it. Hence, each task uses the tasks before it as its priors.

Other Techniques

Multi-armed bandits [138] provide yet another approach to find the source tasks t_j most related to t_{new} [124]. In this analogy, each t_j is one arm, and the (stochastic) reward for selecting (pulling) a particular prior task (arm) is defined in terms of the error in the predictions of a GP-based Bayesian optimizer that models the prior evaluations of t_j as noisy measurements and combines them with the existing evaluations on t_{new}. The cubic scaling of the GP makes this approach less scalable, though.

Another way to define task similarity is to take the existing evaluations $P_{i,j}$, use Thompson Sampling [166] to obtain the optima distribution ρ^j_{max}, and then measure the KL-divergence [80] between ρ^j_{max} and ρ^{new}_{max} [123]. These distributions are then merged into a mixture distribution based on the similarities and used to build an acquisition function that predicts the next most promising configuration to evaluate. It is so far only evaluated to tune 2 SVM hyperparameters using 5 tasks.

Finally, a complementary way to leverage $\mathbf{P}$ is to recommend which configurations should *not* be used. After training surrogate models per task, we can look up which t_j are most similar to t_{new}, and then use $s_j(\theta_i)$ to discover regions of Θ where performance is predicted to be poor. Excluding these regions can speed up the search for better-performing ones. Wistuba et al. [183], do this using a task similarity measure based on the Kendall tau rank correlation coefficient [73] between the ranks obtained by ranking configurations θ_i using $P_{i,j}$ and $P_{i,new}$, respectively.

2.2.4 Learning Curves

We can also extract meta-data about the training process itself, such as how fast model performance improves as more training data is added. If we divide the training in steps s_t, usually adding a fixed number of training examples every step, we can measure the performance $P(\theta_i, t_j, s_t) = P_{i,j,t}$ of configuration θ_i on task t_j after step s_t, yielding a *learning curve* across the time steps s_t. As discussed in Chapter 1, learning curves are also used to speed up hyperparameter optimization on a given task. In meta-learning, learning curve information is transferred across tasks.

While evaluating a configuration on new task t_{new}, we can halt the training after a certain number of iterations $r < t$, and use the partially observed learning curve to predict how well the configuration will perform on the full dataset based on prior experience with other tasks, and decide whether to continue the training or not. This can significantly speed up the search for good configurations.

One approach is to assume that similar tasks yield similar learning curves. First, define a distance between tasks based on how similar the partial learning curves are: $dist(t_a, t_b) = f(P_{i,a,t}, P_{i,b,t})$ with $t = 1, ..., r$. Next, find the k most similar tasks $t_{1..k}$ and use their complete learning curves to predict how well the configuration will perform on the new complete dataset. Task similarity can be measured by comparing the shapes of the partial curves across all configurations tried, and the prediction is made by adapting the 'nearest' complete curve(s) to the new partial curve [83, 84]. This approach was also successful in combination with active testing [86], and can be sped up further by using multi-objective evaluation measures that include training time [133].

Interestingly, while several methods aim to predict learning curves during neural architecture search (see Chapter 3), as of yet none of this work leverages learning curves previously observed on other tasks.

2.3 Learning from Task Properties

Another rich source of meta-data are characterizations (meta-features) of the task at hand. Each task $t_j \in T$ is described with a vector $m(t_j) = (m_{j,1}, ..., m_{j,K})$ of K meta-features $m_{j,k} \in M$, the set of all known meta-features. This can be used to define a task similarity measure based on, for instance, the Euclidean distance between $m(t_i)$ and $m(t_j)$, so that we can transfer information from the most similar tasks to the new task t_{new}. Moreover, together with prior evaluations $\mathbf{P}$, we can train a *meta-learner* L to predict the performance $P_{i,new}$ of configurations θ_i on a new task t_{new}.

2.3.1 Meta-Features

Table 2.1 provides a concise overview of the most commonly used meta-features, together with a short rationale for why they are indicative of model performance. Where possible, we also show the formulas to compute them. More complete surveys can be found in the literature [137, 174, 98, 129, 26].

To build a meta-feature vector $m(t_j)$, one needs to select and further process these meta-features. Studies on OpenML meta-data have shown that the optimal set of meta-features depends on the application [17]. Many meta-features are computed on single features, or combinations of features, and need to be aggregated by summary statistics (min,max,μ,σ,quartiles,...) or histograms [72]. One needs to systematically extract and aggregate them [117]. When computing task similarity, it is also important to normalize all meta-features [9], perform feature selection [171], or employ dimensionality reduction techniques (e.g. PCA) [17]. When learning meta-models, one can also use relational meta-learners [172] or case-based reasoning methods [92, 63, 71].

Beyond these general-purpose meta-features, many more specific ones were formulated. For streaming data one can use streaming landmarks [134, 136],

Name	Formula	Rationale	Variants
Nr instances	n	Speed, Scalability [99]	p/n, $log(n)$, log(n/p)
Nr features	p	Curse of dimensionality [99]	$log(p)$, % categorical
Nr classes	c	Complexity, imbalance [99]	ratio min/maj class
Nr missing values	m	Imputation effects [70]	% missing
Nr outliers	o	Data noisiness [140]	o/n
Skewness	$\frac{E(X-\mu_X)^3}{\sigma_X^3}$	Feature normality [99]	min,max,μ,σ,q_1, q_3
Kurtosis	$\frac{E(X-\mu_X)^4}{\sigma_X^4}$	Feature normality [99]	min,max,μ,σ,q_1, q_3
Correlation	$\rho_{X_1 X_2}$	Feature interdependence [99]	min,max,μ,σ,ρ_{XY}[157]
Covariance	$cov_{X_1 X_2}$	Feature interdependence [99]	min,max,μ,σ,cov_{XY}
Concentration	$\tau_{X_1 X_2}$	Feature interdependence [72]	min,max,μ,σ,τ_{XY}
Sparsity	sparsity(X)	Degree of discreteness [142]	min,max,μ,σ
Gravity	gravity(X)	Inter-class dispersion [5]	
ANOVA p-value	$p_{val_{\mathtt{X}_1 X_2}}$	Feature redundancy [70]	$p_{val_{XY}}$[157]
Coeff. of variation	$\frac{\sigma_Y}{\mu_Y}$	Variation in target [157]	
PCA ρ_{λ_1}	$\sqrt{\frac{\lambda_1}{1+\lambda_1}}$	Variance in first PC [99]	$\frac{\lambda_1}{\sum_i \lambda_i}$[99]
PCA skewness		Skewness of first PC [48]	PCA kurtosis [48]
PCA 95%	$\frac{dim_{95\%var}}{p}$	Intrinsic dimensionality [9]	
Class probability	$P(\mathtt{C})$	Class distribution [99]	min,max,μ,σ
Class entropy	$H(\mathtt{C})$	Class imbalance [99]	
Norm. entropy	$\frac{H(\mathtt{X})}{log_2 n}$	Feature informativeness [26]	min,max,μ,σ
Mutual inform.	$MI(\mathtt{C},\mathtt{X})$	Feature importance [99]	min,max,μ,σ
Uncertainty coeff.	$\frac{MI(\mathtt{C},\mathtt{X})}{H(\mathtt{C})}$	Feature importance [3]	min,max,μ,σ
Equiv. nr. feats	$\frac{H(C)}{\overline{MI(C,X)}}$	Intrinsic dimensionality [99]	
Noise-signal ratio	$\frac{\overline{H(X)}-\overline{MI(C,X)}}{\overline{MI(C,X)}}$	Noisiness of data [99]	
Fisher's discrimin.	$\frac{(\mu_{c1}-\mu_{c2})^2}{\sigma_{c1}^2-\sigma_{c2}^2}$	Separability classes c_1, c_2 [64]	See [64]
Volume of overlap		Class distribution overlap [64]	See [64]
Concept variation		Task complexity [178]	See [177, 178]
Data consistency		Data quality [76]	See [76]
Nr nodes, leaves	$\lvert\eta\rvert, \lvert\psi\rvert$	Concept complexity [113]	Tree depth
Branch length		Concept complexity [113]	min,max,μ,σ
Nodes per feature	$\lvert\eta_X\rvert$	Feature importance [113]	min,max,μ,σ
Leaves per class	$\frac{\lvert\psi_c\rvert}{\lvert\psi\rvert}$	Class complexity [49]	min,max,μ,σ
Leaves agreement	$\frac{n_{\psi_i}}{n}$	Class separability [16]	min,max,μ,σ
Information gain		Feature importance [16]	min,max,μ,σ, gini
Landmarker(1NN)	$P(\theta_{1NN}, t_j)$	Data sparsity [115]	Elite 1NN [115]
Landmarker(Tree)	$P(\theta_{Tree}, t_j)$	Data separability [115]	Stump,RandomTree
Landmarker(Lin)	$P(\theta_{Lin}, t_j)$	Linear separability [115]	Lin.Disciminant
Landmarker(NB)	$P(\theta_{NB}, t_j)$	Feature independence [115]	More models [14, 88]
Relative LM	$P_{a,j} - P_{b,j}$	Probing performance [53]	
Subsample LM	$P(\theta_i, t_j, s_t)$	Probing performance [159]	

Table 2.1: Overview of commonly used meta-features. Groups from top to bottom: simple, statistical, information-theoretic, complexity, model-based, and landmarkers. Continuous features X and target Y have mean μ_X, stdev σ_X, variance σ_X^2. Categorical features $\mathtt{X}$ and class $\mathtt{C}$ have categorical values π_i, conditional probabilities $\pi_{i|j}$, joint probabilities $\pi_{i,j}$, marginal probabilities $\pi_{i+} = \sum_j \pi_{ij}$, entropy $H(\mathtt{X}) = -\sum_i \pi_{i+} log_2(\pi_{i+})$.

for time series data one can compute autocorrelation coefficients or the slope of regression models [7, 121, 146], and for unsupervised problems one can cluster the data in different ways and extract properties of these clusters [158]. In many applications, domain-specific information can be leveraged as well [155, 109].

2.3.2 Learning Meta-Features

Instead of manually defining meta-features, we can also *learn* a joint representation for groups of tasks. One approach is to build meta-models that generate a landmark-like meta-feature representation M' given other task meta-features M and trained on performance meta-data $\mathbf{P}$, or $f : M \mapsto M'$. Sun and Pfahringer [164] do this by evaluating a predefined set of configurations θ_i on all prior tasks t_j, and generating a binary metafeature $m_{j,a,b} \in M'$ for every pairwise combination of configurations θ_a and θ_b, indicating whether θ_a outperformed θ_b or not, thus $m'(t_j) = (m_{j,a,b}, m_{j,a,c}, m_{j,b,c}, ...)$. To compute $m_{new,a,b}$, *meta-rules* are learned for every pairwise combination (a,b), each predicting whether θ_a will outperform θ_b on task t_j, given its other meta-features $m(t_j)$.

We can also learn a joint representation based entirely on the available $\mathbf{P}$ meta-data, i.e. $f : \mathbf{P} \times \Theta \mapsto M'$. We previously discussed how to do this with feed-forward neural nets [114] in Section 2.2.3. If the tasks share the same input space, e.g., they are images of the same resolution, one can also use Siamese networks to learn a meta-feature representation [75]. These are trained by feeding the data of two different tasks to two twin networks, and using the differences between the predicted and observed performance $P_{i,new}$ as the error signal. Since the model parameters between both networks are tied in a Siamese network, two very similar tasks are mapped to the same regions in the latent meta-feature space. They can be used for warm starting Bayesian hyperparameter optimization [75] and neural architecture search [2].

2.3.3 Warm-Starting Optimization from Similar Tasks

Meta-features are a very natural way to estimate task similarity and initialize optimization procedures based on promising configurations on similar tasks. This is akin to how human experts start a manual search for good models, given experience on related tasks.

Starting a *genetic search* algorithm in regions of the search space with promising solutions can significantly speed up convergence to a good solution. Gomes et al. [59] recommend initial configurations by finding the k most similar prior tasks t_j based on the L1 distance between vectors $m(t_j)$ and $m(t_{new})$, where each $m(t_j)$ includes 17 simple and statistical meta-features. For each of the k most similar tasks, the best configuration is evaluated on t_{new}, and used to initialize a genetic search algorithm (Particle Swarm Optimization), as well as Tabu Search. Reif et al. [128] follow a very similar approach, using 15 simple, statistical, and landmarking meta-features. They use a forward selection technique to find the most useful meta-features, and warm-start a standard genetic algorithm (GAlib) with a modified Gaussian mutation operation. Variants of active testing (see Sect. 2.2.3) that use meta-features were also tried [100, 85], but did not perform better than the approaches based on relative landmarks.

Also model-based optimization approaches can benefit greatly from an initial set of promising configurations. SCoT [9] trains a single surrogate ranking model $f : M \times \Theta \rightarrow R$, predicting the rank of θ_i on task t_j. M contains 4 meta-features (3 simple ones and one based on PCA). The surrogate model is trained on all the rankings, including those on t_{new}. Ranking is used because the scale of evaluation values can differ greatly between tasks. A GP regression converts the ranks to probabilities to do Bayesian optimization, and each new $P_{i,new}$ is used to retrain the surrogate model after every step.

Schilling et al. [147] use a modified multilayer perceptron as a surrogate model, of the form $s_j(\theta_i, m(t_j), b(t_j)) = P_{i,j}$ where $m(t_j)$ are the meta-features and $b(t_j)$ is a vector of j binary indications which are 1 if the meta-instance is from t_j and 0 otherwise. The multi-layer perceptron uses a modified activation function based on factorization machines [131] in the first layer, aimed at learning a latent representation for each task to model task similarities. Since this model cannot represent uncertainties, an ensemble of 100 multilayer perceptrons is trained to get predictive means and simulate variances.

Training a single surrogate model on all prior meta-data is often less scalable. Yogatama and Mann [188] also build a single Bayesian surrogate model, but only include tasks similar to t_{new}, where task similarity is defined as the Euclidean distance between meta-feature vectors consisting of 3 simple meta-features. The $P_{i,j}$ values are standardized to overcome the problem of different scales for each t_j. The surrogate model learns a Gaussian process with a specific kernel combination on all instances.

Feurer et al. [48] offer a simpler, more scalable method that warm-starts Bayesian optimization by sorting all prior tasks t_j similar to [59], but including 46 simple, statistical, and landmarking meta-features, as well as $H(\mathtt{C})$. The t best configurations on the d most similar tasks are used to warm-start the surrogate model. They search over many more hyperparameters than earlier work, including preprocessing steps. This warm-starting approach was also used in later work [46], which is discussed in detail in Chapter 6.

Finally, one can also use *collaborative filtering* to recommend promising configurations [161]. By analogy, the tasks t_j (users) provide ratings ($P_{i,j}$) for the configurations θ_i (items), and matrix factorization techniques are used to predict unknown $P_{i,j}$ values and recommend the best configurations for any task. An important issue here is the cold start problem, since the matrix factorization requires at least some evaluations on t_{new}. Yang et al. [187] use a D-optimal experiment design to sample an initial set of evaluations $P_{i,new}$. They predict both the predictive performance and runtime, to recommend a set of warm-start configurations that are both accurate and fast. Misir and Sebag [102, 103] leverage meta-features to solve the cold start problem. Fusi et al. [54] also use meta-features, following the same procedure as [46], and use a probabilistic matrix factorization approach that allows them to perform Bayesian optimization to further optimize their pipeline configurations θ_i. This approach also yields useful latent embeddings of both the tasks and configurations.

2.3.4 Meta-Models

We can also *learn* the complex relationship between a task's meta-features and the utility of specific configurations by building a meta-model L that recommends the most useful configurations Θ^*_{new} given the meta-features M of the new task t_{new}. There exists a rich body of earlier work [22, 87, 56, 94] on building meta-models for algorithm selection [15, 115, 70, 19] and hyperparameter recommendation [79, 157, 4, 108]. Experiments showed that boosted and bagged trees often yielded the best predictions, although much depends on the exact meta-features used [72, 76].

Ranking

Meta-models can also generate a *ranking* of the top-K most promising configurations. One approach is to build a k-nearest neighbor (kNN) meta-model to predict which tasks are similar, and then rank the best configurations on these similar tasks [23, 146]. This is similar to the work discussed in Section 2.3.3, but without ties to a follow-up optimization approach. Meta-models specifically meant for ranking, such as predictive clustering trees [170] and label ranking trees [29] were also shown to work well. Approximate Ranking Trees Forests (ART Forests) [164], ensembles of fast ranking trees, prove to be especially effective, since they have 'built-in' meta-feature selection, work well even if few prior tasks are available, and the ensembling makes the method more robust. *autoBagging* [116] ranks Bagging workflows including four different Bagging hyperparameters, using an XGBoost-based ranker, trained on 140 OpenML datasets and 146 meta-features. Lorena et al. [93] recommends SVM configurations for regression problems using a kNN meta-model and a new set of meta-features based on data complexity.

Performance Prediction

Meta-models can also directly predict the performance, e.g. accuracy or training time, of a configuration on a given task, given its meta-features. This allows us to estimate whether a configuration will be interesting enough to evaluate in any optimization procedure. Early work used linear regression or rule-base regressors to predict the performance of a discrete set of configurations and then rank them accordingly [14, 77]. Guerra et al. [61] train an SVM meta-regressor per classification algorithm to predict its accuracy, under default settings, on a new task t_{new} given its meta-features. Reif et al. [129] train a similar meta-regressor on more meta-data to predict its *optimized* performance. Davis et al. [32] use a MultiLayer Perceptron based meta-learner instead, predicting the performance of a specific algorithm configuration.

Instead of predicting predictive performance, a meta-regressor can also be trained to predict algorithm training/prediction time, for instance, using an SVM regressor trained on meta-features [127], itself tuned via genetic algorithms [119]. Yang et al. [187] predict configuration runtime using polynomial regression, based only on the number of instances and features. Hutter et al. [68] provide a general treatise on predicting algorithm runtime in various domains.

Most of these meta-models generate promising configurations, but don't actually tune these configurations to t_{new} themselves. Instead, the predictions can be used to warm-start or guide any other optimization technique, which allows for all kinds of combinations of meta-models and optimization techniques. Indeed, some of the work discussed in Section 2.3.3 can be seen as using a distance-based meta-model to warm-start Bayesian optimization [48, 54] or evolutionary algorithms [59, 128]. In principle, other meta-models could be used here as well.

Instead of learning the relationship between a task's meta-features and configuration performance, one can also build surrogate models predicting the performance of configurations on specific tasks[40]. One can then learn how to combine these per-task predictions to warm-start or guide optimization techniques on a new task t_{new} [45, 114, 160, 185], as discussed in Section 2.2.3. While meta-features could also be used to combine per-task predictions based on task similarity, it is ultimately more effective to gather new observations $P_{i,new}$, since these allow to refine the task similarity estimates with every new observation [47, 185, 85].

2.3.5 Pipeline Synthesis

When creating entire machine learning pipelines [152], the number of configuration options grows dramatically, making it even more important to leverage prior experience. One can control the search space by imposing a fixed structure on the pipeline, fully described by a set of hyperparameters. One can then use the most promising pipelines on similar tasks to warm-start a Bayesian optimization [46, 54].

Other approaches give recommendations for certain pipeline steps [118, 162], and can be leveraged in larger pipeline construction approaches, such as planning [105, 74, 55, 182] or evolutionary techniques [110, 163]. Nguyen et al. [105] construct new pipelines using a beam search focussed on components recommended by a meta-learner, and is itself trained on examples of successful prior pipelines. Bilalli et al. [18] predict which pre-processing techniques are recommended for a given classification algorithm. They build a meta-model per target classification algorithm that, given the t_{new} meta-features, predicts which preprocessing technique should be included in the pipeline. Similarly, Schoenfeld et al. [151] build meta-models predicting when a preprocessing algorithm will improve a particular classifier's accuracy or runtime.

AlphaD3M [38] uses a *self-play* reinforcement learning approach in which the current state is represented by the current pipeline, and actions include the addition, deletion, or replacement of pipeline components. A Monte Carlo Tree Search (MCTS) generates pipelines, which are evaluated to train a recurrent neural network (LSTM) that can predict pipeline performance, in turn producing the action probabilities for the MCTS in the next round. The state description also includes meta-features of the current task, allowing the neural network to learn across tasks.

2.3.6 To Tune or Not to Tune?

To reduce the number of configuration parameters to be optimized, and to save valuable optimization time in time-constrained settings, meta-models have also been proposed to predict whether or not it is worth tuning a given algorithm *given the meta-features of the task at hand* [132] and how much improvement we can expect from tuning a specific algorithm versus the additional time investment [143]. More focused studies on specific learning algorithms yielded meta-models predicting when it is necessary to tune SVMs [96], what are good default hyperparameters for SVMs given the task (including interpretable meta-models) [97], and how to tune decision trees [95].

2.4 Learning from Prior Models

The final type of meta-data we can learn from are prior machine learning models themselves, i.e., their structure and learned model parameters. In short, we want to train a *meta-learner* L that learns how to train a (base-) learner l_{new} for a new task t_{new}, given similar tasks $t_j \in T$ and the corresponding optimized models $l_j \in \mathcal{L}$, where $\mathcal{L}$ is the space of all possible models. The learner l_j is typically defined by its model parameters $W = \{w_k\}$, $k = 1..K$ and/or its configuration $\theta_i \in \Theta$.

2.4.1 Transfer Learning

In *transfer learning* [169], we take models trained on one or more *source* tasks t_j, and use them as starting points for creating a model on a similar *target* task t_{new}. This can be done by forcing the target model to be structurally or otherwise similar to the source model(s). This is a generally applicable idea, and transfer learning approaches have been proposed for kernel methods [41, 42], parametric Bayesian models [139, 122, 8], Bayesian networks [107], clustering [167] and reinforcement learning [62, 36]. Neural networks, however, are exceptionally suitable for transfer learning because both the structure and the model parameters of the source models can be used as a good initialization for the target model, yielding a *pre-trained* model which can then be further fine-tuned using the available training data on t_{new} [168, 11, 13, 24]. In some cases, the source network may need to be modified before transferring it [154]. We will focus on neural networks in the remainder of this section.

Especially large image datasets, such as ImageNet [78], have been shown to yield pre-trained models that transfer exceptionally well to other tasks [37, 153]. However, it has also been shown that this approach doesn't work well when the target task is not so similar [189]. Rather than hoping that a pre-trained model 'accidentally' transfers well to a new problem, we can purposefully imbue meta-learners with an inductive bias (learned from many similar tasks) that allows them to learn new tasks much faster, as we will discuss below.

2.4.2 Meta-Learning in Neural Networks

An early meta-learning approach is to create recurrent neural networks (RNNs) able to modify their own weights [148, 149]. During training, they use their own weights as additional input data and observe their own errors to learn how to modify these weights in response to the new task at hand. The updating of the weights is defined in a parametric form that is differentiable end-to-end and can jointly optimize both the network and training algorithm using gradient descent, yet is also very difficult to train. Later work used reinforcement learning across tasks to adapt the search strategy [150] or the learning rate for gradient descent [31] to the task at hand.

Inspired by the feeling that backpropagation is an unlikely learning mechanism for our own brains, Bengio et al. [12] replace backpropagation with simple biologically-inspired parametric rules (or evolved rules [27]) to update the synaptic weights. The parameters are optimized, e.g. using gradient descent or evolution, across a set of input tasks. Runarsson and Jonsson [141] replaced these parametric rules with a single layer neural network. Santoro et al. [145] instead use a memory-augmented neural network to learn how to store and retrieve 'memories' of prior classification tasks. Hochreiter et al. [65] use LSTMs [66] as a meta-learner to train multi-layer perceptrons.

Andrychowicz et al. [6] also replace the optimizer, e.g. stochastic gradient descent, with an LSTM trained on multiple prior tasks. The loss of the meta-learner (optimizer) is defined as the sum of the losses of the base-learners (optimizees), and optimized using gradient descent. At every step, the meta-learner chooses the weight update estimated to reduce the optimizee's loss the most, based on the learned model weights $\{w_k\}$ of the previous step as well as the current performance gradient. Later work generalizes this approach by training an optimizer on synthetic functions, using gradient descent [28]. This allows meta-learners to optimize optimizees even if these do not have access to gradients.

In parallel, Li and Malik [89] proposed a framework for learning optimization algorithms from a reinforcement learning perspective. It represents any particular optimization algorithm as a policy, and then learns this policy via guided policy search. Follow-up work [90] shows how to leverage this approach to learn optimization algorithms for (shallow) neural networks.

The field of *neural architecture search* includes many other methods that build a model of neural network performance for a specific task, for instance using Bayesian optimization or reinforcement learning. See Chapter 3 for an in-depth discussion. However, most of these methods do not (yet) generalize across tasks and are therefore not discussed here.

2.4.3 Few-Shot Learning

A particularly challenging meta-learning problem is to train an accurate deep learning model using only a few training examples, given prior experience with very similar tasks for which we have large training sets available. This is called *few-shot learning*. Humans have an innate ability to do this, and we wish to build machine learning agents that can do the same [82]. A particular example of this is 'K-shot N-way' classification, in which we are given many examples (e.g., images) of certain classes (e.g., objects), and want to learn a classifier l_{new} able to classify N new classes using only K examples of each.

Using prior experience, we can, for instance, learn a common feature representation of all the tasks, start training l_{new} with a better model parameter initialization W_{init} and acquire an inductive bias that helps guide the optimization of the model parameters, so that l_{new} can be trained much faster than otherwise possible.

Earlier work on *one-shot* learning is largely based on hand-engineered features [44, 43, 50, 10]. With meta-learning, however, we hope to learn a common feature representation for all tasks in an end-to-end fashion.

Vinyals et al. [179] state that, to learn from very little data, one should look to non-parameteric models (such as k-nearest neighbors), which use a memory component rather than learning many model parameters. Their meta-learner is a Matching Network that apply the idea of a memory component in a neural net. It learns a common representation for the labelled examples, and *matches* each new test instance to the memorized examples using cosine similarity. The network is trained on minibatches with only a few examples of a specific task each.

Snell et al. [156] propose Prototypical Networks, which map examples to a p-dimensional vector space such that examples of a given output class are close together. It then calculates a prototype (mean vector) for every class. New test instances are mapped to the same vector space and a distance metric is used to create a softmax over all possible classes. Ren et al. [130] extend this approach to semi-supervised learning.

Ravi and Larochelle [125] use an LSTM-based meta-learner to learn an update rule for training a neural network learner. With every new example, the learner returns the current gradient and loss to the LSTM meta-learner, which then updates the model parameters $\{w_k\}$ of the learner. The meta-learner is trained across all prior tasks.

Model-Agnostic Meta-Learning (MAML) [51], on the other hand, does not try to learn an update rule, but instead learns a model parameter initialization W_{init} that generalizes better to similar tasks. Starting from a random $\{w_k\}$, it iteratively selects a batch of prior tasks, and for each it trains the learner on K examples to compute the gradient and loss (on a test set). It then backpropagates the *meta-gradient* to update the weights $\{w_k\}$ in the direction in which they would have been easier to update. In other words, after each iteration, the weights $\{w_k\}$ become a better W_{init} to start finetuning any of the tasks. Finn and Levine [52] also argue that MAML is able to approximate any learning algorithm when using a sufficiently deep ReLU network and certain losses. They also conclude that the MAML initializations are more resilient to overfitting on small samples, and generalize more widely than meta-learning approaches based on LSTMs.

REPTILE [106] is an approximation of MAML that executes stochastic gradient descent for K iterations on a given task, and then gradually moves the initialization weights in the direction of the weights obtained after the K iterations. The intuition is that every task likely has more than one set of optimal weights $\{w_i^*\}$, and the goal is to find a W_{init} that is close to at least one of those $\{w_i^*\}$ for every task.

Finally, we can also derive a meta-learner from a black-box neural network. Santoro et al [144] propose Memory-Augmented Neural Networks (MANNs), which train a Neural Turing Machine (NTM) [60], a neural network with augmented memory capabilities, as a meta-learner. This meta-learner can then memorize information about previous tasks and leverage that to learn a learner l_{new}. SNAIL [101] is a generic meta-learner architecture consisting of interleaved temporal convolution and causal attention layers. The convolutional networks learn a common feature vector for the training instances (images) to aggregate information from past experiences. The causal attention layers learn which pieces of information to pick out from the gathered experience to generalize to new tasks.

Overall, the intersection of deep learning and meta-learning proves to be particular fertile ground for groundbreaking new ideas, and we expect this field to become more important over time.

2.4.4 Beyond Supervised Learning

Meta-learning is certainly not limited to (semi-)supervised tasks, and has been successfully applied to solve tasks as varied as reinforcement learning, active learning, density estimation and item recommendation. The base-learner may be unsupervised while the meta-learner is supervised, but other combinations are certainly possible as well.

Duan et al. [39] propose an end-to-end reinforcement learning (RL) approach consisting of a task-specific *fast* RL algorithm which is guided by a general-purpose *slow* meta-RL algorithm. The tasks are interrelated Markov Decision Processes (MDPs). The meta-RL algorithm is modeled as an RNN, which receives the observations, actions, rewards and termination flags. The activations of the RNN store the state of the fast RL learner, and the RNN's weights are learned by observing the performance of fast learners across tasks.

In parallel, Wang et al. [180] also proposed to use a deep RL algorithm to train an RNN, receiving the actions and rewards of the previous interval in order to learn a base-level RL algorithm for specific tasks. Rather than using relatively unstructured tasks such as random MDPs, they focus on structured task distributions (e.g., dependent bandits) in which the meta-RL algorithm can exploit the inherent task structure.

Pang et al. [112] offer a meta-learning approach to active learning (AL). The base-learner can be any binary classifier, and the meta-learner is a deep RL network consisting of a deep neural network that learns a representation of the AL problem across tasks, and a policy network that learns the optimal policy, parameterized as weights in the network. The meta-learner receives the current state (the unlabeled point set and base classifier state) and reward (the performance of the base classifier), and emits a query probability, i.e. which points in the unlabeled set to query next.

Reed et al. [126] propose a few-shot approach for density estimation (DE). The goal is to learn a probability distribution over a small number of images of a certain concept (e.g., a handwritten letter) that can be used to generate images of that concept, or compute the probability that an image shows that concept. The approach uses autoregressive image models which factorize the joint distribution into per-pixel factors, usually conditioned on (many) examples of the target concept. Instead, a MAML-based few-shot learner is used, trained on examples of many other (similar) concepts.

Finally, Vartak et al. [176] address the cold-start problem in matrix factorization. They propose a deep neural network architecture that learns a (base) neural network whose biases are adjusted based on task information. While the structure and weights of the neural net recommenders remain fixed, the meta-learner learns how to adjust the biases based on each user's item history.

All these recent new developments illustrate that it is often fruitful to look at problems through a *meta-learning lens* and find new, data-driven approaches to replace hand-engineered base-learners.

2.5 Conclusion

Meta-learning opportunities present themselves in many different ways, and can be embraced using a wide spectrum of learning techniques. Every time we try to learn a certain task, whether successful or not, we gain useful experience that we can leverage to learn new tasks. We should never have to start entirely from scratch. Instead, we should systematically collect our 'learning exhaust' and learn from it to build AutoML systems that continuously improve over time, helping us tackle new learning problems ever more efficiently. The more new tasks we encounter, and the more similar those new tasks are, the more we can tap into prior experience, to the point that most of the required learning has already been done beforehand. The ability of computer systems to store virtually infinite amounts of prior learning experiences (in the form of meta-data) opens up a wide range of opportunities to use that experience in completely new ways, and we are only starting to learn how to learn from prior experience effectively. Yet, this is a worthy goal: learning how to learn any task empowers us far beyond knowing how to learn specific tasks.

Acknowledgments

The author would like to thank Pavel Brazdil, Matthias Feurer, Frank Hutter, Raghu Rajan, and Jan van Rijn for many invaluable discussions and feedback on the manuscript.

Bibliography

[1] Abdulrahman, S., Brazdil, P., van Rijn, J., Vanschoren, J.: Speeding up Algorithm Selection using Average Ranking and Active Testing by Introducing Runtime. Machine Learning 107, 79–108 (2018)

[2] Afif, I.N.: Warm-Starting Deep Learning Model Construction using Meta-Learning. Master's thesis, TU Eindhoven (2018)

[3] Agresti, A.: Categorical Data Analysis. Wiley Interscience (2002)

[4] Ali, S., Smith-Miles, K.A.: Metalearning approach to automatic kernel selection for support vector machines. Neurocomput. 70(1), 173–186 (2006)

[5] Ali, S., Smith-Miles, K.A.: On learning algorithm selection for classification. Applied Soft Computing 6(2), 119 – 138 (2006)

[6] Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., De Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Advances in Neural Information Processing Systems. pp. 3981–3989 (2016)

[7] Arinze, B.: Selecting appropriate forecasting models using rule induction. Omega 22(6), 647–658 (1994)

[8] Bakker, B., Heskes, T.: Task Clustering and Gating for Bayesian Multitask Learning. Journal of Machine Learning Research 4, 83–999 (2003)

[9] Bardenet, R., Brendel, M., Kégl, B., Sebag, M.: Collaborative hyperparameter tuning. In: Proceedings of ICML 2013. pp. 199–207 (2013)

[10] Bart, E., Ullman, S.: Cross-generalization: Learning novel classes from a single example by feature replacement. In: CVPR. pp. 672–679 (2005)

[11] Baxter, J.: Learning Internal Representations. In: Advances in Neural Information Processing Systems, NIPS (1996)

[12] Bengio, S., Bengio, Y., Cloutier, J.: On the search for new learning rules for anns. Neural Processing Letters 2(4), 26–30 (1995)

[13] Bengio, Y.: Deep learning of representations for unsupervised and transfer learning. In: ICML Unsupervised and Transfer Learning. pp. 17–36 (2012)

[14] Bensusan, H., Kalousis, A.: Estimating the predictive accuracy of a classifier. Lecture Notes in Computer Science 2167, 25–36 (2001)

[15] Bensusan, H., Giraud-Carrier, C.: Discovering task neighbourhoods through landmark learning performances. In: PKDD. pp. 325–330 (2000)

[16] Bensusan, H., Giraud-Carrier, C., Kennedy, C.: A higher-order approach to meta-learning. In: ILP. pp. 33 – 42 (2000)

[17] Bilalli, B., Abelló, A., Aluja-Banet, T.: On the predictive power of meta-features in OpenML. International Journal of Applied Mathematics and Computer Science 27(4), 697 – 712 (2017)

[18] Bilalli, B., Abelló, A., Aluja-Banet, T., Wrembel, R.: Intelligent assistance for data pre-processing. Computer Standards & Interf. 57, 101 – 109 (2018)

[19] Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., Vanschoren, J.: ASLib: A benchmark library for algorithm selection. Artificial Intelligence 237, 41–58 (2016)

[20] Bishop, C.M.: Pattern recognition and machine learning. Springer (2006)

[21] Brazdil, P., Soares, C., da Costa, J.P.: Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. Machine Learning 50(3), 251–277 (2003)

[22] Brazdil, P., Giraud-Carrier, C., Soares, C., Vilalta, R.: Metalearning: Applications to Data Mining. Springer-Verlag Berlin Heidelberg (2009)

[23] Brazdil, P.B., Soares, C., Da Coasta, J.P.: Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. Machine Learning 50(3), 251–277 (2003)

[24] Caruana, R.: Learning many related tasks at the same time with backpropagation. Neural Information Processing Systems pp. 657–664 (1995)

[25] Caruana, R.: Multitask Learning. Machine Learning 28(1), 41–75 (1997)

[26] Castiello, C., Castellano, G., Fanelli, A.M.: Meta-data: Characterization of input features for meta-learning. In: 2nd International Conference on Modeling Decisions for Artificial Intelligence (MDAI). pp. 457 – 468 (2005)

[27] Chalmers, D.J.: The evolution of learning: An experiment in genetic connectionism. In: Connectionist Models, pp. 81–90. Elsevier (1991)

[28] Chen, Y., Hoffman, M.W., Colmenarejo, S.G., Denil, M., Lillicrap, T.P., Botvinick, M., de Freitas, N.: Learning to learn without gradient descent by gradient descent. arXiv preprint arXiv:1611.03824 (2016)

[29] Cheng, W., Hühn, J., Hüllermeier, E.: Decision tree and instance-based learning for label ranking. In: ICML. pp. 161–168 (2009)

[30] Cook, W.D., Kress, M., Seiford, L.W.: A general framework for distance-based consensus in ordinal ranking models. European Journal of Operational Research 96(2), 392–397 (1996)

[31] Daniel, C., Taylor, J., Nowozin, S.: Learning step size controllers for robust neural network training. In: AAAI. pp. 1519–1525 (2016)

[32] Davis, C., Giraud-Carrier, C.: Annotative experts for hyperparameter selection. In: AutoML Workshop at ICML 2018 (2018)

[33] De Sa, A., Pinto, W., Oliveira, L.O., Pappa, G.: RECIPE: A grammar-based framework for automatically evolving classification pipelines. In: European Conference on Genetic Programming. pp. 246–261 (2017)

[34] Demšar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research 7, 1–30 (2006)

[35] Dietterich, T.: Ensemble methods in machine learning. In: International workshop on multiple classifier systems. pp. 1–15 (2000)

[36] Dietterich, T., Busquets, D., Lopez de Mantaras, R., Sierra, C.: Action Refinement in Reinforcement Learning by Probability Smoothing. In: 19th International Conference on Machine Learning. pp. 107–114 (2002)

[37] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: Decaf: A deep convolutional activation feature for generic visual recognition. In: ICML. pp. 647–655 (2014)

[38] Drori, I., Krishnamurthy, Y., Rampin, R., de Paula Lourenco, R., Ono, J.P., Cho, K., Silva, C., Freire, J.: AlphaD3M: Machine learning pipeline synthesis. In: AutoML Workshop at ICML (2018)

[39] Duan, Y., Schulman, J., Chen, X., Bartlett, P.L., Sutskever, I., Abbeel, P.: RL^2: Fast reinforcement learning via slow reinforcement learning. arXiv preprint arXiv:1611.02779 (2016)

[40] Eggensperger, K., Lindauer, M., Hoos, H., Hutter, F., Leyton-Brown, K.: Efficient Benchmarking of Algorithm Configuration Procedures via Model-Based Surrogates . Machine Learning 107, 15–41 (2018)

[41] Evgeniou, T., Micchelli, C., Pontil, M.: Learning Multiple Tasks with Kernel Methods. Journal of Machine Learning Research 6, 615–637 (2005)

[42] Evgeniou, T., Pontil, M.: Regularized multi-task learning. In: Tenth Conference on Knowledge Discovery and Data Mining (2004)

[43] Fei-Fei, L.: Knowledge transfer in learning to recognize visual objects classes. In: Intern. Conf. on Development and Learning. p. Art. 51 (2006)

[44] Fei-Fei, L., Fergus, R., Perona, P.: One-shot learning of object categories. Pattern analysis and machine intelligence 28(4), 594–611 (2006)

[45] Feurer, M., Letham, B., Bakshy, E.: Scalable meta-learning for Bayesian optimization. arXiv 1802.02219 (2018)

[46] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems 28. pp. 2944–2952 (2015)

[47] Feurer, M., Letham, B., Bakshy, E.: Scalable meta-learning for bayesian optimization using ranking-weighted gaussian process ensembles. In: AutoML Workshop at ICML 2018 (2018)

[48] Feurer, M., Springenberg, J.T., Hutter, F.: Using meta-learning to initialize Bayesian optimization of hypxerparameters. In: International Conference on Meta-learning and Algorithm Selection. pp. 3 – 10 (2014)

[49] Filchenkov, A., Pendryak, A.: Dataset metafeature description for recommending feature selection. In: ISMW FRUCT. pp. 11–18 (2015)

[50] Fink, M.: Object classification from a single example utilizing class relevance metrics. In: Neural information processing syst. pp. 449–456 (2005)

[51] Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: ICML. pp. 1126–1135 (2017)

[52] Finn, C., Levine, S.: Meta-learning and universality. arXiv 1710.11622 (2017)

[53] Fürnkranz, J., Petrak, J.: An evaluation of landmarking variants. ECML/PKDD 2001 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning pp. 57–68 (2001)

[54] Fusi, N., Sheth, R., Elibol, H.M.: Probabilistic matrix factorization for automated machine learning. arXiv preprint arXiv:1705.05355 (2017)

[55] Gil, Y., Yao, K.T., Ratnakar, V., Garijo, D., Ver Steeg, G., Szekely, P., Brekelmans, R., Kejriwal, M., Luo, F., Huang, I.H.: P4ML: A phased performance-based pipeline planner for automated machine learning. In: AutoML Workshop at ICML 2018 (2018)

[56] Giraud-Carrier, C.: Metalearning-a tutorial. In: Tutorial at the International Conference on Machine Learning and Applications. pp. 1–45 (2008)

[57] Giraud-Carrier, C., Provost, F.: Toward a justification of meta-learning: Is the no free lunch theorem a show-stopper. In: Proceedings of the ICML-2005 Workshop on Meta-learning. pp. 12–19 (2005)

[58] Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., Sculley, D.: Google vizier: A service for black-box optimization. In: ICDM. pp. 1487–1495 (2017)

[59] Gomes, T.A., Prudêncio, R.B., Soares, C., Rossi, A.L., Carvalho, A.: Combining meta-learning and search techniques to select parameters for support vector machines. Neurocomputing 75(1), 3–13 (2012)

[60] Graves, A., Wayne, G., Danihelka, I.: Neural turing machines. arXiv preprint arXiv:1410.5401 (2014)

[61] Guerra, S.B., Prudêncio, R.B., Ludermir, T.B.: Predicting the performance of learning algorithms using support vector machines as meta-regressors. In: ICANN. pp. 523–532 (2008)

[62] Hengst, B.: Discovering Hierarchy in Reinforcement Learning with HEXQ. In: International Conference on Machine Learning. pp. 243–250 (2002)

[63] Hilario, M., Kalousis, A.: Fusion of meta-knowledge and meta-data for case-based model selection. Lecture Notes in Computer Science 2168, 180–191 (2001)

[64] Ho, T.K., Basu, M.: Complexity measures of supervised classification problems. Pattern Analysis and Machine Intellig. 24(3), 289–300 (2002)

[65] Hochreiter, S., Younger, A., Conwell, P.: Learning to learn using gradient descent. In: Lecture Notes on Computer Science, 2130. pp. 87–94 (2001)

[66] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)

[67] Hutter, F., Hoos, H., Leyton-Brown, K.: An Efficient Approach for Assessing Hyperparameter Importance. In: Proceedings of ICML (2014)

[68] Hutter, F., Xu, L., Hoos, H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. Artificial Intelligence 206, 79–111 (2014)

[69] Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization 13(4), 455–492 (1998)

[70] Kalousis, A.: Algorithm Selection via Meta-Learning. Ph.D. thesis, University of Geneva, Department of Computer Science (2002)

[71] Kalousis, A., Hilario, M.: Representational issues in meta-learning. Proceedings of ICML 2003 pp. 313–320 (2003)

[72] Kalousis, A., Hilario, M.: Model selection via meta-learning: a comparative study. Intl Journ. on Artificial Intelligence Tools 10(4), 525–554 (2001)

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

[74] Kietz, J.U., Serban, F., Bernstein, A., Fischer, S.: Designing KDD-workflows via HTN-planning for intelligent discovery assistance. In: 5th Planning to Learn Workshop at ECAI 2012 (2012)

[75] Kim, J., Kim, S., Choi, S.: Learning to warm-start Bayesian hyperparameter optimization. arXiv preprint arXiv:1710.06219 (2017)

[76] Köpf, C., Iglezakis, I.: Combination of task description strategies and case base properties for meta-learning. ECML/PKDD Workshop on Integration and Collaboration Aspects of Data Mining pp. 65–76 (2002)

[77] Köpf, C., Taylor, C., Keller, J.: Meta-analysis: From data characterization for meta-learning to meta-regression. In: PKDD Workshop on Data Mining, Decision Support, Meta-Learning and ILP. pp. 15–26 (2000)

[78] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)

[79] Kuba, P., Brazdil, P., Soares, C., Woznica, A.: Exploiting sampling and meta-learning for parameter setting support vector machines. In: Proceedings of IBERAMIA 2002. pp. 217–225 (2002)

[80] Kullback, S., Leibler, R.A.: On information and sufficiency. The annals of mathematical statistics 22(1), 79–86 (1951)

[81] Lacoste, A., Marchand, M., Laviolette, F., Larochelle, H.: Agnostic Bayesian learning of ensembles. In: ICML. pp. 611–619 (2014)

[82] Lake, B.M., Ullman, T.D., Tenenbaum, J.B., Gershman, S.J.: Building machines that learn and think like people. Beh. and Brain Sc. 40 (2017)

[83] Leite, R., Brazdil, P.: Predicting relative performance of classifiers from samples. Proceedings of ICML pp. 497–504 (2005)

[84] Leite, R., Brazdil, P.: An iterative process for building learning curves and predicting relative performance of classifiers. Lecture Notes in Computer Science 4874, 87–98 (2007)

[85] Leite, R., Brazdil, P., Vanschoren, J.: Selecting Classification Algorithms with Active Testing. Lecture Notes in Artif. Intel. 10934, 117–131 (2012)

[86] Leite, R., Brazdil, P.: Active testing strategy to predict the best classification algorithm via sampling and metalearning. In: ECAI 2010. pp. 309–314 (2010)

[87] Lemke, C., Budka, M., Gabrys, B.: Metalearning: a survey of trends and technologies. Artificial intelligence review 44(1), 117–130 (2015)

[88] Ler, D., Koprinska, I., Chawla, S.: Utilizing regression-based landmarkers within a meta-learning framework for algorithm selection. Technical Report 569. University of Sydney pp. 44–51 (2005)

[89] Li, K., Malik, J.: Learning to optimize. arXiv preprint arXiv:1606.01885 (2016)

[90] Li, K., Malik, J.: Learning to optimize neural nets. arXiv preprint arXiv:1703.00441 (2017)

[91] Lin, S.: Rank aggregation methods. WIREs Computational Statistics 2, 555–570 (2010)

[92] Lindner, G., Studer, R.: AST: Support for algorithm selection with a CBR approach. In: ICML Workshop on Recent Advances in Meta-Learning and Future Work. pp. 38–47. J. Stefan Institute (1999)

[93] Lorena, A.C., Maciel, A.I., de Miranda, P.B.C., Costa, I.G., Prudêncio, R.B.C.: Data complexity meta-features for regression problems. Machine Learning 107(1), 209–246 (2018)

[94] Luo, G.: A review of automatic selection methods for machine learning algorithms and hyper-parameter values. Network Modeling Analysis in Health Informatics and Bioinformatics 5(1), 18 (2016)

[95] Mantovani, R.G., Horváth, T., Cerri, R., Vanschoren, J., de Carvalho, A.C.: Hyper-parameter tuning of a decision tree induction algorithm. In: Brazilian Conference on Intelligent Systems. pp. 37–42 (2016)

[96] Mantovani, R.G., Rossi, A.L., Vanschoren, J., Bischl, B., Carvalho, A.C.: To tune or not to tune: recommending when to adjust SVM hyper-parameters via meta-learning. In: Proceedings of IJCNN. pp. 1–8 (2015)

[97] Mantovani, R.G., Rossi, A.L., Vanschoren, J., Carvalho, A.C.: Meta-learning recommendation of default hyper-parameter values for SVMs in classifications tasks. In: ECML PKDD Workshop on Meta-Learning and Algorithm Selection (2015)

[98] Mantovani, R.: Use of meta-learning for hyperparameter tuning of classification problems. Ph.D. thesis, University of Sao Carlos, Brazil (2018)

[99] Michie, D., Spiegelhalter, D.J., Taylor, C.C., Campbell, J.: Machine Learning, Neural and Statistical Classification. Ellis Horwood (1994)

[100] Miranda, P., Prudêncio, R.: Active testing for SVM parameter selection. In: Proceedings of IJCNN. pp. 1–8 (2013)

[101] Mishra, N., Rohaninejad, M., Chen, X., Abbeel, P.: A simple neural attentive meta-learner. In: Proceedings of ICLR (2018)

[102] Misir, M., Sebag, M.: Algorithm Selection as a Collaborative Filtering Problem. Research report, INRIA (2013)

[103] Mısır, M., Sebag, M.: Alors: An algorithm recommender system. Artificial Intelligence 244, 291–314 (2017)

[104] Nadaraya, E.A.: On estimating regression. Theory of Probability & Its Applications 9(1), 141–142 (1964)

[105] Nguyen, P., Hilario, M., Kalousis, A.: Using meta-mining to support data mining workflow planning and optimization. Journal of Artificial Intelligence Research 51, 605–644 (2014)

[106] Nichol, A., Achiam, J., Schulman, J.: On first-order meta-learning algorithms. arXiv 1803.02999v2 (2018)

[107] Niculescu-Mizil, A., Caruana, R.: Learning the Structure of Related Tasks. In: Proceedings of NIPS Workshop on Inductive Transfer (2005)

[108] Nisioti, E., Chatzidimitriou, K., Symeonidis, A.: Predicting hyperparameters from meta-features in binary classification problems. In: AutoML Workshop at ICML (2018)

[109] Olier, I., Sadawi, N., Bickerton, G., Vanschoren, J., Grosan, C., Soldatova, L., King, R.: Meta-QSAR: learning how to learn QSARs. Machine Learning 107, 285–311 (2018)

[110] Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of GECCO. pp. 485–492 (2016)

[111] Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on knowledge and data engineering 22(10), 1345–1359 (2010)

[112] Pang, K., Dong, M., Wu, Y., Hospedales, T.: Meta-learning transferable active learning policies by deep reinforcement learning. In: AutoML Workshop at ICML (2018)

[113] Peng, Y., Flach, P., Soares, C., Brazdil, P.: Improved dataset characterisation for meta-learning. Lecture Notes in Com. Sc. 2534, 141–152 (2002)

[114] Perrone, V., Jenatton, R., Seeger, M., Archambeau, C.: Multiple adaptive Bayesian linear regression for scalable Bayesian optimization with warm start. arXiv preprint arXiv:1712.02902 (2017)

[115] Pfahringer, B., Bensusan, H., Giraud-Carrier, C.G.: Meta-learning by landmarking various learning algorithms. In: 17th International Conference on Machine Learning (ICML). pp. 743 – 750 (2000)

[116] Pinto, F., Cerqueira, V., Soares, C., Mendes-Moreira, J.: autoBagging: Learning to rank bagging workflows with metalearning. arXiv 1706.09367 (2017)

[117] Pinto, F., Soares, C., Mendes-Moreira, J.: Towards automatic generation of metafeatures. In: Proceedings of PAKDD. pp. 215–226 (2016)

[118] Post, M.J., van der Putten, P., van Rijn, J.N.: Does Feature Selection Improve Classification? A Large Scale Experiment in OpenML. In: Advances in Intelligent Data Analysis XV. pp. 158–170 (2016)

[119] Priya, R., De Souza, B.F., Rossi, A., Carvalho, A.: Using genetic algorithms to improve prediction of execution times of ML tasks. In: Lecture Notes in Comp. Science. vol. 7208, pp. 196–207 (2012)

[120] Probst, P., Bischl, B., Boulesteix, A.L.: Tunability: Importance of hyperparameters of machine learning algorithms. ArXiv 1802.09596 (2018)

[121] Prudêncio, R., Ludermir, T.: Meta-learning approaches to selecting time series models. Neurocomputing 61, 121–137 (2004)

[122] Raina, R., Ng, A.Y., Koller, D.: Transfer Learning by Constructing Informative Priors. In: Proceedings of ICML (2006)

[123] Ramachandran, A., Gupta, S., Rana, S., Venkatesh, S.: Information-theoretic transfer learning framework for Bayesian optimisation. In: Proceedings of ECMLPKDD (2018)

[124] Ramachandran, A., Gupta, S., Rana, S., Venkatesh, S.: Selecting optimal source for transfer learning in Bayesian optimisation. In: Proceedings of PRICAI. pp. 42–56 (2018)

[125] Ravi, S., Larochelle, H.: Optimization as a model for few-shot learning. In: Proceedings of ICLR (2017)

[126] Reed, S., Chen, Y., Paine, T., Oord, A.v.d., Eslami, S., Rezende, D., Vinyals, O., de Freitas, N.: Few-shot autoregressive density estimation: Towards learning to learn distributions. arXiv preprint arXiv:1710.10304 (2017)

[127] Reif, M., Shafait, F., Dengel, A.: Prediction of classifier training time including parameter optimization. In: Proc. of GfKI. pp. 260 – 271 (2011)

[128] Reif, M., Shafait, F., Dengel, A.: Meta-learning for evolutionary parameter optimization of classifiers. Machine learning 87(3), 357–380 (2012)

[129] Reif, M., Shafait, F., Goldstein, M., Breuel, T., Dengel, A.: Automatic classifier selection for non-experts. Pattern Analysis and Applications 17(1), 83 – 96 (2014)

[130] Ren, M., Triantafillou, E., Ravi, S., Snell, J., Swersky, K., Tenenbaum, J.B., Larochelle, H., Zemel, R.S.: Meta-learning for semi-supervised few-shot classification. arXiv 1803.00676 (2018)

[131] Rendle, S.: Factorization machines. In: ICDM 2015. pp. 995–1000 (2010)

[132] Ridd, P., Giraud-Carrier, C.: Using metalearning to predict when parameter optimization is likely to improve classification accuracy. In: ECAI Workshop on Meta-learning and Algorithm Selection. pp. 18–23 (2014)

[133] van Rijn, J., Abdulrahman, S., Brazdil, P., Vanschoren, J.: Fast Algorithm Selection Using Learning Curves. In: Proceedings of IDA (2015)

[134] van Rijn, J., Holmes, G., Pfahringer, B., Vanschoren, J.: The Online Performance Estimation Framework. Heterogeneous Ensemble Learning for Data Streams. Machine Learning 107, 149–176 (2018)

[135] van Rijn, J.N., Hutter, F.: Hyperparameter importance across datasets. In: Proceedings of KDD. pp. 2367–2376 (2018)

[136] van Rijn, J.N., Holmes, G., Pfahringer, B., Vanschoren, J.: Algorithm selection on data streams. In: Discovery Science. pp. 325–336 (2014)

[137] Rivolli, A., Garcia, L., Soares, C., Vanschoren, J., de Carvalho, A.: Towards reproducible empirical research in meta-learning. arXiv preprint 1808.10406 (2018)

[138] Robbins, H.: Some aspects of the sequential design of experiments. In: Herbert Robbins Selected Papers, pp. 169–177. Springer (1985)

[139] Rosenstein, M.T., Marx, Z., Kaelbling, L.P.: To Transfer or Not To Transfer. In: NIPS Workshop on transfer learning (2005)

[140] Rousseeuw, P.J., Hubert, M.: Robust statistics for outlier detection. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1(1), 73 – 79 (2011)

[141] Runarsson, T.P., Jonsson, M.T.: Evolution and design of distributed learning rules. In: IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. pp. 59–63 (2000)

[142] Salama, M.A., Hassanien, A.E., Revett, K.: Employment of neural network and rough set in meta-learning. Memetic Comp. 5(3), 165–177 (2013)

[143] Sanders, S., Giraud-Carrier, C.: Informing the use of hyperparameter optimization through metalearning. In: Proc. ICDM. pp. 1051–1056 (2017)

[144] Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., Lillicrap, T.: Meta-learning with memory-augmented neural networks. In: International conference on machine learning. pp. 1842–1850 (2016)

[145] Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., Lillicrap, T.: One-shot learning with memory-augmented neural networks. arXiv preprint arXiv:1605.06065 (2016)

[146] dos Santos, P., Ludermir, T., Prudêncio, R.: Selection of time series forecasting models based on performance information. 4th International Conference on Hybrid Intelligent Systems pp. 366–371 (2004)

[147] Schilling, N., Wistuba, M., Drumond, L., Schmidt-Thieme, L.: Hyperparameter optimization with factorized multilayer perceptrons. In: Proceedings of ECML PKDD. pp. 87–103 (2015)

[148] Schmidhuber, J.: Learning to control fast-weight memories: An alternative to dynamic recurrent networks. Neural Comp. 4(1), 131–139 (1992)

[149] Schmidhuber, J.: A neural network that embeds its own meta-levels. In: Proceedings of ICNN. pp. 407–412 (1993)

[150] Schmidhuber, J., Zhao, J., Wiering, M.: Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. Machine Learning 28(1), 105–130 (1997)

[151] Schoenfeld, B., Giraud-Carrier, C., Poggeman, M., Christensen, J., Seppi, K.: Feature selection for high-dimensional data: A fast correlation-based filter solution. In: AutoML Workshop at ICML (2018)

[152] Serban, F., Vanschoren, J., Kietz, J., Bernstein, A.: A survey of intelligent assistants for data analysis. ACM Computing Surveys 45(3), Art.31 (2013)

[153] Sharif Razavian, A., Azizpour, H., Sullivan, J., Carlsson, S.: Cnn features off-the-shelf: an astounding baseline for recognition. In: Proceedings of CVPR 2014. pp. 806–813 (2014)

[154] Sharkey, N.E., Sharkey, A.J.C.: Adaptive Generalization. Artificial Intelligence Review 7, 313–328 (1993)

[155] Smith-Miles, K.A.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys 41(1), 1 – 25 (2009)

[156] Snell, J., Swersky, K., Zemel, R.: Prototypical networks for few-shot learning. In: Neural Information Processing Systems. pp. 4077–4087 (2017)

[157] Soares, C., Brazdil, P., Kuba, P.: A meta-learning method to select the kernel width in support vector regression. Mach. Learn. 54, 195–209 (2004)

[158] Soares, C., Ludermir, T., Carvalho, F.D.: An analysis of meta-learning techniques for ranking clustering algorithms applied to artificial data. Lecture Notes in Computer Science 5768, 131–140 (2009)

[159] Soares, C., Petrak, J., Brazdil, P.: Sampling based relative landmarks: Systematically testdriving algorithms before choosing. Lecture Notes in Computer Science 3201, 250–261 (2001)

[160] Springenberg, J., Klein, A., Falkner, S., Hutter, F.: Bayesian optimization with robust Bayesian neural networks. In: Advances in Neural Information Processing Systems (2016)

[161] Stern, D.H., Samulowitz, H., Herbrich, R., Graepel, T., Pulina, L., Tacchella, A.: Collaborative expert portfolio management. In: Proceedings of AAAI. pp. 179–184 (2010)

[162] Strang, B., van der Putten, P., van Rijn, J.N., Hutter, F.: Don't Rule Out Simple Models Prematurely. In: Adv. in Intelligent Data Analysis (2018)

[163] Sun, Q., Pfahringer, B., Mayo, M.: Towards a Framework for Designing Full Model Selection and Optimization Systems. In: International Workshop on Multiple Classifier Systems. pp. 259–270 (2013)

[164] Sun, Q., Pfahringer, B.: Pairwise meta-rules for better meta-learning-based algorithm ranking. Machine Learning 93(1), 141–161 (2013)

[165] Swersky, K., Snoek, J., Adams, R.P.: Multi-task Bayesian optimization. In: Adv. in neural information processing systems. pp. 2004–2012 (2013)

[166] Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika 25(3/4), 285–294 (1933)

[167] Thrun, S.: Lifelong Learning Algorithms. In: Learning to Learn, chap. 8, pp. 181–209. Kluwer Academic Publishers, MA (1998)

[168] Thrun, S., Mitchell, T.: Learning One More Thing. In: Proceedings of IJCAI. pp. 1217–1223 (1995)

[169] Thrun, S., Pratt, L.: Learning to Learn: Introduction and Overview. In: Learning to Learn, pp. 3–17. Kluwer (1998)

[170] Todorovski, L., Blockeel, H., Džeroski, S.: Ranking with predictive clustering trees. Lecture Notes in Artificial Intelligence 2430, 444–455 (2002)

[171] Todorovski, L., Brazdil, P., Soares, C.: Report on the experiments with feature selection in meta-level learning. PKDD 2000 Workshop on Data mining, Decision support, Meta-learning and ILP pp. 27–39 (2000)

[172] Todorovski, L., Dzeroski, S.: Experiments in meta-level learning with ILP. Lecture Notes in Computer Science 1704, 98–106 (1999)

[173] Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: networked science in machine learning. ACM SIGKDD Explorations Newsletter 15(2), 49–60 (2014)

[174] Vanschoren, J.: Understanding Machine Learning Performance with Experiment Databases. Ph.D. thesis, Leuven Univeristy (2010)

[175] Vanschoren, J., Blockeel, H., Pfahringer, B., Holmes, G.: Experiment databases. Machine Learning 87(2), 127–158 (2012)

[176] Vartak, M., Thiagarajan, A., Miranda, C., Bratman, J., Larochelle, H.: A meta-learning perspective on cold-start recommendations for items. In: Advances in Neural Information Processing Systems. pp. 6904–6914 (2017)

[177] Vilalta, R.: Understanding accuracy performance through concept characterization and algorithm analysis. ICML Workshop on Recent Advances in Meta-Learning and Future Work (1999)

[178] Vilalta, R., Drissi, Y.: A characterization of difficult problems in classification. Proceedings of ICMLA (2002)

[179] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shot learning. In: Advances in Neural Information Processing Systems. pp. 3630–3638 (2016)

[180] Wang, J.X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J.Z., Munos, R., Blundell, C., Kumaran, D., Botvinick, M.: Learning to reinforcement learn. arXiv preprint arXiv:1611.05763 (2016)

[181] Weerts, H., Meuller, M., Vanschoren, J.: Importance of tuning hyperparameters of machine learning algorithms. Tech. rep., TU Eindhoven (2018)

[182] Wever, M., Mohr, F., Hüllermeier, E.: Ml-plan for unlimited-length machine learning pipelines. In: AutoML Workshop at ICML 2018 (2018)

[183] Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Hyperparameter search space pruning, a new component for sequential model-based hyperparameter optimization. In: ECML PKDD 2015. pp. 104–119 (2015)

[184] Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Learning hyperparameter optimization initializations. In: 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA). pp. 1–10 (2015)

[185] Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Scalable Gaussian process-based transfer surrogates for hyperparameter optimization. Machine Learning 107(1), 43–78 (2018)

[186] Wolpert, D., Macready, W.: No free lunch theorems for search. Tech. Rep. SFI-TR-95-02-010, The Santa Fe Institute (1996)

[187] Yang, C., Akimoto, Y., Kim, D., Udell, M.: Oboe: Collaborative filtering for automl initialization. arXiv preprint arXiv:1808.03233 (2018)

[188] Yogatama, D., Mann, G.: Efficient transfer learning method for automatic hyperparameter tuning. In: AI and Statistics. pp. 1077–1085 (2014)

[189] Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: Advances in neural information processing systems. pp. 3320–3328 (2014)