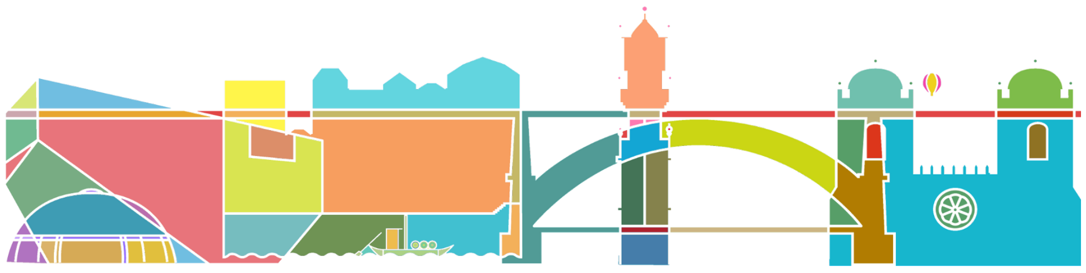


META-LEARNING AND ALGORITHM SELECTION WORKSHOP - ECMLPKDD 2015

MetaSel 2015

September 7, 2015

Porto, Portugal

Edited by

Joaquin Vanschoren, Pavel Brazdil, Christophe Giraud-Carrier and Lars Kotthoff

Preface

Algorithm Selection and configuration are increasingly relevant today. Researchers and practitioners from all branches of science and technology face a large choice of parameterized machine learning algorithms, with little guidance as to which techniques to use. Moreover, data mining challenges frequently remind us that algorithm selection and configuration are crucial in order to achieve the best performance, and drive industrial applications.

Meta-learning leverages knowledge of past algorithm applications to select the best techniques for future applications, and offers effective techniques that are superior to humans both in terms of the end result and especially in the time required to achieve it. In this workshop we will discuss different ways of exploiting meta-learning techniques to identify the potentially best algorithm(s) for a new task, based on meta-level information and prior experiments. We also discuss the prerequisites for effective meta-learning systems such as recent infrastructure such as OpenML.org.

Many problems of today require that solutions be elaborated in the form of complex systems or workflows which include many different processes or operations. Constructing such complex systems or workflows requires extensive expertise, and could be greatly facilitated by leveraging planning, meta-learning and intelligent system design. This task is inherently interdisciplinary, as it builds on expertise in various areas of AI.

This ECMLPKDD 2015 workshop will provide a platform for researchers and research students interested to exchange their knowledge about:

- Problems and solutions of algorithm selection and algorithm configuration
- How to use software and platforms to select algorithms in practice
- How to provide advice to end users about which algorithms to select in diverse domains, including optimization, SAT etc. and incorporate this knowledge in new platforms.

These proceedings include 15 contributions discussing the nature of algorithm selection which arises in many diverse domains, such as machine learning, data mining, optimization and satisfiability solving, among many others. We thank everybody for their sincere interest and their contributions, our programme committee for reviewing all submissions, and especially our invited speakers:

- **Bernd Bischl**
  Applying Model-Based Optimization to Hyperparameter Optimization in Machine Learning
- **Bernhard Pfahringer**
  On a Few Recent Developments in Meta-Learning for Algorithm Ranking and Selection

We hope you will find it an interesting and inspiring workshop, leading to fruitful new collaborations.

Porto, September 2015

Joaquin Vanschoren
Pavel Brazdil
Christophe Giraud-Carrier
Lars Kotthoff

Main areas covered by the workshop

Of particular interest are methods and proposals that address the following issues:

– Algorithm / Model Selection and Configuration
– Meta-learning and exploitation of meta-knowledge
– Experimentation and evaluation of learning processes
– Hyper-parameter optimization
– Planning to learn and to construct workflows
– Applications of workflow planning
– Exploitation of ontologies of tasks and methods
– Exploitation of benchmarks and experimentation
– Representation of learning goals and states in learning
– Control and coordination of learning processes
– Meta-reasoning
– Layered learning
– Multi-task and transfer learning
– Learning to learn
– Intelligent design
– Performance modeling and process mining

Program Committee

Table of Contents

Applying Model-Based Optimization to Hyperparameter Optimization in Machine Learning

Bernd Bischl

Ludwig-Maximilians-Universität München, München, Germany,
bernd.bischl@stat.uni-muenchen.de

Abstract. This talk will cover the main components of sequential model-based optimization algorithms. Algorithms of this kind represent the state-of-the-art for expensive black-box optimization problems and are getting increasingly popular for hyper-parameter optimization of machine learning algorithms, especially on larger data sets.

The talk will cover the main components of sequential model-based optimization algorithms, e.g., surrogate regression models like Gaussian processes or random forests, initialization phase and point acquisition.

In a second part I will cover some recent extensions with regard to parallel point acquisition, multi-criteria optimization and multi-fidelity systems for subsampled data. Most covered applications will use support vector machines as examples for hyper-parameter optimization.

The talk will finish with a brief overview of open questions and challenges.

On a few recent developments in Meta-Learning for Algorithm Ranking and Selection

Bernhard Pfahringer

Waikato University, Hamilton, New Zealand,
bernhard@cs.waikato.ac.nz

Abstract. This talk has two main parts. The first part will focus on the use of pair-wise meta-rules for algorithm ranking and selection. Such rules can provide interesting insights on their own, but they are also very valuable features for more sophisticated schemes like Random Forests. A hierarchical variant is able to address complexity issues when the number of algorithms to compare is substantial.

The second part of the talk will focus on meta-learning for data streams, which is a very active area of research currently. Stream algorithms need to be incremental, and be able to adapt to change in the distribution of the data. This poses new challenges for meta-learning.

The Potential Benefits of Data Set Filtering and Learning Algorithm Hyperparameter Optimization

Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier

Department of Computer Science, Brigham Young University, Provo, UT 84602 USA
msmith@axon.cs.byu.edu,martinez@cs.byu.edu,cgc@cs.byu.edu
http://axon.cs.byu.edu

Abstract. The quality of a model induced by a learning algorithm is dependent upon the training data *and* the hyperparameters supplied to the learning algorithm. Prior work has shown that a model's quality can be significantly improved by filtering out low quality instances or by tuning the learning algorithm hyperparameters. The potential impact of filtering and hyperparameter optimization (HPO) is largely unknown. In this paper, we estimate the *potential* benefits of instance filtering and HPO. While both HPO and filtering significantly improve the quality of the induced model, we find that filtering has a greater potential effect on the quality of the induced model than HPO, motivating future work in filtering.

1 Introduction

Given a set of training instances composed of input feature vectors and corresponding labels, the goal of supervised machine learning is to induce an accurate generalizing function (hypothesis) that maps feature vectors to labels. The quality of the induced function is dependent on the learning algorithm's hyperparameters *and* the quality of the training data. It is known that no learning algorithm or hyperparameter setting is best for all data sets (no free lunch theorem [26]) and that the performance of many learning algorithms is sensitive to their hyperparameter settings. It is also well-known that real-world data sets are typically noisy.

Prior work has shown that the generalization performance of an induced model can be significantly improved through hyperparameter optimization (HPO) [1], or by increasing the quality of the training data using techniques such as noise correction [11], instance weighting [17], or instance filtering [20]. Searching the hyperparameter space and improving the quality of the training data have generally been examined in isolation and the potential impact of their usage has not been examined. In this paper, we compare the effects of HPO with the effects of improving the quality of the training data through filtering. The results of our experiments provide insight into the potential effectiveness of both HPO and filtering.

We evaluate 6 commonly used learning algorithms and 46 data sets. We examine the effects of HPO and filtering by: 1) using a standard approach that selects the hyperparameters of an algorithm by maximizing the accuracy on a validation set and 2) using an optimistic approach that sets the hyperparameters for an algorithm using the 10-fold cross-validation accuracy. The standard and optimistic approaches are explained in more detail in Section 4. Essentially, the optimistic approach indicates how well a technique *could* perform if the training set were representative of the test set and provides insight into the *potential* benefit of a given technique. The standard approach provides a representative view of HPO and filtering in their present state and allows an evaluation of how well current HPO and filtering techniques fulfill their potential.

Using the standard approach, we find that in most cases both HPO and filtering significantly increase classification accuracy over using a learning algorithm with its default parameters trained on unfiltered data. For the optimistic estimates of HPO and filtering, we find that *filtering significantly improves the classification accuracy over HPO* for all of the investigated learning algorithms–increasing the accuracy more than HPO for almost all of the considered data sets. HPO achieves an average accuracy of 84.8% while filtering achieves an average accuracy of 89.1%. The standard approach for HPO and filtering achieves an average accuracy of 82.6% and 82.0% respectively. These results provide motivation for further research into developing algorithms that improve the quality of the training data.

2 Related Work

Smith et al. [21] found that a significant number of instances are difficult to classify correctly, that the hardness of each instance is dependent on its relationship with the other instances in the training set and that some instances can be detrimental. Thus, there is a need for improving how detrimental instances are handled during training as they affect the classification of other instances. Improving the quality of the training data has typically fallen into three approaches: filtering, cleaning, and instance weighting [7].

Each technique within an approach differs in how detrimental instances are identified. A common technique for filtering removes instances from a data set that are misclassified by a learning algorithm or an ensemble of learning algorithms [3]. Removing the training instances that are suspected to be noise and/or outliers prior to training has the advantage that they do not influence the induced model and generally increase classification accuracy. A negative side-effect of filtering is that beneficial instances can also be discarded and produce a worse model than if all of the training data had been used [18]. Rather than discarding the instances from a training set, noisy or possibly corrupted instances can be cleaned or corrected [11]. However, this could artificially corrupt valid instances. Alternatively, weighting weights suspected detrimental instances rather than discards them and allows for an instance to be considered on a continuum of detrimentality rather than making a binary decision [17].

Other methods exist for improving the quality of the training data, such as feature selection/extraction [8]. While feature selection and extraction can improve the quality of the training data, we focus on improving quality via filtering – facilitating a comparison between filtering and HPO on the same feature set.

Much of the previous work in improving the quality of the training data artificially corrupts training instances to determine how well an approach would work in the presence of noisy or mislabeled instances. In some cases, a given approach *only* has a significant impact when there are large degrees of artificial noise. In contrast, we do not artificially corrupt a data set to create detrimental instances. Rather, we seek to identify the detrimental instances that are already contained in a data set and show that correctly labeled, non-noisy instances can *also* be detrimental for inducing a model of the data. Properly handling detrimental instances can result in significant gains in accuracy.

The grid search and manual search are the most common types of HPO techniques in machine learning and a combination of the two approaches are commonly used [12]. Bergstra and Bengio [1] proposed to use a random search of the hyperparameter space. The premise of random HPO is that most machine learning algorithms have very few hyperparameters that considerably affect the final model while the other hyperparameters have little to no effect. Random search provides a greater variety of the hyperparameters that considerably affect the model. Given the same amount of time constraints, random HPO has been shown to outperform a grid search. Random search, while providing improvements over a grid-search, is unreliable for tuning the hyperparameters for some learning algorithms such as deep belief networks [2]. Bayesian optimization has also been used to search the hyperparameter space [23]. Bayesian optimization techniques model the dependence of an error function $\mathcal{E}$ on the hyperparameters λ as $p(\mathcal{E}|\lambda)$ using, for example, random forests [10] or Gaussian processes [23].

3 Preliminaries

Let T represent a training set composed of a set of input vectors $X = \{x_1, x_2, \ldots, x_n\}$ and corresponding label vectors $Y = \{y_1, y_2, \ldots, y_n\}$, i.e., $T = \{\langle x_i, y_i \rangle : x_i \in X \wedge y_i \in Y\}$. Given that in most cases, all that is known about a task is contained in the set of training instances T, at least initially, the training instances are generally considered equally. Most machine learning algorithms seek to induce a hypothesis $h : X \rightarrow Y$ that minimizes a specified loss function $\mathcal{L}(\cdot)$. As most real-world data sets contain some level of noise, there is generally a model-dependent regularization term $\mathcal{R}(\cdot)$ added to $\mathcal{L}(\cdot)$ that penalizes more complex models and aids in overfit avoidance. The noise in T may arise from errors in the data collection process such as typos or errors in data collection equipment. In addition to noise from errors, there may be non-noisy outlier instances due to the stochastic nature of the task. A hypothesis h is induced by a learning algorithm g trained on T with hyperparameters λ ($h = g(T, \lambda)$), such that:

$$h^* = \operatorname*{argmin}_{h \in \mathcal{H}} \frac{1}{|T|} \sum_{\langle x_i, y_i \rangle \in T} \mathcal{L}(h(x_i), y_i) + \alpha \mathcal{R}(h) \tag{1}$$

where α is a regularization parameter greater than or equal to 0 that determines how much weight to apply to the regularization term and $h(\cdot)$ returns the predicted class for a given input. The quality of the induced hypothesis h is characterized by its empirical error for a specified error function $\mathcal{E}$ on a test set V: $E(h, V) = \frac{1}{|V|} \sum_{\langle x_i, y_i \rangle \in V} \mathcal{E}(h(x_i), y_i)$ where V can be T or a disjoint set of instances. In k-fold cross-validation, the empirical error is the average empirical error from the k folds (i.e., $1/k\ E(h_i, V_i)$).

Characterizing the success of a learning algorithm at the data set level (e.g., accuracy or precision) optimizes over the entire training set and marginalizes the impact of a single training instance on an induced model. Some sets of instances can be more beneficial than others for inducing a model of the data and some can even be detrimental. By *detrimental instances*, we mean instances that have a negative impact on the induced model. For example, outliers or mislabeled instances are not as beneficial as border instances and are detrimental in many cases. In addition, other instances can be detrimental for inducing a model of the data even if they are labeled correctly. Formally, a set $\mathcal{D}$ of detrimental instances is a subset of the training data that, when used in training, increases the empirical error, i.e., $E(g(T, \lambda), V) > E(g(T - \mathcal{D}, \lambda), V)$.

The effect of training with detrimental instances is demonstrated in the hypothetical two-dimensional data set shown in Figure 1. Instances A and B represent detrimental instances. The solid line represents the "actual" classification boundary and the dashed line represents a potential induced classification boundary. Instances A and B adversely affect the induced classification boundary because they "pull" the classification boundary and cause several other instances to be misclassified that otherwise would have been classified correctly.

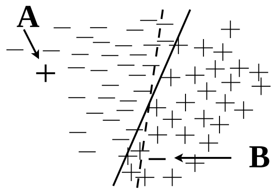


Fig. 1. Hypothetical 2-dimensional data set that shows the potential effects of detrimental instances in the training data on a learning algorithm.

Despite most learning algorithms having a mechanism to avoid overfitting, the presence of detrimental instances may still affect the induced model for many learning algorithms. Mathematically, the effect of each instance on the induced hypothesis is shown in Equation 1. The loss from each instance in T, including detrimental instances, is equally weighted. Detrimental instances have the most significant impact during the early stages of training where it is difficult to identify them [6]. The presence of $\mathcal{D}$ may also affect the value of $\mathcal{R}(h)$. For example, removing $\mathcal{D}$ from T could produce a "simpler" h that reduces $\mathcal{R}(h)$.

3.1 Hyperparameter Optimization

The quality of an induced model by a learning algorithm depends in part on the learning algorithm's hyperparameters. With hyperparameter optimization (HPO), the hyperparameter space Λ is searched to minimize the empirical error on V:

$$\underset{\lambda \in \Lambda}{\operatorname{argmin}} \, E(g(T, \lambda), V). \quad (2)$$

The hyperparameters can have a significant effect on the quality of the induced model as well as suppressing the effects of detrimental instances. For example, in a support vector machine, [4] use the ramp-loss function which limits the penalty on instances that are too far from the decision boundary rather than the more typical 0-1 loss function to handle detrimental instances. Suppressing the effects of detrimental instances with HPO improves the induced model, but does not change the fact that detrimental instances *still* affect the model. Each instance is still considered during the learning process though its influence may be lessened. We describe the method we use for HPO in Section 4.1.

3.2 Filtering

The quality of an induced model also depends on the quality of the training data where, for example, the quality of the training data can be measured by the amount of detrimental instances present. Low quality training data results in lower quality induced models. Improving the quality of the training data involves searching the training set space to find an optimal subset that minimizes the empirical error:

$$\operatorname*{argmin}_{t \in \mathcal{P}(T)} E(g(t, \lambda), V)$$

where t is a subset of T and $\mathcal{P}(T)$ is the power set of T. The removed instances obviously have no effect on the induced model. In Section 4.2, we describe how we identify detrimental instances and search for an optimal subset of the training data that minimizes empirical error.

4 Implementation Details

4.1 Bayesian Hyperparameter Optimization

In this paper, we use Bayesian optimization for HPO. Specifically, we use *sequential model-based optimization* (SMBO) [10] as it has been shown to yield better performance than grid and random search [23,24]. SMBO is a stochastic optimization framework that builds a probabilistic model $\mathcal{M}$ that captures the dependence of $\mathcal{E}$ on λ. SMBO first initializes $\mathcal{M}$. After initializing $\mathcal{M}$, SMBO searches the search space by 1) querying $\mathcal{M}$ for a promising λ to evaluate, 2) evaluating the loss $\mathcal{E}$ of using configuration λ, and then 3) updating $\mathcal{M}$ with λ and $\mathcal{E}$. Once the budgeted time is exhausted, the hyperparameter configuration with the minimal loss is returned.

To select a candidate hyperparameter configuration, SMBO relies on an acquisition function $a_{\mathcal{M}} : \Lambda \rightarrow \mathbb{R}$ which uses the predictive distribution of $\mathcal{M}$ to quantify how useful knowledge about λ would be. SMBO maximizes $a_{\mathcal{M}}$ over Λ to select the most useful hyperparameter configuration λ to evaluate next. One of the most prominent acquisition functions is the *positive expected improvement* (EI) over an existing error rate $\mathcal{E}_{min}$ [19]. If $\mathcal{E}(\lambda)$ represents the error rate of hyperparameter configuration λ, then the EI function over $\mathcal{E}_{min}$ is: $EI_{\mathcal{E}_{min}}(\lambda) = max\{\mathcal{E}_{min} - \mathcal{E}(\lambda), 0\}$. As $\mathcal{E}(\lambda)$ is unknown, the expectation of $\mathcal{E}(\lambda)$ with respect to the current model $\mathcal{M}$ can be computed as: $\mathbb{E}_{\mathcal{M}}[EI_{\mathcal{E}_{min}}(\lambda)] = \int_{-\infty}^{\mathcal{E}_{min}} max\{\mathcal{E}_{min} - \mathcal{E}, 0\} \cdot p(\mathcal{E}|\lambda) d\mathcal{E}$.

SMBO is dependent on the model class used for $\mathcal{M}$. Following [24], we use sequential model-based algorithm configuration (SMAC) [10] for $\mathcal{M}$ with EI as $a_{\mathcal{M}}$, although others could be used such as the tree-structured Parzen estimator. To model $p(\mathcal{E}|\lambda)$, we use random forests as they tend to perform well with discrete and continuous input data.

Using random forests, SMAC obtains a predictive mean μ_λ and variance σ^2_λ of $p(\mathcal{E}|\lambda)$ calculated using the predictions from the individual trees in the forest for λ. $p(\mathcal{E}|\lambda)$ is then modeled as a Gaussian distribution $\mathcal{N}(\mu_\lambda, \sigma^2_\lambda)$. To create diversity in the evaluated configurations, every second configuration is selected at random as suggested [24]. For k-fold cross-validation, the **standard approach** finds the hyperparameters that minimize the error for each of the k validation sets as shown in Equation 2. The **optimistic approach** finds the hyperparameters that minimize the k-fold cross-validation error: $\mathrm{argmin}_{\lambda \in \Lambda} \frac{1}{k} E(g(T_i, \lambda), V_i)$ where T_i and V_i are the training and validation sets for the ith fold. The hyperparameter space Λ is searched using Bayesian hyperparameter optimization for both approaches.

4.2 Filtering

Identifying detrimental instances is a non-trivial task. Fully searching the space of subsets of training instances generates 2^N subsets of training instances where N is the number of training instances. Even for small data sets, it is computationally infeasible to induce 2^N models to determine which instances are detrimental. There is no known way to determine how a set of instances will affect the induced classification function from a learning algorithm without inducing a classification function with the investigated set of instances removed from the training set.

The Standard Filtering Approach ($\mathcal{G}$-Filter) Previous work in noise handling has shown that class noise (e.g. mislabeled instances) is more detrimental than attribute noise [15]. Thus, searching for detrimental instances that are likely to be misclassified is a natural place to start. In other words, we search for instances where the probability of the class label is low given the feature values (i.e., low $p(y_i|x_i)$). In general, $p(y_i|x_i)$ does not make sense outside the context of an induced hypothesis. Thus, using an induced hypothesis h from a learning algorithm trained on T, the quantity $p(y_i|x_i)$ can be approximated as $p(y_i|x_i, h)$. After training a learning algorithm on T, the class distribution for an instance x_i can be estimated based on the output from the learning algorithm. Prior work has examined removing instances that are misclassified by a learning algorithm or an ensemble of learning algorithms [3]. We filter instances using an ensemble filter that removes instances that are misclassified by more than $x\%$ of the algorithms in the ensemble.

The dependence of $p(y_i|x_i, h)$ on a particular h can be lessened by summing over the space of all possible hypotheses:

$$p(y_i|x_i) = \sum_{h \in \mathcal{H}} p(y_i|x_i, h) p(h|T). \tag{3}$$

However, this formulation is infeasible to compute in most practical applications as $p(h|T)$ is generally unknown and $\mathcal{H}$ is large and possibly infinite. To sum over $\mathcal{H}$, one would have to sum over the complete set of hypotheses, or, since $h = g(T, \lambda)$, over the complete set of learning algorithms and their associated hyperparameters.

The quantity $p(y_i|x_i)$ can be estimated by restricting attention to a diverse set of representative algorithms (and hyperparameters). The diversity of the learning algorithms refers to the likelihood that the learning algorithms classify instances differently.

Table 1. Set of learning algorithms $\mathcal{G}$ used to estimate $p(y_i|x_i)$.

LEARNING ALGORITHMS
* MULTILAYER PERCEPTRON TRAINED WITH BACK PROPAGATION (MLP)
* DECISION TREE (C4.5)
* LOCALLY WEIGHTED LEARNING (LWL)
* 5-NEAREST NEIGHBORS (5-NN)
* NEAREST NEIGHBOR WITH GENERALIZATION (NNGE)
* NAÏVE BAYES (NB)
* RIPPLE DOWN RULE LEARNER (RIDOR)
* RANDOM FOREST (RANDFOREST)
* REPEATED INCREMENTAL PRUNING TO PRODUCE ERROR REDUCTION (RIPPER)

A natural way to approximate the unknown distribution $p(h|T)$ is to weight a set of representative learning algorithms, and their associated hyperparameters, $\mathcal{G}$, a priori with an equal, non-zero probability while treating all other learning algorithms as having zero probability. We select a diverse set of learning algorithms using unsupervised metalearning (UML) [13] to get a good representation of $\mathcal{H}$, and hence a reasonable estimate of $p(y_i|x_i)$. UML uses Classifier Output Difference (COD) [16] measures the diversity between learning algorithms as the probability that the learning algorithms make different predictions. UML clusters the learning algorithms based on their COD scores with hierarchical agglomerative clustering. Here, we consider 20 commonly used learning algorithms with their default hyperparameters as set in Weka [9]. A cut-point of 0.18 was chosen to create nine clusters and a representative algorithm from each cluster was used to create $\mathcal{G}$ as shown in Table 1.

Given a set $\mathcal{G}$ of learning algorithms, we approximate Equation 3 to the following:

$$p(y_i|x_i) \approx \frac{1}{|\mathcal{G}|}\sum_{j=1}^{|\mathcal{G}|} p(y_i|x_i, g_j(T,\lambda)) \tag{4}$$

where $p(h|T)$ is approximated as $\frac{1}{|\mathcal{G}|}$ and g_j is the jth learning algorithm from $\mathcal{G}$. As not all learning algorithms produce probabilistic outputs, the distribution $p(y_i|x_i, g_j(T,\lambda))$ is estimated using the Kronecker delta function in this paper.

The Optimistic Filtering Approach ($\mathcal{A}$-Filter) To measure the *potential* impact of filtering, we need to know how removing an instance or set of instances affects the generalization capabilities of the model. We measure this by dynamically creating an ensemble filter from $\mathcal{G}$ using a greedy algorithm for a given data set and learning algorithm. This allows us to find a specific ensemble filter that is best for filtering a given data set and learning algorithm combination. The adaptive ensemble filter is constructed by iteratively adding the learning algorithm g from $\mathcal{G}$ that produces the highest cross-validation classification accuracy when g is added to the ensemble filter. Because we are using the probability that an instance will be misclassified rather than a binary yes/no decision (Equation 4), we also use a threshold ϕ to determine which instances are detrimental. Instances with a $p(y_i|x_i)$ less than ϕ are discarded from the training set. A constant threshold value for ϕ is set to filter the instances for all iterations. The baseline accuracy for the adaptive approach is the accuracy of the learning algorithm without filtering. The search stops once adding one of the remaining learning algorithms to the ensemble filter does not increase accuracy, or all of the learning algorithms in $\mathcal{G}$ have been used.

Even though all of the detrimental instances are included for evaluation, the adaptive filter ($\mathcal{A}$-Filter) overfits the data since the cross-validation accuracy is used to determine which set of learning algorithms to use in the ensemble filter. This allows us to find the detrimental instances to examine the effects that they can have on an induced model. This is not feasible in practical settings, but provides insight into the potential improvement gained from filtering.

5 Filtering and HPO

In this section, we compare the effects of filtering with those of HPO using the optimistic and standard approaches presented in Section 4. The optimistic approach provides an approximation of the potential of HPO and filtering. In addition to reporting the average classification accuracy, we also report the average rank of each approach. The average accuracy and rank for each algorithm is determined using 5 by 10-fold cross-validation. Statistical significance between pairs of algorithms is determined using the Wilcoxon signed-ranks test (as suggested by [5]) with an alpha value of 0.05.

5.1 Experimental Methodology

For HPO, we use the version of SMAC implemented in auto-WEKA [24] as described in Section 4.1 Auto-WEKA searches the hyperparameter spaces for the learning algorithms in the Weka machine learning toolkit [9] for a specified amount of time. To estimate the amount of time required for a learning algorithm to induce a model of the data, we ran our selected learning algorithms with ten random hyperparameter settings and calculated the average and max running times. On average, a model was induced in less than 3 minutes. The longest time required to induce a model was 845 minutes. Based on this analysis, we run auto-WEKA for one hour for most of the data sets. An hour long search explores more than 512 hyperparameter configurations for most of the learning algorithm/data set combinations. The time limit is adjusted accordingly for the larger data sets. Following [24], we run four runs with different random seeds provided to SMAC.

For filtering using the ensemble filter ($\mathcal{G}$-filter), we use thresholds ϕ of 0.5, 0.7, and 0.9. Instances that are misclassified by more than $\phi\%$ of the learning algorithms are removed from the training set. The $\mathcal{G}$-filter uses all of the learning algorithms in the set $\mathcal{G}$ (Table 1). The accuracy on the test set from the value of ϕ that produces the highest accuracy on the training set is reported.

To show the effect of filtering detrimental instances and HPO on an induced model, we examine filtering and HPO in six commonly used learning algorithms (MLP trained with backpropagation, C4.5, kNN, Naïve Bayes, Random Forest, and RIPPER) on a set of 46 UCI data sets [14]. The LWL, NNge, and Ridor learning algorithms are not used

Table 2. Results for maximizing the 10-fold cross-validation accuracy for HPO and filtering.

	MLP	C4.5	*k*NN	NB	RF	RIP
ORIG	82.28 (2.98)	81.30 (2.91)	80.56 (2.74)	77.66 (2.70)	82.98 (2.89)	79.86 (2.96)
HPO	86.37 (1.87)	84.25 (1.96)	83.89 (2.22)	80.89 (1.96)	86.81 (1.85)	82.08 (1.80)
VS ORIG	45,0,1	42,1,3	34,1,11	34,0,12	44,0,2	**46,0,0**
A-FILTER	**89.96 (1.13)**	**88.74 (1.09)**	**91.14 (1.02)**	**82.74 (1.30)**	**91.02 (1.20)**	**88.16 (1.24)**
VS ORIG	**46,0,0**	**46,0,0**	**46,0,0**	**44,2,0**	**43,3,0**	44,0,2
VS HPO	39,1,6	41,1,4	45,0,1	32,0,14	37,0,9	37,0,9

for analysis because they do not scale well with the larger data sets–not finishing due to memory overflow or large amounts of running time.[1]

5.2 Optimistic Approach

The optimistic approach indicates how well a model *could* generalize on novel data. Maximizing the cross-validation accuracy is a type of overfitting. However, using 10-fold cross-validation accuracy for HPO and filtering, essentially measures the generalization capability of a learning algorithm for a given data set.

The results comparing the potential benefits of HPO and filtering are shown in Table 2. Each section gives the average accuracy and average rank for each learning algorithm as well as the number of times the algorithm is greater than, equal to, or less than a compared algorithm. HPO and the adaptive filter significantly increase the classification accuracy for all of the investigated learning algorithms. The values in bold represent if HPO or the adaptive filter is significantly greater than the other. For all of the investigated learning algorithms, the $\mathcal{A}$-filter significantly increases the accuracy over HPO. The closest the two techniques come to each other is for NB, where the $\mathcal{A}$-filter achieves an accuracy of 82.74% and an average rank of 1.30 while HPO achieves an accuracy of 80.89% and an average rank of 1.96. For all learning algorithms other than NB, the average accuracy is about 89% for filtering and 84% for HPO. Thus, filtering has a greater potential for increase in generalization accuracy. The difficulty lies in how to find the optimal set of training instances.

As might be expected, there is no set of learning algorithms that is the optimal ensemble filter for all algorithms and/or data sets. Table 3 shows the frequency for which a learning algorithm with default hyperparameters was selected for filtering by the $\mathcal{A}$-filter. The greatest percentage of cases an algorithm is selected for filtering for each learning algorithm is in bold. The column "ALL" refers to the average from all of the learning algorithms as the base learner. No instances are filtered in 5.36% of the cases. Thus, given the right filter, filtering to some extent increases the classification accuracy in about 95% of the cases. Furthermore, random forest, NNge, MLP, and C4.5 are the most commonly chosen algorithms for inclusion in the ensemble filter. However, no one learning algorithm is selected in more than 27% of the cases. The filtering algorithm that is most appropriate is dependent on the data set and the learning algorithm.

[1] For the data sets on which the learning algorithms did finish, the effects of HPO and filtering on LWL, NNge, and Ridor are consistent with the other learning algorithms.

Table 3. The frequency of selecting a learning algorithm when adaptively constructing an ensemble filter. Each row gives the percentage of cases that an algorithm was included in the ensemble filter for the learning algorithm in the column.

	ALL	MLP	C4.5	*k*NN	NB	RF	RIP
NONE	5.36	2.69	2.95	3.08	5.64	5.77	1.60
MLP	18.33	16.67	15.77	20.00	**25.26**	23.72	16.36
C4.5	17.17	17.82	15.26	22.82	14.49	13.33	20.74
5NN	12.59	11.92	14.23	1.28	10.00	17.18	16.89
LWL	6.12	3.59	3.85	4.36	23.72	3.33	3.59
NB	7.84	5.77	6.54	8.08	5.13	10.26	4.92
NNGE	19.49	**26.67**	21.15	21.03	11.15	**24.74**	23.40
RF	**21.14**	22.95	**26.54**	**23.33**	15.77	15.13	**24.20**
RID	14.69	14.87	16.79	18.33	11.92	16.54	12.77
RIP	8.89	7.82	7.69	8.85	13.08	7.44	4.39

This coincides with the findings from [18] that the efficacy of noise filtering in the nearest-neighbor classifier is dependent on the characteristics of the data set. Understanding the efficacy of filtering and determining *which* filtering approach to use for a given algorithm/data set is a direction of future work.

Analysis. In some cases HPO achieves a lower accuracy than orig, showing the complexity of HPO. The $\mathcal{A}$-Filter, on the other hand, never fails to improve the accuracy. Thus, higher quality data can compensate for hyperparameter settings and suggests that the instance space may be less complex and/or richer than the hyperparameter space. Of course, filtering does not outperform HPO in all cases, but it does so in the majority of cases.

5.3 Standard Approach

The previous results show the potential impact of filtering and HPO. We now examine HPO and filtering using the standard approach to highlight the need for improvement in filtering. The results comparing the $\mathcal{G}$-filter, HPO, and using the default hyperparameters trained on the original data set are shown in Table 4. HPO significantly increases the classification accuracy over not using HPO for all of the learning algorithms. Filtering significantly increases the accuracy for all of the investigated algorithms except for random forests. Comparing HPO and the G-Filter, only HPO for naïve Bayes and random forests significantly outperforms the $\mathcal{G}$-filter.

Analysis. In their current state, HPO and filtering generally improve the quality of the induced model. The results justify the computational overhead required to run HPO. Despite these results, using the default hyperparameters result in higher classification accuracy for 11 of the 46 data sets for C4.5, 12 for *k*NN, and 12 for NB, highlighting the complexity of searching over the hyperparameter space Λ. The effectiveness of HPO is dependent on the data set as well as the learning algorithm. Typically, as was done here, a single filtering technique is used for a set of data sets with no model of the dependence of a learning algorithm on the training instances. The accuracies for

Table 4. Performance comparison of using the default hyperparameters, HPO, and the $\mathcal{G}$-filter.

	MLP	C4.5	*k*NN	NB	RF	RIP
ORIG	82.28 (2.54)	81.3 (2.13)	80.56 (2.26)	77.66 (2.28)	82.98 (2.28)	79.86 (2.46)
HPO	83.08 (1.78)	82.42 (1.76)	82.85 (1.59)	**81.11 (1.63)**	**84.72 (1.54)**	81.15 (1.74)
VS ORIG	32,0,14	29,1,16	31,5,10	31,2,13	34,0,12	30,2,14
G-FILTER	84.17 (1.61)	81.9 (1.96)	81.61 (2.00)	79.49 (2.02)	83.49 (2.17)	81.25 (1.72)
VS ORIG	39,0,7	23,5,18	27,1,18	28,1,17	25,0,21	37,0,9
VS HPO	22,3,21	19,1,26	17,1,28	16,0,30	13,0,33	20,2,24

filtering and HPO are significantly lower than the optimistic estimate given in Section 5.2 motivating future work in HPO and especially in filtering.

6 Conclusion

In this paper, we compared the potential benefits of filtering with HPO. HPO may reduce the effects of detrimental instances on an induced model but the detrimental instances are still considered in the learning process. Filtering, on the other hand, removes the detrimental instances–completely eliminating their effects on the induced model.

We used an optimistic approach to estimate the potential accuracy of each method. Using the optimistic approach, both filtering and HPO significantly increase the classification accuracy for all of the considered learning algorithms. However, *filtering has a greater potential effect* on average, increasing the classification accuracy from 80.8% to 89.1% on the observed data sets. HPO increases the average classification accuracy to 84.8%. Future work includes developing models to understand the dependence of the performance of learning algorithms given the instances used for training. To better understand how instances affect each other, we are examining the results from machine learning experiments stored in repositories that include which instances were used for training and their predicted class [25,22]. We hope that the presented results provide motivation for improving the quality of the training data.

References

1. J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.
2. J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.
3. C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.
4. R. Collobert, F. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *Proceedings of the 23rd International Conference on Machine learning*, pages 201–208, 2006.
5. J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

6. J. L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99, 1993.
7. B. Frénay and M. Verleysen. Classification in the presence of label noise: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014.
8. I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.
9. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
10. F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Learning and Intelligent Optimization Conference*, pages 507–523, 2011.
11. J. Kubica and A. Moore. Probabilistic noise identification and data cleaning. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pages 131–138, 2003.
12. H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, pages 473–480, 2007.
13. J. Lee and C. Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841, 2011.
14. M. Lichman. UCI machine learning repository, 2013.
15. D. F. Nettleton, A. Orriols-Puig, and A. Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33(4):275–306, 2010.
16. A. H. Peterson and T. R. Martinez. Estimating the potential for combining learning models. In *Proceedings of the ICML Workshop on Meta-Learning*, pages 68–75, 2005.
17. U. Rebbapragada and C. E. Brodley. Class noise mitigation through instance weighting. In *Proceedings of the 18th European Conference on Machine Learning*, pages 708–715, 2007.
18. J. A. Sáez, J. Luengo, and F. Herrera. Predicting noise filtering efficacy with data complexity measures for nearest neighbor classification. *Pattern Recognition*, 46(1):355–364, 2013.
19. M. Schonlau, W. J. Welch, and D. R. Jones. *Global versus local search in constrained optimization of computer models*, volume Volume 34 of *Lecture Notes–Monograph Series*, pages 11–25. Institute of Mathematical Statistics, Hayward, CA, 1998.
20. M. R. Smith and T. Martinez. Improving classification accuracy by identifying and removing instances that should be misclassified. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2690–2697, 2011.
21. M. R. Smith, T. Martinez, and C. Giraud-Carrier. An instance level analysis of data complexity. *Machine Learning*, 95(2):225–256, 2014.
22. M. R. Smith, A. White, C. Giraud-Carrier, and T. Martinez. An easy to use repository for comparing and improving machine learning algorithm usage. In *Proceedings of the 2014 International Workshop on Meta-learning and Algorithm Selection (MetaSel)*, pages 41–48, 2014.
23. J. Snoek, H. Larochelle, and R. Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. 2012.
24. C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *proceedings of the 19th International Conference on Knowledge Discovery and Data Mining*, pages 847–855, 2013.
25. J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.
26. D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

Learning Data Set Similarities for Hyperparameter Optimization Initializations

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme

Information Systems and Machine Learning Lab
Universitätsplatz 1, 31141 Hildesheim, Germany
{wistuba,schilling,schmidt-thieme}@ismll.uni-hildesheim.de

Abstract. Current research has introduced new automatic hyperparameter optimization strategies that are able to accelerate this optimization process and outperform manual and grid or random search in terms of time and prediction accuracy. Currently, meta-learning methods that transfer knowledge from previous experiments to a new experiment arouse particular interest among researchers because it allows to improve the hyperparameter optimization.

In this work we further improve the initialization techniques for sequential model-based optimization, the current state of the art hyperparameter optimization framework. Instead of using a static similarity prediction between data sets, we use the few evaluations on the new data sets to create new features. These features allow a better prediction of the data set similarity. Furthermore, we propose a technique that is inspired by active learning. In contrast to the current state of the art, it does not greedily choose the best hyperparameter configuration but considers that a time budget is available. Therefore, the first evaluations on the new data set are used for learning a better prediction function for predicting the similarity between data sets such that we are able to profit from this in future evaluations. We empirically compare the distance function by applying it in the scenario of the initialization of SMBO by meta-learning. Our two proposed approaches are compared against three competitor methods on one meta-data set with respect to the average rank between these methods and show that they are able to outperform them.

1 Introduction

Most machine learning algorithms depend on hyperparameters and their optimization is an important part of machine learning techniques applied in practice. Automatic hyperparameter tuning is catching more and more attention by the machine learning community for two simple but important reasons. Firstly, the omnipresence of hyperparameters affects the whole community such that everyone is affected by the time-consuming task of optimizing hyperparameters either by manually tuning them or by applying a grid search. Secondly, in many cases the final hyperparameter configurations decides whether an algorithm is state of the art or just moderate such that the task of hyperparameter optimization is as important as developing new models [2,4,13,17,20]. Furthermore, hyperparameter optimization has shown to be able to also automatically perform algorithm and preprocessing selection by considering this as a further hyperparameter [20].

Sequential model-based optimization (SMBO) [9] is the current state of the art for hyperparameter optimization and has proven to outperform alternatives such as grid search or random search [17,20,2]. Recent research try to improve the SMBO framework by applying meta-learning on the hyperparameter optimization problem. The key concept of meta-learning is to transfer knowledge gained for an algorithm on past experiments on different data sets to new experiments. Currently, two different, orthogonal ways of transferring this knowledge exist. One possibility is to initialize SMBO by using hyperparameter configurations that have been best on previous experiments [5,6]. Another possibility is to use surrogate models that are able to learn across data sets [1,19,23].

We improve the former strategy by using an adaptive initialization strategy. We predict the similarity between data sets by using meta-features and features that express the knowledge gathered about the new data set so far. Having a more accurate approximation of the similarity between data sets, we are able to provide a better initialization. We provide empirical evidence that the new features provide better initializations in two different experiments.

Furthermore, we propose an initialization strategy that is based on the active learning idea. We try to evaluate hyperparameter configurations that are non-optimal for the short term but promise better results than choosing greedily the hyperparameter configuration that will provide the best result in expectation. To the best of our knowledge, we are the first that propose this idea in context of hyperparameter optimization for SMBO.

2 Related Work

Initializing hyperparameter optimization through meta-learning was proven to be effective [5,7,15,6]. Reif et al. [15] suggests to choose those hyperparameter configurations for a new data set that were best on a similar data set in the context of evolutionary parameter optimization. Here, the similarity was defined through the distance among meta-features, descriptive data set features. Recently, Feurer et al. [5] followed their lead and proposed the same initialization for sequential model-based optimization (SMBO), the current state of the art hyperparameter optimization framework. Later, they extended their work by learning a regression model on the meta-features that predicts the similarity between data sets [6].

Learning a surrogate model, the component in the SMBO framework that tries to predict the performance for a specific hyperparameter configuration on a data set, that is not only learned on knowledge of the new data set but additionally across knowledge from experiments on other data sets [1,19,23,16] is another option to transfer knowledge as well as pruning [22]. This idea is related but orthogonal to our work and can benefit from a good initialization and is no replacement for a good initialization strategy.

We propose to add features based on the performance of an algorithm on a data set for a specific hyperparameter configuration. Pfahringer et al. [12] propose to use landmark features. These features are estimated by evaluating simple algorithms on the data sets of the past and new experiment. In comparison to our features, these features are no by-product of the optimization process but have to be computed and hence need additional time. Even though these are simple algorithms, these features are problem-dependent (classification, regression, ranking, structured prediction all need their own landmark features). Furthermore, "simple" classifiers such as nearest neighbors as proposed by the authors can become very time-consuming for large data sets which are those data sets we are interested in.

Relative landmarks proposed by Leite et al. [10] are not and cannot be used as meta-features. They are used within the hyperparameter optimization strategy that is used instead of SMBO but are similar in that way that they are also given as a by-product and are computed using the relationship between hyperparameter configurations on each data set.

3 Background

In this section the hyperparameter optimization problem is formally defined. For the sake of completeness, also the sequential model-based optimization framework is presented.

3.1 Hyperparameter Optimization Problem Setup

A machine learning algorithm $\mathcal{A}_{\boldsymbol{\lambda}}$ is a mapping $\mathcal{A}_{\boldsymbol{\lambda}} : \mathcal{D} \to \mathcal{M}$ where $\mathcal{D}$ is the set of all data sets, $\mathcal{M}$ is the space of all models and $\boldsymbol{\lambda} \in \Lambda$ is the chosen hyperparameter configuration with $\Lambda = \Lambda_1 \times \ldots \times \Lambda_P$ being the P-dimensional hyperparameter space. The learning algorithm estimates a model $M_{\boldsymbol{\lambda}} \in \mathcal{M}$ that minimizes the objective function that linearly combines the loss function $\mathcal{L}$ and the regularization term $\mathcal{R}$:

$$\mathcal{A}_{\boldsymbol{\lambda}}\left(D^{(train)}\right) := \arg\min_{M_{\boldsymbol{\lambda}} \in \mathcal{M}} \mathcal{L}\left(M_{\boldsymbol{\lambda}}, D^{(train)}\right) + \mathcal{R}\left(M_{\boldsymbol{\lambda}}\right). \tag{1}$$

Then, the task of *hyperparameter optimization* is finding the hyperparameter configuration $\boldsymbol{\lambda}^*$ that minimizes the loss function on the validation data set i.e.

$$\boldsymbol{\lambda}^* := \arg\min_{\boldsymbol{\lambda} \in \Lambda} \mathcal{L}\left(\mathcal{A}_{\boldsymbol{\lambda}}\left(D^{(train)}\right), D^{(valid)}\right) =: \arg\min_{\boldsymbol{\lambda} \in \Lambda} f_D\left(\boldsymbol{\lambda}\right). \tag{2}$$

3.2 Sequential Model-based Optimization

Exhaustive hyperparameter search methods such as grid search are becoming more and more expensive. Data sets are growing, models are getting more complex and have high-dimensional hyperparameter spaces Sequential model-based optimization (SMBO) [9] is a black-box optimization framework that replaces the time-consuming function f to evaluate with a cheap-to-evaluate surrogate function Ψ that approximates f. With the help of an acquisition function such as expected improvement [9], sequentially new points are chosen such that a balance between exploitation and exploration is met and f is optimized. In our scenario, evaluating f is equivalent to learning a machine learning algorithm on some training data for a given hyperparameter configuration and estimate the models performance on a hold-out data set.

Algorithm 1 outlines the SMBO framework. It starts with an observation history $\mathcal{H}$ that equals the empty set in cases where no knowledge from past experiments is used [2,8,17] or is non-empty in cases where past experiments are used [1,19,23] or SMBO has been initialized [5,6]. First, the surrogate model Ψ is fitted to $\mathcal{H}$ where Ψ can be any regression model. Since the acquisition function a usually needs some certainty about the prediction, common choices are Gaussian processes [1,17,19,23] or ensembles such as random forests [8]. The acquisition function chooses the next candidate to evaluate. A common choice for the acquisition function is expected improvement [9] but further acquisition functions exist such as probability of improvement [9], the conditional entropy of the minimizer [21] or a multi-armed bandit based criterion [18]. The evaluated candidate is finally added to the set of observations. After T-many SMBO iterations, the best currently found hyperparameter configuration is returned.

Algorithm 1 Sequential Model-based Optimization

```
Input: Hyperparameter space $\Lambda$, observation history $\mathcal{H}$, number of iterations $T$, acquisition function $a$, surrogate model $\Psi$.
Output: Best hyperparameter configuration found.
1: for $t = 1$ to $T$ do
2:     Fit $\Psi$ to $\mathcal{H}$
3:     $\boldsymbol{\lambda}_* \leftarrow \arg\max_{\boldsymbol{\lambda}_* \in \Lambda} a(\boldsymbol{\lambda}_*, \Psi)$
4:     Evaluate $f(\boldsymbol{\lambda}_*)$
5:     $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\boldsymbol{\lambda}_*, f(\boldsymbol{\lambda}_*))\}$
6: return $\arg\min_{(\boldsymbol{\lambda}^*, f(\boldsymbol{\lambda}^*)) \in \mathcal{H}} f(\boldsymbol{\lambda}^*)$
```

4 Adaptive Initialization

Recent initialization techniques for sequential model-based optimization compute a static initialization hyperparameter configuration sequence [5,6]. The advantage of this idea is that during the initialization there is no time overhead for computing the next hyperparameter configuration. The disadvantage is that knowledge gained during the initialization about the new data set is not used for further initialization queries. Hence, we propose to use an adaptive initialization technique. Firstly, we propose to add some additional meta-features generated from this knowledge, which follows the idea of landmark features [12] and relative landmarks [10], respectively. Secondly, we apply the idea of active learning and try to choose the hyperparameter configurations that will allow to learn a precise ranking of data sets with respect to their similarity to the new data set.

4.1 Adaptive Initialization Using Additional Meta-Features

Feurer et al. [5] propose to improve the SMBO framework by an initialization strategy as shown in Algorithm 2. The idea is to use the hyperparameter configurations that have been best on other data sets. Those hyperparameter configurations are ranked with respect to the predicted similarity to the new data set D_{new} for which the best hyperparameter configuration needs to be found. The true distance function $d : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ between data sets is unknown such that Feurer et al. [5] propose to approximate it by $\hat{d}\left(\mathbf{m}_i, \mathbf{m}_j\right) = \left\|\mathbf{m}_i - \mathbf{m}_j\right\|_p$ where $\mathbf{m}_i$ is the vector of meta-features of data set D_i. In their extended work [6], they propose to use a random forest to learn $\hat{d}$ using training instances of the form $\left(\left(\mathbf{m}_i, \mathbf{m}_j\right)^T, d\left(D_i, D_j\right)\right)$. This initialization does not consider the performance of the hyperparameters configurations already evaluated on the new data set.

We propose to keep Feurer's initialization strategy [6] untouched and only add additional meta-features that capture the information gained on the new data set. Meta-features as defined in Equation 3 are added to the set of meta-features for all hyperparameter configurations $\boldsymbol{\lambda}_k, \boldsymbol{\lambda}_l$ that are evaluated on the new data set. The symbol $\oplus$ denotes an exclusive or. An additional difference is that now after each step the meta-features and the model $\hat{d}$ needs to be updated (before Line 2).

$$m_{D_i,D_j,\boldsymbol{\lambda}_k,\boldsymbol{\lambda}_l} = \mathbb{I}\left(f_{D_i}\left(\boldsymbol{\lambda}_k\right) > f_{D_i}\left(\boldsymbol{\lambda}_l\right) \oplus f_{D_j}\left(\boldsymbol{\lambda}_k\right) > f_{D_j}\left(\boldsymbol{\lambda}_l\right)\right) \tag{3}$$

We make here the assumption that the same set of hyperparameter configurations were evaluated across all training data sets. If this is not the case, this problem can be overcome by approximating the respective value by learning surrogate models for the training data sets as well. Since for these data sets much information is available, the prediction will be reliable enough. For simplicity, we assume that the former is the case.

The target $\mathbf{d}$ can be any similarity measure that reflects the true similarity between data sets. Feurer et al. [6] propose to use the Spearman correlation coefficient while we are using the number of discordant pairs in our experiments, i.e.

$$d\left(D_i, D_j\right) := \frac{\sum_{\boldsymbol{\lambda}_k,\boldsymbol{\lambda}_l \in \Lambda} \mathbb{I}\left(f_{D_i}\left(\boldsymbol{\lambda}_k\right) > f_{D_i}\left(\boldsymbol{\lambda}_l\right) \oplus f_{D_j}\left(\boldsymbol{\lambda}_k\right) > f_{D_j}\left(\boldsymbol{\lambda}_l\right)\right)}{|\Lambda|\left(|\Lambda| - 1\right)} \tag{4}$$

where Λ is the set of hyperparameter configurations observed on the training data sets. This change will have no influence on the prediction quality for the traditional meta-features but is better suited for the proposed landmark features in Equation 3.

Algorithm 2 Sequential Model-based Optimization with Initialization

Input: Hyperparameter space Λ, observation history $\mathcal{H}$, number of iterations T, acquisition function a, surrogate model Ψ, set of data sets $\mathcal{D}$, number of initial hyperparameter configurations I, prediction function for distances between data sets $\hat{d}$.

Output: Best hyperparameter configuration found.

```
1: for $i = 1$ to $I$ do
2:     Predict the distances $\hat{d}\left(D_j, D_{new}\right)$ for all $D_j \in \mathcal{D}$.
3:     $\boldsymbol{\lambda}_* \leftarrow$ Select best hyperparameter configuration on the i-th closest data set.
4:     Evaluate $f\left(\boldsymbol{\lambda}_*\right)$
5:     $\mathcal{H} \leftarrow \mathcal{H} \cup \left\{\left(\boldsymbol{\lambda}_*, f\left(\boldsymbol{\lambda}_*\right)\right)\right\}$
6: return SMBO$(\Lambda, \mathcal{H}, T - I, a, \Psi)$
```

4.2 Adaptive Initialization Using Active Learning

We propose to extend the method from the last section by investing few initialization steps by carefully selecting hyperparameter configurations that will lead to good additional meta-features and provide a better prediction function $\hat{d}$. An additional meta-feature is useful if the resulting regression model $\hat{d}$ predicts the distances of the training data sets to the new data set such that the ordering with respect to the predicted distances reflects the ordering with respect to the true distances. If I is the number of initialization steps and $K < I$ is the number of steps to choose additional meta-features, then the K hyperparameter configurations need to be chosen such that the precision at $I - K$ with respect to the ordering is optimized. The precision at n is defined as

$$\text{prec@n} := \frac{\left|\{\text{n closest data sets to } D_{new} \text{ wrt. } d\} \cap \left\{\text{n closest data sets to } D_{new} \text{ wrt. } \hat{d}\right\}\right|}{n} \tag{5}$$

Algorithm 3 presents the method we used to find the best first K hyperparameter configurations. In a leave-one-out cross-validation over all training data sets $\mathcal{D}$ the pair of hyperparameter configurations $(\boldsymbol{\lambda}_j, \boldsymbol{\lambda}_k)$ is sought that achieves the best precision at $I - K$ on average (Lines 1 to 7). Since testing all different combinations of K different hyperparameter configurations is too expensive, only the best pair is searched. The remaining $K - 2$ hyperparameter configurations are greedily added to the final set of initial hyperparameter configurations Λ_{active} as described in Lines 8 to 15. The hyperparameter configuration is added to Λ_{active} that performs best on average with all hyperparameter configurations chosen so far. After choosing K hyperparameter configurations as described in Algorithm 3, the remaining $I - K$ hyperparameter configurations are chosen as described in Section 4.1. The time needed for computing the first K hyperparameter highly depends on the size of Λ. To speed up the process, we reduced Λ to those hyperparameter configurations that have been best on at least one training data set.

Algorithm 3 Active Initialization

Input: Set of training data sets $\mathcal{D}$, set of observed hyperparameter configurations on $\mathcal{D}$ Λ, number of active learning steps K, number of initial hyperparameter configurations I.

Output: List of hyperparameter configurations that will lead to good predictors for $\hat{d}$.

```
 1: for $i = 1$ to $|\mathcal{D}|$ do
 2:   $\mathcal{D}_{test} \leftarrow \{D_i\}$
 3:   $\mathcal{D}_{train} \leftarrow \mathcal{D} \setminus \{D_i\}$
 4:   for all $\boldsymbol{\lambda}_j, \boldsymbol{\lambda}_k \in \Lambda, j \neq k$ do
 5:     Train $\hat{d}$ on $\mathcal{D}_{train}$ using the additional feature $m_{\cdot,\cdot,\boldsymbol{\lambda}_j,\boldsymbol{\lambda}_k}$ (Eq. 3)
 6:     Estimate the precision at $I - K$ of $\hat{d}$ on $\mathcal{D}_{test}$.
 7: $\Lambda_{active} \leftarrow \{\boldsymbol{\lambda}_j, \boldsymbol{\lambda}_k\}$ with highest average precision.
 8: for $k = 1$ to $K - 2$ do
 9:   for $i = 1 \ldots |\mathcal{D}|$ do
10:     $\mathcal{D}_{test} \leftarrow \{D_i\}$
11:     $\mathcal{D}_{train} \leftarrow \mathcal{D} \setminus \{D_i\}$
12:     for all $\boldsymbol{\lambda} \in \Lambda \setminus \Lambda_{active}$ do
13:       Train $\hat{d}$ on $\mathcal{D}_{train}$ using the additional features (Eq. 3) implied by $\Lambda_{active} \cup \{\boldsymbol{\lambda}\}$.
14:       Estimate the precision at $I - K$ of $\hat{d}$ on $\mathcal{D}_{test}$.
15:   Add $\boldsymbol{\lambda}$ with highest average precision to $\Lambda_{active}$.
16: return $\Lambda_{active}$
```

5 Experimental Evaluation

5.1 Initialization Strategies

Following initialization strategies will be considered in our experiments.

Random Best Initialization (RBI) This initialization is a very simple initialization. I training data sets from the training data sets $\mathcal{D}$ are chosen at random and its best hyperparameter configurations are used for the initialization.

Nearest Best Initialization (NBI) This is the initialization strategy proposed by Reif et al. and Feurer et al. [5,15]. Instead of choosing I training data sets at random, they are chosen with respect to the similarity between the meta-features listed in Table 1. Then, like for RBI, the best hyperparameter configurations on these data sets are chosen for initialization.

Predictive Best Initialization (PBI) Feurer et al. [6] propose to learn the distance between data sets using a regression model based on the meta-features. These predicted distances are used to find the most similar data sets to the new data set and like before, the best hyperparameter configurations on these data sets are chosen for initialization.

Adaptive Predictive Best Initialization (aPBI) This is our extension to PBI presented in Section 4.1 that adapts to the new data set during initialization by including the features defined in Equation 3.

Active Adaptive Predictive Best Initialization (aaPBI) Active Adaptive Predictive Best Initialization is described in Section 4.2 and extends aPBI by using the first K steps to choose hyperparameter configurations that will result in promising meta-features. After the first K iterations, it behaves equivalent to aPBI.

Table 1. List of all meta-features used.

Number of Classes	Class Probability Max
Number of Instances	Class Probability Mean
Log Number of Instances	Class Probability Standard Deviation
Number of Features	Kurtosis Min
Log Number of Features	Kurtosis Max
Data Set Dimensionality	Kurtosis Mean
Log Data Set Dimensionality	Kurtosis Standard Deviation
Inverse Data Set Dimensionality	Skewness Min
Log Inverse Data Set Dimensionality	Skewness Max
Class Cross Entropy	Skewness Mean
Class Probability Min	Skewness Standard Deviation

5.2 Meta-Features

Meta-features are supposed to be discriminative for a data set and can be estimated without evaluating f. Many surrogate models [1,19,23] and initialization strategies [5,15] use them to predict the similarity between data sets. For the experiments, the meta-features listed in Table 1 are used. For an in-depth explanation we refer the reader to [1,11].

5.3 Meta-Data Set

For creating the two meta-data sets, 50 classification data sets from the UCI repository are chosen at random. All instances are merged in cases there were already train/test splits, shuffled and split into 80% train and 20% test. A support vector machine (SVM) [3] was used to create the meta-data set. The SVM was trained using three different kernels (linear, polynomial and Gaussian) and the labels of the meta-instances were estimated by evaluating the trained model on the test split. The total hyperparameter dimension is six, three dimensions for indicator variables that indicates which kernel was chosen, one for the trade-off parameter C, one for the width γ of the Gaussian kernel and one for the degree of the polynomial kernel d. If the hyperparameter is not involved, e.g. the degree if the Gaussian kernel was used, it is set to 0. The test accuracy is precomputed on a grid $C \in \{2^{-5}, \ldots, 2^{6}\}$, $\gamma \in \{10^{-4}, 10^{-3}, 10^{-2}, 0.05, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 10^{2}, 10^{3}\}$ and $d \in \{2, \ldots, 10\}$ resulting into 288 meta-instances per data set. The meta-data sets is extended by the meta-features listed in Table 1.

5.4 Experiments

Two different experiments are conducted. First, state of the art initialization strategies are compared with respect to the average rank after I initial hyperparameter configurations. Second, the long term effect on the hyperparameter optimization is compared. Even though the initial hyperparameter configuration lead to good results after I results, the ultimate aim is to have good results at the end of the hyperparameter optimization after T iterations.

We evaluated all methods in a leave-one-out cross-validation per data set. All data sets but one are used for training and the data set not used for training is the new data set. The results reported are the average over 100 repetitions. Due to the randomness, we used 1,000 repetitions whenever RBI was used.

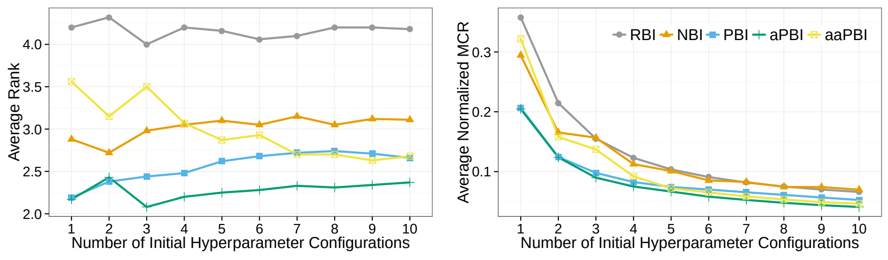


Fig. 1. Five different tuning strategies are compared. Our strategy aPBI is able to outperform the state of the art. Our second strategy aaPBI is executed under the assumption that it has 10 initial steps. The first steps are used for active learning and hence the bad results in the beginning are not surprising.

Comparison to Other Initialization Strategies For all our experiments 10 initialization steps are used, $I = 10$. In Figure 1 the different initialization strategies are compared with respect to the average rank in the left plot. Our initialization strategy aPBI benefits from the new, adaptive features and is able to outperform the other initialization strategies. Our second strategy aaPBI uses three active learning steps, $K = 3$. This explains the behavior in the beginning. After these initial steps it is able to catch up and finally surpass PBI but it is not as good as aPBI. Furthermore, a distance function learned on the meta-features (PBI) provides better results than a fixed distance function (NBI) which confirms the results by Feurer et al. [6]. All initialization strategies are able to outperform RBI which confirms that the meta-features contain information that concludes information about the similarity between data sets. The right plot shows the average misclassification (MCR) rate where the MCR is scaled to 0 and 1 for each data set.

Comparison with Respect to the Long Term Effect To have a look at the long term effect of the initialization strategies, we compare the different initialization strategies in the SMBO framework using two common surrogate models. One is a Gaussian process with a squared exponential kernel with automatic relevance determination [17]. The kernel parameters are estimated by maximizing the marginal likelihood on the meta-training set [14]. The second is a random forest [8]. Again, aPBI provides strictly better results than PBI for both surrogate models and outperforms any other initialization strategy for the Gaussian process. Our alternative strategy aaPBI performs mediocre for the Gaussian process but good for the random forest.

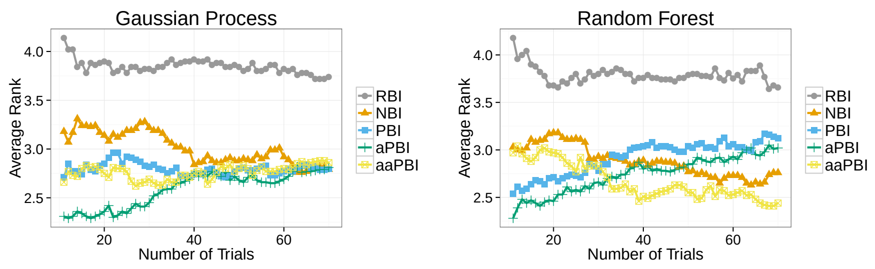


Fig. 2. The effect of the initialization strategies on the further optimization process in the SMBO framework using a Gaussian process (left) and a random forest (right) as a surrogate model is investigated. Our initialization strategy aPBI seems to be strictly better than PBI while aaPBI performs especially well for the random forest in the later phase.

6 Conclusion and Future Work

Predicting the similarity between data sets is an important topic for hyperparameter optimization since it allows to successfully transfer knowledge from past experiments to a new experiment. We have presented an easy way of achieving an adaptive initialization strategy by adding a new kind of landmark features. We have shown for two popular surrogate models that these new features improve over the same strategy without these features. Finally, we introduced a new idea that in contrast to the current methods considers that there is a limit of evaluations. It tries to exploit this knowledge by applying a guided exploitation at first that will lead to worse decisions for the short term but will deliver better results when the end of the initialization is reached. Unfortunately, the results for this method are not fully convincing but we believe that it can be a good idea to choose hyperparameter configurations in a smarter way but always assuming that the next hyperparameter configuration chosen is the last one.

In this work we were able to provide a more accurate prediction of the similarity between data sets and used this knowledge to improve the initialization. Since not only initialization strategies but also surrogate models rely on an exact similarity prediction, we plan to investigate the impact on these models. For example Yogatama and Mann [23] use a kernel that measures the distance between data sets by using the p-norm between meta-features of data sets only. A better distance function may help to improve the prediction and will help to improve the SMBO beyond the initialization.

Acknowledgments. The authors gratefully acknowledge the co-funding of their work by the German Research Foundation (Deutsche Forschungsgesellschaft) under grant SCHM 2583/6-1.

References

1. Bardenet, R., Brendel, M., Kégl, B., Sebag, M.: Collaborative hyperparameter tuning. In: Dasgupta, S., Mcallester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning (ICML-13). vol. 28, pp. 199–207. JMLR Workshop and Conference Proceedings (May 2013)
2. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 24, pp. 2546–2554. Curran Associates, Inc. (2011)
3. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011), software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
4. Coates, A., Lee, H., Ng, A.: An analysis of single-layer networks in unsupervised feature learning. In: Gordon, G., Dunson, D., Dudík, M. (eds.) Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings, vol. 15, pp. 215–223. JMLR W&CP (2011)
5. Feurer, M., Springenberg, J.T., Hutter, F.: Using meta-learning to initialize bayesian optimization of hyperparameters. In: ECAI workshop on Metalearning and Algorithm Selection (MetaSel). pp. 3–10 (2014)
6. Feurer, M., Springenberg, J.T., Hutter, F.: Initializing bayesian hyperparameter optimization via meta-learning. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. pp. 1128–1135 (2015)
7. 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), brazilian Symposium on Neural Networks (SBRN 2010) International Conference on Hybrid Artificial Intelligence Systems (HAIS 2010)
8. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proceedings of the 5th International Conference on Learning and Intelligent Optimization. pp. 507–523. LION'05, Springer-Verlag, Berlin, Heidelberg (2011)

9. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. of Global Optimization 13(4), 455–492 (Dec 1998)
10. Leite, R., Brazdil, P., Vanschoren, J.: Selecting classification algorithms with active testing. In: Perner, P. (ed.) Machine Learning and Data Mining in Pattern Recognition, Lecture Notes in Computer Science, vol. 7376, pp. 117–131. Springer Berlin Heidelberg (2012)
11. Michie, D., Spiegelhalter, D.J., Taylor, C.C., Campbell, J. (eds.): Machine Learning, Neural and Statistical Classification. Ellis Horwood, Upper Saddle River, NJ, USA (1994)
12. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. In: In Proceedings of the Seventeenth International Conference on Machine Learning. pp. 743–750. Morgan Kaufmann (2000)
13. Pinto, N., Doukhan, D., DiCarlo, J.J., Cox, D.D.: A high-throughput screening approach to discovering good forms of biologically inspired visual representation. PLoS Computational Biology 5(11), e1000579 (2009), PMID: 19956750
14. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press (2005)
15. Reif, M., Shafait, F., Dengel, A.: Meta-learning for evolutionary parameter optimization of classifiers. Machine Learning 87(3), 357–380 (2012)
16. Schilling, N., Wistuba, M., Drumond, L., Schmidt-Thieme, L.: Hyperparameter Optimization with Factorized Multilayer Perceptrons. In: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015 (2015)
17. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 25, pp. 2951–2959. Curran Associates, Inc. (2012)
18. Srinivas, N., Krause, A., Seeger, M., Kakade, S.M.: Gaussian process optimization in the bandit setting: No regret and experimental design. In: Fürnkranz, J., Joachims, T. (eds.) Proceedings of the 27th International Conference on Machine Learning (ICML-10). pp. 1015–1022. Omnipress (2010)
19. Swersky, K., Snoek, J., Adams, R.P.: Multi-task bayesian optimization. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 26, pp. 2004–2012. Curran Associates, Inc. (2013)
20. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 847–855. KDD '13, ACM, New York, NY, USA (2013)
21. Villemonteix, J., Vazquez, E., Walter, E.: An informational approach to the global optimization of expensive-to-evaluate functions. Journal of Global Optimization 44(4), 509–534 (2009)
22. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Hyperparameter Search Space Pruning - A New Component for Sequential Model-Based Hyperparameter Optimization. In: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015 (2015)
23. Yogatama, D., Mann, G.: Efficient transfer learning method for automatic hyperparameter tuning. In: International Conference on Artificial Intelligence and Statistics (AISTATS 2014) (2014)

Limitations of Using Constraint Set Utility in Semi-Supervised Clustering

Toon Van Craenendonck and Hendrik Blockeel

Department of Computer Science
KU Leuven

Abstract. Semi-supervised clustering algorithms allow the user to incorporate background knowledge into the clustering process. Often, this background knowledge is specified in the form of must-link (ML) and cannot-link (CL) constraints, indicating whether certain pairs of elements should be in the same cluster or not. Several traditional clustering algorithms have been adapted to operate in this setting. We compare some of these algorithms experimentally, and observe that their performances vary significantly, depending on the data set and constraints. We use two previously introduced constraint set utility measures, consistency and coherence, to help explain these differences. Motivated by the correlation between consistency and clustering performance, we also examine its use in algorithm selection. We find this consistency-based approach to be unsuccessful, and explain this result by observing that the previously found correlation between utility measures and clustering performance is only present when we look at results of different data sets jointly. This limits the use of these constraint set utility measures, as often we are interested in using them in the context of a particular data set.

Keywords: semi-supervised clustering, constraint set utility, algorithm selection

1 Introduction

Clustering is the task of grouping data into clusters, or groups of similar objects. Traditional unsupervised clustering algorithms only rely on information intrinsic to the data. In contrast, in semi-supervised clustering [2, 19, 20] the user can provide background knowledge to guide the algorithm towards better clusterings. Often, such background knowledge is given in the form of pairwise constraints, stating whether elements should be in the same cluster (must-link) or not (cannot-link). Semi-supervised extensions have been developed for most of the traditional clustering algorithms, such as K-means [19], DBSCAN [10, 14] and spectral clustering [12]. A user who wants to cluster a data set, and influence this clustering with pairwise constraints, has to select one of these algorithms. In addition, appropriate values have to be chosen for the algorithm hyperparameters. While these problems have received significant attention in the context of supervised learning [3,16], little work has been done for clustering, both unsupervised and semi-supervised. In this paper we focus on semi-supervised clustering, which is closer to the well-studied supervised setting. The contributions of this paper are (a) a comparison of a diverse set of semi-supervised clustering algorithms on several UCI data sets and (b) the exploration of the semi-supervised clustering algorithm selection strategy based on constraint-set utility measures suggested in [18].

2 Semi-supervised clustering algorithms

Semi-supervised clustering algorithms can be broadly divided into three categories: methods that use the constraints to adapt their similarity measure, methods that adapt the actual clustering procedure to satisfy the constraints, and hybrid algorithms that combine these two approaches. In the remainder of this section, we briefly discuss these three approaches and the algorithms that we use in our experiments. We consider these algorithms in combination with hyperparameter selection methods, as ultimately we are interested in mappings of the following form:

$$\Gamma(\mathcal{X}, \mathcal{M}, \mathcal{C}) = y \tag{1}$$

with $\mathcal{X} = \{x_i\}_{i=1}^n$ the data set, $\mathcal{M} = \{(x_i, x_j)\}$ a set of must-link constraints, $\mathcal{C} = \{(x_i, x_j)\}$ a set of cannot-link constraints and $y = \{c_1, c_2, \ldots, c_K\}$ s.t. $\cup_i c_i = \mathcal{X}$ (we only consider partitional clusterings). Γ encapsulates the clustering method as well as the hyperparameter selection procedure.

2.1 Methods that adapt the clustering algorithm directly

The first category consists of methods that alter the clustering procedure to satisfy constraints. One such algorithm is COP-KMeans [19], an adaptation of the traditional K-Means algorithm in which points are only assigned to clusters if the assignment does not result in a constraint violation. Since the introduction of COP-K-Means, several other variants of the original K-means algorithm have been developed. Semi-supervised extensions have also been developed for other types of clustering algorithms, including density-based methods [10, 14] and spectral clustering algorithms [9, 12]. In the remainder of this section we discuss two such methods that are used in the experiments.

FOSC-OpticsDend
In [5] Campello et al. introduce FOSC, a “Framework for Optimal Selection of Clusters” from clustering hierarchies. Given a local unsupervised clustering quality measure (one that can be computed for each cluster individually) and a set of constraints, FOSC determines a local cut of a given hierarchy that is optimal with respect to the quality measure and the constraint set. The clustering hierarchy on which FOSC operates can be provided by any hierachical

clustering algorithm. In our experiments, these hierarchies will be provided by OPTICS, a density-based clustering algorithm (we use the implementation provided in the ELKI environment [1]). It produces a reachability plot, from which a dendrogram is constructed using the algorithm by Sander et al. [15]. Campello et al. also experiment with this combination and find that this approach, which they call FOSC-OpticsDend, outperforms SSDBSCAN [10], a semi-supervised extension of DBSCAN. OPTICS requires setting *minPts*, but as this parameter is non-critical it is common to fix its value for all runs [5, 10]. As in [5], we set it to $minPts = 4$. Often several cuts of the dendrogram yield a partitional clustering that respects all constraints. In this case it is the unsupervised quality measure that will determine the chosen cut. As in [5], we use cluster lifetime for this purpose, which can be seen as the length along the dendrogram for which a cluster exists.

Constrainted 1-Spectral Clustering
Constrained 1-Spectral Clustering (COSC) [12] is an extension of spectral clustering to the semi-supervised setting.[1] Spectral clustering methods aim to partition a similarity graph such that edges within clusters have high weights and edges between clusters have low weights [17]. Several types of similarity graphs can be constructed from a set of data points. In our experiments we use a symmetric K-NN graph with local scaling as in [4], which avoids the need to select a scaling parameter [21]. Parameter K, indicating the number of neighbors, is not critical for the clustering result, and we set it to $K = 10$, as in [4]. The resulting graph can be represented by an affinity matrix consisting of pairwise similarities. Some semi-supervised spectral clustering algorithms incorporate must-link and cannot-link constraints by modifiying this matrix directly. Others, such as COSC, adapt the optimization objective of spectral clustering to incorporate constraints and propose alternative optimization procedures. COSC requires setting the number of clusters, k. In our experiments we run COSC for $k \in [2, 10]$, and select the clustering that violates the lowest number of constraints. If multiple solutions score equally on this measure, the clustering with the smallest number of clusters is chosen.

2.2 Methods based on metric-learning

The second type of methods does not alter the clustering algorithm directly, but modifies the underlying similarity measure. One of the first such methods was proposed by Xing et al. [20], who introduce an algorithm to learn a Mahalanobis distance measure that minimizes the distance between pairs involved in must-link constraints, while keeping pairs involved in cannot-link constraints far apart. Since the work of Xing et al., many others have focused on learning Mahalanobis metrics, which can be defined as

$$d_A(x, y) = \sqrt{(x - y)^T A(x - y)} \tag{2}$$

[1] Code is available at http://www.ml.uni-saarland.de/code/cosc/cosc.htm

The formula simplifies to the Euclidean distance if A is the identity matrix. If a diagonal matrix A is learned, this corresponds to feature weighting. Using a full matrix corresponds to feature generation, with the newly generated features being linear combinations of existing ones [2]. Using a Mahalanobis metric defined by A is equivalent to using the Euclidean distance in the transformed space obtained by muliplying by $A^{1/2}$, i.e. $X_{\text{transformed}} = A^{1/2}X$. Note that adapting an algorithm's similarity metric can only modify the bias of a clustering algorithm to some extent. For example, with a Mahalanobis distance K-means can find parallel ellipsoidal clusters instead of only spherical ones, but still no non-parallel ellipsoidal or non-convex clusters.

Information-Theoretic Metric Learning
In our experiments, we use the Information-Theoretic Metric Learning (ITML) algorithm [7][2], which has been shown to outperform the earlier algorithm by Xing et al. [7,8]. ITML requires setting one hyperparameter, γ, which determines the importance of satisfying the constraints. For each data set and constraint collection, we learn Mahalanobis matrices for $\gamma \in \{.01, .1, 1, 10\}$ (the values suggested in [7]), and select the best one as the one that results in the clustering that violates the lowest number of constraints. If multiple solutions score equally, we select the one that corresponds to the lowest value of γ. We learn a full metric matrix A, transform the data using this matrix, and construct clusterings using one of the following unsupervised algorithms:

- **K-Means**: We run the traditional K-Means algorithm for $k \in [2, 10]$, and experiment with two ways of selecting the "best" solution from the generated candidates: (a) the one violating the lowest number of constraints, as before, or (b) the one with the highest silhouette index, an unsupervised measure [13]. In the latter case the silhouette index is calculated in the transformed space.
- **Self-Tuning Spectral Clustering**: We also apply Self-Tuning Spectral Clustering [21][3] to the transformed data, which does not require any parameters to be set. The affinity matrix is constructed using local scaling (as with the COSC experiments), and the number of clusters is determined by examining the structure of the eigenvectors of the Laplacian.

2.3 Hybrid methods

The last group consists of hybrid methods, which combine metric learning with adapting the clustering procedure. A common representative of this type of algorithms is MPCK-Means (Metric Pairwise Constrained K-Means, [2][4]). Briefly, MPCK-Means iterates between 1) assigning elements to clusters, in this step the within-cluster sum of squares is minimized but also constraint satisfaction

[2] Code is available at http://www.cs.utexas.edu/~pjain/itml/
[3] Code is available at http://www.vision.caltech.edu/lihi/Demos/SelfTuningClustering.html
[4] Code is available at http://www.cs.utexas.edu/users/ml/risc/code/

is incorporated 2) updating the means, just like in the traditional K-Means algorithm and 3) re-estimating the metric, as to minimize the objective function. Bilenko et al. [2] define several variants of this scheme. For example, one can learn a separate metric for each cluster, or a global one. These metrics can either be defined by full matrices, which corresponds to doing feature generation, or diagonal ones, which corresponds to feature weighting. In our experiments we use MPCK-Means with a single diagonal metric matrix. As before, we vary the number of clusters $k \in [2, 10]$ for each problem instance, and select the best solution as either the one that violates the lowest number of constraints, or the one that has the highest silhouette score.

3 Experiments

In this section we compare the performance of the previously discussed algorithms on several UCI data sets.

3.1 Overview of algorithms and experimental methodology

In total, we compare 7 clusterers:

- **FOSC-OpticsDend**
- **COSC**: Constrained 1-Spectral Clustering, selecting k based on the constraints
- **K-Means-ITML-NumSat**: K-Means on ITML transformed data, selecting k and γ based on the constraints
- **K-Means-ITML-Silhouette**: K-Means on ITML transformed data, selecting k and γ based on the silhouette index
- **Self-Tuning-Spectral-ITML-NumSat**: Spectral clustering, selecting γ based on the constraints
- **MPCK-Means-NumSat**: MPCK-Means, selecting k based on the constraints
- **MPCK-Means-Silhouette**: MPCK-Means, selecting k based on the silhouette index

For each data set, we also show the results of four unsupervised variants of the algorithms:

- **K-Means-Unsup**: the traditional K-Means algorithm, selecting k based on the silhouette index
- **Self-Tuning-Spectral-Unsup**: Self-Tuning-Spectral clustering on the original data
- **FOSC-OpticsDend-Unsup**: extraction of a partitional clustering from the OPTICS dendrogram based on the unsupervised cluster lifetime measure
- **MPCK-Means-Unsup**: running MPCK-Means without constraints, which is different from the traditional K-Means algorithm, as MPCK-Means also performs unsupervised metric learning

For each constraint set size in $\{25, 100, 400, 1600\}$, we generate 10 constraint sets in the following way:

1. We select 70% of the data set randomly
2. From this subset, pairs of elements are randomly selected and a must-link or cannot-link constraint is added depending on whether the selected elements belong to the same class or not.

Clusterings are evaluated using the adjusted Rand Index (ARI) [11], which is calculated using only the elements of the 30% of the data that were not selected in step one. The first step ensures that there will be enough elements (i.e. at least 30% of the data) to evaluate the clusterings on. For larger numbers of constraints and relatively small data sets, it might otherwise occur that all elements are involved in constraints, leaving none for evaluation.

3.2 Results and discussion

Figures 1 and 2 show the results of our experiments for the *iris* and *wine* data sets. Similar figures for the *dermatology, column, glass* and *ecoli* data sets are added in the appendix. For the *iris* data set, three algorithms show consistent average improvement when constraints are added: FOSC-OpticsDend, K-Means-ITML-NumSat and MPCK-Means-NumSat. In general, for *iris* the performance seems to improve with constraints, or does at least not decline drastically (e.g. for K-Means-ITML-Silhouette the performance remains largely constant).

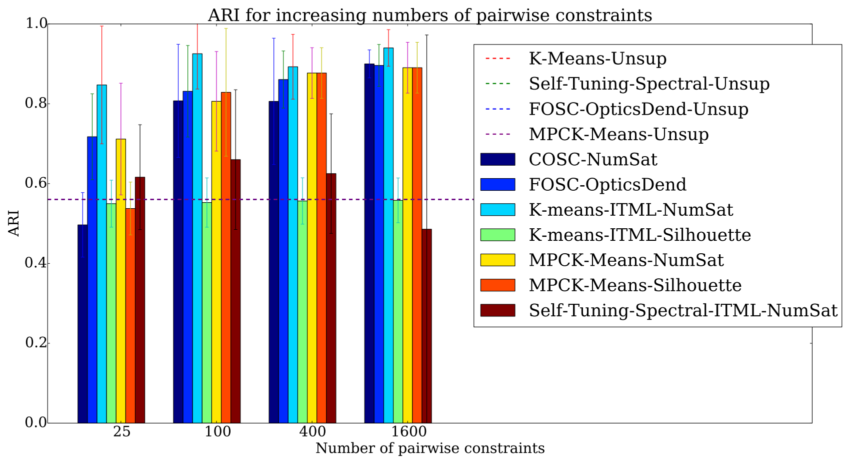


Fig. 1. Scores for the *iris* data set (150 instances, 4 features, 3 classes). All unsupervised algorithms produced the same clustering for this data set (only the MPCK-Means line is visible in the plot, as they all overlap).

Figure 2 shows that the results are quite different for the *wine* data set. Most clustering algorithms already attain a high ARI score without any constraints, and in many cases adding constraints leads to an average decrease in performance. For this data set, MCPK-Means-Silhouette and Self-Tuning-Spectral-ITML seem to be the most robust in this aspect.

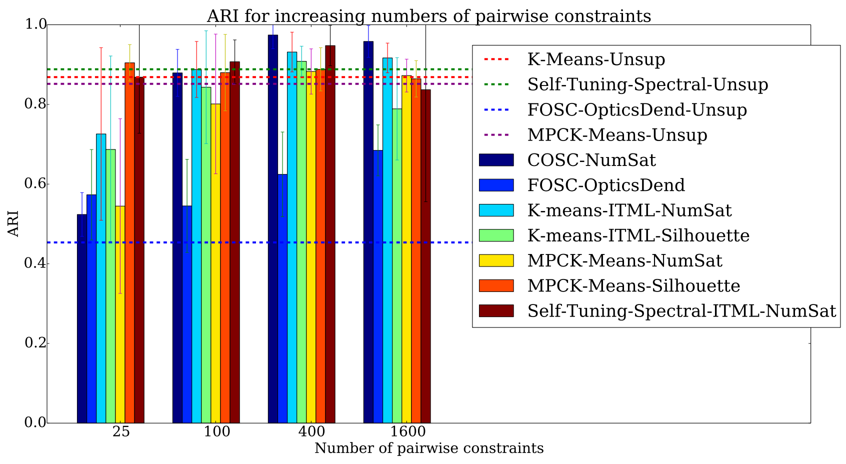


Fig. 2. Scores for the *wine* data set (178 instances, 13 features, 3 classes)

In general, from the results on these two data sets and the four others that are added in the appendix, it is clear that no single semi-supervised clustering algorithm outperforms all others in all scenarios. The preferable option for a certain task depends on the data set, the size of the constraint set, and even the specific constraint set under consideration. Often, the preferrable option even seems to be to not make use of the constraints at all. This is quite counter-intuitive, as one would expect constraints to "point the algorithm in the right direction". Davidson et al. [6, 18] also observe this potentially negative effect of adding constraints. They point out that, while performance improves *on average* when more constraints are provided, individual constraint sets can have a detrimental effect. In our experiments this happened frequently: clustering performance decreased when constraints were added for 45% of the considered runs (with a run we indicate a data set, particular constraint set and clusterer combination).

Constraint set consistency and coherence
The observation that adding constraints can result in decreased performance is of course crucial, as we are mostly interested in the performance gain that we can obtain with one particular constraint set. To provide insight into this behaviour, Davidson et al. propose two measures that characterize the utility of constraint sets [6]:

- **Consistency** is defined by the number of constraints that are satisfied by the clustering produced without any constraints. It is a property of both the constraint set and the algorithm producing the clustering.
- **Coherence** measures the amount of agreement between constraints. In [18], Wagstaff et al. define two variants of this measure: distance coherence and direction coherence. We use the former one in our experiments, which is defined as the fraction of constraint pairs (one ML and one CL constraint) for which the distance between the points in the ML constraint is greater than the distance between the points in the CL constraint [18]. This property only depends on the data set and the constraints.

Wagstaff et al. show that consistency and coherence are strongly correlated with clustering performance: if a consistent and coherent constraint set is provided, performance is likely to increase, whereas inconsistent and incoherent constraint sets have an adverse effect. They study these properties in the context of MPCK-Means and variants thereof. Here, we verify whether this correlation also holds for the diverse set of clustering algorithms used in our experiments. We perform an analysis similar to the one in [18]. If the property holds, it may provide insight into whether a particular constraint set should be used or not in combination with a clustering algorithm.

Figure 3 shows the relation between the constraint utility measures and ARI, illustrating that also in our experiments these are strongly correlated (Pearson coefficient of 0.66 for consistency, Pearson coefficient of 0.75 for coherence). Table 1 shows the correlation coefficients for the separate algorithms, providing insight into the sensitivities of the different algorithms to constraint set inconsistency and incoherence. For example, the correlation between consistency and performance is larger for the K-Means-ITML algorithms than for the MPCK-Means algorithms, meaning that consistency might be a better predictor for performance for the former ones. The correlation between consistency and performance is lowest for COSC, but this can be explained by the fact that we used the outcome of the Self-Tuning spectral clustering algorithm as an unsupervised baseline for COSC, as otherwise we would not be able to select the number of clusters k, which COSC requires.

Consistency-based algorithm selection
Given the correlation between consistency and clustering performance, Wagstaff et al. [18] suggest the simple algorithm selection strategy of choosing the one with the highest consistency, given a data set and constraints. We explore the

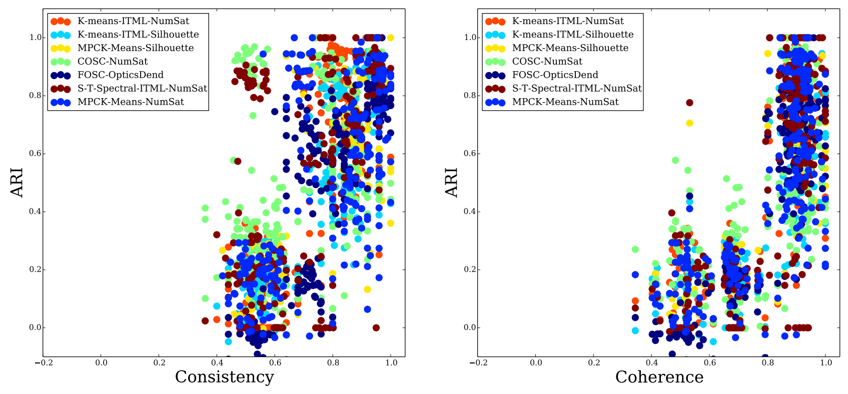


Fig. 3. (left) Consistency vs. ARI, Pearson correlation coefficient = 0.66 , (right) Coherence vs. ARI, Pearson correlation coefficient = 0.75

Algorithm	**Consistency vs. ARI**	**Coherence vs. ARI**
FOSC-OpticsDend	0.81	0.88
COSC-NumSat	0.45	0.62
K-Means-ITML-NumSat	0.83	0.82
K-Means-ITML-Silhouette	0.82	0.76
Self-Tuning Spectral-ITML-NumSat	0.56	0.69
MPCK-Means-NumSat	0.63	0.72
MPCK-Means-Silhouette	0.67	0.80

Table 1. Pearson correlation coefficients of consistency and coherence vs. ARI for each algorithm

effectiveness of this strategy by applying it to our clustering experiments. For each problem instance, which consists of a data set and constraints, we compute the relative score for each algorithm as its ARI for this instance divided by the largest ARI obtained for that problem instance by all algorithms. We compute the average relative score for each algorithm over all 1680 problem instances (6 data sets, 7 algorithms, 4 constraint set sizes and 10 constraint sets per size). These averages are shown in Table 2. The table also shows the average relative score that is obtained by using the consistency-based algorithm selection strategy. It is clear that, despite the observed correlation between consistency and clustering performance, the algorithm selection strategy does not perform well, as we would be better of by simply picking the (on average) best algorithm for each problem instance, which is MPCK-Means-Silhouette.

Algorithm	Avg. rel. score
MPCK-Means-Sil	0.814
COSC-NumSat	0.798
K-Means-ITML-NumSat	0.770
ST-Spectral-ITML-NumSat	0.725
Consistency-based AS	**0.713**
MPCK-Means-NumSat	0.712
K-Means-ITML-Sil	0.695
FOSC-OpticsDend	0.638

Table 2. Average relative scores for each algorithm, and also for the consistency-based algorithm selection strategy (AS)

Data set	Consistency vs. ARI	Coherence vs. ARI
iris	0.20	-0.14
ecoli	-0.06	0.04
column_2C	-0.17	-0.19
glass	-0.12	-0.10
dermatology	0.16	-0.07
wine	0.04	-0.07

Table 3. Pearson correlation coefficients of consistency and coherence vs. ARI for each data set

Why consistency-based algorithm selection does not work

In this section we explain the unsatisfactory results of the consistency-based algorithm selection strategy, and in doing so identify an important property of the consistency and coherence measures. We observe that, while these measures correlate strongly with performance if we look at all problem instances combined, this correlation disappears when we consider the data sets separately. This can be seen in Figure 4, which is similar to Figure 3 but colored by data set instead of algorithm. This visual observation is confirmed by looking at the correlation coefficients for each data set separately, shown in Table 3. These results indicate that, while consistency and coherence can be indicative of the difficulty of clustering a particular data set given constraints, these measures cannot be used to decide between clustering algorithms for a given data set.

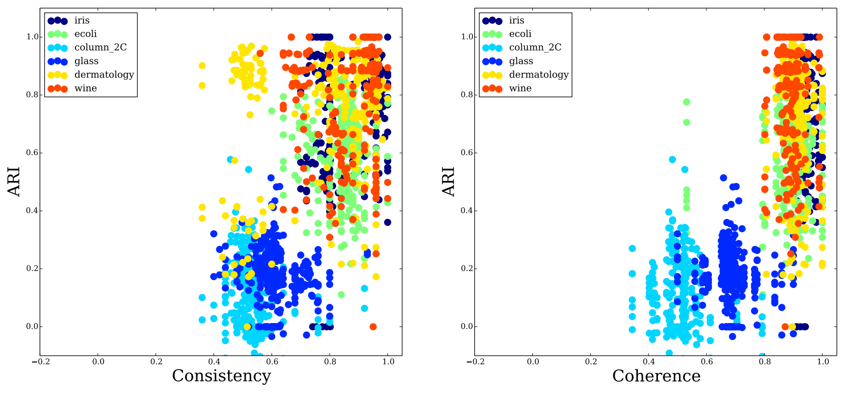


Fig. 4. (left) Consistency vs. ARI (right) Coherence vs. ARI (same as Figure 3, but colored by data set)

Furthermore, for a consistency-based algorithm selection strategy to be succesful, the different clusterers should produce significantly different clusterings when no constraints are given. We verify whether this is actually the case for the algorithms and data sets that are considered in the experiments, by comparing the unsupervised clusterings using the adjusted Rand index (ARI). While ARI is mostly used to compare a produced clustering to a ground truth one, it can more generally be used to measure the similarity between any two clusterings. When no constraints are given, all algorithms produce the same solution for the *iris* data set (pairwise ARI scores of 1.0). Also for the *wine* and *ecoli* data sets all clusterings are quite similar (ARIs > 0.9), except for the ones generated by FOSC-OpticsDend (ARIs < 0.5). The clusterings generated for the *dermatology*, *glass* and *column* data sets are more diverse (with an average pairwise ARI of 0.49, not taking into account the similarities between the K-Means and MPCK-Means solutions, which are more similar with an average pairwise ARI of 0.95). These observations complement the lack of correlation between consistency and ARI in explaining the failure of the consistency-based algorithm selection strategy for the *iris*, *ecoli* and *wine* data sets.

4 Conclusions

Semi-supervised clustering is a popular research topic, and its usefulness has been demonstrated in several practical applications. Most major clustering algorithms have been extended to incorporate domain knowledge, often in the form of must-link and cannot-link constraints. It has been frequently demonstrated that, on average, performance increases when constraints are added. However, it is known that individual constraint sets can harm perfomance. We have experimented with a diverse set of semi-supervised clustering algorithms, and have observed that this is indeed often the case. Given data to cluster and a set of constraints, a user then has to determine which semi-supervised clustering algorithm to use, if any. Previous work proposed to use constraint set consistency and coherence for this purpose, two constraint set utility measures that were shown to correlate strongly with clustering performance. We have experimented with such consistency-based algorithm selection, but found it to be unsuccessful. For some data sets, these results can be explained by the similarities between the clusterings that are produced when no constraints are given. If these similarities are high, comparing the corresponding consistency values is not informative. More importantly, we also explain the unsatisfactory results of the selection strategy for data sets for which the clusterings produced without constraints are more diverse. We do this by showing that the utility measures only correlate strongly with clustering performance if we look at problem instances from several data sets combined, and that this correlation disappears when we consider individual data sets. These results severely restrict the use of these measures in semi-supervised clustering, as practitioners are mainly interested in applying them in the context of a particular data set.

Acknowledgements

Toon Van Craenendonck is supported by the Agency for Innovation by Science and Technology in Flanders (IWT).

References

[1] Elke Achtert, Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. Interactive data mining with 3d-parallel-coordinate-trees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1009–1012, 2013.

[2] Mikhail Bilenko, Sugato Basu, and Raymond J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 11–, New York, NY, USA, 2004. ACM.

[3] Pavel B. Brazdil, Carlos Soares, and Joaquim Pinto da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

[4] Thomas Bühler and Matthias Hein. Spectral clustering based on the graph p-laplacian. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 81–88, New York, NY, USA, 2009. ACM.

[5] Ricardo J.G.B. Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. A framework for semi-supervised and unsupervised optimal extraction of clusters from hierarchies. *Data Mining and Knowledge Discovery*, 27(3):344–371, 2013.

[6] Ian Davidson, Kiri L. Wagstaff, and Sugato Basu. Measuring constraint-set utility for partitional clustering algorithms. In *Knowledge Discovery in Databases: PKDD 2006*, volume 4213 of *Lecture Notes in Computer Science*, pages 115–126. Springer Berlin Heidelberg, 2006.

[7] Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 209–216, New York, NY, USA, 2007. ACM.

[8] Amir Globerson and Sam T. Roweis. Metric learning by collapsing classes. In Y. Weiss, B. Schölkopf, and J.C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 451–458. MIT Press, 2006.

[9] Sepandar D. Kamvar, Dan Klein, and Christopher D. Manning. Spectral learning. In *In IJCAI*, pages 561–566, 2003.

[10] Levi Lelis and Jörg Sander. Semi-supervised density-based clustering. In *ICDM '09. Ninth IEEE International Conference on Data Mining, 2009*, pages 842–847, Dec 2009.

[11] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[12] Syama S. Rangapuram and Matthias Hein. Constrained 1-spectral clustering. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (AISTATS-12)*, volume 22, pages 1143–1151, 2012.

[13] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[14] Carlos Ruiz, Myra Spiliopoulou, and Ernestina Menasalvas. C-dbscan: Density-based clustering with constraints. In *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, volume 4482 of *Lecture Notes in Computer Science*, pages 216–223. Springer Berlin Heidelberg, 2007.

[15] Jörg Sander, Xuejie Qin, Zhiyong Lu, Nan Niu, and Alex Kovarsky. Automatic extraction of clusters from hierarchical clustering representations. In *Proceedings of the 7th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD '03, pages 75–87, Berlin, Heidelberg, 2003. Springer-Verlag.

[16] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, page 847–855, 2013.

[17] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[18] Kiri Wagstaff, Sugato Basu, and Ian Davidson. When is constrained clustering beneficial, and why? In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, 2006.

[19] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 577–584, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[20] Eric P. Xing, Michael I. Jordan, Stuart Russell, and Andrew Y. Ng. Distance metric learning with application to clustering with side-information. In *Advances in neural information processing systems*, pages 505–512, 2002.

[21] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. *Advances in Neural Information Processing Systems 17*, 2:1601–1608, 2004.

A Performance comparison for UCI data sets

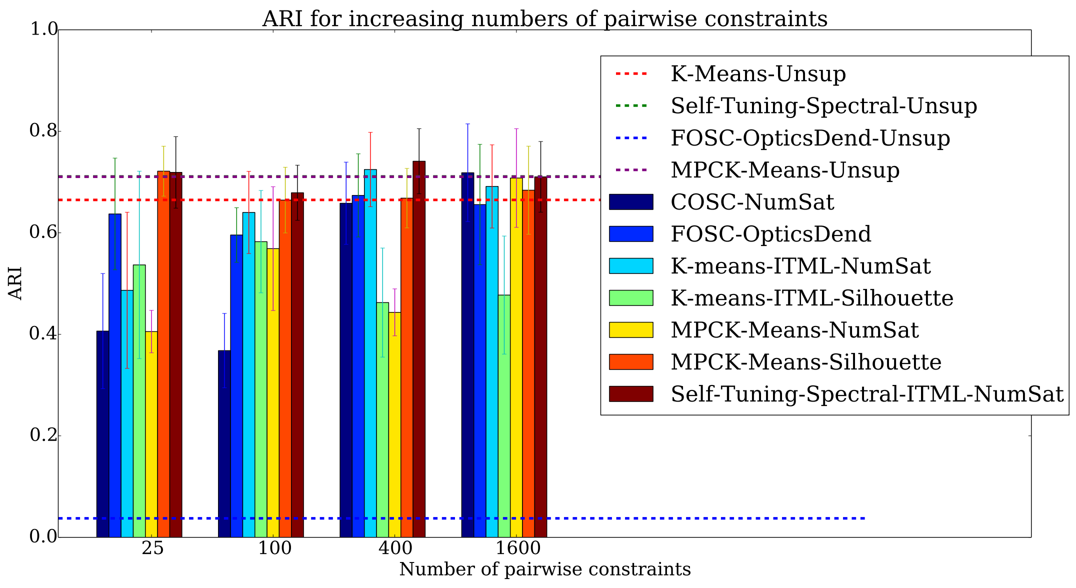


Fig. 5. Scores for the *ecoli* data set (336 instances, 7 features, 8 classes)

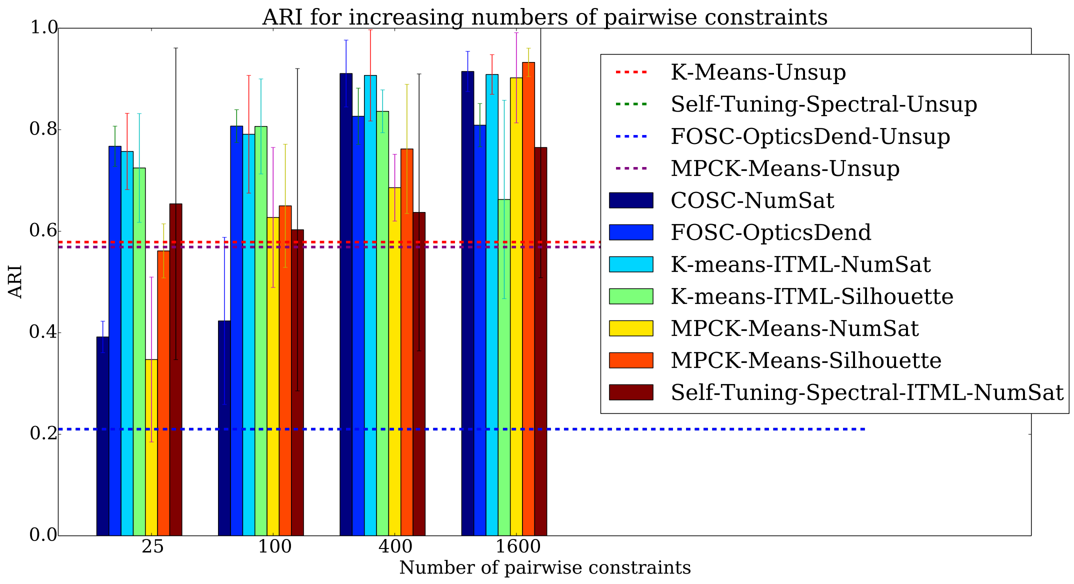


Fig. 6. Scores for the *dermatology* data set (366 instances, 34 attributes, 6 classes)

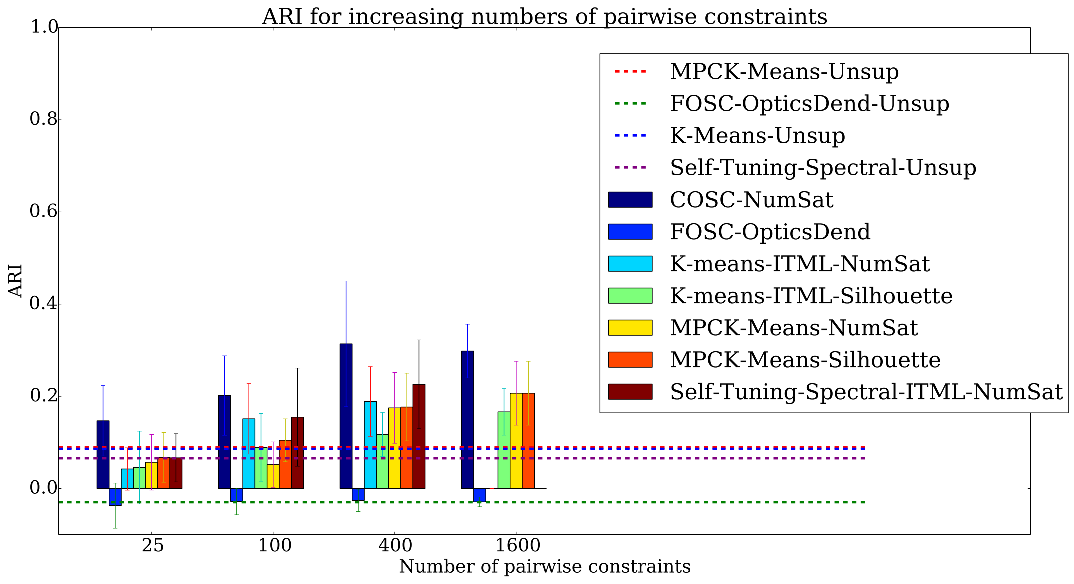


Fig. 7. Scores for the *column* data set (310 instances, 6 attributes, 2 classes)

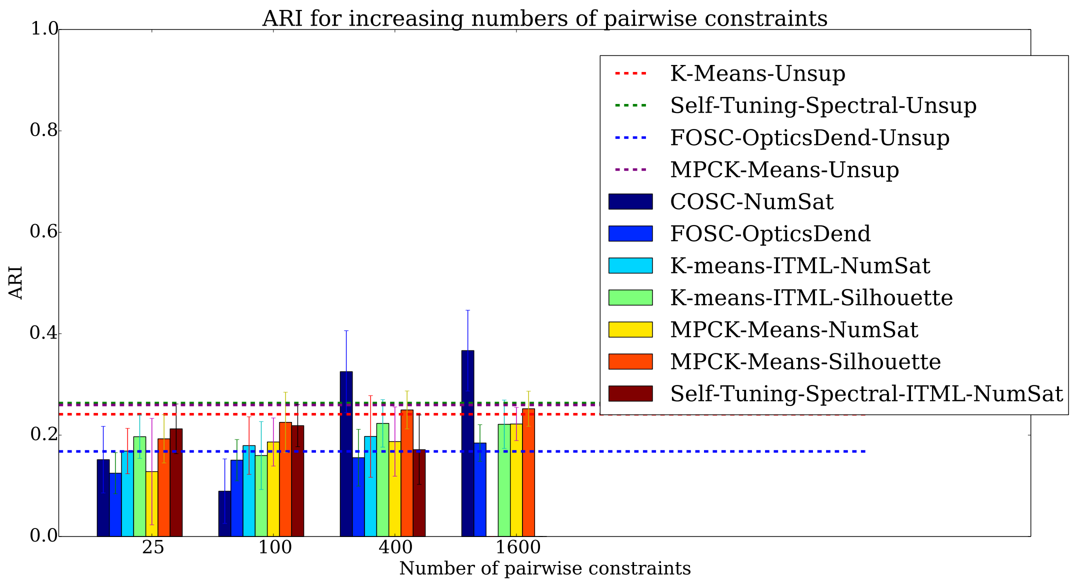


Fig. 8. Scores for the *glass* data set (214 instances, 9 attributes, 6 classes)

Dealing with overlapping clustering: a constraint-based approach to algorithm selection

Antoine Adam and Hendrik Blockeel

KULeuven, Department of Computer Science, Celestijnenlaan 200A, 3001 Leuven, Belgium

Abstract. When confronted to a clustering problem, one has to choose which algorithm to run. Building a system that automatically chooses an algorithm for a given task is the algorithm selection problem. Unlike the well-studied task of classification, clustering algorithm selection cannot rely on labels to choose which algorithm to use. However, in the context of constraint-based clustering, we argue that using constraints can help in the algorithm selection process. We introduce CBOvalue, a measure based on must-link and cannot-link constraints that quantifies the overlapping in a dataset. We demonstrate its usefulness by choosing between two clustering algorithm, EM and spectral clustering. This simple method shows an average performance increase, demonstrating the potential of using constraints in clustering algorithm selection.

Keywords: clustering, algorithm selection, constraints

1 Introduction

Constraints have been used to improve clustering performance by incorporating some background knowledge in a clustering problem. In a study on constraint-based clustering, Davidson et al. [4] show that using constraints can sometimes decrease this performance. They introduce the notion of coherence between constraints, and show that the more incoherent a constraint set is, the more chance it has to decrease clustering performance. Two constraints are called incoherent if they carry information that is a priori contradictory. For instance, in figure 1, the must-link constraint (in blue) implies that the left area must be clustered with the right area, while the cannot-link constraint (in red) says the opposite.

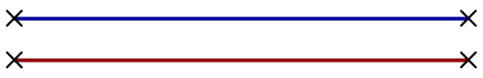


Fig. 1. Incoherent constraints

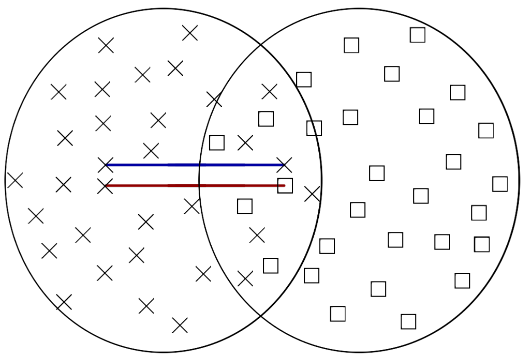


Fig. 2. Overlapping clusters can lead to incoherent constraints

Beyond the possible presence of noise in the data, a problem that we will ignore in this paper, we identified other circumstances where such incoherent constraints can appear: overlapping clusters, as shown in figure 2. Overlapping clusters is an issue that is not often tackled by clustering algorithms. Some state-of-the-art algorithms such as spectral clustering [18], which is very good at discovering arbitrary shaped clusters, will fail in the presence of overlapping. On the contrary, the EM [6] algorithm has a bias towards spherical clusters but can handle overlapping quite well as we show in section 3. As an example, we artificially created a cross dataset, see figure 3, where two clusters overlap in the middle. With a few random initialisations, EM is always able to find the correct clusters, while spectral clustering always fails. What is more, the model built by the EM algorithm incorporates the uncertainty about the cluster assignments in the overlapping area.

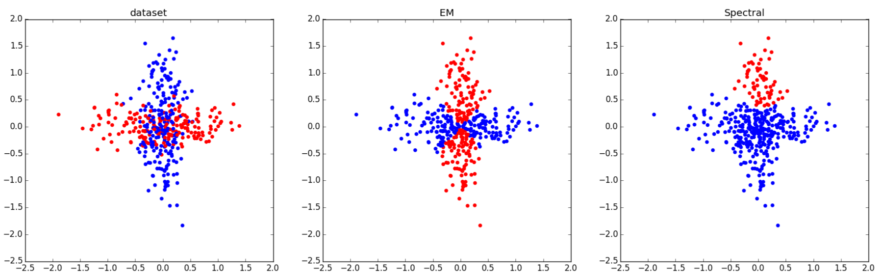


Fig. 3. Cross dataset. Colours are from left to right: the generated clusters, clusters found by EM, clusters found by spectral clustering.

This toy example illustrates the variety of clustering algorithms: different algorithms will produce different partitionings. Moreover, in a real clustering problem, we cannot say one of these partitionings is better as we do not know the true labels. Even on the same dataset, two users might be interested in a different partitioning of the data. Only if some constraints are specified can we build a system that selects the algorithm best fitting a user requirements.

In this paper, we present some preliminary results in this direction. We introduce the CBOvalue to measure the overlapping from must-link and cannot-link constraints. We use this measure as a meta-feature in a basic meta-learning system that chooses between EM and spectral clustering. The goal of the paper is not to present an advanced meta-learning system, but to show the potential of using constraints in clustering algorithm selection.

The content of the paper is organised as follows. In section 2, we present some related work. In section 3, we define more concretely what we call overlapping and show through experiments that EM performs better than spectral clustering when it occurs. In section 4, we introduce the CBOvalue, an overlapping measure based on equivalence constraints. In section 5, we show that a simple algorithm selection method based on this measure increases clustering performance. In section 6, we draw some conclusions and leads for future work.

2 Related work

2.1 Constraint-based clustering

Clustering is the unsupervised learning task of identifying groups of similar instances in a dataset. Although these groups are initially unknown, some information can be available as to what the desired solution is. This information takes the form of constraints on the resulting clusters. These constraints can be provided to the clustering algorithm to guide the search towards a more desirable solution. We then talk about constraint-based, constrained, or semi-supervised clustering.

Constraints can be defined on different levels. On a cluster level, one can ask for clusters that are balanced in size, or that have a maximum diameter in space. On an instance level, one might know some partial labelling of the data. A well-used type of constraints are must-link and cannot-link constraints, also called equivalence constraints. These are pair-wise constraints which state that two instances must be or cannot be in the same cluster.

Multiple methods have been developed to use these constraints, some of which are mentioned below. A metric can be learnt that complies with the constraints [2]. The constraints can be used in the algorithm for the cluster assignment in a hard [19] or soft way [13], [15], [20]. Some hybrid algorithms use constraints for both metric learning and clustering [3], [9]. Other approaches include constraints in general solver methods like constraint programming [7] or integer linear programming [1].

2.2 Algorithm selection for clustering

Not much research has been conducted on algorithm selection for clustering. Existing methods usually predict the ranking of clustering algorithms [5], [16],

[14] [8]. The meta-features used are unsupervised and/or domain-specific. None of these approaches use constraints.

3 Overlapping clustering

3.1 Overlapping

We talk about overlapping when two clusters are present in the same area of the data space. It is a local property of a dataset as it happens in some parts only. Several reasons can produce overlapping clusters: there might be noise in the data, the features may not capture all the necessary information to clearly separate clusters or the overlap may be inherent to the processes that produced the data. It is a problem for algorithms that assume a clear separation of the clusters, or at least a zone of lower density points. As already mentioned for the cross dataset, spectral clustering cannot cluster it correctly. With a few random initialisations, EM always finds the right partition and what is more, the model includes that the cluster assignment is uncertain in the overlapping area.

3.2 Rvalue

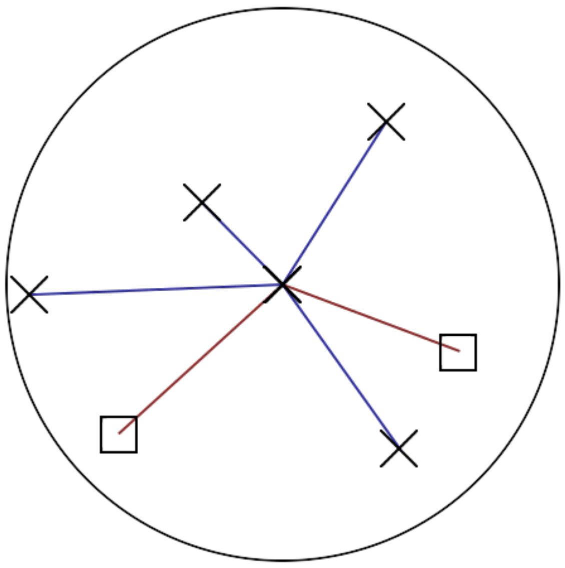


Fig. 4. Rvalue, $k = 6$, $\theta = 1$.

To numerically measure overlapping, we use the Rvalue introduced by [11]. The principle is illustrated in figure 4. For each object of a dataset, the labels of its k neighbours are checked. If stricly more than θ are from another class, it is counted as overlapped. The Rvalue of the dataset is the proportion of overlapped objects. It is a local measure that requires two parameters, k and θ. In all our experiments, we use $k = 6$ and $\theta = 1$, i.e. we allow one neighbour to be of another class. This limits the false overlapping measurement when two clusters are next to each other but not overlapping. As an example, the cross dataset figure 3 has an Rvalue of 0.41 which means that 41% of the data points are overlapping. Figure 5 shows the distribution of the Rvalue for 14 datasets from the UCI repository, namely *iris*, *glass*, *ionosphere*, *wine*, *vertebral*, *ecoli*, *seeds*, *students*, *yeast*, *zoo*, *breast cancer wisconsin*, *mammographic*, *banknote*, *haberman*. Each feature of these datasets is normalised to an average of 0 and standard value of 1 and the metric used is the euclidean distance. This normalisation and metric are kept throughout all experiments. We can see from this figure that overlapping is not uncommon in real world datasets.

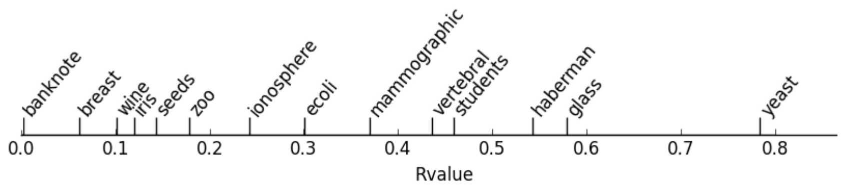


Fig. 5. Rvalue of UCI datasets, $k = 6$, $\theta = 1$.

3.3 Clustering performance

We now compare two clustering algorithms, namely EM [6] and spectral clustering [18] that we will call SC. EM is run 10 times with randomly initialised gaussians while SC is run with various parameter settings. The right number of clusters was given to both algorithms, whose performances were measured in terms of ARI (Ajusted Rand Index, [10]) and AMI (Ajusted Mutual Information, [17]). The best run was kept for comparison, as we want to compare the potential of each algorithm. On figure 6, we show the ARI of EM (in red) and SC(in blue) on the same datasets, as well as on 22 artificially made datasets.

As expected, both algorithms lose performance when overlapping increases. However, EM decreases more slowly than SC, as it is presented in table 1. These results show that EM can handle overlapping better than SC.

4 Detecting overlapping from constraints

In a clustering problem, the Rvalue cannot be directly computed as the labels are unknown. However, a user might have some partial knowledge of the clustering

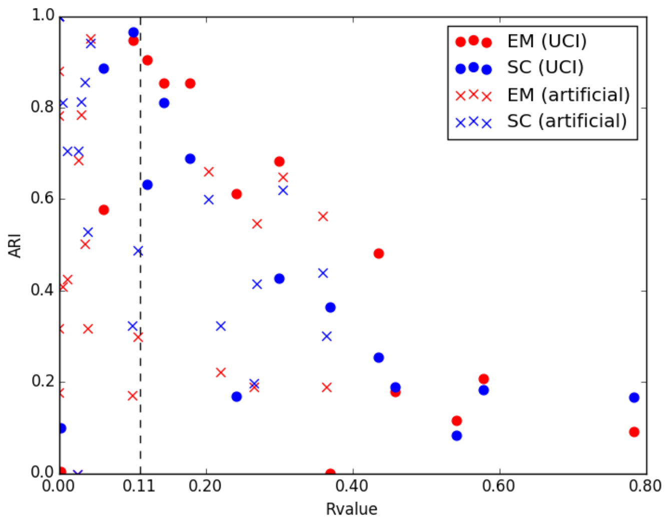


Fig. 6. Clustering performance vs Rvalue.

UCI	EM	SC
$Rvalue < 0.11$	0.509	**0.65**
$Rvalue > 0.11$	**0.452**	0.36
ALL	EM	SC
$Rvalue < 0.11$	0.511	**0.728**
$Rvalue > 0.11$	**0.443**	0.38

Table 1. Average clustering performance measured with ARI.

he is looking for. This is the setting of semi-supervised clustering, presented in section 2. We now present our method to detect overlapping based on these constraints. Like the Rvalue, it is based on the idea that overlapping is a local property.

4.1 CBOvalue: Constraint-Based Overlapping value

Overlapping translates in two cases in terms of equivalence constraints: one short cannot-link constraint or two close parallel must-link and cannot-link constraints.

CLOvalue: Cannot-Link Overlapping value. A short cannot-link means that in a close neighbourhood, two points are in two distinct clusters. Figure 7 illustrates the principle. For a cannot-link constraint *cl* between points x_1 and x_2, we define

$$CLOvalue(cl) = exp(-\frac{1}{2}(\frac{dist(x_1, x_2)}{max(\epsilon_1, \epsilon_2)})^p)$$

where ϵ_i the distance between x_i and it's k^{th} nearest neighbour.

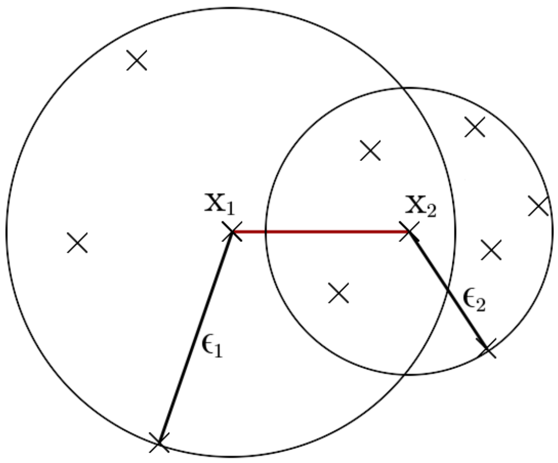


Fig. 7. CLOvalue for k=6.

Unlike for the Rvalue, we take a soft approach with the exponential because experience showed that for a limited number of constraints a hard approach was too sensitive to noise. However, the usual $p = 2$ of a gaussian turned out to be a bit too soft hence we also experiment with $p = 4$. This provides a soft neighbourhood with still a major drop at the epsilon. Using $k = 6$ produced relatively low values, so we also consider a broader neighbourhood by raising k to 10.

With CL the set of cannot-link constraints, we define

$$CLOvalue = \frac{1}{|CL|} \sum_{cl \in CL} CLOvalue(cl)$$

MLCLOvalue: Must-Link and Cannot-link Overlapping value. The case of two close must-link and cannot-link constraints was shown figure 2. Figure 8 illustrates the principle of the measure. It is defined for a cannot-link constraint cl between points x_1 and x_2 and a must-link constraint ml between two other points. We name these points x_3 and x_4 such that $dist(x_1, x_3) + dist(x_2, x_4) \leq dist(x_1, x_4) + dist(x_2, x_3)$. This ensures that we pair up neighbour points together. For instance in figure 8, we want to compare x_3 with x_1 and not x_2. We then define

$$MLCLOvalue(ml, cl) = \frac{exp(-\frac{1}{2}(\frac{d_1+d_2}{max(\epsilon_1,\epsilon_3)+max(\epsilon_2,\epsilon_4)})^p)}{2}$$

where ϵ_i the distance between x_i and its k^{th} neighbour, $d_1 = dist(x_1, x_3)$ and $d_2 = dist(x_2, x_4)$.

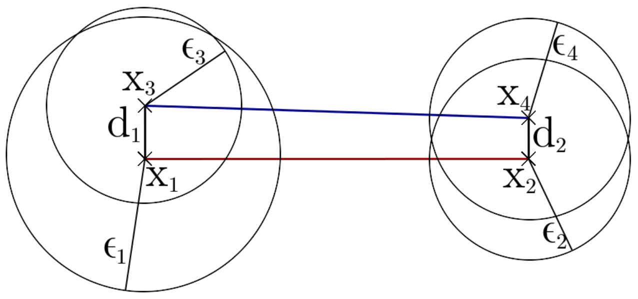


Fig. 8. MLCLOvalue.

If CL is the set of cannot-link constraints and ML the set of must-link constraints, we define

$$MLCLOvalue = \frac{1}{|CL| \times |ML|} \sum_{cl \in CL, ml \in ML} MLCLOvalue(ml, cl)$$

$$CBOvalue = \frac{CLOvalue + MLCLOvalue}{2}$$

For each dataset, we randomly generated 100 equivalence constraints from the real classes and we computed the CBO value for $k \in \{6, 10\}$. Figure 9 plots the CBO-value versus the Rvalue. The correlation is not perfect, but is enough for the algorithm selection as we will see in the next section.

5 Algorithm selection

Now that we have an overlapping measure from the constraints, we can build a system that picks which algorithm to use based on this measure. For each parameter setting, we put a threshold at the optimal position in terms of ARI. For example on figure 10 where the CBOvalue is computed with k=6 and p=4, we put a threshold at 0.011. If the CBOvalue is bigger, we use EM, otherwise we use SC. We call this method AS for Algorithm Selection. To provide an upper bound, we compute the performance of an oracle that would always pick the algorithm with highest performance. Table 2 compares the average performance of EM, SC, AS, and oracle. To visualise the improvement of the algorithm selection method, we plot on figure 11 the loss of each method for the UCI datasets. The

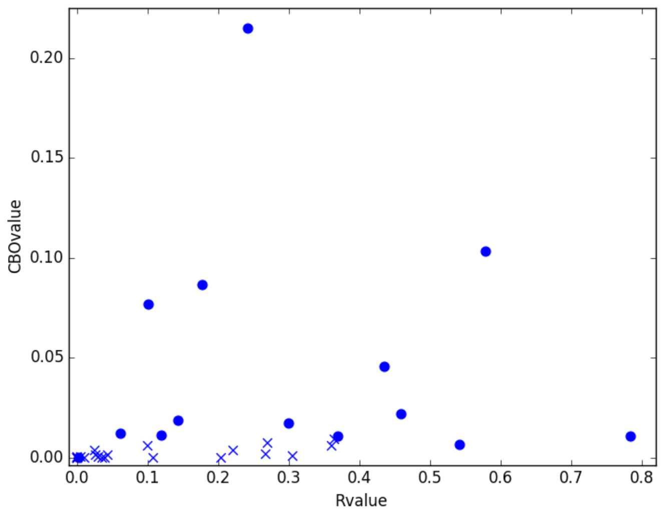


Fig. 9. CBOvalue with k=6 and p=4 vs Rvalue with k=6 and th=1.

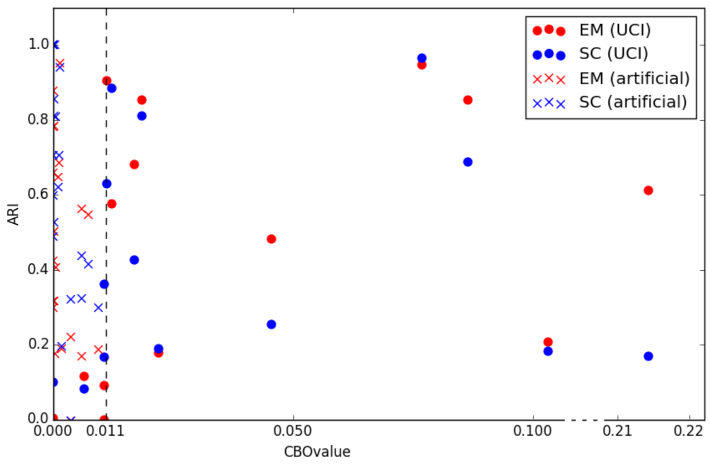


Fig. 10. Clustering performance vs CBOvalue for k=6 and p=4.

		EM	SC	AS				oracle
				k=6		k=10		
				p=2	p=4	p=2	p=4	
UCI	*ARI*	0.464	0.422	0.497	0.5	0.497	**0.522**	0.526
	AMI	0.481	0.47	0.508	**0.514**	0.487	0.487	0.534
ALL	*ARI*	0.477	0.554	0.58	0.585	0.58	**0.593**	0.605
	AMI	0.522	0.614	0.626	0.631	0.626	**0.638**	0.642

Table 2. Average clustering performance of EM, SC (Spectral Clustering), AS (a selection between the two based on the CBOvalue for several parameters), and oracle(an ideal system that would keep the best between the EM and SC).

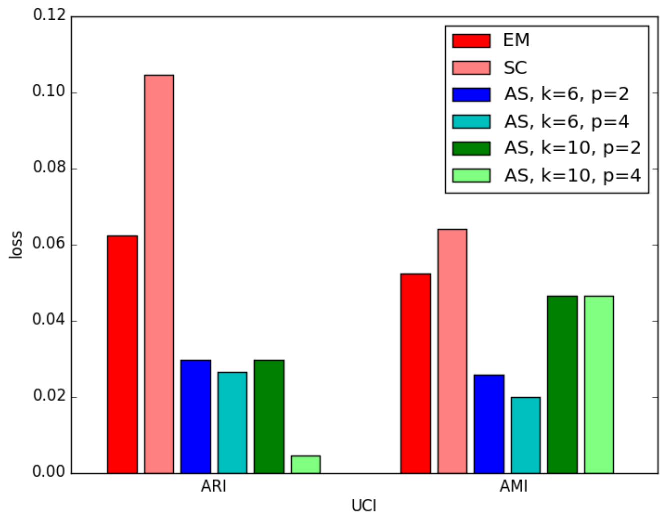


Fig. 11. Performance loss to oracle.

loss is simply the difference between the average performance of a method and the oracle average performance.

In all experiments, AS performs on average better than EM and SP, in terms of ARI or AMI. The meta-learning system used here is very simplistic: we consider only one meta-feature and two clustering algorithms. However, the goal here is not so much to build a very elaborate system, but to show the potential of using constraints in clustering algorithm selection. We see here that despite the simplicity of the selection process, the Constraint-Based Overlapping value increases the average clustering performance.

6 Conclusion

In this paper, we introduced the CBOvalue to measure the amount of overlapping in a dataset based on must-link and cannot-link constraints. On the basis that the EM algorithm handles overlapping better than spectral clustering, we select which algorithm to run depending on the CBOvalue. This simple algorithm selection system shows an increase in average performance compared to the two algorithms. Through this promising result, we demonstrate the potential of using constraints in clustering algorithm selection.

More in-depth research on the CBOvalue still has to be conducted to answer remaining questions: How robust is this measure? How sensitive is it with respect to the constraint set? How does high dimensionality affect it? We should also integrate the CBOvalue in a more complex meta-learner that uses constrained and unconstrained features.

The approach we used can be generalised as follows. A first step is to identify the strong and weak point of different algorithms, in our case the fact that EM can produce overlapping clusters. In a second step, a measure is engineered based on constraints and/or data to discriminate situations where algorithms perform differently. Finally, these measures can be used as meta-features in an algorithm selection system which can then make use of the strong points of each algorithm. Despite the remaining questions on the CBOvalue, we believe the encouraging results promote the validity of this approach for the problem of clustering algorithm selection.

Acknowledgements This work is funded by the KU Leuven Research Fund (project IDO/10/012). Experiments have been implemented in python with the scikit-learn package [12].

References

1. Behrouz Babaki, Tias Guns, and Siegfried Nijssen. Constrained clustering using column generation. In *Integration of AI and OR Techniques in Constraint Programming*, pages 438–454. Springer, 2014.
2. Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research*, 6(6):937–965, 2005.
3. Mikhail Bilenko, Sugato Basu, and Raymond J Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 11. ACM, 2004.
4. Ian Davidson, Kiri L Wagstaff, and Sugato Basu. *Measuring constraint-set utility for partitional clustering algorithms.* Springer, 2006.
5. Marcilio CP De Souto, Ricardo BC Prudencio, Rodrigo GF Soares, Rodrigo GF De Araujo, Ivan G Costa, Teresa B Ludermir, Alexander Schliep, et al. Ranking and selecting clustering algorithms using a meta-learning approach. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 3729–3735. IEEE, 2008.

6. Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.
7. Khanh-Chuong Duong, Christel Vrain, et al. Constrained clustering by constraint programming. *Artificial Intelligence*, 2015.
8. Daniel Gomes Ferrari and Leandro Nunes de Castro. Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods. *Information Sciences*, 301:181–194, 2015.
9. Pan Hu, Celine Vens, Bart Verstrynge, and Hendrik Blockeel. Generalizing from example clusters. In *Discovery Science*, pages 64–78. Springer, 2013.
10. Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.
11. Sejong Oh. A new dataset evaluation method based on category overlap. *Computers in Biology and Medicine*, 41(2):115–122, 2011.
12. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
13. Dan Pelleg and Dorit Baras. K-means with large and noisy constraint sets. In *Machine Learning: ECML 2007*, pages 674–682. Springer, 2007.
14. Ricardo BC Prudêncio, Marcilio CP De Souto, and Teresa B Ludermir. Selecting machine learning algorithms using the ranking meta-learning approach. In *Meta-Learning in Computational Intelligence*, pages 225–243. Springer, 2011.
15. Carlos Ruiz, Myra Spiliopoulou, and Ernestina Menasalvas. C-dbscan: Density-based clustering with constraints. In *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, pages 216–223. Springer, 2007.
16. Rodrigo GF Soares, Teresa B Ludermir, and Francisco AT De Carvalho. An analysis of meta-learning techniques for ranking clustering algorithms applied to artificial data. In *Artificial Neural Networks–ICANN 2009*, pages 131–140. Springer, 2009.
17. Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research*, 11:2837–2854, 2010.
18. Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.
19. Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. Constrained k-means clustering with background knowledge. In *ICML*, volume 1, pages 577–584, 2001.
20. Xiang Wang and Ian Davidson. Flexible constrained spectral clustering. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 563–572. ACM, 2010.

Algorithm Selection via Meta-learning and Sample-based Active Testing

Salisu Mamman Abdulrahman[1], Pavel Brazdil[1,2], Jan N. van Rijn[3], and Joaquin Vanschoren[4]

[1] LIAAD-INESC Tec, Porto, Portugal
salisu.abdul@gmail.com
[2] FEP, Univiversity of Porto, Porto, Portugal
pbrazdil@inesctec.pt
[3] Leiden University, Leiden, Netherlands
j.n.van.rijn@liacs.leidenuniv.nl
[4] Eindhoven University of Technology, Eindhoven, Netherlands
j.vanschoren@tue.nl

Abstract. Identifying the best machine learning algorithm for a given problem continues to be an active area of research. In this paper we present a new method which exploits both meta-level information acquired in past experiments and active testing, an algorithm selection strategy. Active testing attempts to iteratively identify an algorithm whose performance will most likely exceed the performance of previously tried algorithms. The novel method described in this paper uses tests on smaller data sample to rank the most promising candidates, thus optimizing the schedule of experiments to be carried out. The experimental results show that this approach leads to considerably faster algorithm selection.

Keywords: Algorithm selection, Meta-learning, Active testing, Algorithm Ranking

1 Introduction

A large number of data mining algorithms exist, rooted in the fields of machine learning, statistics, pattern recognition, artificial intelligence, and database systems, which are used to perform different data analysis tasks on large volumes of data. The task to recommend the most suitable algorithms has thus become rather challenging. Moreover, the problem is exacerbated by the fact that it is necessary to consider different combinations of parameter settings, or the constituents of composite methods such as ensembles.

The algorithm selection problem, originally described by Rice [17], has attracted a great deal of attention, as it endeavours to select and apply the best or near best algorithm(s) for a given task [4, 19]. The algorithm selection problem can be cast as a *learning* problem: the aim is to learn a model that captures the relationship between the properties of the datasets, or meta-data, and the algorithms, in particular their performance. This model can then be used to predict the most suitable algorithm for a given new dataset.

This paper presents a new method, which builds on ranking approaches for algorithm selection [2, 3] in that it exploits meta-level information acquired in past experiments. Moreover, the proposed method combines this information with an algorithm selection strategy known as *active testing* [10, 11]. The aim of active testing is to iteratively select and evaluate a candidate algorithm whose performance will most likely exceed the performance of previously tested algorithms.

The method described in this paper differs from earlier approaches in various aspects. First, two methods presented earlier [10, 11] were combined, and adapted to optimize a multi-objective measure, called A3R, that combines predictive accuracy and time. The first method is known as an average ranking method, as it calculates an average ranking for all algorithms over all datasets. The upgrade here consists of using A3R as the measure to optimize, rather than simply accuracy. The second method uses fast sample-based tests instead of full cross-validation (CV) tests to identify the most promising candidates by evaluating them on small data samples. Again, we will use A3R, instead of accuracy, in the process of identifying the best competitor of the current alternative.

Fast sample-based tests allow selecting a good algorithm in less time, but are less reliable. This needs to be taken into account in the design of the method. One further contribution of this work is to show how the sample-based tests should be integrated with the elaboration of the ranking.

Finally, the experimental results are presented in the form of loss-time curves, where time is represented on a log scale. This representation is very useful in the evaluation of rankings representing schedules, as was shown in recent findings [18].

The remainder of this paper is organized as follows. In the next section we present an overview of work in related areas. Section 3 is dedicated to the description of our new method of active testing. We explain how it relates to earlier proposals. Section 4 presents the empirical evaluation of the newly proposed method. The final section presents conclusions and future work.

2 Related Work

In this paper we are addressing a particular case of the algorithm selection problem [17], oriented towards the selection of classification algorithms. Various researchers addressed this problem in the course of the last 25 years. One very common approach that could now be considered as "*the classical approach*" uses a set of measures to characterize datasets and establish their relationship to algorithm performance. This information is often referred to as *meta-data*.

The meta-data typically includes a set of simple measures, statistical measures, information-theoretic measures and/or the performance of simple algorithms referred to as landmarkers [4, 14, 19]. The aim is to obtain a model that characterizes the relationship between the given meta-data and the performance of algorithms evaluated on these datasets. This model can then be used to predict the most suitable algorithm for a given new dataset, or alternatively, provide a ranking of algorithms, ordered by their suitability for the task at hand. Many studies conclude that ranking is in fact better, as it enables the user to iterative test the top candidates to identify the algorithms most suitable in practice. This strategy is sometimes referred to as the Top-N strategy [4].

The Top-N strategy has the disadvantage that it is unable to leverage what is learned from previous evaluations. For instance, if the top algorithm performs worse than expected, this may tell us something about the given dataset that can be used to update the ranking. Indeed, very similar algorithms are now also likely to perform worse than expected. This led researchers to investigate an alternative testing strategy, known as active testing [11]. This strategy intelligently selects the most useful cross-validation tests using the concept of relative landmarkers [7]. These landmarkers estimate the *relative probability* that a particular algorithm will outperform the current best candidate.

The method presented in [10] exploits partial learning curves created on small samples of the data, as suggested by the authors of [15]. It makes pairwise algorithm comparisons and represents the results in the form of a partially ordered ranking. The method can be evaluated by comparing the predicted partial order of algorithms to the actual partial order, representing the golden standard obtained by exhaustive testing. An extension to this method was presented in [11]. It relies on earlier performed cross-validation tests to calculate relative landmarkers. The authors showed that this led to better results than traditional top-N ranking strategies.

The novel aspect of the method described in this paper is the use of relatively fast sample-based tests to reorder the algorithms in the top-N ranking. Using tests on small data samples represents a trade-off in that they lead to less accurate performance estimation. Indeed, the tests carried out on small data samples are less reliable and thus the best algorithms may sometimes not be identified. Hence, it is difficult to say a priori which variant would be better. This is one of the motivations to investigate this issue.

Active testing is somewhat related to experiment design [6] and also to active learning. However, there has been relatively little work on active learning for algorithm selection. One notable exception is [12], who use the notion of *Expected Loss Optimization* (ELO). Another application in this area is [8], whose aim was to identify the most interesting substances for drug screening using a minimum number of tests. In these experiments, the authors have focused on the top-10 substances. Several different strategies were considered and evaluated.

Another notable active learning approach to meta-learning was presented in [16], where the authors use active learning to support the selection on informative examples. A prototype was implemented using the k-NN algorithm as a meta-learner and a certainty-based method was used for active learning. The prototype was evaluated in two different case studies, and the results obtained by the active learning method were in general better than a random method for selecting meta-examples.

In this paper, we attribute particular importance to the tests on the new dataset. Our aim is to propose a way that minimizes the time before the best (or near best) algorithm is identified.

3 Active Sample-based Testing (ASbT) Method

We propose a new method, called *Active Sample-based Testing* (ASbT), which is detailed in Algorithm 1. It exploits meta-level information acquired from past experiments. This information includes the performance evaluations of various machine learning algorithms, on prior datasets, over multiple performance measures (e.g., accuracy, AUC, time). Moreover, we also use *data samples* or various sizes to evaluate algorithms. Further details concerning the meta-level information are provided below.

This information enables us to construct an average ranking of algorithms (line 4). The term *average ranking* is used here to indicate that it is averaged over all previously seen datasets. The average ranking can be followed by the user to identify the best performing algorithm, i.e., by performing a cross-validation test on all algorithms in the order of the ranking. This strategy was referred to as the Top-N strategy [2] and constitutes a baseline for other more competitive approaches.

Our method also exploits the active testing strategy [11]. However, in the ASbT approach, we use faster sample-based tests to identify competitive algorithms (line 5). In our experiments, these algorithms are also evaluated on the full dataset in order to construct a loss curve. To evaluate this method, experiments are carried out in a leave-one-out fashion. One dataset at a time is left out to evaluate our approach. The complete method, including evaluation, is summarized in Algorithm 1. The following sections discuss key parts of the method in more detail.

Algorithm 1 Active sample-based testing (ASbT)

```
1: Identify datasets $D_s$ and algorithms
2: for all $D_i$ in $D_s$ do
3:     {Leave-one-out cycle; $D_i$ represents $D_{new}$}
4:     Construct the average ranking
5:     Carry-out sample-based active testing and evaluate recommended algorithms
6:     Return a loss curve
7: end for
8: Aggregate the loss curves for all $D_i$ in $D_s$
Return: Mean loss curve
```

3.1 Constructing the average ranking

The challenge here is to order the algorithms in the form of a top-N ranking. The underlying assumption is that the rankings obtained on past problems will transfer to new problems. Among many popular ranking criteria we find, for instance, *average ranks*, *success rates* and *significant wins* [3, 5, 10].

In this work we use average ranks, inspired by Friedman's *M statistic* [13]. For each dataset, the algorithms are ordered according to the performance measure chosen (e.g., predictive accuracy) and assigned ranks accordingly. The best algorithm is assigned rank 1, the runner-up is assigned rank 2, and so on. Let r_i^j be the rank of algorithm i on dataset j. The average rank for each algorithm is obtained using

$$r_i = \left(\sum_{j=1}^{D} r_i^j \right) \div D \tag{1}$$

where D is the number of datasets. The final ranking is obtained by ordering the average ranks and assigning ranks to the individual algorithms accordingly.

The average ranking is used here both as a baseline against which we can compare other methods, and as an input to the proposed algorithm (line 5), as discussed below.

3.2 Active Testing on Data Samples

Do we really need a full cross-validation test to establish a good sequence of algorithms to test? In this section we discuss a novel method that also follows an active testing strategy, but uses sample-based tests instead of full cross-validation tests. Hence, instead of deciding whether a candidate algorithm a_c is better than the currently best algorithm a_{best} using a full cross-validation test, we perform a test on a smaller sample of the new dataset. This is motivated by the fact that a sample-based test is much faster than a full cross-validation test.

However, as the sample-based test only yields a proxy for the actual performance, it may identify an apparent winner that differs from the one selected by the full cross-validation test. Hence, if a candidate algorithm beats the currently best algorithm on the sample-based test, we additionally carry out an evaluation using a full cross-validation test. This approach differs from [11] in three key aspects, i.e., the use of small data samples as proxies, the use of the A3R criterium that combines accuracy and run time (see below), and the strategy that we occasionally run a full cross-validation test. Algorithm 2 shows this method in detail.

3.3 Evaluating the returned ranking

To evaluate the quality of the returned ranking, the whole process is repeated in a leave-one-out fashion for all datasets D_i belonging to the set D_s (line 2 of Algorithm 1). For each generated algorithm ranking we generate a loss curve that

Algorithm 2 Active testing with sample-based tests

```
Require: Datasets $D_i$ (representing $D_{new}$), $D_s$; Average ranking for $D_i$
 1: Use the average ranking of $D_i$ to identify the topmost algorithm and initialize $a_{best}$;
    Obtain the performance of $a_{best}$ on dataset $D_i$ using a full CV test;
 2: Find the most promising competitor $a_c$ of $a_{best}$ using relative landmarkers
    (previous test results on datasets)
 3: Obtain the performance of $a_c$ on new dataset using a sample-based test;
    Record the accuracy and time of the test to compute A3R;
 4: Compare the performance of $a_c$ with $a_{best}$;
    use the winner as the current best algorithm $a_{best}$;
    If the current best algorithm has changed in the previous step, do:
    Carry out evaluation of the new $a_{best}$ using a full CV test
 5: Repeat the whole process starting with step 2 until reaching a stopping criterion
Return: Loss curve
```

plots the loss in accuracy (see below) versus the time spent on the evaluation of the algorithms using full cross-validation tests. Finally, all individual loss curves are aggregated into a *mean loss curve*.

Loss-time curves In a typical loss curve, the x-axis represents the number of cross-validation tests and the y-axis shows the *loss* in accuracy with respect to the ideal ranking. Loss is defined to be the difference in accuracy between the current best evaluated classifier and the actual best classifier [11]. As tests proceed following the Top-N strategy, the loss either maintains its value, or decreases when the newly selected algorithm improved upon the previously selected algorithms. Each test represents the result of a full cross-validation evaluation on one dataset. However, some algorithms are much slower learners than others (sometimes by orders of magnitude), and these simple loss curves do not capture this.

This is why, in this article, we take into account the actual time required to evaluate each algorithm and update the loss curve accordingly. We will refer to this type of curve as a *loss versus time curve*, or *loss-time curve* for short.

As train/test times include both very small and very large numbers, it is natural to use the logarithm of the time, instead of the actual time. This has the effect that the same time intervals appear to be shorter as we shift further on along the time axis. This graphical representation is advantageous if we wish to give more importance to the initial items in the ranking.

To evaluate our algorithm (ASbT), we need to carry out tests on different datasets in a leave-one-out fashion and construct the mean loss-time curve by aggregating the individual loss-time curves.

3.4 A3R: combining Accuracy and Run time

In many situations, we have a preference for algorithms that are fast and also achieve high accuracy. However, the question is whether such a preference would lead to better loss-time curves. To investigate this, we have adopted a multi-objective evaluation measure, A3R, described in [1], that combines both accuracy and time. The measure A3R is defined as:

$$A3R^{d_i}_{a_{ref},a_j} = \frac{\dfrac{SR^{d_i}_{a_j}}{SR^{d_i}_{a_{ref}}}}{\sqrt[N]{T^{d_i}_{a_j}/T^{d_i}_{a_{ref}}}} \tag{2}$$

Here, $SR^{d_i}_{a_j}$ and $SR^{d_i}_{a_{ref}}$ represent the *success rates* (accuracies) of algorithms a_j and a_{ref} on dataset d_i, where a_{ref} represents a given *reference algorithm.* Similarly, $T^{d_i}_{a_j}$ and $T^{d_i}_{a_{ref}}$ represent the run times of the algorithms, in seconds. To trade-off the importance of time, A3R includes the N^{th} root parameter. This is motivated by the observation that run times vary much more than accuracies. It is not uncommon that one particular algorithm is three orders of magnitude slower (or faster) than another one. Obviously, we do not want the time ratios to dominate the equation. If we take the N^{th} root of the run time, we will get a number that goes to 1 in the limit.

For instance, if we used $N = 256$, an algorithm that is 1,000 times slower would yield a denominator of 1.027. It would thus be equivalent to the faster reference algorithm only if its accuracy was 2.7% higher than the reference algorithm. Table 1 shows how a ratio of 1,000 (one algorithm is 1,000 times slower than the reference algorithm) is reduced for increasing values of N. As N gets higher, the time is given less and less importance.

Table 1. Effect of varying N on time ratio of 1000

C	$N = 2^C$	$1{,}000^{(1/N)}$	C	$N = 2^C$	$1{,}000^{(1/N)}$
0	1	1000.000	6	64	1.114
1	2	31.623	7	128	1.055
2	4	5.623	8	256	1.027
3	8	2.371	9	512	1.013
4	16	1.539	10	1,024	1.006
5	32	1.241	20	1,048,576	1.000

The performance measure A3R can be used to rank a given set of algorithms on a particular dataset in a similar way than using simply accuracy. Hence, the average rank method described earlier can easily be upgraded to generate a time-aware average ranking: the *A3R-based average ranking.* The method of active testing with sample-based tests can also be easily updated. We note that the algorithm shown in Algorithm 2 mentions performance on lines 3 and 4. If we use A3R instead of accuracy, we obtain a multi-objective variant of the active testing method.

Obviously, we can expect somewhat different results for each particular choice of N. Which value of N will lead to the best results in loss-time space? Another important question is whether the use of A3R is beneficial when compared to the approach that only uses accuracy. The answers to these questions will be answered empirically in the next section.

4 Experiments

This section describes the empirical evaluation of the proposed method. We have constructed a dataset from evaluation results retrieved from OpenML [20], a collaborative science platform for machine learning. This dataset contains the results of 53 parameterized learning algorithms from the Weka workbench [9] on 39 datasets[1]. Section 4.1 evaluates our proposed method independently of the A3R criterion, thus solely using accuracy. Section 4.2 explores the further impact of the A3R criterion, combining accuracy and run time.

4.1 Active Sample-based Testing using Accuracy

Figure 1 presents our results in the form of loss-time curves, with time represented on a log scale. First, it shows the loss curve of the average ranking method (AvgRank-ACC) which uses the Top-N strategy when carrying out the tests.

The figure also includes the loss-time curve of one predecessor method that uses active testing with full cross-validation tests (ATCV-ACC). This is ASbT in the extreme, using the full dataset instead of a smaller sample. In the earlier paper [11] this method was referred as AT0.

Finally, the figure shows the loss-time curves of three variants of the active sample-based testing method presented here. These variants use a different sample size for the sample-based testing. Here we use the notation ASbT-4-ACC to refer to a variant of active sample-based testing that uses accuracy on sample number 4 when conducting tests. Similarly, ASbT-5-ACC uses sample number 5 in testing. The sample sizes grow geometrically following the equation $2^{(5.5+0.5*n)}$, where n represents the sample number. So, the sizes of the first few samples, after rounding, are 64, 91, 128, 181, 256, 362 examples. Hence, sample number 4 includes 181 data points.

In Figure 1, $\log_{10}$ time is used on the x-axis. The values on the x-axis represent the time that has elapsed as we repeatedly conduct cross-validation tests.

What can we conclude from this figure? The first observation we can make is that all the three variants of the sample-based active testing methods compete quite well with the (more complex) predecessor method (ATCV-ACC) that uses full CV test to identify the best competitor. These two approaches beat the simple average ranking method (AvgRank-ACC). In Section 4.2, we investigate the effect of adopting the A3R measure instead of accuracy in the selection algorithms problem.

[1] Full details: `http://www.openml.org/project/tag/ActiveTestingSamples/u/1`

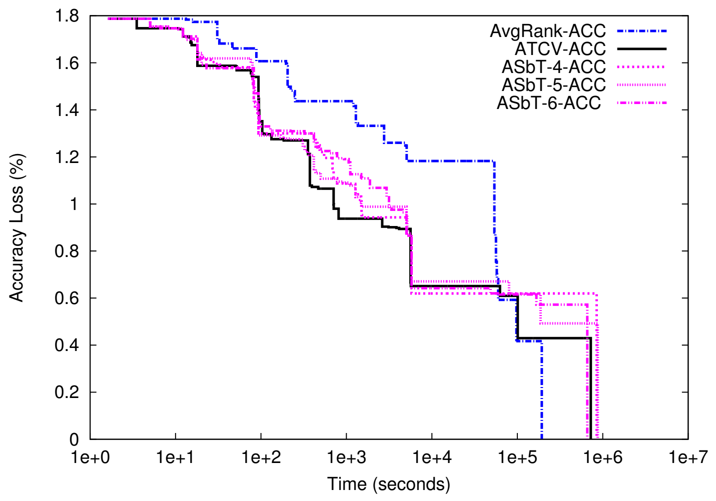


Fig. 1. Mean loss-time curves for 3 variants of ASbT (using sample number 4, 5 and 6), ACTV-ACC and the average ranking method (AvgRank-ACC).

Early stopping criteria The main active testing algorithm (ATCV) described in Algorithm 2 includes a stopping criterion: when the probability that a new candidate algorithm a_c will win over the currently best algorithm a_{best} becomes too small, it will not be considered. Since ASbT derives from ATCV, we can use the same criterion, and we have empirically evaluated the effect using ASbT-4. To do this, we need to define a minimal improvement in the sum of the relative landmarkers, which is a parameter of the method. This was set to 0.02 (2%) and the effect was that the algorithm stopped after about half of the algorithms were tested. The result of this study shows that it is only slightly better, but overall not much different. It does manage to skip less promising algorithms early on.

4.2 Active Sample-based Testing using Accuracy and Time

Our next aim in this paper is to analyze the effects of adopting A3R as a performance measure within the methods presented earlier. The first objective is to analyze the effects on the average ranking method. The second objective is to examine the effects of adopting A3R within the active testing strategy that uses sample-based tests. In each case we seek the best solution by tuning the value of parameter N of A3R. The third objective is to compare the proposed method with one predecessor method that generates a revised ranking first before conducting evaluation.

Figure 2 shows the first set of results. Note that the upgraded average ranking method (AvgRank-A3R) has an excellent performance when compared to the average ranking method that uses only accuracy as a measure (AvgRank-ACC).

Hence, the new average ranking method represents a new useful algorithm that can be exploited in practice for the selection of algorithms.

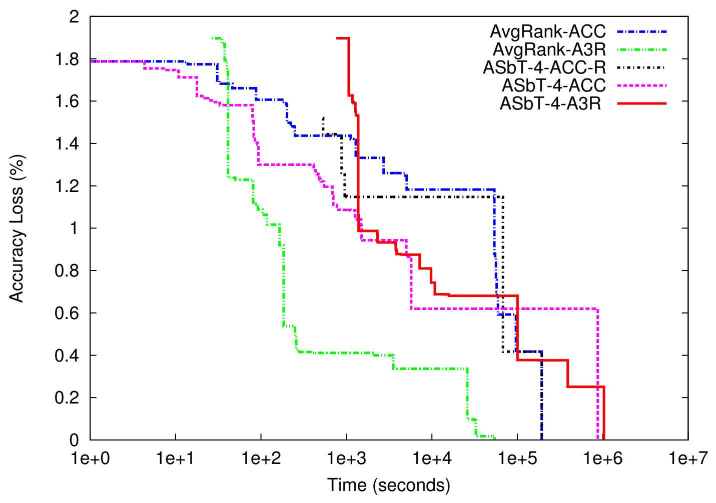


Fig. 2. Loss-time curves for sample-based active testing and average ranking methods, accuracy-based and A3R-based versions ($N = 256$).

Next, let us analyze the effects of adopting A3R within the active testing strategy that uses sample-based tests. As the experiments described in the previous section have shown that there is not much difference between using sample number 4, 5 or 6, we have simply opted for the variant that uses the smallest sample number 4 (ASbT-4-A3R).

We compare our new sample-based active testing method ASbT-4-A3R that uses A3R in the selection process with ASbT-4-ACC which represents the method that uses accuracy instead. The results show that our method ASbT-4-A3R does not beat the previous method (ASbT-4-ACC).

For completeness, we also include another variant, ASbT-4-ACC-R. This variant works by first re-ranking all algorithms by evaluating them on a small sample of the new dataset. Hence, it starts later than the other algorithms, on average 500 seconds later, because it needs to run this process first. Then, in a second phase, it conducts the final evaluation using full CV tests. Merging the two phases, as is done in ASbT-4-ACC, results in lower losses sooner.

The average ranking method (AvgRank-A3R), which represents a much simpler method than the sample-based variant, achieves surprisingly good results which warrants further investigation.

One possible explanation is that our meta-dataset is perhaps too simple, including 53 algorithms with default parameters. In future work we will investigate the effect of adopting the A3R measure in the ASbT method, while using more algorithms (in the order of hundreds).

5 Conclusions

We have described two novel algorithm selection methods. The first method uses fast sample-based tests to identify the most promising candidates, as its aim is to intelligently select the next algorithm to test on the new dataset. The second method is a rather simple one. It calculates an average ranking for all algorithms, but uses A3R as the measure to rank on. The novelty here lies in the use of A3R, instead of just accuracy.

The experimental results are presented in the form of loss-time curves. Since exhaustive testing is not always practically feasible, we focus on the behavior of the algorithm within smaller time interval. This is achieved by using a loss curves with $\log_{10}$ of time on the x-axis. This representation stresses the losses at the beginning of the curve, corresponding to the initial tests.

The results show that both methods lead to considerable time savings, when compared to the previous approaches that exploited just accuracy. The experimental results suggest that the new version of the average ranking method represents a very good alternative that could be used in practice.

The next challenge then is to explore ways on how to improve the ASbT method that uses A3R in selecting the best competitor for active testing method by using more algorithms in the order of hundreds.

Acknowledgments This work has been funded by Federal Government of Nigeria Tertiary Education Trust Fund under the TETFund 2012 AST$D Intervention for Kano University of Science and Technology, Wudil, Kano State, Nigeria for PhD Overseas Training. This work was also partially funded by FCT/MEC through PIDDAC and ERDF/ON2 within project NORTE-07-0124-FEDER-000059 and through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT Portuguese Foundation for Science and Technology within project FCOMP-01-0124-FEDER-037281.We wish to thank Carlos Soares for his useful comments on the method presented in this paper.

References

1. Abdulrahman, S.M., Brazdil, P.: Measures for Combining Accuracy and Time for Meta-learning. In: Meta-Learning and Algorithm Selection Workshop at ECAI 2014. pp. 49–50 (2014)
2. Brazdil, P., Soares, C.: A Comparison of Ranking Methods for Classification Algorithm Selection. In: Machine Learning: ECML 2000, pp. 63–75. Springer (2000)
3. 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)

4. Brazdil, P., Giraud-Carrier, C., Soares, C., Vilalta, R.: Metalearning: Applications to data mining. Springer Science & Business Media (2008)
5. Demšar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. The Journal of Machine Learning Research 7, 1–30 (2006)
6. Fedorov, V.V.: Theory of Optimal Experiments. Academic Press (1972)
7. Fürnkranz, J., Petrak, J.: An Evaluation of Landmarking Variants. In: Working Notes of the ECML/PKDD 2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning. pp. 57–68 (2001)
8. de Grave, K., Ramon, J., de Raedt, L.: Active Learning for Primary Drug Screening. In: Benelearn 08, The Annual Belgian-Dutch Machine Learning Conference. vol. 2008, pp. 55–56 (2008)
9. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. ACM SIGKDD Explorations Newsletter 11(1), 10–18 (2009)
10. Leite, R., Brazdil, P.: Active Testing Strategy to Predict the Best Classification Algorithm via Sampling and Metalearning. In: ECAI. pp. 309–314 (2010)
11. Leite, R., Brazdil, P., Vanschoren, J.: Selecting Classification Algorithms with Active Testing. In: Machine Learning and Data Mining in Pattern Recognition, pp. 117–131. Springer (2012)
12. Long, B., Chapelle, O., Zhang, Y., Chang, Y., Zheng, Z., Tseng, B.: Active Learning for Ranking through Expected Loss Optimization. In: Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval. pp. 267–274. ACM (2010)
13. Neave, H.R., Worthington, P.L.: Distribution-free Tests. Unwin Hyman London (1988)
14. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Tell me who can learn you and I can tell you who you are: Landmarking various learning algorithms. In: Proceedings of the 17th International Conference on Machine Learning. pp. 743–750 (2000)
15. Provost, F., Jensen, D., Oates, T.: Efficient Progressive Sampling. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 23–32. ACM (1999)
16. Prudencio, R.B., Ludermir, T.B.: Active Selection of Training Examples for Meta-Learning. In: Hybrid Intelligent Systems, 2007. HIS 2007. 7th International Conference on. pp. 126–131. IEEE (2007)
17. Rice, J.R.: The Algorithm Selection Problem. Advances in Computers 15, 65–118 (1976)
18. van Rijn, J.N., Abdulrahman, S.M., Brazdil, P., Vanschoren, J.: Fast Algorithm Selection using Learning Curves. In: Advances in Intelligent Data Analysis XIV. Springer (2015)
19. Smith-Miles, K.A.: Cross-disciplinary Perspectives on Meta-Learning for Algorithm Selection. ACM Computing Surveys (CSUR) 41(1), 6:1–6:25 (2008)
20. 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)

Metalearning for multiple-domain Transfer Learning

Catarina Félix[1,2], Carlos Soares[1,3], and Alípio Jorge[2,4]

[1] INESC TEC, Portugal
[2] Faculdade de Ciências da Universidade do Porto, Portugal
[3] Faculdade de Engenharia da Universidade do Porto, Portugal
[4] LIAAD-INESC TEC, Portugal
cfo@inescporto.pt, csoares@fe.up.pt, amjorge@fc.up.pt

Abstract. Machine learning processes consist in collecting data, obtaining a model and applying it to a given task. Given a new task, the standard approach is to restart the learning process and obtain a new model. However, previous learning experience can be exploited to assist the new learning process. The two most studied approaches for this are metalearning and transfer learning. Metalearning can be used for selecting the predictive model to use over a determined dataset. Transfer learning allows the reuse of knowledge from previous tasks. Our aim is to use metalearning to support transfer learning and reduce the computational cost without loss in terms of performance, as well as the user effort needed for the algorithm selection. In this paper we propose some methods for mapping the transfer of weights between neural networks to improve the performance of the target network, and describe some experiments performed in order to test our hypothesis.

1 Introduction

Machine learning processes consist of 1) collecting training data for the new task; 2) obtaining a model; 3) applying the model to new data. This is done even when the new task is related to one or more tasks previously solved, for example, when there are relationships between variables or between the processes used to obtain the models.

There are two approaches to taking advantage of previous learning experience in new tasks: metalearning and transfer learning. Both transfer learning and metalearning use information about a domain to learn efficiently and effectively in a new one. Metalearning focuses on the choice of a learning algorithm and transfer learning on experience obtained from previous tasks. This suggests that transfer learning and metalearning may be used together.

Our aim is to investigate if metalearning can be used to support transfer learning in tasks consisting of very diverse subtasks, reducing computational cost without loss in predictive performance and cutting down the time data scientists need to perform their tasks.

In this paper we describe some aspects of the state of the art in metalearning and transfer learning. We propose some methods for mapping the transfer of weights between neural networks and describe experiments performed to test the hypothesis that the transfer of weights improves the results of the target network.

2 Metalearning and Transfer Learning

This section presents the basic concepts related with our work. First we describe metalearning, some of its methods and examples of use. After that, we present transfer learning, its motivation, operation mode and some techniques used. Finally, we describe some examples of the combination of metalearning and transfer learning.

2.1 Metalearning

Metalearning aims at helping in the process of selecting a predictive algorithm to use on a determined dataset. It also aims at taking advantage of the repetitive use of a determined method over a set of similar tasks.

There are several applications for metalearning. It can be used in combining base learners: using several learners together to create a composite model that better predicts the result. Another application of metalearning is bias management, mostly used for data streams (continuous flows of data, for example from large and continuously growing databases) that require context adaptation due to the fact that the domain is not static. Metalearning can also be used to transfer metaknowledge across tasks. It is mostly used for the Algorithm Selection Problem, described next.

Algorithm Recommendation Choosing the best algorithm for processing a given dataset is a difficult process. Besides, the algorithms normally have parameters that affect its efficiency and tuning them can be a difficult and slow task. This constitutes the motivation for the Algorithm Selection Problem [1], originally formulated by Rice [2].

This problem consists in determining the best algorithm to use for a certain dataset. The metalearning approach takes advantage of information previously obtained on several datasets and also on several algorithms. This knowledge is used to build a metamodel that, given a new dataset, gives the system the ability to recommend the best suited algorithm.

Earlier applications of metalearning addressed the most common tasks - classification [3], regression [4] and time series [11]. These approaches were then extended to selecting parameter settings for a single algorithm [12], the whole data mining process [13] and also to problems from domains other than machine learning, e.g.: different optimization problems [14,15]. More recently, they were also used to deal with new problems in data mining: data streams [16].

2.2 Transfer Learning

A definition of transfer learning can be found in [17]: given a source domain D_S and a learning task T_S, a target domain D_T and a learning task T_T, transfer learning aims to help improve the learning of the target predictive function $f_T(.)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$.

Transfer learning allows the tasks and distributions used in training and testing to be different. Here, the knowledge is transferred from one task, the source task, to another, the target task. It is inspired in the logic used by the human brain: the methods that allow, for example, someone to recognize pears based on previous knowledge on recognizing apples.

Transfer learning allows algorithms to adapt to new tasks based on the knowledge obtained in previous ones, and the three main research issues in this topic are related to *what*, *how* and *when* to transfer.

What to transfer? This question concerns the type of information transferred between problems: instance-transfer, where instances from the source domain are used together with the ones on the target domain, to improve the performance of the target model, as in TrAdaBoost [20] algorithm; feature-representation-transfer, where a set of feature representations is extracted from the source domain and transferred, obtaining a feature representation of the target domain as in [21]; parameter-transfer that is done by calculating the source model, extracting its parameters and, assuming that the models for related tasks share some parameters, transferring them to build the target model as in [22]; and relational-knowledge-transfer, that consists in trying to transfer the knowledge about data between the domains, as is the case of the TAMAR [23] algorithm.

How to transfer? After knowing the information that should be transferred, the focus is on *how to transfer?*, that is, on the development of learning algorithms to perform the transfer. For example, the DBT (Discriminability-based transfer) algorithm [24] consists in modifying the neural network weights obtained in the source classification task in order to use them on a target network. In [25], a "transfer-aware" naive Bayes classification algorithm is proposed. In [26], first order decision trees are used for reinforcement learning, and some tree statistics are transferred from the source to the target problem. In [27], graph-based transferability is determined: it automatically determines the parameters to transfer between biased logistic regression tasks. The Kolmogorov complexity between tasks is used in [28] to transfer knowledge between bayesian decision trees. [29] introduces a context-sensitive multi task learning that helps improving performance in neural networks for classification. In [30] the authors use clustering to perform a feature selection to be transferred, improving the performance of a Bayesian algorithm.

When to transfer? The last question means to know in which situations the transfer should be performed. Ultimately, the objective is to avoid *negative* *transfer*: when the transfer can harm the learning process in the target task. This issue is referred in [25], where the authors wish to identify when transfer learning will hurt the performance of the algorithm instead of improving it.

2.3 Metalearning and Transfer Learning

Some work has been performed in using metalearning together with transfer learning. We analyzed some literature related to classification tasks that is described next.

Metafeatures are used in [31] for calculating similarities between the datasets. The algorithms used for this task is the *k-nearest neighbors*. In [32, 33] there is no use of metafeatures, since the transfers are made without choosing the best source dataset to use with a certain target dataset. In [32], metalearning is used to find matrix transformations capable of producing good kernel matrices for the source tasks. The matrices are then transferred to the target tasks.

The results are evaluated by performance measures as accuracy [33] and more precisely by the area under the ROC curve in [31, 32].

The transferred objects found on the studied papers are SVM parameter settings in [31], the kernel matrices in [32] and the *parameter function* (responsible for mapping statistics to parameters in "bag-of-words" text classification problems) in [33].

3 Mapping of variables for transfer

We now propose some methods for mapping variables for transfer and show the results of applying the methods in some experiments, using neural networks with three neurons on the hidden layer. The transfer is made from one variable on the source dataset to another one on the target dataset. In a neural network each neuron corresponds to a variable on the dataset, and has a connection to all the neurons on the hidden layer.

The methods proposed are described next:

1. **Random:** the weights are randomly ordered. We repeat this 100 times and generate 100 sets of randomly ordered weights.
2. **Direct:** the weights are transferred directly between corresponding variables, when the datasets have the same structure.
3. **Mapped:** the weights are ordered according to some criteria:
   (a) **Kullback-Leibler divergence:** we obtain the KL divergence between all the attributes of the source dataset and and all the attributes of the target dataset. The transfer is made between the attributes with smaller divergence.
   (b) **Pearson, Spearman and Kendall correlations:** we obtain the correlation between every attribute in each dataset and its target. The transfer is made between the attributes with the most similar correlation to the respective target.

4 Experiments performed

Some experiments have been performed to study if the transfer of knowledge improves the performance of an algorithm. The aim is to measure the success of transferring the weights of a neural network learned on a source dataset to a new neural network that will be trained on a target dataset. All the weights are transferred, according to some mapping, and are used to initialize the network in a non-random way. The resulting error is compared to the one obtained with random initial weights, to assess in which cases occurs an improvement.

In the experiments, the source and target datasets may be unrelated or related (e.g. generated by the same process in different times or generated by processes with the same structure). Weight transfer is performed from one variable in the source model to one variable in the target model. The datasets used were retrieved from UCI [34] and different experiments have been made.

4.1 Experiment 1

The objective of this experiment is to study the behavior of the transfer of knowledge between tasks. We compared random transfer (made between randomly chosen variables) with direct transfer (performed between correspondent variables, in related datasets).

Experiment Description In this experiment, source and target datasets may be unrelated or related (e.g. generated by the same process in different times or generated by processes with the same structure). Weight transfer is performed from one variable in the source model to one variable in the target model. The mapping of variables for the transfer can be random or between corresponding variables.

To perform this experiment the datasets used were:

1. Forest Fires
2. Concrete Compressive Strength
3. Wine Quality (red wine)
4. Challenger USA Space Shuttle O-Ring (erosion only)
5. Concrete Slump Test
6. Computer Hardware
7. Breast Cancer Wisconsin (Prognostic) (Wisconsin Breast Cancer Database)
8. Breast Cancer Wisconsin (Prognostic) (Wisconsin Prognostic Breast Cancer)
9. Parkinsons Telemonitoring
10. Communities and Crime
11. Airfoil Self-Noise
12. Buzz in Social Media (Toms Hardware)
13. Energy efficiency
14. Yacht Aerodinamics
15. Communities and Crime Unnormalized

One of the datasets used, Communities and Crime Unnormalized, has 18 target variables. It was used to generate new datasets using the same original independent variables. These datasets are, then, related to each other. The other datasets are, in principle, independent among themselves. All the datasets were normalized in a preprocessing phase.

For this experiment we ran each dataset through a neural network with three neurons in the hidden layer, using ten-fold cross-validation. First the networks are trained with a random initial set of weights, and we measure the *Mean Squared Error*,$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y_i} - y_i\right)^2$. Then each network is trained with the best set of weights found for the other networks and we also measure the MSE for each network. For each network we compare the error obtained with random initial weights (MSE_R) with the ones obtained with the weights transferred from other networks (MSE_T). We consider that the transfer has improved the result when MSE_R is bigger than MSE_T.

For the unrelated datasets, the transfer was performed randomly. For the related datasets the transfer was performed in two different ways: randomly and also directly between corresponding variables.

Results Figures 1 to 3 show the distribution of the improvements for the experiments. In the x and y axis we have the source and target datasets, respectively. We calculated the number of times when transfer improves the MSE. In these charts the color of the squares represents the number of times the transfer between those datasets improved the performance on the target task: darker squares represent a higher probability of reducing the error when using transfer of weights.

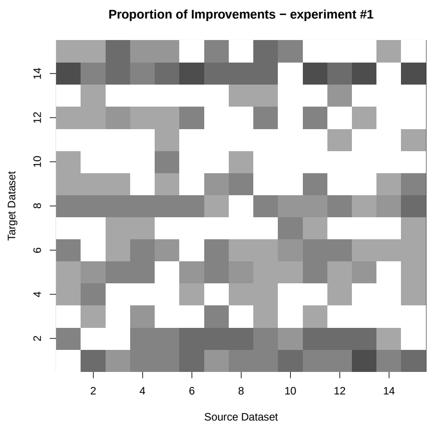


Fig. 1. Distribution of the number of improvements for the first variant of the experiment

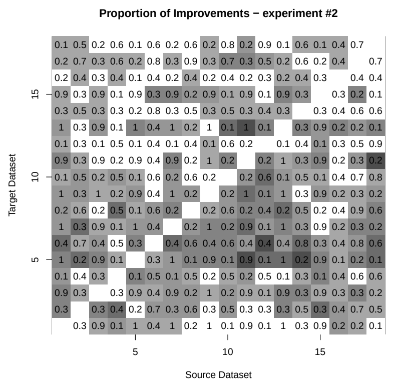


Fig. 2. Distribution of the number of improvements for the second variant of the experiment

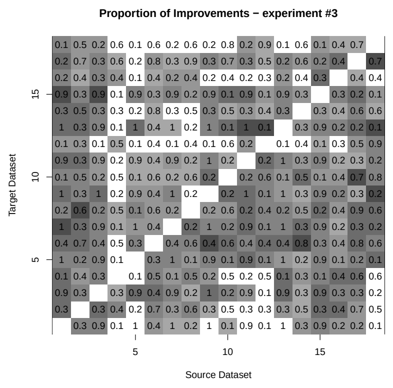


Fig. 3. Distribution of the number of improvements for the third variant of the experiment

In Figures 2 and 3, the values inside the squares represent the Pearson Correlation of the target variables for each pair of datasets.

Figure 4 shows boxplots with the distributions of improvements for the 20 runs of each variant of the experiment. This chart shows that in the first variant the improvement is lowest. Note that datasets used in the second and third variants of the experiment are related, unlike the ones used in the first.

A plausible cause for the last variant of the experiment being the one with more improvements is that not only the datasets are related, but also the transfer of weights is made directly between corresponding variables from one dataset to another, because the structure of the neural network is the same.

The improvement obtained was near 50% for the random transfer between unrelated datasets. This means that random transfer has the same probability of improving the result as it has of deteriorating it. The random and direct transfers between related datasets (with the same attributes but different target variables) show, respectively, around 60% and 70% of improvements. This means that the transfer between related datasets increases the probability of improving the result of a neural network. This probability increases even more when the transfer is made directly between corresponding variables, showing that the transfer between similar (in this case, the same) variables is advantageous.

4.2 Experiment 2

The objective of this experiment was to study the behavior of the transfer of knowledge between similar variables, comparing it to the random transfer of knowledge.

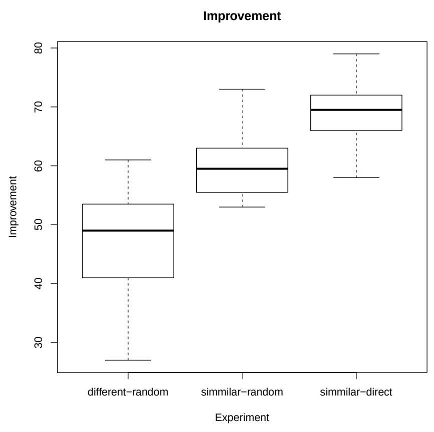


Fig. 4. Global distribution of the MSE's improvement with transfer over the three experiments

Experiment description In this experiment, source and target datasets are in principle unrelated. The datasets used were the ones considered as unrelated on Experiment 1.

1. Concrete Compressive Strength
2. Wine Quality (red wine)
3. Challenger USA Space Shuttle O-Ring (erosion only)
4. Concrete Slump Test[5]
5. Airfoil Self-Noise
6. Energy efficiency[6]
7. Yacht Aerodynamics

Before running the experiment, the datasets were subject to a preprocessing phase, which included the normalization. For this experiment we ran the datasets through a neural network with three neurons in the hidden layer, using ten-fold cross-validation.

First, the initial set of weights fed to each neural network is composed by values generated randomly between 0 and 1. In order to outwit the randomness of the weight generation, the whole processed is repeated 100 times for each dataset. The dataset and weights are fed to the neural network and, using ten-fold cross-validation, we obtain the Mean Squared Error and the Aggregated Weights (mean of the ten sets of weights obtained from the network).

[5] This dataset has 3 target variables. For this experiment only one of them was used: SLUMP (cm).

[6] This dataset has 2 target variables. For this experiment only one of them was used: Y1.

These aggregated weights are transfered to other neural networks to try to improve their performance. The transfer is performed in two ways: random and mapped and the weights are fed to the neural network, together with the target dataset. The learning process occurs and the resulting mean squared error is saved.

The errors obtained in the first learning process (with randomly generated weights - MSE_R) are compared with the ones obtained in the second learning process (with the weights transferred from the other datasets - MSE_T). For this, we calculate: $\frac{MSE_O - MSE_T}{MSE_O}$.

For each pair of datasets, we repeat the transfer several times: $10000\times$ (100×100) for random transfer and $100\times$ for mapped transfer.

Results The chart in Figure 5 shows the probability of improving the performance of the neural network by transferring the weights using the same dataset as source and target.

We can see in the chart that the transfer of the same set of weights generates more improvements than using a new random set of weights. This is because the first is equivalent to running the neural network for twice the iterations, leading to a better fitting of the result.

The charts in Figures 6 to 10 show the results for the different types of mapping: random, using Kullback-Leibler divergence, Pearson, Spearman and Kendall correlations, respectively.

For the random mapping the figure shows, in the left, the mean number of times the transfer improves the predictions and, in the right, the histograms of the same information, where the colors match the ones on the images on the left: gray tones for when the transfer increases the error and the other colors for when there is an improvement.

The same information is shown in the charts that refer to the other types of mappings used. For these, we added a chart, in the middle, that shows the difference, in terms of improvement, between the measured mapping methods and the random mapping method. The colors also match the ones in the histogram.

In all cases the proportion of improvements is below (but near) 50%. Our aim is to find the proper features that allow this proportion to increase.

5 Conclusions and Future Work

We can use related variables to identify characteristics of the model that can be transfered with the advantage of reducing the computational cost and the user effort on the process.

In this paper we described methods for mapping the transfer of weights between neural networks. We also show results of some experiments performed to test the hypothesis that the transfer of weights will improve the results of the neural network.

In the first experiment we obtained an improvement near 50% for the random transfer between unrelated datasets and around 60% and 70% of improvements

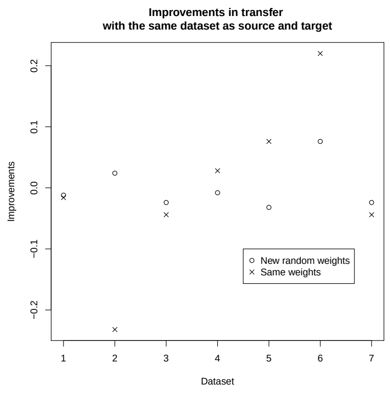


Fig. 5. Improvements in transfer with the same dataset as source and target

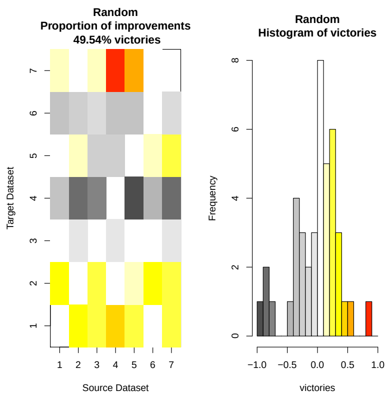


Fig. 6. Results for random mapping

random and direct transfers between related datasets, respectively. This shows that the transfer between similar datasets is advantageous, and the advantages increase even more when the transfer is performed between similar variables.

In the second experiment, that was performed with unrelated datasets, we obtained probabilities of improvement below (but near) 50% for all the mappings considered. We aim to find the proper features that allow increasing this value.

Acknowledgments

This work is financed by the ERDF - European Regional Development Fund through the COMPETE programme (operational programme for competitiveness) within project GNOSIS, cf. FCOMP-01-0202-FEDER-038987.

References

1. Brazdil, P., Giraud-Carrier, C.G., Soares, C., Vilalta, R.: Metalearning - Applications to Data Mining. Cognitive Technologies. Springer (2009)
2. Rice, J.R.: The algorithm selection problem. Advances in Computers **15** (1976) 65–118
3. 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) (2003) 251–277
4. Gama, J., Brazdil, P.: Characterization of classification algorithms. In: Progress in Artificial Intelligence, 7th Portuguese Conference on Artificial Intelligence, EPIA '95, Funchal, Madeira Island, Portugal, October 3-6, 1995, Proceedings. (1995) 189–200
5. Kalousis, A., Gama, J., Hilario, M.: On data and algorithms: Understanding inductive performance. Machine Learning **54**(3) (2004) 275–312

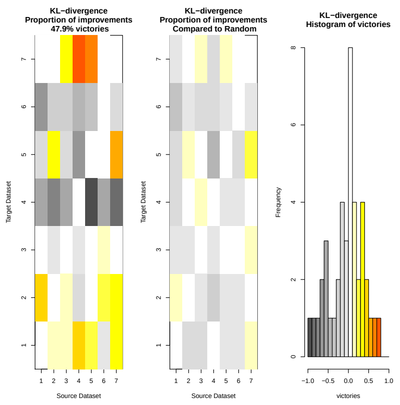


Fig. 7. Results for mapping with Kullback-Leibler divergence

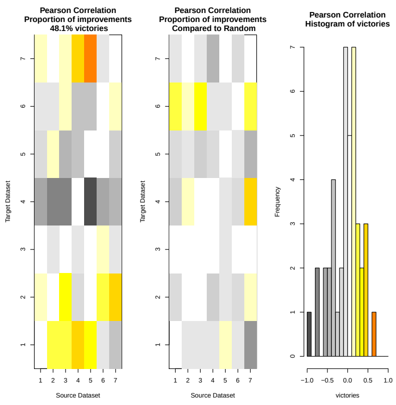


Fig. 8. Results for mapping with Pearson correlation

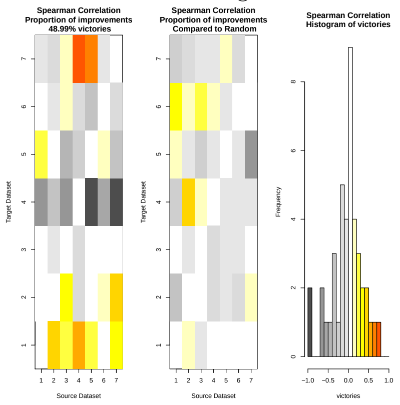


Fig. 9. Results for mapping with Spearman correlation

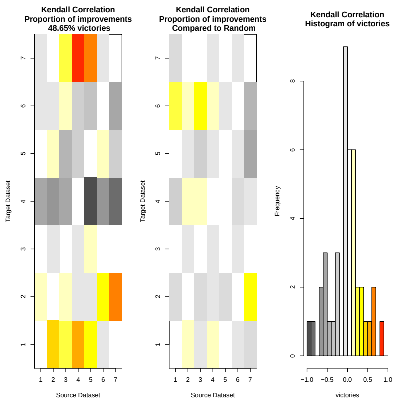


Fig. 10. Results for mapping with Kendall correlation

6. Bensusan, H., Giraud-Carrier, C.G.: Discovering task neighbourhoods through landmark learning performances. In: Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000, Proceedings. (2000) 325–330
7. Soares, C.: UCI++: improved support for algorithm selection using datasetoids. In: Advances in Knowledge Discovery and Data Mining, 13th Pacific-Asia Conference, PAKDD 2009, Bangkok, Thailand, April 27-30, 2009, Proceedings. (2009) 499–506
8. Macià, N., Orriols-Puig, A., Bernadó-Mansilla, E.: Genetic-based synthetic data sets for the analysis of classifiers behavior. In: Hybrid Intelligent Systems, 2008. HIS'08. Eighth International Conference on, IEEE (2008) 507–512
9. Blockeel, H., Vanschoren, J.: Experiment databases: Towards an improved experimental methodology in machine learning. In: Knowledge Discovery in Databases: PKDD 2007, 11th European Conference on Principles and Practice of Knowledge

Discovery in Databases, Warsaw, Poland, September 17-21, 2007, Proceedings. (2007) 6–17

10. Prudêncio, R.B.C., Soares, C., Ludermir, T.B.: Uncertainty sampling-based active selection of datasetoids for meta-learning. In: Artificial Neural Networks and Machine Learning - ICANN 2011 - 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part II. (2011) 454–461
11. Prudêncio, R.B.C., Ludermir, T.B.: Meta-learning approaches to selecting time series models. Neurocomputing **61** (2004) 121–137
12. Gomes, T.A.F., Prudêncio, R.B.C., Soares, C., Rossi, A.L.D., Carvalho, A.C.P.L.F.: Combining meta-learning and search techniques to select parameters for support vector machines. Neurocomputing **75**(1) (2012) 3–13
13. Serban, F., Vanschoren, J., Kietz, J.U., Bernstein, A.: A survey of intelligent assistants for data analysis. ACM Comput. Surv. **45**(3) (July 2013) 31:1–31:35
14. Abreu, P., Soares, C., Valente, J.M.S.: Selection of heuristics for the job-shop scheduling problem based on the prediction of gaps in machines. In: Learning and Intelligent Optimization, Third International Conference, LION 3, Trento, Italy, January 14-18, 2009. Selected Papers. (2009) 134–147
15. Smith-Miles, K.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. **41**(1) (2008)
16. Gama, J., Kosina, P.: Learning about the learning process. In: Advances in Intelligent Data Analysis X - 10th International Symposium, IDA 2011, Porto, Portugal, October 29-31, 2011. Proceedings. (2011) 162–172
17. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10) (2010) 1345–1359
18. Yoshida, Y., Hirao, T., Iwata, T., Nagata, M., Matsumoto, Y.: Transfer learning for multiple-domain sentiment analysis - identifying domain dependent/independent word polarity. In: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011. (2011)
19. Caruana, R.: Multitask learning. Machine Learning **28**(1) (1997) 41–75
20. Dai, W., Yang, Q., Xue, G., Yu, Y.: Boosting for transfer learning. In: Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007. (2007) 193–200
21. Blitzer, J., McDonald, R., Pereira, F.: Domain adaptation with structural correspondence learning. In: Proceedings of the 2006 conference on empirical methods in natural language processing, Association for Computational Linguistics (2006) 120–128
22. Gao, J., Fan, W., Jiang, J., Han, J.: Knowledge transfer via multiple model local structure mapping. In: In International Conference on Knowledge Discovery and Data Mining, Las Vegas, NV. (2008)
23. Mihalkova, L., Huynh, T., Mooney, R.J.: Mapping and revising markov logic networks for transfer learning. In: In Proceedings of the 22 nd National Conference on Artificial Intelligence (AAAI. (2007) 608–614
24. Pratt, L.Y., Pratt, L.Y., Hanson, S.J., Giles, C.L., Cowan, J.D.: Discriminability-based transfer between neural networks. In: Advances in Neural Information Processing Systems 5, Morgan Kaufmann (1993) 204–211
25. Rosenstein, M.T., Marx, Z., Kaelbling, L.P., Dietterich, T.G.: To transfer or not to transfer. In: In NIPS'05 Workshop, Inductive Transfer: 10 Years Later. (2005)
26. Ramon, J., Driessens, K., Croonenborghs, T.: Transfer learning in reinforcement learning problems through partial policy recycling. In: Machine Learning: ECML 2007. Springer (2007) 699–707

27. Eaton, E., Desjardins, M., Lane, T.: Modeling transfer relationships between learning tasks for improved inductive transfer
28. Mahmud, M., Ray, S.: Transfer learning using kolmogorov complexity: basic theory and empirical evaluations. In: Advances in neural information processing systems. (2007) 985–992
29. Silver, D.L., Poirier, R., Currie, D.: Inductive transfer with context-sensitive neural networks. Machine Learning **73**(3) (2008) 313–336
30. Mishra, M., Huan, J.: Multitask learning with feature selection for groups of related tasks. In: Data Mining (ICDM), 2013 IEEE 13th International Conference on, IEEE (2013) 1157–1162
31. Biondi, G., Prati, R.: Setting parameters for support vector machines using transfer learning. Journal of Intelligent & Robotic Systems (2015) 1–17
32. Aiolli, F.: Transfer learning by kernel meta-learning. In: ICML Unsupervised and Transfer Learning. (2012) 81–95
33. Do, C., Ng, A.Y.: Transfer learning for text classification. In: NIPS. (2005)
34. Bache, K., Lichman, M.: UCI machine learning repository (2013)

Meta-learning Recommendation of Default Hyper-parameter Values for SVMs in Classifications Tasks

Rafael G. Mantovani[1], André L. D. Rossi[2], Joaquin Vanschoren[3], and André C. P. L. F. Carvalho[1]

[1] Universidade de São Paulo (USP), So Carlos - Brazil,
{rgmantov,andre}@icmc.usp.br

[2] Universidade Estadual Paulista (UNESP), Itapeva - SP, Brazil
alrossi@itapeva.unesp.br

[3] Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands
j.vanschoren@tue.nl

Abstract. Machine learning algorithms have been investigated in several scenarios, one of them is the data classification. The predictive performance of the models induced by these algorithms is usually strongly affected by the values used for their hyper-parameters. Different approaches to define these values have been proposed, like the use of default values and optimization techniques. Although default values can result in models with good predictive performance, different implementations of the same machine learning algorithms use different default values, leading to models with clearly different predictive performance for the same dataset. Optimization techniques have been used to search for hyper-parameter values able to maximize the predictive performance of induced models for a given dataset, but with the drawback of a high computational cost. A compromise is to use an optimization technique to search for values that are suitable for a wide spectrum of datasets. This paper investigates the use of meta-learning to recommend default values for the induction of Support Vector Machine models for a new classification dataset. We compare the default values suggested by the Weka and LibSVM tools with default values optimized by meta-heuristics on a large range of datasets. This study covers only classification task, but we believe that similar ideas could be used in other related tasks. According to the experimental results, meta-models can accurately predict whether tool suggested or optimized default values should be used.

Keywords: Meta-learning. Hyper-parameter tuning. Default Values. Support Vector Machines.

1 Introduction

Support Vector Machine (SVMs) have been successfully used for classification tasks [21]. However, their predictive performance for a given dataset is affected by their hyper-parameter values. Several approaches have been proposed to choose these values. Some machine learning tools suggest hyper-parameter values for SVMs regardless of the dataset analyzed, or employ simple heuristics [8]. Although these values can induce models with good predictive performance [6] this does not occur in many situations, requiring a fine tuning process [4,13,25].

However, the optimization of these hyper-parameters usually has a high computational cost, since a large number of candidate solutions needs to be evaluated. An alternative is to generate a new set of default values by optimizing these hyper-parameter values over several datasets rather than for each one. The optimized common values may improve the model accuracy, when compared with the use of the default values, and reduce the computation cost to induce models, when compared with a optimization for each dataset.

This study proposes a recommendation system able to indicate which default hyper-parameters values should be used in SVMs when applied to new datasets. This recommendation is based on Meta-learning (MTL) [7] ideas to induce a classification model that, based on some features of a dataset, indicates which hyper-parameters default values should be used: those proposed by ML tools or those achieved by an optimization technique considering a set of prior datasets.

The proposed recommendation system is evaluated experimentally using a large number of classification datasets and considering three sets of hyper-parameters values for SVMs: default values from LibSVM [9], default values from Weka [16], and those obtained from an pre-optimization process with prior datasets, from here on referred to as "Optimized". We employed a Particle Swarm Optimization (PSO) [20] algorithm to perform the optimization. This study covers only classification task, but we believe that similar ideas could be used in other related tasks.

This paper is structured as follows: section 2 contextualizes the hyper-parameter tuning problem and cites some techniques explored by related work. Section 3 presents our experimental methodology and steps covered to evaluate the approaches. The results are discussed in section 4. The last section presents our conclusions and future work.

2 Hyper-parameter tuning

Hyperparameter optimization is a crucial step in the process of applying ML in practice [12]. Setting a suitable configuration for the hyperparameters of a ML algorithm requires specific knowledge, intuition and, often, trial and error. Depending on the training time of the algorithm at hand, finding good hyper-parameters values manually is time-consuming and tedious. As a result, much recent work in ML has focused on the study of methods able to find the best hyper-parameter values [19].

The tuning of these hyperparameters is usually treated as an optimization problem, whose objective function captures the predictive performance of the model induced by the algorithm. As related in [24], this tuning task may present many aspects that can make it difficult: i) some hyperparameter values that lead to a model with high predictive performance for a given dataset may not lead to good results for other datasets; ii) the hyperparameter values often depend on each other, and this must be considered in the optimization; and iii) the evaluation of a specific hyperparameter configuration, let alone many, can be very time consuming.

Many approaches have been proposed for the optimization of hyperparameters of classification algorithms. Some studies used Grid Search (GS), a simple deterministic approach that provides good results in low dimensional problems [6]. For optimization of many hyperparameters on large datasets, however, GS becomes computationally infeasible due to the combinatorial explosion. In these scenarios, probabilistic approaches, such as Genetic Algorithms (GA), are generally recommended [13]. Other authors explored the use of Pattern Search (PS) [11] or techniques based on gradient descent [10]. Many automated tools are also available in the literature, such as methods based on local search (ParamILS [18]), estimation of distributions (REVAC [23]) and Bayesian optimization (Auto-Weka [28]).

Recent studies have shown the effectiveness of Random Sampling (RS) methods **[1]** for hyper-parameter fine tuning. In [5], the authors use RS to tune Deep Belief Networks (DBNs), comparing its performance with grid methods and showed empirically and theoretically that RS are more efficient for hyperparameter optimization than trials on a grid. Other recent works use a collaborative filtering solution [3], or combine optimization techniques for tuning algorithms in computer vision problems [4].

3 Materials and methods

In addition to the default values suggested by LibSVM and Weka, an optimization technique was used to search for a new set of values suitable for a group of datasets. For such, the predictive performance of models induced by SVMs for public data sets using a PSO algorithm to tune SVM's hyper-parameters was evaluated.

In the PSO optimization, each particle encodes one hyper-parameter setting composed of a pair of real values representing the SVM hyper-parameter C (cost) and the width of the Gaussian kernel γ. The former is a SVM parameter and the latter is the Gaussian kernel parameter [17]. Table 1 shows the range of values for C and γ [26] used in the optimization. The default values provided by the Weka [16] and LibSVM tools [9], and the obtained optimized values are listed in Table 2.

Table 1. SVM hyper-parameters range values investigated during optimization [26].

Hyper-parameter	**Minimum**	**Maximum**
cost (C)	2^{-2}	2^{15}
gamma (γ)	2^{-15}	2^{3}

Table 2. Default values tested in the datasets and used to generate meta-labels.

Approach	**Cost (C)**	**Gamma (γ)**
DF-Weka	1	0.1
DF-LibSVM[4]	1	1/attrs
DF-Optimized[5]	$2^{5.6376}$	$2^{-8.2269}$

3.1 Datasets

For the experiments, 145 classification datasets with different characteristics were collected from the UCI repository [2] and OpenML [29]. These datasets were split into two groups:

- One group contains 21 datasets that were used in the optimization process to find common values of the hyper-parameters. These datasets were randomly selected from the total amount of 145;
- The second group, containing the 124 remaining datasets, were used to test the models induced with the hyper-parameters values found by the optimization process. These 124 datasets and models results were used in the meta-learning system.

Only few datasets were selected to the optimization to not spend too much time, and because we need the other for the meta-learning. All datasets were standardized with $\mu = 0$ e $\sigma = 1$ internally by package 'e1071' (R interface for 'LibSVM' library), employed here to train SVMs.

3.2 Optimization process

Figure 1 illustrates the optimization process. The PSO algorithm is run with the 21 training datasets. The evaluation of the hyper-parameters uses 10-fold cross-validation (CV). Whenever a pair of SVM hyper-parameter values is generated by the tuning technique, one model is induced for each dataset using 8 partitions (training folds). One of the remaining partitions is used to validate the induced models, and will guide the search for the best hyper-parameter values (validation fold). The final one is used to asses the predictive performance of the induced models (test fold) only, not for hyper-parameter selection. This way, each dataset has validation and testing accuracies averaged over the 10-fold CV. The *fitness criteria* was defined as the median validation accuracy.

The PSO algorithm was implemented in R using the "pso" package, available on CRAN[6]. Since PSO is a stochastic method, the technique was run 30 times

[4] attrs: the number of attributes in the dataset (except the target attribute)

[5] Those are the values that presented the median accuracy over 30 solutions found in the optimization process. See Section3.2

[6] http://cran.r-project.org/

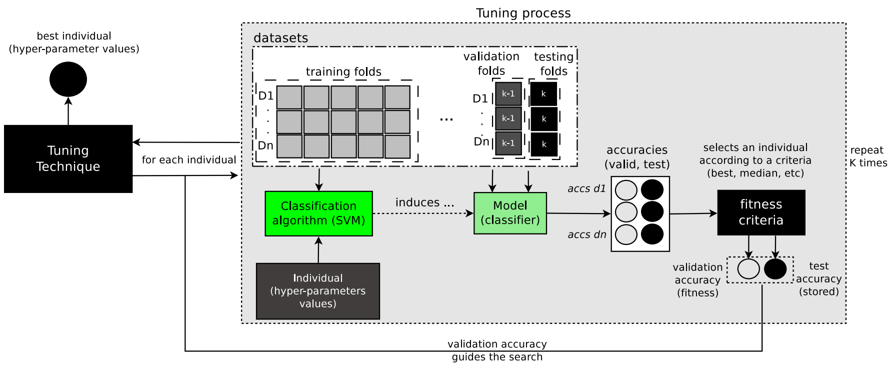


Fig. 1. SVM hyper-parameter tuning process with multiple datasets.

for each training dataset, so we obtain 30 solutions. The hyper-parameters values that resulted in the best median testing accuracy, considering the training datasets and executions, are defined as the "Optimized Default" values found by the optimization technique. Those values will be compared to the default ones provided by ML tools in Section 4.

3.3 Meta-learning system

The problem of choosing one of the default values shown in Table 2 can be viewed as a classification task, and solved using a meta-learning approach. A meta-dataset is created by extracting characteristics from the datasets and used to induce a meta-model that predicts the best set of hyper-parameters based on these data characteristics. Then, this meta-model can be applied to predict which default values are more likely to lead to good predictive SVM models for a new dataset.

3.4 Meta-data set

Each meta-example of the meta-data set is composed of meta-features and a target feature. The meta-features are extracted from the 124 datasets from the total amount of 145 (Sec. 3.1). The other 21 datasets were used to find the *DF-Optimized* parameter settings, and are therefore excluded in the meta-learning system. Since each dataset results in one meta-example, the meta-data set contains 124 meta-examples, each one composed of 81 meta-features. Table 3 shows an overview of the meta-features obtained from these datasets, subdivided into 7 subsets. These meta-features were used before in many similar studies [14, 15, 25, 27].

For each subset of these meta-features, a different meta-data set was created to explore their utility for this task. Furthermore, we built a meta-data set merging all meta-features, referred to as *ALL*, and another one, referred as *FEAT.SELEC.*, obtained using a meta-feature selection method on the subset

Table 3. Classes and number of meta-features used in experiments.

Meta-features	**Num.**	**Description**
Statlog	17	Simple measures, such as number of attributes classes and attributes.
Statistical	7	Statistics measures, such as the skewness and kurtosis.
Information	7	Information theory measures, such as the attributes' entropy, and so on.
Landmarking	10	The performance of some ML algorithms on the datasets
Model	18	Features extracted from DTs models, such as the number of leaves, nodes, rules.
Time	9	The execution time of some ML algorithms on these dataset.
Complexity	13	measures that analyze the complexity of a classification problem.
Total	81	All meta-features

ALL. Specifically, we employed the correlation rank method from R package 'FSelector', selecting the 25% most correlated meta-features.

Besides the meta-features, each meta-example has a target, whose label indicates which default hyper-parameter values should be used on the corresponding dataset. In order to define the label of the meta-examples, we run the three sets of default values (DF-LibSVM, DF-Weka, and DF-Optimized) on the 124 test datasets. The hyper-parameters values that yielded the median accuracy value over 30 executions are selected.

All of the default approaches were evaluated performing 10-CV strategy on testing datasets. This procedure was repeated 30 times and the predictive performance of models assessed by the mean balanced accuracy. The Wilcoxon sign-test was applied for each pair of alternatives for the default values to assess the significance of the differences of accuracy measures per dataset. Table 4 shows the win-tie-loss results based on this significance test with a confidence level of 95%.

Table 4. Win-tie-loss of the approaches for 124 datasets.

Technique	**Win**	**Tie**	**Loss**
DF-Weka	13	21	90
DF-LibSVM	6	20	98
DF-Optimized	84	6	34

In these initial experiments, we considered the problem as binary, specially due to a small number of DF-Weka and DF-LibSVM wins and eventual ties. Thus, if the best mean accuracy for the dataset was obtained by the DF-Optimized with statistical significance (Wilcoxon test) compared to the other both approaches, a meta-example receives the label "OPTM". Otherwise, it is labeled as "DF".

According to this criteria, 84 of the 124 datasets were labeled with the *OPTM* class: the induced models presented the best predictive performance when it used the parameter values obtained by the optimization process. The other 40 meta-examples were labeled with *DF* class: default values provided by tools were enough. Due to the small number of meta-examples, the Leave-One-Out

Cross-Validation (LOO-CV) methodology was adopted to evaluate the predictive performance of the meta-learners.

3.5 Meta-learner

Six ML classification algorithms were used as meta-learners: J48 Decision Tree (J48), Naïve Bayes (NB), k-Nearest Neighbors (k-NN) with $k = 3$, Multilayer Perceptron (MLP), Random Forest (RF) and Support Vector Machines (SVM). These algorithms follow different learning paradigms, each one with a distinct bias, and may result in different predictions. An ensemble (ENS) of these classifiers was also used, with prediction defined by majority voting.

The predictive performance of each meta-learner, including the ensemble, was averaged over all LOO-CV iterations/executions for four performance measures. Each meta-learner was evaluated with meta-data sets composed by meta-features extracted by different approaches, described in Table 3, and the meta-feature sets *ALL*, which combines all meta-features, and *FEAT.SELEC.*, which applies feature selection to ALL.

4 Experimental results

The predictive performance of models induced using the optimized default values for SVMs were compared with hyper-parameter values provided by SVMs tools. This comparison was performed by applying the Friedman statistical test and the Nemenyi post-hoc test with a confidence level of 95%. According to the test, the hyper-parameter values optimized by the PSO technique for several datasets (DF-Optimized) led to SVMs models with significantly better predictive performance than the default values provided by both SVMs tools (DF-Weka and DF-LibSVM) (see Table 4). Moreover, the test showed that there is no significance difference between the performance of DF-Weka and DF-LibSVM values.

4.1 MTL predictive performance

Table 5 summarizes the predictive performance of the meta-learners for different sets of meta-features. The first column identifies the meta-learning algorithm. The second column shows the meta-feature set used. The other columns present the predictive performance of the meta-learner according to different predictive performance measures: balanced accuracy, precision, recall, and F-Score. A trivial classifier would have a mean balanced accuracy equal to 0.500. The performance measures of this baseline method (*MAJ.CLASS*) and of a *RANDOM* method are included at the bottom of the Table 5. The random method selects labels randomly. The best results for each meta-feature set according to the F-score measure are highlighted.

A general picture of the predictive performance of the meta-learners is provided by the F-Score measure, which is a balance between precision and recall measures, and mean balanced classification accuracy. According to these values, the J48 algorithm using all the meta-features was the best meta-learner overall, with an F-Score of 0.821 and balanced accuracy of 0.847. The same combination of meta-learner and meta-features also achieved the best results according to the precision measure. For the recall measure, the best result was also obtained by J48 algorithm, but using the Statlog meta-features subset.

4.2 Hits and Misses

Figure 2 depicts the hits and misses of the top-10 meta-models analyzing the F-score measure. The y-axis represents the meta-models: the algorithm and the set of meta-features used in the experiments. The x-axis represents all the 124 meta-examples of the meta-data set. In the figure, a hit is represented by a light gray square, and a miss by a black one.

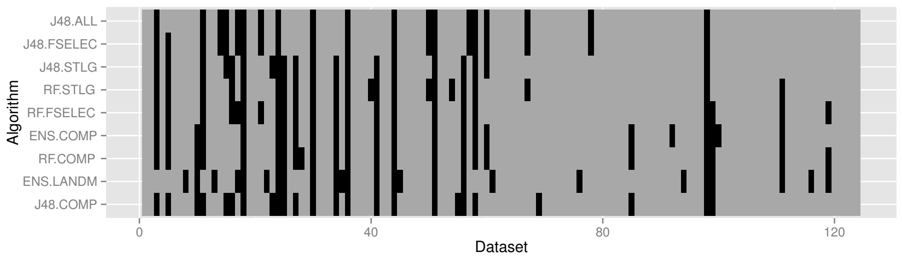


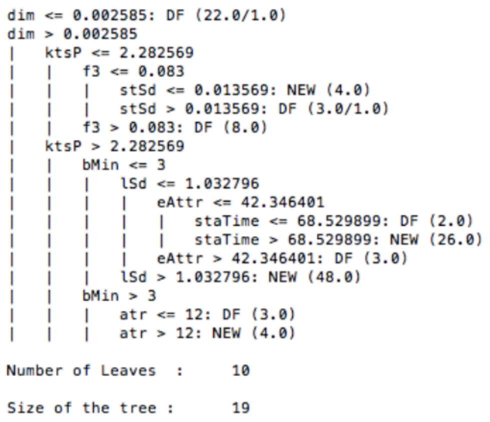
Fig. 2. Hits and misses of the top 10 meta-models regarding the F-score.

The J48 algorithm appears four times in the list, while RF and ENS appear three times each one. These results indicate the superiority of J48 for this task, differently from other similar meta-learning studies, such as [22]. The intrinsic feature selection mechanism of J48 performed slightly better than the rank correlation based method (FEAT.SELEC.), since the meta-model J48-ALL is the first in the ranking followed by "J48.FSELEC". Another feature selection method may further improve the meta-learners predictive performance. Figure 2 illustrates that few meta-examples were misclassified by all meta-models. In these cases, all meta-examples are labeled as DF.

4.3 Tree Analysis

The decision tree in Figure 3 was the most frequently induced model during the meta-level learning using the J48 algorithm with all meta-features and performing LOO-CV. This pruned tree was obtained in most of the experiments and kept basically the same structure with 19 nodes, of which 10 are leaf nodes, and 10 rules. The meta-features selected by J48 as the most relevant ones were:

1. *dim*: the problem dimensionality (Statlog);
2. *ktsP*: kurtosis pre-processed (Statistical);
3. *f3*: maximum individual feature efficiency (Complexity);
4. *stSd*: standard deviation of stump time (Landmarking);
5. *bMin*: minimum level of branches (tree) (Model-based);
6. *lSd*: standard deviation of leaves (Model-based);
7. *eAttr*: attribute entropy (Information);
8. *staTime*: the execution time of a statistical model (Time);
9. *attr*: number of attributes (Statlog).



```
dim <= 0.002585: DF (22.0/1.0)
dim > 0.002585
|   ktsP <= 2.282569
|   |   f3 <= 0.083
|   |   |   stSd <= 0.013569: NEW (4.0)
|   |   |   stSd > 0.013569: DF (3.0/1.0)
|   |   f3 > 0.083: DF (8.0)
|   ktsP > 2.282569
|   |   bMin <= 3
|   |   |   lSd <= 1.032796
|   |   |   |   eAttr <= 42.346401
|   |   |   |   |   staTime <= 68.529899: DF (2.0)
|   |   |   |   |   staTime > 68.529899: NEW (26.0)
|   |   |   |   eAttr > 42.346401: DF (3.0)
|   |   |   lSd > 1.032796: NEW (48.0)
|   |   bMin > 3
|   |   |   atr <= 12: DF (3.0)
|   |   |   atr > 12: NEW (4.0)

Number of Leaves  :     10

Size of the tree :      19
```

Fig. 3. Most common J48 DT with all meta-features.

It is interesting to observe that about one meta-feature from each subset was used to generate the tree. The predictive meta-feature most frequently selected as the root node was *dim*: the problem dimensionality, i.e., $dim = \frac{attributes}{samples}$. The LibSVM library considers the dimensionality of the dataset (Table 2) to assign the γ hyper-parameter value. However, the meta-feature *dim* is a ratio between the number of attributes and examples.

According to the tree, this ratio is close to zero, DF hyper-parameter values are already good solutions, and the pre-optimized values do not improve the model's accuracy. However, if the execution time of a statistical model (*staTime*) is superior to 68.25, it indicates that the optimized hyper-parameter values should be used. The pre-optimized values are also recommended if a standard deviation of the number of leaves generated by model-based DTs is higher than 1.

5 Conclusion

Many experiments with SVMs use default values for the hyper-parameters.Thus, a good set default values allow non-expert users to have good models with low computational costs. This study investigated the development of a meta-learning system to recommend hyper-parameter values for Support Vector Machines (SVMs) from a set of predefined default values. The meta-learning system was experimentally evaluated using 124 datasets from UCI and OpenML.

Besides the default values proposed by ML tools, we used an optimization technique to define new default hyper-parameter values based on a group of datasets. The use of this new set of hyper-parameter values, referred to as optimized default values, produced significantly better models than the default values suggested by ML tools.

According to the experiments to assess the performance of the meta-learning system, it is possible to create a recommendation system able to select which default values must be used for SVM hyper-parameters for classification tasks. Observing the most frequent decision tree, a small number of simple meta-features was sufficient to characterize the datasets. According to this decision tree, default values proposed by ML tools are suitable for problems with a dimensionality ratio close to zero.

As future work, we intend to expand the experiments by increasing the number of datasets and meta-features and exploring other ML algorithms. We also plan to cluster datasets according to their similarities to generate better optimized hyper-parameter values. The fitness value used in experiments is an aggregate measure of performance across different datasets. It would be interesting to explore other measures such as average ranks. We pretend to build on, and make all our experiments available in OpenML [29].

Acknowledgments. The authors would like to thank CAPES, CNPq (Brazilian Agencies) for the financial support. This project is supported by São Paulo Research Foundation (FAPESP) under the grant#2012/23114-9.

References

1. Andradottir, S.: A review of random search methods. In: Fu, M.C. (ed.) Handbook of Simulation Optimization, International Series in Operations Research & Management Science, vol. 216, pp. 277 – 292. Springer New York (2015)
2. Bache, K., Lichman, M.: UCI machine learning repository (2013), `http://archive.ics.uci.edu/ml`
3. Bardenet, R., Brendel, M., Kégl, B., Sebag, M.: Collaborative hyperparameter tuning. In: Dasgupta, S., Mcallester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning (ICML-13). vol. 28, pp. 199–207. JMLR Workshop and Conference Proceedings (2013)
4. Bergstra, J., Yamins, D., Cox, D.D.: Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: Proc. 30th Intern. Conf. on Machine Learning. pp. 1–9 (2013)
5. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. 13, 281–305 (Mar 2012)
6. Braga, I., do Carmo, L.P., Benatti, C.C., Monard, M.C.: A note on parameter selection for support vector machines. In: Castro, F., Gelbukh, A., González, M.

(eds.) Advances in Soft Computing and Its Applications, LNCC, vol. 8266, pp. 233–244. Springer Berlin Heidelberg (2013)

7. Brazdil, P., Giraud-Carrier, C., Soares, C., Vilalta, R.: Metalearning: Applications to Data Mining. Springer Verlag, 2 edn. (2009)
8. Chang, C.C., Lin, C.J.: LIBSVM: a Library for Support Vector Machines (2001), software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
9. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011), software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
10. Chapelle, O., Vapnik, V., Bousquet, O., Mukherjee, S.: Choosing multiple parameters for support vector machines. Machine Learning 46(1-3), 131–159 (Mar 2002)
11. Eitrich, T., Lang, B.: Efficient optimization of support vector machine learning parameters for unbalanced datasets. Journal of Comp. and Applied Mathematics 196(2), 425–436 (2006)
12. Feurer, M., Springenberg, T., Hutter, F.: Initializing bayesian hyperparameter optimization via meta-learning. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (Jan 2015)
13. Friedrichs, F., Igel, C.: Evolutionary tuning of multiple svm parameters. Neurocomput. 64, 107–117 (2005)
14. Garcia, L.P.F., de Carvalho, A.C., Lorena, A.C.: Noisy data set identification. In: Pan, J.S., Polycarpou, M.M., Wo?niak, M., de Carvalho, A.C., Quintin, H., Corchado, E. (eds.) Hybrid Artificial Intelligent Systems, Lecture Notes in Computer Science, vol. 8073, pp. 629–638. Springer Berlin Heidelberg (2013)
15. Gomes, T.A.F., Prudêncio, R.B.C., Soares, C., Rossi, A.L.D., nd André C. P. L. F. De Carvalho: Combining meta-learning and search techniques to select parameters for support vector machines. Neurocomput. 75(1), 3–13 (Jan 2012)
16. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explor. Newsl. 11(1), 10–18 (Nov 2009)
17. Hsu, C.W., Chang, C.C., Lin, C.J.: A Practical Guide to Support Vector Classification. Department of Computer Science - National Taiwan University, Taipei, Taiwan (2007)
18. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: Paramils: an automatic algorithm con- figuration framewor. Journal of Artificial Intelligence Research (36), 267–306 (2009)
19. Hutter, F., Hoos, H.H., Stützle, T.: Automatic algorithm configuration based on local search. In: Proceedings of the 22nd national conference on Artificial intelligence - Volume 2. pp. 1152–1157. AAAI'07, AAAI Press (2007)
20. Kennedy, J.: Particle swarms: optimization based on sociocognition. In: Castro, L., Zuben, F.V. (eds.) Recent Development in Biologically Inspired Computing, pp. 235–269. Idea Group (2005)
21. Koch, P., Bischl, B., Flasch, O., Bartz-Beielstein, T., Weihs, C., Konen, W.: Tuning and evolution of support vector kernels. Evolutionary Intelligence 5(3), 153–170 (2012)
22. Mantovani, R.G., Rossi, A.L.D., Bischl, B., Vanschoren, J., Carvalho, A.C.P.L.F.: To tune or not to tune: recommending when to adjust svm hyper-parameters via meta-learning. In: Proceedings of 2015 International Joint Conference on Neural Network (Jul 2015)
23. Nannen, V., Eiben, A.E.: Relevance estimation and value calibration of evolutionary algorithm parameters. In: Proc. of the 20th Intern. Joint Conf. on Art. Intelligence. pp. 975–980. IJCAI'07 (2007)

24. Reif, M., Shafait, F., Dengel, A.: Meta-learning for evolutionary parameter optimization of classifiers. Machine Learning 87, 357–380 (2012)
25. 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)
26. Rossi, A.L.D., Carvalho, A.C.P.L.F.: Bio-inspired optimization techniques for svm parameter tuning. In: Proceed. of 10th Brazilian Symp. on Neural Net. pp. 435–440. IEEE Computer Society (2008)
27. Soares, C., Brazdil, P.B., Kuba, P.: A meta-learning method to select the kernel width in support vector regression. Machine Learning 54(3), 195–209 (2004)
28. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: Proc. of KDD-2013. pp. 847–855 (2013)
29. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: Networked science in machine learning. SIGKDD Explorations 15(2), 49–60 (2013)

Table 5. Meta-learning results using LOO-CV.

Classifier	Meta-features	Bal. Acc.	Precision	Recall	F-Score
J48	STATLOG	0.839	0.757	0.884	**0.785**
MLP	STATLOG	0.766	0.703	0.737	0.714
NB	STATLOG	0.427	0.518	0.522	0.424
3-NN	STATLOG	0.734	0.679	0.693	0.685
RF	STATLOG	0.823	0.764	0.815	0.781
SVM	STATLOG	0.758	0.645	0.781	0.654
ENS	STATLOG	0.798	0.733	0.785	0.749
J48	STATISTICAL	0.734	0.660	0.695	0.669
MLP	STATISTICAL	0.677	0.572	0.608	0.570
NB	STATISTICAL	0.492	0.592	0.608	0.489
3-NN	STATISTICAL	0.750	0.717	0.715	**0.716**
RF	STATISTICAL	0.742	0.672	0.705	0.682
SVM	STATISTICAL	0.702	0.616	0.649	0.622
ENS	STATISTICAL	0.718	0.667	0.675	0.670
J48	INFORMATION	0.806	0.726	0.817	**0.747**
MLP	INFORMATION	0.782	0.708	0.767	0.724
NB	INFORMATION	0.637	0.601	0.596	0.597
3-NN	INFORMATION	0.677	0.638	0.634	0.636
RF	INFORMATION	0.782	0.695	0.782	0.713
SVM	INFORMATION	0.758	0.645	0.781	0.654
ENS	INFORMATION	0.774	0.689	0.765	0.705
J48	LANDMARKING	0.766	0.710	0.734	0.719
MLP	LANDMARKING	0.758	0.717	0.723	0.719
NB	LANDMARKING	0.702	0.649	0.655	0.652
3-NN	LANDMARKING	0.750	0.724	0.716	0.719
RF	LANDMARKING	0.782	0.721	0.758	0.734
SVM	LANDMARKING	0.774	0.715	0.746	0.726
ENS	LANDMARKING	0.798	0.753	0.773	**0.761**
J48	MODEL	0.734	0.673	0.693	0.680
MLP	MODEL	0.734	0.686	0.694	0.689
NB	MODEL	0.677	0.579	0.610	0.579
3-NN	MODEL	0.677	0.651	0.641	0.644
RF	MODEL	0.782	0.735	0.753	**0.742**
SVM	MODEL	0.734	0.627	0.714	0.633
ENS	MODEL	0.774	0.722	0.744	0.730
J48	TIME	0.718	0.635	0.673	0.642
MLP	TIME	0.790	0.701	0.801	0.720
NB	TIME	0.403	0.546	0.601	0.376
3-NN	TIME	0.766	0.729	0.732	**0.731**
RF	TIME	0.774	0.715	0.746	0.726
SVM	TIME	0.766	0.638	0.872	0.642
ENS	TIME	0.774	0.722	0.744	0.730
J48	COMPLEXITY	0.806	0.739	0.799	0.757
MLP	COMPLEXITY	0.774	0.715	0.746	0.726
NB	COMPLEXITY	0.750	0.730	0.718	0.723
3-NN	COMPLEXITY	0.710	0.648	0.663	0.653
RF	COMPLEXITY	0.806	0.746	0.793	0.761
SVM	COMPLEXITY	0.806	0.713	0.844	0.736
ENS	COMPLEXITY	0.815	0.758	0.801	**0.773**
J48	ALL	0.847	0.815	0.829	**0.821**
MLP	ALL	0.718	0.674	0.676	0.675
NB	ALL	0.573	0.619	0.607	0.569
3-NN	ALL	0.766	0.716	0.733	0.723
RF	ALL	0.806	0.746	0.793	0.761
SVM	ALL	0.782	0.669	0.847	0.685
ENS	ALL	0.782	0.735	0.753	0.742
J48	FEAT.SELEC.	0.839	0.802	0.821	**0.810**
MLP	FEAT.SELEC.	0.710	0.661	0.666	0.663
NB	FEAT.SELEC.	0.581	0.632	0.619	0.578
3-NN	FEAT.SELEC.	0.774	0.722	0.744	0.730
RF	FEAT.SELEC.	0.823	0.758	0.822	0.777
SVM	FEAT.SELEC.	0.774	0.657	0.842	0.669
ENS	FEAT.SELEC.	0.782	0.735	0.753	0.742
BASELINE	MAJ. CLASS	0.500	0.500	0.339	0.404
BASELINE	RANDOM	0.501	0.505	0.505	0.486

Sharing RapidMiner workflows and experiments with OpenML

Jan N. van Rijn[1] and Joaquin Vanschoren[2]

[1] Leiden University, Leiden, Netherlands,
j.n.van.rijn@liacs.leidenuniv.nl
[2] Eindhoven University of Technology, Eindhoven, Netherlands,
j.vanschoren@tue.nl

Abstract. OpenML is an online, collaborative environment for machine learning where researchers and practitioners can share datasets, workflows and experiments. While it is integrated in several machine learning environments, it was not yet integrated into environments that offer a graphical interface to easily build and experiment with many data analysis workflows. In this work we introduce an integration into the popular RapidMiner environment, that will allow RapidMiner users to import data directly from OpenML and automatically share all their workflows and experiments. OpenML will then link these results to all other results obtained by other people, possibly with other tools, creating a single connected overview of the best workflows on a large set of machine learning problems. This is useful to learn and build on the results of others, to collaborate with many people online, and it provides a wealth of information to study how to construct workflows for new machine learning problems. We demonstrate the capabilities of this integration and identify several research opportunities.

Keywords: Meta Learning, Workflows, Algorithm Selection

1 Introduction

The field of meta-learning studies which Machine Learning algorithms work well on what kind of data. The algorithm selection problem is one of its most natural applications [24]: given a dataset, identify which learning algorithm (and which hyperparameter setting) performs best on it. Different approaches leverage meta-learning in different ways, such as building predictive meta-models based on data characterizations [3, 20], iteratively testing the most promising algorithms [17] and model-based hyperparameter optimization [8]. However, all these solutions focus on recommending just a single algorithm.

Even-though the obtained results are very useful, it has been widely recognized that the quality of the results can be markedly improved by also selecting the right pre-processing and post-processing operators [7, 19, 32]. For example, the quality of k Nearest Neighbour algorithms typically degrades when the number of features increases [9], so it makes sense to combine these algorithms with feature selection [14] or feature construction. The complete chain of pre-processing operators, algorithms and post-processing operators is typically referred to as a *workflow*.

Meta-learning research is built on the premise that the relation between data and a good algorithm can be learned. For this, a key prerequisite is to have access to a vast amount of executed experiments to serve as historical training data. Experiment databases [34] have been designed to collect previously executed experiments, divesting researchers from the burden of executing these many experiments over and over again. OpenML [26, 35] is an online platform for machine learning where researchers and practitioners can share datasets, workflows and experiments *automatically* from many machine learning environments, and build directly on each other's results. Hence, it provides an exceedingly rich resource for meta-learning research. However, it currently has limited support for workflows.

One of the additional challenges is the enormous increase of the search space: besides finding the right (order of) operators, each operator also has its own parameters to be tuned. Currently, there is only little work that addresses the question whether the relation between the dataset and the complete workflow of pre-processing, modelling and post-processing operators can be learned.

In order to foster progress in this challenging research area, we have integrated OpenML into RapidMiner. RapidMiner is a data analysis environment that has a graphical interface for users to easily experiment with many (slightly) different workflows, as well as support for generating machine learning workflows. By means of this integration, the full set of RapidMiner workflows can be shared on OpenML, as well as the ensuing experimental results as these workflows are run and evaluated on many input datasets. Collecting this information in an organized fashion will open up novel research opportunities in meta-learning.

2 Related Work

The algorithm selection problem has attracted a lot of attention. In meta-learning approaches, the data is characterised by so-called meta-features, over which a model can be built [3]. Significant effort has been devoted to creating these features, and they typically fall in one of the following categories [31]: statistical, information theoretic or landmarker [20].

Another way to find an appropriate algorithm is by subsampling the data: when training a algorithm on a small subset of the data, the time to execute and evaluate an algorithm is much lower. The underlying assumption is that if a algorithm works well on a small subsample of the data, it also works well on more data. Much research has been done to study which sample sizes and techniques are appropriate to obtain a reliable model [15, 21].

Even though these techniques work well, it has been correctly observed that learning curves do cross [16]. Some algorithms perform particularly well when trained on large amounts of data. In [25], a partial learning curve is built on small data samples to iteratively select the most promising algorithms in a time-constrained setting, showing significant time savings.

However, it usually doesn't suffice to recommend a single algorithm. Typically, the algorithm contains many hyperparameters that need to be tuned, and the model might benefit from certain pre-processing and post-processing operators. Many strategies have been proposed to optimise the hyperparameters of a given algorithm, including gradient decent methods [4], Bayesian optimization techniques [33] and genetic algorithms [13, 22]. However, most of these approaches do not leverage historical information on the performance of hyperparameter settings on previously seen problems. One simple way to do this is to use meta-learning to build a model that recommends parameter settings [30], or to view multiple algorithm configurations as individual algorithms [17]. Other research leverages meta-learning to recommend when certain hyperparameters should be tuned, or to predict a good initial parameter setting to speed up Bayesian optimization [8].

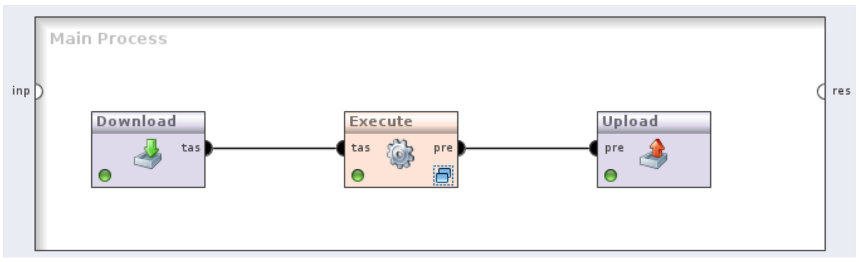
The task of selecting appropriate pre-processing and post-processing operators has been less studied in the literature. Case-based reasoning has been an early approach to select the most promising workflow out of a repository of previously successful workflows [10, 18]. Planning algorithms were also leveraged to construct and test possible workflows on the fly [1]. Most interestingly, the authors of [6, 12, 19] have independently from each other created a technique that exploits a meta-algorithm to predict what workflow to use. Their results suggests that the even the structure and operators of a workflow can be learned. In [7], the term *Full Model Selection Problem* was introduced, along with a particle swarm optimisation method to converge to a solution. The authors of [32] propose a framework in which these methods can be defined, along with particle swarm optimisation and genetic algorithm methods. Finally, graphical or other interfaces have emerged as a practical solution to manually construct and test many workflows, e.g. Weka [11], KNIME [2], ADAMS [23], and RapidMiner [28]. For a more complete overview of existing work in this area, see [29].

A common theme in this area is that a large body of workflows, and their evaluations on large numbers of datasets, is required for almost any of these methods to work well. In fact, it is often a limiting factor in demonstrating the practical applicability of these systems. By integrating OpenML and RapidMiner, it is our aim to work towards building the right infrastructure to foster large scale research in this direction.

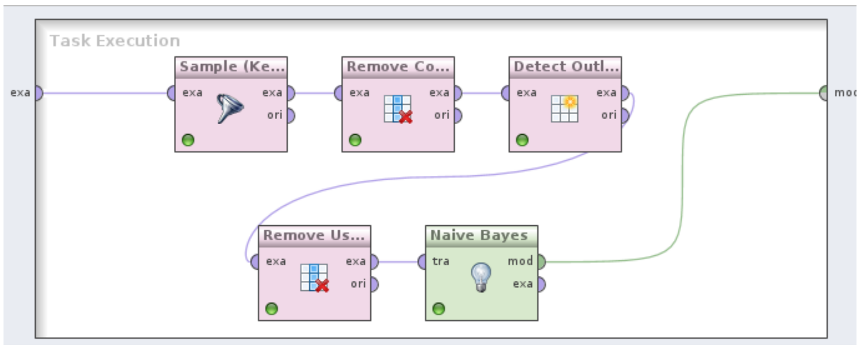
3 OpenML Connector

The integration[1] consists of three new RapidMiner operators: one for downloading OpenML tasks, one for executing them and one for uploading the results. Typically, they will be connected as shown in Figure 1(a). However, this modularization in three operators will likely be beneficial in some cases. The operators require an OpenML account to interact with the server.

[1] Available on `http://www.openml.org/`



(a) Main Workflow



(b) Subroutine solving OpenML task

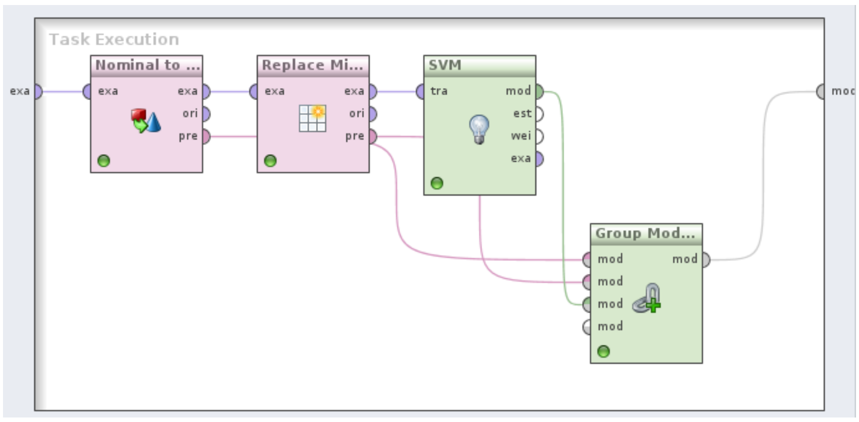
Fig. 1. Example of a RapidMiner workflow solving an OpenML task.

Download OpenML Task In order to make experiments reproducible, OpenML works with the concept of *tasks* [27, 35]. A task is a container that includes the input dataset(s), the data splits depending on the chosen evaluation procedure (e.g., cross-validation or holdout), and other necessary inputs. The "Download OpenML Task" operator downloads such tasks from OpenML and passes it to the output port.

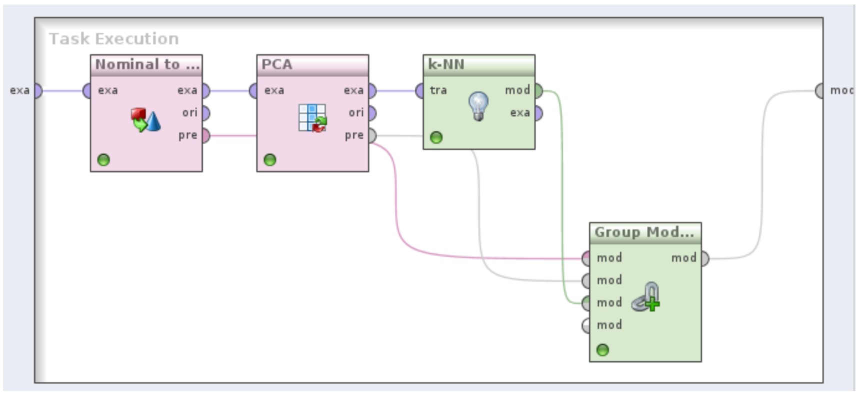
Execute OpenML Task The "Execute OpenML Task" is a so-called *super-operator*; it contains a sub-workflow that is expected to solve the task that is delivered at the input port. The subroutine is executed for each defined training set, and produces a model. This model is then used to predict the labels for the observations in the associated test set. An example of such a sub-workflow, including several pre-processing steps, is shown in Figure 1(b). The output of this super-operator is a data structure containing predictions for all instances in the test sets, and basic measurements such as run times.

Upload OpenML Task This operator uploads all relevant details of the workflow and the resulting predictions to OpenML. Details of the workflow are the set of all operators, and the parameter settings for each operators. The predictions contain the class label and confidences per class for classification tasks, or the predicted values for regression tasks. This enables OpenML to calculate all relevant performance measures, such as area under the ROC curve or RMSE. Also the run times for each fold are uploaded.

Example Workflows The main contribution of the RapidMiner plugin is that it automates the export of workflows to OpenML. Typically, a chain of operators is executed in order, possibly consisting of pre-processing operators that change the feature set. For example, the Support Vector Machine algorithm of RapidMiner can not operate on nominal features. In contrast to many WEKA algorithms, which automatically provide a workaround, in RapidMiner the work-

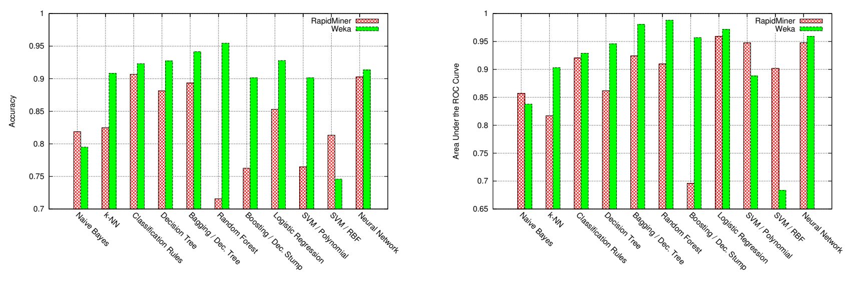


(a) SVM



(b) PCA / *k*-NN

Fig. 2. Subprocess of the Execute OpenML Task operator, combining various preprocessing models and the actual model using the "Group Models" operator.



(a) Accuracy (b) Area Under the ROC Curve

Fig. 3. Performance of RapidMiner and Weka algorithms on the "Spambase" dataset.

flow creator needs to define a solution. A possible solution could be to use the *Nominal to Numerical* operator. As this operator changes the feature set for each subsample of training set, the same pre-processing operations need to be performed on the test set. Figure 2 shows how the RapidMiner *Group Models* operator should be used to combine all pre-processing models and the algorithm model. This ensures that evaluation process is executed on the same features as constructed in the training set.

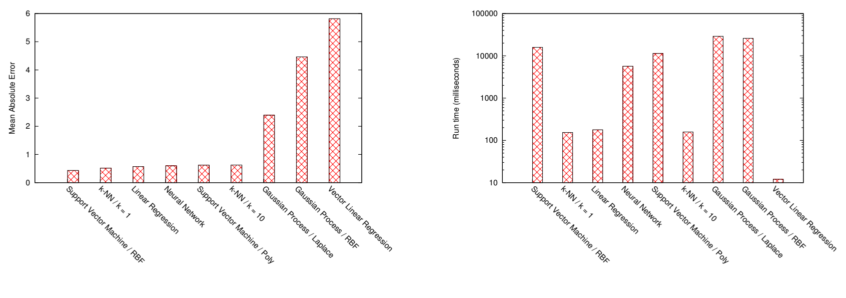
4 Research Opportunities

RapidMiner contains a large number of operators, covering a wide range of methods on various Machine Learning tasks. The OpenML integration thus opens up many research opportunities. We present some basic results obtained by using the RapidMiner plugin, and point out directions for future work.

Decent Machine Learning research is conducted over a wide range of datasets. Yet, the results we present cover only a very small number of datasets. Therefore, no real conclusions can be drawn from these experiments, and interesting patterns should be addressed in future work.

Classification We can now easily compare the performance of various RapidMiner algorithms against similar algorithms implemented in a different workbench, for example WEKA, because evaluations of other workbenches are already available on OpenML. Figure 3 shows the result of 11 algorithm implementations on the "spambase" dataset.All algorithms are executed with their respective default parameter settings. In the case of ensembles the base-algorithm is denoted; in case of Support Vector Machines, the kernel was denoted.

One surprising observation is that performance differs significantly between different implementations of the same basic algorithms. For example, even a fairly simple algorithm like Naive Bayes does not yield the same performance in both workbenches. In particular the difference in performance of the Random



(a) Mean Absolute Error (b) Run time

Fig. 4. Performance of regression algorithms on the "Wine Quality" dataset.

Forest and Boosting implementations of WEKA and RapidMiner is striking. Figure 3(a) shows the predictive accuracy of the algorithms. The results suggest that most WEKA algorithms are superior to their RapidMiner equivalents in terms of predictive accuracy. However, when measuring the Area Under the ROC Curve, most RapidMiner algorithms perform somewhat better, see for example the Support Vector Machines as shown in Figure 3(b).

The fact that two implementations of the same algorithm yield very different results can have various reasons. For example, different implementations can handle missing values differently, e.g., by replacing missing values by the mean of that attribute, or removing all observations that contain missing values.[2] If there is no parameter to control this behaviour, important aspects of model building are hidden from us. Finding these differences can therefore help us understand what kind of pre-processing steps are important for certain algorithms. Another possible explanation for why these results differ may be that the default parameter settings of the different implementations were optimized differently.

Regression RapidMiner also contains many algorithms than can perform regression tasks. In the next setup, we run some of these on the "Wine Quality" dataset [5].Figure 4 shows some results.

Figure 4(a) shows the *Mean Absolute Error* of all regression algorithms. There is a large group of algorithms with equivalent performance. Three algorithm perform eminently worse. Figure 4(b) shows run times. There seems no direct relation between good performance and higher run times.

It seems reasonable to assume that for regression tasks pre-processing steps become even more important, as many algorithms do not natively deal with irrelevant features (e.g., k-NN) or nominal values (e.g., Support Vector Machines). Fairly simple workflows can already make a big difference for regression tasks.

Learning Curves When running a algorithm on samples of increasing size, a learning curve can be constructed. These can be used to perform algorithm

[2] Note that the Spambase dataset used in Figure 3 has no missing features.

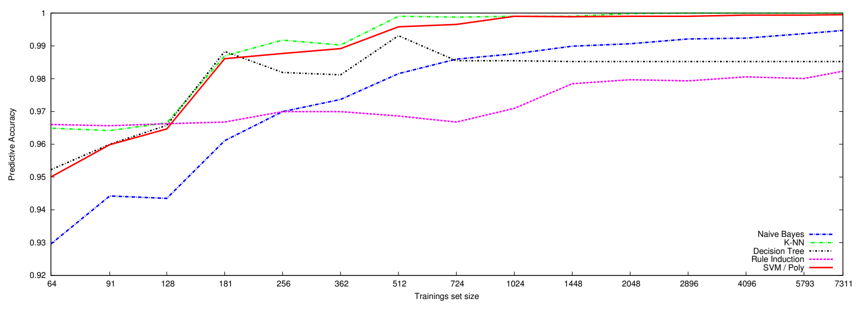


Fig. 5. Learning Curves of RapidMiner algorithms on the "Mushroom" dataset.

selection, as is done in [16, 25]. In [21] algorithm selection is used done by running algorithms on small samples of the data. The RapidMiner plugin can also operate on these data samples, and supports the creation of learning curves.

Figure 5 shows some of these curves. The x-axis shows the size of the sample, and the y-axis shows the performance on each sample. The curves behave as expected. In most cases, a larger sample size results in a higher accuracy. Furthermore, the curves do cross occasionally. An algorithm that performs well on a small sample is not necessarily competitive on larger samples, e.g., Rule Induction.

The Pairwise Comparison method described in [25] selects algorithms based on their performance on small samples. Enabling this method to operate on workflows would be a non-trivial but very useful extension.

Full Model Selection The authors of [7] first introduced this term, describing it as: given a set of pre-processing methods, feature selection algorithms and algorithms, select the combination of these that obtains the highest predictive accuracy for a given data set. The authors of [32] developed an uniform framework for solving this problem, and also came up with a competitive algorithm based on genetic algorithms. Workflows constructed in RapidMiner seem well fitted for this application, and the plugin introduced here could help with the experimentation and exploitation of such techniques.

Workflow Mining The authors of [19] propose a data mining advisor that attempts to extract knowledge from previously ran workflows to build new ones. Having access to a large repository of executed workflows gives the possibility of extracting knowledge about which components work well in combination with each other. By collecting a large set of workflow results in OpenML similar experiments can be conducted on large scale.

5 Conclusions

We have developed and presented an integration of the OpenML online collaboration platform within the RapidMiner workbench. OpenML currently contains over half a million experiments, yet few of those cover complex workflows. Recent work in meta-learning suggests that pre-processing and post-processing operators are an important part of successful algorithm selection solutions. It seems a logical next step to stimulate meta-learning and algorithm selection research on complete Machine Learning workflows, and this plugin is meant to enable and foster the automated sharing and collection of experiments that explore the performance of many possible workflows on many machine learning problems. Several opportunities for ongoing and future research have been identified, which can be pursued on an unprecedented scale through OpenML. We also hope to stimulate the integration of OpenML into other workbenches that deal with complex machine learning workflows.

Acknowledgments This work is supported by grant 600.065.120.12N150 from the Dutch Fund for Scientific Research (NWO).

References

1. Bernstein, A., Provost, F., Hill, S.: Toward Intelligent Assistance for a Data Mining Process: An Ontology-Based Approach for Cost-Sensitive Classification. Knowledge and Data Engineering, IEEE Transactions on 17(4), 503–518 (2005)
2. Berthold, M.R., Cebron, N., Dill, F., Gabriel, T.R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K., Wiswedel, B.: KNIME: The Konstanz Information Miner. In: Data Analysis, Machine Learning and Applications. pp. 319–326. Springer Berlin Heidelberg (2008)
3. Brazdil, P., Gama, J., Henery, B.: Characterizing the Applicability of Classification Algorithms using Meta-Level Learning. In: Machine Learning: ECML-94. pp. 83–102. Springer (1994)
4. Chapelle, O., Vapnik, V., Bousquet, O., Mukherjee, S.: Choosing Multiple Parameters for Support Vector Machines. Machine Learning 46(1-3), 131–159 (2002)
5. Cortez, P., Cerdeira, A., Almeida, F., Matos, T., Reis, J.: Modeling wine preferences by data mining from physicochemical properties. Decision Support Systems 47(4), 547–553 (2009)
6. Diamantini, C., Potena, D., Storti, E.: Mining Usage Patterns from a Repository of Scientific Workflows. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing. pp. 152–157. ACM (2012)
7. Escalante, H.J., Montes, M., Sucar, L.E.: Particle Swarm Model Selection. The Journal of Machine Learning Research 10, 405–440 (2009)
8. Feurer, M., Springenberg, T., Hutter, F.: Initializing Bayesian Hyperparameter Optimization via Meta-Learning. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence. pp. 1128–1135 (2015)
9. Friedman, J.H.: On Bias, Variance, 0/1-Loss, and the Curse-of-Dimensionality. Data Mining and Knowledge Discovery 1(1), 55–77 (1997)

10. Goble, C.A., De Roure, D.C.: myExperiment: Social Networking for Workflow-using e-Scientists. In: Proceedings of the 2nd workshop on Workflows in support of large-scale science. pp. 1–2. ACM (2007)
11. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. ACM SIGKDD Explorations Newsletter 11(1), 10–18 (2009)
12. Hilario, M., Nguyen, P., Do, H., Woznica, A., Kalousis, A.: Ontology-Based Meta-Mining of Knowledge Discovery Workflows. In: Meta-Learning in Computational Intelligence, pp. 273–315. Springer (2011)
13. Huang, C.L., Wang, C.J.: A GA-based feature selection and parameters optimization for support vector machines. Expert Systems with applications 31(2), 231–240 (2006)
14. Jain, A., Zongker, D.: Feature Selection: Evaluation, Application, and Small Sample Performance. Pattern Analysis and Machine Intelligence, IEEE Transactions on 19(2), 153–158 (1997)
15. John, G.H., Langley, P.: Static Versus Dynamic Sampling for Data Mining. In: In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. pp. 367–370. AAAI Press (1996)
16. Leite, R., Brazdil, P.: Predicting Relative Performance of Classifiers from Samples. In: Proceedings of the 22nd International Conference on Machine Learning. pp. 497–503. ACM (2005)
17. Leite, R., Brazdil, P., Vanschoren, J.: Selecting Classification Algorithms with Active Testing. In: Machine Learning and Data Mining in Pattern Recognition, pp. 117–131. Springer (2012)
18. Morik, K., Scholz, M.: The MiningMart Approach to Knowledge Discovery in Databases. In: Intelligent Technologies for Information Analysis, pp. 47–65. Springer (2004)
19. 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)
20. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Tell me who can learn you and I can tell you who you are: Landmarking various learning algorithms. In: Proceedings of the 17th International Conference on Machine Learning. pp. 743–750 (2000)
21. Provost, F., Jensen, D., Oates, T.: Efficient Progressive Sampling. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 23–32. ACM (1999)
22. Reif, M., Shafait, F., Dengel, A.: Meta-learning for evolutionary parameter optimization of classifiers. Machine learning 87(3), 357–380 (2012)
23. Reutemann, P., Vanschoren, J.: Scientific workflow management with ADAMS. In: Machine Learning and Knowledge Discovery in Databases, pp. 833–837. Springer (2012)
24. Rice, J.R.: The Algorithm Selection Problem. Advances in Computers 15, 65118 (1976)
25. van Rijn, J.N., Abdulrahman, S.M., Brazdil, P., Vanschoren, J.: Fast Algorithm Selection using Learning Curves. In: Advances in Intelligent Data Analysis XIV. Springer (2015)
26. van Rijn, J.N., Bischl, B., Torgo, L., Gao, B., Umaashankar, V., Fischer, S., Winter, P., Wiswedel, B., Berthold, M.R., Vanschoren, J.: OpenML: A Collaborative Science Platform. In: Machine Learning and Knowledge Discovery in Databases, pp. 645–649. Springer (2013)

27. van Rijn, J.N., Umaashankar, V., Fischer, S., Bischl, B., Torgo, L., Gao, B., Winter, P., Wiswedel, B., Berthold, M.R., Vanschoren, J.: A Rapidminer extension for Open Machine Learning. In: RCOMM 2013. pp. 59–70 (2013)
28. Ritthoff, O., Klinkenberg, R., Fischer, S., Mierswa, I., Felske, S.: Yale: Yet another learning environment. In: LLWA 01-Tagungsband der GI-Workshop-Woche, Dortmund, Germany. pp. 84–92 (2001)
29. Serban, F., Vanschoren, J., Kietz, J.U., Bernstein, A.: A Survey of Intelligent Assistants for Data Analysis. ACM Computing Surveys (CSUR) 45(3), 31:1–31:35 (2013)
30. Soares, C., Brazdil, P., Kuba, P.: A Meta-Learning Method to Select the Kernel Width in Support Vector Regression. Machine Learning 54(3), 195–209 (2004)
31. Sun, Q., Pfahringer, B.: Pairwise meta-rules for better meta-learning-based algorithm ranking. Machine Learning 93(1), 141–161 (2013)
32. Sun, Q., Pfahringer, B., Mayo, M.: Towards a Framework for Designing Full Model Selection and Optimization Systems. In: Multiple Classifier Systems, pp. 259–270. Springer (2013)
33. Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K.: Auto-WEKA: Combined selection and Hyperparameter Optimization of Classification Algorithms. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 847–855. ACM (2013)
34. Vanschoren, J., Blockeel, H., Pfahringer, B., Holmes, G.: Experiment databases. A new way to share, organize and learn from experiments. Machine Learning 87(2), 127–158 (2012)
35. 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)

Meta-QSAR: learning how to learn QSARs

Ivan Olier[1], Crina Grosan[2], Noureddin Sadawi[2], Larisa Soldatova[2], and Ross D. King[1]

[1] Manchester Institute of Biotechnology
University of Manchester, United Kingdom

[2] Department of Computer Science
University of Brunel, United Kingdom

1 Introduction

Quantitative structure activity relationships (QSARs) are functions that predict bioactivity from compound structure. Although almost every form of statistical and machine learning method has been applied to learning QSARs, there is no single best way of learning QSARs. Therefore, currently the QSAR scientist has little to guide her/him on which QSAR approach to choose for a specific problem.

The aim of this work is to introduce Meta-QSAR, a meta-learning approach aimed to learning which QSAR method is most appropriate for a particular problem. For the preliminary results presented here, we used ChEMBL[1], a public available chemoinformatic database, to systematically run extensive comparative QSAR experiments. We further apply meta-learning in order to generalise these results.

2 Data and Methods

The datasets involved in this research have been formed by computing molecular properties and fingerprints of chemical compounds with associated bioactivity to a particular target (protein). Learning a QSAR model consists on fitting a regression method to a dataset which has as input variables the descriptors, as response variable (output) the associated bioactivities, and as instances, the chemical compounds. We extracted 2,750 targets from ChEMBL with a very diverse number of chemical compounds, ranging from 10 to about 6,000. Two sets of properties – one, using 43 constitutional properties, and another, using 1,683 additional properties – and one fingerprint (FCFP4, 1024bits) were used to form the datasets. Further datasets were generated by imputing missing values using the median and performing feature selection based on the chi-squared test. For the QSAR methods, we have selected 20 algorithms typically used in QSAR experiments, which include: linear regression, support vector machines, artificial neural networks, regression trees, and random forest, amongst others. Model performance in all experiments has been assessed by taking

[1] ChEMBL database is available from: https://www.ebi.ac.uk/chembl/

the average root mean squared error (RMSE) after 10-fold crossvalidation of the datasets.

For the meta-learning stage, we conceived a classification problem that indicates which QSAR method should be used for a particular QSAR problem. The training and learning dataset is formed by meta-features extracted from the datasets of the base learning level and are based on target properties (hydrophobicity, molecular weight, aliphatic index, etc) and on information theory (mean, mutual information, entropy, etc). We used random forests as meta-learning algorithm.

3 Results and Discussion

Fig. 1 shows preliminary results of the experiments. The graph on the left confirms the hypothesis that there is no single way to learning QSARs. Random forests proofs successful for the FCFP4 fingerprint representation, although other QSAR methods had good performance, too. The graph on the right is an evidence of the fact that Meta-QSAR learning is correctly suggesting for almost all targets which QSAR method should be used.

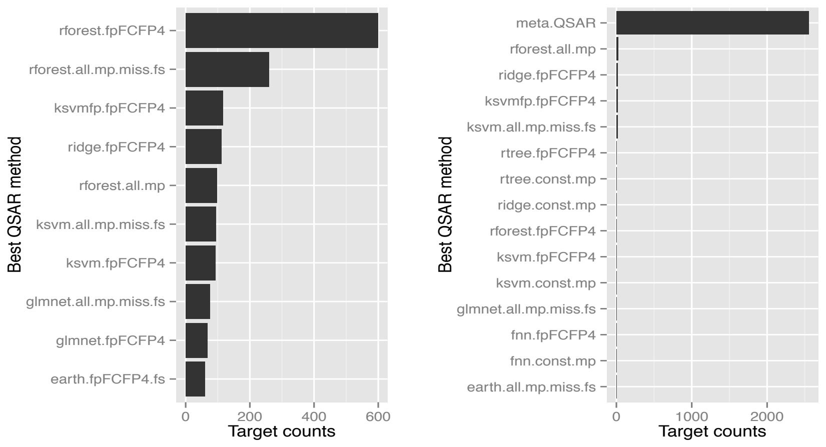


Fig. 1. Graphical representation of the number of times (target counts) a particular QSAR learning method obtains the best performance (minimum RMSE). Left: Results from the QSAR experiments. Right: Results using Meta-QSAR. The method names follow this convention: first term indicates the algorithm ('rforest', random forest; 'ksvm', support vector machine (SVM) with radial basis functions kernel; 'ksvmfp'; SVM with Tanimoto kernel; 'ridge', linear regression with ridge penalisation term; 'glmnet', elastic-net regularized generalised linear model; 'earth', multivariate adaptive regression splines; 'rtree', regression trees; 'fnn', fast k-Nearest Neighbour), second term, the kind of chemical compound descriptor set ('fpFCFP4', FCFP4 fingerprint; 'all.mp', full set of molecular properties; and 'const.mp', constitutional set of molecular properties), and then, optionally, whether missing value imputation ('miss') and feature selection ('fs') methods were used.

Concept of rule-based configurator for Auto-WEKA using OpenML

Patryk Kiepas, Szymon Bobek, and Grzegorz J. Nalepa

AGH University of Science and Technology,
al. A. Mickiewicza 30, 30-059 Krakow, Poland
kiepas@student.agh.edu.pl,{sbobek,gjn}@agh.edu.pl

Abstract. Despite a large amount of research devoted to improving meta-learning techniques, providing and using background knowledge for this task remains a challenge. In this paper we propose a mechanism for automatic recommendation of suitable machine learning algorithms and their parameters. We used OpenML database and use rule-based configurator to improve Auto-WEKA tool. This paper discusses the concept of our approach and the prototype tool based on the HEARTDROID rule engine being developed.

Introduction The objective of our work is to build a meta-learning recommendation system that guides a user through the process of solving a machine learning task. We use the data from OpenML's experiments to build a meta-knowledge which is later encoded with rules. This knowledge is then used for matching new dataset's meta-attributes with current meta-knowledge to obtain a set of possibly best algorithms. Finally, we use Auto-WEKA for optimizing the parameters of this narrowed set of algorithms.

In our approach we follow the general meta-learning architecture previously proposed by Pavel Brazdil et.al. [1]. We use data about machine learning from on-line collaborative platform known as OpenML[1]. In the creation of meta-knowledge we use the Amelia-II algorithm for imputation of missing data which could not be obtained with OpenML [2]. In rule-based configurator we take advantage of HEARTDROID inference engine[2]. Auto-WEKA does hyper-parameter optimization which we use for additional tuning of created recommendation [3].

We distinguish three phases in the recommendation mechanism: 1) knowledge acquisition, 2) recommendation, and 3) tuning. During the 1st phase meta-knowledge is built from OpenML's data only. In 2nd one the system uses that meta-knowledge and a new dataset to build a set of suitable algorithms. Finally an automatic configuration of these algorithms is performed with an usage of Auto-WEKA.

Building meta-knowledge In the acquisition phase main goal is to build meta-knowledge that describes dependecies between datasets and performance of machine learning algorithms executed on them. For every dataset in the OpenML database, a set of meta-attributes is available that includes: statistical information (e.g. number of classes and features, kurtosis of numeric attributes), information-theoretic characteristics (e.g. class

[1] http://www.openml.org/
[2] http://bitbucket.org/sbobek/heartdroid

or mean attribute entropy), and model-based information (e.g. J48 or kNN AUC). Each of such characteristics has a different non-missing value coverage that varies from 6.5% to 100%. We choose threshold for required values coverage to 20% to leave meaningful meta-attributes. Missing values are filled with Amelia-II algorithm [2].

Meta-knowledge combines meta-attributes from dataset characteristics with corresponding algorithm label or ranking. We choose only fixed number of algorithms that are taken into consideration (usually N top used in OpenML). After that we filter the results with respect to performance and leave only the set of best algorithms. Afterwards we consider meta-knowledge as labeled dataset. Using the WEKA J48 algorithm we create decision tree which is converts to the XTT2 rule representation (ang. *eXtended Tabular Trees*).

Making recommendation We start with computing meta-attributes of new dataset by uploading it to OpenML. Then we choose only characteristics used in created meta-knowledge. In the next step we match meta-attributes of new dataset with meta-knowledge. This is done with use of meta-rules and rule-based configurator. The result consist of algorithm name or ranking and set of parameters that according to the configurator fits best the given dataset.

In the third stage we reduce Auto-WEKA's search space only to the recommended algorithms. This is done by preparing experiment with so called XML-based *BATCH* file. In that file we fill path to our new dataset in *ARFF* format and set up list of allowed classifiers. Then we create an experiment and run optimization process. Result is in form of classifier name with single set of parameters.

Conclusion The main contribution of our work is a mechanism that allows to speed-up the meta-learning task by reducing search space for Auto-WEKA software with an usage of knowledge from OpenML database. We tested our approach and the tool on 570 datasets. We built meta-knowledge using 15 most used algorithms from OpenML focused on optimizing area under ROC. We benchmarked our best recommendations against Random Forest method as standard criteria. In general for most datasets area under ROC of our recommendations were higher (for 401 datasets with avg. 0.044). For 169 datasets AUC of our suggestions were lower (avg. 0.057). It is worth to notice that after a single setup, our system makes an instant recommendation.

Our future work includes learning and gaining additional meta-knowledge during recommendation mode, adding parameter suggestion in form of value ranges, adding guidance for data preprocessing methods and including more data sources.

References

1. Brazdil, P., Giraud-Carrier, C., Soares, C., Vilalta, R.: Meta-learning: Concepts and techniques. In: Metalearning: Applications to Data Mining. Springer Publishing Company, Incorporated, 1 edn. (2008)
2. Honaker, J., King, G., Blackwell, M.: Amelia II: A program for missing data. Journal of Statistical Software 45(7), 1–47 (12 2011), http://www.jstatsoft.org/v45/i07
3. Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: Proc. of KDD-2013. pp. 847–855 (2013)

Generating Workflow Graphs Using Typed Genetic Programming

Tomáš Křen[1], Martin Pilát[1], Klára Pešková[1], and Roman Neruda[2]

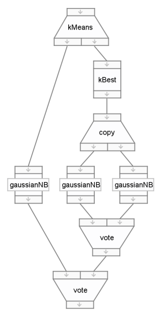
[1] Charles University in Prague, Faculty of Mathematics and Physics, Malostranské nám. 25, Prague, Czech Republic
[2] Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 18207 Prague, Czech Republic

In this paper we further develop our research line of chaining several preprocessing methods with classifiers [1], by generating the complete workflow schemes. These schemes, represented as directed acyclic graphs (DAGs), contain computational intelligence methods together with preprocessing algorithms and various methods of combining them into ensembles.

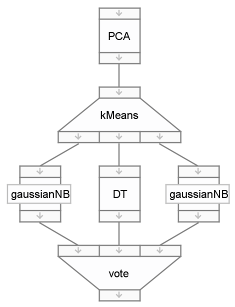
We systematically generate trees representing workflow DAGs using typed genetic programing initialization designed for polymorphic and parametric types. Terminal nodes of a tree correspond to the nodes of the DAG, where each node contains a computational intelligence method. Function nodes of a tree represent higher-order functions combining several DAGs in a serial or parallel manner. We use types to distinguish between input data (D) and predictions (P) so the generated trees represent meaningful workflows. In order to make the method general enough to handle methods like k-means (where k affects the topology of the DAG) correctly, we had to use the polymorphic type "*list of αs of size n*" ($[\alpha]_n$) with a natural number parameter n and an element type parameter α. The generating method systematically produces workflow DAGs from simple ones to more complex and larger ones, working efficiently with symmetries.

To demonstrate our first results we have chosen the winequality-white [2] and wilt [3] datasets from the UCI repository. They both represent medium size classification problems. The nodes of the workflow DAG contain three types of nodes; they can be *preprocessing* nodes (type $D \to D$) – k-Best (it selects k features most correlated with the target) or principal component analysis (PCA), or *classifier* nodes ($D \to P$) – gaussian naïve Bayes (gaussianNB), support vector classification (SVC), logistic regression (LR) or decision trees (DT). The last type of nodes implements *ensemble* methods – there is a copy node and a k-means node, which divides the data into clusters by the k-means algorithm (both $D \to [D]_n$), and two aggregating nodes – simple voting to combine the outputs of several methods, and merging for k-means node ($[P]_n \to P$).

To provide a baseline, we tested each of the four classifiers separately on the two selected datasets. The parameters of the classifiers were set using an extensive grid search with 5-fold cross-validation; the classifiers were compared using the quadratic weighted kappa metric. Next, we generated more than 65,000 different workflows using the proposed approach, and evaluated all of them. All computational intelligence methods used the default settings, or the tuned settings of the individual methods (denoted as 'default' or 'tuned' in Fig. 1c).



(a)



(b)

dataset	winequality		wilt	
params	default	tuned	default	tuned
SVC	0.1783	0.3359	0.0143	0.8427
LR	0.3526	0.3812	0.3158	0.6341
GNB	0.4202	0.4202	0.2916	0.2917
DT	0.3465	0.4283	0.7740	0.8229
workflow	**0.4731**	**0.4756**	**0.8471**	**0.8668**

(c)

Fig. 1: Best workflows for the winequality (a) and wilt (b) datasets, and comparison of κ metric from the cross-validation of the classifiers and the workflows (c).

The best workflows for the two datasets are presented in Figs. 1a and 1b, and their numerical results are presented in Fig.1c.

We have demonstrated how the valid workflow DAGs can be easily generated by a typed genetic programming initialization method. The generated workflows beat the baseline obtained by the hyper-parameter tuning of single classifier by a grid search, which is not surprising as the single method is also among the generated DAGs. On the other hand the workflows do not use any hyper-parameter tuning. In our future work, we will extend this approach to a full genetic programming solution, which will also optimize the hyper-parameters of the workflows and we intend to include the method in our multi-agent system for meta-learning – Pikater [4].

Acknowledgment

This work was supported by SVV project no. 260 224, GAUK project no. 187 115, and Czech Science Foundation project no. P103-15-19877S.

References

1. Kazík, O., Neruda, R.: Data mining process optimization in computational multi-agent systems. In: Agents and Data Mining Interaction. Volume 9145 of Lecture Notes in Computer Science. Springer (2015) 93–103
2. Cortez, P., Cerdeira, A., Almeida, F., Matos, T., Reis, J.: Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier **47**(4) (2009) 547–553
3. Johnson, B.A., Tateishi, R., Hoan, N.T.: A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees. Int. J. Remote Sens. **34**(20) (October 2013) 6969–6982
4. Pešková, K., Šmíd, J., Pilát, M., Kazík, O., Neruda, R.: Hybrid multi-agent system for metalearning in data mining. In Vanschoren, J., Brazdil, P., Soares, C., Kotthoff, L., eds.: Proc. of the MetaSel@ECAI 2014. Volume 1201 of CEUR Workshop Proceedings., CEUR-WS.org (2014) 53–54

Preference-Based Meta-Learning using Dyad Ranking: Recommending Algorithms in Cold-Start Situations (Extended Abstract)

Dirk Schäfer[1] and Eyke Hüllermeier[2]

[1] University of Marburg, Germany
[2] Department of Computer Science, University of Paderborn, Germany
dirk.schaefer@uni-marburg.de, eyke@upb.de

Preference learning in general and label ranking in particular have been applied successfully for meta-learning problems in the past [1, 4, 3]. The benefits of incorporating additional feature descriptions of alternatives in the context of preference learning have recently been shown for the *dyad ranking* framework [6]. Additional descriptions in the form of feature vectors are known in the recommender systems domain, too, where they are typically called *side-information* and used for tackling *cold-start problems*. These problems refer to situations where preference indicators (e.g., ratings) for new users or new items are not yet available (see Figure 1). In these situations, side-information helps by putting existing and new entities into relation. In this work, we make use

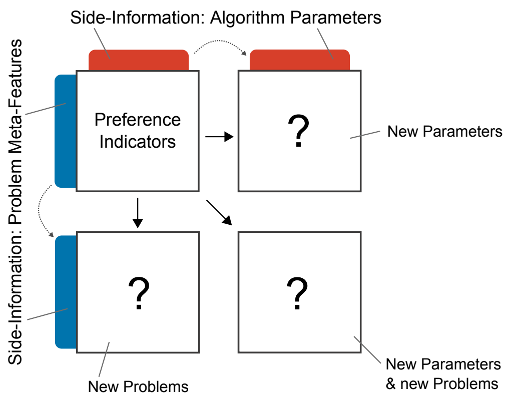

Side-Information: Algorithm Parameters
Side-Information: Problem Meta-Features
Preference Indicators
?
New Parameters
?
?
New Problems
New Parameters & new Problems

Fig. 1. Three kinds of cold-start problems are shown. They are characterized in that no preference indicators are available for algorithms or problems. Side-information can help in these situations for inferring preferences and thus recommendations.

of dyad ranking to predict a good ranking of candidate algorithms contextualized by problem instances, assuming that algorithms exhibit a representation in terms of a feature description. By generalizing over both, attributes of problems as well as algorithms, it becomes possible to tackle cold-start scenarios in which predictions are sought for algorithms that never occurred in the training data. A similar viewpoint towards meta-learning has been taken in [7, 5], where algorithm recommendation is tackled by means of collaborative filtering (CF) techniques. However, in contrast to the description of users and items in standard CF, side-information describing problems in meta-learning is usually carefully crafted [2]. As testbed, we present experimental results on the task of genetic algorithm (GA) recommendation in the cold-start situation corresponding to the lower right box in Figure 1. The (preference) meta-learning data set[3] for this experiment consists of rankings over 72 different parameterized GAs applied on the traveling salesman problem. The following leave-one-out cross validation (LOOCV) procedure over a total number of 246 examples (problems) and 72 GAs (referred to as labels) is applied: for a label A_j $(1 \leq j \leq 72)$ the bilinear Plackett-Luce model [6] is trained on 245 examples and is then used to predict the ranking over all 72 labels for the left out example in two variants.

In the first variant (the "reference" situation), a method is trained on data where the label A_j *is part* of the *training set*, whereas in the second variant (the "cold start" situation) the same method is trained on data where A_j is completely *omitted*. In addition to the Kendall τ value that is used to quantify the quality of a predicted ranking in relation to a ground truth ranking, the deviation between the predicted rank of A_j and the true rank is recorded.

In the reference and the cold start situation, the Kendall τ values are almost identical. Moreover, the average deviation from the true rank in the reference case is 5.653 and in the cold-start scenario 5.712. These are first encouraging results. Future work could comprise experiments on further meta data sets and address the development of further approaches for cold-start problems.

References

1. Artur Aiguzhinov, Carlos Soares, and Ana Paula Serra. A Similarity-based Adaption of Naive Bayes for Label Ranking: Application to Metalearning for Algorithm Selection. *Planning to Learn Workshop (PlanLearn10) at ECAI*, pages 75–78, 2010.
2. Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to Data Mining.* Springer Publishing Company, Incorporated, 1st edition, 2008.
3. Johannes Fürnkranz and Eyke Hüllermeier. *Preference Learning.* Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.
4. Jorge Kanda, Carlos Soares, Eduardo Hruschka, and Andre De Carvalho. A Meta-Learning Approach to Select Meta-Heuristics for the Traveling Salesman Problem Using MLP-Based Label Ranking. *19th International Conference on Neural Information Processing (ICONIP 2012)*, 7665 LNCS:488–495, 2012.
5. Mustafa Misir and Michèle Sebag. Algorithm Selection as a Collaborative Filtering Problem. Research report, INRIA, December 2013.
6. Dirk Schäfer and Eyke Hüllermeier. Dyad Ranking Using a Bilinear Plackett-Luce Model. In *Proceedings of the European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases.* Springer-Verlag, 2015.
7. David Stern, Horst Samulowitz, Luca Pulina, and Universita Genova. Collaborative Expert Portfolio Management. *Artificial Intelligence*, 116(3):179–184, 2010.

[3] Available at https://www.cs.uni-paderborn.de/fachgebiete/intelligente-systeme/

Towards a Collaborative Platform for Advanced Meta-Learning in Health care Predictive Analytics

Milan Vukicevic[1], Sandro Radovanovic[1], Joaquin Vanschoren[2], Giulio Napolitano[3], Boris Delibasic[1]

[1] University of Belgrade, Faculty of Organizational Sciences, Jove Ilica 154, Belgrade, Serbia
[2] Eindhoven University of Technology, Department of Mathematics and Computer Science, Eindhoven, Netherlands
[3] Bonn University, Germany

Modern medical research and clinical practice are more dependent than ever on multi-factorial data sets originating from various sources, such as medical imaging, DNA analysis, patient health records and contextual factors. This data drives research, facilitates correct diagnoses and ultimately helps to develop and select the appropriate treatments. The volume and impact of this data has increased tremendously through technological developments such as high-throughput genomics and high-resolution medical imaging techniques. Additionally, the availability and popularity of different wearable health care devices has allowed the collection and monitoring of fine-grained personal health care data. The fusion and combination of these heterogeneous data sources has already led to many breakthroughs in health research and shows high potential for the development of methods that will push current reactive practices towards predictive, personalized and preventive health care. This potential is recognized and has led to the development of many platforms for the collection and statistical analysis of health care data (e.g. Apple Health, Microsoft Health Vault, Oracle Health Management, Philips HealthSuite, and EMC Health care Analytics). However, the heterogeneity of the data, privacy concerns, and the complexity and multiplicity of health care processes (e.g. diagnoses, therapy control, and risk prediction) creates significant challenges for data fusion, algorithm selection and tuning. These challenges leave a *gap between the actual and the potential data usage* in health care, which prevents a paradigm shift from delayed generalized medicine to predictive personalized medicine [1]. As such, a platform for collaborative and privacy-preserving sharing, analysis and evaluation of health care data would drastically facilitate the creation of advanced models on heterogeneous fused data, as well as ensure the reproducibility of results, and provide a solid basis for the development of algorithm ranking and selection methods based on collaborative meta-learning.

In this work we present an extensions of the OpenML platform that will be addressed in our future work in order to meet the needs of meta-learning in health care predictive analytics: privacy preserving sharing of data, workflows and evaluations, reproducibility of the results, and rich meta-data spaces about both data and algorithms.

OpenML.org [2] is a collaboration platform which is designed to organize datasets, machine learning workflows, models and their evaluations. Currently, OpenML is not fully distributed but can be installed on local instances which can communicate with the main OpenML database using mirroring techniques. The downside of this approach is that code (machine learning workflows), datasets, experiments (models and evaluations) are physically kept on local instances, so users cannot communicate and share. We plan to turn OpenML into a fully distributed machine learning platform, which will be accessible from different data mining and machine learning platforms such as RapidMiner, R, WEKA, KNIME, or similar. Such a distributed platform would allow the ease of sharing data and knowledge. Currently, regulations and privacy concerns often prevent hospitals to learn from each other's approaches (e.g. machine learning workflows), reproduce work done by others (data version control, preprocessing and statistical analysis), and build models collaboratively.

On the other hand, meta-data such as type of the hospital, percentage of readmitted patients or indicator of emergency treatment, as well as the learned models and their evaluations can be shared and have great potential for the development of a cutting edge meta-learning system for ranking, selection and tuning of machine learning algorithms.

The success of meta-learning systems is greatly influenced by the size of problem (data) and algorithm spaces, but also by the quality of the data and algorithm descriptions (meta-features). Thus, we plan to employ domain knowledge provided by expert and formal sources (e.g. ontologies) in order to extend the meta-feature space for meta-learning in health care applications. For example, in meta-analyses of gene expression microarray data, the type of chip is very important in predicting algorithm performance. Further, in fused data sources it would be useful to know which type of data contributed to the performance (electronic health records, laboratory tests, data from wearables etc.). In contrast to data descriptions, algorithm descriptions are much less analyzed and applied in the meta-learning process. Recent results [3] showed that descriptions on the level of algorithm parts (e.g. initialization type and internal evaluation measures in clustering algorithms), could improve quality of meta-learning predictions, and additionally identify which algorithm parts really influenced the overall performance. Hence, we will include component based algorithm definitions as meta-features and allow their usage as predictors in meta-learning systems. The development of such a collaborative meta-learning system would address different challenging tasks in health care predictive analytics like early diagnostics and risk detection, hospital re-admission prediction, automated therapy control or similar with many potential stakeholders: patients, doctors, hospitals, insurance companies, among others.

Acknowledgement

This research was supported by SNSF Joint Research project (SCOPES), ID: IZ73Z0–152415.

References

[1] Olga Golubnitschaja, Judita Kinkorova, and Vincenzo Costigliola. Predictive, preventive and personalised medicine as the hardcore of horizon 2020: Epma position paper. *EPMA J*, 5(1):6, 2014.

[2] Joaquin Vanschoren, Jan N van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

[3] Milan Vukicevic, Sandro Radovanovic, Boris Delibasic, and Milija Suknovic. Extending meta-learning framework for clustering gene expression data with component based algorithm design and internal evaluation measures. *International Journal of Data Mining and Bioinformatics*, "In Press".

Study on Meta-Learning Approach Application in Rank Aggregation Algorithm Selection

Alexey Zabashta[1], Ivan Smetannikov[1], and Andrey Filchenkov[1]

[1] ITMO University, St. Petersburg, Russia
{zabashta}@rain.ifmo.ru, {smeivan, aaafil}@mail.ru

Abstract. Rank aggregation is an important task in many areas, nevertheless, none of rank aggregation algorithms is best for all cases. The main goal of this work is to develop a method, which for a given rank list finds the best rank aggregation algorithm with respect to a certain optimality criterion. Two approaches based on meta-feature description are proposed and one of them shows promising results.

Keywords: meta-learning, rank aggregation, permutations, algorithm selection.

1 Introduction

In many fields where multiple ranking algorithms are applied in practice such as computational biology, web search, or social choice, the important task of rank aggregation arises. A ranked list of objects is a permutation on these objects. Formally, the task of rank aggregation consists in finding a permutation π for a given permutation list Q, which minimizes the error function $E_\mu(\pi, Q)$, depending on a metric μ.

The problem of finding the best possible resulting rank is usually NP-hard, approximate algorithms are used, and they show different quality of results. Therefore, the problem of algorithm selection arises.

One of the possible solutions of this problem is the meta-learning approach [1]. Meta-learning systems were developed to solve different machine learning tasks, but to the best of our knowledge, the problem of rank aggregation algorithm selection has never been considered in scientific literature.

The main goal of this work is to develop an algorithm for rank aggregation algorithm selection. The proposed approach is based on meta-learning.

2 Algorithms and approaches

The *basic meta-features approach* (BMFA) for each μ-th metric looks over all possible pairs of permutations from the input permutation list Q and then constructs a sequence $X_\mu = \{\mu(a, b) | a, b \in Q\}$.

After that it mines statistic characteristics from each sequence X_μ as meta-features:

$$\text{features}(Q) = \bigcup_{\mu \in M} \begin{Bmatrix} \text{Min}(X_\mu) & \text{E}(X_\mu) & \text{Skew}(X_\mu) \\ \text{Max}(X_\mu) & \text{Var}(X_\mu) & \text{Kurt}(X_\mu) \end{Bmatrix},$$

where M is the set composed of the following seven metrics: described in [2, 3]: the *Manhattan distance*, the *Euclidean distance*, the *Chebyshev distance*, the *Cayley distance*, the *Kendall tau rank distance*, and the *Ulam distance*, the *Canberra distance*.

The *accelerated meta-features approach* (AMFA) aggregates the input permutation list into a single permutation c by means of the faster method — we use the Borda count. Then we construct a sequence X_μ from the distances between c and permutations from the input list $X_\mu = \{\mu(\pi, c) | \pi \in Q\}$. Then we mine features in the same way as in BMFA.

The *algorithm for Generating Permutation List* (AGPL) uses parameters α, β and we use three different algorithms for single permutation generation. The *Hidden Variable Approach* (HVA) describes a permutation list with α and β, with which it was generated. It can be applied only to generic data.

3 Experiments and results

In this paper we use four popular rank aggregation algorithms [2, 4]: *Borda count*, the *Copeland's Score*, the *Markov chain method*, and the "*Pick a perm"* method.

For experiments with generic data we generate permutation lists of the length 36 with 25 elements. For real-world experiments we use popular benchmark datasets for rank aggregation "LETOR4.0 MQ2007-agg".

Table 1 shows the F_1-measure for different classifiers built by the introduced approaches on three generic datasets and one real-world dataset. The table shows that the AMFA outperforms all the other approaches.

Table 1. Comparison of approaches by F_1-measure on the generic and real-work datasets.

	AGPL-A		AGPL-B		AGPL-C		Real-world	
	HVA	AMFA	HVA	AMFA	HVA	AMFA	BMFA	AMFA
IBk	0.534	**0.561**	0.400	**0.448**	0.485	**0.557**	0.367	**0.447**
J48	0.535	**0.545**	0.406	**0.413**	0.486	**0.548**	0.346	**0.413**
LogitBoost	0.529	**0.590**	0.401	**0.461**	0.487	**0.593**	0.399	**0.468**
NaiveBayes	0.522	**0.542**	**0.400**	0.393	0.476	**0.509**	**0.400**	0.396
SMO	0.386	**0.608**	0.382	**0.516**	0.431	**0.602**	0.407	**0.489**

4 Conclusion and future work

In this work we have proposed three approaches, and one of them, namely AMFA, has shown promising results on both types of on generic and real-world data. Based on this work we can conclude that meta-learning could be applied for best aggregation algorithm prediction, but current results may be improved.

In our future work we would try to: use more rank aggregation algorithm models and algorithms; apply feature selection algorithms; introduce wider generic data class, and test on other real-world data; create quality measure depending also on execution time and explore its behavior; predict best strategies for stochastic rank aggregation algorithms; create new meta-features, including task-specific meta-features.

This work was partially financially supported by the Government of Russian Federation, Grant 074-U01.

References

1. P. Brazdil, C. Giraud Carrier, C. Soares, R. Vilalta, *Metalearning. Applications to Data Mining*. Springer, 2009.
2. A. Burkovski, L. Lausser, J.M. Kraus, H.A. Kestler, *Rank Aggregation for Candidate Gene Identification*, M. Spiliopoulou et al. (eds.) Data Analysis, Machine Learning and Knowledge Discovery, Springer International Publishing, 285–293, (2014).
3. M. Deza, T. Huang, '*Metrics on Permutations, a Survey*', Journal of Combinatorics, Information and System Sciences, 23, 173–185, (1998).
4. C. Dwork, R. Kumar, M. Noar, D. Sivakumar, *Rank aggregation methods for the web*, 10th International Conf. on the World Wide Web, ACM Press and Addison Wesley, 613–622, (2001).

Author Index