META-LEARNING AND ALGORITHM SELECTION WORKSHOP AT ECAI 2014

MetaSel 2014

August 19, 2014

Prague, Czech Republic

Edited by

Joaquin Vanschoren, Pavel Brazdil, Carlos Soares 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 ECAI-2014 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 14 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:

- Lars Kotthoff: Towards an Algorithm Selection Standard: Data Format and Tools.
- Frank Hutter: Bayesian Optimization for More Automatic Machine Learning.

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

Eindhoven, July 2014

Joaquin Vanschoren
Pavel Brazdil
Carlos Soares
Lars Kotthoff

Main areas covered by the workshop

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

- Algorithm Selection and Configuration
- Planning to learn and construct workflows
- Applications of workflow planning
- Meta-learning and exploitation of meta-knowledge
- 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
- Experimentation and evaluation of learning processes
- Layered learning
- Multi-task and transfer learning
- Learning to learn
- Intelligent design
- Performance modeling
- Process mining

Program Committee

- Pavel Brazdil, LIAAD-INESC Porto L.A. / FEP, University of Porto, Portugal
- André C. P. Carvalho, USP, Brasil
- Claudia Diamantini, UniversitĂ Politecnica delle Marche, Italy
- Johannes Fuernkranz, TU Darmstadt, Germany
- Christophe Giraud-Carrier, Brigham Young Univ., USA
- Krzysztof Grabczewski, Nicolaus Copernicus University, Poland
- Melanie Hilario, Switzerland
- Frank Hutter, University of Freiburg, Germany
- Christopher Jefferson, University of St Andrews, UK
- Alexandros Kalousis, U Geneva, Switzerland
- Jörg-Uwe Kietz, U.Zurich, Switzerland
- Lars Kotthoff, University College Cork, Ireland
- Yuri Malitsky, University College Cork, Ireland
- Bernhard Pfahringer, U Waikato, New Zealand
- Vid Podpecan, Jozef Stefan Institute, Slovenia
- Ricardo Prudêncio, Univ. Federal de Pernambuco Recife (PE), Brasil
- Carlos Soares, FEP, University of Porto, Portugal
- Guido Tack, Monash University, Australia
- Joaquin Vanschoren, Eindhoven University of Technology
- Ricardo Vilalta, University of Houston, USA
- Filip Zelezný, CVUT, Prague, R.Checa

Table of Contents

Towards an algorithm selection standard: data format and tools

Lars Kotthoff[1]

The Algorithm Selection Problem is attracting increasing attention from researchers and practitioners from a variety of different backgrounds. After decades of fruitful applications in a number of domains, a lot of data has been generated and many approaches tried, but the community lacks a standard format or repository for this data. Furthermore, there are no standard implementation tools. This situation makes it hard to effectively share and compare different approaches and results on different data. It also unnecessarily increases the initial threshold for researchers new to this area.

In this talk, I will first give a brief introduction to the Algorithm Selection Problem and approaches to solving it [4, 3]. Then, I will present a standardized format for representing algorithm selection scenarios and a repository that contains a growing number of data sets from the literature, Aslib [1]. The format has been designed to be able to express a wide variety of different scenarios. In addition to encoding instance features and algorithm performances, there are facilities for providing feature costs, the status of algorithm execution and feature computations, cross-validation splits and meta-information. In addition to the data format itself, there is an R package that implements parsers and basic analysis tools. I will illustrate its usage through a series of examples.

I will further present LLAMA [2], a modular and extensible toolkit implemented as an R package that facilitates the exploration of a range of different portfolio techniques on any problem domain. It implements the algorithm selection approaches most commonly used in the literature and leverages the extensive library of machine learning algorithms and techniques in R. I will provide an overview of the architecture of LLAMA and the current implementation.

Leveraging the standard data format and the LLAMA toolkit, I will conclude this talk by presenting a set of example experiments that build and evaluate algorithm selection models. The models are created and evaluated with LLAMA on the problems in the algorithm selection benchmark repository. The results demonstrate the potential of algorithm selection to achieve significant performance improvements even through straightforward application of existing techniques.

Together, Aslib and LLAMA provide a low-threshold starting point for researchers wanting to apply algorithm selection to their domain or prototype new approaches. Both are under active development.

Joint work with Bernd Bischl, Pascal Kerschke, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren.

[1] University College Cork, larsko@4c.ucc.ie

REFERENCES

[1] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren, 'Aslib: A benchmark library for algorithm selection', *under review for AIJ*, (2014).

[2] Lars Kotthoff, 'LLAMA: leveraging learning to automatically manage algorithms', Technical Report arXiv:1306.1031, arXiv, (June 2013).

[3] Lars Kotthoff, 'Algorithm selection for combinatorial search problems: A survey', *AI Magazine*, (2014).

[4] Lars Kotthoff, Ian P. Gent, and Ian Miguel, 'An evaluation of machine learning in algorithm selection for search problems', *AI Communications*, **25**(3), 257–270, (2012).

Bayesian Optimization for More Automatic Machine Learning

(extended abstract for invited talk)

Frank Hutter[1]

Bayesian optimization (see, e.g., [2]) is a framework for the optimization of expensive blackbox functions that combines prior assumptions about the shape of a function with evidence gathered by evaluating the function at various points. In this talk, I will briefly describe the basics of Bayesian optimization and how to scale it up to handle structured high-dimensional optimization problems in the sequential model-based algorithm configuration framework SMAC [6].

Then, I will discuss applications of SMAC to two structured high-dimensional optimization problems from the growing field of *automatic machine learning*:

- Feature selection, learning algorithm selection, and optimization of its hyperparameters are crucial for achieving good performance in practical applications of machine learning. We demonstrate that a combined optimization over all of these choices can be carried out effectively by formulating the problem of finding a good instantiation of the popular WEKA framework as a 768-dimensional optimization problem. The resulting Auto-WEKA framework [7] allows non-experts with some available compute time to achieve state-of-the-art learning performance on the push of a button.
- Deep learning has celebrated many recent successes, but its performance is known to be very sensitive to architectural choices and hyperparameter settings. Therefore, so far its potential could only be unleashed by deep learning experts. We formulated the combined problem of selecting the right neural network architecture and its associated hyperparameters as a 81-dimensional optimization problem and showed that an automated procedure could find a network whose performance exceeded the previous state-of-the-art achieved by human domain experts using the same building blocks [3]. Computational time remains a challenge, but this result is a step towards deep learning for non-experts.

To stimulate discussion, I will finish by highlighting several further opportunities for combining meta-learning and Bayesian optimization:

- Prediction of learning curves [3],
- Learning the importance of hyperparameters (and of meta-features) [4, 5], and
- Using meta-features to generalize hyperparameter performance across datasets [1, 8], providing a prior for Bayesian optimization.

Based on joint work with Tobias Domhan, Holger Hoos, Kevin Leyton-Brown, Jost Tobias Springenberg, and Chris Thornton.

[1] University of Freiburg, Germany. Email: fh@cs.uni-freiburg.de.

REFERENCES

[1] R. Bardenet, M. Brendel, B. Kgl, and M. Sebag, 'Collaborative hyperparameter tuning', in *Proc. of ICML*, (2013).
[2] E. Brochu, V. M. Cora, and N. de Freitas, 'A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning', *CoRR*, **abs/1012.2599**, (2010).
[3] Tobias Domhan, Tobias Springenberg, and Frank Hutter, 'Extrapolating learning curves of deep neural networks', in *ICML 2014 AutoML Workshop*, (June 2014).
[4] F. Hutter, H. Hoos, and K. Leyton-Brown, 'Identifying key algorithm parameters and instance features using forward selection', in *Learning and Intelligent Optimization*, pp. 364–381, (2013).
[5] F. Hutter, H. Hoos, and K. Leyton-Brown, 'An efficient approach for assessing hyperparameter importance', in *International Conference on Machine Learning*, (2014).
[6] F. Hutter, H. H. Hoos, and K. Leyton-Brown, 'Sequential model-based optimization for general algorithm configuration', in *Proc. of LION-5*, (2011).
[7] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, 'Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms', in *Proc. of KDD'13*, (2013).
[8] D. Yogatama and G. Mann, 'Efficient transfer learning method for automatic hyperparameter tuning', in *Proc. of AISTATS*, (2014).

Using Meta-Learning to Initialize Bayesian Optimization of Hyperparameters

Matthias Feurer and **Jost Tobias Springenberg** and **Frank Hutter**[1]

Abstract. Model selection and hyperparameter optimization is crucial in applying machine learning to a novel dataset. Recently, a subcommunity of machine learning has focused on solving this problem with Sequential Model-based Bayesian Optimization (SMBO), demonstrating substantial successes in many applications. However, for expensive algorithms the computational overhead of hyperparameter optimization can still be prohibitive. In this paper we explore the possibility of speeding up SMBO by transferring knowledge from previous optimization runs on similar datasets; specifically, we propose to initialize SMBO with a small number of configurations suggested by a metalearning procedure. The resulting simple MI-SMBO technique can be trivially applied to any SMBO method, allowing us to perform experiments on two quite different SMBO methods with complementary strengths applied to optimize two machine learning frameworks on 57 classification datasets. We find that our initialization procedure mildly improves the state of the art in low-dimensional hyperparameter optimization and substantially improves the state of the art in the more complex problem of combined model selection and hyperparameter optimization.

1 Introduction

Hyperparameter optimization is a crucial step in the process of applying machine learning algorithms in practice. Depending on the training time of the algorithm at hand, finding good hyperparameter settings manually is often a time-consuming, tedious process requiring many ad-hoc choices by the practitioner. As a result, much recent work in machine learning has focused on the development of better hyperparameter optimization methods [14, 3, 29, 4, 26, 21, 10, 33, 5].

Recently, Sequential Model-based Bayesian Optimization (SMBO) [16, 7, 14] has emerged as a successful hyperparameter optimization method in machine learning. It has been conclusively shown to yield better performance than both grid and random search [3, 29, 33, 9]. In practical applications, it has unveiled new state-of-the-art performance on the challenging CIFAR-10 object recognition benchmark by tuning the hyperparameters of a deep neural network [29] and was repeatedly found to match or outperform human practitioneers in tuning complex neural network models [3] as well as computer vision architectures with up to 238 hyperparameters [5]. It has also enabled AutoWEKA [33], which performs combined algorithm selection and hyperparameter optimization in the space of algorithms defined by the WEKA package [11]. We describe SMBO in detail in Section 2.

However, SMBO is defined as a generic function optimization framework, and—like any other generic optimization method—it requires a substantial number of function evaluations to detect high-performance regions when started on a new optimization problem. The resulting overhead is computationally infeasible for expensive-to-evaluate machine learning algorithms. To combat this problem, metalearning has been applied in two ways. Firstly, a method similar to SMBO that reasons across datasets has been developed[19]. Secondly, metalearning was used to initialize hyperparameter optimization methods with hyperparameter configurations that previously yielded good performance on similar datasets [26, 21, 10]. We follow this latter approach to yield a simple and effective initialization procedure that applies generically to all variants of SMBO; we refer to the resulting SMBO approach with Meta-learning-based Initialization as MI-SMBO. In contrast to another recent line of work on collaborative SMBO methods [1, 35, 32], MI-SMBO does not require any adaptation of the underlying SMBO procedure. It is hence easy to implement and can be readily applied to several off-the-shelf hyperparameter optimizers.

Using a comprehensive suite of 57 datasets and 46 metafeatures, we empirically studied the impact of our meta-learning-based initialization procedure to two SMBO variants with complementary strengths. First, we applied it to optimize the 2 hyperparameters C and γ of a support vector machine (SVM), which control the SVM's learning process. Here, our MI-Spearmint variant of the Gaussian-process-based SMBO method Spearmint [29] (a state-of-the-art approach for low-dimensional hyperparameter optimization) yielded mild improvements: in particular, MI-Spearmint performed better than Spearmint initially, but after 50 function evaluations the differences levelled off. Second, we applied our method to optimize 10 hyperparameters describing a choice between three classifiers from the prominent Scikit-Learn package [22] and their hyperparameters. Here, our MI-SMAC variant of the random-forest-based SMBO method SMAC [14] (a state-of-the-art approach for high-dimensional hyperparameter optimization) yielded substantial improvements, significantly outperforming the previous state of the art for this problem. To enable other researchers to reproduce and build upon our results, we will provide our software on the first author's github page.[2]

2 Foundations

Before we describe our MI-SMBO approach in detail we formally describe hyperparameter optimization and SMBO.

2.1 Hyperparameter Optimization

Let $\theta_1, \ldots, \theta_n$ denote the hyperparameters of a machine learning algorithm, and let $\Theta_1, \ldots, \Theta_n$ denote their respective domains. The al-

[1] University of Freiburg, Germany, {feurerm,springj,fh}@cs.uni-freiburg.de

[2] https://github.com/mfeurer

Algorithm 1: Generic Sequential Model-based Optimization. SMBO(f^D, T, $\boldsymbol{\Theta}$, $\boldsymbol{\theta}_{1:t}$)

Input: Target function f^D; limit T; hyperparameter space $\boldsymbol{\Theta}$; initial design $\boldsymbol{\theta}_{1:t} = \langle \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_t \rangle$
Result: Best hyperparameter configuration $\boldsymbol{\theta}^*$ found

```
for i ← 1 to t do  y_i ← Evaluate f^D(θ_i)
for j ← t + 1 to T do
    M ← fit model on performance data ⟨θ_i, y_i⟩_{i=1}^{j-1}
    Select θ_j ∈ arg max_{θ∈Θ} a(θ, M)
    y_j ← Evaluate f^D(θ_j)
return θ* ∈ arg min_{θ_j ∈ {θ_1,...,θ_T}} y_j
```

gorithm's hyperparameter space is then defined as $\boldsymbol{\Theta} = \Theta_1 \times \cdots \times \Theta_n$. When trained with $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ on data $\mathcal{D}_{\text{train}}$, we denote the algorithm's validation error on data $\mathcal{D}_{\text{valid}}$ as $\mathcal{V}(\boldsymbol{\theta}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}})$. Using k-fold cross-validation, the hyperparameter optimization problem for a given dataset D then is to minimize:

$$f^D(\boldsymbol{\theta}) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{V}(\boldsymbol{\theta}, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)}). \tag{1}$$

Hyperparameters θ_i can be numerical (real or integer, as, e.g., the strength of a regularizer) or categorical (unordered, with finite domain, as, e.g., the choice between different kernels). Furthermore, there can be *conditional* hyperparameters, which are only active if another hyperparameter takes a certain value; for example, the hyperparameter "number of principal components" only needs to be instantiated when the hyperparameter "preprocessing method" is set to PCA.

The space of hyperparameter configurations can be searched either manually or automatically. Since manual search is tedious, time-consuming, and often not sample-efficient, much recent work in machine learning has focused on the development of automated methods. Grid search, the most frequently used automated method, does not scale to high-dimensional hyperparameter spaces, and has been shown to be outperformed by random search in the presence of low effective dimensionality [3]. Various types of direct search have been applied to the problem as well, such as genetic algorithms [26], particle swarm optimization [21], and tabu search [10]. Most recently, several SMBO algorithms have been presented for hyperparameter optimization [14, 3, 29]; we discuss these in the following section.

2.2 Sequential Model-based Bayesian Optimization

Sequential Model-based Bayesian Optimization (SMBO) [16, 7, 14] is a powerful method for the global optimization of expensive black-box functions f. As described in Algorithm 1, SMBO starts by querying the function f at the t values in an initial design and recording the resulting ⟨input, output⟩ pairs $\langle \boldsymbol{\theta}_i, f(\boldsymbol{\theta}_i) \rangle_{i=1}^{t}$. Afterwards, it iterates the following three phases: (1) fit a probabilistic model $\mathcal{M}$ to the ⟨input, output⟩ pairs collected so far; (2) use the probabilistic model $\mathcal{M}$ to select a promising input $\boldsymbol{\theta}$ to evaluate next by quantifying the desirability of obtaining the function value at arbitrary inputs $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ through a so-called acquisition function $a(\boldsymbol{\theta}, M)$; (3) evaluate the function at the new input $\boldsymbol{\theta}$.

The SMBO framework offers several degrees of freedom to be instantiated, including the procedure's initialization, the type of probabilistic model to use, and the acquisition function. We discuss three prominent hyperparameter optimization methods in terms of these components: SMAC [14], Spearmint [29], and TPE [3].

The role of the acquisition function $a(\boldsymbol{\theta}, \mathcal{M})$ is to trade off *exploration* in hyperparameter regions where the model $\mathcal{M}$ is uncertain with *exploitation* in regions with low predicted validation error. The most commonly-used acquisition function is the Expected positive improvement (EI) over the best input found so far [16]:

$$a_{EI}(\boldsymbol{\theta}, \mathcal{M}) = \int_{-\infty}^{\infty} \max(y^* - y, 0) p_{\mathcal{M}}(y|\boldsymbol{\theta}) d_y. \tag{2}$$

Other prominent acquisition functions are Upper Confidence Bounds [31] and Entropy Search [12]. All of SMAC, Spearmint, and TPE use the expected improvement criterion.

Several different model types can be used inside of SMBO. The most popular choice, used for example by Spearmint, are Gaussian processes [24] because they provide good predictions in low-dimensional numerical input spaces and allow the computation of the posterior Gaussian process model in closed form. The other popular model type are tree-based approaches, which are particularly well suited to handle high-dimensional input spaces and partially categorical input spaces. In particular, SMAC uses random forests [6], modified to yield an uncertainty estimate [15]. Random forests are particularly well suited for SMBO in high dimensions due to their robustness and automated feature selection. Another tree-based approach, applied by TPE, is to use the Tree Parzen Estimator [3] in order to construct a density estimate over good and bad instantiations of each hyperparameter: instead of predicting $p(y \mid \boldsymbol{\theta})$ directly, TPE constructs estimates of $p(\boldsymbol{\theta} \mid y \geq q)$ and $p(\boldsymbol{\theta} \mid y < q)$ for a given quantile q. Expected improvement can then be shown to be proportional to $\frac{p(\boldsymbol{\theta}|y>q)}{p(\boldsymbol{\theta}|y<q)}$.

The final degree of freedom in SMBO is its initialization. To date, this component has not received much attention, and is instantiated in a fairly ad-hoc manner: Spearmint evaluates f at two pre-defined input points, SMAC evaluates it at a pre-defined 'default' input, and TPE evaluates 20 points selected at random according to a user-defined prior distribution. It is this initialization procedure that our MI-SMBO approach aims to improve.

An empirical evaluation of Bayesian hyperparameter optimization methods in the framework of the hyperparameter optimization library (HPOlib [9]) has shown Spearmint to yield the best results for low-dimensional continuous hyperparameter optimization problems, and SMAC to perform best for high-dimensional hyperparameter optimization problems and problems with categorical and/or conditional hyperparameters.

3 Initializing SMBO With Configurations Suggested by Meta-Learning

Building on the foundations from Section 2 we will now describe our proposed MI-SMBO method that uses meta-learning to initialize SMBO.

The core idea behind MI-SMBO is to follow the common practice machine learning experts employ when applying a known machine learning method to a new dataset D_{new}: they first study D_{new}, relating it to datasets they previously experienced. When manually optimizing hyperparameters for D_{new}, they would begin the search with hyperparameter configurations that were optimal for the most similar previous datasets. Our MI-SMBO method automates this approach and uses it to initialize an SMBO method. In addition to eliminating the need for manual exploration, this can lead to better results as more time can be spend on improving known configurations. We note that in settings where only a few performance evaluations of the algorithm to be optimized are feasible using additional information

Algorithm 2: SMBO with Meta-Learning Initialization. MI-SMBO(D_{new}, $f^{D_{\text{new}}}$, $D_{1:N}$, $\hat{\boldsymbol{\theta}}^{1:N}$, d, t, T, $\boldsymbol{\Theta}$)

Input: new dataset D_{new}; target function $f^{D_{\text{new}}}$; training datasets $D_{1:N} = (D_1, \ldots, D_N)$; best configurations for training datasets, $\hat{\boldsymbol{\theta}}^{1:N} = \hat{\boldsymbol{\theta}}^1, \ldots, \hat{\boldsymbol{\theta}}^N$; distance metric d; number of configurations to include in initial design, t; limit T; hyperparameter space $\boldsymbol{\Theta}$

Result: Best hyperparameter configuration $\boldsymbol{\theta}^*$ found

Sort dataset indices $\pi(1), \ldots \pi(N)$ by increasing distance to D_{new}, i.e.: $(\pi(i) \leq \pi(j)) \Leftrightarrow (d(D_{\text{new}}, D_i) \leq d(D_{\text{new}}, D_j))$

for $i \leftarrow 1$ **to** t **do** $\boldsymbol{\theta}_i \leftarrow \hat{\boldsymbol{\theta}}^{\pi(i)}$

$\boldsymbol{\theta}^* \leftarrow$ SMBO(f^D, T, $\boldsymbol{\Theta}$, $\boldsymbol{\theta}_{1:t}$)

return $\boldsymbol{\theta}^*$

from other datasets might be the only possibility to achieve reasonable performance.

Formally, MI-SMBO can be stated as follows. Let $\hat{\boldsymbol{\theta}}^1, \ldots, \hat{\boldsymbol{\theta}}^N$ denote the best known hyperparameters for the previously encountered datasets $D_1, \ldots, D_N$, respectively. These may originate from an arbitrary source, e.g., a manual search or the application of an SMBO method during an offline training phase. Further, let D_{new} denote a new dataset, let d denote a distance metric between datasets, and let π denote a permutation of $(1, \ldots, N)$ sorted by increasing distance between D_{new} and D_i (i.e., $(\pi(i) \leq \pi(j)) \Leftrightarrow (d(D_{\text{new}}, D_i) \leq d(D_{\text{new}}, D_j))$). Then, MI-SMBO with an initial design of t configurations initializes SMBO with configurations $\hat{\boldsymbol{\theta}}^{\pi(1)}, \ldots, \hat{\boldsymbol{\theta}}^{\pi(t)}$. Algorithm 2 provides pseudocode for the approach.

We would like to highlight the fact that MI-SMBO is agnostic of the SMBO algorithm used, as long as the algorithm's implementation accepts an initial design as input or can be warmstarted with a given list of performance data $\langle \boldsymbol{\theta}_i, y_i \rangle_{i=1}^t$. All of SMAC, TPE, and Spearmint fulfill these criteria. We would also like to highlight that SMBO is a particularly good match for initialization with meta-learning: in contrast to existing approaches that initialize other types of hyperparameter optimization algorithms via metalearning [10, 21, 26], SMBO can make effective use of all performance data it receives as input (and does not have to adapt population sizes or alike to the size of the initial design).

To implement MI-SMBO, we still need to define a distance metric between datasets. This is a well studied problem which was, to our knowledge, first discussed by Soares and Brazdil [30]. For the purpose of this work we assume that each dataset D_i can be described through a set of F metafeatures $\boldsymbol{m}^i = (m_1^i, \ldots, m_F^i)$. We discuss the metafeatures we used in Section 3.1. In practice, we precompute the metafeatures for all training datasets $D_1, \ldots, D_N$ along with the best configurations $(\hat{\boldsymbol{\theta}}^1, \ldots, \hat{\boldsymbol{\theta}}^N)$. We then measure the distance between a new dataset D_{new} and a previous dataset D_i as the norm of the distance between their metafeatures:

$$d(D_{\text{new}}, D_j) = \|\boldsymbol{m}^{\text{new}} - \boldsymbol{m}^j\|. \quad (3)$$

3.1 Implemented Metafeatures

To evaluate our approach in a realistic setting we implemented the 46 metafeatures from the literature listed in Table 1. Based on their types and underlying assumptions, these metafeatures can be divided into at least five groups:

Table 1. List of implemented metafeatures

Simple metafeatures:	**Statistical metafeatures:**
number of patterns	min # categorical values
log number of patterns	max # categorical values
number of classes	mean # categorical values
number of features	std # categorical values
log number of features	total # categorical values
number of patterns with missing values	kurtosis min
percentage of patterns with missing values	kurtosis max
number of features with missing values	kurtosis mean
percentage of features with missing values	kurtosis std
number of missing values	skewness min
percentage of missing values	skewness max
number of numeric features	skewness mean
number of categorical features	skewness std
ratio numerical to categorical	
ratio categorical to numerical	**PCA metafeatures:**
dataset dimensionality	pca 95%
log dataset dimensionality	pca skewness first pc
inverse dataset dimensionality	pca kurtosis first pc
log inverse dataset dimensionality	
class probability min	**Landmarking metafeatures:**
class probability max	One Nearest Neighbor
class probability mean	Linear Discriminant Analysis
class probability std	Naive Bayes
	Decision Tree
Information-theoretic metafeature:	Decision Node Learner
class entropy	Random Node Learner

- *Simple metafeatures*, such as the number of features, patterns or classes, describe the basic dataset structure [20, 17, 1, 35].
- *PCA metafeatures* [1] perform principal component analysis and compute various statistics of the principal components.
- The *information-theoretic metafeature* measures the class entropy in the data [20].
- *Statistical metafeatures* [20] attempt to characterize the data distribution via descriptive statistics such as the kurtosis or the dispersion of the label distribution.
- *Landmarking metafeatures* [23, 2] are computed by running fast machine learning algorithms to characterize properties of the dataset. Since they characterize which simple approaches work well (and, in combination, also which simple approaches work better than others) they are intuitively very relevant for determining which hyperparameter configuration of a given algorithm would perform well.

While most of the metafeatures can be computed for a whole dataset, some of them (e.g., skewness) are defined for each attribute of a dataset. In this case, we compute the metafeature for each attribute of the dataset and use the mean, standard deviation, minimum and maximum of the resulting vector as proposed in [27]. Importantly, as our datasets are relatively small, the metafeatures for one dataset can be computed within less than one minute. Furthermore, for every dataset we use, the time needed to compute the metafeatures is less than the average time it takes to evaluate a hyperparameter configuration.

4 Experimental Methodology

We now discuss the datasets we used in our experiments, as well as the machine learning algorithms and their hyperparameters we optimized for them.

Table 2. List of the 57 datasets used for the experiments; the names refer to the names on the OpenML project website[34].

abalone	anneal.ORIG	arrhythmia
audiology	autos	balance-scale
braziltourism	breast-cancer	breast-w
car	cmc	credit-a
credit-g	cylinder-bands	dermatology
diabetes	ecoli	eucalyptus
glass	haberman	heart-c
heart-h	heart-statlog	hepatitis
ionosphere	iris	kr-vs-kp
labor	letter	liver-disorders
lymph	mfeat-factors	mfeat-fourier
mfeat-karhunen	mfeat-morphological	mfeat-pixel
mfeat-zernike	mushroom	nursery
optdigits	page-blocks	pendigits
postoperative-patient-data	primary-tumor	satimage
segment	sonar	soybean
spambase	tae	tic-tac-toe
vehicle	vote	vowel
waveform-5000	yeast	zoo

Table 3. Hyperparameters of the SVM. We optimized the base-2 logarithm of C and γ.

Hyperparameter	Values	Steps
$\log_2(C)$	$\{-5, -4, \ldots, 15\}$	21
$\log_2(\gamma)$	$\{-15, -14, \ldots, 3\}$	19

4.1 Datasets and Preprocessing

For our experiments, we obtained the 57 datasets listed in Table 2 from the OpenML project website[34]. We first shuffled each dataset and then split it in stratified fashion into 2/3 training and 1/3 test data. Validation performance for Bayesian optimization was then computed by ten-fold crossvalidation on the training dataset.

To use the same dataset for each classification algorithm, we coded categorical features using a one-hot (aka 1-in-k) encoding, replacing each categorical feature f with domain $\{v_1, \ldots, v_k\}$ by k binary variables, only the i-th of which is set to true for data points where f is set to v_i. To retain sparsity, we replaced any missing values with zero. Finally, we scaled numerical features linearly to the range $[0, 1]$ by subtracting the minimum value and dividing by the maximum. [3]

4.2 Machine Learning Algorithms and Their Hyperparameters

We empirically evaluated our MI-SMBO approach to optimize two practically relevant machine learning frameworks, one with few and one with many hyperparameters. The first framework are Support Vector Machines (SVMs) [28], namely the SVM implementation in Scikit-Learn (short sklearn) [22]. We used an RBF kernel and, in accordance with the LibSVM user guide [8] optimized two hyperparameters: the complexity penalty C and the kernel width of the RBF kernel γ. We chose the range of allowed values according to the LibSVM user guide; see Table 3 for details.

Our second machine learning framework comprises a range of machine learning algorithms in sklearn [22]. We combined all algorithms into a single hierarchical optimization problem using the Combined Algorithm Selection and Hyperparameter optimization (CASH) setting by Thornton et al. [33]: there was one top-level hyperparameter $\theta_{\text{classifier}}$ choosing between several classification algorithms and all hyperparameters of classification algorithm A_i were conditional on $\theta_{\text{classifier}}$ being set to A_i. This CASH problem is of high practical relevance since it describes precisely the problem an end user faces when given a new dataset.[4] To keep the computation bearable and the results interpretable, we only included three classification algorithms: an SVM with an RBF kernel (as in our first experiment), a linear SVM, and random forests [6] (one of the most robust classifiers available). Since we expected noise and redundancies in the training data, we also allowed the optimization procedure to use Principal Component Analysis (PCA) for preprocessing, with the number of PCA components being conditional on PCA being applied. In total this lead to 10 hyperparameters, as detailed in Table 4.

Table 4. Hyperparameters for the CASH problem in sklearn. All hyperparameters except $\theta_{\text{classifier}}$ and preprocessing are conditional. Hyperparameters not mentioned were set to their default value.

Component	Hyperparameter	Values	# Values
Main	$\theta_{\text{classifier}}$	{RF, SVM, LinearSVM}	3
Main	preprocessing	{PCA, None}	2
SVM	$\log_2(C)$	$\{-5, -4, \ldots, 15\}$	21
SVM	$\log_2(\gamma)$	$\{-15, -14, \ldots, 3\}$	19
LinearSVM	$\log_2(C)$	$\{-15, -14, \ldots, 15\}$	21
LinearSVM	penalty	$\{L_1, L_2\}$	2
RF	min splits	$\{1, 2, 4, 7, 10\}$	5
RF	max features	$\{1\%, 4\%, \ldots, 100\%\}$	10
RF	criterion	{Gini, Entropy}	2
PCA	variance to keep	$\{80\%, 90\%\}$	2

4.3 Experimental Setup

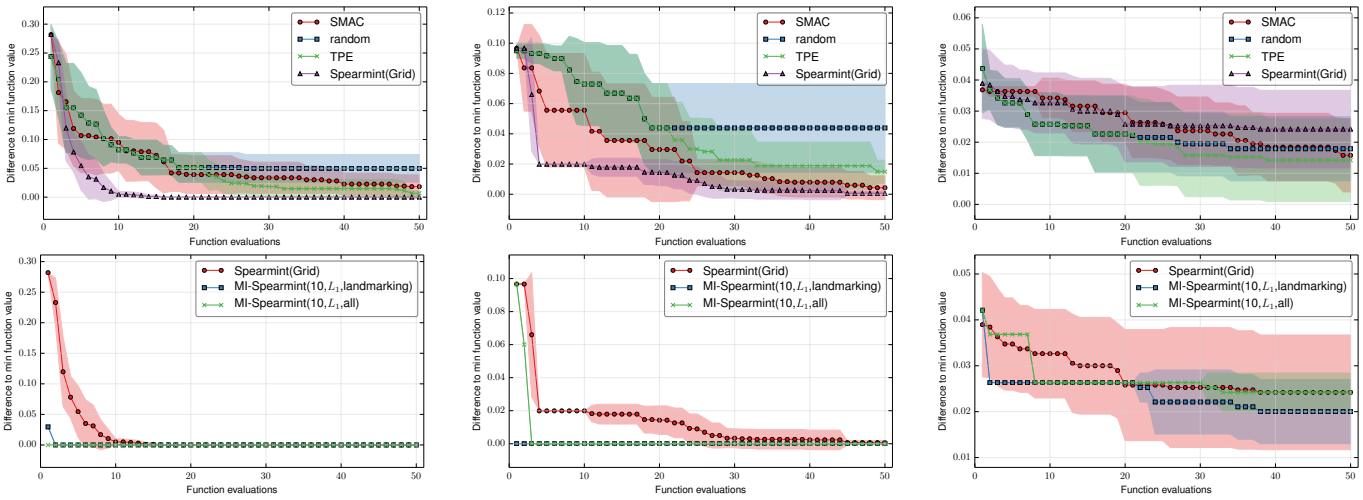
For both machine learning frameworks, we precomputed the 10-fold crossvalidation error on all 57 datasets over a grid with all possible hyperparameter combinations. For the SVM, this grid contained all 399 combinations of the 19 values for C and 21 values for γ listed in Table 3. For sklearn, it contained an additional 1 224 possible hyperparameter configurations, due to the additional flexibility of preprocessing and the two other model classes (linear SVMs and random forests, see Table 4). Therefore, in total, we evaluated 1 623 hyperparameter configurations on each dataset. Although the classification datasets were no larger than medium-sized ($<$ 30 000 data points), calculating the grid took up to three days per dataset on a modern CPU. This extensive precomputation allowed us to run all our experiments in simulation, by using a lookup table in lieu of running an actual algorithm. We will make the gathered algorithm performance data publicly available to facilitate both reproducibility of our experiments and follow-up work using the same data.

We evaluated our MI-SMBO approach in a leave-one-dataset-out fashion: to evaluate it on a dataset D_{new}, we assumed knowledge of the other 56 datasets and their best hyperparameter settings. Because Bayesian optimization contains random factors, we repeated each optimization run ten times on each dataset. In total, we thus executed each optimization procedure 570 times.

Our metalearning initialization approach has several free design choices we had to instantiate for our experiments. Firstly, we had to

[3] This is the standard practice for SVMs, as for example advised in the LibSVM user guide: `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

[4] We note that several others have also studied variants of the CASH problem in sklearn [13, 18].



dataset: heart-c dataset: arrhythmia dataset: breast-cancer

Figure 1. Difference in SVM validation error between the best found hyperparameters at evaluation t and the best value obtained via a full grid search on three datasets. MI-Spearmint($10, L_1, X$) stands for MI-Spearmint with an initial design of $t = 10$ configurations suggested by metalearning using metafeatures X. Note the differently scaled y-axes in the top and bottom plots.

choose a norm for the distance metric in the space of meta-features in Equation 3; we experimented with both the L_1 and L_2 norms. Next to using the full set of metafeatures, we experimented with various subsets. Since previous empirical results suggested that landmarking metafeatures are superior to other metafeatures [23, 25, 26], we experimented with using only the landmarking features used in the first experiment of Pfahringer et al. [23]. We also experimented with the subsets of metafeatures used in previous works on collaborative SMBO [1, 35]. Lastly, we had to decide how many hyperparameter configurations to evaluate as part of the warmstart phase before switching to the SMBO algorithm. Here we tried the values $t \in \{5, 10, 20, 25\}$. In total, we evaluated 40 different instantiations of our metalearning procedure.

Due to space restrictions, we only report results for a subset of these instantiations. Concerning distance measures, in preliminary experiments the results with the L_1 and L_2 norms were qualitatively similar, with slightly better results for the L_1 norm. Thus, all distances in the experiments reported here were calculated with the L_1-norm. Preliminary results with different subsets of metafeatures showed that the metafeatures used by existing collaborative SMBO methods did not match the performance of the other sets; we therefore restricted ourselves to only show results for all metafeatures and for only the landmarking metafeatures. Finally, we report performance only for MI-SMBO with $t = 10$ hyperparameter configurations suggested by metalearning; however, preliminary results suggest that for larger configuration spaces larger values of t improve results.

5 Experimental Results

We now report our results for optimizing SVMs and sklearn. For each of the two machine learning frameworks we studied, we first assessed the state of the art and then improved it with MI-SMBO. Specifically, we evaluated the base performance of the hyperparameter optimization procedures random search, Spearmint, TPE, and SMAC (described in Section 2; note that for TPE the prior distributions were uniform) on our 57 datasets and then added warmstarts via MI-SMBO to the best of these.

5.1 Warmstarting Spearmint for Optimizing SVMs

For the low-dimensional problem of optimizing SVMs, the Spearmint optimizer tended to perform best. Figure 1 (top) compares its qualitative performance on three representative datasets to that of TPE, SMAC, and random search, showing that it typically performed best, but that there was still room for improvement. A statistical analysis using a two-sided t-test on the performances for each of the 57 datasets shows that Spearmint indeed significantly outperformed TPE, SMAC, and random search in 35%, 44%, and 52% of the datasets, respectively, and only lost in 4%, 4%, and 8% of the cases, respectively. These findings are in line with previous results showing Spearmint to be the best choice for hyperparameter optimization benchmarks with a small number of continuous hyperparameters [9].

We thus applied our MI-SMBO approach to Spearmint, using either all meta-features or just the landmarking features, to suggest the first $t = 10$ hyperparameter settings Spearmint should evaluate. Figure 1 (bottom) compares the resulting warm-start versions of Spearmint against vanilla Spearmint on the same three representative datasets as above. For the two datasets on the left, metalearning directly identified one of the optimal hyperparameter configurations in the first function evaluation; this is in contrast to vanilla Spearmint, which required 17 and 45 function evaluations, respectively, to eventually reach a configuration of equal performance. In contrast, for the dataset on the right, metalearning only yielded small improvements (a comparison to the right top plot in Figure 1 shows that neither variant of Spearmint performed better than random search in this case).

Next, we analyzed the performance of MI-Spearmint using the same ranking-based evaluation as Bardenet et al. [1] to aggregate over datasets. For each dataset and for each function evaluation budget from 1 to 50, we computed the ranks of the four baselines (random search, SMAC, TPE, and Spearmint) and the two MI-Spearmint variants. More precisely, since we had available 10 runs of each of the 6 methods for each dataset (which give rise to 10^6 possible com-

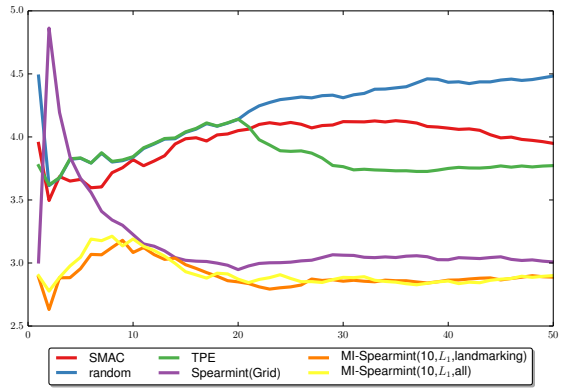


Figure 2. Average rank of each optimizer, computed over all datasets, for the SVM classification experiment. (WS10,l1,X) denotes MI-Spearmint warm-started with 10 configurations suggested by metalearning using metafeatures X.

binations), we drew a bootstrap sample of 1 000 joint runs of the six optimizers and computed the average ranks across these runs. We then further averaged these average ranks across the 57 datasets and show the results in Figure 2. We remind the reader that the rank is a measure of performance *relative* to the performance of the other optimizers; thus, a method's rank can increase over time (with larger function evaluation budgets) even though its error decreases if the other methods achieve greater error reductions. Furthermore, we note that this plot simply ranks raw function values and does not include information about how much the errors of the various methods differ. With this disclaimer noted, the results are as expected: random search performed worst, followed by SMAC and TPE, which are known to be outperformed by Spearmint for low-dimensional continuous problems [9]. The three variants of Spearmint performed best, converging to a similar rank with larger function evaluation budgets; meta-learning yielded dramatically better results for very small function evaluation budgets, and after about 10 function evaluations Spearmint (almost) caught up. We note, however, that even after 50 function evaluations Spearmint still had not fully caught up to its warmstart variants, indicating that an initialization with 10 configurations determined by meta learning provided not only good performance with few function evaluations but also a good basis for Spearmint to improve upon further.

To complement the above ranking analysis, Figure 3 quantifies on how many datasets MI-Spearmint (based on the landmarking features) performed better and worse than the other methods according to a two-sided t-test (over the ten repetitions of runs per dataset). The upper plot of Figure 3 shows the ratio of datasets for which MI-Spearmint performed significantly better than the other methods, and the lower plot shows the statistically significant losses. Both of these quantities are plotted over time, as the function evaluation budget increases. We observe that MI-Spearmint started off much better than all other methods. Given larger function evaluation budgets, using its Spearmint part, it even increased the performance advantage over random search, TPE, and SMAC. Compared to Spearmint, MI-Spearmint started off significantly better in 70% of the datasets, but these differences leveled off over time. There was very little difference between the two MI-Spearmint variants (based on landmarking features vs. based on all features).

5.2 Warmstarting SMAC for Optimizing sklearn

We used the same approach as in the above experiment to assess MI-SMBO's performance on the combined algorithm selection and hyperparapeter optimization problem in sklearn. First, we assessed the state of the art for this problem. Due to the conditional hyperparameters in the sklearn space, we excluded Spearmint (which does not natively support them and is known to perform poorly in their presence [9]) and only evaluated SMAC, TPE, and random search. Figure 4 (top) presents the qualitative performance of these optimizers on three representative datasets, showing that both SMAC and TPE performed better than random search. Overall, in line with the results of Eggensperger et al. [9] for large hyperparameter spaces, we found SMAC and TPE to perform best. We applied our metalearning initialization to SMAC, but would also expect TPE to benefit from it.

Figure 4 (bottom) shows the qualitative results of MI-SMAC compared to vanilla SMAC. In the left plot, the metalearning suggestions were reasonable and MI-SMAC's second part could improve on these over time. In the middle plot the second configuration suggested by metalearning was already the best, leaving no room for improvement by SMAC. The right plot highlights the fact that metalearning can also fail and decrease the performance of SMAC.

Figure 5 shows the percentage of statistically significant wins of MI-SMAC against the other optimizers. As before, we evaluated two different versions of MI-SMAC, based on all features and based on only the landmarking metafeatures from Pfahringer [23]; the figure shows that MI-SMAC based on the landmarking features alone worked somewhat better than based on all features, winning statistically significantly on 11% of the datasets (and loosing on 8%). Compared to the optimizers without metalearning, MI-SMAC performed much better from the start. Even after 50 iterations, it performed significantly better than TPE on 14% of the datasets (in 8% worse), better than SMAC on 25% of the datasets (in 10% worse), and better than random search on 35% of the dataset (in 9% worse). We would like to point out that the improvement MI-SMAC yielded over SMAC is nearly as large as the improvement that SMAC yielded over random search (in 29% better). This is in contrast to the (only) slight improvements MI-Spearmint yielded over Spearmint for optimizing SVMs. We attribute the success for sklearn to its much larger search space, which not even SMAC can effectively search in as little as 50 function evaluations. Drawing on successful optimizations from previous datasets clearly helped SMAC in this complex search space.

6 Conclusion

We have presented a simple, yet effective, method for improving Sequential Model-based Bayesian Optimization (SMBO) of hyperparameters by transferring knowledge from previous optimization runs. Our method combines ideas from both the metalearning and the Bayesian optimization community by initializing SMBO with configurations suggested by a metalearning procedure. We dub the resulting metalearning-initialized SMBO variant MI-SMBO. Importantly, MI-SMBO is agnostic of the actual SMBO method used and can thus be applied to the method best suited for a particular problem.

We demonstrated MI-SMBO's efficacy by improving the initialization of two quite different SMBO methods for optimizing two machine learning frameworks on a total of 57 datasets. For optimization in the low-dimensional hyperparameter space of a support vector machine, our MI-Spearmint variant of the best-performing SMBO method Spearmint mainly improved upon Spearmint in the

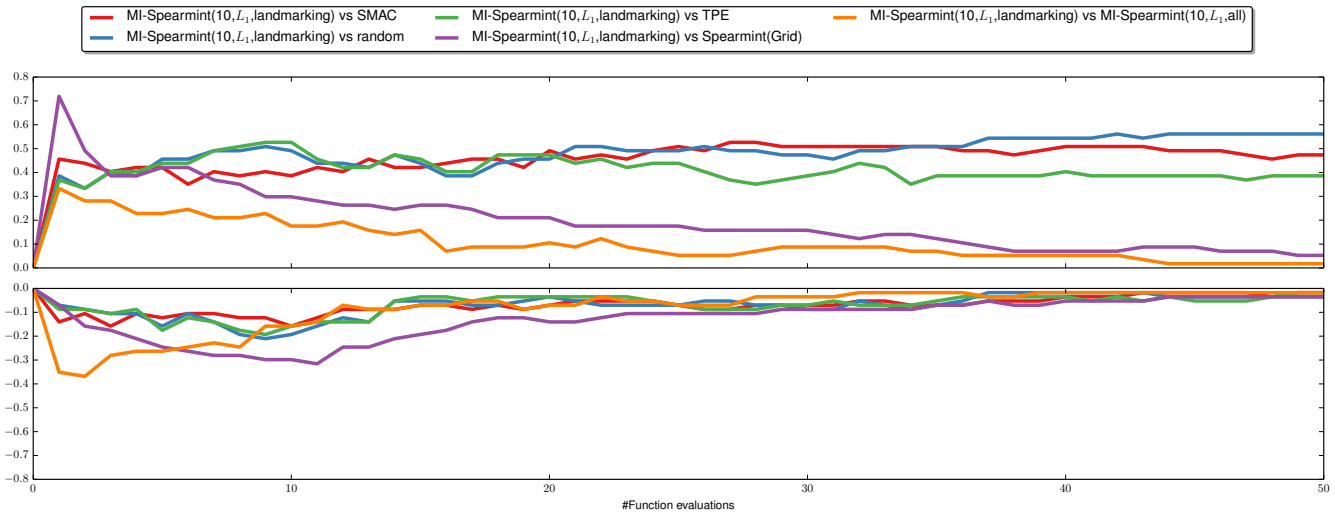
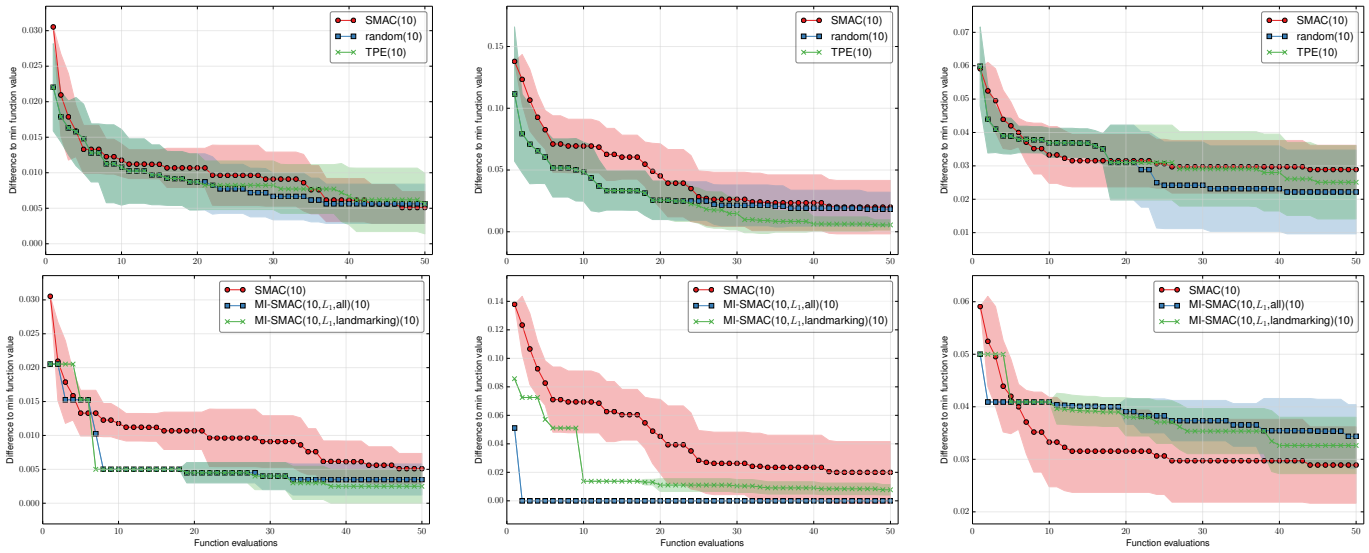


Figure 3. Percentage of wins of MI-Spearmint with an initial design of $t = 10$ configurations suggested by metalearning using the L_1 distance on the metafeature subset from Pfahringer [23]. The upper plot shows significant wins of MI-Spearmint against each other approach according to the two-sided t-test while the lower plot shows the statistically significant losses.



dataset: sonar dataset: heart-h dataset: hepatitis

Figure 4. Difference in validation error between sklearn instantiated with the best found hyperparameters and the best value obtained via a full grid search, for three datasets. (WS10,l1,X) stands for MI-SMAC with an initial design of $t = 10$ configurations suggested by metalearning using metafeatures X. Note the differently scaled y-axes in the top and bottom plots.

early stages of optimization, thus helping it find good configurations quickly. For a large configuration space describing a combined algorithm selection and hyperparameter optimization problem in scikit-learn, our MI-SMAC variant of the best-performing SMBO variant SMAC substantially improved over SMAC (and all other optimizers we tested) across a range of function evaluation budgets, showing the potential of our approach especially for large scale hyperparameter optimization.

In future work, we plan to evaluate MI-SMAC for even larger configuration spaces, such as those of Auto-WEKA [33] and Hyperopt-Sklearn[18]. We also noticed the lack of a canonical implementation of metafeatures and are aiming to provide such an implementation. Finally, we plan to integrate metalearning into the SMBO procedure and compare the result with recent work on collaborative SMBO [1, 35, 32].

REFERENCES

[1] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, 'Collaborative hyperparameter tuning', in *Proc. of ICML*, (2013).

[2] H. Bensusan and C. Giraud-Carrier, 'Discovering task neighbourhoods through landmark learning performances', in *Proc. of 4th PKDD*. Springer, (September 2000).

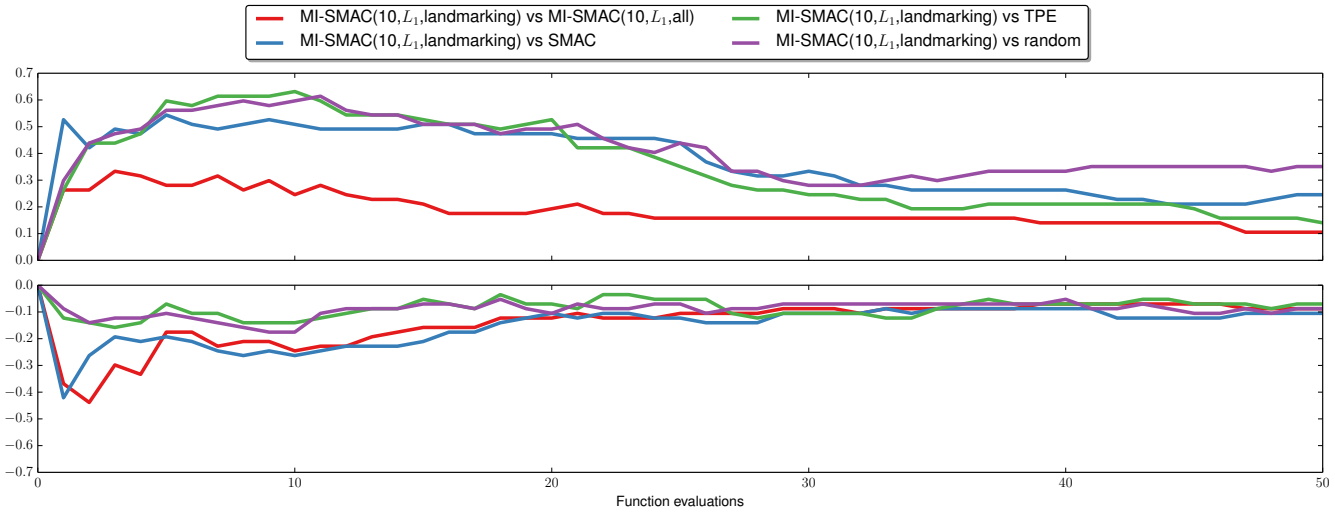


Figure 5. Percentage of wins of MI-SMAC with an initial design of $t = 10$ configurations suggested by metalearning using the L_1 distance on the metafeature subset from Pfahringer [23]. The upper plot shows the number of significant wins of MI-SMAC over competing approaches according to the two-sided t-test while the lower plot shows the statistically significant losses.

[3] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, 'Algorithms for hyper-parameter optimization', in *Proc. of NIPS*, (2011).

[4] J. Bergstra and Y. Bengio, 'Random search for hyper-parameter optimization', *JMLR*, **13**, (February 2012).

[5] J. Bergstra, D. Yamins, and D. D. Cox, 'Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures', in *Proc. of ICML*, (2013).

[6] L. Breiman, 'Random forests', *Machine Learning*, **45**, (2001).

[7] E. Brochu, V. M. Cora, and N. de Freitas, 'A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning', *CoRR*, **abs/1012.2599**, (2010).

[8] Chih-Chung Chang and Chih-Jen Lin, 'LIBSVM: A library for support vector machines', *ACM Transactions on Intelligent Systems and Technology*, **2**, (2011).

[9] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. H. Hoos, and K. Leyton-Brown, 'Towards an empirical foundation for assessing bayesian optimization of hyperparameters', in *NIPS workshop on Bayesian Optimization*, (2013).

[10] T.A.F. Gomes, R.B.C. Prudêncio, C. Soares, A. Rossi, and A. Carvalho, 'Combining meta-learning and search techniques to select parameters for support vector machines', *Neurocomputing*, **75**(1), (2012).

[11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten, 'The WEKA data mining software: an update', *ACM SIGKDD Explorations Newsletter*, **11**(1), 10–18, (2009).

[12] P. Hennig and C. Schuler, 'Entropy search for information-efficient global optimization', *JMLR*, **13**, (2012).

[13] M. W. Hoffman, B. Shahriari, and N. de Freitas, 'Exploiting correlation and budget constraints in Bayesian multi-armed bandit optimization', *ArXiv e-prints*, (March 2013).

[14] F. Hutter, H. H. Hoos, and K. Leyton-Brown, 'Sequential model-based optimization for general algorithm configuration', in *Proc. of LION-5*, (2011).

[15] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, 'Algorithm runtime prediction: Methods and evaluation', *JAIR*, **206**(0), 79 – 111, (2014).

[16] D.R. Jones, M. Schonlau, and W. Welch, 'Efficient global optimization of expensive black box functions', *Journal of Global Optimization*, **13**, (1998).

[17] A. Kalousis, *Algorithm Selection via Meta-Learning. University of Geneve, Department of Computer Science*, Ph.D. dissertation, University of Geneve, 2002.

[18] B. Komer, J. Bergstra, and C. Eliasmith, 'Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn', in *ICML workshop on AutoML*, (2014).

[19] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren, 'Selecting classification algorithms with active testing on similar datasets', in *5th PLANLEARN WORKSHOP at ECAI*, (2012).

[20] *Machine Learning, Neural and Statistical Classification*, eds., Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell, Ellis Horwood, 1994.

[21] P.B.C. Miranda, R.B.C. Prudêncio, A. Carvalho, and C. Soares, 'Combining meta-learning with multi-objective particle swarm algorithms for SVM parameter selection: An experimental analysis', in *Brazilian Symposium on Neural Networks*, (2012).

[22] 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', *JMLR*, **12**, (2011).

[23] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier, 'Meta-learning by landmarking various learning algorithms', in *Proc. of ICML*, (2000).

[24] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*, The MIT Press, 2006.

[25] M. Reif, F. Shafait, and A. Dengel, 'Prediction of classifier training time including parameter optimization', in *KI 2011: Advances in Artificial Intelligence*, (2011).

[26] M. Reif, F. Shafait, and A. Dengel, 'Meta-learning for evolutionary parameter optimization of classifiers', *Machine Learning*, **87**, (2012).

[27] M. Reif, F. Shafait, and A. Dengel. Meta2-features: Providing meta-learners more information, 2012. Poster and Demo Track of the 35th German Conference on AI.

[28] Bernhard Scholkopf and Alexander J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, Cambridge, MA, USA, 2001.

[29] J. Snoek, H. Larochelle, and R.P. Adams, 'Practical bayesian optimization of machine learning algorithms', in *Proc. of NIPS*, (2012).

[30] C. Soares and P.B. Brazdil, 'Zoomed ranking: Selection of classification algorithms based on relevant performance information', in *Proc. of PKDD'00*, Springer, (2000).

[31] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, 'Gaussian process optimization in the bandit setting: No regret and experimental design', in *Proc. of ICML*, (2010).

[32] K. Swersky, J. Snoek, and R.P. Adams, 'Multi-task bayesian optimization', in *Proc. of NIPS*, (2013).

[33] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, 'Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms', in *Proc. of KDD'13*, (2013).

[34] J. N. van Rijn, B. Bischl, L. Torgo, B. Gao, V. Umaashankar, S. Fischer, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren, 'OpenML: a collaborative science platform', in *Proc. of ECML/PKDD'13*, (2013).

[35] D. Yogatama and G. Mann, 'Efficient transfer learning method for automatic hyperparameter tuning', in *Proc. of AISTATS*, (2014).

Similarity measures of algorithm performance for cost-sensitive scenarios

Carlos Eduardo Castor de Melo[1] and **Ricardo Bastos Cavalcante Prudêncio**[2]

Abstract.
Knowledge about algorithm similarity is an important feature of meta-learning, where information gathered from previous learning tasks can be used to guide the selection or combination of algorithms for a new dataset. Usually this task is done by comparing global performance measures across different datasets or, alternatively, comparing the performance of algorithms at the instance-level. These previous similarity measures do not consider misclassification costs, and hence they neglect an important information that can be exploited in different learning contexts. In this paper we present algorithm similarity measures that deals with cost proportions and different threshold choice methods for building crisp classifiers from learned models. Experiments were performed in a meta-learning study with 50 different learning tasks. The similarity measures were adopted to cluster algorithms according to their aggregated performance on the learning tasks. The clustering process revealed similarity between algorithms under different perspectives.

1 Introduction

Meta-learning is a framework developed in supervised machine learning for acquiring knowledge from empirical case studies and relating features of learning problems to algorithm performance [2]. The knowledge acquired in meta-learning has been used to support algorithm selection and combination in different contexts [13] [11]. In meta-learning, it is assumed that similar problems present similar algorithm performance. Hence, to know how similar is the performance obtained by different algorithms is important for meta-learning. This information can be used to discover similarities between algorithms and between datasets [9] and also to support the prediction of suitable algorithms for a given dataset based on the algorithms' performance on past datasets [11].

Deploying global metrics, such as accuracy and AUC, to compare the performance of algorithms on a dataset has been the most common approach to deal with the above issue. Despite the popularity of this approach, it may lose important knowledge about algorithm similarity since it is based on average algorithm performance without considering differences of algorithm behaviour in an instance-level. In order to overcome this limitation, alternative measures of algorithm similarity (e.g., based on error correlation [9] or classifier output difference [10]) have been adopted. Here, algorithms are considered similar if they produce the same classifications, or alternatively the same mistakes, on the same instances.

[1] Centro de Informática, Universidade Federal de Pernambuco, Brasil, email: cecm2@cin.ufpe.br
[2] Centro de Informática, Universidade Federal de Pernambuco, Brasil, email: rbcp@cin.ufpe.br

Algorithm similarity measures adopted in these previous works are limited since they are not flexible enough to consider different misclassification costs and different strategies to build classifiers. Algorithms usually produce models (scoring functions) that return scores or class probabilities for the input examples. A classifier is then built from a model by adopting a decision threshold. The prediction for a given example depends on both the score returned by the learned model for that example and the decision threshold. The most common approach to produce classifiers is to adopt a fixed decision threshold (e.g., 0.5). Alternatively, decision thresholds can be chosen according to the operating conditions or contexts (e.g., class frequencies and misclassification costs) observed when the learned model is deployed. In [7], the authors showed that different threshold choice methods require the use of different performance measures to evaluate a learned model. Similarly, concerning algorithm similarity, specific measures have to be defined when misclassification costs and adaptive threshold choice methods are taken into account. Previous work on algorithm similarity measures does not consider such aspects. The previous similarity measures are defined implicitly assuming fixed decision thresholds and thus are only suitable for comparing the performance of crisp classifiers.

Based on the above motivation, in our work we developed similarity measures for algorithm performance taking into account cost proportions and different threshold choice methods. As in [13], we adopted an instance-level strategy. For that, we initially proposed instance hardness measures to indicate how difficult is an instance to be correctly classified by a model. Specific instance hardness measures were proposed for three threshold choice methods: the score-fixed [7], the score-driven [6] and rate-driven methods [8]. By assuming a threshold choice method and a learned model, we built for each instance a corresponding *cost curve*, which indicates the model loss for that instance along different misclassification costs. The instance hardness is defined as the area under the cost curve. The dissimilarity[3] between two algorithms for a given dataset is defined as the average absolute difference between the hardness values over all instances of the dataset.

In our work, we applied the proposed measures in a meta-learning study in which 50 datasets and 8 learned models were adopted. Clusters of models were produced by adopting the score-fixed, score-driven and rate-driven measures. Each cluster reveals which algorithms produced similar learned models on the 50 datasets under each cost-sensitive perspective. We observed that the algorithm behaviour among the datasets was quite distinct for each measure. By adopting the score-fixed method, two algorithms were similar if they return the same classification for a high number of instances by using

[3] In this paper we treat *dissimilarity* as the complement of *similarity*, henceforth we will use only the latter term

a given threshold. By adopting the score-driven method, algorithms were considered similar if they produced similar scores for the instances. By adopting the rate-driven method in turn, algorithms were considered similar if they produced similar rankings of instances despite how calibrated are their scores.

This paper is organized as follows. Applications of algorithms similarity measures are presented in Section 2, with a review of related works. Section 3 introduces the notation and basic definitions used through this paper. The concepts of instance hardness and threshold choice methods are explained in Section 4 and three threshold choice methods are presented: score-fixed, score-driven and rate-driven. The proposed method for measuring the similarity of algorithms performance is presented on Section 5. Section 6 presents the experimental methodology. An analysis of the similarity measures over a single dataset is provided in Section 7, followed by the Section 8 which demonstrates the aggregate similarity across a group of datasets. Finally, Section 9 concludes the paper with a discussion of the results and insights for future works.

2 Measuring algorithm similarity

Meta-learning deals with methods to exploit knowledge gathered from learning on different tasks [2]. Differently from base-level learning, which focuses on acquiring experience on a single task, meta-learning acquires knowledge from a *meta-data* set produced when a pool of algorithms is applied on distinct learning tasks. Given a learning task, a meta-example usually stores characteristics of the task and the performance obtained by a pool of candidate algorithms. Meta-learning can be applied: (1) in a descriptive sense aiming to discover similarities between datasets and algorithms and (2) in a predictive sense to select candidate algorithms for new datasets based on the knowledge acquired in the meta-learning process.

Different meta-learning approaches rely on the use of similarity measures of algorithm performance. For instance, in [9] the authors clustered a pool of learning algorithms based on the patterns of the errors observed in different datasets. Clustering of algorithms based on their performance was also performed more recently in [10]. Clustering algorithms may provide useful information to guide the processes of algorithm selection and combination [9]. As another line of research in meta-learning is the landmarking approach [11], which uses the performance of simpler algorithms (called landmarkers) to guide the selection of more complex ones. In this approach, datasets are characterized based on the performance obtained by the landmarkers (which are usually simple and diverse algorithms, such as decision stumps and linear models, or simplified versions of the candidate algorithms). One method to handle this approach is measuring the similiarity of a new dataset with the meta-examples retrived from the meta-data based on the similarity of performance obtained by the landmarkers[5]. The best candidate algorithms adopted in the retrieved problems are then suggested for the new dataset.

The most common approach for measuring algorithm similarity is the use of global performance measures, like accuracy or AUC measure, estimated from an empirical evaluation process (such as cross-validation). Similarity between algorithms can be obtained by computing the differences or ratios between the performance measures, as performed in [5]. Although this approach is widely applied, it is strongly criticized since it may lose important information about algorithms behaviour and may not characterize similarity properly. For example, if a dataset has 100 instances and a given algorithm misclassifies 20 instances and another one misclassifies 20 instances completely different from the first ones, the accuracy of both algorithms will be the same but their behaviour is quite different.

A more fine-grained approach to algorithm similarity is to consider the performance at each instance of the dataset. This approach has the advantage of showing local discrepancies of performance among the space of instances. In [9], error correlation is adopted as similarity measure between two algorithms, in such a way that two algorithms are similar if they produce the same errors in the same instances. In [10], the authors present the Classifier Output Difference as an alternative measure for this approach. This metric is defined as the probability of two distinct classifiers make different predictions [15].

Although this approach represents a more refined view about algorithm performance, the measures proposed in the literature are computed from predictions generated by crisp classifiers (i.e., with fixed decision thresholds chosen a priori). However as stated in section 1, in the context of misclassification costs, decision thresholds can be adaptively defined to minimize the loss of a learned model. Hence, two instances considered equally easy by a classifier with a fixed threshold can be rather difficult under different decision thresholds and costs. In this work, we derived similarity measures for algorithm performance in cost sensitive scenarios, by considering different methods to choose decision thresholds of classifiers based on the knowledge about misclassification costs.

3 Notation and Basic Definitions

The basic definitions adopted in our work are mostly based on [7]. Instances can be classified into one of the classes $Y = \{0, 1\}$, in which 0 is the positive class and 1 is the negative class. A learned model m is a scoring function that receives an instance x as input and returns a score $s = m(x)$ that indicates the chance of a negative class prediction. A model can be transformed in a crisp classifier assuming a decision threshold t in such a way that if $s \leq t$ then x is classified as positive, and it is classified as negative if $s > t$.

The errors of a classifier are associated to costs related to the classes. The cost of a *false negative* is represented as c_0 and the cost of a *false positive* in turn is represented as c_1. As in [7], we normalize the costs by setting $c_0 + c_1 = b$ and adopt the *cost proportion* $c = c_0/b$ to represent the operating condition faced by a model this deployment. For simplicity, we adopted $b = 2$ and hence $c \in [0, 1]$, $c_0 = 2c$ and $c_1 = 2(1 - c)$.

The loss function produced assuming a decision threshold t and a cost proportion c is defined as:

$$\begin{aligned} Q(t, c) &= c_0\pi_0 FN(t) + c_1\pi_1 FP(t) \\ &= 2\{c\pi_0 FN(t) + (1 - c)\pi_1 FP(t)\} \end{aligned} \quad (1)$$

In the above equation $FN(t)$ and $FP(t)$ are respectively the *False Negative* and *False Positive* rates produced by a model when a threshold t is adopted. The variables π_0 and π_1 represent the proportion of positive and negative examples.

The Positive Rate $R(t)$ is the proportion of instances predicted as positive at the decision threshold t and can be expressed as $\pi_0(1 - FN(t)) + \pi_1 FP(t)$.

4 Instance Hardness and Cost Curves

In Equation 1, the loss produced by a classifier is an aggregation of the errors observed for the positive and the negative instances. A positive instance will be associated to a cost $2c$ when it is incorrectly

classified. An error for a negative instance in turn will be associated to a cost $2(1-c)$.

In our work, we decompose the Equation 1 to derive the loss functions for individual instances. The individual loss for a positive instance x is defined as:

$$QI(x,t,c) = 2cFN(x,t) \tag{2}$$

In the above equation, $FN(x,t) = 1$ if x is a false negative when threshold t is adopted and $FN(x,t) = 0$ otherwise. A similar definition of loss function can be done for a negative instance x:

$$QI(x,t,c) = 2(1-c)FP(x,t) \tag{3}$$

In the above equation, $FP(x,t) = 1$ if x is a false positive at threshold t and $FP(x,t) = 0$ otherwise.

Given a threshold choice method, $QI(x,t,c)$ produce a specific curve for the example x along the range of operating conditions ($c \in [0,1]$). The *instance hardness* measures in our work are defined as the area under the individual cost curves and compute the total expected loss for the range of operation conditions. In the general case, given a threshold choice method $t = T(c)$, the hardness of an instance is:

$$IH^T(x) = \int_0^1 QI(x, T(c), c)w(c)dc \tag{4}$$

In the above equation, $w(c)$ represents the distribution of c. In our work, we derived the instance hardness measures for different threshold choice methods assuming uniform distribution of cost proportions.

4.1 Score-Fixed Instance Hardness

In this method, the decision threshold assumes a fixed value regardless the operation condition:

$$T(c) = t \tag{5}$$

Usually, t is set to 0.5. If x is an incorrectly classified positive instance, then $FN(x,t) = 1$. By replacing $FN(x,t)$ in Equation 2, the instance cost curve is defined as:

$$QI(x,t,c) = 2c \tag{6}$$

If x is an incorrectly classified negative instance, then $FP(x,t) = 1$ and its corresponding cost curve is:

$$QI(x,t,c) = 2(1-c) \tag{7}$$

For correctly classified instances (either positive or negative), the instance cost curve is a constant line $QI(x,t,c) = 0$. Figure 1 illustrates the score-fixed instance cost curve considering positive and negative instances.

By integration QI, we derive the instance hardness value for incorrect positive instance as follows:

$$IH^{sf}(x) = \int_0^1 2c\,\mathrm{d}c = \Big[c^2\Big]_0^1 = 1 \tag{8}$$

Similarly, for incorrect instances the instance hardness is defined as:

$$IH^{sf}(x) = \int_0^1 2(1-c)\,\mathrm{d}c = \Big[2c - c^2\Big]_0^1 = 1 \tag{9}$$

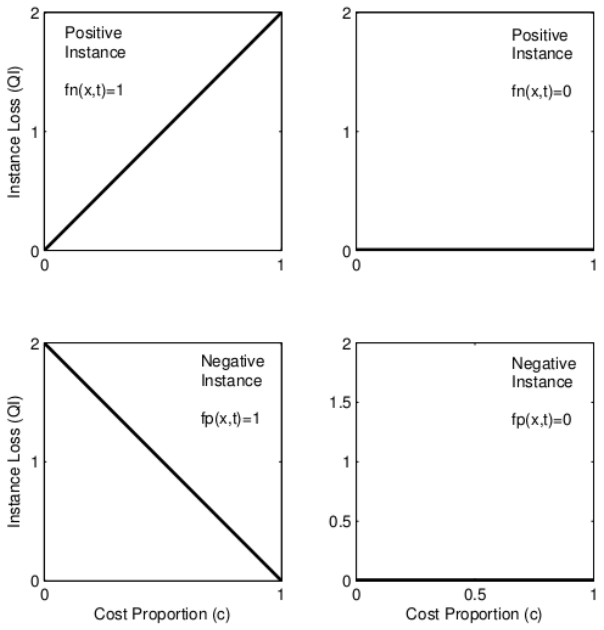


Figure 1. Example of instance cost curve for a positive and negative instances assuming the score-fixed method.

For correctly classified instances (either positive or negative), $IH^{sf}(x) = 0$.

4.2 Score-Driven Instance Hardness

In the score-driven method [6], the decision threshold t is defined by simply setting it equal to the operating condition c:

$$T(c) = c \tag{10}$$

For instance, if $c = 0.7$, the cost of false negatives are higher than the costs of false positives. In this case by setting $t = c = 0.7$ the produced classifier tends to predict more instances as positive, thus minimizing the relative number of false negatives. According to [6], the score-driven method is a natural choice when the model scores are assumed to be class probability estimators and the scores are well calibrated.

Under the score-driven method, we can derive the loss function for positive instances as follows:

$$FN(x,c) = \begin{cases} 1, \text{ if } s > c \\ 0, \text{ otherwise} \end{cases} \tag{11}$$

Replacing $FN(x,c)$ in Equation 2 we have the following score-driven cost curve for a positive instance:

$$QI(x,t,c) = \begin{cases} 2c, \text{ if } s > c \\ 0, \text{ otherwise} \end{cases} \tag{12}$$

Fig. 2(a) illustrates the score-driven cost curve for a positive instance with score $s = 0.32$. For $s \leq c$ no cost is assumed; on the other hand, the cost varies linearly. The area under the cost curve is defined as an instance hardness measure:

$$IH^{sd}(x) = \int_0^s 2c \, \mathrm{d}c = \Big[c^2\Big]_0^s = s^2 \tag{13}$$

Since $y = 0$ for positive instances, the above measure can be replaced by $(y - s)^2$, which correspond to the squared-error obtained by the model.

Equations 14 and 15 define the score-driven cost curve for negative instances. Figure 2(c) illustrates this curve when the negative instance has a score $s = 0.41$.

$$FP(x, c) = \begin{cases} 1, \text{ if } s \leq c \\ 0, \text{ otherwise} \end{cases} \tag{14}$$

$$QI(x, t, c) = \begin{cases} 2(1 - c), \text{ if } s \leq c \\ 0, \text{ otherwise} \end{cases} \tag{15}$$

Similar to the positive instance, instance hardness for a negative instance is defined as the area under the cost curve:

$$IH^{sd}(x) = \int_s^1 2(1 - c) \, \mathrm{d}c = \Big[2c - c^2\Big]_s^1 = (1 - s)^2 \tag{16}$$

For negative instances we have $y = 1$ and then the above measure corresponds to $(y - s)^2$. Hence, for both positive and negative instances, hardness is defined as the squared-error obtained by the model.

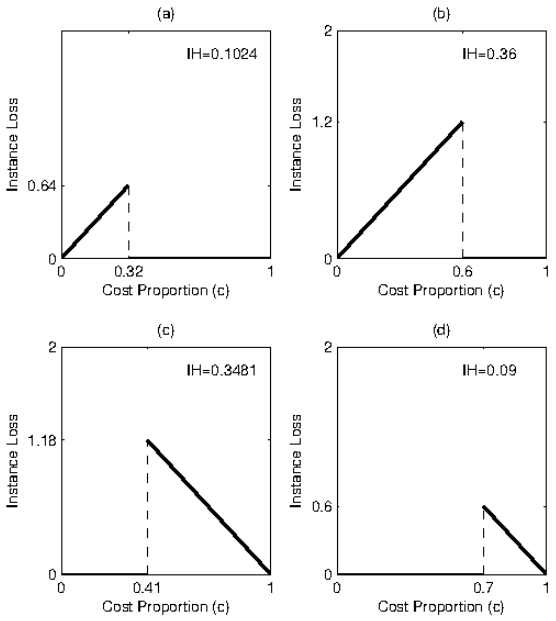


Figure 2. Instance cost curves for an instance assuming the rate-driven and score-driven methods.

4.3 Rate-Driven Instance Hardness

According to [8], the score-driven method is sensitive to the estimation of the scores. For instance, if the scores are highly concentrated, a small change in the operating condition (and consequently in the decision threshold) may drastically affect the classifier performance. As an alternative, in [8] the authors proposed to use the proportion of instances predicted as positive (i.e., $R(t)$) to define the decision thresholds.

In the rate-driven method, the decision threshold is set to achieve a desired proportion of positive predictions. The threshold choice method is defined as:

$$T(c) = R^{-1}(c) \tag{17}$$

For instance, if $c = 0.7$ the threshold t is set in such a way that 70% of the instances are classified as positive. The operating condition c is then expressed as the desired positive rate: $c = R(t)$. The probability of a false negative under the rate-driven choice method can be defined as:

$$FN(x, R^{-1}(c)) = \begin{cases} 1, \text{ if } s > R^{-1}(c) \\ 0, \text{ otherwise} \end{cases} \tag{18}$$

Replacing $FN(x, R^{-1}(c))$ in Equation 2 we have the following rate-driven cost curve for a positive instance:

$$QI(x, t, c) = \begin{cases} 2c, \text{ if } s > R^{-1}(c) \\ 0, \text{ otherwise} \end{cases} \tag{19}$$

Fig. 2(b) illustrates the rate-driven cost curve for a positive instance with a score $s = 0.6$. For $s \leq R^{-1}(c)$, or alternatively for $R(s) \leq c$, no cost is assumed. When $R(s) > c$, the cost of the instance varies linearly. The area under the rate-driven cost can be adopted as an instance hardness measure:

$$IH^{rd}(x) = \int_0^{R(s)} 2c \, \mathrm{d}c = \Big[c^2\Big]_0^{R(s)} = R(s)^2 \tag{20}$$

The above measure states that the instance hardness is related to position of the instance in the ranking produced by the learned model. The worse is the ranking of the positive instance, the higher is the instance hardness.

Equations 21 and 22 define the rate-driven cost curve for negative instances with score s, illustrated in Figure 2(d).

$$FP(x, R^{-1}(c)) = \begin{cases} 1, \text{ if } s \leq R^{-1}(c) \\ 0, \text{ otherwise} \end{cases} \tag{21}$$

$$QI(x, t, c) = \begin{cases} 2(1 - c), \text{ if } s \leq R^{-1}(c) \\ 0, \text{ otherwise} \end{cases} \tag{22}$$

Similar to the positive instance, instance hardness for a negative instance is defined as the area under the cost curve:

$$IH^{rd}(x) = \int_{R(s)}^1 2(1 - c) \, \mathrm{d}c = \Big[2c - c^2\Big]_{R(s)^1} = (1 - R(s))^2 \tag{23}$$

Notice that $(1 - R(s))$ corresponds to the negative rate of a classifier at the point s. The instance hardness for negative instances is then related to the ranking of the most likely negative instances produced by the learned model.

Different from the score-driven method, which measures the magnitude of the errors obtained by a model, the rate-driven method is more related to ranking performance. By adopting the score-driven method, an instance is considered hard if its score is not well calibrated. On other hand, the same instance may be easy by assuming

the rate-driven method if it is well ranked relative to the other instances. Instance hardness by adopting the score-driven method only depends on the score of the instance. Instance hardness by adopting the rate-driven method in turn depends not only on the instance score but also on how the other instances are ordered.

5 Cost Sensitive Algorithm Similarity

The application of global metrics fails to properly measure algorithm similarity since it is based on average performance, losing important information during the evaluation process. More fine-grained measures provide a solution for this limitation by verifying the algorithm performance at the instance level. However, the proposed measures are not well defined when misclassification costs are involved.

In this work, we derived different measures for algorithm similarity based on the concept of instance hardness. Given a pair of learned models, they will be similar if the hardness values computed on a test set by using the two models are similar. More specifically, in order to relate two models, we initially collect the scores produced by them on a test set of instances. Following, we compute the hardness value for each test instance considering each model. Finally, the similarity between the models is defined by aggregating the instance hardness values as follows:

$$\mathcal{D}(m_a, m_b, D) = \frac{1}{|D|} \sum_{x \in D} |IH^T_{m_a}(x) - IH^T_{m_b}(x)| \qquad (24)$$

The above equation measures the pairwise distance between models m_a and m_b for each instance x belonging to the test set D. In our work, in order to deal with costs, we derived in the previous section three distinct instance hardness by adopting different threshold choice methods T. By adopting the score-fixed method, two models will be similar if they return the same classification result for a high number of instances. By adopting the score-driven method, two models will be similar with their scores are similar (the squared-errors produced by the models are similar at instance level). By adopting the rate-driven method in turn two models will be similar if the test instances are similarly ranked. The three methods correspond to three different evaluation strategies. Other instance hardness measures and their corresponding algorithm similarity measures can be developed in the future by considering other threshold choice methods, such as the probabilistic ones [7].

Once we have the algorithm similarity measure on a single dataset, we can compute the overall similarity over different datasets in order to achieve a wider view about the relative performance of algorithms. There are many strategies to do this aggregation, such as the use of the median or the average similarity as well as other consensus similarity methods [3] that can be taken over all datasets involved. In this work, we adopt a simple aggregation strategy by computing the average similarities measured over all datasets observed:

$$\mathcal{A}(m_a, m_b) = \frac{1}{N} \sum_{j=1}^{N} \mathcal{D}(m_a, m_b, D_j) \qquad (25)$$

In the above equation, N stands for the number of datasets available for measuring algorithm similarity. As it will be seen next section, this measure will be adopted in a case study to cluster algorithms based on average similarity over a set of learning tasks.

6 Experimental Setup

In order to achieve a good diversity of problems, we computed the instance hardness on a group of 50 datasets[4] representing problems of binary classification. Most problems were collected from the UCI repository[1].

We compare the performance of 6 algorithms available on the Scikit-Learn Project[14]. The scores for each instance were calculated using a 10-fold cross-validation. Usually, the classification algorithms return the label predicted for an informed example but to produce real-values scores (varying between 0 and 1)for the input instances, we adopted specific procedures for each algorithm. For the Nearest Neighbours (KNN), the score returned is the number of neighbours belonging to the negative class divided by K. The procedure used for the Random Forest (RF) is similar: we count the number of votes for the negative class and divide this by the number of trees in the forest. For the Decision Tree (DT), Naive Bayes (NB) and Logistic Regression (LR) the score represents the probability of the instance being negative. For the Support Vector Machine (SVM), we get the decision function output and normalize it between 0 and 1.

In order to have a reference point, we compare variations of algorithm at the clustering process. For the KNN, we applied the algorithm with 3 and 5 nearest neighbours (3NN and 5NN) and for the SVM, we adopted the experiments with the linear and RBF kernel functions (SVM_LIN and SVM_RBF, respectively).

We expect that the models learned with variations of a same algorithm will be clustered together. In order to cluster the results obtained by the similarity measures, we applied the agglomerative hierarchical clustering method with the average linkage function [4]. For the experiments with the score-fixed method, we set $t = 0.5$.

7 Dataset Algorithm Similarity

After computing the scores for all datasets instances as described in the previous section, we use the results to calculate the instance hardness measures for the three threshold choice methods presented in Section 4. Then we measure the pairwise distance among the models for each dataset using the Equation 24. In order to illustrate the use of the proposed measures, we initially present the results obtained by a single dataset (the Ecoli dataset). Next section will present the clustering results obtained by considering all the collected datasets.

Table 1. Mean Instance Hardness Measures for Ecoli Dataset

Model	$\overline{IH^{rd}}$	$\overline{IH^{sd}}$	$\overline{IH^{sf}}$
3NN	0,269	0,0694	0,0833
5NN	0,2583	0,0583	0,0744
DT	0,2858	0,0774	0,0774
LR	0,252	0,0671	0,1047
NB	0,2585	0,1949	0,2024
RF	0,256	0,051	0,0714
SVM_LIN	0,2554	0,1828	0,244
SVM_RBF	0,2553	0,1811	0,2381

Table 1 presents the average instance hardness measures for the Ecoli dataset. The average hardness values observed suggest that the models are better at calibrating scores than at ranking instances for this dataset, as can be seen from the values presented by both methods based on score. In fact, the score-fixed and score-driven instance

[4] The list of datasets used is available at http://www.cin.ufpe.br/∼cecm2

hardness measures are in general smaller than the rate-driven ones. Also, large differences in the quality of scores produced by some algorithms do not reflect in large differences in the ranking quality. For instance, although the NB produced bad results for the score-fixed and score-driven methods, its results for the rate-driven method are similar to the other algorithms.

Figures 3, 4 and 5 present the dendrograms derived from the clustering process by adopting the proposed measures for the Ecoli dataset (respectively for the score-fixed, score-driven and rate-driven measures). The dendrograms for the score-fixed and score-driven methods show that the learned models produced the same clusters, differing on the cut-points. The clustering presented by the rate-driven method is also related to the methods based on score. The cluster formed by the SVM models is observed in all cases. The remaining clusters have a slight difference. In the score-fixed and score-driven methods, the NB model is a cluster by itself, but in the rate-driven case NB was considered similar to LR (i.e., they produced similar rankings of instances although their scores are different). In all cases, the KNN and the RF models were considered similar, which can be supported in [12].

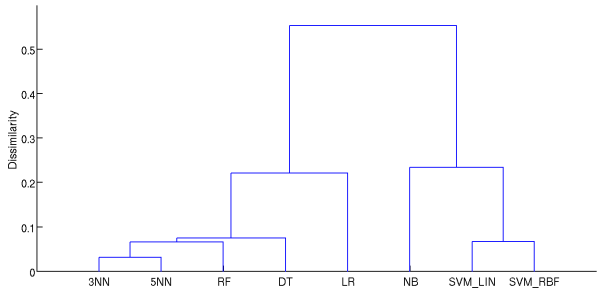


Figure 3. Clustered score-fixed performance for the models learned on the Ecoli dataset

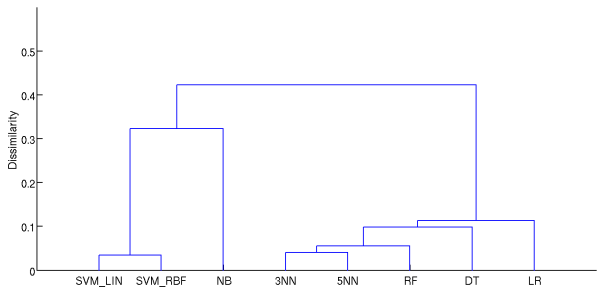


Figure 4. Clustered score-driven performance for the models learned on the Ecoli dataset

8 Aggregated Algorithm Similarity

In order to have a better view about the algorithms similarities, we applied the equation 25 to compute the average distance between algorithms across the 50 datasets adopted in our experiments. Figures 6 and 7 present the dendrograms of algorithms considering the score-fixed and score-driven measures. These dendrogram shows that the

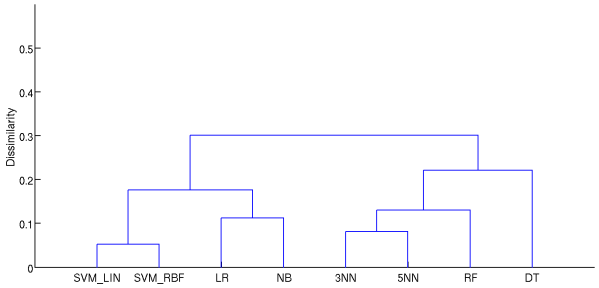


Figure 5. Clustered rate-driven performance for the models learned on the Ecoli dataset

models learned by KNN were the most similar, as expected. The second most similar pair of models was produced by the RF and the LR algorithms (with a low dissimilarity measure around 0.1 in both cases). Depending on the cut-point (e.g., 0.3) adopted to produce clusters from the dendrogram, the DT algorithm is clustered together with 3NN, 5NN, RF and LR. The models obtained by NB are the ones that present the most distinct behaviour. Finally, the SVM models are similar between them (relative to the other algorithms), but their similarity level is not so high. The change in the kernel function produced in many datasets very distinct models. Some of the results derived from the dendrogram are expected (such as the similarity between 3NN and 5NN). Other results in turn, such as the similarity between RF and LR, were not expected and hence have to be better investigated in the future.

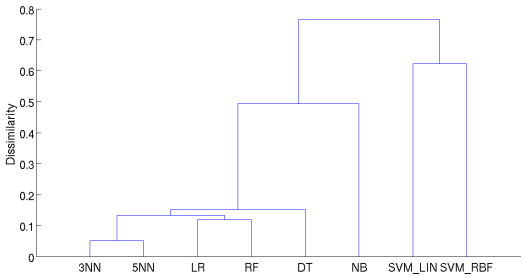


Figure 6. Clustered average score-fixed performance

Figure 8 displays the dendrogram of algorithms produced by adopting the rate-driven measure. The algorithms are more similar to each other when similarity is measured in terms of ranking quality. As in the score-fixed and score-driven dendrograms, the 3NN, 5NN, DT and RF are clustered together. The algorithms LR, SVM_LIN and NB formed another cluster. Different from the dendrograms for the score-fixed and score-driven methods, we see that the SVM models do not belong to the same group. Again, a deeper investigation has to be done to provide further explanations on why a given pair of algorithms was similar. In our work, we provide a general framework that can be extended with new algorithms and datasets and can be used to identify which algorithms are similar under different cost-sensitive perspectives.

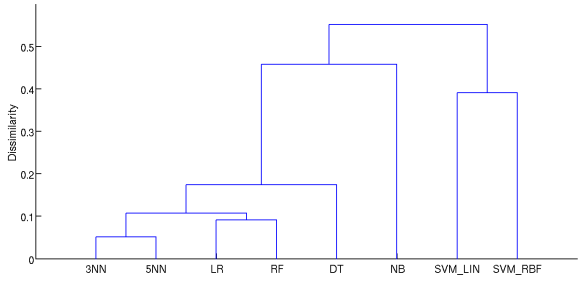


Figure 7. Clustered average score-driven performance

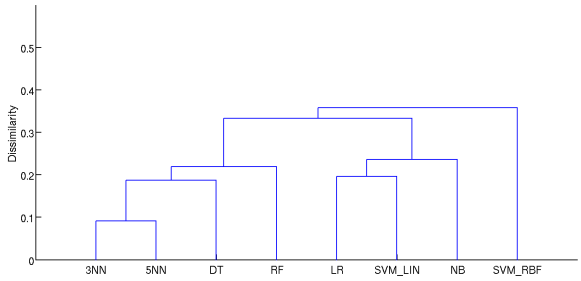


Figure 8. Clustered average rate-driven performance

9 Conclusion

In this work, we proposed similarity measures between algorithms by considering three threshold choice methods: score-fixed, score-driven and rate-driven. For each method, we proposed a corresponding instance hardness measure that was then deployed to define the algorithm similarity measure. The similarity between algorithms can be quite different depending on the dataset and the cost-sensitive scenario being tackled. For the score-fixed, two algorithms are similar if they produce the same classification for a high number of instances for a chosen threshold. For the score-driven method, two algorithms are similar if they produce similar scores. For the rate-driven method, in turn, two algorithms are similar if they produce similar rankings of instances.

In order to infer overall similarity on different datasets, we computed the average similarity between algorithms over 50 datasets and performed clustering of algorithms. The results revealed some unexpected similarities that can serve as starting points for further investigations to discover and explain hidden relationships between algorithms and learning strategies.

As future work, the ideas presented here can be applied on a predictive meta-learning strategy to select algorithms for new datasets depending on the threshold choice method and the input costs. In our work, we aggregate similarity across datasets using the mean method, which is simple but possibly naive. Other consensus similarity methods can be investigated. Finally, we highlight that our work is very limited concerning the number of algorithms and datasets adopted. Hence, more extensive meta-learning studies can be done in the future, with stronger conclusions and insights.

ACKNOWLEDGEMENTS

This research is supported by CNPq, CAPES and FACEPE (Brazilian agencies).

REFERENCES

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013.
[2] Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and Ricardo Vilalta, *Metalearning: Applications to Data Mining*, Springer Publishing Company, Incorporated, 1 edn., 2008.
[3] Francisco de A.T. de Carvalho, Yves Lechevallier, and Filipe M. de Melo, 'Relational partitioning fuzzy clustering algorithms based on multiple dissimilarity matrices', *Fuzzy Sets and Systems*, **215**(0), 1 – 28, (2013). Theme : Clustering.
[4] Chris Ding, Xiaofeng He, and Lawrence Berkeley, 'Cluster merging and splitting in hierarchical clustering algorithms', 139–146, (2002).
[5] Johannes Fürnkranz and Johann Petrak, '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).
[6] José Hernández-Orallo, Peter Flach, and Cèsar Ferri, 'Brier curves: A new cost-based visualisation of classifier performance', in *Proceedings of the 28th International Conference on Machine Learning*, (2011).
[7] José Hernández-Orallo, Peter Flach, and Cèsar Ferri, 'A unified view of performance metrics: Translating threshold choice into expected classification loss', *J. Mach. Learn. Res.*, **13**(1), 2813–2869, (October 2012).
[8] José Hernández-Orallo, Peter Flach, and César Ferri, 'ROC curves in cost space', *Machine Learning*, **93**(1), 71–91, (February 2013).
[9] A Kalousis, J Gama, and M Hilario, 'On data and algorithms: Understanding inductive performance', *Machine Learning*, 275–312, (2004).
[10] JW Lee and C Giraud-Carrier, 'A metric for unsupervised metalearning', *Intelligent Data Analysis*, **15**, 827–841, (2011).
[11] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren, 'Selecting classification algorithms with active testing', in *Machine Learning and Data Mining in Pattern Recognition*, 117–131, Springer, (2012).
[12] Yi Lin and Yongho Jeon, 'Random forests and adaptive nearest neighbors', *Journal of the American Statistical Association*, **101**(474), 578–590, (2006).
[13] Tony Martinez Michael R. Smith and Christophe Giraud-Carrier, 'An instance level analysis of data complexity', *Machine Learning*, (2013).
[14] 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).
[15] AH Peterson and TR Martinez, 'Estimating the potential for combining learning models', *Proceedings of the ICML workshop on meta-learning*, (2005).

Using Metalearning to Predict When Parameter Optimization Is Likely to Improve Classification Accuracy

Parker Ridd[1] and **Christophe Giraud-Carrier**[2]

Abstract. Work on metalearning for algorithm selection has often been criticized because it mostly considers only the default parameter settings of the candidate base learning algorithms. Many have indeed argued that the choice of parameter values can have a significant impact on accuracy. Yet little empirical evidence exists to provide definitive support for that argument. Recent experiments do suggest that parameter optimization may indeed have an impact. However, the distribution of performance differences has a long tail, suggesting that in most cases parameter optimization has little effect on accuracy. In this paper, we revisit some of these results and use metalearning to characterize the situations when parameter optimization is likely to cause a significant increase in accuracy. In so doing, we show that 1) a relatively simple and efficient landmarker carries significant predictive power, and 2) metalearning for algorithm selection should be effected in two phases, the first in which one determines whether parameter optimization is likely to increase accuracy, and the second in which algorithm selection actually takes place.

1 Introduction

The availability of a large number of classification learning algorithms together with the No Free Lunch theorem for classification present business users with a significant challenge, namely that of deciding which algorithm is likely to induce the most accurate model for their particular classification task. This selection process is further compounded by the fact that many classification learning algorithms include parameters, and the various possible settings of these parameters may give rise to models whose accuracy on the target classification task varies significantly. As a result, the algorithm selection problem in machine learning consists not only in choosing an algorithm, but rather in choosing an algorithm *and* an associated parameter setting. Formally, let:

- $\mathcal{L} = \{L_1, L_2, \ldots, L_n\}$ be a finite set of n classification learning algorithms (e.g., C4.5, Naïve Bayes-NB, Backpropagation-BP, Support Vector Machine-SVM).
- $\mathcal{P}_{L_i} = P^1_{L_i} \times P^2_{L_i} \times \ldots P^{k_i}_{L_i}$ be the set of parameter settings associated with L_i, where each $P^k_{L_i}$ represents one of the parameters of L_i (e.g., $P^1_{C4.5}$ =pruning indicator, $P^2_{C4.5}$ =splitting criterion, ..., P^1_{BP} =number of hidden layers, P^2_{BP} =learning rate, P^3_{BP} =momentum term, ...).
- $T = \mathcal{X} \times \mathcal{Y}$ be a training set for a classification task where $\mathcal{X}$ is a set of features and $\mathcal{Y}$ is a finite set of labels.
- $\mathcal{I}(L, P, T)$ be the model induced by algorithm L with parameter setting P on some classification learning task T. Hence, $\mathcal{I}$ maps objects in $\mathcal{X}$ to labels in $\mathcal{Y}$.
- $\mathcal{A}(I)$ be the predictive accuracy of model I (typically measured by cross-validation).

The classification learning algorithm selection problem can be formulated as follows.

Classification Learning Algorithm Selection Given a training set T for some classification learning task, find the pair (L^*, P^*) where $L^* \in \mathcal{L}$ and $P^* \in \mathcal{P}_{L^*}$, such that

$$\forall (L, P) \in \mathcal{L} \times \mathcal{P}_L \;\; \mathcal{A}(\mathcal{I}(L, P, T)) \leq \mathcal{A}(\mathcal{I}(L^*, P^*, T)).$$

The above formulation is generic in that it says nothing about the process used to search the combined spaces of classification learning algorithms and parameter settings to find the optimal algorithm. Metalearning for classification learning algorithm selection is the specific instance of that general problem wherein the search is effected by a learning algorithm [6]. In other words, the past performance of classification learning algorithms on a variety of tasks is used to induce a predictive model that takes as input a classification learning task and produces as output a classification learning algorithm and its associated parameter setting.

In practice, one has access to a set $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$ of m training sets (corresponding to m classification learning tasks). For each T_j, the learning algorithms in $\mathcal{L}$, together with their parameter settings, are used one at a time to induce a model on T_j. The pair of classification learning algorithm and parameter setting that maximizes $\mathcal{A}(\mathcal{I}(L, P, T_j))$ is recorded. Each T_j with its corresponding winning pair becomes a training example for a (meta)learning algorithm. Since storing complete learning tasks is unfeasible and likely undesirable, one uses instead some characterization of learning tasks by meta-features. Meta-features may be drawn from basic statistics and information-theoretic measures (e.g., ratio of nominal features, skewness, class entropy) [14, 7, 22], landmarking measures (i.e., performances of simple learners that serve as signpost for more complex ones) [2, 17, 8], and model-based measures (e.g., properties of induced decision trees) [1, 3, 16]. Given a training set T for some classification learning task, let $\mathcal{C}(T)$ be the characterization of T by some set of meta-features.

One can now take a number of classification tasks, characterize them via the function $\mathcal{C}$, and record the corresponding meta-features together with the accuracy of the best learning algorithm and associated parameter setting. The classification learning algorithm selec-

[1] Brigham Young University, USA, email: parker.ridd@byu.net
[2] Brigham Young University, USA, email: cgc@cs.byu.edu

tion problem, as solved by metalearning, can then be reformulated as follows.[3]

Metalearning for Classification Learning Algorithm Selection Given a set $\mathcal{T}^m = \{(\mathcal{C}(T), \text{argmax}_{L \in \mathcal{L}, P \in \mathcal{P}_L} \mathcal{A}(\mathcal{I}(L, P, T))\}$ of past classification learning episodes, together with a classification learning algorithm L^m, which may belong to $\mathcal{L}$, and an associated parameter setting P_{L^m}:

1. Construct $\mathcal{M} = \mathcal{I}(L^m, P_{L^m}, \mathcal{T}^m)$.
2. For any training set T', $(L^*, P^*) = \mathcal{M}(T')$.

By convention, let $P^0_{L_i}$ denote the default parameter setting of L_i, as determined by the implementation of L_i under consideration (e.g., Weka, IBM SPSS Modeler). Most of the work in metalearning so far has addressed the above problem with the further assumption that for all learning algorithms the parameter setting is fixed to its default. This, of course, creates a much restricted, yet also greatly simplified, version of the selection problem, since the large, possibly infinite, space of parameter settings need not be considered at all. However, that restriction has also been the source of much criticism, and sometimes dismissal, by a part of the machine learning community, who has maintained that:

Parameter Optimization Claim Parameter setting has a significant impact (for the better) on the predictive accuracy of classification learning algorithms.

It would seem that most metalearning researchers, and indeed most machine learning researchers, have taken this claim to be well founded, and considered ways to address it. There have been two main approaches.

- Two-stage Metalearning. Some metalearning researchers have adopted a two-stage approach to metalearning, wherein they continue to use the restricted form of the Metalearning for Classification Learning Algorithm Selection problem to choose a learning algorithm, but then follow up with an optimization phase to find the best set of parameter values for the selected algorithm.[4] The problem in this case, however, is that one may reach a suboptimal solution. Indeed, let L_1 and L_2 be two classification learning algorithms, such that, for some classification task T', $\mathcal{M}(T') = (L_1, P^0_{L_1})$. Then, L_1 would be selected and its parameter setting optimized to $P^*_{L_1}$. Yet, despite the fact that $\mathcal{A}(\mathcal{I}(L_1, P^0_{L_1}, T')) > \mathcal{A}(\mathcal{I}(L_2, P^0_{L_2}, T'))$ (assuming the metalearner is accurate), it is entirely possible that there exists some parameter setting $P^k_{L_2}$ of algorithm L_2 such that $\mathcal{A}(\mathcal{I}(L_1, P^*_{L_1}, T')) < \mathcal{A}(\mathcal{I}(L_2, P^k_{L_2}, T'))$. In other words, the early greedy commitment to L_1 makes it impossible to explore other parts of the combined spaces of learning algorithms and parameter settings.

- Unrestricted Metalearning. Other metalearning researchers have lifted the traditional restriction, and recently begun to design solutions from the unrestricted Metalearning for Classification Learning Algorithm Selection problem. Their systems seek to select both a learning algorithm an associated parameter setting (e.g., see [13, 24, 21]).

Interestingly, and somewhat surprisingly, very few researchers have bothered to check the validity of the Parameter Optimization Claim. Yet, to the best of our knowledge —and as presented by most, it is just that: a claim. We have been hard-pressed to find any systematic study in the literature that addresses the impact of parameter settings over a wide variety of classification learning algorithms. What if the Parameter Optimization Claim does not hold in general? What if it only holds in some specific cases? Would it not be useful to know what these cases are? And what about using metalearning to characterize when the claim holds? We address these questions here.

2 Impact of Parameter Optimization on Classification Learning Algorithm Performance

There is one exception to our statement that the literature contains no systematic study of the impact of parameter optimization on the performance of classification learning algorithms, found in [23]. In that paper, the authors considered 466 datasets and for each, computed the difference in accuracy between their default parameter setting and the best possible parameter setting after optimization. The optimization procedure is based on particle swarm optimization (PSO), wherein the authors specify which parameters should be optimized (e.g., kernel function and complexity constant for SVM) and the range of values that PSO should consider. We obtained the data from the authors and reproduced their results, with two small exceptions: 1) we found that one of the datasets in the list was redundant, so we removed it; and 2) our dataset characterization tool (see below) did not terminate on two of the datasets after several days so we stopped it, and omitted the corresponding datasets from our study. Hence, the results here are for 463 datasets. For each dataset, Figure 1 shows the percentage of improvement of the best AUC score among 20 classification learning algorithms after parameter optimization over the best AUC score among the same classification learning algorithms with their default parameter setting. The data is ordered by increasing value of improvement.

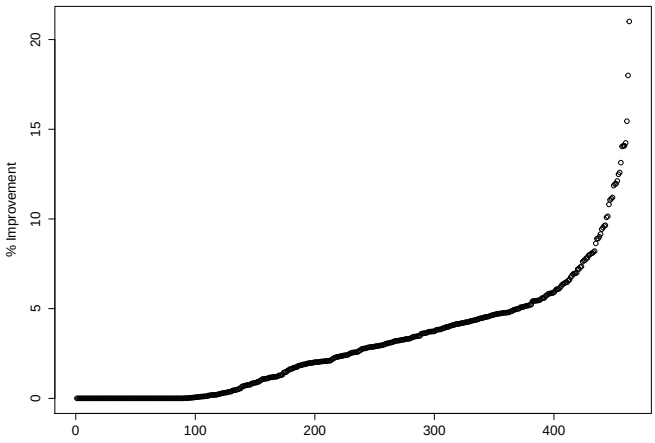


Figure 1. Impact of Parameter Optimization on 463 Datasets

Figure 1 makes it clear that the impact of parameter optimization

[3] We recognize that it is possible to use metalearning to predict rankings of learning algorithms (e.g., see [5, 23]), or even the actual performance of learning algorithms via regression (e.g., see [4, 10, 20]). We restrict our attention here to the prediction of a single best algorithm although much of the discussion extends naturally to these settings.

[4] Some have also simply picked a classification learning algorithm manually, and used metalearning to choose the best parameter setting (e.g., see [9, 19]).

is highly variable across datasets, and seems to be rather small for a large number of them. We are a little surprised that with such a skewed distribution, the authors of [23] concluded that "the result demonstrates the benefit of using the performances of optimised algorithms for generating algorithm rankings", and thus carried on with a blanket application of their combined classification learning algorithm / parameter setting selection approach.

To make the relationship even clearer, consider Figure 2 that shows the cumulative distribution of the same datasets, where each successive bar represents the proportion of datasets for which the improvement in AUC due to parameter optimization is less than or equal to the value indicated on the x-axis in increments of 1%.

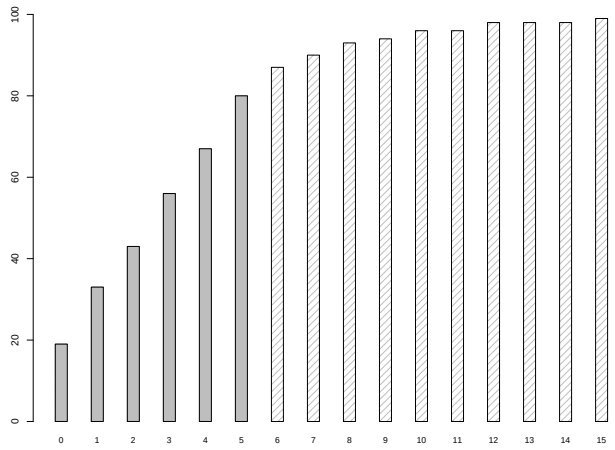


Figure 2. Cumulative Distribution of the Impact of Parameter Optimization on 463 Datasets

According to Figure 2, for 19% of the datasets considered parameter optimization offers no gain in performance. The ascent is actually very steep, as shown by the shaded portion of the distribution, reaching 80% of the datasets for an improvement in performance of no more than 5%. From 0% to 5%, the relationship is virtually linear (r=0.999) with a slope of 12. These results seem robust as an independent analysis of 129 datasets and 9 classification learning algorithms reveals that there is no gain in performance with optimization for about 15% of the datasets and 96% of the datasets exhibit less than 5% improvement.[5]

We note that the above analysis was not performed on a per-algorithm basis. As stated, the differences in performance are computed from among the best in 20 algorithms, which means that the best optimized version could be obtained with algorithm A, while the best default version for the same dataset would be obtained with algorithm B. It is possible, however, that some classification learning algorithms are more sensitive to parameter settings and may thus be more likely to exhibit significant differences with parameter optimization. It may be worthwhile in a future study to consider such algorithm-level variation. In spite of this limitation, several conclusions seem inescapable from the foregoing analysis:

1. Parameter optimization does not improve performance uniformly across datasets.
2. For many datasets, parameter optimization yields very little improvement and may be viewed as computational overkill.
3. For a few datasets, parameter optimization makes a significant difference and should thus be performed.

From a practitioner's standpoint, the last two conclusions are particularly relevant. If no improvement is to be expected from parameter optimization, then one would gladly save the extra computational time required to effect it. Conversely, if a significant improvement can be expected and one is focused on maximizing predictive accuracy, then one would be willing to bear the extra cost. The question then, for such practitioners, is: Will the predictive accuracy on the classification learning task I am considering be improved by parameter optimization?

It should be obvious to machine learning researchers that if one labels the datasets falling under conclusion 2 above as *no advantage* and the datasets falling under conclusion 3 as *advantage*, one obtains a training dataset for a classification learning task. We propose to do exactly that and essentially take our own medicine, by applying machine learning to the problem of distinguishing between datasets that may benefit from parameter optimization and datasets that would not. Because our data consists of information about the performance of learning algorithms, this is an instance of metalearning.

3 Metalearning the Impact of Parameter Optimization

As per standard metalearning practice, we build our training metadata by characterizing each of our 463 datasets by a set of meta-features. However, because smaller datasets may produce unreliable meta-features, we remove from the analysis all of the datasets containing less than 100 instances. This leaves us with 326 datasets.

We use an existing R script that, for each dataset, creates a meta-feature vector by extracting over 68 meta-features, including statistical and information-theoretic meta-features, landmarkers, model-based meta-features, and timing information [18]. Prior to our experiments, we manually remove a number of meta-features that carry little information in this context (e.g., timing data, model-based meta-features, redundant meta-features). In each experiment, we also use correlation-based feature subset selection (CFS) [11], as implemented by the function `CfsSubsetEval` in Weka [12], to further reduce the number of meta-features.

The metalearning task consists in distinguishing datasets where parameter optimization is deemed to offer no advantage (class 0) from datasets where parameter optimization is likely to yield a performance advantage (class 1). Each meta-feature vector is labeled appropriately based on the observed difference in performance over its corresponding dataset. We use various threshold values to separate class 1 from class 0, as shown below.

There are two types of error our (meta)models can make. A false positive (FP) error is made when the model considers a class 0 dataset but labels it as class 1. When this happens, there is wasted effort in performing parameter optimization when no significant gain will be achieved by such optimization. Conversely, a false negative (FN) error is made when the model considers a class 1 dataset but labels it as class 0. When this happens, valuable parameter optimization is omitted and there is a resulting loss in predictive accuracy in the classification task under consideration. Assuming that a practitioner's ultimate goal is to get the best accuracy possible on their specific classification task, one can argue that FN errors are more costly than FP errors, and thus should be minimized. This is, of course, is equivalent to maximizing recall, $R = \frac{TP}{TP+FN}$ (where TP is the number of

[5] There may be some overlap in the datasets used in this study and those used in [23], although the datasets were not transformed into binary classification tasks in the former as they were in the latter.

true positive classifications). Yet, one must be careful as it is trivial to maximize R by simply assigning all classification tasks to class 1. Clearly, this would defeat the purpose of the model and is unacceptable due to the unnecessary computational burden it places on the system. Hence, we focus on obtaining models with high recall, but also good precision, $P = \frac{TP}{TP+FP}$.

We are faced at the metalevel with the same challenge of selecting a (meta)learning algorithm adequate for the task at hand. We are guided here by a desire for comprehensibility of the induced model and the results of preliminary experiments. Hence, we use for our metalearner, a decision tree learning algorithm, specifically Weka's J48, and implement the training and testing processes in RapidMiner [15]. Interestingly, J48 also resulted in higher accuracy than other algorithms such as SVM and Random Forest.

3.1 Threshold = 1.5

We begin by setting the threshold relatively low, assuming that there is an advantage to parameter optimization if the performance improvement exceeds 1.5%.

The default accuracy in this case is 61.96%, obtained by predicting class 1 uniformly. This, of course, produces maximum recall, R=100%, but very poor precision, P=61.96%.

Applying CFS selects the following mixed set of meta-features: `attributes`, `kurtosis`, `joint entropy`, `NB`, `LDA` and `NN1`. Further experimentation shows that performance can be improved still by using only `joint entropy`, `NB` and `NN1`. The confusion matrix is as follows.

		Predicted	
		0	**1**
Actual	**0**	84	40
	1	28	174

This yields an accuracy of 79.17%, significantly above default, with both high recall R=86.14% and high precision P=81.31%. The decision tree is shown below.

```
nn_1 <= 0.902439
|   joint_entropy <= 0.606044: 0 (14.0/2.0)
|   joint_entropy > 0.606044
|   |   naive_bayes <= 0.952: 1 (208.0/30.0)
|   |   naive_bayes > 0.952
|   |   |   nn_1 <= 0.71134
|   |   |   |   joint_entropy <= 2.08461: 0 (2.0)
|   |   |   |   joint_entropy > 2.08461: 1 (5.0)
|   |   |   nn_1 > 0.71134: 0 (12.0/2.0)
nn_1 > 0.902439: 0 (85.0/15.0)
```

To further test the metamodel, we collected 42 independent datasets with more than 100 instances each. It is possible that some of these correspond to some of the tasks used in the training data. However, all 463 training tasks have been binarized [23], while these were left untouched. Hence, we are training on 2-class datasets and testing on n-class datasets, which may be somewhat unfair, but still interesting.

We extracted the meta-features of the 42 datasets, and ran the corresponding meta-feature vectors against the metamodel.[6] The accuracy on the test datasets is 54.76%, which is rather poor given a default accuracy of 73.81%. However, the default is obtained by predicting class 0 uniformly. While the test accuracy is not very good, recall is very high at R=90.91% (compared to R=0 for the default), with 10 of the 11 datasets of class 1 being predicted in class 1. This suggests that the model has picked up useful information to identify learning tasks where parameter optimization would be advantageous (i.e., expected improvement greater than 1.5%).

In addition to its own performance at the metalevel, we consider how the metamodel affects performance at the base level. To do so, we compare $maxImp$, the total amount of possible improvement due to parameter optimization to $predImp$, the amount of improvement one would obtain by using the metamodel. For each dataset d, let l_d be d's label, p_d be d's predicted class, and I_d be the improvement one would experience if parameter optimization were used with d. Then,

$$maxImp = \sum_d I_d$$

i.e., $maxImp$ is the sum of the individual improvement values across all of our 326 datasets. Here, $maxImp = 949.26$. Similarly,

$$predImp = \sum_d \begin{cases} I_d & \text{if } (p_d = l_d) \wedge (l_d = 1) \\ 0 & \text{otherwise} \end{cases}$$

i.e., $predImp$ is the sum of the improvement values for all datasets where the metamodel makes the correct class 1 prediction. We do not include correct class 0 predictions as these would artificially inflate the results. Here, $predImp = 789.92$. Hence, the metamodel would allow us to claim 83.21% of the total performance improvement available at the base level.

3.2 Threshold=2.5

We next raise the threshold, assuming that there is an advantage to parameter optimization if the performance improvement exceeds 2.5%.

The default accuracy in this case is 50.31%, obtained by predicting class 1 uniformly. This, again, produces maximum recall, R=100%, but very poor precision, P=50.31%.

Applying CFS selects the following mixed set of meta-features: `kurtosis prep`, `normalized attribute entropy`, `joint entropy`, `NB`, `LDA`, `stump min gain` and `NN1`. Further experimentation shows that performance can be improved still by using only `kurtosis prep`, `normalized attribute entropy`, `joint entropy`, `NB` and `NN1`. The confusion matrix is as follows.

		Predicted	
		0	**1**
Actual	**0**	105	57
	1	26	138

This yields an accuracy of 74.52%, significantly above default, with both high recall R=84.15% and high precision P=70.77%. The decision tree is shown below.

```
nn_1 <= 0.857143
|   joint_entropy <= 0.606044
|   |   joint_entropy <= 0.508598
|   |   |   kurtosis_prep <= 26.479217: 1 (2.0)
|   |   |   kurtosis_prep > 26.479217: 0 (2.0)
|   |   joint_entropy > 0.508598: 0 (8.0)
|   joint_entropy > 0.606044
|   |   naive_bayes <= 0.969466: 1 (194.0/49.0)
|   |   naive_bayes > 0.969466
|   |   |   normalized_attribute_entropy <= 0.84991
|   |   |   |   kurtosis_prep <= 45.518635: 0 (2.0)
|   |   |   |   kurtosis_prep > 45.518635: 1 (2.0)
|   |   |   normalized_attribute_entropy > 0.84991: 0 (7.0)
nn_1 > 0.857143: 0 (109.0/15.0)
```

[6] As the results for the test datasets were obtained with a different set of classification learning algorithms and a different parameter optimization procedure, we are not entirely sure the comparison is fair. Yet, we feel there is value in including these results.

As with the threshold value of 1.5, using the metamodel against the test datasets produces poor accuracy (35.71% for a default of 88.10%), but recall is significantly better with R=60% (3 of the 5 datasets in class 1 are predicted correctly) against a default of R=0.

Considering performance at the base level, we have here $predImp = 698.61$, so that the metamodel would allow us to still claim 73.60% of the total performance improvement available.

3.3 Larger Threshold Values

Based on the distribution of performance improvements, raising the threshold will cause the majority class to shift to 0 and result in far fewer class 1 datasets. On the other hand, while fewer, correctly identifying these datasets offers the highest benefit to the practitioner as the expected improvement in accuracy with parameter optimization is rather significant. However, the metalearning task is also more difficult as the class distribution of the training data is rather skewed.

Setting the threshold to 5.0% results in a default accuracy of 83.44% with a model that predicts class 0 uniformly. While it was not possible in this case to improve on the default accuracy with the available set of meta-features, recall rose to R=25.94% (10 of the 54 datasets in class 1 are predicted correctly). Interestingly, CFS selected `LDA` and `NN1`, and while the tree induced without feature selection is a little larger than the previous ones, it also splits on `NN1` at the root node. Using the metamodel against the test datasets produces reasonable accuracy (76.19%) but only because the test data is skewed in favor of class 0. Only two datasets belong to class 1 and neither is predicted correctly.

Setting the threshold to 10.0% results in a default accuracy of 96.93% again with a model that predicts class 0 uniformly. Only 10 datasets belong to class 1, making for a very imbalanced class distribution. Using all the meta-features, the induced decision tree has a slightly improved accuracy (97.85%), with recall R=30%. The root of the tree is `mutual information`. There are no datasets in our test set for which the improvement with parameter optimization exceeds 10%.

4 Discussion

As mentioned previously, one of the biggest issues with parameter optimization is the amount of time it takes to optimize the parameters. In [24], it took 6,000 single core CPU hours to optimize the parameters of all their datasets, which means that it took on average about 13 hours per dataset. For obvious reasons, a practitioner would probably not wish to wait for that amount of time when it may result in little or no improvement from the models baseline performance.

What this short study demonstrates is that it is possible to predict, with good recall value, whether parameter optimization will significantly increase a classification learning models accuracy by using the metafeatures of any particular dataset, which are less computationally intensive than the parameter optimization procedure. Of course, as the threshold increases, or as the expected performance improvement gets larger, the prediction becomes less accurate. Nevertheless, it is better than default.

One of the unexpected findings of our study, valid across almost all metamodels, is that the root node of the decision tree is `NN1`. In other words, a dataset's performance on `NN1` is indicative of whether parameter optimization will significantly improve the performance of learning algorithms trained against it. There are at least two interesting things about this finding. First, it confirms prior work on metalearning that suggest that landmarking meta-features are generally better than other meta-features, especially `NN1` and `NB` [20]. Second, and significantly more valuable, is that `NN1` is a very efficient algorithm, suggesting that it would be very fast to determine whether parameter optimization is likely to improve accuracy for any dataset.

To further test the predictive power of `NN1`, we ran Weka's linear regression on our dataset, without CFS, and with the target set to the actual percentage of accuracy improvement (`Imp`) observed for each dataset. The resulting model is shown below.

```
Imp =
      11.4405 * class_prob_min +
       0.1626 * skewness_prep +
      -0.0051 * kurtosis_prep +
      -2.003  * cancor_1 +
      -6.3391 * class_entropy +
      -8.2398 * mutual_information +
      -0.0008 * noise_signal_ratio +
      -0.5321 * attribute_mean +
      -5.7852 * stump_min +
      -4.8382 * stump_max +
      13.3081 * stump_mean +
       8.4397 * stump_sd +
       3.3769 * stump_min_gain +
      -7.5918 * nn_1 +
       6.4233
```

The correlation coefficient of the model is 0.40, and the relatively large multiplicative factor associated with `NN1` in the model suggests its influence on the value of `Imp`. Furthermore, if we regress `Imp` on `NN1` alone, we obtain the following very simple model.

```
Imp =
      -8.3476 * nn_1 +
       9.3173
```

The correlation coefficient of this simpler model is 0.44. There is still error in this model, and it will be interesting to see whether using only the `NN1` metafeature could in general reasonably predict the datasets improvement if classification learning algorithm parameters are optimized.

5 Conclusion

We have showed that the observed improvement in performance due to parameter optimization in classification learning has a rather skewed distribution, with most datasets seeing none or very little improvement. However, there remains a portion of datasets for which parameter optimization has a significant impact. We have used metalearning to build a model capable of predicting whether parameter optimization can be expected to improve classification accuracy. As a result, it would seem that both current approaches to metalearning, the one that ignores parameter optimization and considers only default settings, and the other that applies parameter optimization in all cases, are misinformed. Instead, a two-phase approach where one first determines whether parameter optimization is likely to produce an improvement, and then if so applies it, is more appropriate.

Furthermore, one unexpected and very interesting side effect of our metalearning study is that our results show that good recall can be achieved at the metalevel, and that, with a very small set of efficient meta-features. In particular, it would appear that `NN1` may serve as a great landmarker in this context.

For future work, it may be valuable to revisit error costs. In particular, we have argued here that FN errors were more costly than FP errors on the basis that practitioners are likely to wish to obtain the highest possible accuracy on their classification tasks. This is certainly also true when the computational cost of parameter optimization is prohibitive. Were computational cost not to matter, the best decision would be to always perform parameter optimization, and

there would consequently be no need for a metamodel. In practice, however, it is well-known that parameter optimization is computationally intensive. As a result, FP errors may actually be rather costly, and possibly even more so than FN errors. In any case, there would seem to exist a range of operating conditions between these extremes, or different cost-benefits between FN and FP associated with parameter optimization. It may be interesting to perform a cost-sensitive analysis at the metalevel, for example, relating expected gain in performance (or threshold) and incurred cost. We cannot do this with the data available since we only have final performance improvement (after some fixed search time). We would need to re-run experiments to gather information about improvement over time allowed for optimization.

REFERENCES

[1] Bensusan, H. (1998). Odd Bites into Bananas Don't Make You Blind: Learning about Simplicity and Attribute Addition. In *Proceedings of the ECML'98 Workshop on Upgrading Learning to the Meta-level: Model Selection and Data Transformation*, 30-42.

[2] Bensusan, H. and Giraud-Carrier, C. (2000). Discovering Task Neighbourhoods through Landmark Learning Performances. In *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases, LNAI 1910*, 325-330.

[3] Bensusan, H., Giraud-Carrier, C. and Kennedy, C. (2000). A Higher-order Approach to Meta-learning. In *Proceedings of the ECML-2000 Workshop on Meta-learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, 109-118.

[4] Bensusan, H. and Kalousis, A. (2001). Estimating the Predictive Accuracy of a Classifier. In *Proceedings of the Twelfth European Conference on Machine Learning (LNCS 2167)*, 25-36.

[5] Brazdil, P., Soares, C. and Pinto da Costa, J. (2003). Ranking Learning Algorithms: Using IBL and Meta-Learning on Accuracy and Time Results. *Machine Learning*, **50**(3):251-277.

[6] Brazdil, P., Giraud-Carrier, C., Soares, C. and Vilalta, R. (2009). *Metalearning: Applications to Data Mining*, Springer.

[7] Engels, R. and Theusinger, C. (1998). Using a Data Metric for Offering Preprocessing Advice in Data-mining Applications. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, 430-434.

[8] Fuernkranz, J. and Petrak, J. (2001). An Evaluation of Landmarking Variants. In *Proceedings of the ECML/PKDD-01 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-learning*, 57-68.

[9] Gomes, T.A.F., Prudêncio, R.B.C., Soares, C., Rossi, A.L.D. and Carvalho, A. (2012). Combining Meta-learning and Search Techniques to Select Parameters for Support Vector Machines. *Neurocomputing*, **75**:3-13.

[10] Guerra, S.B., Prudêncio, R.B.C. and Ludermir, T.B. (2008). Predicting the Performance of Learning Algorithms Using Support Vector Machines as Meta-regressors. In *Proceedings of the Eighteenth International Conference on Artificial Neural Networks (LNCS 5163)*, 523-532.

[11] Hall, M.A. (1999). Correlation-based Feature Subset Selection for Machine Learning. PhD Thesis, Department of Computer Science, The University of Waikato, Hamilton, New Zealand.

[12] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I.H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, **11**(1):10-18.

[13] Leite, R., Brazdil, P. and Vanschoren, J. (2012). Selecting Classification Algorithms with Active Testing. In *Proceedings of the Eighth International Conference on Machine Learning and Data Mining in Pattern Recognition (LNCS 7376)*, 117-131.

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

[15] North, M. (2012). *Data Mining for the Masses*, Global Textbook Project.

[16] Peng, Y., Flach, P.A., Brazdil, P. and Soares, C. (2002). Improved Data Set Characterisation for Meta-learning. In *Proceedings of the Fifth International Conference on Discovery Science*, 141-152.

[17] Pfahringer, B., Bensusan, H. and Giraud-Carrier, C. (2000). Meta-learning by Landmarking Various Learning Algorithms. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 743-750.

[18] Reif, M. (2012). A Comprehensive Dataset for Evaluating Approaches of various Meta-Learning Tasks. In *Proceedings of the First International Conference on Pattern Recognition Applications and Methods*, 273-276.

[19] Reif, M., Shafait, F. and Dengel, A. (2012). Meta-learning for Evolutionary Parameter Optimization of Classifiers. *Machine Learning*, **87**(3):357-380.

[20] Reif, M., Shafait, F., Goldstein, M., Breuel, T. and Dengel, A. (2014). Automatic Classifier Selection for Non-experts. *Pattern Analysis & Applications*, **17**(1):83-96.

[21] Smith, M.R., Mitchell, L., Giraud-Carrier, C. and Martinez, T. (2014). Recommending Learning Algorithms and Their Associated Hyperparameters. Submitted.

[22] Sohn, S.Y. (1999). Meta Analysis of Classification Algorithms for Pattern Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **21**(11):1137-1144.

[23] Sun, Q. and Pfahringer, B. (2013). Pairwise Meta-rules for Better Meta-learning-based Algorithm Ranking. *Machine Learning*, **93**(1):141-161.

[24] Thornton, C., Hutter, F., Hoos, H.H. and Leyton-Brown, K. (2013). Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proceedings of the Nineteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 847-855.

Surrogate Benchmarks for Hyperparameter Optimization

Katharina Eggensperger[1] and **Frank Hutter**[1] and **Holger H. Hoos**[2] and **Kevin Leyton-Brown**[2]

Abstract. Since hyperparameter optimization is crucial for achieving peak performance with many machine learning algorithms, an active research community has formed around this problem in the last few years. The evaluation of new hyperparameter optimization techniques against the state of the art requires a set of benchmarks. Because such evaluations can be very expensive, early experiments are often performed using synthetic test functions rather than using real-world hyperparameter optimization problems. However, there can be a wide gap between the two kinds of problems. In this work, we introduce another option: cheap-to-evaluate surrogates of real hyperparameter optimization benchmarks that share the same hyperparameter spaces and feature similar response surfaces. Specifically, we train regression models on data describing a machine learning algorithm's performance under a wide range of hyperparameter configurations, and then cheaply evaluate hyperparameter optimization methods using the model's performance predictions in lieu of the real algorithm. We evaluate the effectiveness for using a wide range of regression techniques to build these surrogate benchmarks, both in terms of how well they predict the performance of new configurations and of how much they affect the overall performance of hyperparameter optimizers. Overall, we found that surrogate benchmarks based on random forests performed best: for benchmarks with few hyperparameters they yielded almost perfect surrogates, and for benchmarks with more complex hyperparameter spaces they still yielded surrogates that were qualitatively similar to the real benchmarks they model.

1 Introduction

The performance of many machine learning methods depends critically on hyperparameter settings and thus on the method used to set such hyperparameters. Recently, sequential model-based Bayesian optimization methods, such as SMAC[16], TPE[2], and Spearmint[29] have been shown to outperform more traditional methods for this problem (such as grid search and random search [3]) and to rival—and in some cases surpass—human domain experts in finding good hyperparameter settings [29, 30, 5]. One obstacle to further progress in this nascent field is a paucity of reproducible experiments and empirical studies. Until recently, a study introducing a new hyperparameter optimizer would typically also introduce a new set of hyperparameter optimization benchmarks, on which the optimizer would be demonstrated to achieve state-of-the-art performance (as compared to, e.g., human domain experts). The introduction of the hyperparameter optimization library (HPOlib [8]), which offers a unified interface to different optimizers and benchmarks, has made it easier to reuse previous benchmarks and to systematically compare different approaches [4].

[1] University of Freiburg, email:{eggenspk,fh}@cs.uni-freiburg.de
[2] University of British Columbia, email: {hoos,kevinlb}@cs.ubc.ca

However, a substantial problem remains: performing a hyperparameter optimization experiment requires running the underlying machine learning algorithm, often at least hundreds of times. This is infeasible in many cases. The first (mundane, but often significant) obstacle is to get someone else's research code working on one's own system—including resolving dependencies and acquiring required software licenses—and to acquire the appropriate input data. Furthermore, some code requires specialized hardware; most notably, general-purpose graphics processing units (GPGPUs) have become a standard requirement for the effective training of modern deep learning architectures [20, 21]. Finally, the computational expense of hyperparameter optimization can be prohibitive for research groups lacking access to large compute clusters. These problems represent a considerable barrier to the evaluation of new hyperparameter optimization algorithms on the most challenging and interesting hyperparameter optimization benchmarks, such as deep belief networks [2], convolutional neural networks [29, 5], and combined model selection and hyperparameter optimization in machine learning frameworks [30].

Given this high overhead for studying complex hyperparameter optimization benchmarks, most researchers have drawn on simple, synthetic test functions from the global continuous optimization community [12]. While these are simple to use, they are often poorly representative of the hyperparameter optimization problem: in contrast to the response surfaces of actual such problems, these synthetic test functions are smooth and often have unrealistic shapes. Furthermore, they only involve real-valued parameters and hence do not incorporate the categorical and conditional parameters typical of actual hyperparameter optimization benchmarks.

In the special case of small, finite hyperparameter spaces, a much better alternative is simply to record the performance of every hyperparameter configuration, thereby speeding future evaluations via a table lookup. The result is a perfect *surrogate* of an algorithm's true performance that takes time $O(1)$ to compute (using a hash) and that can be used in place of actually running the algorithm and evaluating its performance. This table-based surrogate can trivially be transported to any new system, without the complicating factors involved in running the original algorithm (setup, special hardware requirements, licensing, computational cost, etc.). In fact, several researchers have already applied this approach to simplifying their experiments: for example, Bardenet et al. [1] saved the performances of a parameter grid with 108 points of Adaboost on 29 datasets, and Snoek et al. [29] saved the performance of parameter grids with 1400 and 288 points for a structured SVM [31] and an online LDA [13], respectively. The latter two benchmarks are part of HPOlib and are, in fact, HPOlib's most frequently used benchmarks, due to their simplicity of setup and low computational cost.

Of course, the drawback of this table lookup idea is that it is limited

to small, finite hyperparameter spaces. Here, we generalize the idea of machine learning algorithm surrogates to arbitrary, potentially high-dimensional hyperparameter spaces (including, e.g., real-valued, categorical, and conditional hyperparameters). As in the table-lookup strategy, we first evaluate many hyperparameter configurations during an expensive offline phase. We then use the resulting performance data to train a regression model to approximate future evaluations via model predictions. As before, we obtain a surrogate of algorithm performance that is cheap to evaluate and trivially portable. However, model-based surrogates offer only *approximate* representations of performance. Thus, a key component of our work presented in the following is an investigation of the quality of these approximations.

We are not the first to propose the use of learned surrogate models that stand in for computationally complex functions. In the field of metalearning [6], regression models have been extensively used to predict the performance of algorithms across various datasets based on dataset features [11, 26]. The statistics literature on the design and analysis of computer experiments (DACE) [27, 28] uses similar surrogate models to guide a sequential experimental design strategy aiming to achieve either an overall strong model fit or to identify the minimum of a function. Similarly, the SUrrogate MOdeling (SUMO) Matlab toolkit[10] provides an environment for building regression models to describe the outputs of expensive computer simulations based on active learning. Such an approach for finding the minimum of a blackbox function also underlies the sequential model-based Bayesian optimization framework [7, 16] (SMBO, the framework underlying all hyperparameter optimizers we study here). While all of these lines of work incrementally construct surrogate models of a function in order to inform an active learning criterion that determines new inputs to evaluate, our work differs in its goals: We train surrogates on a set of data gathered offline (by some arbitrary process—in our case the combination of many complete runs of several different SMBO methods plus random search) and use the resulting surrogates as stand-in models for the entire hyperparameter optimization benchmark.

The *surrogate benchmarks* resulting from our work can be used in several different ways. Firstly, like synthetic test functions and table lookups, they can be used for extensive debugging and unit testing. Since the large computational expense of running hyperparameter optimizers is typically dominated by the cost of evaluating algorithm performance under different selected hyperparameters, our benchmarks can also substantially reduce the time required for running a hyperparameter optimizer, facilitating whitebox tests of an optimizer using exactly the hyperparameter space of the machine learning algorithm whose performance is modelled by the surrogate. This functionality is gained even if the surrogate model only fits algorithm performance quite poorly (e.g., due to a lack of sufficient training data). Finally, a surrogate benchmark whose model fits algorithm performance very well can also facilitate the evaluation of new features inside the hyperparameter optimizer, or even the (meta-)optimization of a hyperparameter optimizer's own hyperparameters (which can be useful even without the use of surrogates, but is typically extremely expensive [17]).

The rest of this paper is laid out as follows. We first provide some background on hyperparameter optimization (Section 2). Then, we discuss our methodology for building surrogate benchmarks (Section 3) using several types of machine learning models. Next, we evaluate the performance of these surrogates in practice (Section 4). We demonstrate that random forest models tend to fit the data better than a broad range of competing models, both in terms of raw predictive model performance and in terms of the usefulness of the resulting surrogate benchmark for comparing hyperparameter optimization procedures.

2 Background: Hyperparameter Optimization

The construction of machine learning models typically gives rise to two optimization problems. The first is internal optimization, such as selecting a neural network's likelihood-maximizing weights; the second is tuning the method's hyperparameters, such as setting a neural network's regularization parameters or number of neurons. The former problem is closely coupled with the machine learning algorithm at hand and is very well studied; here, we consider the latter. Let $\lambda_1, \ldots, \lambda_n$ denote the hyperparameters of a given machine learning algorithm, and let $\Lambda_1, \ldots, \Lambda_n$ denote their respective domains. The algorithm's hyperparameter space is then defined as $\boldsymbol{\Lambda} = \Lambda_1 \times \cdots \times \Lambda_n$. When trained with hyperparameters $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ on data $\mathcal{D}_{\text{train}}$, the algorithm's loss (e.g., misclassification rate) on data $\mathcal{D}_{\text{valid}}$ is $\mathcal{L}(\boldsymbol{\lambda}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}})$. Using k-fold cross-validation, the optimization problem is then to minimize:

$$f(\boldsymbol{\lambda}) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(\boldsymbol{\lambda}, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)}). \quad (1)$$

A hyperparameter λ_n can have one of several types, such as continuous, integer-valued or categorical. For example, the learning rate for a neural network is continuous; the random seed given to initialize an algorithm is integer-valued; and the choice between various preprocessing methods is categorical. Furthermore, there can be *conditional* hyperparameters, which are only active if another hyperparameter takes a certain value; for example, the hyperparameter "number of principal components" only needs to be instantiated when the hyperparameter "preprocessing method" is PCA.

Evaluating $f(\boldsymbol{\lambda})$ for a given $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ is computationally costly, and so many techniques have been developed to find good configurations $\boldsymbol{\lambda}$ with few function evaluations. The methods most commonly used in practice are manual search and grid search, but recently, it has been shown that even simple random search can yield much better results [3]. The state of the art in practical optimization of hyperparameters is defined by Bayesian optimization methods [16, 29, 2], which have been successfully applied to problems ranging from deep neural networks to combined model selection and hyperparameter optimization [2, 29, 30, 19, 5].

Bayesian optimization methods use a probabilistic model $\mathcal{M}$ to model the relationship between a hyperparameter configuration $\boldsymbol{\Lambda}$ and its performance $f(\boldsymbol{\lambda})$. They fit this model using previously gathered data and then use it to select a next point $\boldsymbol{\lambda}_{\text{new}}$ to evaluate, trading off exploitation and exploration in order to find the minimum of f. They then evaluate $f(\boldsymbol{\lambda}_{\text{new}})$, update $\mathcal{M}$ with the new data $(\boldsymbol{\lambda}_{\text{new}}, f(\boldsymbol{\lambda}_{\text{new}}))$ and iterate. Throughout this paper, we will use the following three instantiations of Bayesian optimization:

SPEARMINT [29] is a prototypical Bayesian optimization method that models $p_{\mathcal{M}}(f \mid \boldsymbol{\lambda})$ with Gaussian process (GP) models. It supports continuous and discrete parameters (by rounding), but no conditional parameters.

Sequential Model-based Algorithm Configuration (SMAC) [16] models $p_{\mathcal{M}}(f \mid \boldsymbol{\lambda})$ with random forests. When performing cross validation, SMAC only evaluates as many folds as necessary to show that a configuration is worse than the best one seen so far (or to replace it). SMAC can handle continuous, categorical, and conditional parameters.

Tree Parzen Estimator (TPE) [2] models $p_{\mathcal{M}}(f \mid \boldsymbol{\lambda})$ indirectly. It models $p(f < f^*)$, $p(\boldsymbol{\lambda} \mid f < f^*)$, and $p(\boldsymbol{\lambda} \mid f \geq f^*)$, where f^* is defined as a fixed quantile of the function values observed so

far, and the latter two probabilities are defined by tree-structured Parzen density estimators. TPE can handle continuous, categorical, and conditional parameters.

An empirical evaluation on the three methods on the HPOlib hyperparameter optimization benchmarks showed that SPEARMINT performed best on benchmarks with few continuous parameters and SMAC performed best on benchmarks with many, categorical, and/or conditional parameters, closely followed by TPE. SMAC also performed best on benchmarks that relied on cross-validation [8].

3 Methodology

We now discuss our approach, including the algorithm performance data we used, how we preprocessed the data, the types of regression models we evaluated, and how we used them to construct surrogate benchmarks.

3.1 Data collection

In principle, we could construct surrogate benchmarks using algorithm performance data gathered by any means. For example, we could use existing data from a manual exploration of the hyperparameter space, or from an automated approach, such as grid search, random search or one of the more sophisticated hyperparameter optimization methods discussed in Section 2.

It is more important for surrogate benchmarks to exhibit strong predictive quality in some parts of the hyperparameter space than in others. Specifically, our ultimate aim is to ensure that *hyperparameter optimizers perform similarly on the surrogate benchmark as on the real benchmark*. Since most optimizers spend most of their time in high-performance regions of the hyperparameter space, and since relative differences between the performance of hyperparameter configurations in such high-performance regions tend to impact which hyperparameter configuration will ultimately be returned, accuracy in this part of the space is more important than in regions of poor performance. The training data should therefore densely sample high-performance regions. We thus advocate collecting performance data primarily via runs of existing hyperparameter optimization procedures. As an additional advantage of this strategy, we can obtain this costly performance data as a by-product of executing hyperparameter optimization procedures on the original benchmark.

Of course, it is also important to accurately identify poorly performing parts of the space: if we only trained on performance data for the very best hyperparameter settings, no machine learning model could be expected to infer that performance in the remaining parts of the space is poor. This would typically lead to underpredictions of performance in poor parts of the space. We thus also included performance data gathered by a random search. (An alternative is grid search, which can also cover the entire space. We did not adopt this approach because it cannot deal effectively with large hyperparameter spaces.) To gather the data for each surrogate benchmark in this paper, we therefore executed $r = 10$ runs of each of the three Bayesian optimization methods described in Section 2 (each time with a different seed), as well as random search, with each run gathering the performance of a fixed number of configurations.

3.2 Data preprocessing

For each benchmark we studied for this paper, after running the hyperparameter optimizers and random search, we preprocessed the data as follows:

Table 1. Overview of evaluated regression algorithms. When we used random search to optimize hyperparameters, we considered 100 samples over the stated hyperparameters (their names refer to the SCIKIT-LEARN implementation [25]); the model was trained on 50% of the data, and the best configuration was chosen based on the performance on the other 50% and then trained on all data.

Model	Hyperparameter optimization	Impl.
Random Forest	None	[25]
Gradient Boosting	None	[25]
Extra Trees	None	[25]
Gaussian Process	MCMC sampling over hyperparameters	[29]
SVR	Random search for `C` and `gamma`	[25]
NuSVR	Random search for `C`, `gamma` and `nu`	[25]
Bayesian Neural Network	None	[24]
k-nearest-neighbours	Random search for `n_neighbors`	[25]
Linear Regression	None	[25]
Least Angle Regression	None	[25]
Ridge Regression	None	[25]

1. We extracted all available configuration/performance pairs from the runs. For benchmarks that used cross-validation, we encoded the cross-validation fold of each run as an additional categorical parameter (for benchmarks without cross validation, that parameter was set to a constant).
2. We removed entries with invalid results caused by algorithm crashes. Since some regression models used in preliminary experiments could not handle duplicated configurations, we also deleted these, keeping the first occurrence.
3. For data from benchmarks featuring conditional parameters, we replaced the values of inactive conditional parameters with a default value.
4. To code categorical parameters, we used a one-hot (aka 1-in-k) encoding, which replaces any single categorical parameter λ with domain $\Lambda = \{k_1, \dots k_n\}$ by n binary parameters, only the i-th of which is true for data points where λ is set to k_i.

3.3 Choice of Regression Models

We considered a broad range of commonly used regression algorithms as candidates for our surrogate benchmarks. To keep the results comparable, all models were trained on data encoded as detailed in the previous section. If necessary for the algorithm, we also normalized the data to have zero mean and unit variance (by subtracting the mean and dividing by the standard deviation). If not stated otherwise for a model, we used the default configuration of its implementation.

Table 1 details the regression models and implementations we used. We evaluated three different tree-based models, because SMAC uses a random forest (RF), and because RFs have been shown to yield high-quality predictions of algorithm performance data [18]. As a specialist for low-dimensional hyperparameter spaces, we used SPEARMINT's Gaussian process (GP) implementation, which performs MCMC to marginalize over hyperparameters. Since SMAC performs particularly well on high-dimensional hyperparameter spaces and SPEARMINT on low-dimensional continuous problems [8], we expected their respective models to mirror that pattern. The remaining prominent model types we experimented with comprised k-nearest-neighbours (kNN), linear regression, least angle regression, ridge regression, SVM methods (all as implemented by scikit-learn [25]), and Bayesian neural networks (BNN) [24].

Table 2. Properties of our data sets. "Input dim." is the number of features of the training data; it is greater than the number of hyperparameters because categorical hyperparameters and the crossvalidation fold are one-hot-encoded. For each benchmark, before preprocessing the number of data points was $10 \times 4\times$ (#evals. per run).

	hyperparameter			Input	#evals.	#data
	#λ	cond.	cat. / cont.	dim.	per run	
Branin	2	-	- / 2	3	200	7402
Log. Reg. 5CV	4	-	- / 4	9	500	18521
HP-NNET convex	14	4	7 / 7	25	200	7750
HP-DBNET mrbi	36	27	19 / 17	82	200	7466

3.4 Construction and Use of Surrogate Benchmarks

To construct surrogates for a hyperparameter optimization benchmark X, we trained the previously mentioned models on the performance data gathered on benchmark X. The surrogate benchmark X'_M based on model M is identical to the original benchmark X, except that evaluations of the machine learning algorithm to be optimized in benchmark X are replaced by a performance prediction obtained from model M. In particular, the surrogate's configuration space (including all parameter types and domains) and function evaluation budget are identical to the original benchmark.

Importantly, the wall clock time to run an algorithm on X'_M is much lower than that required on X, since expensive evaluations of the machine learning algorithm underlying X are replaced by cheap model predictions. The model M is simply saved to disk and is queried when needed. We could implement each evaluation in X'_M as loading M from disk and then using it for prediction, but to avoid the repeated cost of loading M, we also allow for storing M in an independent process and communicate with it via a local socket.

To evaluate the performance of a surrogate benchmark scenario X'_M we ran the same optimization experiments as on X, using the same settings and seeds. In addition to evaluating the raw predictive performance of model M, we assessed the quality of surrogate benchmark X'_M by measuring the similarity of hyperparameter optimization performance on X and X'_M.

4 Experiments and Results

In this section, we experimentally evaluate the performance of our surrogates. We describe the data upon which our surrogates are based, evaluate the raw performance of our regression models on this data, and then evaluate the quality of the resulting surrogate benchmarks.

4.1 Experimental Setup

We collected data for four benchmarks from the hyperparameter optimization benchmark library, HPOLIB [8]. For each benchmark, we executed 10 runs of SMAC, SPEARMINT, TPE and random search (using the same Hyperopt implementation of random search as for TPE), yielding the data detailed in Table 2. The four benchmarks comprised two low-dimensional and two high-dimensional hyperparameter spaces.

The two low-dimensional benchmarks were the synthetic Branin test function and a logistic regression [29] on the MNIST dataset [23]. Both of these have been extensively used before to benchmark hyperparameter optimization methods. While the 2-dimensional Branin test function is trivial to evaluate and therefore does not require a surrogate, we nevertheless included it to study how closely a surrogate can approximate the function. The logistic regression is an actual hyperparameter optimization benchmark with 4 hyperparameters that includes a 5-fold cross-validation. That means for each configuration that the optimizers TPE, SPEARMINT and random search evaluated there were 5 data points that only differ in which fold they corresponded to. Since SMAC saves time by not evaluating all folds for configurations that appear worse than the optimum, it only evaluated a subset of folds for most of the configurations. The evaluation of a single cross-validation fold required roughly 1 minute on a single core of an Intel Xeon E5-2650 v2 CPU.

The high-dimensional benchmarks comprised a simple and a deep neural network, HP-NNET and HP-DBNET (both taken from [2]) to classify the MRBI and convex datasets, respectively [22]. Their dimensionalities are 14 and 36, respectively, and many categorical hyperparameters further increase the input dimension to the regression model. Evaluating a single HP-NNET configuration required roughly 12 minutes using 2 cores of an Intel Xeon E5-2650 v2 with OpenBlas. The HP-DBNET required a GPGPU to run efficiently; on a modern Geforce GTX780 GPU, it took roughly 15 minutes to evaluate a single configuration. In contrast, using the surrogate benchmark model we built, one configuration can be evaluated in less than a second on a standard CPU.

For some model types, training with all the data from Table 2 was computationally infeasible, and we had to subsample 2 000 data points (uniformly at random[3]) for training. This was the case for nuSVR, SVR, and the Bayesian neural network. For the GP model, we had to limit the dataset even further to 1 500 data points. On this reduced training set, the GP model required 255 minutes to train on the most expensive data set (HP-DBNET MRBI), and the Bayesian neural networks required 36 minutes; all other models required less than one minute for training.

We used HPOLIB to run the experiments for all optimizers with a single format, both for the original hyperparameter optimization benchmarks and for our surrogates. To make our results reproducible, we fixed the pseudo-random number seed in each function evaluation to 1. The version of the SPEARMINT package we used crashed for about 1% of all runs due to a numerical problem. In evaluations where we require entire trajectories, for these crashed SPEARMINT runs, we imputed the best function value found before the crash for all evaluations after the crash.

4.2 Evaluation of Raw Model Performance

We first studied the raw predictive performance of the models we considered on our preprocessed data.

4.2.1 Using all data

To evaluate the raw predictive performance of the models listed in Table 1, we used 5-fold cross-validation performance and computed the cross-validated root mean squared error (RMSE) and Spearman's rank correlation coefficient (CC) between model predictions and the true responses in the test fold. Here, the responses correspond to validation error rate in all benchmarks except for the Branin one (where they correspond to the value of the Branin function).

Table 3 presents these results, showing that the GP and the SVR approaches performed best on the smooth low-dimensional synthetic Branin test function, but that RF-based models are better for predicting the performance of actual machine learning algorithms. This

[3] For a given dataset and fold, all models based on the same number of data points used the same subsampled data set. We note that model performance sometimes was quite noisy with respect to the pseudorandom number seed for this subsampling step and we thus used a fixed seed.

Table 3. Average RMSE and CC for a 5-fold cross validation for different regression models. For each entry, bold face indicates the best performance on this dataset, and underlined values are not statistically significantly different from the best according to a paired t-test (with $p = 0.05$). Models marked with an * ($^+$) are trained on only a subset of 2000 (1500) data points per fold.

Model	Branin RMSE	Branin CC	Log.Reg. 5CV RMSE	Log.Reg. 5CV CC	HP-NNET convex RMSE	HP-NNET convex CC	HP-DBNET mrbi RMSE	HP-DBNET mrbi CC
RF	1.86	1.00	**0.03**	**0.98**	0.04	0.94	**0.06**	**0.90**
Gr.Boost	7.10	0.96	0.07	0.94	0.05	0.89	0.06	0.86
Ex.Trees	1.10	1.00	0.04	0.98	**0.03**	**0.95**	0.06	0.90
GP $^+$	0.03	**1.0**	0.13	0.88	0.04	0.92	0.10	0.78
SVR *	0.06	1.0	0.13	0.87	0.06	0.82	0.08	0.80
nuSVR *	**0.02**	1.0	0.18	0.84	0.06	0.85	0.08	0.82
BNN *	6.84	0.91	0.10	0.91	0.05	0.84	0.10	0.72
kNN	1.78	1.00	0.14	0.88	0.06	0.85	0.08	0.78
Lin.Reg.	45.33	0.28	0.23	0.78	0.08	0.60	0.1	0.70
LeastAngleReg.	45.33	0.28	0.23	0.78	0.08	0.60	0.1	0.7
Ridge Reg.	46.01	0.30	0.26	0.77	0.09	0.61	0.10	0.67

Table 4. Average RMSE and CC of 5 regression algorithms in the leave-one-optimizer-out setting. Bold face indicates the best value across all regression models on this dataset.

Model	Branin RMSE	Branin CC	Log.Reg. 5CV RMSE	Log.Reg. 5CV CC	HP-NNET convex RMSE	HP-NNET convex CC	HP-DBNET mrbi RMSE	HP-DBNET mrbi CC
RF	2.04	0.95	**0.10**	**0.93**	**0.04**	0.82	**0.07**	**0.85**
Gr.Boost	6.96	0.85	0.12	0.84	0.05	0.81	**0.07**	0.83
GP	0.12	**0.99**	0.16	0.76	0.05	**0.84**	0.10	0.64
nuSVR	**0.03**	0.98	0.16	0.74	0.10	0.61	0.10	0.73
kNN	2.08	0.96	0.19	0.72	0.07	0.73	0.09	0.67

strong performance was to be expected for the higher-dimensional hyperparameter space of the neural networks, since RFs perform automatic feature selection.[4] The logistic regression example is rather low-dimensional, but the categorical cross-validation fold is likely harder to model for GPs than for RFs.[5] Extra Trees predicted the performance of the actual machine learning algorithms nearly as good as the RF, Gradient boost was slightly worse. Bayesian neural networks, k-nearest-neighbours and our linear regression models could not achieve comparable performance. Based on these results, we decided to focus the remainder of our study on a diverse subset of models: two tree-based approaches (RFs and gradient boost), Gaussian processes, nuSVR, and, as an example of a popular, yet poorly-performing model, k-nearest-neighbours. We paid special attention to RFs and Gaussian processes, since these have been used most prominently in Bayesian hyperparameter optimization methods.

4.2.2 *Leave one optimizer out*

In practice, we will want to use our surrogate models to predict the performance of a machine learning algorithm with hyperparameter configurations selected by some new optimization method. The configurations it evaluates might be quite different from those considered by the optimizers whose data we trained on. Next to the standard cross-validation setting from above, we therefore evaluated our models in the leave-one-optimizer-out setting, which means that the regression model learns from data drawn from all but one optimizer, and its performance is measured on the held out data.

Table 4 reports RMSE and CC analogous to those of Table 3, but for the leave-one-optimizer-out setting. This setting is more difficult and, consequently, the results were slightly worse, but the best-performing models stayed the same: nuSVR and GP for low dimensional and RFs for higher dimensional datasets.

[4] We note that RFs could also handle the categorical hyperparameters in these benchmarks natively. We used the one-hot encoding for comparability with other methods. It is remarkable that even with this encoding, they outperformed all other methods.

[5] As we will see later (Figure 2), the RF-based optimizer SMAC also performed better on this benchmark than the GP-based optimizer SPEARMINT.

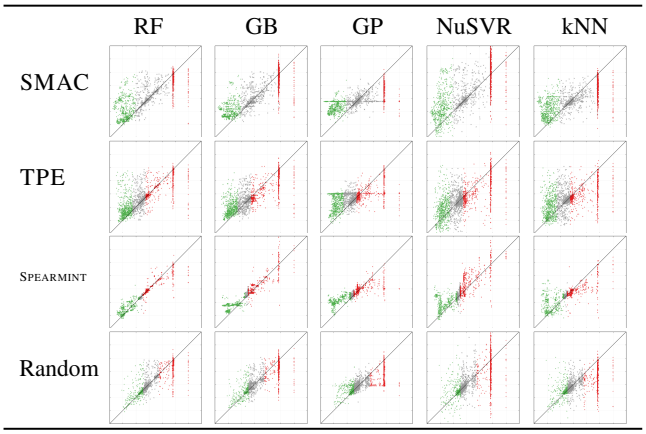


Figure 1. True performance (x-axis) vs. regression model predictions (y-axis) for the **HP-DBNET mrbi** dataset. All plots have the same axes, showing error rates ranging from 0.4 to 1.1. Each marker represents the performance of one configuration; green and red crosses indicate 1/3 best and worst true performance, respectively. Configurations on the diagonal are predicted perfectly, error predictions above the diagonal are too high, and predictions for configurations below the diagonal are better than the configuration's actual performance. The first column shows which data was left out for training and used for testing.

Figure 1 studies the predictive performance in more detail for the HP-DBNET mrbi benchmark, demonstrating that tree-based models also performed best in a qualitative sense. The figure also shows that the models tended to make the largest mistakes for the worst configurations; especially the non-tree-based models predicted some of these to be better than some of the best configurations. The same patterns also held for the logistic regression 5CV and the HP-NNET convex data (not shown). The models also completely failed to identify neural network configurations which did not converge within the time limit and therefore received an error rate of 1.0. Interestingly, the GP failed almost entirely on the high-dimensional HP-DBNET MRBI benchmark in two cases, predicting all data points around the data mean.

4.3 Evaluation of Surrogate Benchmarks

We now study the performance of the surrogate benchmarks X'_M obtained for random forest (RF) and Gaussian process (GP) models M. We assess the quality of X'_M by comparing the performance of various hyperparameter optimizers on X'_M and the real benchmark X.

4.3.1 *Using all data*

We first analyzed the performance of surrogate benchmarks based on models trained on the entire data we have available. We note that in this first experiment, a surrogate that perfectly remembers the training data would achieve perfect performance, because we used the same hyperparameter optimizers for evaluation as we did to gather the training data. However, after the first imperfect prediction, the trajectories of the optimizers will diverge. Thus, since none of our models is perfect on training data, this initial experiment serves as an

evaluation of surrogate benchmarks based on training data gathered through the same mechanism as at test time.

We performed experiments for our three actual hyperparameter optimization benchmarks, logistic regression, a simple and a deep neural network. For each of them, we repeated the 10 runs for TPE, SMAC and SPEARMINT we previously conducted on the real benchmarks, but now used surrogate benchmarks based on RFs and GPs, respectively.

Figure 2 shows that the surrogate benchmarks based on Gaussian process models differed substantially from the true benchmarks. The figures show the best function values found by the various optimizers over time. Visually comparing the first column (real benchmark) to the third (surrogate benchmark based on GP model), the most obvious difference is that the surrogate benchmark fails completely on HP-DBNET MRBI: since the GP model is unable to properly fit the high-dimensional data (predicting all configurations to perform roughly equally, around the data mean) all optimizers basically stay at the same performance level (the data mean). Note in the plot for the true benchmark that the GP-based optimizer SPEARMINT also performed very poorly on this benchmark.

In the other two cases (logistic regression 5CV and HP-NNET convex), the performance of the optimizers appears visually similar to the true benchmark at first glance. However, for the logistic regression 5CV the GP model predicts some parts of the hyperparameter space to be better than the actual best part of the space, leading to the final optimization results on the surrogate benchmark to appear better than optimization results on the true benchmark. Another difference is a zig-zag pattern in the trajectories for logistic regression surrogates: these are also (mildly) present in the real benchmark (mild enough to only be detectable when zooming into the figure) and are due to the slightly different performance in the 5 folds of cross validation; the impact of the folds is very small, but the GP model predicts it to be large, causing the zig-zag. Interestingly, for the GP model trained on the HP-NNET convex dataset, regions with "better" performance appear hard to find: only SMAC and TPE identified them, causing a larger gap between SPEARMINT and SMAC/TPE than on the real benchmark.

Conversely, the RF surrogates yielded results much closer to those obtained on the real benchmark. Visually, the first column (true benchmark) and second column appear very similar, indicating that the RF captured the overall pattern well. There are some differences in the details. For example, on HP-NNET convex, the surrogate does not capture that TPE finds very good configurations before SMAC and yields the overall best performance. Nevertheless, overall, our results for the RF surrogates qualitatively resemble those for the true benchmarks, and for the logistic regression example, the correspondence is almost perfect.

4.3.2 Leave one optimizer out

Next, we studied the use of a surrogate benchmark to evaluate a new optimizer. For each optimizer o and each of the three hyperparameter optimization benchmarks X, we trained RF and GP models M on the respective leave-one-optimizer-out training data discussed in Section 4.2.2 and compared the performance of optimizer o on X and X'_M. Figure 3 reports the results of this experiment, showing that surrogate benchmarks based on RF models qualitatively resembled the real benchmarks.

The results for the logistic regression 5CV benchmark (top row of Figure 3) show that surrogate benchmarks based on RF models mirrored the performance of each optimizer o on the real benchmark well, even when the training data did not include data gathered with optimizer o. In contrast, surrogates based on Gaussian process models performed poorly: the Gaussian process again underestimated the error, predicting better performance in some regions than possible on the real benchmark.[6] Again, these regions with "better" performance appear hard to find: only SMAC and SPEARMINT found them, causing their performances on the GP-based surrogate benchmark to differ substantially from their performance on the true benchmark.

Results for HP-NNET convex were also better for the surrogate benchmark based on RFs (especially for SPEARMINT), but not as much better as for the logistic regression 5CV case. As was already the case when the surrogate was based on all training data, the RF-based surrogate benchmarks only approximately captured the strong performance TPE showed on the real benchmark.

Results on HP-DBNET MRBI show a fairly close correspondence between the real benchmark and the RF-based surrogate benchmarks. In contrast, the GP-based surrogate was dismal, once again due to the GP's near-constant predictions (close to the data mean).

After this qualitative evaluation of the surrogate benchmarks, Table 5 offers a quantitative evaluation. We judge the quality of a surrogate benchmark X'_M by how closely it resembles the real benchmark X it was derived from, in terms of the absolute error between the best found values for our four optimizers (SMAC, TPE, SPEARMINT, and random search) after evaluating i configurations. For logistic regression 5CV, in line with our qualitative results we obtained a very small error for the RF-based surrogate. The GP-based surrogate underestimated the achievable error rates, resulting in larger differences between performances on the true and the surrogate runs. After 50 evaluations the GP-based surrogate trained on all data yielded a quite high error because it underestimated the performance for configurations selected by the optimizer SMAC. Training the GP-surrogate on the leave-one-optimizer-out dataset causes worse performance for the optimizer SPEARMINT and too much variation for SMAC resulting in a higher error as well. This misprediction decreases with more evaluated configurations.

The results for the HP-NNET convex look quite similar, with a somewhat smaller difference between RF-based and GP-based surrogates. Indeed, SMAC and TPE behaved similarly on both RF-based and GP-based surrogates as on the real benchmark; only SPEARMINT behaved very differently on the GP-based surrogate, causing an overall higher error than for the RF-based surrogates.

On the high dimensional HP-NNET mrbi the surrogates performed differently. Whereas the RF-based surrogate could still reproduce similar optimizer behavior as on the real benchmark, the GP completely failed to do so. Remarkably, overall quantitative performance was similar for surrogate benchmarks trained on all data and those trained on leave-one-optimizer-out datasets.

Overall, these results confirmed our expectation from previous findings in Section 3.3 and the raw regression model performance results in Table 3: good regression models facilitate good surrogate benchmarks. In our case, RFs performed best for both tasks. We note that using the surrogate benchmarks reduced the time requirements substantially; for example, evaluating a surrogate 100 times instead of the HP-NNET convex or HP-DBNET MRBI took less than 1 minute on a single CPU, compared to roughly 10 hours on two CPUs (HP-NNET convex) and over a day on a modern GPU (HP-DBNET MRBI).[7]

[6] We noticed similar behavior for the nuSVR, which even returned negative values for configurations and caused the optimizer to search completely different areas of the configuration space (data not shown here).

[7] Of course, the overhead due to the used hyperparameter optimizer comes on top of this; e.g., SPEARMINT's overhead for a run with 200 evaluations was roughly one hour, whereas SMAC's overhead was less than one minute.

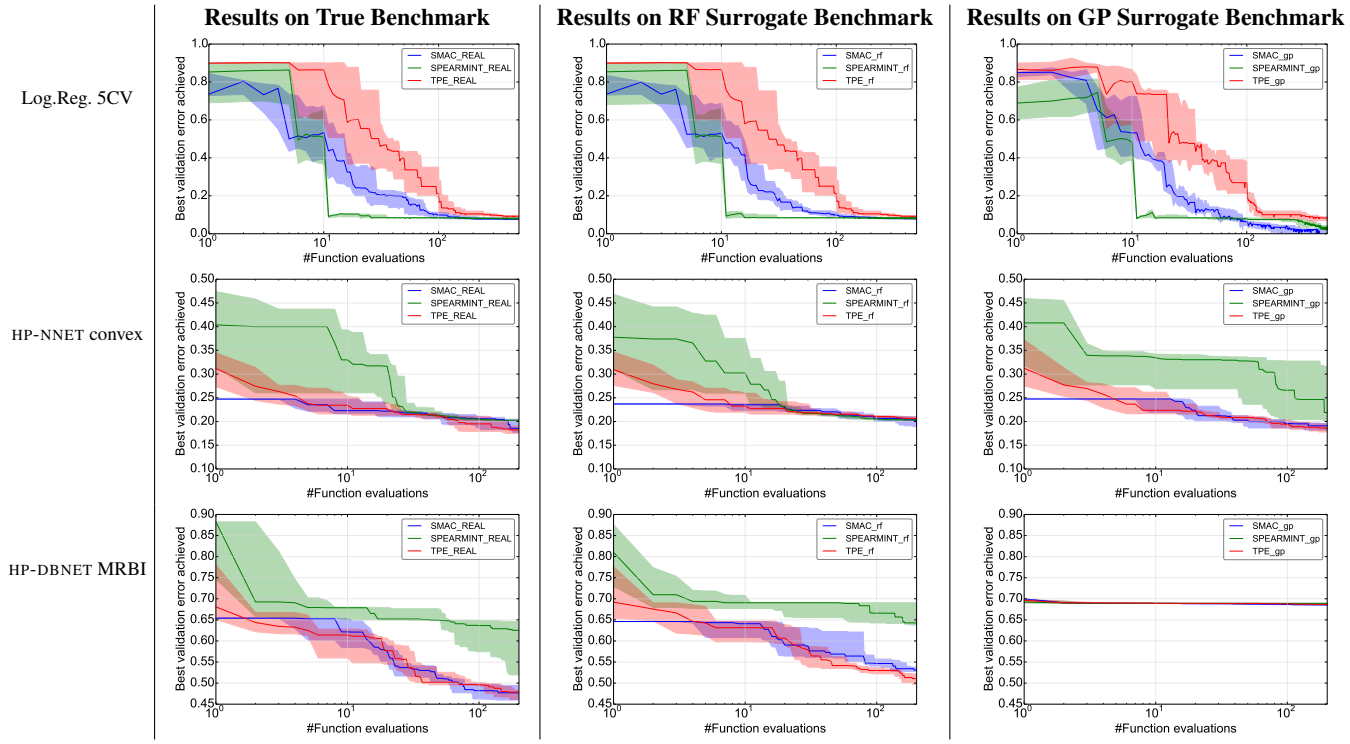


Figure 2. Median and quartile of best performance over time on the real benchmark (left column) and on surrogates (middle: based on RF models; right: based on GP models). Both types of surrogate benchmarks were trained on all available data. For logistic regression 5CV each fold is plotted as a separate function evaluation.

Table 5. Quantitative evaluation of surrogate benchmarks at three different time steps each. We show the mean difference between the best found values for corresponding runs (having the same seed) of the four optimizers (SMAC, TPE, SPEARMINT, and random search) after i function evaluations on the real and surrogate benchmark. For each experiment and optimizer we conducted 10 runs and report the mean error averaged over $4 \times 10 = 40$ comparisons. We evaluated RF-based and GP-based surrogates. For each problem we measured the error for surrogates trained on *all* and the leave-one-optimizer-out (*leave-ooo*) data; e.g., the TPE trajectories are from optimizing on a surrogate that is trained on all training data except that gathered using TPE. Bold face indicates the best performance for this dataset and i function evaluations. Results are underlined when the one-sigma confidence intervals of the best and this result overlaps.

#Function evaluations	50		200		500	
Surrogate	RF	GP	RF	GP	RF	GP
Log.Reg. SCV all	**0.02**	0.06	**0.00**	0.04	**0.00**	0.05
Log.Reg. SCV leave-ooo	**0.02**	0.07	**0.01**	0.03	**0.00**	0.03

#Function evaluations	50		100		200	
Surrogate	RF	GP	RF	GP	RF	GP
HP-NNET convex all	**0.01**	0.03	**0.01**	0.03	**0.01**	0.02
HP-NNET convex leave-ooo	**0.02**	0.03	**0.02**	0.04	**0.02**	0.03
HP-DBNET MRBI all	**0.05**	0.13	**0.05**	0.16	**0.05**	0.17
HP-DBNET MRBI leave-ooo	**0.04**	0.13	**0.04**	0.16	**0.05**	0.17

5 Conclusion and Future Work

To tackle the high computational cost and overhead of performing hyperparameter optimization benchmarking, we proposed surrogate benchmarks that behave similarly to the actual benchmarks they are derived from, but are far cheaper and simpler to use. The key idea is to collect (configuration, performance) pairs from the actual benchmark and to learn a regression model that can predict the performance of a new configuration and therefore stand in for the expensive-to-evaluate algorithm. These surrogates reduce the algorithm overhead to a minimum, which allows extensive runs and analyses of new hyperparameter optimization techniques. We empirically demonstrated that we can obtain surrogate benchmarks that closely resemble the real benchmarks they were derived from.

In future work, we intend to study the use of surrogates for general algorithm configuration. In particular, we plan to support optimization across a set of problem instances, each of which can be described by a fixed-length vector of characteristics, and to assess the resulting surrogates for several problems that algorithm configuration has tackled successfully, such as propositional satisfiability [14], mixed integer programming [15], and AI planning [9]. Finally, good surrogate benchmarks should enable us to explore the configuration options of the optimizers themselves, and we plan to use surrogate benchmarks to enable efficient meta-optimization of the hyperparameter optimization and algorithm configuration methods themselves.

REFERENCES

[1] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, 'Collaborative hyperparameter tuning', in *Proc. of ICML'13*, (2013).

[2] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, 'Algorithms for hyper-parameter optimization', in *Proc. of NIPS'11*, (2011).

[3] J. Bergstra and Y. Bengio, 'Random search for hyper-parameter optimization', *JMLR*, **13**, 281–305, (2012).

[4] J. Bergstra, B. Komer, C. Eliasmith, and D. Warde-Farley, 'Preliminary evaluation of hyperopt algorithms on HPOLib', in *ICML workshop on AutoML*, (2014).

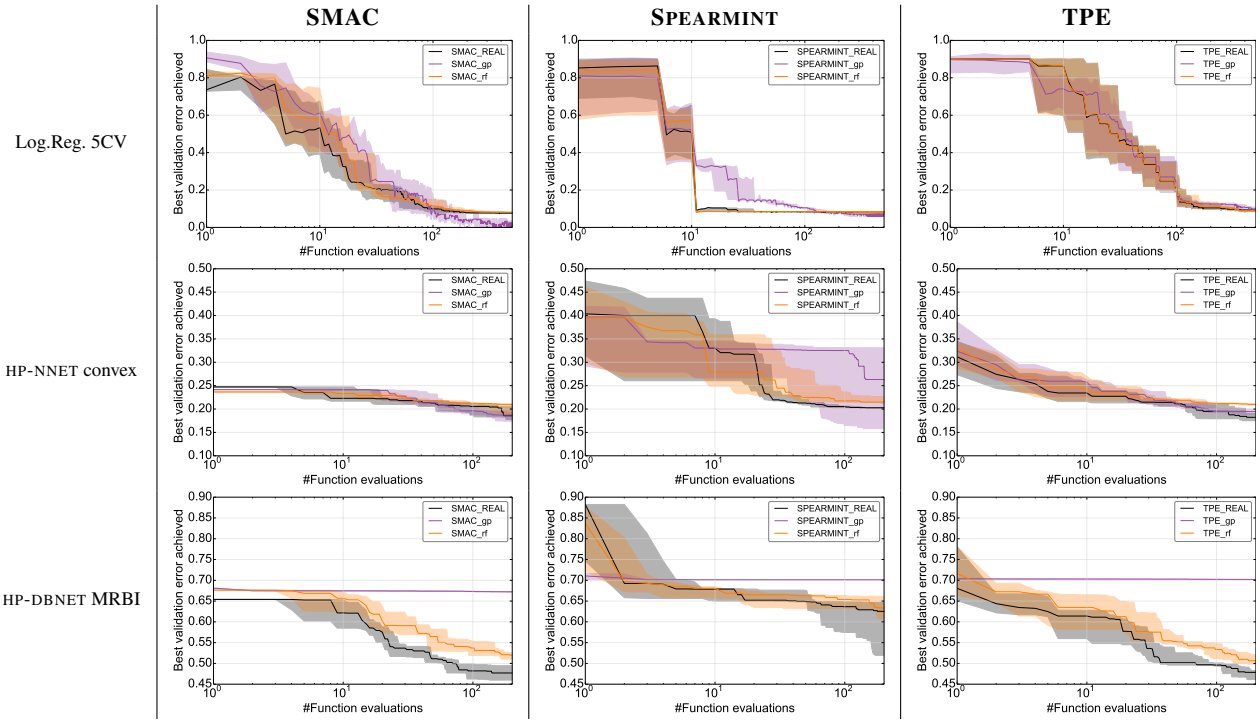


Figure 3. Median and quartile of optimization trajectories for surrogates trained in the leave-one-optimizer-out setting. Black trajectories correspond to true benchmarks, coloured trajectories to optimization runs on a surrogate. The first row names the optimizer used to obtain the trajectories; their data was left out for training the regression models.

[5] J. Bergstra, D. Yamins, and D. D. Cox, 'Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures', in *Proc. of ICML'13*, (2013).

[6] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta, *Metalearning: Applications to Data Mining*, Springer, 2008.

[7] E. Brochu, V. M. Cora, and N. de Freitas, 'A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning', *CoRR*, **abs/1012.2599**, (2010).

[8] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. H. Hoos, and K. Leyton-Brown, 'Towards an empirical foundation for assessing bayesian optimization of hyperparameters', in *NIPS workshop on Bayesian Optimization*, (2013).

[9] C. Fawcett, M. Helmert, H. H. Hoos, E. Karpas, G. Röger, and J. Seipp, 'FD-Autotune: Domain-specific configuration using fast-downward', in *Proc. of ICAPS-PAL*, (2011).

[10] D. Gorissen, I. Couckuyt, P. Demeester, T. Dhaene, and K. Crombecq, 'A surrogate modeling and adaptive sampling toolbox for computer based design', *JMLR*, **11**, 2051–2055, (2010).

[11] S. B. Guerra, R. B. C. Prudêncio, and T. B. Ludermir, 'Predicting the performance of learning algorithms using support vector machines as meta-regressors', in *Proc. of ICANN'08*, volume 5163, pp. 523–532, (2008).

[12] N. Hansen, A. Auger, S. Finck, R. Ros, et al. Real-parameter black-box optimization benchmarking 2010: Experimental setup, 2010.

[13] M. D. Hoffman, D. M. Blei, and F. R. Bach, 'Online learning for latent dirichlet allocation.', in *Proc. of NIPS'10*, (2010).

[14] F. Hutter, D. Babić, H.H. Hoos, and A.J. Hu, 'Boosting Verification by Automatic Tuning of Decision Procedures', in *Proc. of FMCAD'07*, pp. 27–34, Washington, DC, USA, (2007). IEEE Computer Society.

[15] F. Hutter, H. H. Hoos, and K. Leyton-Brown, 'Automated configuration of mixed integer programming solvers', in *Proc. of CPAIOR-10*, pp. 186–202, (2010).

[16] F. Hutter, H. H. Hoos, and K. Leyton-Brown, 'Sequential model-based optimization for general algorithm configuration', in *Proc. of LION-5*, (2011).

[17] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, 'ParamILS: an automatic algorithm configuration framework', *JAIR*, **36**(1), 267–306, (2009).

[18] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, 'Algorithm runtime prediction: Methods and evaluation', *JAIR*, **206**(0), 79 – 111, (2014).

[19] B. Komer, J. Bergstra, and C. Eliasmith, 'Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn', in *ICML workshop on AutoML*, (2014).

[20] A. Krizhevsky, 'Learning multiple layers of features from tiny images', Technical report, University of Toronto, (2009).

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, 'Imagenet classification with deep convolutional neural networks', in *Proc. of NIPS'12*, pp. 1097–1105, (2012).

[22] 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 *Proc. of ICML'07*, (2007).

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, 'Gradient-based learning applied to document recognition', *Proc. of the IEEE*, **86**(11), 2278–2324, (1998).

[24] R. M. Neal, *Bayesian learning for neural networks*, Ph.D. dissertation, University of Toronto, 1995.

[25] 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', *JMLR*, **12**, 2825–2830, (2011).

[26] M. Reif, F. Shafait, M. Goldstein, T. Breuel, and A. Dengel, 'Automatic classifier selection for non-experts', *PAA*, **17**(1), 83–96, (2014).

[27] J. Sacks, W. J. Welch, T. J. Welch, and H. P. Wynn, 'Design and analysis of computer experiments', *Statistical Science*, **4**(4), 409–423, (November 1989).

[28] T. J. Santner, B. J. Williams, and W. I. Notz, *The design and analysis of computer experiments*, Springer, 2003.

[29] J. Snoek, H. Larochelle, and R.P. Adams, 'Practical Bayesian optimization of machine learning algorithms', in *Proc. of NIPS'12*, (2012).

[30] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, 'Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms', in *Proc. of KDD'13*, (2013).

[31] C. N. J. Yu and T. Joachims, 'Learning structural svms with latent variables', in *Proc. of ICML'09*, pp. 1169–1176, (2009).

A Framework To Decompose And Develop Metafeatures

Fábio Pinto[1] and **Carlos Soares**[2] and **João Mendes-Moreira**[3]

Abstract.

This paper proposes a framework to decompose and develop metafeatures for Metalearning (MtL) problems. Several metafeatures (also known as data characteristics) are proposed in the literature for a wide range of problems. Since MtL applicability is very general but problem dependent, researchers focus on generating specific and yet informative metafeatures for each problem. This process is carried without any sort of conceptual framework. We believe that such framework would open new horizons on the development of metafeatures and also aid the process of understanding the metafeatures already proposed in the state-of-the-art. We propose a framework with the aim of fill that gap and we show its applicability in a scenario of algorithm recommendation for regression problems.

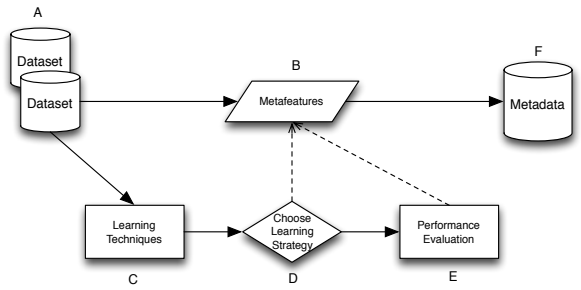


Figure 1. Metalearning: knowledge acquisition. Adapted from [2]

1 Introduction

Researchers have been using MtL to overcome innumerous challenges faced by several data mining practitioners, such as algorithm selection [3][23], time series forecasting [9], data streams [19][20][5], parameter tuning [22] or understanding of learning behavior [6].

As the study of principled methods that exploit metaknowledge to obtain efficient models and solutions by adapting machine learning and data mining processes [2], MtL is used to extrapolate knowledge gained in previous experiments to better manage new problems. That knowledge is stored as metadata, particularly, metafeatures and metatarget, as outlined in Figure 1. The metafeatures (extracted from *A* to *B* and stored in *F*) consist in data characteristics that describe the correlation between the learning algorithms and the data under analysis, *i.e.*, correlation between numeric attributes of a dataset. The metatarget (extracted through *C-D-E* and stored in *F*) represents the meta-variable that one wishes to understand or predict, *i.e.*, the algorithm with best performance for a given dataset.

Independently of the problem at hands, the main issue in MtL concerns defining the metafeatures. If the user is able to generate informative metafeatures, it is very likely that his application of MtL is going to be successful. The state-of-the-art shows that there is three types of metafeatures: 1) simple, statistical and information-theoretic. In this group we can found the *number of examples* of the dataset, *correlation between numeric attributes* or *class entropy*, to name a few. Application of these kind of metafeatures provides not only informative metafeatures but also interpretable knowledge about the problems [3] 2) model-based ones [13]. These capture some characteristic of a model generated by applying a learning algorithm to a dataset, *i.e.*, the *number of leaf nodes of decision tree*. Finally, a metafeature can also be a 3) *landmarker* [14]. These are generated by making a quick performance estimate of a learning algorithm in a particular dataset.

Although the state-of-the-art proposes several metafeatures of all types for a wide range of problems, we state that the literature lacks an unifying framework to categorize and develop new metafeatures. Such framework could help MtL users by systematizing the process of generating new metafeatures. Furthermore, the framework could be very useful to compare different metafeatures and assess if there is no overlap of the information that they capture. In this paper, we propose a framework with that purpose and we use it in the analysis of the metafeatures used in several MtL applications. We also show its applicability to generate metafeatures in a scenario of algorithm recommendation for regression problems.

The paper is organized as follows. In Section 2 we present a brief overview of MtL applications and respective metafeatures. Section 3 details the proposed framework to decompose and develop metafeatures. In Section 4 we use the framework to decompose and understand how our framework would characterize metafeatures already proposed in the literature. Section 5 exemplifies how the framework could be used to develop new metafeatures in a scenario of algorithm recommendation for regression problems. Finally, we conclude the paper with some final remarks and future work.

2 Metalearning

MtL emerges as the most promising solution from machine learning researchers to the need for an intelligent assistant for data analysis [21]. Since the majority of data mining processes include several non-trivial decisions, it would be useful to have a system that could guide the users to analyze their data.

[1] LIAAD-INESC TEC, Universidade do Porto, Portugal, e-mail: fhpinto@inesctec.pt
[2] CESE-INESC TEC, Universidade do Porto, Portugal, e-mail: csoares@fe.up.pt
[3] LIAAD-INESC TEC, Universidade do Porto, Portugal, e-mail: jmoreira@fe.up.pt

The main focus of MtL research has been the problem of algorithm recommendation. Several works proposed systems in which data characteristics were related with the performance of learning algorithms in different datasets. Brazdil et al. [3] system provides recommendations in the form of rankings of learning algorithms. Besides the MtL system, they also proposed an evaluation methodology for ranking problems that is useful for the problem of algorithm ranking. Sun and Pfahringer [23] extended the work of Brazdil et al. with two main contributions: the pairwise meta-rules, generated by comparing the performance of individual base learners in a one-against-one manner; and a new meta-learner for ranking algorithms.

Another problem addressed by MtL has been the selection of the best method for time series forecasting. The first attempt was carried by Prudêncio and Ludermir [16] with two different systems: one that was able to select among two models to forecast stationary time series and another to rank three models used to forecast time series. Results of both systems were satisfactory. Wang et al. [26] addressed the same problem but with a descriptive MtL approach. Their goal was to extract useful rules with metaknowledge that could aid the users in selecting the best forecasting method for a given time series and develop a strategy to combine the forecasts. Lemke and Bogdan [9] published a similar but with more emphasis on improving forecasts through model selection and combination.

MtL has also been used to tune parameters of learning algorithms. Soares et al. [22] proposed a method that by using mainly simple, statistical and information-theoretic metafeatures was able to predict successfully the width of the Gaussian kernel in Support Vector Regression. Results show that the methodology can select settings with low error while providing significant savings in time. Ali and Miles [1] published a MtL method to automatically select the kernel of a Support Vector Machine in a classification context, reporting results with high accuracy ratings. Reif et al. [17] used a MtL system to provide good starting points to a genetic algorithm that optimizes the parameters of a Support Vector Machine and a Random Forests classifier. Results state the effectiveness of the approach.

Data stream mining can also benefit from MtL, especially in a context where the distribution underlying the observations may change over time. Gama and Kosina [5] proposed a metalearning framework that is able to detect recurrence of contexts and use previously learned models. Their approach differs from the typical MtL approach in the sense that uses the base-level features to train the metamodel. On the other hand, Rossi et al. [19] reported a system for periodic algorithm selection that uses data characteristics to induce the metamodel (all the metafeatures are of the simple, statistical and information-theoretic type).

Another interesting application of MtL is to use it as a methodology to investigate the reasons behind the success or failure of a learning algorithm [25]. In this approach, instead of the typical predictive methodology, MtL is used to study the relation between the generated metafeatures and a metatarget that represents the base-level phenomenon that one wishes to understand. Kalousis et al. [6] published a paper on this matter. They adress the problem of discovering similarities among classification algorithms and among datasets using simple, statistical and information-theoretic metafeatures.

All the MtL applications that we mentioned previously use differents sets of metafeatures. It is mandatory to adapt the set of metafeatures to the problem domain. However, as stated previously, we believe that would be useful to decompose all these metafeatures into a common framework. Furthermore, such framework must also help the MtL user in the development of new metafeatures.

3 Metafeatures Development Framework

In this section, we propose a framework in order to allow a more systematized and standardized development of metafeatures for MtL problems. This framework splits the conception of a metafeature into four components: object, role, data domain and aggregation function. Within each component, the metafeature can be generated by using different subcomponents. Figure 2 illustrates the framework.

The object component concerns which information is going to be used to compute the metafeature. It can be an instance(s), dataset(s), model(s) or a prediction(s). The metafeature can extract information from one subcomponent (*i.e.*, class entropy of a dataset), several units of a subcomponent (*i.e.*, mean class entropy of a subset of datasets) and for some problems it might be useful to select multiple subcomponents (*i.e.*, for dynamic selection of models, one could relate instances with models [11]).

The role component details the function of the object component that is going to be used to generate the metafeature. The focus can be in the target variable, predicted or observed, in a feature or in the structure of the object component (*i.e.*, decision tree model or the representation of a dataset into a graph). Several elements can be selected, *i.e.*, the metafeature can relate the target variable with one or more features.

The third component defines the data domain of the metafeature and it is decomposed into four subcomponents: quantitative, qualitative, mixed or complex. This component is highly dependent of the previous ones and influences the metric used for computation (*i.e.*, if the data domain is qualitative, the user can not use correlation to capture the information). A metric can be quantitative (if the object component is numerical), qualitative (if the object component is categorical), mixed (if the object component has both numerical and categorical data) or complex (in special situations in which the object is a graph or a model).

Finally, the aggregation function component. Typically, this is accomplished by some descriptive statistic, i.e., mean, standard deviation, mode, etc. However, for some MtL problems it might be useful to not aggregate the information computed with the metric component. This is particularly frequent in MtL applications such as time series or data streams [20] were the data has the same morphology. For example, instead of computing the mean of the correlation between pairs of numerical attributes, one could use the correlation between all pairs of numerical attributes.

4 Decomposing Metafeatures

We used the framework to decompose metafeatures proposed in several applications to assess its applicability and consistence. We show examples from the three types of state-of-the-art metafeatures: simple, statistical and information-theoretic; model-based and landmarkers.

Figure 3 illustrates the decomposition of six simple, statistical and information-theoretic metafeatures. The first three (*number of examples*, *class entropy* and *absolute mean correlation between numeric attributes*) are common metafeatures used in several published papers [3][6][22]. The framework allows to detail the computation of the metafeature. Furthermore, it allows to compare two or more metafeatures. For example, the *absolute mean correlation between numeric attributes* is very similar to *correlation between numeric attributes* (used in data streams applications [20]) except for the aggregation function. In this case, the application domain makes it feasible and potentially more informative to not aggregate the correlation values.

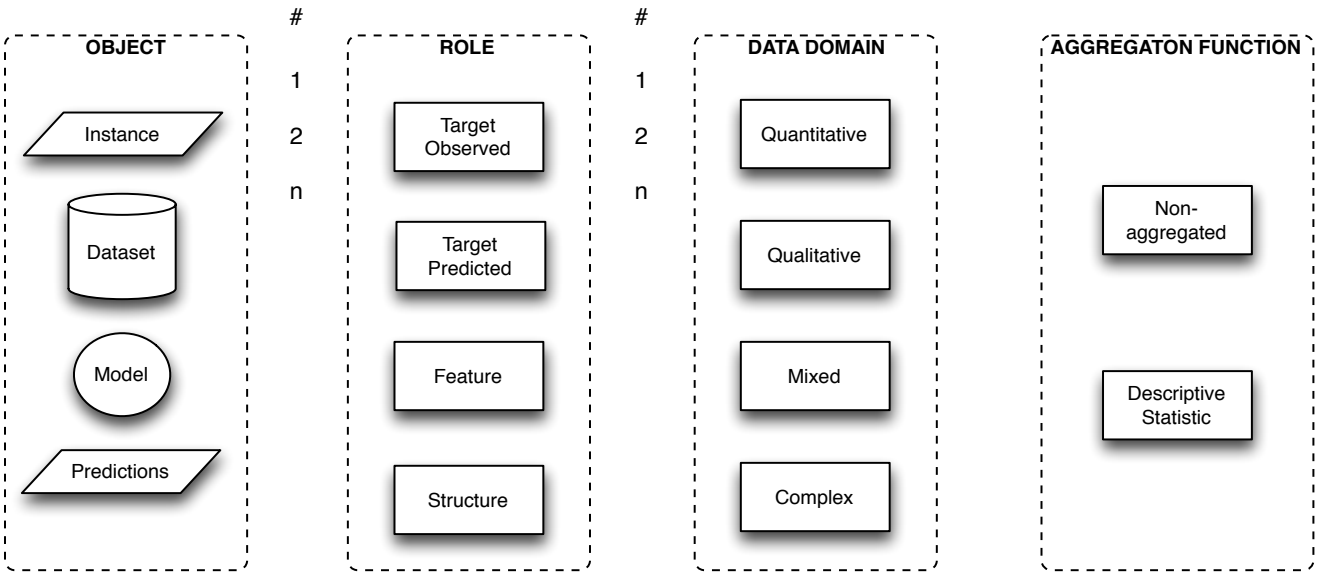


Figure 2. Metafeatures Development Framework.

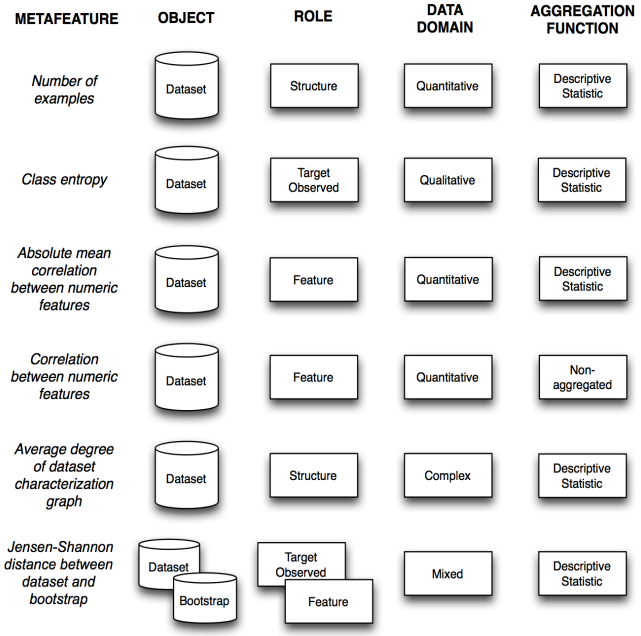


Figure 3. Simple, statistical and information-theoretic metafeatures decomposed using our framework.

Still regarding Figure 3, the decomposition of the two last metafeatures shows that is possible to use the framework for more complex data characteristics. Morais and Prati [12] published a paper in which they use measures from complex network theory to characterize a dataset. Their approach consists in transforming the dataset into a graph by means of similarity between instances. Then, they compute typical measures such as *number of nodes* or *average degree*. Another example would be the Jensen-Shannon distance between dataset and bootstrap [15]. In this example, the authors used the Jensen-Shannon distance to measure the differences caused by the bootstrapping process in the distribution of the variables (features and target).

In Figure 4, we show an example of a model-based metafeature decomposed using our framework. For computing the *number of nodes of a decision tree*, the object component is the model, with particular focus on its structure (as role component). Peng et al.[13] published a paper in which several model-based metafeatures are proposed (for decision trees models).

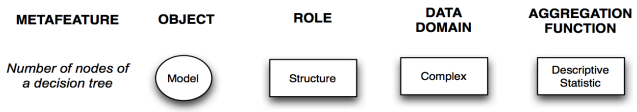


Figure 4. Model-based metafeatures decomposed using our framework.

Finally, in Figure 5, we show the framework applied to landmarkers. The first example, the *decision stump landmarker* [4], uses as object a set of predictions, both the predicted and the observed. Assuming a 0-1 loss function for classification problems, the data domain of a *decision stump landmarker* is always quantitative. Last but not least, the aggregation function in this case is a descriptive statistic, usually a mean. The second example concerns the metafeatures used in the meta decision trees proposed by Todorovski and Džeroski [24]. The authors used the class probabilities of the base-level classifiers as metafeature, particularly, the *highest class probability* of a classifier for a single instance.

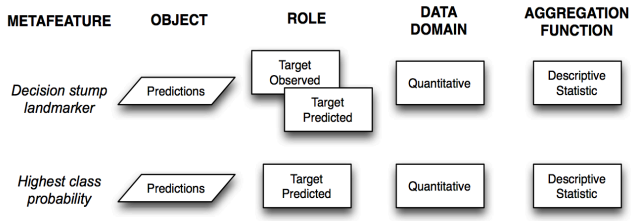


Figure 5. Landmarkers metafeatures decomposed using our framework.

5 Developing Metafeatures

In this Section we present a case study of the proposed framework with a metric widely used in MtL problems [2]: correlation between numeric variables. We show that it is possible to generate new metafeatures by combining elements of different components of the framework. Furthermore, using such framework allows a systematic reasoning in the process of developing metafeatures for a given problem. It becomes easier to detect gaps of non measured information in a set of metafeatures, if it is available a theoretical framework that can guide the user by pointing new research directions.

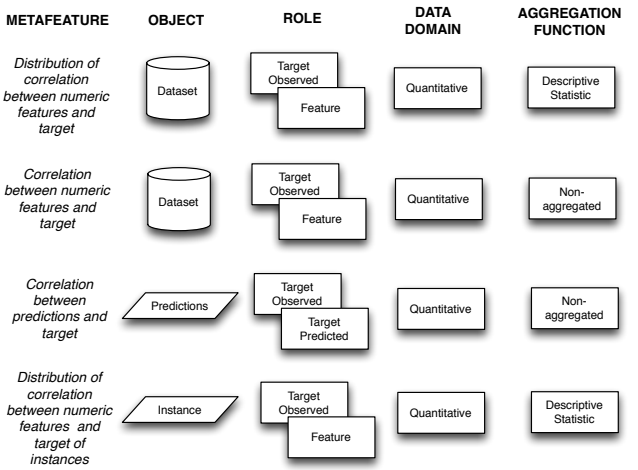


Figure 6. Examples of correlation metafeatures developed using the proposed framework.

As mentioned previously, we use correlation between numeric variables as example in the context of a MtL application for regression algorithm selection [7]. This a problem addressed in a relatively small number of papers in comparison with the classification scenario.

Figure 6 shows an illustration of four metafeatures that use correlation between numeric variables. The first metafeature, *distribution of correlation between numeric features and target*, although present in the literature [2], differs from *absolute mean correlation between numeric features* presented in Figure 3 by adding the element *target* to the role component. This simple change transforms completely the nature of the metafeature in the sense that instead of being a metric of redundancy is a metric of information. The greater the correlation between a numeric feature and target, the more informative that feature can be. Furthermore, it can be more useful to use a specific descriptive statistic (maximum, minimum, etc) instead of the typical mean.

Similarly, the *correlation between numeric features and target* has the same purpose of *distribution of correlation between numeric features and target* but it is indicated for MtL in which the base-level data has the same morphology (as in the data streams scenario [20]). The output of the metafeature is the correlation between the target and each numeric feature.

The two last metafeatures presented in Figure 6, (*correlation between predictions and target* and *absolute mean correlation between numeric features and target of two instances*) were developed using our framework by changing elements of specific components. *Correlation between numeric predictions and target* is another form of landmarker in which instead of using a typical error measure as RMSE, one uses correlation to assess the similarity between the real values and the predicted ones. In terms of the framework decomposition, this metafeature differs from the typical landmarkers in the aggregation function component. Although we did not yet executed experiments on the usefulness of metafeature, it is here proposed to exemplify the applicability of the framework to uncover new metafeatures for a given problem.

Finally, the *distribution of correlation between numeric features and target of instances* can be particularly useful for dynamic selection of algorithms/models in a regression scenario [18][10]. If the MtL problem concerns the selection of an algorithm for each instance of the test set (instead of an algorithm for a dataset) it could be useful to collect information that relates instances. This metafeature would allow to measure the correlation between the numeric variables of the instances. Once again, to the best of our knowledge, there are no reported experiments on the dynamic selection of algorithms using MtL. This metafeature is here proposed as another example of metafeatures that can be developed using correlation as metric.

6 Final Remarks and Future Work

This paper proposes a framework to decompose and develop new metafeatures for MtL problems. We believe that such framework can assist MtL researchers and users by standardizing the concept of metafeature.

We presented the framework and we used it to analyze several metafeatures proposed in the literature for a wide range of MtL scenarios. This process allowed to validate the usefulness of the framework by distinguishing several state-of-the-art metafeatures. We also provide insights on how the framework can be used to develop new metafeatures for a algorithm recommendation in a regression scenario. We use correlation between numeric variables to exemplify the applicability of the framework.

As for future work, we plan to use this framework to generate new metafeatures for algorithm recommendation in a classification scenario and empirically validate the framework. Furthermore, we also plan to use the framework in MtL problems that we have been working on, particularly, MtL for pruning of bagging ensembles and dynamic integration of models.

Acknowledgements

This work is partially funded by FCT/MEC through PIDDAC and ERDF/ON2 within project NORTE-07-0124-FEDER-000059, a

project financed by the North Portugal Regional Operational Programme (ON.2 O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF), and by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT).

REFERENCES

[1] Shawkat Ali and Kate A Smith-Miles, 'A meta-learning approach to automatic kernel selection for support vector machines', *Neurocomputing*, **70**(1), 173–186, (2006).

[2] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta, *Metalearning: applications to data mining*, Springer, 2008.

[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] Johannes Fürnkranz and Johann Petrak, '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).

[5] João Gama and Petr Kosina, 'Recurrent concepts in data streams classification', *Knowledge and Information Systems*, 1–19, (2013).

[6] Alexandros Kalousis, João Gama, and Melanie Hilario, 'On data and algorithms: Understanding inductive performance', *Machine Learning*, **54**(3), 275–312, (2004).

[7] Christian Köpf, Charles Taylor, and Jörg Keller, 'Meta-analysis: from data characterisation for meta-learning to meta-regression', in *Proceedings of the PKDD-00 workshop on data mining, decision support, meta-learning and ILP*. Citeseer, (2000).

[8] Petr Kuba, Pavel Brazdil, Carlos Soares, and Adam Woznica, 'Exploiting sampling and meta-learning for parameter setting for support vector machines', in *Proc. of Workshop Learning and Data Mining associated with Iberamia 2002, VIII Iberoamerican Conference on Artificial Intellignce*, pp. 209–216, Sevilla (Spain), (2002). University of Sevilla.

[9] Christiane Lemke and Bogdan Gabrys, 'Meta-learning for time series forecasting and forecast combination', *Neurocomputing*, **73**(10), 2006–2016, (2010).

[10] João Mendes-Moreira, Alipio Mario Jorge, Carlos Soares, and Jorge Freire de Sousa, 'Ensemble learning: A study on different variants of the dynamic selection approach', in *Machine Learning and Data Mining in Pattern Recognition*, 191–205, Springer, (2009).

[11] João Mendes-Moreira, Carlos Soares, Alípio Mário Jorge, and Jorge Freire De Sousa, 'Ensemble approaches for regression: A survey', *ACM Computing Surveys (CSUR)*, **45**(1), 10, (2012).

[12] Gleison Morais and Ronaldo C Prati, 'Complex network measures for data set characterization', in *Intelligent Systems (BRACIS), 2013 Brazilian Conference on*, pp. 12–18. IEEE, (2013).

[13] Yonghong Peng, Peter A Flach, Carlos Soares, and Pavel Brazdil, 'Improved dataset characterisation for meta-learning', in *Discovery Science*, pp. 141–152. Springer, (2002).

[14] Bernhard Pfahringer, Hilan Bensusan, and Christophe Giraud-Carrier, '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] Fábio Pinto, Carlos Soares, and João Mendes-Moreira, 'An empirical methodology to analyze the behavior of bagging', in *Submitted for publication*, (2014).

[16] Ricardo BC Prudêncio and Teresa B Ludermir, 'Meta-learning approaches to selecting time series models', *Neurocomputing*, **61**, 121–137, (2004).

[17] Matthias Reif, Faisal Shafait, and Andreas Dengel, 'Meta-learning for evolutionary parameter optimization of classifiers', *Machine learning*, **87**(3), 357–380, (2012).

[18] Niall Rooney, David Patterson, Sarab Anand, and Alexey Tsymbal, 'Dynamic integration of regression models', in *Multiple Classifier Systems*, 164–173, Springer, (2004).

[19] André Luis Debiaso Rossi, ACPLF Carvalho, and Carlos Soares, 'Meta-learning for periodic algorithm selection in time-changing data', in *Neural Networks (SBRN), 2012 Brazilian Symposium on*, pp. 7–12. IEEE, (2012).

[20] André Luis Debiaso Rossi, André Carlos Ponce De Leon Ferreira De Carvalho, Carlos Soares, and Bruno Feres De Souza, 'Metastream: A meta-learning based method for periodic algorithm selection in time-changing data', *Neurocomputing*, **127**, 52–64, (2014).

[21] Floarea Serban, Joaquin Vanschoren, Jörg-Uwe Kietz, and Abraham Bernstein, 'A survey of intelligent assistants for data analysis', *ACM Computing Surveys (CSUR)*, **45**(3), 31, (2013).

[22] Carlos Soares, Pavel B Brazdil, and Petr Kuba, 'A meta-learning method to select the kernel width in support vector regression', *Machine Learning*, **54**(3), 195–209, (2004).

[23] Quan Sun and Bernhard Pfahringer, 'Pairwise meta-rules for better meta-learning-based algorithm ranking', *Machine learning*, **93**(1), 141–161, (2013).

[24] Ljupčo Todorovski and Sašo Džeroski, 'Combining classifiers with meta decision trees', *Machine learning*, **50**(3), 223–249, (2003).

[25] Joaquin Vanschoren and Hendrik Blockeel, 'Towards understanding learning behavior', in *Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands*, pp. 89–96, (2006).

[26] Xiaozhe Wang, Kate Smith-Miles, and Rob Hyndman, 'Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series', *Neurocomputing*, **72**(10), 2581–2594, (2009).

Towards Meta-learning over Data Streams (Abstract)

Jan N. van Rijn[1] and **Geoffrey Holmes**[2] and **Bernhard Pfahringer**[3] and **Joaquin Vanschoren**[4]

Modern society produces vast streams of data. Many stream mining algorithms have been developed to capture general trends in these streams, and make predictions for future observations, but relatively little is known about which algorithms perform particularly well on which kinds of data. Moreover, it is possible that the characteristics of the data change over time, and thus that a different algorithm should be recommended at various points in time. Figure 1 illustrates this. As such, we are dealing with the Algorithm Selection Problem [9] in a data stream setting. Based on measurable *meta-features* from a window of observations from a data stream, a *meta-algorithm* is built that predicts the best classifier for the next window. Our results show that this meta-algorithm is competitive with state-of-the-art data streaming ensembles, such as `OzaBag` [6], `OzaBoost` [6] and `Leveraged Bagging` [3].

We first construct a meta-dataset consisting of 49 data streams, generated using various data stream generators from the MOA workbench [2], including the Rotating Hyperplane Generator and Random RBF Generator. In addition, we use a newly created Bayesian Network Generator, which takes a dataset as input, preferably consisting of real-world data and a reasonable amount of features, and builds a Bayesian Network using this dataset as input [12]. The Bayesian Network is then used to generate a data stream, determining each feature of each instance using the probability tables. These streams all contain 1,000,000 instances. We also include commonly used large datasets, such as Covertype, Pokerhand and the 20 Newsgroups dataset.

We run three types of classifiers over these datasets [8]. These are instance incremental classifiers, which learn from each example as it arrives, batch incremental classifiers, which learn from batches of examples, and ensembles of classifiers. The score of these classifiers is recorded at each window of 1,000 instances. Furthermore, we calculate various meta-features for all of these intervals, most of which are described in [10]. These meta-features are typically categorised as one of the following: simple (number of instances, number of attributes, number of classes), statistical (mean standard deviation of attributes, mean kurtosis of attributes, mean skewness of attributes), information theoretic (class entropy, mean entropy of attributes, noise-signal ratio) or landmarkers [7] (performance of a simple classifier on the data). We also introduce stream-specific meta-features based on change detection, which count the number of changes detected by the `ADWIN` [1] and `DDM` [4] change detectors.

The results of all experiments, as well as the generated datasets, classifiers used, and the meta-dataset itself, are available on OpenML [11].

[1] Leiden University, Leiden, Netherlands, j.n.van.rijn@liacs.leidenuniv.nl
[2] University of Waikato, Hamilton, New Zealand, geoff@cs.waikato.ac.nz
[3] University of Waikato, Hamilton, New Zealand,bernhard@cs.waikato.ac.nz
[4] Eindhoven University of Technology, Eindhoven, Netherlands, j.vanschoren@tue.nl

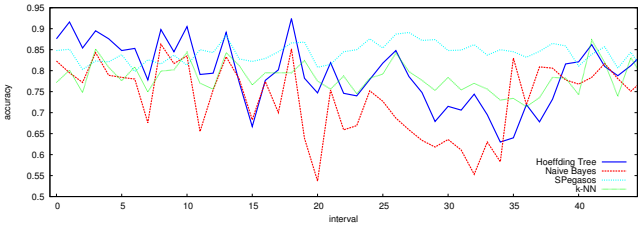


Figure 1: Performance of four instance incremental classifiers on intervals of the electricity dataset. Each interval contains 1,000 instances.

We now aim to determine whether this meta-knowledge can improve the predictive performance of data stream algorithms. We run a sliding window of 1,000 examples over each of the base data streams, and train a meta-algorithm using the meta-features and classifier scores for that window to predict which classifier should be used in the next window. The meta-algorithm is a `Random Forest` using 100 trees and 10 attributes, as implemented in Weka [5]. We distinguish between *meta-level accuracy* and *base-level accuracy*. Meta-level accuracy indicates how the meta-algorithm performs on the meta-learning task of predicting the best algorithm for a given window; base-level accuracy indicates how an ensemble of these base classifiers would actually perform on the base data stream, using the meta-algorithm to decide which base classifier to use for each window. The choice of meta-algorithm and the window size were determined experimentally.

Table 1 shows the results obtained from this experiment. We evaluate how well the meta-learning selects between 13 base stream classifiers, listed in Table 2. All classifiers are run with the default parameter settings as selected in MOA [2]. As described above, we can distinguish between three different types of stream mining algorithms, and we evaluate how the meta-learning approach performs within these subgroups as well.

Column A indicates the number of classifiers of each type, also indicated in Table 2. Column "Majority" denotes which classifier is the overall best in each group; here `HT` is short for `Hoeffding Trees`, `SMO` stands for a `Support Vector Machine` with a `Polynomial Kernel` and `LB-HT` means `Leveraged Bagging Hoeffding Trees`. The column "Percentage" shows the percentage of 1,000-example windows where this overall best algorithm wins. Since the meta-learner has to predict which base classifier to use in each window, this value represents the default accuracy of the meta-learning task.

Next, $\texttt{RF}_{meta}$ shows the accuracy of the `Random Forest` meta-

Table 1: Results of algorithm selection in the stream setting.

Task	A	Majority	Percentage	$\texttt{RF}_{meta}$	$\texttt{ZeroR}_{base}$	$\texttt{RF}_{base}$	$\texttt{MAX}_{base}$
Instance incremental	5	HT	59.75	80.78	80.98	84.07	84.59
Batch incremental	4	SMO	65.56	68.17	74.38	75.33	76.02
Ensembles	4	LB-HT	57.78	56.20	84.27	85.15	86.12
All classifiers	13	LB-HT	50.97	50.92	84.27	85.31	86.30

Table 2: Algorithms used in the experiments.

Key	Classifier	Type	Parameters
NB	NaiveBayes	Instance incremental	
SGD	Stochastic Gradient Descent	Instance incremental	
SPeg	SPegasus	Instance incremental	
k-NN	k Nearest Neighbour	Instance incremental	$k = 10, w = 1000$
HT	Hoeffding Tree	Instance incremental	
SMO	Support Vector Machine / Polynomial Kernel	Batch incremental	$w = 1000$
J48	C4.5 Decision Tree	Batch incremental	$w = 1000$
REP	Reduced-Error Pruning Decision Tree	Batch incremental	$w = 1000$
OneR	One Rule	Batch incremental	$w = 1000$
LB-kNN	Leveraging Bagging / k-NN	Ensemble	$k = 10, n = 10, w = 1000$
LB-HT	Leveraging Bagging / Hoeffding Tree	Ensemble	$n = 10$
Bag-HT	OzaBag / Hoeffding Tree	Ensemble	$n = 10$
Boost-HT	OzaBoost / Hoeffding Tree	Ensemble	$n = 10$

classifier in predicting the best classifier for a given window. The last three columns show the accuracy that can be obtained on the base data stream using three different strategies. Column $\texttt{ZeroR}_{base}$ shows the accuracy obtained by always selecting the best overall base classifier. For instance, the value in the "Ensembles" row shows the accuracy of an ensemble of Leveraged Bagged Hoeffding Trees, averaged over all data streams. $\texttt{RF}_{base}$ shows the accuracy obtained when the Random Forest meta-classifier predicts the base classifier to be used in each window, again averaged over all data streams. Finally, column $\texttt{MAX}_{base}$ shows the accuracy obtained if the meta-classifier always correctly predicted the best classifier for each window. Intuitively, $\texttt{RF}_{base}$ shows the performance of the meta-classifier, $\texttt{ZeroR}_{base}$ can be used as a baseline, and $\texttt{MAX}_{base}$ shows the maximum score that the meta-classifier could have obtained.

Determining the best instance incremental classifier yields good results. In more than 80% of the cases, the correct classifier is predicted. This also translates into good base-level performance. An ensemble of our meta-classifier and only the 5 instance incremental classifiers, which is markedly cheaper to train, yields a score of 84.07%, which not only outperforms the best overall instance incremental classifier, a Hoeffding Tree with 80.98% accuracy, but is also comparable to the best overall base classifier, a Leveraged Bagged Hoeffding Trees ensemble (with 10 base-classifiers), which scores 84.27%. Moreover, it also outperforms the other ensembles, OzaBag (82.58%) and OzaBoost (80.55%). The Random Forest meta-learner has more difficulty selecting among all 13 base-classifiers, which shows room for progress, but even then it performs slightly better than the overall best base classifier. Furthermore, the $\texttt{RF}_{base}$ performances are in many cases close to the maximal possible value, $\texttt{MAX}_{base}$. This indicates that the main challenge is to find ways to improve this limit. Better results are likely to be obtained using parameter optimisation, and by using a larger set of algorithms.

REFERENCES

[1] A. Bifet and R. Gavalda. Learning from Time-Changing Data with Adaptive Windowing. In *SDM*, volume 7, pages 139–148. SIAM, 2007.

[2] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis. *J. Mach. Learn. Res.*, 11:1601–1604, 2010.

[3] A. Bifet, G. Holmes, and B. Pfahringer. Leveraging Bagging for Evolving Data Streams. In *Machine Learning and Knowledge Discovery in Databases*, volume 6321 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2010.

[4] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with Drift Detection. In *SBIA Brazilian Symposium on Artificial Intelligence*, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer, 2004.

[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[6] Nikunj C Oza. Online Bagging and Boosting. In *Systems, man and cybernetics, 2005 IEEE international conference on*, volume 3, pages 2340–2345. IEEE, 2005.

[7] Bernhard Pfahringer, Hilan Bensusan, and Christophe Giraud-Carrier. 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*, pages 743–750, 2000.

[8] J. Read, A. Bifet, B. Pfahringer, and G. Holmes. Batch-Incremental versus Instance-Incremental Learning in Dynamic and Evolving Data. In *Advances in Intelligent Data Analysis XI*, pages 313–323. Springer, 2012.

[9] J. R. Rice. The Algorithm Selection Problem. *Advances in Computers*, 15:65118, 1976.

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

[11] J. N. van Rijn, B. Bischl, L. Torgo, B. Gao, V. Umaashankar, S. Fischer, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren. OpenML: A Collaborative Science Platform. In *Machine Learning and Knowledge Discovery in Databases*, pages 645–649. Springer, 2013.

[12] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. The Bayesian Network Generator: A data stream generator. Technical Report 03/2014, Computer Science Department, University of Waikato, 2014.

Recommending Learning Algorithms and Their Associated Hyperparameters

Michael R. Smith[1] and **Logan Mitchell**[2] and **Christophe Giraud-Carrier**[3] and **Tony Martinez**[4]

Abstract. The success of machine learning on a given task depends on, among other things, which learning algorithm is selected and its associated hyperparameters. Selecting an appropriate learning algorithm and setting its hyperparameters for a given data set can be a challenging task, especially for users who are not experts in machine learning. Previous work has examined using meta-features to predict which learning algorithm and hyperparameters should be used. However, choosing a set of meta-features that are predictive of algorithm performance is difficult. Here, we propose to apply collaborative filtering techniques to learning algorithm and hyperparameter selection, and find that doing so avoids determining which meta-features to use and outperforms traditional meta-learning approaches in many cases.

1 Introduction

Most previous meta-learning work has focused on selecting a learning algorithm or a set of hyperparameters based on meta-features used to characterize datasets [5]. As such, it can be viewed as a form of content-based filtering, a technique commonly-used in recommender systems that captures a set of measured characteristics of an item and/or user to recommend items with similar characteristics. On the other hand, collaborative filtering (CF), also used by some recommender systems, predicts the rating or preference that a user would give to an item, based on the past behavior of a set of users, characterized by ratings assigned by users to a set of items [9]. The underlying assumption of CF is that if users A and B agree on some issues, then user A is more likely to have the same opinion on a new issue X as user B than another randomly chosen user. A key advantage of CF is that it does not rely on directly measurable characteristics of the items. Thus, it is capable of modeling complex items without actually understanding the items themselves.

Here, we propose *meta-CF* (MCF) a novel approach to meta-learning that applies CF in the context of algorithm and/or hyperparameter selection. MCF differs from most previous meta-learning techniques in that it does not rely on meta-features. Instead, MCF considers the similarity of the performance of the learning algorithms with their associated hyperparameter settings from previous experiments. In this sense, the approach is more similar to landmarking [12] and active testing [10] since both also use the performance results from previous experiments to determine similarity among data sets.

While algorithm selection and hyperparameter optimization have been mostly studied in isolation (e.g., see [12, 4, 1, 2, 3, 15]), recent work has begun to consider them in tandem. For example, Auto-WEKA simultaneously chooses a learning algorithm and sets its hyperparameters using Bayesian optimization over a tree-structured representation of the combined space of learning algorithms and their hyperparameters [16]. All of these approaches face the difficult challenge of determining a set of meta-features that capture relevant and predictive characteristics of datasets. By contrast, MCF does consider both algorithm selection and hyperparameter setting at once, but alleviates the problem of meta-feature selection by leveraging information from previous experiments through collaborative filtering.

Our results suggest that using MCF for learning algorithm/hyperparameter setting recommendation is a promising direction. Using MCF for algorithm recommendation has some differences from the traditional CF used for human ratings. For example, CF for humans may have to deal with concept drift, where a user's taste may change over time; working with learning algorithms and hyperparameter settings is deterministic.

2 Empirical Evaluation

For MCF, we examine several CF techniques implemented in the Waffles toolkit [6]: baseline (predict the mean of the previously seen results), Fuzzy K-Means (FKM) [11], Matrix Factorization (MF) [9], Nonlinear PCA (NLPCA) [13], and Unsupervised Backpropagation (UBP) [7].

To establish a baseline, we first calculate the accuracy on a set of 125 data sets and 9 diverse learning algorithms (see [14] for a discussion on diversity) with default parameters as set in Weka [8]. The set of learning algorithms is composed of backpropagation (BP), C4.5, kNN, locally weight learning (LWL), naïve Bayes (NB), nearest neighbor with generalization (NNge), random forest (RF), ridor (Rid), and RIPPER (RIP). We select the accuracy from the learning algorithm that produces the highest classification accuracy. This represents algorithm selection with perfect recall. We also estimate the hyperparameter optimized accuracies for each learning algorithm using random hyperparameter optimization [3]. The results are shown in Table 1, where the accuracy from each learning algorithm is the average hyperparameter optimized accuracy for each data set, "Default" refers to the best accuracy from the learning algorithm with its default parameters, "ALL" refers to the accuracy from the best learning algorithm and hyperparameter setting, and "AW" refers to the results from running Auto-WEKA. For Auto-WEKA, each dataset was allowed to run as long as the longest algorithm took to run on the dataset when doing the random hyperparameter optimization. As Auto-WEKA is a random algorithm, we ran 4 runs each time with a different seed and chose the seed with highest accuracy. This can be seen as equivalent to allowing a user to run on average 16 learning

[1] Brigham Young University, USA, email: msmith@axon.cs.byu.edu
[2] Brigham Young University, USA, email: mitchlam711@gmail.com
[3] Brigham Young University, USA, email: cgc@cs.byu.edu
[4] Brigham Young University, USA, email: martinez@cs.byu.edu

algorithm and hyperparameter combinations on a data set.

Table 1. Average accuracy for the best hyperparameter setting for each learning algorithm, algorithm selection (Default), both algorithm selection and hyperparameter optimization (ALL), and Auto-WEKA (AW).

BP	C4.5	kNN	LWL	NB	NNge
79.89	79.22	78.05	77.48	76.04	76.80

RF	Rid	RIP	Default	ALL	AW
79.58	71.48	77.31	81.93	83.00	82.00

For MCF, we compiled the results from hyperparameter optimization. We randomly removed 10% to 90% of the results by increments of 10% and then used MCF to fill in the missing values. The top 4 learning algorithm/hyperparameter configurations are returned by the CF technique and the accuracy from the configuration that returns the highest classification accuracy is used. This process was repeated 10 times. A summary of the average results for MCF are provided in Table 2. The columns "Best", "Median", and "Average" refer to the accuracies averaged across all of the sparsity levels for the hyperparameter setting for the CF technique that provided the results. The columns 0.1 to 0.9 refer to the percentage of the results used for CF averaged over the hyperparameter settings. The row "Content" refers to meta-learning where a learning algorithm recommends a learning algorithm based on a set of meta-features.

Table 2. Average accuracy from the best of the top 4 recommended learning algorithm and hyperparameter settings from MCF.

	Best	Med	Ave	0.1	0.3	0.5	0.7	0.9
Baseline	81.11	81.11	81.11	**80.49**	80.91	81.12	81.33	81.54
FKM	81.52	81.04	81.29	80.13	80.65	81.07	81.45	81.88
MF	**82.12**	**82.06**	**81.95**	**80.49**	**81.63**	**82.12**	**82.44**	**82.65**
NLPCA	81.73	81.33	81.33	79.98	80.58	81.43	82.08	82.61
UBP	81.73	81.27	81.31	80.05	80.51	81.34	82.05	82.61
Content	81.35	80.47	78.91	-	-	-	-	-

Overall, MF achieves the highest accuracy values. The effectiveness of MCF increases as the percentage of the results increases. MCF significantly increases the classification accuracy compared with both hyperparameter optimization for a given learning algorithm and model selection with their default parameters as well as using the meta-features to predict which learning algorithm and hyperparameters to use. On average, MCF and Auto-WEKA achieve similar accuracy, which highlights the importance of considering *both* algorithm selection and hyperparameter optimization. However, provided one has access to a database of experiments, such as the ExperimentDB [17], MCF only requires the time to run a number of algorithms (often ran in parallel), and retraining the collaborative filter. In the current implementation, retraining takes less than 10 seconds. Thus, MCF presents an efficient method for recommending a learning algorithm and its associated hyperparameters.

While our results show that MCF is a viable technique for recommending learning algorithms *and* hyperparameters, some work remains to be done. Future work for MCF includes addressing the cold-start problem which occurs when a data set is presented and no learning algorithm has been ran on it. MCF is adept at exploiting the space that has already been explored, but (like active testing) it does not explore unknown spaces at all. One way to overcome this limitation would be to use a hybrid recommendation system that combines content-based filtering and MCF.

REFERENCES

[1] S. Ali and K.A. Smith, 'On Learning Algorithm Selection for Classification', *Applied Soft Computing*, **6**2, 119–138, (2006).

[2] S. Ali and K.A. Smith-Miles, 'A Meta-learning Approach to Automatic Kernel Selection for Support Vector Machines', *Neurocomputing*, **70**, 173–186, (2006).

[3] J. Bergstra and Y. Bengio, 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research*, **13**, 281–305, (2012).

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

[5] P. Brazdil and C. Giraud-Carrier and C. Soares and R. Vilalta, 'Metalearning: Applications to Data Mining', Springer, (2009).

[6] M. S. Gashler, 'Waffles: A machine learning toolkit', *Journal of Machine Learning Research*, **MLOSS 12**, 2383–2387, (July 2011).

[7] M. S. Gashler, M. R. Smith, R. Morris, and T. Martinez, 'Missing value imputation with unsupervised backpropagation', *Computational Intelligence*, Accepted, (2014).

[8] 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).

[9] Y. Koren, R. Bell, and C. Volinsky, 'Matrix factorization techniques for recommender systems', *Computer*, **42**(8), 30–37, (2009).

[10] R. Leite, P. Brazdil, and J. Vanschoren, 'Selecting classification algorithms with active testing', in *Machine Learning and Data Mining in Pattern Recognition*, ed., Petra Perner, volume 7376 of *Lecture Notes in Computer Science*, 117–131, Springer Berlin / Heidelberg, (2012).

[11] D. Li, J. Deogun, W. Spaulding, and B. Shuart, 'Towards missing data imputation: A study of fuzzy k-means clustering method', in *Rough Sets and Current Trends in Computing*, volume 3066 of *Lecture Notes in Computer Science*, 573–579, Springer Berlin / Heidelberg, (2004).

[12] B. Pfahringer, H. Bensusan, and C. G. Giraud-Carrier, 'Meta-learning by landmarking various learning algorithms', in *Proceedings of the 17th International Conference on Machine Learning*, pp. 743–750, San Francisco, CA, USA, (2000). Morgan Kaufmann Publishers Inc.

[13] M. Scholz, F. Kaplan, C. L. Guy, J. Kopka, and J. Selbig, 'Non-linear pca: a missing data approach', *Bioinformatics*, **21**(20), 3887–3895, (2005).

[14] M. R. Smith, T. Martinez, and C. Giraud-Carrier, 'An instance level analysis of data complexity', *Machine Learning*, **95**(2), 225–256, (2014).

[15] J. Snoek, H. Larochelle, and R. Adams, 'Practical bayesian optimization of machine learning algorithms', in *Advances in Neural Information Processing Systems 25*, eds., F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, 2951–2959, (2012).

[16] 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*, pp. 847–855, (2013).

[17] J. Vanschoren, H. Blockeel, B. Pfahringer, and G. Holmes, 'Experiment databases - a new way to share, organize and learn from experiments', *Machine Learning*, **87**(2), 127–158, (2012).

An Easy to Use Repository for Comparing and Improving Machine Learning Algorithm Usage

Michael R. Smith[1] and **Andrew White**[2] and **Christophe Giraud-Carrier**[3] and **Tony Martinez**[4]

Abstract. The results from most machine learning experiments are used for a specific purpose and then discarded. This causes significant loss of information and requires rerunning experiments to compare learning algorithms. Often, this also requires a researcher or practitioner to implement another algorithm for comparison, that may not always be correctly implemented. By storing the results from previous experiments, machine learning algorithms can be compared easily and the knowledge gained from them can be used to improve the performance of future machine learning experiments. The purpose of this work is to provide easy access to previous experimental results for learning and comparison. These stored results are comprehensive – storing the prediction for each test instance as well as the learning algorithm, hyperparameters, and training set that were used in the experiment. Previous experimental results are particularly important for meta-learning, which, in a broad sense, is the process of learning from previous machine learning results such that the learning process is improved. While other experiment databases do exist, one of our focuses is on easy access to the data, eliminating any learning curve required to acquire the desired information. We provide meta-learning data sets that are ready to be downloaded for meta-learning experiments. Easy access to previous experimental results aids other researchers looking to do meta-learning and helps in comparing meta-learning algorithms. In addition, simple queries to the underlying database can be made if specific information is desired. We also differ from previous experiment databases in that our database is designed at the instance level, where an instance is an example in a data set. We store the predictions of a learning algorithm trained on a specific training set for each instance in the test set. Data set level information can then be obtained by aggregating the results from the instances. The instance level information can be used for many tasks such as determining the diversity of a classifier or algorithmically determining the optimal subset of training instances for a learning algorithm.

1 Introduction

The quality of an induced model is dependent on, among other aspects, the learning algorithm that is chosen, the hyper-parameter settings for the chosen learning algorithm, and the quality of the training set. Choosing a learning algorithm for a given task, setting its hyper-parameters, and selecting which instances to train on, however, is non-trivial. Meta-learning deals with the problem of how to select a learning algorithm and set its hyper-parameters based on previous experience (results from previous machine learning experiments). Although some research from the machine learning community has focused on meta-learning (e.g., see [17, 5, 2, 3, 8]), much of the focus of machine learning research has been on developing more learning algorithms and/or applying machine learning in specific domains.

Part of the difficulty of meta-learning is due to the lack of accessible results. As meta-learning requires running several learning algorithms and hyperparameter settings over many data sets, gathering results requires large amounts of computational resources. In addition to the computational requirements, results from the learning algorithms may differ due to slight differences in their implementations. Thus, comparing results among meta-learning studies becomes difficult.

To aid in further research in meta-learning, we have developed the *machine learning results repository* (MLRR) that provides data sets ready for download for meta-learning problems, akin to the UCI data repository for machine learning problems. We refer to the data sets for meta-learning as *meta-data sets* to distinguish them from the data sets that are used in the machine learning experiments. The meta-data sets provide a snapshot of an underlying database that stores the results of machine learning experiments. Users can update the database with new results from machine learning experiments and then update the meta-data sets for meta-learning. A revision history is kept so that comparisons among meta-learning algorithms is facilitated. As a starting point, meta-data sets are provided by MLRR for typical meta-learning tasks, such as, given a set of meta-features, predict which learning algorithm and/or hyperparameter setting to use.

The MLRR stores instance level meta-features and the predictions made on each instance by the learning algorithms. Providing information at the instance level allows studies to be performed on the instances themselves. Studying the effects of machine learning on a single instance and/or the effects of a single instance on the performance of an algorithm has generally been overlooked. Instance-level information is important in several areas of machine learning, however. In ensembles, computing the classifier diversity of the ensembled classifiers using the predictions for each instance is important in determining the effectiveness of the ensembling technique [12, 6, 1]. In curriculum learning, the training set is incrementally augmented such that "easier" instances are presented to the learning algorithm first, thus creating a need to understand and identify the easier instances [4]. Smith et al. used instance-level predictions to identify and characterize instances that are likely to be misclassified [23] and used this information to create a curriculum [22]. Other work has also used the instance-level predictions for meta-learning. The classifier output difference (COD) measures the distance between two learning algorithms as the probability that the learning algorithms make different predictions on test instances [16]. Unsupervised meta-

[1] Brigham Young University, USA, email: msmith@axon.cs.byu.edu
[2] Brigham Young University, USA, email: andrewkvavlewhite@gmail.com
[3] Brigham Young University, USA, email: cgc@cs.byu.edu
[4] Brigham Young University, USA, email: martinez@cs.byu.edu

learning (UML) clusters learning algorithms based on their COD scores (rather than accuracy) to examine the behavior of the learning algorithms [13]. Meta-learning for algorithm selection can then be done over the clusters rather than a larger set of learning algorithms to recommend a cluster of learning algorithms that all behave similarly [14]. Additionally, several techniques treat instances individually during the training process, such as filtering instances from the training set based on their instance-level meta-features [21] or weighting the instances [18].

Other attempts have been made at creating a repository for machine learning experiments from which learning can be conducted [20, 24]. However, we feel that they lack simplicity and/or extensibility. In addition to providing instance-level information, we hope to bridge this gap with the MLRR. Probably the most prominent and well-developed data repository is ExpDB, an experiment database that provides a framework for reporting experimental results and their associated workflow [24]. The purpose of ExpDB is to comprehensively store the workflow process of all experiments for reproducibility. One of the results of storing the experiments is that the results can be used for meta-learning. Unfortunately, there is a relatively steep learning curve to access the data due to the inherent complexity involved in storing all of the details about exact reproducibility. Because of this complexity and formality, it is difficult to directly access the information that would be most beneficial for meta-learning, which may deter some potential users. Additionally, ExpDB does not currently support storage and manipulation of any instance level features.

We acknowledge that maintaining a database of previous experiments is not a trivial problem. We do, however, add our voice to support the importance of maintaining a repository of machine learning results and offer an effective solution for storing results from previous experiments. Our primary goal is to maintain simplicity and provide easily accessible data for meta-learning to 1) help promote more research in meta-learning, 2) provide a standard set of data sets for meta-learning algorithm comparison, and 3) continue to stimulate research at the instance level.

We next describe our approach for providing a repository for machine learning meta-data that emphasizes ease of access to the meta-data. MLRR currently has the results from 72 data sets, 9 learning algorithms and 10 hyperparameter settings for each learning algorithm. The database description is provided in Section 3. How to add new experimental results to the database is detailed in Section 4. We then give a more detailed description of the data set level and instance level meta-features that are used in the MLRR. Conclusions and directions for future work are provided in Section 6.

2 Meta-data Set Descriptions

The purpose of the *machine learning results repository* (MLRR) is to provide easy access to the results of previous machine learning experiments for meta-learning at the data set and instance levels. This, in turn, would allow other researchers interested in meta-learning and in better understanding machine learning algorithms direct access to prior results without having to re-run all of the algorithms or learn how to navigate a more complex experiment database. The quality of an induced model for a task is dependent on at least three things:

1. the learning algorithm chosen to induce the model,
2. the hyperparameter settings for the chosen learning algorithm, and
3. the instances used for training.

When we refer to an experiment, we mean the results from training a learning algorithm l with hyperparamter settings λ on a training set t. We first describe how we manage experiment information, and then describe the provided meta-data sets.

2.1 Experiment Information

The information about each experiment is provided in three tables in MLRR. Which learning algorithm and hyperparameters were used is provided in a file structured as shown in Table 1. It provides the toolkit including the version number that was ran, the learning algorithm, and the hyperparameters that were used. This allows for multiple learning algorithms, hyperparameters, and toolkits to be compared. In the examples in Table 1, the class names from the Weka machine learning toolkit [9] and the Waffles machine learning toolkit [7] are shown. LA_seed corresponds to the learning algorithm that was used (LA) and to a seed that represents which hyperparameter setting was used (seed). The LA_seed will be used in other tables as a foreign key to map back to this table. A seed of -1 represents the default hyperparameter settings as many studies examine the default behavior as given in a toolkit and the default parameters are commonly used in practice.

Table 1. The structure of the meta-data set that describes the hyperparameter settings for the learning algorithms stored in the database.

LA_S	Toolkit	Version	Hyperparameters
BP_1	weka	3.6.11	weka.classifiers.functions.MultilayerPerceptron\ – -L 0.261703 -M 0.161703 -H 12 -D
BP_2	weka	3.6.11	weka.classifiers.functions.MultilayerPerceptron\ – -L 0.25807 -M 0.15807 -H 4
BP_3	waffles	13-12-09	neuralnet -addlayer 8 -learningrate 0.1 \ -momentum 0 -windowsepochs 50
⋮	⋮	⋮	⋮
C4.5_1	weka	3.6.11	weka.classifiers.trees.J48 – -C 0.443973 -M 1
⋮	⋮	⋮	⋮

As the parameter values differ between toolkits, there is a mapping provided to distinguish hyperparameter settings. For example, Weka uses the "-L" parameter to set the learning rate in backpropagation while the Waffles toolkit uses "-learningrate". Also, some toolkits have hyperparameters that other implementations of the same learning algorithm do not include. In such cases, an unknown value will be provided in the meta-data set. This mapping is shown in Table 2 for the backpropagation learning algorithm. The first row contains the values used by MLRR. The following rows contain the command-line parameter supplied to a specific toolkit to set that hyperparameter.

Table 2. The structure of the table for mapping learning algorithm hyperparameters between different toolkits for the backpropagation learning algorithm.

	Command line parameters				
toolkit	LR	Mo	HN	DC	WE
weka	-L	-M	-H	-D	?
waffles	-learningrate	-momentum	-addlayer	?	-windowsepochs
⋮	⋮	⋮	⋮	⋮	⋮

A mapping of which instances are used for training is also provided in a separate file. The structure of this table is shown in Table

3. Each row represents an experiment as toolkit_seed_numFolds_fold. The toolkit represents which toolkit was used, the seed represents the random seed that was provided to the toolkit, numFolds represents how many folds were ran, and fold represents in which fold an instance was included for testing. The values in the following columns represent if an instance was used for training or testing. There is one column for each instance in the data set. They are stored as real values. This allows for the situations when training instances have associated weights. In the file, an unknown value of "?" represents a testing instance, otherwise a real value represents a training instance. A value of 0 represents a filtered instance, a value of 1 represents an unweighted training instance and any value between 0 and 1 represents the weight for that training instance. In the cases where there are specific training and testing sets, then the row will be labeled as toolkit_0_0_1 and information for the training set can be entered as before. A random test/training split of the data is represented as toolkit_seed_percentSplit_1 where "percentSplit" represents the percentage of the data set that was used for testing as generated by the toolkit.

Table 3. The structure of the meta-data set that indicates which instances were used for training given a random seed.

toolkit_seed_# folds_fold	1	2	3	…
weka_1_10_1	1	1	1	…
weka_1_10_2	1	0	1	…
⋮	⋮	⋮	⋮	
weka_1_10_10	0.74	1	?	…
weka_2_1_10	?	1	1	…
⋮	⋮	⋮	⋮	

2.2 Meta-data sets

One of the features of MLRR is its focus on storing and presenting instance level information, namely, instance level characteristics and associated predictions from previous experiments. Indeed, the MLRR is designed intentionally from the instance level perspective, from which data set level information can be computed (e.g., accuracy or precision).

As one of the purposes of the MLRR is ease of access, the MLRR stores several data sets in attribute-relation file format (ARFF) which is supported by many machine learning toolkits. In essence, ARFF is a comma or space separated file with attribute information and possible comments. The precomputed meta-data sets include instance level meta-data sets and data set level meta-data sets.

At the instance level, MLRR provides for each data set a meta-data set that stores the instance level meta-features and the prediction from each experiment. This allows for analyses to be done exploring the effects of hyperparameters and learning algorithms at the instance-level, which is currently mostly overlooked. For each data set, a meta-data set is provided that gives the values for the instance level meta-features, the actual class value (stored as a numeric value), and the predicted class value for each experiment. The training set and learning algorithm/hyperparameter information is stored in the column heading as "LA_seed/hyperparameter" where LA is a learning algorithm and hyperparameter is the hyperparameter setting for the learning algorithm. Together, they map to the entries in Table 1. The seed represents the seed that was used to partition the data (see Table 3). The structure of the instance level meta-data set is shown in Table 4. In the given example, instance 77 is shown. The "inst meta" section provides the instance level meta-features for that instance. The actual class label is 2. The predictions from the experiments on this data set are provided in the following columns (i.e., experiment BP_1/1 predicted class 3, BP_N/1 predicted class 2, etc.).

Table 4. The structure of the meta-data set at the instance level.

	inst meta				predictions							
#	*k*AN	MV	…	act	BP_1/1	…	BP_N/1	…	BP_N/M	C4.5_1/1	…	
77	0.92	0	…	2	3	…	2	…	2	3	…	
⋮	⋮	⋮		⋮	⋮		⋮		⋮	⋮		

At the data set level, several meta-data sets are provided:

- a general meta-data set that stores the data set meta-features and the average N by 10-fold cross-validation accuracy for all of the data sets from a learning algorithm with a given hyperparameter setting.
- for each learning algorithm a meta-data set that stores the data set meta-features, the learning algorithm hyperparameter settings, and the average N by 10-fold cross-validation accuracy for all of the data sets for the given hyperparameter setting.

The structure for the general meta-data set is provided in Table 5. The structure and information of this meta-data set is typical of that used in previous meta-learning studies that provides a mapping from data set meta-features to accuracies obtained by a set of learning algorithms. Most previous studies have been limited to only using the default hyperparameters, however. The MLRR includes the accuracies from multiple hyperparameter settings. The hyperparameter settings from each learning algorithm are denoted by a "LA_#" where LA refers to a learning algorithm and # refers to which hyperparameter setting was used for that learning algorithm.

Table 5. The structure of the meta-data set at the data set level.

	data set meta-features			LA accuracies					
data set	numInst	numAttr	…	BP_1	BP_2	…	BP_N	C4.5_1	…
iris	150	4	…	96.80	95.07	…	93.47	95.60	…
abalone	4177	8	…	20.27	29.84	…	21.91	23.24	…
⋮	⋮	⋮		⋮	⋮	⋮	⋮	⋮	

The meta-data sets for each learning algorithm are designed to aid in algorithmic hyperparameter estimation, i.e., given a data set, can we predict which hyperparameter setting will give the highest classification accuracy. For each learning algorithm, a meta-data set is provided that contains the data set meta-features, the toolkit that was used, the hyperparameter setting and the average accuracy for each unique tool kit/hyperparameter combination. The structure of the meta-data set for each learning algorithm is provided in Table 6. The accuracy ("acc") represents the average accuracy for all k-fold validation runs (i.e., multiple runs of the same learning algorithm with different random seeds to partition the folds). The toolkit is also provided to allow a user to compare toolkits or only do hyperparameter estimation for a single toolkit.

MLRR provides easy access for researchers and practitioners to a large and varying set of meta-data information as shown in the tables above. The provided meta-data sets are a snapshot of an underlying

Table 6. The structure of the table for mapping learning algorithm hyperparameters among toolkits.

	DS meta features			toolkit	hyperparameters			
data set	numInst	numAttr	...	weka	LR	Mo	...	acc
iris	150	4	...	weka	0.71	0.61	...	96.80
iris	150	4	...	weka	0.11	0.25	...	97.04
⋮	⋮	⋮		⋮	⋮	⋮		⋮

database that stores all of the previous experimental results that can be updated as more results are obtained. A revision history of the data sets is provided so that results can be compared even if the meta-data set has been updated.

3 Database Description

MLRR uses MongoDB as the database to store the results from machine learning experiments. MongoDB is a NoSQL database that allows for adding new features (such as new learning algorithms and/hyperparameters), thus, escaping the rigidity of the more traditional SQL databases. This allows for easily expanding the database with new learning algorithms and/or hyperparameters. Of course, this is theoretically also possible in a relational database, provided the database has been designed adequately. For example, one could certainly have, and that would indeed be following good design principles, one table for the algorithms and one table for the hyper parameters with appropriate foreign keys. However, such design requires some amount of foresight. In traditional relational databases, the information that needs to be stored (and how) has to be planned for in advance. Otherwise, when new features are desired, a new schema needs to be created and then the database has to be migrated over to the new schema. With a NoSQL database, new learning algorithms/hyperparameters and other pieces of information can easily be added into the MLRR.

The data is stored as a document database as collections of key-value pairs. Each collection represents the experimental results on a particular data set. In each collection, the keys are LA_hyperparameterSetting. The value then is a JSON text document that stores the results of an experiment (e.g., the results of 10-fold cross-validation on the iris data set using C4.5). These documents also contain pointers to other documents that hold information about training/testing sets for each experiment. The data set/instance level meta-features are stored in separate documents in their respective data set collection. A separate collection stores information about the learning algorithms and their hyperparameters.

The best way to visualize the database is as a hierarchy of key-value pairs as shown in Figure 1. At the top-level, there are collections - these are the individual data sets in the database. Each of them holds a collection of documents that represent an output file, or experiment, named by its learning algorithm with two numbers that correspond to the random seed used to partition the data and the hyperparameter setting. In these documents, the predictions for each instance is stored. Collections for which instances were used for training hyperparameter settings are also included.

4 Extending the Database

The data provided by MLRR only contains a snapshot of current machine learning results. To allow more machine learning results to be added and to allow the MLRR to evolve as the state of machine learning evolves, MLRR provides a method to upload new machine learning results. The MLRR also stores the original data sets to allow a user to add results from additional experiments on the current set of data sets. The results from experimentation on a new data set require that the new data set be uploaded as well as the experimental results. Scripts are provided to calculate the meta-features for the new data set. In the case where a data set is proprietary or has other privacy/licensing issues that prevent it from being posted, the meta-features can be calculated on the data set without storing the actual data set.

Currently, scripts are provided to upload the output from running Weka. This provides a simple way to upload experimental results from a commonly used toolkit. The file is slightly modified such that the first line provides which learning algorithm and hyperparameters were used. The database will have the ability to upload files generated by other toolkits in the future.

Of course, there are issues of data reliability. Currently, all of the results stored in the MLRR are from our experiments. To help with data reliability, we require that the script(s) and executable(s) required to reproduce the results are uploaded along with the results. This allows the results to be verified if their validity is questioned. If the results from an experiment are thought to be invalid, they can can be flagged, and inspected for possible removal from the MLRR.

5 Included Meta-features

In this section, we detail the meta-features that are included in the machine learning results repository (MLRR). We store a set of data set meta-features that have been commonly used in previous meta-learning studies. Specifically, we used the meta-features from Brazdil et al. [5], Ho and Basu [10], Pfahringer et al. [17], and Smith et al. [23]. As the underlying database is a NoSQL database, additional meta-features can be easily added in the future. We now describe the meta-features from each study.

The study by Brazdil et al. [5] examined ranking learning algorithms using instance-based learning. The meta-features are designed to be quickly calculated and to represent properties that affect algorithm performance.

- *Number of examples*. This feature helps identify how scalable an algorithm is based on the size of its input.
- *Proportion of symbolic attributes*. This feature can be used to consider how well an algorithm deals with symbolic or numeric attributes.
- *Proportion of missing values*. This features can be used to consider how robust an algorithm is to incomplete data.
- *Proportion of attributes with outliers*. An attribute is considered to have an outlier if the ratio of variances of the mean value and the α-trimmed mean is smaller than 0.7 where $\alpha = 0.05$. This feature can be used to consider how robust an algorithm is to outlying numeric values.
- *Entropy of classes*. This feature measures one aspect of problem difficulty in the form of whether one class outnumbers another.

Ho and Basu [10] sought to measure the complexity of a data set to identify areas of the data set that contribute to its complexity focusing on the geometrical complexity of the class boundary.

- Measures of overlap of individual feature values:
  - *The maximum Fisher's Discriminant ratio*. This is the Fisher's discriminant ratio for an attribute:

$$f = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2},$$

where μ_i and σ_i^2 represent the mean and variance for a class. The maximum Fisher's discriminant value over the attributes is used for this measure. For multiple classes, this measure is expanded to:

$$f = \frac{\sum_{i=1}^{C} \sum_{j=i+1}^{C} p_i p_j (\mu_i - \mu_j)^2}{\sum_{i=1}^{C} p_i \sigma_i^2}$$

where C is the number of classes and p_i is the proportion of instances that belong to the i^{th} class.

- *The overlap of the per-class bounding boxes*. This feature measures the overlap of the tails of the two class-conditional distributions. For data sets with more than 2 classes, the overlap of the per-class bounding boxes is computed for each pair of classes and the sum over all pairs of classes is returned.
- *The maximum (individual) feature efficiency*. This feature measures how discriminative a single feature is. For each attribute, the ratio of instances with differing classes that are not in the overlapping region is returned. The attribute that produces the largest ratio of instances is returned.
- *The collective feature efficiency*. This measure builds off of the previous one. The maximum ratio is first calculated as before. Then, the instances that can be discriminated are removed and the maximum (individual) feature efficiency is recalculated with the remaining instances. This process is repeated until no more instances can be removed. The ratio of instances that can be discriminated is returned.

- Measures of class separability:
  - *The minimized sum of the error distance of a linear classifier*. This feature measures to what extent training data is linearly separable and returns the difference between a linear classifier and the actual class value.
  - *The training error of a linear classifier*. This feature also measures to what extent the training data is linearly separable.
  - *The fraction of points on the class boundary*. This feature estimates the length of the class boundary by constructing a minimum spanning tree over the entire data set and returning the ratio of the number of nodes in the spanning tree that are connected and belong to different classes to the number of instances in the data set.
  - *The ratio of average intra/inter class nearest neighbor distance*. This measure compares the within class spread with the distances to the nearest neighbors of the other classes. For each instance, the distance to its nearest neighbor with the same class ($intraDist(x)$) and to its nearest neighbor with a different class ($interDist(x)$) is calculated. Then the measure returns:

$$\frac{\sum_i^N intraDist(x_i)}{\sum_i^N interDist(x_i)}$$

    where N is the number of instances in the data set.
  - *The leave-one-out error rate of the one-nearest neighbor classifier*. This feature measures how close the examples of different classes are.
- Measures of geometry, topology, and density of manifolds
  - *The nonlinearity of a linear classifier*. Following Hoekstra and Duin [11], given a training set, a test set is created by linear interpolation with random coefficients between pairs of randomly selected instances of the same class. The error rate of a linear classifier trained with the original training set on the generated test set is returned.
  - *The nonlinearity of the one-nearest neighbor classifier*. A test set is created as with the previous feature, but the error rate of a 1-nearest neighbor classifier is returned.
  - *The fraction of maximum covering spheres*. A covering sphere is created by centering on an instance and growing as much as possible before touching an instance from another class. Only the largest spheres are considered. The measure returns the number of spheres divided by the number of instances in the data set and provides an indication of how much the instances are clustered in hyperspheres or distributed in thinner structures.
  - *The average number of points per dimension*. This measure is the ratio of instances to attributes and roughly indicates how sparse a data set is.

Multi-class modifications are made according to the implementation of the data complexity library (DCoL) [15].

Pfahringer et al. [17] introduced the notion of using performance values (i.e., accuracy) of simple and fast classification algorithms as meta-features. The landmarkers that are included in the MLRR are listed below.

- *Linear discriminant learner*. Creates a linear classifier that finds a linear combination of the features to separate the classes.
- *One nearest neighbor learner*. Redundant with the leave-one-out error rate of the one-nearest neighbor classifier from Ho and Basu [10].
- *Decision node learning*. A decision stump that splits on the attribute that has the highest information gain. A decision stump is a decision tree with only one node.
- *Randomly chosen node learner*. A decision stump that splits on a randomly chosen attribute.
- *Worst node learner*. A decision stump that splits on the attribute that has the lowest information gain.
- *Average node learner*. A decision stump is created for each attribute and the average accuracy is returned.

The use of landmarkers has been shown to be competitive with the best performing meta-features with a significant decrease in computational effort [19].

Smith et al. [23] sought to identify and characterize instances that are difficult to classify correctly. The difficulty of an instance was determined based on how frequently it was misclassified. To characterize why some instances are more difficult than others to classify correctly, the authors used different hardness measures. They include:

- *k-Disagreeing Neighbors*. The percentage of k nearest neighbors that do not share the target class of an instance. This measures the local overlap of an instance in the original space of the task.
- *Disjunct size*. This feature indicates how tightly a learning algorithm has to divide the task space to correctly classify an instance. It is measured as the size of a disjunct that covers an instance divided by the largest disjunct produced, where the disjuncts are formed using the C4.5 learning algorithm.
- *Disjunct class percentage*. This features measure the overlap of an instance on a subset of the features. Using a pruned C4.5 tree, the disjunct class percentage is the number of instances in a dis-

junct that belong to the same class divided by the total number of instances in the disjunct.

- *Tree depth (pruned and unpruned)*. Tree depth provides a way to estimate the description length, or Kolmogorov complexity, of an instance. It is the depth of the leaf node that classifies an instance in an induced tree.
- *Class likelihood*. This features provides a global measure of overlap and the likelihood of an instance belonging to the target class. It is calculated as:

$$\prod_{i}^{|x|} p(x_i|t(x))$$

  where $|x|$ represents the number of attributes for the instance x and $t(x)$ is the target class of x.
- *Minority value*. This feature measures the skewness of the class that an instance belongs to. It is measured as the ratio of instances sharing the target class of an instance to the number of instances in the majority class.
- *Class balance*. This feature also measures the class skew. First, the ratio of the number of instances belonging the target class to the total number of instances is calculated. The difference of this ratio with the ratio of one over the number of possible classes is returned. If the class were completely balanced (i.e. all class had the same number of instances), a value of 0 would be returned for each instance.

The hardness measures are designed to capture the characteristics of why instances are hard to classify correctly. Data set measures can be generated by averaging the hardness measures over the instances in a data set.

6 Conclusions and Future Work

In this paper, we presented the *machine learning results repository* (MLRR) an easily accessible and extensible database for meta-learning. MLRR was designed with the main goals of providing an easily accessible data repository to facilitate meta-learning and providing benchmark meta-data sets to compare meta-learning experiments. To this end, the MLRR provides ready to download meta-data sets of previous experimental results. One of the important features of MLRR is that it provides meta-data at the instance level. Of course, the results could also be used as a means of comparing one's work with prior work as they are stored in the MLRR. The MLRR can be accessed at `http://axon.cs.byu.edu/mlrr`.

The MLRR allows for reproducible results as the data sets are stored on the server and as the class names and toolkits are provided. The ExpDB tends to be a lot more rigid in its design as it is based on relational databases and PMML (predictive model markup language), thus exhibiting a relatively steep learning curve to import and extract data. The MLRR is less rigid in its design allowing for easier access to the data and more extensibility, with the trade-off of less formality.

One direction for future work is to integrate the API provided at OpenML[5] (an implementation of an experiment database) to incorporate their results with those that are in the MLRR. This will help provide easy access to the results that are already stored in OpenML without having to incur the learning cost associated with understanding the database schema.

Another open problem is how to store information about how a data set is preprocessed. Currently, the MLRR can store the instance level information resulting from preprocessing, but it lacks a mechanism to store the preprocessing process. Integrating this information in an efficient way is a direction of current research.

[5] `www.openml.org`

REFERENCES

[1] M. Aksela and J. Laaksonen, 'Using diversity of errors for selecting members of a committee classifier', *Pattern Recognition*, **39**(4), 608–623, (2006).

[2] S. Ali and K.A. Smith, 'On Learning Algorithm Selection for Classification', *Applied Soft Computing*, **6**2, 119–138, (2006).

[3] S. Ali and K.A. Smith-Miles, 'A Meta-learning Approach to Automatic Kernel Selection for Support Vector Machines', *Neurocomputing*, **70**, 173–186, (2006).

[4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, 'Curriculum learning', in *Proceedings of the 26th International Conference on Machine Learning*, pp. 41–48. ACM, (2009).

[5] P. B. Brazdil, C. Soares, and J. Pinto Da Costa, 'Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results', *Machine Learning*, **50**(3), 251–277, (2003).

[6] G. Brown, J. L. Wyatt, and P. Tino, 'Managing diversity in regression ensembles.', *Journal of Machine Learning Research*, **6**, 1621–1650, (2005)

[7] M. S. Gashler, 'Waffles: A machine learning toolkit', *Journal of Machine Learning Research*, **MLOSS 12**, 2383–2387, (July 2011).

[8] T.A.F. Gomes and R.B.C. Prudêncio and C. Soares and A.L.D. Rossi and A. Cravalho, 'Combining Meta-learning and Search Techniques to Select Parameters for Support Vector Machines', *Neurocomputing*, **75**, 3–13, (2012).

[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] T. K. Ho and M. Basu, 'Complexity measures of supervised classification problems', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**, 289–300, (March 2002).

[11] A. Hoekstra and R. P.W. Duin, 'On the nonlinearity of pattern classifiers', in *Proceedings of the 13th International Conference on Pattern Recognition*, pp. 271–275, (1996).

[12] L. I. Kuncheva and C. J. Whitaker, 'Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy.', *Machine Learning*, **51**(2), 181–207, (2003).

[13] J. Lee and C. Giraud-Carrier, 'A metric for unsupervised metalearning', *Intelligent Data Analysis*, **15**(6), 827–841, (2011).

[14] J. Lee and C. Giraud-Carrier, 'Automatic selection of classification learning algorithms for data mining practitioners', *Intelligent Data Analysis*, **17**(4), 665–678, (2013).

[15] A. Orriols-Puig, N. Macià, E. Bernadó-Mansilla, and T. K. Ho, 'Documentation for the data complexity library in c++', Technical Report 2009001, La Salle - Universitat Ramon Llull, (April 2009).

[16] A. H. Peterson and T. R. Martinez, 'Estimating the potential for combining learning models', in *Proceedings of the ICML Workshop on Meta-Learning*, pp. 68–75, (2005).

[17] B. Pfahringer, H. Bensusan, and C. G. Giraud-Carrier, 'Meta-learning by landmarking various learning algorithms', in *Proceedings of the 17th International Conference on Machine Learning*, pp. 743–750, San Francisco, CA, USA, (2000). Morgan Kaufmann Publishers Inc.

[18] U. Rebbapragada and C. E. Brodley, 'Class noise mitigation through instance weighting', in *Proceedings of the 18th European Conference on Machine Learning*, pp. 708–715, (2007).

[19] M. Reif, F. Shafait, M. Goldstein, T. Breuel, and A. Dengel, 'Automatic classifier selection for non-experts', *Pattern Analysis & Applications*, **17**(1), 83–96, (2014).

[20] M. Reif, 'A Comprehensive Dataset for Evaluating Approaches of Various Meta-learning Tasks', in *Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods*, pp. 273–276, (2012).

[21] 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*, pp. 2690–2697, (2011).

[22] M. R. Smith and T. Martinez, 'A comparative evaluation of curriculum learning with filtering and boosting in supervised classification problems', *Computational Intelligence*, accepted, (2014).

[23] M. R. Smith, T. Martinez, and C. Giraud-Carrier, ‘An instance level analysis of data complexity’, *Machine Learning*, **95**(2), 225–256, (2014).
[24] J. Vanschoren, H. Blockeel, Bernhard Pfahringer, and Geoffrey Holmes, ‘Experiment databases - a new way to share, organize and learn from experiments’, *Machine Learning*, **87**(2), 127–158, (2012).

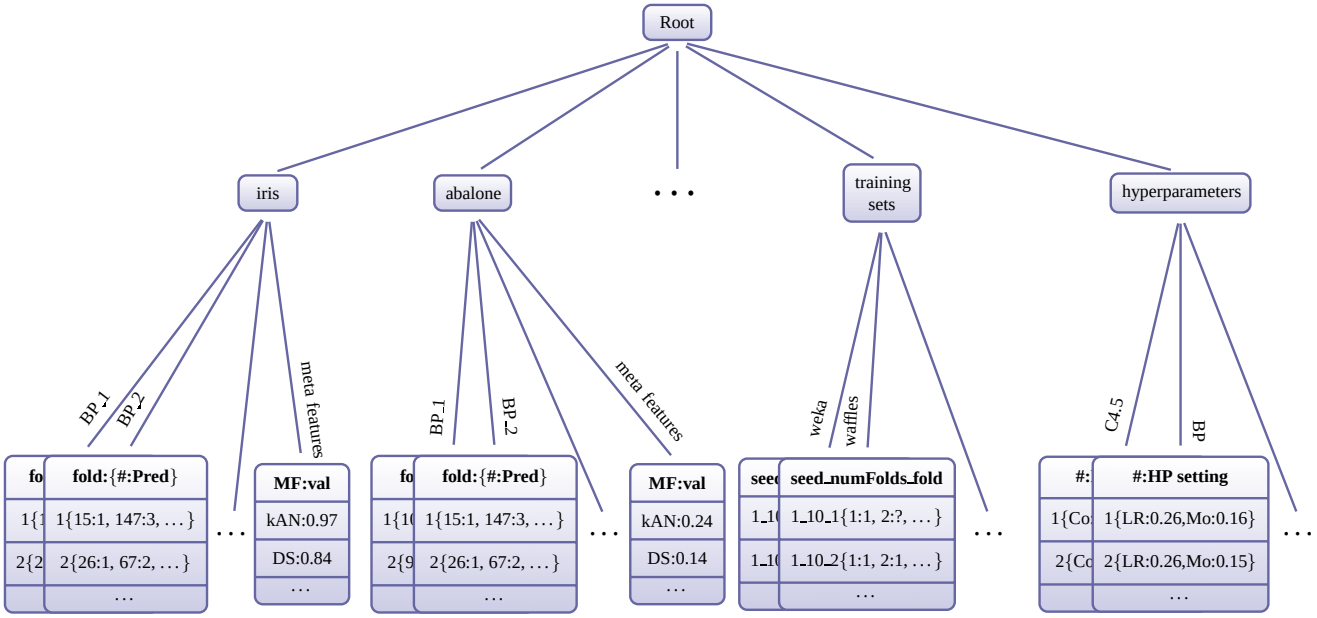


Figure 1. Hierarchical representation of how the results from machine learning experiments are stored in the NoSQL database for the MLRR. Each data set has a collection containing the predictions for each instance from a learning algorithm as well as its meta-features. A separate collection stores all of the information for the learning algorithms and which hyperparameters were used. Another collection stores the information for which instances were used for training.

Measures for Combining Accuracy and Time for Meta-learning

Salisu Mamman Abdulrahman[1] and **Pavel Brazdil**[1,2]

Abstract. The vast majority of studies in meta-learning uses only few performance measures when characterizing different machine learning algorithms. The measure *Adjusted Ratios of Ratio (ARR)* addresses the problem of how to evaluate the quality of a model based on the *accuracy* and *training time*. Unfortunately, this measure suffers from a shortcoming that is described in this paper. A new solution is proposed and it is shown that the proposed function satisfies the criterion of monotonicity, unlike ARR.

1 INTRODUCTION

The major reason why data mining has attracted a great deal of attention in the information industry and in society as a whole in recent years is due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge. The information and knowledge gained can be used for applications ranging from market analysis, fraud detection, customer retention to production control and science exploration.

Data mining tools such as Weka, Knime, and RapidMiner contain hundreds of operators covering a wide range of data analysis tasks, but unfortunately provide only limited advice on how to select the right method according to the nature of the problem under analysis.

To alleviate these problems, different systems have been developed that "intelligently" help users to analyze their data. The goal of *Meta-learning* systems is to help the user by providing some guidance [1, 2, 3]. This is done by suggesting a particular algorithm or operation(s) (e.g. application of particular preprocessing operation or classification algorithm) to the user that would lead to good performance.

The vast majority of studies in meta-learning uses only few performance measures when characterizing different machine learning algorithms. Regards classification, for instance, one common measure is *predictive accuracy*. Other researchers have used also AUC, area under the ROC curve, or else precision, recall and F1. What is common to all these measures is the higher the value, the better. Costs of operations, and in particular training time, are different though, as the lower the value, the better.

An aggregate metric that combine both accuracy and time as metric was presented in [4], ARR, the *adjusted ratio of ratios*, which allows the user to add more emphasis either on the predictive accuracy or on the training time. This measure suffers however, from a shortcoming, which is described in the next section.

2 RANKING BASED ON ACCURACY AND TIME

The *Adjusted Ratio of Ratios* (ARR) measure aggregates information concerning accuracy and time. It can be seen as an extension of the *success rate ratios* (SRR) method. This method was presented in [4] together with two other basic measures, *average ranks* (AR) and *significant wins* (SW). This multicriteria evaluation measure combines the information about the accuracy and total training/execution time of learning algorithms and is defined as:

$$ARR_{a_p a_q}^{d_i} = \frac{\frac{SR_{a_p}^{d_i}}{SR_{a_q}^{d_i}}}{1+AccD*\log(\frac{T_{a_p}^{d_i}}{T_{a_q}^{d_i}})} \quad (1)$$

where $SR_{a_p}^{d_i}$ and $T_{a_p}^{d_i}$ represent the success rate and time of algorithm a_p on dataset d_i, respectively. The term $SR_{a_p}^{d_i}/SR_{a_q}^{d_i}$ is the ratio of success rates which can be seen as a measure of the advantage of algorithm a_p over algorithm a_q (i.e., a benefit). The equivalent ratio for time, $T_{a_p}^{d_i}/T_{a_q}^{d_i}$, can be seen as a measure of the disadvantage of algorithm a_p over algorithm a_q (i.e., a cost). Thus, the authors have taken the ratio of the benefit and the cost, obtaining thus a measure of the overall quality of algorithm a_p.

However, we note that time ratios have, in general, a much wider range of possible values than success rate ratios. If a simple time ratio were used it would dominate the ratio of ratios. This effect can be controlled by re-scaling using $\log(T_{a_p}^{d_i}/T_{a_q}^{d_i})$ which provide a measure of the order of magnitude of the ratio. The relative importance between accuracy and time is taken into account by multiplying this expression by the *AccD* parameter. This parameter is provided by the user and represents the amount of accuracy he/she is willing to trade for a 10 times speedup or slowdown. For example, *AccD* = 10% means that the user is willing to trade 10% of accuracy for 10 times speedup/slowdown. Finally, the value of 1 is added to $\log(T_{a_p}^{d_i}/T_{a_q}^{d_i})$ to yield values that vary around 1, as happens with the success rate ratio.

The ARR should ideally be monotonically increasing. Higher success rate ratios should lead to higher values of ARR. Higher time ratios should lead to lower values of ARR. The overall effect of combining the two should again be monotonic.

We have decided to verify whether this property can be verified on data. We have fixed the value of SRR to 1 and varied the time ratio from very small values (2^{-20}) to very high values (2^{20}) and calculated the ARR for three different values of AccD (0.2, 0.3 and 0.7). The result can be seen in the plot in Fig. 1. The horizontal axis shows the log of the time ratio (logRT). The vertical axis shows the ARR value.

As can be seen, the resulting ARR function is not monotonic and even approaching infinity at some point. Obviously, this can lead to incorrect rankings provided by the meta-learner. However, what is even more worrying is that this can affect the evaluation results. In the next section, we propose a solution to this problem.

3 OUR PROPOSED SOLUTION

When devising a new solution we did not wish to change the overall philosophy underlying ARR. We believe that it is indeed a good idea to work with ratios, as absolute numbers do not carry much meaning.

[1] LIAAD Inesc Tec, Porto, sma@inescporto.pt, pbrazdil@inescporto.pt

[2] FEP, University of Porto.

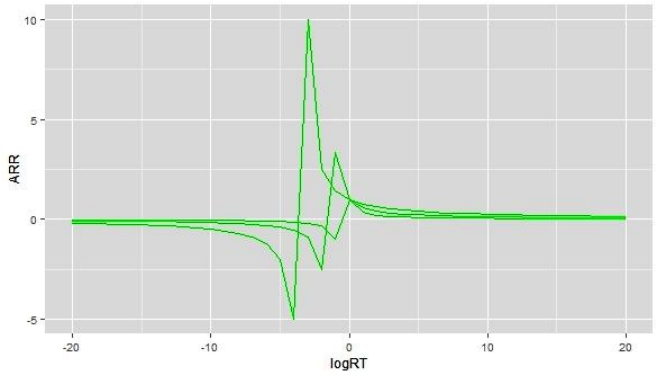


Figure 1. ARR with three different values for AccD (0.2, 0.3 and 0.7)

The accuracy of 90% can be considered good in one situation, but very bad in another. After some reflection, we have realized that the problem lies in the way how the time ratio has been re-scaled. So, we considered another way of re-scaling, which does not use *log*, but n-th root instead, where *n* is a parameter. The proposed function is referred to as A3R and is defined as follows:

$$A3R_{a_p a_q}^{d_i} = \frac{\frac{SR_{a_p}^{d_i}}{SR_{a_q}^{d_i}}}{\sqrt[n]{T_{a_p}^{d_i}/T_{a_q}^{d_i}}}, \qquad (2)$$

As Fig. 2 shows, this function is monotonic. The higher the A3R, the better.

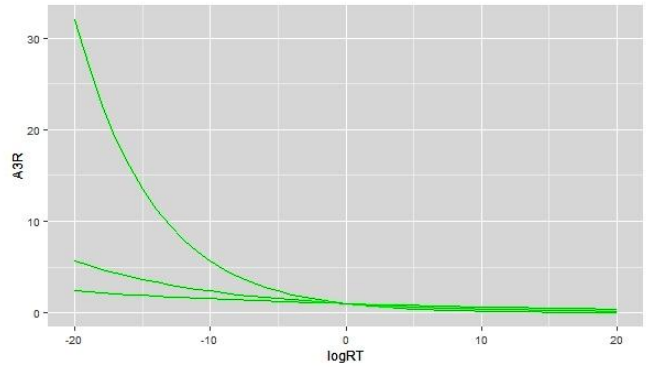


Figure 2. A3R for three different settings for the n-th root (4, 8, and 16)

Taking n-th root in the denominator of eq.(2) enables to rescale the ratio of times. The higher the value of n, the greater the rescaling. So, for instance, if one algorithm is 10 slower than another, the ratio is 10. Taking for 8-th (2nd) root of this will decrease it to 1.33 (3.16). If the ratio were 0.1 this would result in 0.74 (0.31). All numbers get closer to 1 after rescaling.

The change from ARR to A3R is important, as we wish to recalculate many meta-learning experiments and consider both accuracy ratios (and possibly AUC ratios) together with time ratios, suitably rescaled.

To understand the relationship between the success rate ratios (SRR) and time ratios (RT), we have constructed iso-A3R curves (Fig.3). The horizontal axis plots *logRT* in an increasing order of time rate ratios. Thus negative values on the left characterize fast algorithms, while the positive values on the right characterize slow ones. The vertical axis shows the success rate ratios (SRR). Each curve shows the values of A3R where the values are constant. The blue (red, green) curve represents situations where *A3R* is 0.9 (1.0, 1.1). As the ratio of times decreases (i.e. the algorithm is faster), it is sufficient to have lower values of the success rate ratio (SRR) to obtain the same value of A3R.

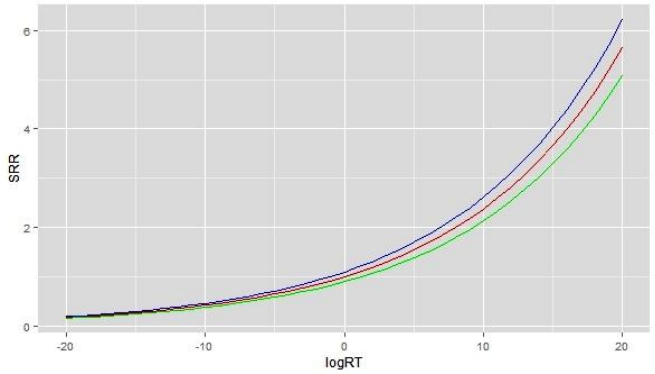


Figure 3. Iso-curves with three different values of A3R (0.9, 1.0 and 1.1). Here n=8 was used to calculate the root.

4 FUTURE PLANS

We intend to improve the methods presented in [5] which rely on relatively pairwise comparison involving two algorithms. We plan to upgrade this work by considering the information concerning both accuracy (or AUC) ratios and time ratios. Hence, the new function proposed will be very useful.

Besides, another challenge is that the new set-up would use many more algorithms (in the order of 100's) than in previous studies. We will exploit the OpenML [6] database in this process and collaboration is underway with U.Leiden on running some of the experiments and re-using the results. Considering that the number of algorithms is high, we need to re-think the method based on pairwise comparisons.

Furthermore, we plan to use the method based on sampling landmarks, as in [5]. To simplify the whole procedure, we will probably use a fixed set of samples, rather than using some dynamic sampling strategy, as proposed in [5]. Still, we need to evaluate what the best number of samples is from the benefit-cost perspective.

5 CONCLUSION

We have presented a new measure A3R for evaluating the performance of algorithms that considers both accuracy and time ratios suitably re-scaled. We have shown that this measure satisfies the criterion of monotonicity, unlike the previous version ARR. We have discussed the usage of A3R in further experiments on meta-learning.

This work is funded (or part-funded) by the ERDF – European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project «FCOMP - 01-0124-FEDER-022701»

REFERENCES

[1] P. Brazdil, C. Giraud-Carrier, C.Soares, and R. Vilalta, *Metalearning:* Applications *to Data Mining*, Springer, 2009.

[2] Kalousis, A. (2002). Algorithm selection via meta-learning. *PhD* Thesis. *University of Geneva*.

[3] Gama, J. and P. Brazdil (1995). Characterization of classification algorithms. *Lecture Notes in Computer Science 990*, 189–200.

[4] Brazdil, P. B., C. Soares, and Joaquin Pinto Da Costa. "Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results." *Machine Learning* 50.3 (2003): 251-277.

[5] Leite, R., and P. Brazdil. "Active Testing Strategy to Predict the Best Classification Algorithm via Sampling and Metalearning." *ECAI*. 2010.

[6] Vanschoren, J.. "The experiment database for machine learning." *5th Planning to Learn Workshop, WS28 at ECAI-2012*. 2012.

Determining a proper initial configuration of Red-Black planning by machine learning

Otakar Trunda and **Roman Barták** [1]

1 INTRODUCTION

Planning deals with finding a sequence of actions that transforms the world from a given initial state to a state that satisfies a certain goal condition [8]. For the purposes of this paper we can define a planning problem simply as a state-transition system where states are the world states and transitions correspond to application of actions. States are defined by values of state variables. Let X be the set of state variables, each variable x_i has a finite domain D_i of its possible values. Then state s is a mapping from X to $\bigcup_i D_i$. $s : X \mapsto \bigcup_i D_i$, where $\forall i, s(x_i) \in D_i$.

The state space has a form of the Cartesian product of variables' domains. $Space = \prod D_i$ Every state $s \in Space$ has assigned a (possibly empty) set of its successor states designed $succ(s)$, every $t \in succ(s)$ is labeled by the action that transforms s to t (i.e. performing actions changes values of state variables). The task is to find a path p in this state-transition system that leads from a given initial state to some state satisfying a goal condition (a goal state). $p = \{s_0, s_1, \ldots, s_n\}$, where s_0 is the initial state, s_n is some goal state and $\forall 0 \leq i < n : s_{i+1} \in succ(s_i)$. Such a path is called a *solution plan*. The goal is to reach a state where some variables have specified values.

One of the most promising approaches to solve the planning problem (based on the results of several International Planning Competitions [2]) is heuristic-guided forward search. (Mostly in a form of A^* or a hill-climbing). These approaches make use of a heuristic estimation during search and the accuracy of the heuristic estimator has a great impact on the performance. Hence designing a powerful and easy-to-compute heuristic is of paramount importance.

Heuristics are usually based on relaxations of the problem. When estimating the quality of the best solution, we relax the problem by ignoring some constraints (making the problem easier), then solve the relaxed problem and use the quality of that solution as a lower bound on the quality of the best solution to the original problem. In planning, this principle is represented by the well known *delete relaxation heuristic* and its variants [8, 3, 4]. Heuristics based on this principle often work well, but in some situations they greatly underestimate the real value making them inaccurate (see [6] for example).

Delete relaxation allows the state variables to hold several values simultaneously, so the *relaxed state* subsumes several ordinary states. Furthermore, performing actions (i.e. making transitions) only adds new elements to the set of values that each variable currently holds (never removes any value). Hence the set of ordinary states that the relaxed state subsumes monotonically increases on every path. A path is a *relaxed solution plan* if it leads to a relaxed state which subsumes some goal state. The length of relaxed plan is then used to estimate the length of the real plan.

[1] Charles University in Prague, Faculty of Mathematics and Physics, email: otakar.trunda@mff.cuni.cz, roman.bartak@mff.cuni.cz

2 RED-BLACK PLANNING

Red-Black planning is a new approach to heuristics design which generalizes the delete relaxation and compensates for many of its shortcomings with a reasonable computational effort [7, 6, 5]. It divides the set of state variables into two disjoint subsets - *Red* and *Black*, which are treated differently during the planning. The *Red* variables are treated as in the *delete relaxation* while the *Black* variables are not relaxed. If all variables are *Red* then the heuristic works same as the delete relaxation.

The authors showed that by the proper selection of *Red* variables, we can reduce the underestimation (in most cases) and still keep the method polynomial. They also observed that the selection of *Red* variables has a great impact on the overall performance. While proper selection leads to good performance, with poor selection the performance degrades. Selecting the proper variables, however, appears to be a hard problem.

The authors performed several tests with *intuitive* and *counter-intuitive* variable selection methods (where *intuitive* relaxes the least important variables, while the *counter-intuitive* method relaxes the most important variables). It turned out that the *counter-intuitive* method often beats the *intuitive* one (with respect to the time required for solving the problem) which makes the problem quite unpredictable. This led the authors to hypothesize that no *simple and efficient* method for selecting the variables can be found.

3 OUR METHOD

We believe that different domains require different ways of selecting the variables. We propose a method based on machine learning that works as follows: first it creates a set of small sub-problems of the original problem and then it determines the proper variable selection for these sub-problems (by enumerating all possibilities). Finally, it uses the solutions of sub-problems to derive the solution to the original problem.

3.1 Creating samples

We create the sub-problems by selecting small subsets of variables and restricting the original problem to these variables only. The restriction has a form of projection which preserves the paths - i.e. if there is a path from s to t in the original state-transition system, then there is a path from $restriction(s)$ to $restriction(t)$ in the new system. Of course, new paths may emerge during the restriction that were not present before.

Formally, let $\mathcal{A}$ be a planning problem as defined earlier, X its state variables, and $Space = \prod D_i$ its state space. Then for every $P \subseteq X$ called *pattern* and every state $s \in Space$ we define a *restriction* of s to P as $s^P : P \mapsto \bigcup D_i$, where $\forall x_i \in P, s^P(x_i) = s(x_i)$. A *restriction* of $\mathcal{A}$ to a pattern P is a planning problem $\mathcal{A}^P$ with state variables P, state space $Space^P = \prod_{\{i|x_i \in P\}} D_i$, and for $s, t \in Space^P : s \in succ(t)$ if and only if there exist $u, v \in Space$ such that $s = u^P, t = v^P$ and $u \in succ(v)$. The initial state and goal states of the restricted problem are restrictions of the originals.

In each sub-problem induced by a pattern, we create samples by enumerating all ways of selecting the Red variables. A *sample* then consists of a pair *(pattern, selected Red variables)*.

3.2 Evaluating samples

Let X be the set of state variables of the original planning problem. A sample q is given in a form $q = (P_q, R_q)$, where P_q is the pattern and R_q is the set of selected red variables, $R_q \subseteq P_q \subseteq X$. To evaluate the sample q, we chose the following procedure:

1. Restrict the original problem to the pattern P_q
2. Solve the restricted problem by A^* with the Red-Black heuristic using R_q as a set of red variables.
3. Measure the time required to perform step 2 in seconds and use it to evaluate the sample. ($Val(q)$ denotes the value of a sample q.)

We decided to use the run-time to evaluate the sample rather than other characteristics like heuristic calls or expanded nodes. We believe that using such characteristics would bias the selection in favor of large patterns and small *Red* sets, since such combination would lead to a very accurate heuristic. However, such heuristic might take a long time to compute and probably wouldn't be the best alternative.

Since run-time of the whole process is the criterion we want to optimize, it seems appropriate to use it to evaluate samples.

3.3 Learning from samples

After evaluating enough samples, we have to select the red variables for the original problem. In our preliminary experiments, we used the following simple procedure, but we believe that this phase can yet be perfected by using more sophisticated approach.

1. Given the set of samples Q, a sample $q = (P_q, R_q)$, divide the samples to groups by the pattern they use. $Q_P = \{q \in Q \mid P_q = P\}$
2. Select the best sample in each group Q_P (one with the lowest evaluation), and denote its Red set as $Best^P$.
3. For each state variable count how many times it appears in some $Best$ set. $val(x_i) = \left|\{Best^P \mid x_i \in Best^P\}\right|$
4. Select variables with the highest evaluation.

In step four, the number of variables to select can be a fixed constant or a fixed ratio, but we chose a different approach. Suppose there are n state variables. We sort the variables nonincreasingly by their evaluation: $\{x_1, x_2, \ldots, x_n\}$, where $val(x_i) \geq val(x_{i+1})$. We add the first variable and then keep adding more until $val(x_i) - val(x_{i+1}) > \frac{val(x_1) - val(x_n)}{n}$. This stopping criterion should find the gap between the *good* variables and the *bad* variables. We intend to test other selection policies as well.

Step three can be generalized by introducing weights to the *Best* sets. Currently, each *Best* set has a weight of 1, but larger patterns give us more information since they are closer to the original problem. Step three can be modified to $val(x_i) = \sum_{\{Best|x_i \in Best\}} w(Best)$, where w is a weight function. We used $w(Best) = |Best|$, but different functions are also possible.

Finally, step two can be modified to work with more samples than just the best one. Imagine that there might be a variable which is rarely in the best sample, but often in the second best one. This would still be a good candidate to pick. In step three we would then average the evaluation of all samples that contain the variable x_i, possibly weighted according to the size of the pattern they use. This modification should lead to more accurate results, but it takes more time to compute. Therefore it is not yet clear whether or not it will improve the overall performance.

4 CONCLUSIONS AND FUTURE WORK

We present the parameter learning method in a very simple form, many issues remain unresolved. Preliminary experiments show promising results, but the method still needs to be adjusted and properly tested on a larger set of planning domains.

One part we didn't address yet is the selection of patterns during the creation of samples. Unlike typical machine learning applications, here we can decide what samples we use for the learning. Patterns should be selected iteratively and the selection should be based on previous results and should support both exploration and exploitation. We intend to make use of some Monte-Carlo technique, possibly Monte-Carlo Tree Search [1]. The method is guaranteed to converge to optimal solution if patterns are chosen incrementally (as the size of the pattern grows, the sub-problem converges to the original problem). The speed of convergence, however, needs yet to be determined for various domains.

The proposed method of learning from pattern-induced subproblems is not bound to the Red-Black planning heuristic only, but can be used to gain information about other features of the planning problem as well. Such information might then help to improve various search methods.

ACKNOWLEDGEMENTS

The research is supported by the Grant Agency of Charles University under contract no. 390214 and it is also supported by SVV project number 260 104.

REFERENCES

[1] C.B. Browne et al., 'A survey of monte carlo tree search methods', *Computational Intelligence and AI in Games, IEEE Transactions on*, **4**(1), 1–43, (March 2012).
[2] ICAPS Competitions. http://ipc.icaps-conference.org, June 2014.
[3] Jörg Hoffmann, 'Where "ignoring delete lists" works: Local search topology in planning benchmarks', *J. Artif. Int. Res.*, **24**(1), 685–758, (November 2005).
[4] Jörg Hoffmann, 'Where Ignoring Delete Lists Works, Part II: Causal Graphs', in *21st International Conference on Automated Planning and Scheduling*, Freiburg, Allemagne, (2011).
[5] Michael Katz and Jörg Hoffmann, 'Red-black relaxed plan heuristics reloaded.', in *SOCS*, eds., Malte Helmert and Gabriele Rger. AAAI Press, (2013).
[6] Michael Katz, Jörg Hoffmann, and Carmel Domshlak, 'Red-black relaxed plan heuristics', in *AAAI'13*, (2013).
[7] Michael Katz, Jörg Hoffmann, and Carmel Domshlak. Who said we need to relax all variables?, 2013.
[8] Dana Nau, Malik Ghallab, and Paolo Traverso, *Automated Planning: Theory & Practice*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

Hybrid Multi-Agent System for Metalearning in Data Mining

Klára Pešková and **Jakub Šmíd** and **Martin Pilát** and **Ondřej Kazík** [1] and **Roman Neruda** [2]

Abstract. In this paper, a multi-agent system for metalearning in the data mining domain is presented. The system provides a user with intelligent features, such as recommendation of suitable data mining techniques for a new dataset, parameter tuning of such techniques, and building up a metaknowledge base. The architecture of the system, together with different user scenarios, and the way they are handled by the system, are described.

1 Introduction

Lately, *data mining* — an automated process of gaining information from datasets — has become an issue of interest in the artificial intelligence. This interest have been whetted by the progress in the computational technology, such as high performance machine clusters or large storage devices, but most importantly by the possibility of an access to enormous amount of data that are collected on daily basis. The datasets vary in many factors as they origin in different areas of human or nature activities. It is hard even for a data mining expert to choose from the wide range of machine learning methods that are used in data mining and to set its parameters to the values that would produce a reasonable results for the specific dataset. Tools that ease up the parameter set up can significantly boost up the productivity of data mining process. Moreover, the automation of the whole process would help those researchers, who are not data mining experts, to enjoy the benefits of this research line. This is where the *metalearning* [3] comes into play.

Metalearning over data mining methods and datasets is a very demanding task, especially with respect to computational performance as it uses results of data mining methods applied on various datasets as its training/testing data. The software that is capable of both data mining and metalearning is by definition a large and complex system. To design the architecture of our system, we have chosen the agent-based approach as it brings many advantages to this complex task. The main one being its distributed and parallel nature — the system can spread over computer networks and be accessed by many users who only by using the system and running their experiments provide the data needed for metalearning algorithms. It also supplies a fast parallel execution of performance demanding tasks. The interconnection of different parts of the system (i.e. the *communication among agents*) is done only by sending messages which results in an easy extensibility and re-usability of the parts of the system — *agents*. It enables researchers to easily add their own components (e.g. custom data mining methods) and to re-use the implemented components in different situations.

We have designed and implemented a multi-agent system (*MAS*) which is capable of executing simple data mining tasks as well as complex metalearning problems (involving not only recommending of data mining methods but also setting their parameters), and it provides all the mechanisms necessary for experimenting with different metalearning approaches. The system is hybrid — it employs combination of different artificial intelligence methods [4].

We use JADE [2] — the multi-agent framework, as a base for our agents; most of the computational agents in our system use Weka [6] data mining methods. The extensibility of our system is assured by the use of the structured ontology language and following the FIPA [1] international standards of agents' communication.

2 Scenarios

To propose an appropriate architecture of our computational MAS, we have considered the following basic scenarios for processing a dataset. In the most simple case the user knows which method and what parameters of this method she would like to use. In the other two basic scenarios, the system uses its intelligent meta-learning features: If the user knows what method to use but does not know how to set its parameters, the system is able to search the parameter space of the method and find a setting that provides good results. In the third case, the user does not even know what method to use and lets the system decide by itself. In this case the system recommends the best possible method or provides a ranking of the methods based on predicted errors and duration. These simple scenarios can be extended into more complex ones — e.g. it is also possible to combine the recommendation of the best method the with parameter space search, when the recommender chosen by the user recommends an interval of the parameter's values.

As a positive side effect, the *metaknowledge base* for metalearning purposes is being build up by each experiment.

3 Role-based Architecture

In order to effectively design our system, we have chosen the organization-centered formalism *AGR* (*Agent-Group-Role*). The *role* is a set of capabilities and responsibilities that the agent accepts by handling the role. *Group* — the building block of a MAS — is a set of agents with allowed roles and interactions, defined by a *group structure*. The multi-agent system then consists of multiple groups which can overlap by agents belonging to more than one group. In this formalism, we abstract from the algorithmic details and inner logic of the agents in the MAS. In our previous work [7], we have

[1] Charles University in Prague, Faculty of Mathematics and Physics, emails: klara@pisecko.cz, jakub.smid@ktiml.mff.cuni.cz, martin.pilat@gmail.com, kazik.ondrej@gmail.com

[2] Institute of Computer Science, Academy of Sciences of the Czech Republic, email: roman@cs.cas.cz

used the ontological formalism of OWL-DL to describe the organizational model.

The following group structures were defined according to the aforementioned scenarios: *administrative group structure*, *computational group structure*, *search group structure*, *recommendation group structure*, *data group structure* and *data-management group structure*.

Our MAS is composed of groups that are instances of these group structures. The architecture is depicted in the Figure 1.

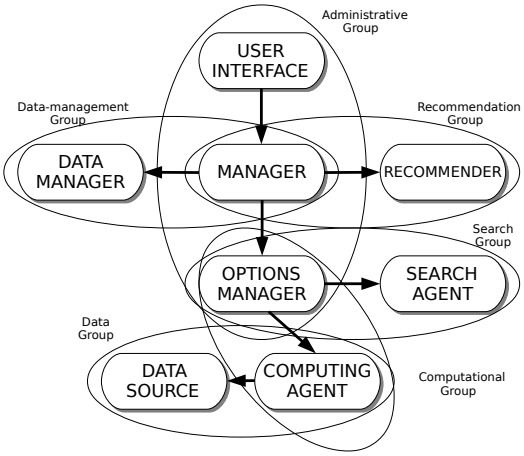


Figure 1. Architecture of our MAS: Group structures

4 Metalearning

The key parts of our system are those providing intelligent metalearning behavior, i.e. agents that provide parameter space search methods and recommender agents. These agents are intended to (at least partially) replace a human expert. They make use of the previous experience gathered by the system, which is captured in the metaknowledge base. It contains results of machine learning experiments and *metadata* — general features of the datasets.

The MAS-based solution allows a flexibility in choice of the parameter space search algorithms, each of these is encapsulated in a search agent. General tabulation, random search, simulated annealing [9], or parallel methods, such as evolutionary algorithms [5], are implemented in our system. Another great benefit of the agent-based approach is the natural capability of parallel execution of computations with various parameters which significantly decreases the time needed for the execution of the parameter space search process.

One of essential features of our MAS is its capability of recommending a suitable computational method for a new dataset, according to datasets similarity and previously gathered experience. The choice of the similar dataset(s) is based on various previously proposed metrics [8], which measure the similarity of their metadata. Our database contains over two million records, that are used to suggest the proper method (including its parameters) and estimate its performance on a new dataset.

The latest version of our MAS contains the following types of recommenders, which differ in the metric used and in the number of recommended methods they provide:

- *Basic recommender* chooses a method based on the single closest dataset using the unweighted metadata metric.
- *Clustering Based Classification* [8] chooses the whole cluster of similar datasets and the corresponding methods, using different sets of metadata features.
- *Evolutionary Optimized Recommenders* are similar to the two above described recommender types, using different weighted metrics, optimized by an evolutionary algorithm.
- *Interval Recommender* recommends intervals of suitable parameter values and leaves their fine-tuning to the parameter space search methods.

Another functionality of our system is a multi-objective optimization of data mining configurations. The search algorithm is employed in order to find beneficial combinations of pre-processings and machine learning methods to the presented data. The minimization is performed in error-rate as well as run-time criteria.

5 Conclusions

In this paper, we presented a multi-agent system for metalearning in data mining, which includes solving of the most important and challenging metalearning tasks – the recommendation of a suitable method for a new dataset, and the tuning of parameters of such methods. We have proposed the systems architecture and proved its usability by an implementation that is used by our research team on a regular basis to conduct metalearning and data mining experiments. The role-based multi-agent approach brings in many advantages into a complex task of metalearning, the main benefit being its easy extensibility. The multi-agent parallel nature of the system speeds up the time consuming tasks significantly.

6 Acknowledgements

Jakub Šmíd and Klára Pešková have been supported by the Charles University Grant Agency project no. 610214, R. Neruda has been supported by the Ministry of Education of the Czech Republic project COST LD 13002.

REFERENCES

[1] The Foundation for Intelligent Physical Agents (FIPA). http://www.fipa.org/.
[2] Java Agent DEvelopment framework. http://jade.tilab.com/.
[3] Pavel Brazdil, Christophe G. Giraud-Carrier, Carlos Soares, and Ricardo Vilalta, *Metalearning - Applications to Data Mining.*, Cognitive Technologies, Springer, 2009.
[4] Oscar Castillo, Patricia Melin, Janusz Kacprzyk, and Witold Pedrycz, *Hybrid Intelligent Systems*, Springer, 2007.
[5] Agoston E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, SpringerVerlag, 2003.
[6] M. Hall et al., 'The weka data mining software: An update.', *SIGKDD Explorations*, **11**, (2009).
[7] Ondřej Kazík and Roman Neruda, 'Ontological modeling of meta learning multi-agent systems in OWL-DL', *IAENG International Journal of Computer Science*, **39**(4), 357–362, (Dec 2012).
[8] Ondrej Kazík, Klára Pešková, Jakub Šmíd, and Roman Neruda, 'Clustering based classification in data mining method recommendation', in *ICMLA '13: Proceedings of the 2013 12th International Conference on Machine Learning and Applications*, pp. 356–361, Washington, DC, USA, (2013). IEEE Computer Society.
[9] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, 'Optimization by simulated annealing', *Science*, **220**, 671–680, (1983).

Model Selection in Data Analysis Competitions

David Kofoed Wind[1] and **Ole Winther**[2]

Abstract. The use of data analysis competitions for selecting the most appropriate model for a problem is a recent innovation in the field of predictive machine learning. Two of the most well-known examples of this trend was the Netflix Competition and recently the competitions hosted on the online platform Kaggle.

In this paper, we will state and try to verify a set of qualitative hypotheses about predictive modelling, both in general and in the scope of data analysis competitions. To verify our hypotheses we will look at previous competitions and their outcomes, use qualitative interviews with top performers from Kaggle and use previous personal experiences from competing in Kaggle competitions.

The stated hypotheses about feature engineering, ensembling, overfitting, model complexity and evaluation metrics give indications and guidelines on how to select a proper model for performing well in a competition on Kaggle.

1 Introduction

In recent years, the amount of available data has increased exponentially and "Big Data Analysis" is expected to be at the core of most future innovations [2, 4, 5]. A new and very promising trend in the field of predictive machine learning is the use of data analysis competitions for model selection. Due to the rapid development in the field of competitive data analysis, there is still a lack of consensus and literature on how one should approach predictive modelling competitions.

In his well-known paper "Statistical Modeling : The Two Cultures" [1], Leo Breiman divides statistical modelling into two cultures, the *data modelling culture* and the *algorithmic modelling culture*. The arguments put forward in [1] justifies an approach to predictive modelling where the focus is purely on predictive accuracy. That this is the right way of looking at statistical modelling is the underlying assumption in statistical prediction competitions, and consequently also in this paper.

The concept of machine learning competitions was made popular with the Netflix Prize, a massive open competition with the aim of constructing the best algorithm for predicting user ratings of movies. The competition featured a prize of 1,000,000 dollars for the first team to improve Netflix's own results by 10% and multiple teams achieved this goal. After the success with the Netflix Prize, the website Kaggle was born, providing a platform for predictive modelling. Kaggle hosts numerous data prediction competitions and has more than 170,000 users worldwide.

[1] Technical University of Denmark, Denmark, email: dawi@dtu.dk
[2] Technical University of Denmark, Denmark, email: olwi@dtu.dk

The basic structure of a predictive modelling competition – as seen for example on Kaggle and in the Netflix competition – is the following: A predictive problem is described, and the participants are given a dataset with a number of samples and the true target values (the values to predict) for each sample given, this is called the training set. The participants are also given another dataset like the training set, but where the target values are not known, this is called the test set. The task of the participants is to predict the correct target values for the test set, using the training set to build their models. When participants have a set of proposed predictions for the test set, they can submit these to a website, which will then evaluate the submission on a part of the test set known as the quiz set, the validation set or simply as the public part of the test set. The result of this evaluation on the quiz set is shown in a leaderboard giving the participants an idea of how they are progressing.

Using a competitive approach to predictive modelling is being praised by some as the modern way to do science:

> Kaggle recently hosted a bioinformatics contest, which required participants to pick markers in a series of HIV genetic sequences that correlate with a change in viral load (a measure of the severity of infection). Within a week and a half, the best submission had already outdone the best methods in the scientific literature. [3]
>
> (Anthony Goldbloom, Founder and CEO at Kaggle)

> These prediction contests are changing the landscape for researchers in my area an area that focuses on making good predictions from finite (albeit sometimes large) amount of data. In my personal opinion, they are creating a new paradigm with distinctive advantages over how research is traditionally conducted in our field. [6]
>
> (Mu Zhu, Associate Professor, University of Waterloo)

This competitive approach is interesting and seems fruitful – one can even see it as an extension of the aggregation ideas put forward in [1] in the sense that the winning model is simply the model with the best accuracy, not taking computational efficiency or interpretability into account. Still one could ask if the framework provided by for example Kaggle gives a trustworthy resemblance of real-world predictive modelling problems where problems do not come with a quiz set and a leaderboard.

In this paper we state five hypotheses about building and selecting models for competitive data analysis. To verify these hypotheses we will look at previous competitions and their outcomes, use qualitative interviews with top performers from Kaggle and use previous personal experiences from competing in Kaggle competitions.

2 Interviews and Previous Competitions

In this section we will shortly describe the data we are using. We will list the people whom we interviewed and name the previous Kaggle competition we are using for empirical data.

2.1 Interviews

To help answer the questions we are stating, we have asked a series of questions to some of the best Kaggle participants throughout time. We have talked (by e-mail) with the following participants (name, Kaggle username, current rank on Kaggle):

- Steve Donoho (BreakfastPirate #2)
- Lucas Eustaquio (Leustagos #6)
- Josef Feigl (Josef Feigl #7)
- Zhao Xing (xing zhao #10)
- Anil Thomas (Anil Thomas #11)
- Luca Massaron (Luca Massaron #13)
- Gábor Takács (Gábor Takács #20)
- Tim Salimans (Tim Salimans #48)

Answers and parts of answers to our questions are included in this paper as quotes when relevant.

2.2 Previous competitions

Besides the qualitative interviews with Kaggle masters, we also looked at 10 previous Kaggle competitions, namely the following:

- Facebook Recruiting III - Keyword Extraction
- Partly Sunny with a Chance of Hashtags
- See Click Predict Fix
- Multi-label Bird Species Classification - NIPS 2013
- Accelerometer Biometric Competition
- AMS 2013-2014 Solar Energy Prediction Contest
- StumbleUpon Evergreen Classification Challenge
- Belkin Energy Disaggregation Competition
- The Big Data Combine Engineered by BattleFin
- Cause-effect pairs

These competitions were selected as 10 consecutive competitions, where we excluded a few competitions which did not fit the standard framework of statistical data analysis (for example challenges in optimization and operations research).

Throughout this paper, these competitions are referenced with the following abbreviated names: FACEBOOK, SUNNYHASHTAGS, SEECLICKPREDICT, BIRD, ACCELEROMETER, SOLARENERGY, STUMBLEUPON, BELKIN, BIGDATA and CAUSEEFFECT.

3 Hypotheses

In this section we state 5 hypotheses about predictive modelling in a competitive framework. We will try to verify the validity of each hypothesis using a combination of mathematical arguments, empirical evidence from previous competitions and qualitative interviews we did with some of the top participants at Kaggle. The five hypotheses to be investigated are:

1. Feature engineering is the most important part of predictive machine learning
2. Overfitting to the leaderboard is a real issue
3. Simple models can get you very far
4. Ensembling is a winning strategy
5. Predicting the right thing is important

3.1 Feature engineering is the most important part

With the extensive amount of free tools and libraries available for data analysis, everybody has the possibility of trying advanced statistical models in a competition. As a consequence of this, what gives you most "bang for the buck" is rarely the statistical method you apply, but rather the features you apply it to. By feature engineering, we mean using domain specific knowledge or automatic methods for generating, extracting, removing or altering features in the data set.

> For most Kaggle competitions the most important part is feature engineering, which is pretty easy to learn how to do.
> (Tim Salimans)

> The features you use influence more than everything else the result. No algorithm alone, to my knowledge, can supplement the information gain given by correct feature engineering. (Luca Massaron)

> Feature engineering is certainly one of the most important aspects in Kaggle competitions and it is the part where one should spend the most time on. There are often some hidden features in the data which can improve your performance by a lot and if you want to get a good place on the leaderboard you have to find them. If you screw up here you mostly can't win anymore; there is always one guy who finds all the secrets.
>
> However, there are also other important parts, like how you formulate the problem. Will you use a regression model or classification model or even combine both or is some kind of ranking needed. This, and feature engineering, are crucial to achieve a good result in those competitions.
>
> There are also some competitions were (manual) feature engineering is not needed anymore; like in image processing competitions. Current state of the art deep learning algorithms can do that for you. (Josef Feigl)

There are some specific types of data which have previously required a larger amount of feature engineering, namely text data and image data. In many of the previous competitions with text and image data, feature engineering was a huge part of the winning solutions (examples of this are for example SUNNYHASHTAGS, FACEBOOK, SEECLICKPREDICT and BIRD). At the same time (perhaps due to the amount of work needed to do good feature engineering here) deep learning approaches to automatic feature extraction have gained popularity.

In the competition SUNNYHASHTAGS which featured text data taken from Twitter, feature engineering was a major part of the winning solution. The winning solution used a simple regularized regression model, but generated a lot of features from the text:

> My set of features included the basic tfidf of 1,2,3-grams and 3,5,6,7 ngrams. I used a CMU Ark Twitter dedicated tokenizer which is especially robust for processing tweets + it tags the words with part-of-speech tags which can be useful to derive additional features. Additionally, my base feature set included features derived from sentiment dictionaries that map

each word to a positive/neutral/negative sentiment. I found this helped to predict S categories by quite a bit. Finally, with Ridge model I found that doing any feature selection was only hurting the performance, so I ended up keeping all of the features ~ 1.9 mil. The training time for a single model was still reasonable.
(aseveryn - 1st place winner)

In the competitions which did not have text or image data, feature engineering sometimes still played an important role in the winning entries. An example of this is the CAUSEEFFECT competition, where the winning entry created thousands of features, and then used genetic algorithms to remove non-useful features again. On the contrary, sometimes the winning solutions are those which go a non-intuitive way and simply use a black-box approach. An example of this is the SOLARENERGY competition where the Top-3 entries almost did not use any feature engineering (even though this seemed like the most intuitive approach for many) – and simply combined the entire dataset into one big table and used a complex black-box model.

Having too many features (making the feature set overcomplete), is not advisable either, since redundant or useless features tend to reduce the model accuracy.

3.1.1 Mathematical justification for feature engineering

When using simple models, it is often necessary to engineer new features to capture the right trends in the data. The most common example of this, is attempting to use a linear method to model non-linear behaviour.

To give a simple example of this, assume we want to predict the price of a house H given the dimensions (length l_H and width w_H of the floor plan) of the house. Assume also that the price $p(H)$ can be described as a linear function $p(H) = \alpha a_H + \beta$, where $a_H = l_H \cdot w_H$ is the area. By fitting a linear regression model to the original parameters l_H, w_H, we will not capture the quadratic trend in the data. If we instead construct a new feature $a_H = l_H \cdot w_H$ (the area), for each data sample (house), and fit a linear regression model using this new feature, then we will be able to capture the trend we are looking for.

3.2 Simple models can get you very far

When looking through descriptions of people's solutions after a competition has ended, there is often a surprising number of very simple solutions obtaining good results. What is also (initially) surprising, is that the simplest approaches are often described by some of the most prominent competitors.

I think beginners sometimes just start to "throw" algorithms at a problem without first getting to know the data. I also think that beginners sometimes also go too-complex-too-soon. There is a view among some people that you are smarter if you create something really complex. I prefer to try out simpler. I "try" to follow Albert Einsteins advice when he said, "Any intelligent fool can make things bigger and more complex. It takes a touch of genius – and a lot of courage – to move in the opposite direction". (Steve Donoho)

My first few submissions are usually just "baseline" submissions of extremely simple models – like "guess the average" or "guess the average segmented by variable X". These are simply to establish what is possible with very simple models. You'd be surprised that you can sometimes come very close to the score of someone doing something very complex by just using a simple model. (Steve Donoho)

I think a simple model can make you top 10 in a Kaggle competition. In order to get a money prize, you have to go to ensembles most of time. (Zhao Xing)

You can go very far [with simple models], if you use them well, but likely you cannot win a competition by a simple model alone. Simple models are easy to train and to understand and they can provide you with more insight than more complex black boxes. They are also easy to be modified and adapted to different situations. They also force you to work more on the data itself (feature engineering, data cleaning, missing data estimation). On the other hand, being simple, they suffer from high bias, so they likely cannot catch a complex mapping of your unknown function. (Luca Massaron)

Simplicity can come in multiple forms, both regarding the complexity of the model, but also regarding the pre-processing of the data. In some competitions, regularized linear regression can be the winning model in spite of its simplicity. In other cases, the winning solutions are those who do almost no pre-processing of the data (as seen in for example the SOLARENERGY competition).

3.3 Ensembling is a winning strategy

As described in [1], complex models and in particular models which are combinations of many models should perform better when measured on predictive accuracy. This hypothesis can be backed up by looking at the winning solutions for the latest competitions on Kaggle.

If one considers the 10 Kaggle competitions mentioned in Section 2.2 and look at which models the top participants used, one finds that in 8 of the 10 competitions, model combination and ensemble-models was a key part of the final submission. The only two competitions where no ensembling was used by the top participants were FACEBOOK and BELKIN, where a possible usage of model combination was non-trivial and where the data sets were of a size that favored simple models.

No matter how faithful and well tuned your individual models are, you are likely to improve the accuracy with ensembling. Ensembling works best when the individual models are less correlated. Throwing a multitude of mediocre models into a blender can be counterproductive. Combining a few well constructed models is likely to work better. Having said that, it is also possible to overtune an individual model to the detriment of the overall result. The tricky part is finding the right balance. (Anil Thomas)

[The fact that most winning entries use ensembling] is natural from a competitors perspective, but potentially very hurtful for Kaggle/its clients: a solution consisting of an ensemble of 1000 black box models does not give any insight and will be extremely difficult to reproduce. This will not translate to real business value for the comp organizers.
(Tim Salimans)

I am a big believer in ensembles. They do improve accuracy. BUT I usually do that as a very last step. I usually try to squeeze all that I can out of creating derived variables and using individual algorithms. After I feel like I have done all that I can on that front, I try out ensembles.
(Steve Donoho)

Ensembling is a no-brainer. You should do it in every competition since it usually improves your score. However, for me it is usually the last thing I do in a competition and I don't spend too much time on it. (Josef Feigl)

Besides the intuitive appeal of averaging models, one can justify ensembling mathematically.

3.3.1 Mathematical justification for ensembling

To justify ensembling mathematically, we refer to the approach of [7]. They look at a *one-of-K* classification problem and model the probability of input x belonging to class i as

$$f_i(x) = p(c_i|x) + \beta_i + \eta_i(x),$$

where $p(c_i|x)$ is an a posteriori probability distribution of the i-th class given input x, where β_i is a bias for the i-th class (which is independent of x) and where $\eta_i(x)$ is the error of the output for class i.

They then derive the following expression for how the added error (the part of the error due to our model fit being wrong) changes when averaging over the different models in the ensemble:

$$E_{\text{add}}^{\text{ave}} = E_{\text{add}} \left(\frac{1 + \delta(N-1)}{N} \right),$$

where δ is the average correlation between the models (weighted by the prior probabilities of the different classes) and N is the number of models trained.

The important take-away from this result is that ensembling works best if the models we combine have a low correlation. A key thing to note though, is that low correlation between models in itself is not enough to guarantee a lowering of the overall error. Ensembling as described above is effective in lowering the variance of a model but not in lowering the bias.

3.4 Overfitting to the leaderboard is an issue

During a competition on Kaggle, the participants have the possibility of submitting their solutions (predictions on the public and private test set) to a public leaderboard. By submitting a solution to the leaderboard you get back an evaluation of your model on the public-part of the test set. It is clear that obtaining evaluations from the leaderboard gives you additional information/data, but it also introduces the possibility of overfitting to the leaderboard-scores:

The leaderboard definitely contains information. Especially when the leaderboard has data from a different time period than the training data (such as with the heritage health prize). You can use this information to do model selection and hyperparameter tuning. (Tim Salimans)

The public leaderboard is some help, [...] but one needs to be careful to not overfit to it especially on small datasets. Some masters I have talked to pick their final submission based on a weighted average of their leaderboard score and their CV score (weighted by data size). Kaggle makes the dangers of overfit painfully real. There is nothing quite like moving from a good rank on the public leaderboard to a bad rank on the private leaderboard to teach a person to be extra, extra careful to not overfit. (Steve Donoho)

Having a good cross validation system by and large makes it unnecessary to use feedback from the leaderboard. It also helps to avoid the trap of overfitting to the public leaderboard.
(Anil Thomas)

Overfitting to the leaderboard is always a major problem. The best way to avoid it is to completely ignore the leaderboard score and trust only your cross-validation score. The main problem here is that your cross-validation has to be correct and that there is a clear correlation between your cv-score and the leaderboard score (e.g. improvement in your cv-score lead to improvement on the leaderboard). If that's the case for a given competition, then it's easy to avoid overfitting. This works usually well if the test set is large enough.

If the testset is only small in size and if there is no clear correlation, then it's very difficult to only trust your cv-score. This can be the case if the test set is taken from another distribution than the train set. (Josef Feigl)

In the 10 last competitions on Kaggle, two of them showed extreme cases of overfitting and four showed mild cases of overfitting. The two extreme cases were BIGDATA and STUMBLEUPON. In Table 1 the Top-10 submissions on the public test set from BIGDATA is shown, together with the results of the same participants on the private test set.

Name	# Public	# Private	Public score	Private score
Konstantin Sofiyuk	1	378	0.40368	0.43624
Ambakhof	2	290	0.40389	0.42748
SY	3	2	0.40820	0.42331
Giovanni	4	330	0.40861	0.42893
asdf	5	369	0.41078	0.43364
dynamic24	6	304	0.41085	0.42782
Zoey	7	205	0.41220	0.42605
GKHI	8	288	0.41225	0.42746
Jason Sumpter	9	380	0.41262	0.44014
Vikas	10	382	0.41264	0.44276

Table 1. Results of the Top-10 participants on the leaderboard for the competition: "Big Data Combine"

In BIGDATA, the task was to predict the value of stocks multiple hours into the future, which is generally thought to be extremely difficult [3]. The extreme jumps on the leaderboard is most likely due to the sheer difficulty of predicting stocks combined with overfitting.

In the cases where there were small differences between the public leaderboard and the private leaderboard, the discrepancy can also sometimes be explained by the scores for the top competitors being so close that random noise affected the positions.

[3] This is similar to what is known as the Efficient Market Hypothesis.

3.5 Predicting the right thing is important

One task that is sometimes trivial, and other times not, is that of "predicting the right thing". It seems quite trivial to state that it is important to predict the right thing, but it is not always a simple matter in practice.

> A next step is to ask, "What should I actually be predicting?". This is an important step that is often missed by many – they just throw the raw dependent variable into their favorite algorithm and hope for the best. But sometimes you want to create a derived dependent variable. I'll use the GE Flightquest as an example you dont want to predict the actual time the airplane will land; you want to predict the length of the flight; and maybe the best way to do that is to use that ratio of how long the flight actually was to how long it was originally estimate to be and then multiply that times the original estimate. (Steve Donoho)

There are two ways to address the problem of predicting the right thing: The first way is the one addressed in the quote from Steve Donoho, about predicting the correct derived variable. The other is to train the statistical models using the appropriate loss function.

> Just moving from RMSE to MAE can drastically change the coefficients of a simple model such as a linear regression. Optimizing for the correct metric can really allow you to rank higher in the LB, especially if there is variable selection involved. (Luca Massaron)

> Usually it makes sense to optimize the correct metric (especially in your cv-score). [...] However, you don't have to do that. For example one year ago, I've won the Event Recommendation Engine Challenge which metric was MAP. I never used this metric and evaluated all my models using LogLoss. It worked well there. (Josef Feigl)

As an example of why using the wrong loss function might give rise to issues, look at the following simple example: Say you want to fit the simplest possible regression model, namely just an intercept a to the data:

$$x = (0.1,\ 0.2,\ 0.4,\ 0.2,\ 0.2,\ 0.1,\ 0.3,\ 0.2,\ 0.3,\ 0.1,\ 100)$$

If we let a_{MSE} denote the a minimizing the mean squared error, and let a_{MAE} denote the a minimizing the mean absolute error, we get the following

$$a_{\text{MSE}} \approx 9.2818, \qquad a_{\text{MAE}} \approx 0.2000$$

If we now compute the MSE and MAE using both estimates of a, we get the following results:

$$\frac{1}{11}\sum_i |x_i - a_{\text{MAE}}| \approx 9 \qquad \frac{1}{11}\sum_i |x_i - a_{\text{MSE}}| \approx 16$$

$$\frac{1}{11}\sum_i (x_i - a_{\text{MAE}})^2 \approx 905 \qquad \frac{1}{11}\sum_i (x_i - a_{\text{MSE}})^2 \approx 822$$

We see (as expected) that for each loss function (MAE and MSE), the parameter which was fitted to minimize that loss function achieves a lower error. This should come as no surprise, but when the loss functions and statistical methods become complicated (such as Normalized Discounted Cumulative Gain used for some ranking competitions), it is not always as trivial to see if one is actually optimizing the correct thing.

4 Additional advice

In addition to the quotes related to the five hypotheses, the top Kaggle-participants also revealed helpful comments for performing well in a machine learning competition. Some of their statements are given in this section.

> The best tip for a newcomer is the read the forums. You can find a lot of good advices there and nowaday also some code to get you started. Also, one shouldn't spend too much time on optimizing the parameters of the model at the beginning of the competition. There is enough time for that at the end of a competition. (Josef Feigl)

> In each competiton I learn a bit more from the winners. A competiton is not won by one insight, usually it is won by several careful steps towards a good modelling approach. Everything play its role, so there is no secret formula here, just several lessons learned applied together. I think new kagglers would benefit more of carefully reading the forums and the past competitions winning posts. Kaggle masters aren't cheap on advice! (Lucas Eustaquio)

> My most surprising experience was to see the consistently good results of Friedman's gradient boosting machine. It does not turn out from the literature that this method shines in practice. (Gabor Takacs)

> The more tools you have in your toolbox, the better prepared you are to solve a problem. If I only have a hammer in my toolbox, and you have a toolbox full of tools, you are probably going to build a better house than I am. Having said that, some people have a lot of tools in their toolbox, but they don't know *when* to use *which* tool. I think knowing when to use which tool is very important. Some people get a bunch of tools in their toolbox, but then they just start randomly throwing a bunch of tools at their problem without asking, "Which tool is best suited for this problem?" (Steve Donoho)

5 Conclusion

This paper looks at the recent trend of using data analysis competitions for selecting the most appropriate model for a specific problem. When participating in data analysis competitions, models get evaluated solely based on their predictive accuracy. Because the submitted models are not evaluated on their computational efficiency, novelty or interpretability, the model construction differs slightly from the way models are normally constructed for academic purposes and in industry.

We stated a set of five different hypotheses about the way to select and construct models for competitive purposes. We then used a combination of mathematical theory, experience from past competitions and qualitative interviews with top participants from Kaggle to try and verify these hypotheses.

Although there is no secret formula for winning a data analysis competition, the stated hypotheses together with additional good advice from top performing Kaggle competitors, give indications and guidelines on how to select a proper model for performing well in a competition on Kaggle.

REFERENCES

[1] Leo Breiman, 'Statistical modeling: The two cultures', *Statistical Science*, (2001).

[2] World Economic Forum. Big data, big impact: New possibilities for international development. http://bit.ly/1fbP4aj, January 2012. [Online].

[3] A. Goldbloom, 'Data prediction competitions – far more than just a bit of fun', in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pp. 1385–1386, (Dec 2010).

[4] Steve Lohr. The age of big data. http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html, February 2012. [Online; posted 11-February-2012].

[5] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela Hung Byers. Big data: The next frontier for innovation, competition and productivity. http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation, May 2011. [Online; posted May-2011].

[6] Zhum Mu. The impact of prediction contests, 2011.

[7] K. Tumer and J. Ghosh, 'Error correlation and error reduction in ensemble classifiers', *Connection Science*, **8**(3-4), 385–403, (1996).

Author Index